University of Liège

Faculty of Applied Sciences
Department of Electrical Engineering & Computer Science

Doctoral Thesis

Supervised machine learning under constraints

Author:
Jean-Michel Begon

Supervisor:
Pr. Pierre Geurts

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

December 28, 2021

Jury members

Acknowledgments

This thesis would not have seen the day without the help of many people to whom I extend my thanks.

First of all, I must thank Luck, Fate, or whatever name is suitable for all the hidden variables which conspired to align well enough for me to see the end of this journey. From being born here and now to all the favorable winds that drove the sail, there are countless inconspicuous (hence countless) casts of dices which ended up benefiting me from the shadow they landed in.

Hopefully, all help did not go unnoticed. First and foremost, I would like to thank *Prof. Pierre Geurts*, my adviser and mentor. Working with Pierre has been a delight since day one. His encouragements have made the darkest times bearable, his kindness has been life-inspiring, while his fast thinking keeps leaving me in awe, even though I should long have been accustomed to it by now. Most impressive was his ability to quickly catch up with a ton (sometimes less, to be honest) of new information presented in a less-than-orderly fashion and still follow the reasoning and ask relevant and unlocking questions. Pierre, I thank you for the freedom you have given me and the time you took (hopefully away from questionable action movies and comedies) to follow and guide me during this thesis. It is not that the thesis would be different without you, it would simply not be.

I would also like to thank all the *jury members* for their valuable comments and feedbacks which have made this manuscript all the better. If you enjoy the reading ride laid out from here out, it is undoubtedly thanks to their insightful inputs. I must also thank them for turning the defense into such an interesting and pleasant discussion. I wish I had the time to test some of the ideas they brought about.

I could not avoid giving *Prof. Louis Wehenkel* his dedicated paragraph. Being my teacher in probability theory, information theory, and machine learning, he basically paved the way for this thesis. I remember one cigarette break where the discussion led back to independence. As a naive student, my take was that independence meant easier computation. Louis' was different: "things only become interesting when there is information to be gleaned from additional observations". His remark has stuck with me ever since.

Some more specific praises are also due to the president of the jury, *Prof. Gilles Louppe*. Although we did not work together, he has been feeding this thesis in many ways. From providing quick pointers to qualitative pedagogical material, to wroughting an elegant and easy-to-work-with codebase which served as a basis for my first contribution. He is also an inspiration in other ways, notably by being aware of so many cutting-edge works in all areas of such a quickly-evolving field as machine learning is.

My next thank-you is for *Prof. Guy Leduc*, who has been there from dawn to dusk. Without him, I would not even be a computer scientist, as he took the time to tailor my three-year transition program. He also made sure I did not need to rush through the final stage so that I could deliver this (long, I am sorry for that) thesis.

What madness would a thesis be without *colleagues*? They trade away the despairing gloom for the dining room and make breaks rhyme with “coffee” rather than “down”. Thank you to Arnaud J., Antonio, Marie S., Van Anh, Renaud, Loic, Julien, Gregoire, Chirstopher, Remy, Raphaël M., Pierre, Romain, Marie W., Laurine, Laurent, Tom, Cyril, Nicolas, Antoine, Matthia, Isabelle, Pascal, Arnaud D., Gilles, Joery, Michael, Anais, Marie L, Célia, and Emeline, Géraldine, Gaspard, Yaen, François and all the others with whom I could not spend the time they deserved.

A special thank goes out to the uncle-and aunti-gineers *Marie, Laurine and Romain*. Working with them has been inspirational and chatting delightful. Truth be told, without them, I would not have managed. Even though I did not systematically follow your advice after asking your opinions (especially when code was concerned, Romain), they always were the seeds of my thoughts.

The thesis is such that we rarely finish with the people who were there at the start. A special thank you to Arnaud, Antonio, Marie, Van Anh, Rémy, Loic, Renaud, Julien, Raphaël and Pierre for those early days at the GIGA. More than available parking lots, you were my motivation to come early and made me feel at ease from the beginning.

A special thank is due to *Raphaël Marée* who has always been supportive, even though I must have let him down when dropping from Cytomine. Raphaël, you also helped me broaden my views on many topics (even though I still believe butter is needed in a cake), a gift that will accompany me long after this thesis.

I would also like to acknowledge the help of the *Montefiore staff*, as I am guessing doing a Ph.D. without a room, furniture, access to the internet or a printer would make the task so much more challenging. A special thank you to *Sophie Cimino* whose enthusiasm might have led me to trouble her more than necessary if I were not afraid to abuse her already-generous availability.

Moving away from Montefiore, I must thank *Prof. Jean-Paul Donnay*. Before him, equations were these funny-brainy things you do in a classroom because the teacher asks you to. From the theory of map-projection alterations to krigin, passing by the first algorithms I encountered, he gave *meaning* where there were only symbols. His true gift, however, was not what he taught but how he teased what he left off. To paraphrase Antoine de Saint-Exupéry, he did not push me to build a boat, he made me yearn for the sea. Jean-Paul, I will always be in your debt for kindling the curiosity in me.

While working as a teaching assistant, I had the pleasure to work with *Prof. Pascal Gribomont*, *Prof. Pascal Fontaine* and *Prof. Laurent Mathy*. This manuscript would be different without them as they each, in their respective manners, helped me broaden my perspectives.

As for my *former students*, I must say it was delightful seeing you learn so easily, even if it meant I was hardly useful.

Moving away from the University grounds, I must thank many *friends*; too many to name exhaustively, but they will recognize themselves (or would if they read those lines, which, let's be honest, they won't). It is funny how late nights can be invigorating sometimes.

Finally, there is my *family* to thank. Mom, dad, Marjorie, Robin, Ambre, Yann, Agnès, Jean-Benoît, Xavier and Corinne. They provided invaluable support and gave me the confidence I could do anything—a helpful lie. They also tweaked many of Fate's dices so that I would not fumble too much along the way. I am also grateful for the battle they won and the mistakes they made before me so that I would not be the one making them.

Robin, you are an incredible godson and I hope I will have more time to see you grow up. For all the boredom the defense must have been to you, you kept remarkably quiet. Well done. To *Yann* (who has yet to leave me finish a story) I thank you for the support. Having an insider was more helpful than you know.

Behind every short man, there is an incredible woman. She would be *Helene Blaise*. Pregnant, dealing with a strong-willed toddler, she nevertheless managed to make sure I could pursue this goal and see the end of it in the best possible manner. We are so different that it is not always easy to understand each other. But in the great puzzle of love, the goal is to find the one-in-seven-billions piece which complements yours. On that we did great. Thank you for taking care of those so many things I am happy to let go. Soon she will have given birth to two children in half the time it took for me to do my Ph.D. How incredible is that?

For *Elise*, my not-so-baby-anymore girl, I have no words. I am not sure your many sleep-timid nights were a helpful addition but your bright smile, playfulness and relentless energy surely were—well, I guess the last one is debatable. Houses are built for shelter and homes for laughter. Thank you for turning (upside down) our house into a home. I can only wish you will experience such fulfilling journeys yourself.

While on the matter of Elise, I must also extend my thanks to all the people who lent hands. There were many but I would like to single out a few of them: her godmother, *Julie Servais*, godfather, *Jérôme Rousseau*, auntie *Coco*, as well as auntie *Laurence*.

To *Lucie*, I cannot wait to meet you. I am sure you will be awesome.

And finally, to *Albus* whose determination to keep me fit while home-working, accepting neither freezing temperature, nor torrential downpours as an excuse to avoid a walk, a good woof.

Abstract

As supervised learning occupies a larger and larger place in our everyday life, it is met with more and more constrained settings. Dealing with those constraints is a key to fostering new progress in the field, expanding ever further the limit of machine learning—a likely necessary step to reach artificial general intelligence.

Supervised learning is an inductive paradigm in which time and data are refined into knowledge, in the form of predictive models. Models which can sometimes be, it must be conceded, opaque, memory demanding and energy consuming. Given this setting, a constraint can mean any number of things. Essentially, a constraint is anything that stand in the way of supervised learning, be it the lack of time, of memory, of data, or of understanding. Additionally, the scope of applicability of supervised learning is so vast it can appear daunting. Usefulness can be found in areas including medical analysis and autonomous driving—areas for which strong guarantees are required.

All those constraints (time, memory, data, interpretability, reliability) might somewhat conflict with the traditional goal of supervised learning. In such a case, finding a balance between the constraints and the standard objective is problem-dependent, thus requiring generic solutions. Alternatively, concerns might arise after learning, in which case solutions must be developed under sub-optimal conditions, resulting in constraints adding up. An example of such situations is trying to enforce reliability once the data is no longer available.

After detailing the background (what is supervised learning and why is it difficult, what algorithms will be used, where does it land in the broader scope of knowledge) in which this thesis integrates itself, we will discuss four different scenarios.

The first one is about trying to learn a good decision forest model of a limited size, without learning first a large model and then compressing it. For that, we have developed the Globally Induced Forest (GIF) algorithm, which mixes local and global optimizations to produce accurate predictions under memory constraints in reasonable time. More specifically, the global part allows to sidestep the redundancy inherent in traditional decision forests. It is shown that the proposed method is more than competitive with standard tree-based ensembles under corresponding constraints, and can sometimes even surpass much larger models.

The second scenario corresponds to the example given above: trying to enforce reliability without data. More specifically, the focus is on out-of-distribution (OOD) detection: recognizing samples which do not come from the original distribution the model was learned from. Tackling this problem with utter lack of data is challenging. Our investigation focuses on image classification with convolutional neural networks. Indicators which can be computed alongside the prediction with little additional cost are proposed. These indicators prove useful, stable and complementary for OOD detection. We also introduce a surprisingly simple, yet effective summary indicator, shown to perform well across several networks and datasets. It can easily be tuned further as soon as samples become available. Overall, interesting results can be reached in all but the most severe settings, for which it was *a priori* doubtful to come up with a data-free solution.

The third scenario relates to transferring the knowledge of a large model in a smaller one in the absence of data. To do so, we propose to leverage a collection of unlabeled data which are easy to come up with in domains such as image classification. Two schemes are proposed (and then analyzed) to provide optimal transfer. Firstly, we proposed a biasing mechanism in the choice of unlabeled data to use so that the focus is on the more relevant samples. Secondly, we designed a teaching mechanism, applicable for almost all pairs of large and small networks, which allows for a much better knowledge transfer between the networks. Overall, good results are obtainable in decent time provided the collection of data actually contains relevant samples.

The fourth scenario tackles the problem of interpretability: what knowledge can be gleaned more or less indirectly from data. We discuss two subproblems. The first one is to showcase that GIFs (*cf. supra*) can be used to derive intrinsically interpretable models. The second consists in a comparative study between methods and types of models (namely decision forests and neural networks) for the specific purpose of quantifying how much each variable is important in a given problem. After a preliminary study on benchmark datasets, the analysis turns to a concrete biological problem: inferring gene regulatory network from data. An ambivalent conclusion is reached: neural networks can be made to perform better than decision forests at predicting in almost all instances but struggle to identify the relevant variables in some situations. It would seem that better (motivated) methods need to be proposed for neural networks, especially in the face of highly non-linear problems.

Résumé

En occupant une plus grande part dans nos vies chaque jour, l'apprentissage supervisé est de plus en plus sujet à des contraintes. La gestion de ces contraintes est un point de passage nécessaire pour promouvoir de nouveaux progrès, repoussant dès lors toujours plus les limites du domaine—une étape vraisenbablement obligatoire en vue de l'émergence d'une intelligence artificielle générale.

L'apprentissage supervisé est basé sur l'induction et transforme le temps et des données en connaissance sous la forme de modèles prédictifs. Ces modèles sont parfois, il faut l'avouer, inintelligibles, volumineux et coûteux en énergie. Par contrainte, il faut comprendre tout ce qui peut se dresser en travers d'objectif de l'apprentissage supervisé, que ça soit les impératifs temporels, les limites mémoires ou l'opacité des modèles. Par ailleurs, le champ d'application de l'apprentissage supervisé est si vaste qu'il peut en devenir intimidant. On en retrouve par exemple dans le diagnostic médical et la conduite autonome—domaine dans lesquels on est en droit d'attendre de solides garanties.

Toutes ces contraintes (le temps, la mémoire, les données, l'interprétabilité, la fiabilité) sont autant de conflits potentiels avec le but traditionel de l'apprentissage automatique. Il s'agit alors de trouver le juste milieu entre ces contraintes et l'objectif de départ. Cet équilibre dépendant des problèmes, il s'agit de proposer des solutions suffisamment génériques pour être particularisées au cas par cas. Par ailleurs, les contraintes peuvent se faire après la phase d'apprentissage initiale, ce qui peut déboucher sur des solutions sous-optimales lorsque les contraintes s'additionnent. Un exemple de ceci est la prise de conscience de besoins de fiabilité postérieure à la disponibilité des données.

Après une première partie dédiée au contexte (qu'est-ce que l'apprentissage supervisé et pourquoi est-ce difficile, quelles algorithmes vont être utilisés, où se situe l'apprentissage supervisé dans le champ plus général de la connaissance) dans lequel s'inscrit cette thèse, quatre scénarios seront examinés.

Le premier concerne l'apprentissage de forets de décisions de taille limitée, le tout sans nécessiter la création de grands modèles devant ensuite être compressés. A cette fin, nous avons développé l'agorithme dit des *Globally Induced Forests* (GIF), qui mixe optimisation locale et globale afin de produire de bonnes prédictions sous la contrainte mémoire dans un temps raisonnable. L'optimisation globale a pour but d'éliminer la redondance présente dans les forets classiques. L'analyse révèle que les GIFs sont plus que compétitives avec d'autres forêts soumises aux mêmes contraintes, pouvant parfois surpasser les modèles plus volumineux.

Le second scénario s'intéresse à la problématique soulevée plus haut, à savoir, l'ajout de fiabilité sans données. Plus précisément, le but est de détecter des exemples hors-distribution (*out-of-distribution*, OOD); des exemples qui ne sont pas tirés de la distribution d'apprentissage. Sans données, ce problème est difficile. Nous nous sommes concentrés sur la classification d'images avec des réseaux de neurones convolutifs. Nous avons proposé des indicateurs peu coûteux à calculer. Ceux-ci se sont avérés utiles, stables et complémentaires. Nous avons aussi proposé un indicateur synthétic, simple et efficace. Au final, les résultats obtenus sont plutôt bons, compte tenu de la sévérité des contraintes. Seul bémol, auquel on pouvait s'attendre, distinguer des exemples d'une distribution différente mais assez proche offre des performances mitigées.

Le troisième scénario a trait au transfert de connaissance d'un modèle volimuneux vers un plus petit en l'absence de données. Pour ce faire, nous proposons d'utiliser une base de données d'images non étiquetées, facile à établir dans des domaines comme la classification d'images. Notre méthodologie repose sur deux éléments. Premièrement, nous proposons de biaiser le choix des exemples en privilégiant ceux qui semblent les plus proche de la distribution d'apprentissage. Deuxièmement, nous proposons d'optimiser le transfert avec un mécanisme suffisamment générique pour être applicable à la plupart des paires de réseaux. Dans l'ensemble, des résultats décent sont atteignables dans des temps raisonnables, pourvu que la base de données soit parsemée de données pertinentes.

La quatrième scénario s'intéresse à la question de l'interprétabilité: que peut on apprendre, plus ou moins indirectement, des données? Deux sous-problèmes sont abordés. Le premier illustre comment les GIFs (*cf supra*) peuvent servir pour dériver des modèles intrinsèquement interprétables. Le second est une étude comparative (entre les forêts de décisions et les réseaux de neurones) portant sur la quantification de l'importance des variables dans un problème donné. Après une étude préliminaire sur des bases de données tests, les méthodes sont évaluées sur un problème de premier intérêt en biologie: l'inférence de réseaux de régulation entre gênes. La conclusion est mitigée: bien qu'il soit possible pour les réseaux de neurones de battre systématiquement les forêts d'un point de vue prédictif, la quantification d'importance reste la force des forêts, en particulier sur les problèmes hautement non-linéaires. Il semble que nous ne disposions pas encore de suffisamment bonnes méthodes (et suffisamment bien motivées) pour les réseaux de neurones.

Contents

Acronyms

AGI Artificial General Intelligence;

AI Artificial Intelligence;

ANN Artificial Neural Network;

AUC Area Under the Curve;

AUPR Area Under the Precision-Recall curve;

CNN Convolutional Neural Network;

CW Candidate Window (size);

DA Domain Adaptation;

ERM Empirical Risk Minimization;

ET Extra-Tree (short for extremely randomized tree);

FN False Negative;

FP False Positive;

GI Gradient Importance (score);

GIF Globally Induced Forest;

ID In-Distribution;

IQPR Inter-Quartile Probability Ratio;

LC Linear Classification;

LI Layer-wise Importace (score);

LR Layer-wise Relevance/Linear Regression;

LRP Layer-wise Relevance Propagation;

LS Learning Set;

MCR MisCalissification Rate;

MDA Mean Decrease of Accuracy;

MDI Mean Decrease of Impurity;

ML Machine Learning;

MLP Multi-Layer Perceptron;

MLR Multi-Class Logistic Regression;

MSE Mean Squared Error;

NFL No Free Lunch (theorems);

NLC Non-Linear Classification;

NLR Non-Linear Regression;

OLS Ordinary Least Square;

OOD Out-Of-Distribution;

OOS Out-Of-Scope;

OSR Open Set Recognition;

PCA Principal Component Analysis;

PR Precision-Recall

PTC Post-Training Constraints;

RBF Radial Basis Function;

RF Random Forest(s);

RMSE Root Mean Squared Error;

ROC Receiver-Operator Curve;

SGD Stochastic Gradient Descent;

SL Selection Layer;

SLT Statistical Learning Theory;

SRM Structural Risk Minimization;

TN(R) True Negative (Rate);

TP(R) True Positive (Rate);

VC Vapnik-Chervonenkis (dimension);

Notations

Notations are standard and introduced as needed. This quick reference describes the overall logic behind them.

Probabilistic notations $\mathbb{P}$, $\mathbb{E}$, $\mathbb{V}$, $\perp\!\!\!\perp$, $\not\perp\!\!\!\perp$ stand for probability/density measure, expectation, variance, conditional independence, conditional dependence; estimators are denoted with a hat;

Sets $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{H}$, $\mathbb{K}$ etc. denote sets (or spaces). In particular, $\mathbb{X}$ and $\mathbb{Y}$ represent the input and output spaces, respectively. $\mathbb{H}$ is the hypothesis space. $\mathbb{R}$ represents the space of real numbers, as usual;

Random variables and distribution $\mathcal{X}, \mathcal{Y}, \mathcal{I}, \mathcal{O}, \mathcal{M}, \mathcal{R}$ and other calligraphic letters represent random variables and distributions;

Vectors vectors are denoted as scalars, with lower-case letters: x, z, u. In particular, x stands for an input sample, z stands for a latent vector. $x^{(j)}$ is the jth component of vector x;

Matrices matrices are denoted with capital letters. A^T denotes the transpose of A;

Quantities n stand for the number of instances in a set, p stand for the number of variables for a problem, K stands for the number of outputs (typically, $K = 1$ for regression, and K is the number of classes in classification);

Hypotheses h, $\hat{y}$ and $\hat{p}$ are hypotheses. $\hat{y}$ outputs a (logit) vector in classification, $\hat{p}$ outputs a probability vector. h_B is the Bayes model, while h_* is the best model from the hypothesis space.

1 Chapter Introduction

Artificial intelligence (AI) is one of those topics on which people tend to have strong feelings. For some, AI is the El Dorado, solving all the problems of mankind and bringing a term to millennia of endless struggle. AI will bring better resource management, fairer justice and overall better policies in every domain by leveraging more and more information and computational foresight.

On the Yang side, some people, encouraged by many science-fiction novels or post-apocalyptic movies, are afraid of where humanity will stand in the face of "super-human intelligence". Our limited, hard-wired cognitive resources and our not-so-long-standing human hardware bodies might soon stop providing the environmental assets necessary to withstand the inexorable, relentless and unforgiving ticking of time.

Yet others are scared of the coming together of humanity and technology in a merging fashion—oblivious at how far we have already traveled along that path—blurring further already fuzzy definitions of what we are.

A minimum requirement to bring about such futures would be to reach what is sometimes referred to as artificial general intelligence (AGI). Recognizing AGI is a notoriously difficult and tricky problem, notably because of intelligence being (defined as) a human-centric concept. The most famous detector of AGI was proposed by Turing (1950) and is known as the imitation game (or the Turing test) wherein a human examiner is tasked with deciding which of its two remote interlocutors is an artificial agent. If an artificial agent was reliably able to pass the test (*i.e.* could fool the human 50% of the time), AGI could be declared reached.

Although impressive chatbots, dedicated to the sole purpose of deceiving the Turing tester, exist nowadays, they lean more on the artificial side than on what we would truly consider as intelligent. As things stand at the time of writing this thesis, both extreme views relating to AI, whether optimistic or less so, are only shades on a distant horizon, useful to gear and steer, but otherwise belonging to the realm of utopia or dystopia.

Notwithstanding these remarks, AI has come a long way since its debut. A good monitor of AI progress has always been the stage of games. In 1997, IBM's Deep Blue beat the famous world champion, Garry Kasparov, at chess. In 2005, five experimental autonomous driving vehicles were able to complete the DARPA Grand Challenge, an off-road course. In 2007, the competition was set in an urban setting. In 2011, IBM's Watson program wins the Jeopardy! quiz show: the program is able to understand questions asked in natural language and answer faster than the then-champions. In 2016, DeepMind's AlphaGo was able to beat the 9-dan player Lee Sedol at go. The next year, AlphaGo Zero, a self-taught system, was able to beat AlphaGo. The current benchmark for progress in games is Starcraft II where "AlphaStar was rated at Grandmaster level for all three StarCraft races and above 99.8% of officially ranked human players" (Vinyals et al., 2019).

Looping on what Alan Turing advocated, recent progress has been achieved thanks to learning, allowing the artificial agent to construct a model of (part of) the world for itself, rather than trying to program a solution from first principles. Machine learning is but one tool (with many flavors) in the AI toolbox, yet it nowadays stands as the most promising stepping stone (or giant's shoulder) toward AGI.

In the meantime, machine learning is chaining successes on smaller scales. From speeding up the digital transition of older businesses to laying the groundwork for brand-new companies, from alleviating the workload of technical experts to opening new venues, machine learning is rapidly nesting itself as the spearhead for new progress and prowess.

This success, machine learning owes it to two factors, the first of which being its generality and versatility. Whether medical diagnosis, quality control, forecasting, robotics, decision planning, spam filtering, recommendation systems, fraud detection, translation, natural language processing, text-to-speech, sentiment analysis, style transfer, biometric authentication, autonomous driving, if it can be learned from data or experiments, machine learning can help. The second factor relates to how learning happens: via data, which is now more plentiful than ever before. From done to dawn, those dormant entries have left the back-row of databases to become anonymous forerunners of new development.

As machine learning is willed into the wild, reality strikes back, with new challenges emerging. This is somewhat reminiscent of prototypes leaving an R&D department to become final products. Robustness, fairness, computational power, energy consumption, data quality, availability, privacy, reliability and so on become forefront issues. Although, some of these are not new, pushing ever further the boundary of how and for what machine learning can be applied comes at the price of more and more constrained settings. On the other hand, as flaws and vulnerabilities are discovered, new concerns arise. For instance, robustness has regained much attention following the discovery of what is called "adversarial examples" in deep learning, while fairness and privacy were almost unheard of a decade ago. Therefore, machine learning must adapt to these new challenges.

Although these constraints arise far from the question of artificial general intelligence, they are still—some would even say more crucially—relevant to AGI. These new and rekindled winds have steered the machine learning community to new directions to ensure machine learning can still operate under today's constraints and will later meet the requirements of tomorrow.

The present thesis is one such foray.

1.1 Contributions

In this thesis, we have tackled four problems from the field of supervised learning. Informally, supervised learning is about learning a predictive model from a collection of data (a more formal definition will be given in the next chapter). The specific problems we will look at are a mix of five constraints. Constraints will be formally defined in Chapter 5, once all the relevant concepts are laid out. Informally, a constraint is anything which stands in the way of supervised learning. The five constraints we will look at are

fast training: how to propose learning algorithm which completes in a reasonable amount of time;

small models: how to end up with predictive models which are lightweight, fast and energy-efficient;

robust models: how to ensure robustness in the form of out-of-distribution detection (recognizing data which does not originate from the same source as the one used to learn);

data-free training: how to learn when no data is available to learn from;

interpretable methods: how to understand what is actually learned.

We will now inspect the four problems examined in this thesis.

Fast training of small decision forest models by design. In Chapter 6, we introduce the **G**lobally **I**nduced **F**orests (GIF) algorithm, whose goal is to produce small yet accurate decision forests by limiting the total number of nodes in the forest. The algorithm is fast in the sense that it does not require building a large forest to compress it. GIFs are also shown to be effective to produce interpretable models (Chapter 9).

Data-free and fast post-training robustness. In Chapter 7, we study robustness and more precisely the problem of out-of-distribution (OOD) detection. OOD detection aims at identifying samples which do not originate from the training distribution. Our contribution was to investigate what can and cannot be done in the absence of data. Our solution is also fast in the sense that it requires only a small amount of additional computation besides making the prediction.

Data-free and fast deep model compression. In Chapter 8, we focus on how to train a small neural network without data. For that, we assume the availability of a larger network we would like to compress and a collection of unlabeled data among which relevant ones are present. The method is fast in the sense that training time is close to what would have been required were data available (compared to alternative methods).

Interpretability: small interpretable models and feature importance scores. In Chapter 9, we tackle interpretability. Firstly we show how GIFs can be used to produce small and interpretable models. Secondly, we look at the problem of quantifying the importance of each variable in a given problem. More precisely, we conducted an empirical study to see how neural networks fare compared to decision forests, held as state-of-the-art for this purpose.

1.2 Outline

This manuscript is articulated around three parts.

Part I: supervised learning. The first part covers supervised learning, the general domain to which we contributed. More precisely

Chapter 2 describes supervised learning, what it aims to do, how it tries to do it (at a high level), why it is challenging and how these challenges can be overcome.

Chapter 3 is about learning algorithms: concretely how can a predictive model be learned from data. We will review all the algorithms which play a part in this thesis, from simple linear methods to decision forests and deep learning.

Chapter 4 situates how supervised learning fits into the broader landscape of knowledge and discusses a few philosophical questions.

Part II: constraints and contributions. The second part covers our contributions regarding supervised learning under constraints, with the following outline.

Chapter 5 delves into constraints in supervised learning: what they are, why they matter and how can they be categorized.

Chapter 6 focuses on small decision forests, why they matter, what makes it possible to obtain small-yet-accurate forests, and how the problem can be, and has been, tackled. Therein we present our Globally Induced Forests (GIF) algorithm.

Chapter 7 discusses the problem of out-of-distribution detection, how it can be formulated, what other problems it resembles, and how it can be solved. The remainder of the chapter focuses on the sample-free case.

Chapter 8 opens with a discussion of why achieving small and accurate models is feasible in the context of deep learning, and then proceeds by looking at the various way of how it can actually be done. The remainder of the chapter describes our sample-free approach to transfer knowledge from a bulk network into a smaller one.

Chapter 9 discusses the topic of interpretability, what is meant by that, what forms it can take, and how it may bias model selection. The first half of the chapter re-casts GIFs (Chapter 6) as interpretable models or interpretation extractors. The second half manages interpretability as a post-training goal where the challenge is to detect which variables are most useful to predict the output. It compares decision forests and deep learning on toy datasets and on the challenging task of reconstructing gene regulatory networks from gene expression data, an open problem in computational biology.

Part III: conclusion. Finally, the last part concludes this thesis in Chapter 10.

Appendices.

Appendix A is dedicated to `Clustertools` a toolbox developed during this thesis and whose goal is to facilitate the empirical analysis of (machine learning) algorithms.

Appendix B contains additional results relating to the problem of out-of-distribution detection we studied (Chapter 7).

Reading guide. Although we have tried to piece an engaging narrative, the background part (Chapters 2, 3 and 4) remains quite standard. The knowledgeable reader may safely skip those, with the possible exception of Sections 3.2.2.2 and 3.6.2.1. Those sections give some geometrical insights useful for Chapter 7 about out-of-distribution.

The contribution chapters (excluding the introductory Chapter 5) can be read in any order with two exceptions. Firstly, Chapter 8 builds upon the summary indicator proposed in Chapter 7. Secondly, the first part of Chapter 9 comes back to the GIFs, the core of Chapter 6.

1.3 Publications

The following publications have been made during this thesis:

- Marée, R., Rollus, L., Stévens, B., Hoyoux, R., Louppe, G., Vandaele, R., **Begon, Jean-Michel**, Kainz, P., Geurts, P., & Wehenkel, L. (2016). Collaborative analysis of multi-gigapixel imaging data using Cytomine. Bioinformatics, 32(9), 1395-1401. (not covered in this thesis)

- Mormont, R., **Begon, Jean-Michel**, Hoyoux, R., & Marée, R. (2016). SLDC: an open-source workflow for object detection in multi-gigapixel images. Proceedings of the 25th Belgian Dutch Conference on Machine Learning (Benelearn 2016). (not covered in this thesis)

- **Begon, Jean-Michel**, Joly, A., & Geurts, P. (2016). Joint learning and pruning of decision forests. Proceedings of the 25th Belgian Dutch Conference on Machine Learning (Benelearn 2016). (Preliminary work on Globally induced forest)
- **Begon, Jean-Michel**, Joly, A., & Geurts, P. (2017, July). Globally induced forest: A pre-pruning compression scheme. In International Conference on Machine Learning (pp. 420-428). PMLR.
  `https://github.com/jm-begon/globally-induced-forest`
- Vecoven, N., **Begon, Jean-Michel**, Sutera, A., Geurts, P., & Vân Anh Huynh-Thu (2020, October). Nets versus trees for feature ranking and gene network inference. In International Conference on Discovery Science (pp. 231-245). Springer, Cham.
- **Begon, Jean-Michel**, & Geurts, P. (2021). Sample-Free White-Box Out-of-Distribution Detection for Deep Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 3290-3299), presented at the fair, data efficient and trusted computer vision workshop.
  `https://github.com/jm-begon/ood_samplefree`
- **Begon, Jean-Michel**, & Geurts, P. (2021) Distillation from heterogeneous unlabeled collections. Currently under review.
  `https://github.com/jm-begon/distill_by_transfer`

Machine learning

Part I

2 Chapter Supervised learning

Chapter overview

This chapter is dedicated to machine learning and especially supervised learning: what it is and why it is an important and difficult topic.

After formalizing supervised learning (Section 2.2), we will state a first general strategy: the empirical risk minimization (ERM) in Section 2.3. Several sections will then be dedicated to the central problem of supervised learning: overfitting. Overfitting will first be addressed informally (Section 2.4) before getting under the microscope of two theoretical frameworks: the bias-variance decomposition (Section 2.5) and statistical learning theory via the notion of bounds on the generalization gap (Section 2.6). This will also offer the opportunity to define the important concept of expressiveness and state some interesting results. After discussing all these results in Section 2.7, we will address how to (attempt to) solve overfitting. The first technique we will mention is called regularization (Section 2.8.1), which can either be explicit or implicit. The second technique, which is not limited to solving overfitting, is a post-processing method to select a model (Section 2.8.2). This concludes the tour of supervised learning, offering some room in Section 2.9 for a few words about other paradigms from the machine learning toolbox.

Machine learning is a vast field, encompassing several widely different paradigms. Grouping them together under the denomination of "machine learning" emphasizes two common characteristics of those techniques.

Firstly, it underlines their algorithmic nature: "learning" is not something which happens magically, but rather it is the careful combination of (billions of) calculation steps. It also brings forth the whole question of computability: what is "learnable"? what is *efficiently* "learnable"? by *what* is it "learnable"?

Answering these requires defining what "learning" means, a challenging task in full generality. Informally, there is this idea that new knowledge will be acquired by the end of the computation. Since this knowledge cannot sprout from nothingness, it must originate from the computation's input: the data. Thus machine learning is about extracting knowledge from data in an algorithmic way. What is the nature of the input data and what kind of knowledge are we seeking to delineate the different machine learning paradigms.

This thesis focuses on *supervised* learning, of which we give a more formal definition below. We will contrast supervised learning against other paradigms in Section 2.9.

Same tools, different toolboxes Over the years and trends, the algorithms we will address in this thesis have moved from one toolbox to another. Disciplines such as pattern recognition, data mining, inductive learning, computational statistics, predictive analytics, or artificial intelligence reflect different flavors sharing a common trend that knowledge is buried in data, waiting to be gathered.

2.1 Illustration

Before giving a formal definition of supervised learning, let us consider a toy example: the iris dataset from none other than R.A. Fisher (Fisher, 1936). It consists of 50 sepal and petal measurements (length and width) for three varieties of iris flowers. We will somewhat divert the original problem for the purpose of illustration.

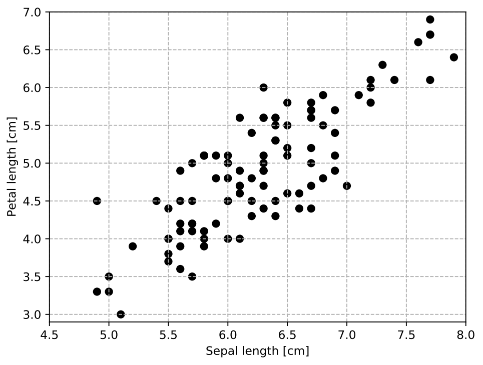
Consider Figures 2.1a and 2.1b offering the two following problems: (i) given the sepal length, is it possible to guess at the petal length? and (ii) is it possible, given the sepal and petal lengths to determine the type of flowers?

Visually both problems seem addressable, although somewhat imperfectly. Solutions to these problems are given in Figure 2.2. In Figure 2.2a, we see two "solutions". The full, red line seems to better represent the relationship we are trying to capture. This solution is, evidently, imperfect. For instance, an iris flower with a sepal of 5 cm might have a petal of 3.5 cm or less, whereas both "solutions" advocate for at least 3.6 cm.

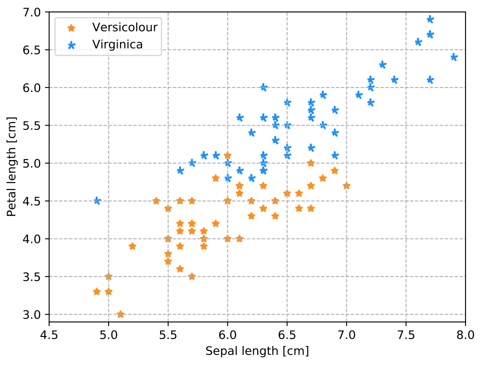
Similarly, the straight line of Figure 2.2b separates the versicolour and virginica well, except at less than 0.5 cm of petal length from the boundary.

Trying to find how to best separate labeled datapoints (such as versicolour vs virginica species) or to find the best relationship between variables (such as sepal and petal lengths) is what supervised learning is all about. Note that, contrary to these examples, there is no limitation to have continuous input variables (such as lengths), to restrict to one or two variables, or to focus on a linear boundary and relationship. For instance, Figure 2.2c illustrates that we can use the species to improve the sepal-to-petal length relationship. It thus relies on a new variable, the species, and the model is no longer simply linear but hierarchical.

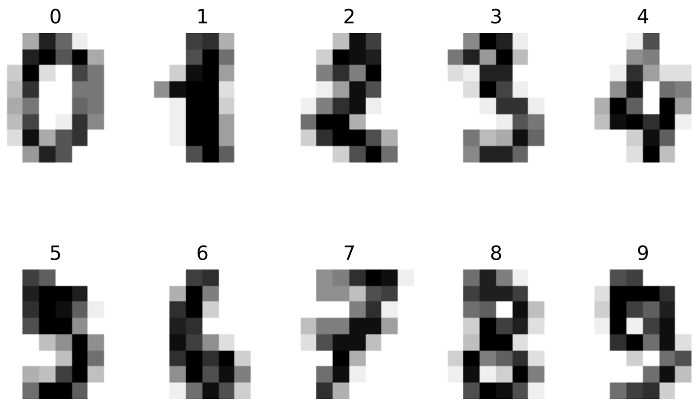
To explore the specificity of higher dimensional relationships, we will also look at a handwritten digit recognition problem (Blake and Merz, 1998). This dataset consists of 1797 8×8 grayscale images of digits (roughly 180 images per digit), and the goal is to recognize which digit appears on each image. Examples are given in Figure 2.1c.



(A) Iris dataset: sepal vs. petal length (versicolour and virginica species).



(B) Iris dataset: versicolour and virginica species (in a sepal-petal lengths space).



(C) Digit dataset: examples of samples.

FIGURE 2.1: Toy dataset for illustration purposes.

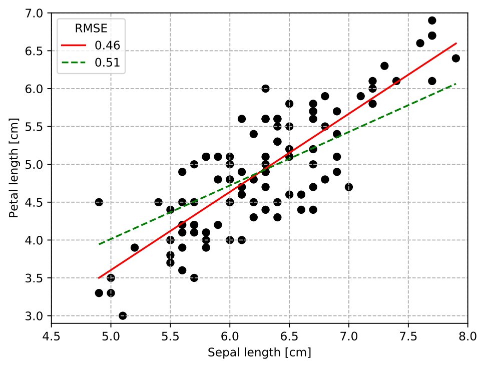
We will now formalize supervised learning, refining the notion of "problem" and "best" (or good) solutions.

2.2 Formalization

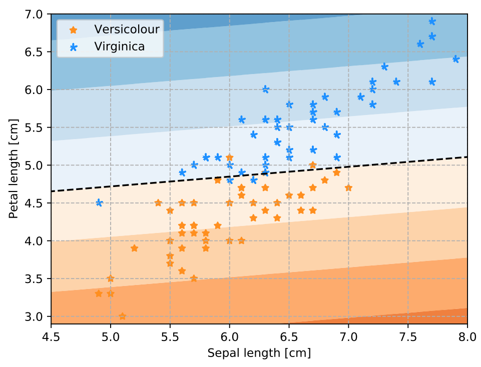
2.2.1 Problem structure

Let $\mathbb{X}$ and $\mathbb{Y}$ be two sets, where $\mathbb{X}$ represents the known input space and $\mathbb{Y}$ the output space. For instance, in the sepal-to-petal problem, $\mathbb{X} \subseteq \mathbb{R}$ refers to the sepal length and $\mathbb{Y} \subseteq \mathbb{R}$ refers to the petal length. In the species classification problem, $\mathbb{X} \subseteq \mathbb{R}^2$ refers to the pair sepal-petal lengths and $\mathbb{Y}$ refers to the species (*i.e.* $\mathbb{Y} = \{\text{versicolour}, \text{virginica}\}$).

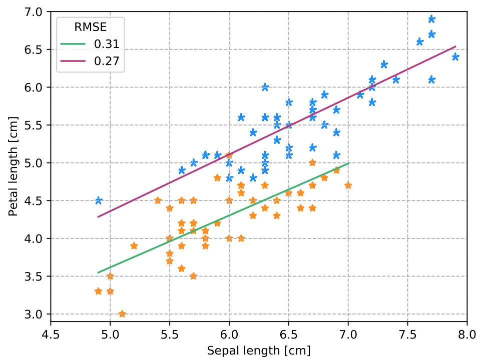
A pair $(x, y) \in (\mathbb{X} \times \mathbb{Y})$ is qualified of labeled. x is a p-dimensional vector where each component represents a (independent) variable. y represents



(A) Possible linear relationships between sepal and petal lengths. The red line captures better the relationship between the lengths.



(B) Possible linear boundary between iris species. The shades reflect the "confidence" of the model. In this case, it relates to how far from the boundary points fall.



(C) Improving the sepal-to-petal length relationship by using the species. Given the species, it is possible to use a hierarchical linear model to obtain species-dependent predictions. The two models portray a lower RMSE than the species-agnostic models of Figure 2.2a.

FIGURE 2.2: Solutions to the sepal-to-petal and versicolour-or-virginica problems.

the dependent variable. When the dependent variable is discrete, the problem is termed as "classification", with element of $\mathbb{Y}$ being called classes, or labels. When the dependent variable is continuous the problem is termed as "regression". When y comes from observations, it is referred to as ground truth.

In both regression and classification, we are trying to capture some relationship between $\mathbb{X}$ and $\mathbb{Y}$. More precisely we are looking for a function $h : \mathbb{X} \to \mathbb{Y}$ (what we called a "solution" until now). h is referred to as the hypothesis, the model, the function, the predictor, the regressor/classifier or the estimator. What comes out of h is referred to as the output or the prediction.

2.2.2 Data

We said that machine learning is about gleaning knowledge from data. Those data points are drawn according to some distribution. Formally, let us consider a probability space $(\Omega, \mathcal{E}, \mathbb{P}_\Omega)$, where $\mathbb{P}_\Omega$ is a probability measure over $\mathcal{E}$, the σ-algebra over the events Ω. On this space are defined a random vector $\mathcal{X} : \Omega \to \mathbb{X}$ and a random variable (or sometimes vector) $\mathcal{Y} : \Omega \to \mathbb{Y}$. We will usually assume that all densities relating to $\mathcal{X}$ and $\mathcal{Y}$ exist, denoting $\mathbb{P}_\mathcal{X}$, $\mathbb{P}_\mathcal{Y}$, $\mathbb{P}_{\mathcal{Y}|\mathcal{X}}$ and $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$ the marginals, conditional and joint densities respectively.

Supervised learning mostly deals with identically and independently distributed (iid) samples. We suppose that a collection of samples, the learning set, is available to learn from.

Definition 2.2.1 (Learning set). *A learning set $LS = \{(x_i, y_i)|1 \leq i \leq n\}$ is such that $LS \sim \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}$. This is the set on which the learning will occur.*

Generality of supervised learning. Now that some components of supervised learning are well defined, we can appreciate how its modeling assumptions make it general.

The inputs can be vectors, images, sounds, texts, videos, graphs, etc. Admittedly, the algorithms are designed to work with vectors and some preprocessing might be needed *in practice* to cast data into an appropriate type.

Regression and classification cover a wide range of applications. Binary classification actually allows encompassing detection/recognition problems. Overall, supervised learning might be used to tackle problems such as reading (*i.e.* recognizing) handwritten writing, understanding a scene (recognizing and locating objects in a video frame), diagnosing, detecting cancer cells in medical imaging, and so much more.

2.2.3 Hypothesis space

The function $h : \mathbb{X} \to \mathbb{Y}$ we are trying to uncover cannot be drawn out of nothing; it must be taken from a set of candidate hypotheses.

Definition 2.2.2 (Hypothesis space (or model class)). *The set of candidate hypotheses considered by a learning algorithm is called the hypothesis space and denoted by $\mathbb{H}$.*

(Non-)parametric hypothesis spaces. A simple way to build a hypothesis space is to consider a family of parametrizable functions. For instance, the set $\{x \to mx + p|(m, p) \in \mathbb{R}^2\}$ of one-input linear functions is parametrized by the slope m and intercept p.

Definition 2.2.3 (Parametric hypothesis space). *A parametric space is such that $\mathbb{H}_\Theta = \{h(\cdot, \theta)|\theta \in \Theta\}$ where $h : \mathbb{H} \times \Theta \to \mathbb{Y}$.*

Hypothesis spaces are not necessarily parametric.

Learning a hypothesis from a parametric space can be viewed as an in-place operation (especially with iterative algorithm starting). Therefore, it is not uncommon to say that a *model* of a given *shape* is *trained* rather than an *hypothesis* is *learned* from a hypothesis *space*.

Definition 2.2.4 (Decision boundary). *In the case of classification, the set of points* $\{x \in \mathbb{X} | \forall \eta \exists \varepsilon : h(x+\varepsilon) \neq h(x-\varepsilon), 0 < ||\varepsilon|| < \eta\}$ *is referred to as the boundary of* h.

In words, the decision boundary corresponds to points of $\mathbb{X}$ where a small shift changes the predicted class.

2.2.4 Loss function

Whether a function is good at capturing the relationship between $\mathbb{X}$ and $\mathbb{Y}$ is captured through the notion of a loss function.

Definition 2.2.5 (Loss function). *A loss function* $\ell : (\mathbb{Y} \times \mathbb{Y}) \to \mathbb{R}^+$ *abides by the following property:*

$$\ell(z, y) = 0 \iff y = z \tag{2.1}$$

and quantifies how much its inputs are dissimilar (the higher the value, the more dissimilar the inputs).

As such, distances and metrics make appropriate loss functions. The symmetry and triangle inequality properties are not strictly necessary, however.

Regression. The most widely used loss function in the case of regression is the squared loss ℓ_2

Definition 2.2.6 (Squared loss).

$$\ell_2(z, y) = L_2(y - z) = ||y - z||_2^2 = \sum_j (y^{(j)} - z^{(j)})^2 \tag{2.2}$$

where L_2 *is the 2-norm squared and* $y^{(j)}$ *is the jth component of vector* y.

The $\epsilon = y - z$ vector is called the residual. Figure 2.3 illustrates the residuals.

Classification. An obvious choice of loss for classification is the $0-1$ loss.

Definition 2.2.7 ($0-1$ loss).

$$\ell_{0-1}(z, y) = \mathbb{I}(z \neq y) = \begin{cases} 0, & \text{if } z = y \\ 1, & \text{otherwise} \end{cases} \tag{2.3}$$

For instance, there are six wrongly classified points (*i.e.* $\ell_{0-1}(z, y) = 1$) in Figure 2.2b.

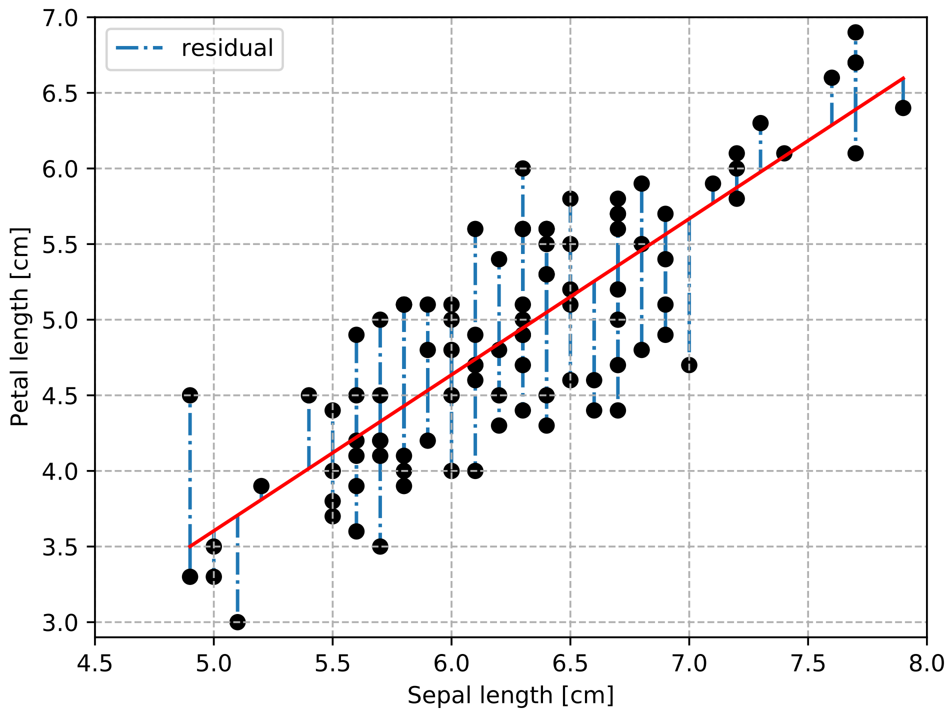


FIGURE 2.3: Residuals for the sepal-to-petal regression problem.

Error/risk. Typically, the loss ℓ is used to compute how (in)adequate the prediction of a model h is. Given a sample $(x, y) \in \mathbb{X} \times \mathbb{Y}$, $\ell(h(x), y)$ is low (resp. high) if the model makes a good (resp. bad) prediction. The actual value of the loss quantifies this goodness (actually its badness). Usually, however, we are not interested in how well the model predicts a single point, but rather how it behaves on average over the data distribution. We thus define the following quantities of interest.

Definition 2.2.8 (Expected risk). *The expected risk of a given hypothesis h under loss ℓ is defined as*

$$\mathcal{R}_{\mathcal{X},\mathcal{Y}}(h; \ell) = \mathbb{E}_{\mathcal{X},\mathcal{Y}}\{\ell(h(x), y)\} \tag{2.4}$$

where $\mathbb{E}_{\mathcal{X},\mathcal{Y}}$ is the expectation over the joint distribution of $\mathcal{X}$ and $\mathcal{Y}$.

Synonyms for the expected risk are actual or theoretical errors, and generalization error.

Given a realization of the dataset, it is possible to compute a (possibly biased) estimate of the expected risk.

Definition 2.2.9 ((Empirical) error)**.** *Given a set* $\mathbb{S} = \{(x_i, y_i) \in (\mathbb{X} \times \mathbb{Y}) | 1 \leq i \leq n\}$*, the empirical error is*

$$E_{\mathbb{S}}(h;\ell) = \frac{1}{|\mathbb{S}|} \sum_{(x,y)\in\mathbb{S}} \ell(h(x), y) \tag{2.5}$$

where $|\mathbb{S}|$ *is the cardinality of* $\mathbb{S}$*. This is simply the average loss over* $\mathbb{S}$*.*

The name of the empirical error is usually refined by involving the set on which it is computed. For instance, $E_{LS}(h)$ is the training (set) error. Furthermore, when h is the hypothesis learned on LS, it is called the resubstitution error. When $\mathbb{S}$ is a disjoint set from LS, $E_{\mathbb{S}}$ is called the test set error (see also Section 2.4.1 about unbiased assessment of the expected risk).

The empirical *error* must not be confused with the empirical *risk*, which is a random variable capturing the stochasticity of estimating the empirical error with a dataset of size n.

Definition 2.2.10 (Empirical risk)**.**

$$\hat{\mathcal{R}}^{(n)}_{\mathcal{X},\mathcal{Y}}(h;\ell) = \frac{1}{n} \sum_{(x,y)^n \sim \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}} \ell(h(x), y) \tag{2.6}$$

this random variable corresponds to the empirical error over a set of size n drawn iid from the joint data distribution.

The dependency on $\mathcal{X}, \mathcal{Y}$ and/or ℓ will be dropped when the context is clear.

Finally, let us introduce some common error functions.

Definition 2.2.11 ((Root) mean square error)**.** *In regression, when using the squared loss, the error is qualified as mean square error (MSE). It is also common to use the square root of MSE as an error measure. This is known as the root mean square error (RMSE).*

Definition 2.2.12 (Misclassification rate)**.** *In classification, when using* ℓ_{0-1}*, the error becomes the misclassification rate (MCR).*

Definition 2.2.13 (Accuracy)**.** *The accuracy is defined as* $1 - E_{\mathbb{S}}(h;\ell_{0-1})$*.*

Note that used solely, the word "loss" might refer to either the loss function or the loss value. In the latter case, it might even refer to the loss *over* some set/distribution, that is the error. Hopefully, the context will shed enough light to overcome this polysemic pitfall.

2.2.5 Goal and Bayes model

Given a learning problem $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$, and a loss function ℓ, we define the following minimizer as the best possible model.

Definition 2.2.14 (Bayes model).

$$h_B = \arg\min_{h \in \mathbb{F}} \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h, \ell) = \arg\min_{h \in \mathbb{F}} \mathbb{E}_{\mathcal{X},\mathcal{Y}}\{\ell(h(x), y)\} \tag{2.7}$$

where $\mathbb{F}$ is the set of all $\mathbb{X} \to \mathbb{Y}$ functions.

Conversely, we can define the best hypothesis with respect to the hypothesis space $\mathbb{H}$.

Definition 2.2.15 (Best hypothesis).

$$h_* = \arg\min_{h \in \mathbb{H}} \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h, \ell) = \arg\min_{h \in \mathbb{H}} \mathbb{E}_{\mathcal{X},\mathcal{Y}}\{\ell(h(x), y)\} \tag{2.8}$$

Since $\mathbb{H} \subseteq \mathbb{F}$, it might happen that $h_B \notin \mathbb{H}$ and therefore $h_* \neq h_B$. In such a case, even with infinite data and irrespective of how it is selected, the hypothesis will never be optimal in the Bayes model sense. The hypothesis space is said to suffer from representational bias.

Given all these, we can now state the goal of supervised learning.

Definition 2.2.16 (Goal of supervised learning.). *Given data drawn from $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$, a loss function ℓ, and a hypothesis space $\mathbb{H}$, the goal of supervised learning is to minimize the expected risk, i.e. find h_*, under reasonable time.*

2.2.6 Learning algorithm

Supervised learning is about investigating data to find a hypothesis which models well (in the sense of some loss) some distributional relationship(s). Selecting the hypothesis is the role of the learning algorithm.

Definition 2.2.17 (Learning algorithm). *Formally, a learning algorithm is a function*

$$\mathcal{L} : (\mathbb{X} \times \mathbb{Y})^{l+} \to \mathbb{H} \tag{2.9}$$

It takes in a learning sample of size at least l and outputs a hypothesis $h \in \mathbb{H}$.

The process of selecting the hypothesis according to the data is called training or learning.

Hyper-parameters. Often, learning algorithms come in the form of a family, with each member parametrized by some so-called hyper-parameter(s) $\phi \in \Phi$. For instance, hyper-parameters can encode the structure of a neural network, the maximum depth of a decision tree, the number of neighbors for nearest neighbor classifiers, etc. Note that hyper-parameters relate to the learning algorithm, while parameters relate to the hypothesis (space). There is somewhat of a gray area between those two extremes when hyper-parameters influence parameters.

Randomization and learning algorithms. Two sources of randomization come into play in the context of learning algorithms. These are from very different natures and must not be confused.

On the one hand, a learning algorithm naturally becomes a random variable when randomizing the learning set it gets. We will denote by $\mathcal{H}_{\mathcal{L}_\phi, \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}$ the distribution over the hypotheses induced by the random sample of n datapoints drawn from $\mathbb{P}^{(n)}_{\mathcal{X},\mathcal{Y}}$ under the learning algorithm $\mathcal{L}_\phi$. We will shorten the notation to $\mathcal{H}^{(n)}_\phi$ when there is no ambiguity regarding the learning algorithm and the distribution, and altogether drop ϕ when there are hyper-parameters involved.

On the other hand, a *randomized* algorithm generates a distribution of hypotheses for a fixed learning set. This happens when model parameters are instantiated (pseudo-)randomly, or when some computation steps require subsampling, for instance. An elegant way to treat this randomness is to assume it originates from a pseudo-random generator whose seed is a hyper-parameter (thus included in ϕ). Assuming some distribution over the seed $\phi \sim \mathfrak{F}$, we will use the previously defined notations substituting ϕ for $\mathfrak{F}$ to denote this randomized version.

A quantity of interest is the expected risk of the *learning algorithm*.

Definition 2.2.18 (Expected risk of a learning algorithm)**.**

$$\mathcal{R}^{(n)}_{\mathfrak{F},(\mathcal{X},\mathcal{Y})}(\mathcal{L}_\phi; \ell) = \mathbb{E}_{\mathcal{H}_{\mathcal{L}_\mathfrak{F}, \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}} \left\{ \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h; \ell) \right\} \tag{2.10}$$

This reflects the average risk incurred by the randomization of both the learning set and the algorithm.

Since the two sources of stochasticity are independent, they can be factorized. As a consequence, only the contextually-relevant source(s) will appear in the notations in the remaining. For instance, when the algorithm does not have hyper-parameters (or those are not discussed) the expected risk will be shortened as

$$\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(\mathcal{L}; \ell) \tag{2.11}$$

2.2.7 Changing input spaces: feature engineering and learning

Most learning algorithms are developed under the assumption that $\mathbb{X} \subseteq \mathbb{R}^p$. However, not all data types fall into that category. For such cases, some steps must be taken in order to cast the data into an appropriate form in order to take advantage of the available learning algorithms. Even when datapoints are vectors, it might be advantageous to adopt another representation. This does not produce information in any way but makes it more leverageable for the algorithm.

For instance, in the case of the digit recognition problem, images can be turned into vectors quite naturally. Images are represented as a 2D, 8×8

array of values representing the gray level. Flattening the array immediately yields a 64-dimensional vector. Alternatively, one can switch to a kernel space. Assume there are m reference images $\{x_i\}_{i=1}^m$, one derive, for any x a new vector $k(x) \in \mathbb{R}^m$ such that the jth component is $k(x)^{(j)} = \exp\left(-\gamma||x_j - x||^2\right)$. This is known as the Gaussian kernel, or radial basis function (RBF) kernel. It is such that, as x moves away from x_j, the component decreases (exponentially fast) toward zero, with $0 < \gamma \leq 1$ controlling the speed of the decrease. More precisely, when γ is high, only local information is taken into account, as most components (the ones relating to faraway basis vectors) will be close to zero. As γ decreases, more and more global information is taken into account.

This kernel representation is better suited for a linear model (for instance), where the interpretation is straightforward (the weighted sum measures how distant x from the m points, possibly favoring some reference points via the weights). In comparison, using a linear boundary to separate the images in the original flatten space is far less intuitive.

This process of changing the input space is called feature engineering. Alternatively, the features can be derived from the base representation automatically (*i.e.* learned), a process known as feature/representation learning and the basis of deep learning.

2.2.8 Changing output spaces: alternative representations (classification)

The discrete nature of the output space in classification is not quite as amenable to work with as in regression. Therefore, hypotheses work usually in two steps: $h = h_1 \circ h_2 : \mathbb{X} \to \mathbb{Y}' \to \mathbb{Y}$. Learning the hypothesis usually amounts to learning h_2, with h_1 being a trivial overlay.

For instance, in binary classification, the intermediate output $y' \in \mathbb{Y}'$ might be a real number and h_2 is just a thresholding function which translates the real-value output y' to an actual class of $\mathbb{Y}$. Another common alternative is to output a probability vector y' describing how confident the model is of the input belonging to each class. In that case, h_2 usually comes down to yielding the y of highest probability. The learning algorithms described in Chapter 3 make use of those representations.

These representations offer more flexibility to build algorithms upon by shifting the loss to some continuous alternatives. One such loss function is the cross entropy. Let $\mathbb{Y}'$ be the set of K-dimensional probability vectors, that is $\mathbb{Y}' = \{\hat{p} \in \mathbb{R}^K | \hat{p}^{(j)} \geq 0 \wedge 1 \leq j \leq K, \sum_{j=1}^K \hat{p}^{(j)} = 1\}$, and let us encode y in a vector p such that $p^{(k)} = 1$ if y is the kth class, and 0 otherwise.

Definition 2.2.19 (Cross entropy loss).

$$\ell_{CE}(\hat{p}, p) = -\sum_{j=1}^{K} p^{(j)} \log \hat{p}^{(j)} \tag{2.12}$$

The cross-entropy loss is a member of a larger information-theoretic family of loss functions (including Jensen–Shannon divergence and Kullback–Leibler divergence).

As is evident with this example, output values must be encoded to match the structure of $\mathbb{Y}'$ and be compatible with the loss function. Since this is on a per learning-algorithm-basis, such encoding will be discussed when describing the algorithm.

Considering the triviality of the second phase (thresholding, selection according to the maximum probability), the focus will be on the first phase and $y \in \mathbb{Y}$ will be used instead of $y' \in \mathbb{Y}'$ to ease notation. $\hat{y}$ and $\hat{p}$ will be used for h_2 to highlight that the output of the "raw" hypothesis is a real value or a probability vector, respectively.

2.3 Empirical risk minimization

So far, little has been said regarding how to practically choose the hypothesis given a learning sample. A first, apparently sound solution would be to select the hypothesis with lowest training set error. This gives rise to the following principle.

Definition 2.3.1 (Empirical risk minimization (ERM) principle)**.** *The ERM principle states that, given a learning sample $LS = \{(x_i, y_i)\}_{i=1}^n$ and a loss ℓ, the hypothesis*

$$\hat{h}_*^{(n)} = \arg\min_{h \in \mathbb{H}} E_{LS}(h; \ell) \tag{2.13}$$

should be selected.

For instance, in the sepal-to-petal regression problem, if the hypothesis space corresponds to the set of linear functions $\mathbb{H} = \{x \to mx + p | (m, p) \in \Theta = \mathbb{R}^2\}$, the ERM principle can be implemented as

$$(m, p) = \arg\min_{(m,p) \in \mathbb{R}^2} E_{LS}(x \to mx + p \quad ; \ell) \tag{2.14}$$

The goal of supervised learning is to select a hypothesis which has low *expected* risk, however. For the ERM principle to be effective, it must be *consistent*.

Definition 2.3.2 (Consistency properties)**.** *A learning algorithm is* consistent *if the random sequence of chosen models with respect to the learning set size n is such that its expected and empirical risks converge in probability to the expected risk of the true minimizer of the hypothesis space. That is,* $\forall \epsilon, \alpha > 0, \exists n_0$:

$$\begin{cases} n \geq n_0 \implies \mathbb{P}_{h \sim \mathcal{H}^{(n)}} \left(\left| \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_*) \right| \geq \epsilon \right) \leq \alpha \\ n \geq n_0 \implies \mathbb{P}_{h \sim \mathcal{H}^{(n)}} \left(\left| \hat{\mathcal{R}}_{\mathcal{X},\mathcal{Y}}^{(n)}(h) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_*) \right| \geq \epsilon \right) \leq \alpha \end{cases} \tag{2.15}$$

Therefore, the ERM principle is consistent if the sequence of models $\hat{h}_*^{(n)}$ should have its empirical and expected risks converge to the hypothesis minimizer expected risk. At the beginning of this chapter, we said that computational aspects were reflected in the name of the discipline because of their importance. In the case of the ERM, it translates to having a fast convergence rate.

Definition 2.3.3 (Fast convergence rate)**.** *The rate of convergence of the random sequence of models selected by the ERM principle is said to be fast if (Vapnik, 1998)*

$$\exists c, n_0 > 0 : \forall n \geq n_0 : \mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\left| \mathcal{R}_{\mathcal{X},\mathcal{Y}} \left(\hat{h}_*^{(n)} \right) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_*) \right| \geq \epsilon \right) \leq e^{-c\epsilon^2 n} \tag{2.16}$$

In words, fast convergence appears when, past some point n_0, the probability of the gap between how the selected model and the true minimizer exceeding some threshold ϵ decreases exponentially fast with a linear increase of the number of samples. Thus, any addition of $\log 2/(c\epsilon^2)$ examples halves that probability.

The definition of fast convergence rate can be restated as tightening the bound when n increases for a fixed probability α of exceeding the bound.

Definition 2.3.4 (Fast convergence rate (version 2))**.** *The rate of convergence of the random sequence of models selected by the ERM principle is said to be fast if* $\exists c, n_0 > 0 : \forall n \geq n_0$ *such that*

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\left| \mathcal{R}_{\mathcal{X},\mathcal{Y}} \left(\hat{h}_*^{(n)} \right) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_*) \right| \geq \sqrt{\frac{1}{cn} \log \frac{1}{\alpha}} \right) \leq \alpha \tag{2.17}$$

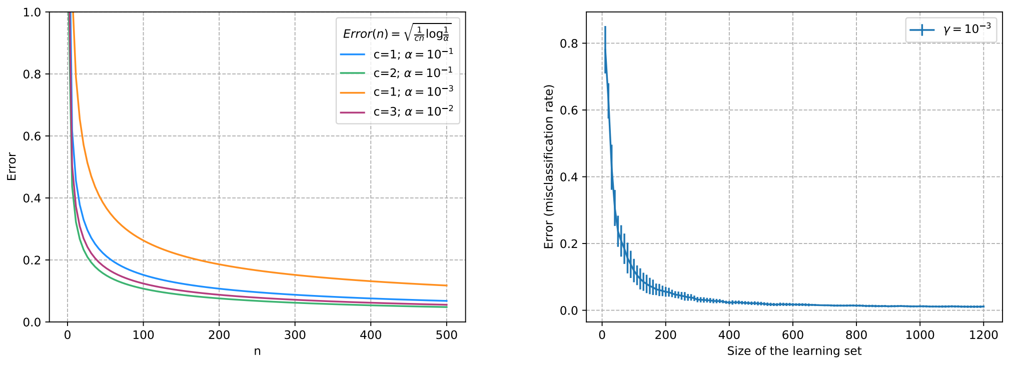
This version is derived from the first version by posing $\alpha = e^{-c\epsilon^2 n}$.

The meaning of the fast convergence rate is illustrated in Figure 2.4a. Most (as defined by α) learning curves should be bounded (at least from n_0 on) by a function of the form $\sqrt{\frac{c'}{cn}}$. Figure 2.4b illustrates the empirical curve for the digit recognition problem (solved by a linear function in the kernel space with increasing numbers of basis vectors). The empirical curve seems to follow nicely the fast convergence rate.

Conditions under which the fast convergence is achieved (and thus the ERM is consistent) will be discussed in Section 2.6.2. At this point, it might not be clear, however, why the ERM is not always consistent. The following sections will delve into this interrogation.

Naive learning algorithm. One straightforward implementation of the ERM principle would be, for each hypothesis, to assess its empirical error and select the best one. This poses two problems, however: infinitude and finitude.

Firstly, this is only possible with a finite hypothesis space, which is usually not the case. Consider the set of straight lines, for instance. There is an



(A) Theoretical learning curve.

(B) Empirical learning curve: test set error for a linear model in a kernel space as the number of basis vectors n increases.

FIGURE 2.4: Comparison between empirical and theoretical learning curves (*i.e.* error as a function of the training set size) illustrating the fast convergence rate.

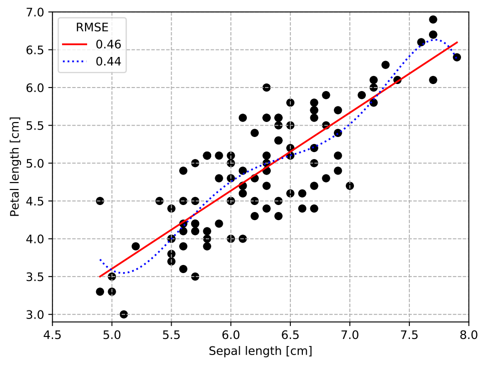
uncountable infinite number of slopes and intercepts. Arguably, close parameters would yield close errors. This might not be so evident for other hypothesis spaces, however. Besides, even finite hypothesis spaces tend to be large. For instance, a complete hypothesis space over boolean functions of p propositions contains 2^{2^p} members. Evaluating how each of them performs on a large-scale problem (large n, large p) remains quite prohibitive. Chapter 3 will present efficient supervised learning algorithms. For now, let us just assume that they exist.

The second problem relates to the finite number of samples. Since our naive algorithm uses the samples to select its hypothesis, it may find one which performs well on its training data but portrays a poor generalization error. This caveat is not specific to this naive algorithm and will be the topic of the next section.

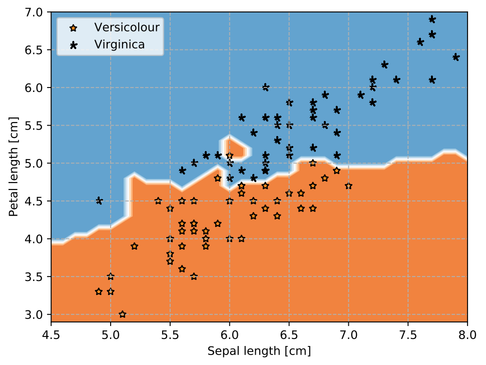
2.4 Overfitting

Consider Figures 2.5a and 2.5b. On the former, the dotted blue line portrays a smaller error (RMSE of 0.44) than the straight-line model (RMSE of 0.46). On the latter figure, there are no more misclassified samples. Despite their better *apparent* performances, these models violate some unspoken assumptions we might have about the problems. For the regression, it seems hard to justify this wave-like behavior. For the classification, we might expect a smoother boundary and certainly no enclave. In these situations, intuition and the ability to visualize the low-dimensional data allow us to detect something is amiss. Is it possible to detect such situations in general? What is actually happening?

The phenomenon at play here is called overfitting and manifests as the model appearing uncharacteristically good on the set it is being evaluated



(A) Possible linear relationships between sepal and petal lengths. Does the dotted blue line feel natural?



(B) Possible boundary between iris species. Colors depict the prediction of the model.

FIGURE 2.5: Solutions to the sepal-to-petal and versicolour-or-virginica problems. Does the boundary feels reasonable?

because it is the one on which the model was chosen; the assessment is biased. As such, E_{LS} is no longer a reliable estimate of $\mathcal{R}_{\mathcal{X},\mathcal{Y}}$. Overfitting is sometimes described as "fitting the *noise* in the data as well as the signal". In this latter idiom, "noise" is used loosely to mean (or incorporate) natural variance in the phenomenon, coupled with finite data.

2.4.1 Unbiased assessments

A simple solution to monitor overfitting is to evaluate the model's performance on samples which have not been used to choose it. Since there is only one pool of samples, these "unseen-during-training" samples must come from this single set. Two common strategies to procure such a set are the train/test split and cross-validation.

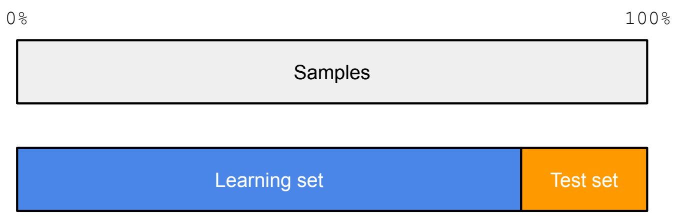
Train/test split. One possible strategy is to divide the set in two parts: an actual training set LS which is fed to the learning algorithm and a held-out set TS set (the test set), on which the performances will be estimated. An illustration of this method is given in Figure 2.6a. As mentioned previously, we usually distinguish between the resubstitution error E_{LS} and the generalization error E_{TS}, which is an unbiased estimator of $\mathcal{R}_{\mathcal{X},\mathcal{Y}}$.

Cross-validation. Another strategy relies on partitioning the dataset S in k subsets S_i (for instance $k = 10$; let us suppose that n is divisible by k for simplicity) so that $S = S_1 \cup \ldots \cup S_k$, $i \neq j \implies S_i \cap S_j = \emptyset$ and $|S_i| = n/k\,(i = 1, \ldots, k)$. Cross-validation consists in

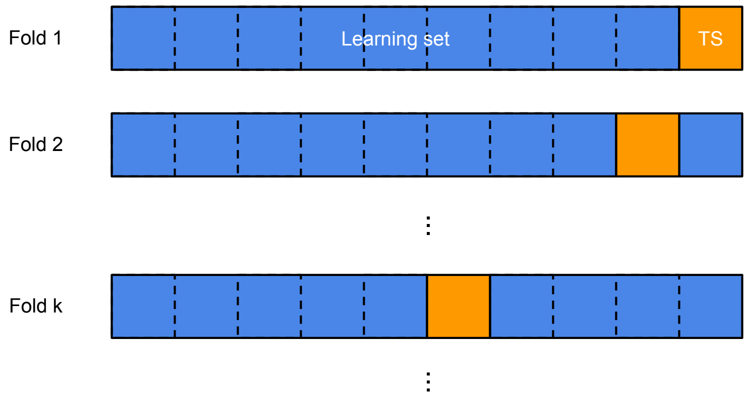
1. For fold $i = 1, \ldots, k$

   (a) use $TS_i = S_{k-i+1}$ as test set and $LS_i = S \setminus S_{k-i+1}$ as learning set;

   (b) learn a model on LS_i;



(A) Illustration of the train/test split method: a subset of data is kept for unbiased estimation.



(B) Illustration of the cross-validation method: the dataset is partitioned into folds. One of the fold is used for unbiased estimation. The process is repeated on each fold and the estimations are aggregated at the end.

FIGURE 2.6: Methods to construct sets for unbiased estimation of model performances.

   (c) compute the error E_i on TS_i;

2. Return the average error $1/k \sum_{i=1}^{k} E_i$.

When $k = n$, cross-validation is known as leave-one-out.

Choosing and parametrizing an assessment method. Both train/test split and cross-validation methods reduce the size of the learning sample to make room for assessment samples. Unfortunately, many samples are required both for learning and evaluating. The former is illustrated by Figure 2.4: removing samples from the learning pool may well result in a suboptimal model. For instance, using only 200 samples on the digit problem yields a much worse error than when using the whole set. On the other hand, assessing model performance on a small set may yield an unreliable estimate because of the high variance due to a small set (law of large number). Thus parametrizing the size of the held-out set is a delicate decision.

The practical guidelines are to use the train/test split method when the amount of data is large (compared to the expressiveness[1] of the hypothesis space, as will be discussed in the following sections). For this approach, it is usually advocated to use between one third to one-fifth of the data as test set, owing to the fact that optimizing the hypothesis is harder and requires more samples than simply evaluating the error.

Corollarily, cross-validation is recommended when data is scarce because one can increase the number of folds and reduce its size to get a less suboptimal model. Cross-validation is a viable option only provided that the time required for the loop is available. Indeed, if the time complexity of the learning algorithm is $O(f(n))$ the cross-validation propels the time complexity to $O(kf(n/k))$. Since f is at least linear in the number of data (we expect the algorithm to look at all the data), increasing k slows down the whole process. A value of $k = 10$ is the standard choice.

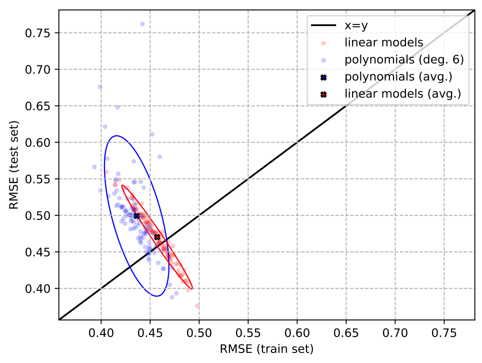
Cross-validation comes with a(nother) technical caveat. It might seem that the average error over the fold is an estimate of the true error (law of total expectation). However, the model learned at each fold is different. Consequently, we are not estimating $\mathcal{R}_{\mathcal{X},\mathcal{Y}}$ (the expected risk of the *model*), but rather $\mathcal{R}_{\mathcal{X},\mathcal{Y}}^{(n(k-1)/k)}$ (the expected risk of the *learning algorithm*). Ironically, $\mathcal{R}_{\mathcal{X},\mathcal{Y}}^{(n(k-1)/k)}$ can serve as a proxy for $\mathcal{R}_{\mathcal{X},\mathcal{Y}}$ only if its variance is low (*i.e.* whatever the selected hypothesis, the performance are roughly equivalent), which, to be true, requires large samples (as can be glimpsed on Figure 2.4 and will be discussed in details in the upcoming sections). Overall, cross-validation should be avoided to assess the *model* performance whenever possible.

Retraining. Since reducing the size of the learning set produces suboptimal models and the learning curve is monotonically decreasing, there is usually no reason, so long as time permits, to refrain from re-learning the model on the whole set and treating the error as a lower bound.

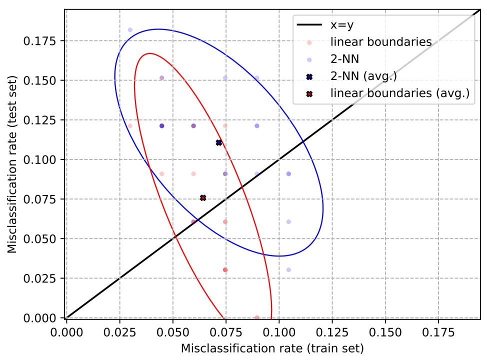
2.4.2 Consequences of overfitting

Figure 2.7 illustrates overfitting on the iris problems, in both regression (Figure 2.7a) and classification (Figure 2.7b). In both cases, the whole dataset has been shuffled to obtain different train/test splits on which models from different hypothesis spaces have been learned and assessed. The scatter plots include how each model fare with respect to the resubtitution error and the generalization error. Two classes of models are included. For regression, they are linear and six-degree-polynomial models. For classification linear boundaries and 2-nearest neighbor (2-NN) are examined. The latter corresponds to attributing the class of the two nearest points. Dispersion ellipses summarizing the point clouds, as well as their centers, are also displayed.

[1]Defining "expressiveness" will be the topic of Section 2.6. Informally, expressiveness conveys the idea of a hypothesis space containing hypotheses suitable for many different problems.



(A) Dispersion ellipses of the error over the training and test sets for the sepal-to-petal problem.



(B) Possible linear boundary between iris species.

FIGURE 2.7: Resubstitution versus generalization errors for several models and hypothesis spaces. The dispersion ellipses summarized the point clouds by hypothesis space by depicting the center of distribution and how the major and minor variance directions.

As we can see, whatever the problem and the hypothesis space, the (empirical) error is lower on the training set on average (*i.e.* centers above the identity line). We also see that the linear models are much closer to the diagonal, indicating lesser overfitting. The explanation for this will be the core of the next section.

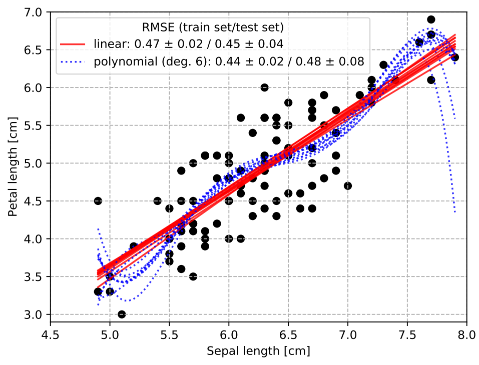
In the regression problem, we also observe that (i) the surface of the ellipsis corresponding to the linear hypothesis space is much smaller, and (ii) its major axis is orthogonal to the identity line. Together, these imply that all models in this class are somewhat equal, trading one type of error for the other one but offering no model which performs better on both accounts.

Overfitting is relatively mild in the examples we have covered. In the extreme case, the learning algorithm could produce a model which does nothing more than memorizing all the information it is given. In such a case, the chosen hypothesis would perform perfectly on the learning set and randomly on any unseen data (or just fail in that case).

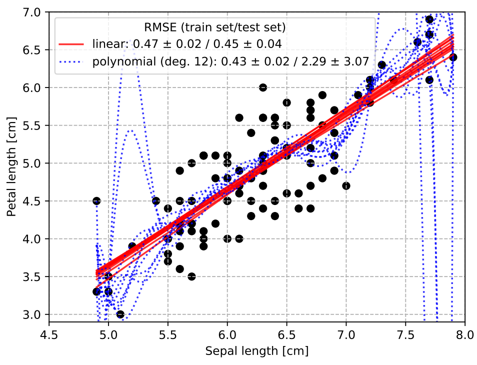
2.4.3 Overfitting and the hypothesis space expressiveness

Overfitting is not so much a property of the model as a phenomenon which relates to the hypothesis space. Admittedly, it might happen that a singleton hypothesis space would produce a model (the only one available) which performs well on the (virtually useless) training set and badly on the test set. Re-drawing the sets would result in different conclusions, however.

On the other hand, a more varied hypothesis space increases the odds of having a model which performs a lot better on the training set. This is what happens in the case of the iris problems. In the regression, the six-degree-polynomials class encompasses the linear hypothesis space, and is much



(A) linear vs. 6-degree polynomial models.



(B) linear vs. 12-degree polynomial models.

FIGURE 2.8: Variability of the prediction due to overfitting for the sepal-to-petal problem.

more expressive (one can choose seven points through which the model passes instead of two). As a result, it is more susceptible to overfitting (ellipsis centers further away from the identity line). On a pointwise basis, the flexibility results in more instability for a prediction as well, as Figure 2.8 illustrates. Increasing the degree of the polynomials in the hypothesis space allows finding a model with a slightly lower training error whatever the points given. On the other hand, the selected models differ much. Since the model class is flexible enough to focus on artifacts due to sampling and natural variance, it does so at the detriment of the actual learning signal, resulting in (much) larger errors on the test set—especially when transitioning from interpolation to extrapolation.

The subsequent sections will formalize the notions which have been intuitively illustrated here.

2.5 The bias-variance decomposition

2.5.1 Decomposition

In the case of regression problems under the square loss, the (pointwise) error decomposes nicely into several insightful terms. Examining this decomposition is the goal of the present section.

Firstly, note that the expected risk of the learning algorithm $\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}$ can be rewritten as

$$\begin{aligned}
\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}} &= \mathbb{E}_{\mathcal{H}^{(n)}}\{\mathbb{E}_{\mathcal{X},\mathcal{Y}}\{\ell(h(x),y)\}\} && (2.18)\\
&= \mathbb{E}_{\mathcal{H}^{(n)}}\{\mathbb{E}_{\mathcal{X}}\{\mathbb{E}_{\mathcal{Y}|x}\{\ell(h(x),y)\}\}\} && (2.19)\\
&= \mathbb{E}_{\mathcal{X}}\{\mathbb{E}_{\mathcal{H}^{(n)}}\{\mathbb{E}_{\mathcal{Y}|x}\{\ell(h(x),y)\}\}\} && (2.20)\\
&= \mathbb{E}_{\mathcal{X}}\{\mathcal{R}^{(n)}_{\mathcal{Y}|x}\} && (2.21)
\end{aligned}$$

using the law of total expectation and linearity. We will investigate how the expected risk of a learning algorithm behaves at a given x. We then have the following three propositions (proofs can be found in standard textbooks, *e.g.* Friedman, Hastie, and Tibshirani, 2001a).

Proposition 2.5.1. *The Bayes model can be rewritten in a pointwise fashion as*

$$h_B(x) = \arg\min_{y' \in \mathbb{Y}} \mathbb{E}_{\mathcal{Y}|x}\{\ell(y', y)\} \tag{2.22}$$

Proposition 2.5.2. *Under the square loss ℓ_2, the Bayes model at x is the conditional expectation of y given x*

$$\mathbb{E}_{\mathcal{Y}|x}\{y\} \tag{2.23}$$

Proposition 2.5.3 (bias-variance decomposition)**.** *Under the square loss ℓ_2*

$$\mathcal{R}^{(n)}_{\mathcal{Y}|x} = \mathbb{E}_{\mathcal{H}^{(n)}}\{\mathbb{E}_{\mathcal{Y}|x}\{(h(x) - y)^2\} \tag{2.24}$$

$$\begin{aligned} = \quad & \mathbb{E}_{\mathcal{Y}|x}\{(y - h_B(x))^2\} && \textit{noise}(x) \\ & + (h_B(x) - \mathbb{E}_{\mathcal{H}^{(n)}}\{h(x)\})^2 && \textit{bias}^2(x) \\ & + \mathbb{E}_{\mathcal{H}^{(n)}}\{(h(x) - \mathbb{E}_{\mathcal{H}^{(n)}}\{h(x)\})^2\} && \textit{variance}(x) \end{aligned} \tag{2.25}$$

In words, the decomposition tells us that, for a regression problem and the ℓ_2 loss, the expected risk of any learning algorithms is the sum of three factors: the noise, the bias and the variance. An illustration of the error components is given in Figure 2.9.

Noise. The noise is an unavoidable residual error caused by the variance of the phenomenon. Remember that in this case $h_B(x) = \mathbb{E}_{\mathcal{Y}|x}\{y\}$, meaning that the noise is the variance of $\mathcal{Y}$ given x:

$$\begin{aligned} \text{noise}(x) &= \mathbb{E}_{\mathcal{Y}|x}\{(y - h_B(x))^2\} && (2.26) \\ &= \mathbb{E}_{\mathcal{Y}|x}\{(y - \mathbb{E}_{\mathcal{Y}|x}\{y\})^2\} = \mathbb{V}_{\mathcal{Y}|x}\{y\} && (2.27) \end{aligned}$$

Since the predictors is a function while the actual phenomenon is stochastic, there is no overcoming this part of the error. On the other hand, if the process is devoid of variance (*i.e.* the phenomenon is totally deterministic), this term vanishes.

Bias. The squared bias reflects how much, on average, the selected model is far from the optimal Bayes model. When there is no bias, the average prediction conforms to the Bayes model prediction. A high bias is the result of some kind of systematic errors in the predictions: they do not cancel each other out on average because all the models make the same mistake.

The most common source of bias is having an inadequate hypothesis space. In the sepal-to-petal problem, a model could constantly predict the average of petal length in the learning set, independently of the sepal length.

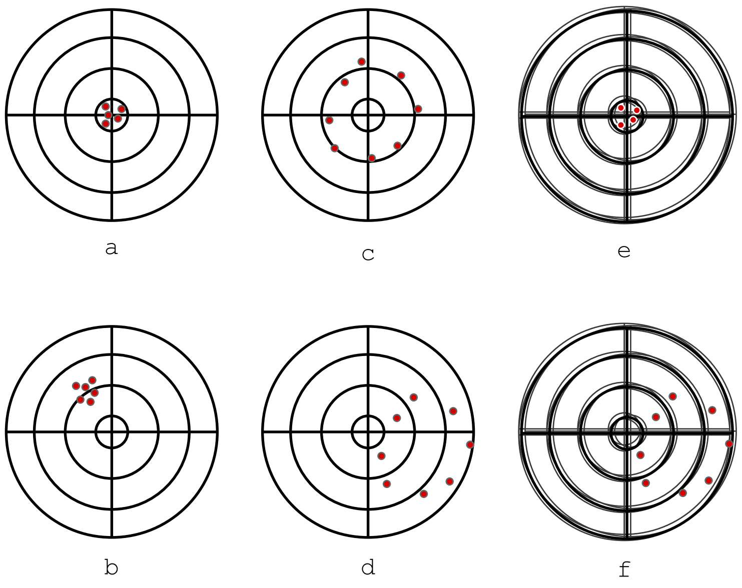


FIGURE 2.9: Bias-variance decomposition—an illustrative perspective. The target center represents the ground truth and each point represents the prediction of a model. (a) represents a low overall error on a noise-free problem: all predictions are close to the ground truth. (b) represents a low variance, high bias situation on a noise-free problem: all predictions are close together but are not centered on the ground truth. (c) represents a low bias, high variance situation on a noise-free problem: the average of the points is well-centered but each point is far away from the ground truth. (d) represents a high bias, high variance situation on a noise-free problem: predictions are not close to each other and are not centered on the ground truth. (e) represents a low bias, low variance situation with residual noise: where the exact target center (*i.e.* the ground-truth) lies is fuzzy. (f) represents a high bias, high variance with residual noise.

Although this is provably the best constant prediction (under the ℓ_2 loss), all such models will underestimate petal lengths of iris with large sepal.

How does this relate to the representational bias discussed in Section 2.2.5? The squared bias can be further decomposed as

$$\begin{aligned}
&\left(h_B(x) - \mathbb{E}_{\mathcal{H}^{(n)}}\{h(x)\}\right)^2 \\
&= \left((h_B(x) - h_*(x)) - \left(\mathbb{E}_{\mathcal{H}^{(n)}}\{h(x)\} - h_*(x)\right)\right)^2 \qquad (2.28)\\
&= (h_B(x) - h_*(x))^2 + \left(\mathbb{E}_{\mathcal{H}^{(n)}}\{h(x)\} - h_*(x)\right)^2 \\
&\qquad - 2\,(h_B(x) - h_*(x))\left(\mathbb{E}_{\mathcal{H}^{(n)}}\{h(x)\} - h_*(x)\right) \qquad (2.29)
\end{aligned}$$

where the first term of Eq. 2.29 is the (squared) representation bias, whereas the second term is called the squared search bias. The former relates solely to the hypothesis space, while the latter expresses how suboptimal the learning algorithm is. Taking the expectation over $\mathcal{X}$ of the last term yields a kind of covariance between both bias types. Unless there are reasons for them to be correlated, this last term is null on average.

Variance. The variance reflects how much the prediction varies depending on the learning set. The major cause of variance is overfitting.

Note that both the noise and this term are variances. The former, $\mathbb{V}_{\mathcal{Y}|x}\{y\}$, is solely due to the phenomenon. The latter is $\mathbb{V}_{\mathcal{H}^{(n)}}\{h(x)\}$ and is due to both the data generating process *and* the learning algorithm.

2.5.2 Bias-variance tradeoff

At first glance, designing a regression algorithm seems straightforward: reduce the bias and the variance. Unfortunately, these terms are not independent; they are inversely affected by the expressiveness of the hypothesis space.

As we have mentioned, overfitting is more likely to happen when the model class is expressive, since there is more flexibility to fit well the learning set. As a consequence, $\mathbb{V}_{\mathcal{H}^{(n)}}\{h(x)\}$ will increase. On the other hand, if the hypothesis space is not expressive enough, the likelihood of having a large bias increases. This is known as the bias-variance tradeoff and is illustrated in Figure 2.10.

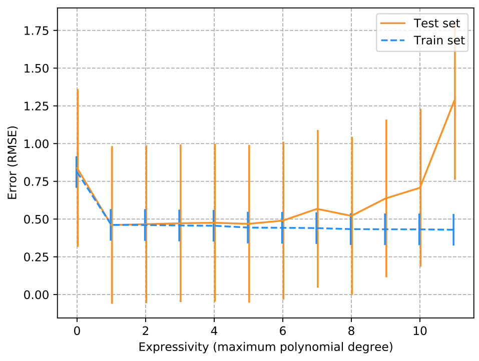
In Figure 2.10a, the training set and test set errors are reported as a function of the maximum degree the polynomials in the hypothesis space can take. When predicting a constant (polynomial of degree 0), both test and train errors are high. This is a case of underfitting caused by having a high (representational) bias. In the linear regime (degree 1), the test error is at its lowest. From there (degrees 2 to 5), the learning algorithm is able to find a marginally better model on training samples but the test error rises slowly. From there on (degree ≥ 6), the gap between the training and test errors widens noticeably: the learning algorithm is clearly overfitting the data.

This is not specific to regression and the iris dataset, Figure 2.10b illustrates the same phenomenon, where there is a clear compromise to find between under- and overfitting. Is also illustrated the fact that the variance on the test set increases with the model complexity.

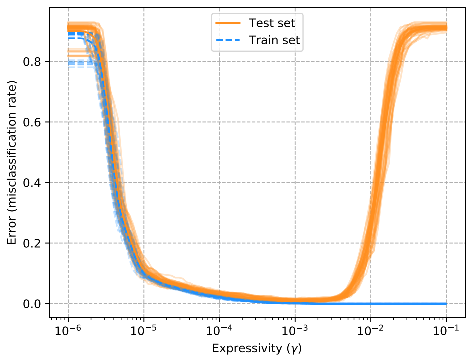
In general, the behavior of the expected risk with respect to the expressiveness of the hypothesis can be schematically described as in Figure 2.10c.

The take-home message is that it is not possible to eliminate both the variance and the bias—at least the representational component. Consequently, a better goal should be to aim for the sweet spot balancing both errors.

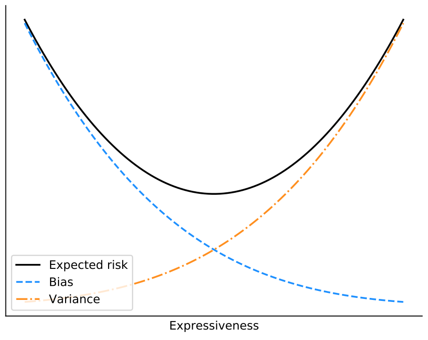
On expressiveness and complexity. It is common to talk about "model complexity", especially when referring to a hypothesis space built upon a parametric class of models. In such a case, the "complexity" (whose precise definition depends on the class of models) controls the expressiveness of the hypothesis space. For instance, in a hypothesis space made of polynomials, the degree of such functions may serve as complexity. Measures of regularity



(A) Sepal-to-petal polynomial regression.



(B) Digit classification with a linear model from the kernel space ($m = 900$). The γ parameter relates to the locality coefficient of the RBF kernel (see Section 2.2.7).



(C) Schematic behavior of bias-variance tradeoff

FIGURE 2.10: Bias-variance tradeoff: errors with respect to model complexity. As the expressiveness increases, the error first decreases (the representation bias is gradually overcome) but then starts increasing again suggesting overfitting (the training set is memorized, resulting in increased variance).

(such as Lipschitz constant) or of memory (such as the raw number of learnable parameters in a model and the minimum description length) may also serve as complexity measures. These are however somewhat indirect measures of expressiveness. More direct measures will be discussed in Section 2.6.2 (in the case of classification).

2.5.3 Approximation-estimation decomposition

The bias-variance decomposition we have seen is tied to regression problems under the ℓ_2 loss. A general decomposition, known as the approximation-estimation, also exists.

Definition 2.5.1 (Approximation-estimation decomposition). *Let h be some hypothesis selected by a learning algorithm. The gap between the risk of h and the* *risk of the Bayes model h_B can be decomposed as (e.g. Bottou and Bousquet, 2007; Luxburg and Schölkopf, 2011)*

$$\mathcal{R}_{\mathcal{X},\mathcal{Y}}(h) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_B) = \underbrace{(\mathcal{R}_{\mathcal{X},\mathcal{Y}}(h) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_*))}_{\textit{estimation error}} + \underbrace{(\mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_*) - \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h_B))}_{\textit{approximation error}} \tag{2.30}$$

Note that the decomposition is valid for all h but referring to the terms as estimation and approximation errors of a learning algorithm only make sense if h is the selected hypothesis.

The decomposition translates the risk of hypothesis h so that it will be null if h is the best model there exists (not necessarily from the hypothesis space). Both terms in the right-hand side are positive (or null). The first one, by definition of h_* (best hypothesis) and the second one by definition of the h_B (best function).

In essence, the approximation error captures the same idea as the representation bias. It is an indicator of the discrepancy between the best hypothesis h_* and the Bayes model h_B. It is solely a property of the hypothesis space (in relationship with the problem but independent of the choice of hypothesis).

The estimation error is indicative of the error made by choosing the hypothesis with a finite sample. Remember from the consistency properties (Section 2.3) that a consistent learning algorithm is such that the estimation error will converge to 0 as the size of the learning set increases.

Overall, the general ideas expressed by the bias-variance decomposition, *i.e.* the fact that the error can be decomposed into a term relating to the hypothesis space (bias) and a term linked to the size of the learning set (variance), apply to all problems.

2.6 Bounds over the generalization gap

The previous section provided useful decompositions to understand overfitting. So far, the link with the expressiveness of the hypothesis space is still intuitive. This section will look at classification to establish a more direct connection with expressiveness. We will tackle this through the lens of bounds on the generalization gap.

Definition 2.6.1 (Generalization gap)**.** *The generalization gap is defined as*

$$\Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) = \mathcal{R}_{\mathcal{X},\mathcal{Y}}(h) - \hat{\mathcal{R}}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \tag{2.31}$$

$$\tag{2.32}$$

and is a random variable (over the n datapoints drawn to compute the empirical risk) representing how much the empirical risk underestimates the true risk.

Definition 2.6.2 (Bound on the generalization gap). *A generalization bound is of the following form.*

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}\left(\exists h \in \mathbb{H} : \Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \epsilon(n, \alpha, \mathbb{H})\right) \leq \alpha \tag{2.33}$$

where ϵ is the tightness of the bounds and typically depends on (i) the number of samples n, (ii) the probabilistic guarantee of not exceeding the bounds, which is controlled by α, and (iii) the hypothesis space itself.

An important note is that such bounds are established irrespective of how a hypothesis might be selected. Rather, they are a kind of worst-case scenario, sweeping agnostically over the whole hypothesis space. The question they address is not, given a "bad" learning set, what can be expected of the selected hypothesis? The question is: given a set of samples (absent of any learning quality) what gap can be expected? As a consequence, the bounds apply to hypothesis space; they are not properties of the learning algorithm. Intuitively, such results are established by taking into account the fact that, as more and more realizations (*i.e.* losses at given points in this case) are summed, the harder for the finite sum to diverge much from the expectation.

From the generic form of the bounds, it is apparent that enforcing more guarantees (*i.e.* reducing α) comes at the price of a looser bound and vice versa. Increasing the number of samples will tighten the bound at a fixed α. The hypothesis space appears in the bound because its expressiveness matters (as we will see, the higher the expressiveness, the looser the bound).

Section 2.6.1 discusses such a bound in the case of a finite hypothesis space, while Section 2.6.2 introduces another measure of expressiveness, together with a bound. Those results hold for a loss such that $0 \leq \ell(y', y) \leq 1$, *e.g.* ℓ_{0-1} (note that the actual restriction is that the loss be bounded. From there, one can always rescale the loss to be in the appropriate range).

2.6.1 Finite hypothesis space bound

Singleton bound. Let us first consider the case of a singleton hypothesis space.

Proposition 2.6.1 (Singleton generalization gap bound). *When $\mathbb{H} = \{h\}$, the generalization can be bounded by*

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}\left(\Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \sqrt{\frac{1}{2n}\log\frac{1}{\alpha}}\right) \leq \alpha \tag{2.34}$$

Proof. The proof is reproduced from Shawe-Taylor (2019). Using Hoeffding's inequality on the risk yields

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}\left(\mathcal{R}_{\mathcal{X},\mathcal{Y}}(h) - \hat{\mathcal{R}}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \epsilon\right) \leq e^{-2n\epsilon^2} \tag{2.35}$$

Posing $\alpha = e^{-2n\epsilon^2}$ and solving for ϵ leads to $\epsilon = \sqrt{\frac{1}{2n}\log\frac{1}{\alpha}}$. Substituting the value of ϵ in the previous equation yields Eq. 2.34. □

The Hoeffding's inequality is only valid for losses such that $0 \leq l(y', y) \leq 1$. Being distribution independent, it is applicable to any problem at the expense of being conservative.

Finite hypothesis space bound.

Proposition 2.6.2 (Finite hypothesis space (FHS) generalization gap bound)**.** *For any hypothesis $h \in \mathbb{H}$ of a finite hypothesis space*

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\exists h \in \mathbb{H} : \Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \sqrt{\frac{1}{n} \log \frac{|\mathbb{H}|}{\alpha}} \right) \leq \alpha \tag{2.36}$$

Proof. The proof is reproduced from Shawe-Taylor (2019). Establishing the FHS bound relies on the sub-additivity of the probability measure and then bounding each term of the sum with Hoeffding's inequality:

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\exists h \in \mathbb{H} : \Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \epsilon \right) = \mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\cup_{h\in\mathbb{H}} \quad \Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \epsilon \right) \tag{2.37}$$

$$\leq \sum_{h\in\mathbb{H}} \mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\Delta\mathcal{R}_{\mathcal{X},\mathcal{Y}}(h; LS) \geq \epsilon \right) \tag{2.38}$$

$$\leq \sum_{h\in\mathbb{H}} e^{-2n\epsilon^2} = |\mathbb{H}| e^{-2n\epsilon^2} \tag{2.39}$$

From there, equating the right-hand side to α and solving for ϵ yields Eq. 2.36. □

Interestingly, the bound is established using sub-additivity, meaning that it can be quite loose. Imagine an unrepresentative set of samples is drawn. It will most likely cause many hypotheses to have a large generalization gap. This co-dependence is not captured by the bound. This also highlights the inadequacy of the cardinality to serve as expressiveness measure: the fact that hypotheses might exhibit similar behaviors is not taken into consideration. For instance, if the hypothesis space is made up of $|\mathbb{H}|$ times the same hypothesis, the bound should collapse to the singleton bound, which is not the case. The next section will propose a new measure of expressiveness to somewhat solve this issue.

2.6.2 Vapnik–Chervonenkis bound

The bounds of the previous section are only defined for a finite hypothesis space. However, most practical spaces are (uncountably) infinite (the space of linear functions, for instance). As a consequence, another measure of expressiveness must be used. Incidentally, this measure will also be better able to capture the intuitive idea of expressiveness. Before introducing this new measure, we need to define several concepts.

Definition 2.6.3 (Error vector)**.** *Given $\mathbb{S} = \{(x_i, y_i)_{i=1}^n\} \subset \mathbb{X} \times \mathbb{Y}$, a hypothesis h, and a loss ℓ the corresponding error vector is*

$$u(h, \mathbb{S}; \ell) = [\ell(h(x_1), y_1), \ldots, \ell(h(x_n), y_n)]^T \tag{2.40}$$

Definition 2.6.4 (Error span)**.** *Given $\mathbb{S} \in (\mathbb{X} \times \mathbb{Y})^n$, a hypothesis space $\mathbb{H}$, and a loss ℓ, the error span of $\mathbb{S}$ in $\mathbb{H}$ under ℓ is the set of all error vectors the hypotheses can produce:*

$$\mathbb{U}(\mathbb{H}, \mathbb{S}; \ell) = \{u(h, \mathbb{S}; \ell) : h \in \mathbb{H}\} \tag{2.41}$$

For the remainder of this section, we will focus on ℓ_{0-1} (and drop it from the notation). The binary nature of this loss will simplify the discussion and is sufficient for the point we are making. The discussion can be extended to other loss functions, however, so long as they are bounded (intuitively, this is done by quantizing the error vectors to a finite set, hence the requirement for a bounded loss).

The cardinality of $\mathbb{U}(\mathbb{H}, \mathbb{S})$ is an interesting quantity. By definition of the error span, $|\mathbb{U}(\mathbb{H}, \mathbb{S})| \leq |\mathbb{H}|$. Just as clearly $|\mathbb{U}(\mathbb{H}, \mathbb{S})| \leq 2^n$, since there are only 2^n binary vectors of size n. If $|\mathbb{U}(\mathbb{H}, \mathbb{S})| \ll |\mathbb{H}|$, the hypothesis space is redundant: many hypotheses share the same error vector. This overcomes the limitation of using $|\mathbb{H}|$ directly as expressiveness measure. However, this new metric depends on $\mathbb{S}$, and, eventually, on the $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$ (provided $\mathbb{S}$ is drawn from it), leading to the following definition.

Definition 2.6.5 (Growth function)**.** *The growth function of $\mathbb{H}$ (under ℓ) is the log of the maximum number of error vectors the hypotheses can produce for any set of n points:*

$$Gr_{\mathbb{H}}(n; \ell) = \log \sup_{\mathbb{S} \in (\mathbb{X} \times \mathbb{Y})^n} |\{\mathbb{U}(\mathbb{H}, \mathbb{S}; \ell)\}| \tag{2.42}$$

The growth function is a distribution-independent version of $|U|$. This offers generality at the expense of specificity: the supremum might only be realized for an unlikely $\mathbb{S}$. The growth function inherits the upper bounds of the error span (barring the logarithm), but the fact that it is established on all $\mathbb{S}$ signifies that if $Gr_{\mathbb{H}}(n) = n \log 2$, whatever the given learning set of size n, there exists at least one hypothesis which will produce a null error vector. As such, this hypothesis may lead to a large generalization gap when it actually does not capture the underlying phenomenon. Therefore we, lastly, define the following quantity.

Definition 2.6.6 (Vapnik–Chervonenkis (VC) dimension)**.** *The VC dimension of $\mathbb{H}$ is the minimum sample size n such that not all error vectors can be produced:*

$$\lfloor \mathbb{H} \rceil = \min_{n \in \mathbb{N}} \{n : Gr_{\mathbb{H}}(n) < n \log 2\} \tag{2.43}$$

when the $Gr_{\mathbb{H}}(n) < n \log 2$ condition is never met, the VC dimension is unbounded (or infinite).

The VC dimension allows the derivation of the following bound.

Proposition 2.6.3 (Vapnik–Chervonenkis (VC) bound)**.** *For any hypothesis space* $\mathbb{H}$ *of finite VC-dimension,*

$$\mathbb{P}^n_{\mathcal{X},\mathcal{Y}} \left(\exists h \in \mathbb{H} : \Delta\mathcal{R}^{(n)}_{\mathcal{X},\mathcal{Y}}(h) \geq \sqrt{\frac{1}{n}\left(\lfloor \mathbb{H} \rceil \log\left(\frac{2n}{\lfloor \mathbb{H} \rceil} + 1 \right) + \log\frac{4}{\alpha} \right)} \right) \leq \alpha \tag{2.44}$$

Proof. See Vapnik (1998). □

Examining the bound, we see that if $\lfloor \mathbb{H} \rceil = n$, the radicant of the square root is greater than $\log 3 \approx 1.1$. Since the loss function is upper bounded by 1, the bound is virtually useless in that case. To be of any use, we need to be in a situation where $\lfloor \mathbb{H} \rceil \ll n$. In such a scenario, the gap may be tightly bounded and the empirical risk *does* reflect the expected risk.

Unbounded VC dimension. The VC dimension is better able to capture the intuitive notion of expressiveness by accounting for hypotheses producing the same error responses (or close responses when quantizing the loss) only once. It is for instance possible to show that the class of linear indicator functions over $\mathbb{X} = \mathbb{R}^p$ has a VC dimension of $d + 1$ although it is uncountably infinite.

Some hypothesis spaces, however, have an unbounded VC dimension (for instance, the hypothesis space of decision trees, which we will look at in Chapter 3), meaning they can always produce the full span of error vectors. For those, the VC bound is not applicable and overfitting looms over the learning algorithm like a sword of Damocles, not even disappearing when enlarging the training set a thousandfold.

One might wonder at the rationale behind using guarantee-free hypothesis spaces. Firstly, note that the absence of the bounds does not *de facto* imply overfitting; rather it becomes (much) more likely. Secondly, unbounded VC dimension might be the price to pay for universal expressiveness.

Definition 2.6.7 (Universal approximator)**.** *The hypothesis space* $\mathbb{H}$ *is a universal approximator of* $\mathbb{G}$ *if for any* $g \in \mathbb{G}$ *and any* $\eta \geq 0$

$$\exists h \in \mathbb{H} : \sup_x |h(x) - g(x)| \leq \eta \tag{2.45}$$

Theorem 2.6.1. *If* $\mathbb{H}$ *is a universal approximator of* $\mathbb{G}$ *then* $\lfloor \mathbb{H} \rceil \geq \lfloor \mathbb{G} \rceil$.

Therefore, if one wants $\mathbb{H}$ to be able to approximate *any* function, it cannot have a finite VC dimension.

Fast convergence of the ERM principle. The theory developed in this section allows us to finally formulate a criterion which ensures the fast convergence rate of the ERM principle (and, consequently its consistency).

Theorem 2.6.2 (Fast convergence rate of the ERM principle)**.** *A necessary and sufficient condition for the ERM to exhibit fast convergence is*

$$\lim_{n\to\infty} \frac{Gr_{\mathbb{H}}(n)}{n} = 0 \tag{2.46}$$

This condition imposes that the VC dimension of the hypothesis space be finite (otherwise, the growth function is always linear in n and the limit is $\log 2$), which makes perfect sense.

2.7 The expressiveness/overfitting dilemma

Both FHS and VC bounds show the same trends. Firstly, as the number of samples increases, the bounds tighten (at least for finite expressiveness). Secondly, as the expressiveness increases, the bounds loosen. Thus, the relative values of the expressiveness and number of samples determine the tightness of the bound.

When the expressiveness is high but the learning set is small, the bounds are loose—even useless. Even though the empirical risk is low, the generalization error may be high; this is a typical overfitting scenario. Note that resorting to a larger training set improves the bounds, at least so long as the expressiveness is bounded. Indeed, there exist hypothesis spaces with unbounded VC dimension. In such a case, the model class is so expressive that overfitting is always a risk. Interestingly, we do not necessarily observe severe overfitting in practice in such situations and the search for a better theory is still a vivid topic (see also Section 8.2.1 for further discussions in the case of deep learning).

Conversely, when the expressiveness is low but the learning set is large, the bounds are tightest. Note that the bound can be tight, yet the risk high. In such a case, both the empirical and expected risks will be high. The bounds are of little help in such a situation and increasing the number of samples does not help. Rather, a change of hypothesis space is needed to find hypotheses with more favorable risks.

Although both the bounds and the bias-variance decomposition shed some light on overfitting, they have different goals and employ different means. As noted, the bounds do not highlight how to reduce the generalization error. On the other hand, the bias-variance decomposition does not explicitly refer to the expressiveness or the size of the learning sample. Nonetheless, taken together, they depict a coherent picture: there is a tradeoff to be found between (i) having an expressive hypothesis space, in which case there is a low representational bias and low empirical risk can be achieved but the high variance opens up the door to overfitting, which is reflected by a loose bound, and (ii) a not so expressive hypothesis space, which might incur a high empirical risk because of representational bias, in which case the resulting tight bound and low variance are not of much use.

Overall, choosing adequately the expressiveness is a key factor to successful supervised learning. Sometimes, one might have enough intuition on the problem to choose a not-so-expressive hypothesis space with low representational bias. This is why we frowned at the idea of modeling the sepal-to-petal relationship with a sixth-degree polynomial. When such intuition is missing, it is still possible to manage the expressiveness, albeit at the expense of recasting the problem somewhat. Indeed, the shortcomings we have discussed do not stem from, but are at least exacerbated by, the ERM principle which increases the likelihood of overfitting (by selecting the best hypothesis on the learning sample). Therefore, we switch to a new principle: structural risk minimization, which is based, as the name suggests, on structuring the hypothesis space.

Definition 2.7.1 (Structure). *A structure over a hypothesis space $\mathbb{H}$ is a sequence $\mathbb{H}_1, \mathbb{H}_2, \ldots$ such that*

$$\begin{cases} \mathbb{H}_1 \subset \mathbb{H}_2 \subset \ldots \subset \mathbb{H} \\ \lfloor \mathbb{H}_1 \rceil \leq \lfloor \mathbb{H}_2 \rceil \leq \ldots \leq \lfloor \mathbb{H} \rceil \end{cases} \tag{2.47}$$

For instance, a structure can be placed over the set of one-dimensional linear functions by forming element of the form $\mathbb{H}_i = \{x \to mx + p | m^2 + p^2 \leq t_i\}$, with $i < j \iff t_i < t_j$.

Definition 2.7.2 (Structural risk minimization (SRM) principle). *The SRM principle state that the hypothesis must be selected by*

1. *choosing an appropriate element $\mathbb{H}_i$ of the structure;*
2. *learning the hypothesis from $\mathbb{H}_i$.*

How the SRM principle is implemented depends, notably, on the nature of the hypothesis space. The following two sections will cover two such implementations: regularization and model selection.

2.8 Managing expressiveness: regularization

2.8.1 Regularization

Regularization amounts to encouraging the selection of a hypothesis from a structure member of low expressiveness.

Penalization. Penalization discourages highly-expressive structure members. It adopts the following form.

Definition 2.8.1 (Penalization). *Given a learning set LS, a penalization-based regularized algorithm learning solves the program*

$$h_* = \arg\min_{h \in \mathbb{H}} E_{LS}(h) + \mu R(h) \tag{2.48}$$

where $\mu \in \mathbb{R}^+$ is a hyper-parameter balancing the goodness of h on LS and the expressiveness penalty given by the regularizer R.

Definition 2.8.2 (Regularizer). *A regularizer $R : \mathbb{H} \to \mathbb{R}^+$ over $\mathbb{H}$ is a function such that $\forall h \in \mathbb{H}_i, h' \in \mathbb{H}_j, R(h) \leq R(h') \iff \lfloor \mathbb{H}_i \rceil \leq \lfloor \mathbb{H}_j \rceil$.*

Penalization works well in tandem with parametric hypothesis spaces where the regularizer is defined directly on the parameters. For instance, on a hypothesis space $\mathbb{H} = \{x \to w^T x\}$ composed of linear functions, one can impose a penalty of the form $||w||_2^2$ (known as L_2 penalty) to favor models with lower slopes. This is how the green dotted line of Figure 2.2a was obtained. We see that the resubstitution error is higher and the slope lower (in that case, using regularization was not necessary).

Penalization closely follows the SRM principle since it can be seen as a Lagrangian over the structure. It comes with the caveat of balancing the hyper-parameter μ. If μ is small, we fall back to a standard ERM. As μ increases, the hypothesis is selected less and less based on the data and more and more based on the expressiveness of its structure member. Choosing the appropriate μ is not straightforward. Fortunately, μ can be treated as a hyper-parameter and is thus tunable by model selection (see Section 2.8.2).

Beyond penalization. Other forms of regularization exist. They are usually more dependent on the learning algorithm, however. In order to fix the idea, imagine a learning procedure which optimizes the model sequentially (*e.g.* using gradient descent), producing a sequence of models $h_1, \ldots, h_m$. An implicit regularization might consist in interrupting the optimization before the mth step.

2.8.2 Model selection

This section covers model selection. Although motivated by the need to control the expressiveness of a hypothesis space, model selection is broader in its applications. Model selection is much like the naive learning algorithm we discussed in Section 2.2, and as such, is subject to the same (computational) limitations. That is why it is not so much used as a direct learning algorithm, but rather as an outer loop (or post-processing step) to select a model from a small, finite collection.

Model selection aims at selecting the best model from a pre-selected few. This situation arises most notably when hyper-parameters are involved. When there is no obvious choice for the hyper-parameter values, the practitioner is facing a family of learning algorithms. Rather than making an arbitrary choice, a better alternative consists in optimizing the hyper-parameters.

Many hyper-parameters relate, one way or another, to the expressiveness of the hypothesis space. The penalty weight μ of a penalization, discussed in the previous section, falls into that category. On the other hand, many hyper-parameters play the role of implicit regularizers. The stopping criterion of a sequential process, as mentioned in the previous section is typically a tunable hyper-parameters. Others will be discussed in the next chapter (learning rate, level of noise, etc.). Alternatively, there are some hyper-parameters which directly control the expressiveness, such as the depth of a decision tree, or the architecture of a neural network.

To select a hypothesis from a pool of learned models, its generalization error must be estimated. The learning set cannot be used for this purpose, since it is likely that a model learned from a more expressive hypothesis space will be better, thus rendering the whole expressiveness calibration redundant. With the train/test technique, it would seem that the test set could be used for this purpose. This raises a new (and subtle) problem though: overfitting the test set. It might feel odd that the test error, an estimate of the generalization error, would no longer be a reliable estimate of the same quantity after serving as selection ground. The problem comes from repeating the assessment, which increases the likelihood of obtaining an unreliable estimate. To better grasp what is happening, let us imagine the following thought experiment.

Alice invites m friends over and tasks them to foretell the results of the n independent and unbiased coin flips she is about to make. Since none of them are seers, they will guess at the results. Because there are 2^n possible outcomes but only one correct, the probability of any of the friends guessing the correct tosses is $1/2^n$. Assuming no collusion, the probability of having at least one perfect guesser is

$$1 - \left(1 - \frac{1}{2^n}\right)^m \tag{2.49}$$

If Alice makes 3 tosses and has 10 friends, there is just below 3/4 chances of getting at least a correct guesser (which is much more than 1/8 chance they stood individually). With 10 tosses, the group is still ten times better off than individuals, although it would take 710 guests to have a 50% chance of at least guessing once correctly. This is a lot of friends, but the important thing to note is that, as m grows, so does the probability of finding a lucky guesser (a guesseer).

In the analogy, the friends embody the hypotheses (for a binary classification problem) and the tosses represent the data. Even though the friends simply guess (*i.e.* hypotheses are bad), the probability of finding a seemingly good one on the toss (*i.e.* the test set) is not null. If we only relied on the tosses, we would be fooled into thinking one of them is actually good; the assessment is biased and the test set is overfitted.

In practice, the pool of pre-selected models is supposed to be learned on the data and should do better than random guessing. They would not be as independent as the metaphor suggests and the test set estimate should be more reliable than in the analogy. Moreover, we expect the number of hypotheses (m) to be small (for computational reasons) and the number of datapoints (n) large (for representativeness reasons). Nonetheless, the probability we derived relates to an extreme case where the hypotheses are all wrong. In practice, a much less extreme situation might prove to be problematic. The bottom line is that if a truly unbiased and totally reliable estimate of the error is needed, it should come from a wholly unseen set of samples. Since we are facing the same problem as in Section 2.4.1, we can use the same solutions: leaving a subset for this purpose or using a cross-validation loop.

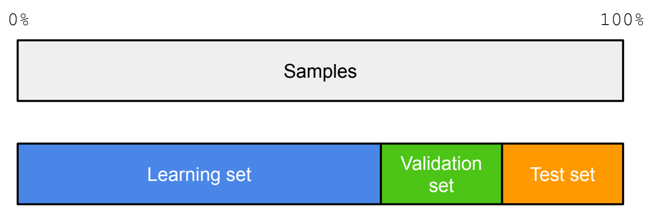


FIGURE 2.11: Illustration of the train/validation/test split.

Train/validation/test split. This technique consists in splitting the samples in three groups whose roles are

1. train set: learn the pre-selected models;
2. validation set: select the best model among the pre-selected ones;
3. test set: assess the final error.

This process is illustrated by Figure 2.11. As with the train/test splits, it is advantageous to retrain the model. It can be done both after selection (once the optimal hyper-parameters have been found) and after the final assessment.

Cross-validation. As far as model selection is concerned, cross-validation is implemented as a two-stage k-fold cross-validation. The outer loop sets apart subsets to estimate the average errors relating to the hyper-parameters chosen by the inner loop.

2.9 Beyond supervised learning

So far, this chapter has mainly been dedicated to supervised learning, which is only one (admittedly major) brick of machine learning. We will now briefly discuss a few other such tools.

2.9.1 Same goal, different means

Supervised learning aims at learning a hypothesis $h \in \mathbb{H}$ with a learning set $LS = \{(x_i, y_i) \in (\mathbb{X} \times \mathbb{Y}) | 1 \leq i \leq n\}$. Sometimes additional sources of data are available to learn from. In the later chapters, we will often tackle settings in which data is scarce. Therefore, leveraging information from other sources will be a major component in working out a solution. Below we briefly describe two settings which will be the basis for our solutions (the relevant facets will be discussed in more depth when describing our solutions).

Semi-supervised learning. In semi-supervised learning, in addition to a (small) learning sample, unlabeled data are available to learn from. This (large) unlabeled set $\mathbb{U}$ is drawn from $\mathbb{P}^m_{\mathcal{X}}$. See the work of Chapelle, Scholkopf, and Zien (2009) for an in-depth tour of semi-supervised learning.

Unlabeled data will be used in Chapter 7 to aggregate indicators in the context of out-of-distribution and in Chapter 8 to transfer the knowledge from a model to another.

Transfer learning. In transfer learning, the goal is to leverage knowledge learned on a *source* task to help in a *target* task. The source task is modeled by some distribution $\mathbb{P}_{\mathcal{X}_s,\mathcal{Y}_s}$ over some $\mathbb{X}_s \times \mathbb{Y}_s$ space, and the target task is modeled by $\mathbb{P}_{\mathcal{X}_t,\mathcal{Y}_t} \neq \mathbb{P}_{\mathcal{X}_s,\mathcal{Y}_s}$ over some $\mathbb{X}_t \times \mathbb{Y}_t$. Depending on how those differ, transfer learning can be further categorized.

For instance, in covariate shift, the task in unchanged except for the distributions over the input space: $\mathbb{X}_s \times \mathbb{Y}_s = \mathbb{X}_t \times \mathbb{Y}_t$ and $\mathbb{P}_{\mathcal{Y}_s|\mathcal{X}} = \mathbb{P}_{\mathcal{Y}_t|\mathcal{X}}$ but $\mathbb{P}_{\mathcal{X}_s} \neq \mathbb{P}_{\mathcal{X}_t}$. Covariate shift happens, for instance, when the lighting conditions change for camera acquisitions (see also Section 7.2.1.2).

Conversely, when only $\mathbb{P}_{\mathcal{Y}_s} \neq \mathbb{P}_{\mathcal{Y}_t}$, the problem is denoted as prior probability shift. Imagine for instance that $\mathbb{Y} = \{healthy, sick\}$ in the context of some disease. If an outbreak occurs, the prior probabilities will shift: without knowing anything about a person, it is more likely sick than before.

Those two examples are cases of domain adaptation (DA) because $\mathbb{X}_s \times \mathbb{Y}_s = \mathbb{X}_t \times \mathbb{Y}_t$. In DA, one usually wants to build a model of $\mathbb{P}_{\mathcal{Y}_t|\mathcal{X}}$ but has access to data from $\mathbb{P}_{\mathcal{X}_s,\mathcal{Y}_s}$ (and possibly some unlabeled data from the target task). Transfer learning also encompasses problems where data are more varied.

Chapter 8 draws ideas from domain adaptation. Zhuang et al. (2021) propose a recent review of transfer learning.

2.9.2 Different goals

Unsupervised learning. Similar to density estimation, in unsupervised learning a finite sample $x_i|x \in \mathbb{X}^n_{i=1}$ drawn iid from a distribution $\mathbb{P}_{\mathcal{X}}$ is available. In unsupervised learning, the goal is to find and exploit structure in the data. Two examples are

Dimensionality reduction In dimensionality reduction, the goal is to find a compression function $h : \mathbb{X} = \mathbb{R}^p \to \mathbb{R}^q$ with $q \ll p$ such that "information is preserved". Dimensionality reduction is performed by techniques such as principal component analysis (PCA), independent component analysis (ICA), t-SNE, etc.

Clustering In clustering the goal is to group datapoints together according to their closeness (rather than in actual classes). Clustering in the realm of algorithms such as hierarchical clustering and k-Means.

Density estimation. In density estimation, the goal is to infer a density distribution function from a finite sample $\{x_i|x \in \mathbb{X}\}^n_{i=1}$ drawn iid from a distribution $\mathbb{P}_{\mathcal{X}}$. Contrary to regression or classification, there is no $\mathbb{Y}$ space involved. Aside from this major difference, density estimation resembles regression on many accounts.

Other tools. There are still other tools (reinforcement learning being an important one, for instance) in the machine learning toolbox. Those are outside the scope of this thesis however and will not be detailed.

2.10 Conclusion

This chapter has been dedicated to a tour of supervised learning. After formalizing it and proposing a first framework with the empirical risk minimization (ERM) principle, we have detailed the main difficulty of supervised learning: overfitting, that is modeling even the natural, sampling randomness embedded in the data.

Starting from an intuitive perspective, we have moved on to two theoretical frameworks which shed some light on the phenomenon. The bias-variance decomposition shows how to decompose the expected risk of a regression algorithm under the squared loss function. This was our first glimpse at the dilemma regarding overfitting and expressiveness. It became apparent that in order to have a low (representational) bias, we needed an expressive hypothesis space. In turn, a highly expressive hypothesis space was likely to raise the variance of the learning algorithm. Although not directly applicable to classification, the phenomenon highlighted by the decomposition also manifests.

To get an even stronger grip on overfitting, we reviewed some results of the statistical learning theory, applicable for bounded loss functions (such as the zero-one loss). These results come in the form of bounds on the generalization gap (the difference between the empirical risk and the actual risk). These are distribution-free, worst-case results. As such the tightness of the bound relates somehow to the variance of the learning algorithm. Establishing the bounds also allows us to (i) define more precisely (or to give one definition of) the hypothesis space's expressiveness and (ii) validate the ERM principle. Although the notion of bias does not appear in the bounds, all the results complement each other and show that overfitting is inextricable under blind application of the ERM principle.

We, therefore, proposed a new principle—structural risk minimization—whose goal is to control the expressiveness of the hypothesis space, and saw how to implement it in practice, either via regularization, or via model selection (as a post-processing step or an outer-loop over a nested learning algorithm). Finally, we discussed a few other paradigms machine learning encompasses.

So far, we have supposed that learning algorithms, capable of enforcing the ERM principle (or its variants), existed. The next chapter will be dedicated to such algorithms and how well they are equipped to deal with this task.

Chapter 3 Learning algorithms for supervised machine learning

Chapter overview

In this chapter, we take a closer look at a few supervised learning algorithms related to our contributions.

Specifically, Section 3.1 will detail the linear regression algorithms which were illustrated in Chapter 2. Section 3.2 will look at a classification counterpart. We will then delve into the realm of decision trees (Section 3.3) and forests (Section 3.4). Our tour of learning algorithms will end with deep learning in Section 3.6. The remainder of the chapter will be dedicated to some tricks and tools relating to learning algorithms (Section 3.7).

As was introduced in the previous chapter, a learning algorithm is a procedure which turns a learning samples $LS = \{(x_i, y_i) \in \mathbb{X} \times \mathbb{Y} | i = 1, \ldots, n\}$ drawn independently and identically (iid) from some unknown data distribution $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$ into a model $h \in \mathbb{H}$ with the goal of minimizing, over that distribution, the expected risk in the sense of some loss function ℓ.

Many learning algorithms assume that $\mathbb{X} = \mathbb{R}^p$, in which case any sample, and in particular the learning sample, may be represented by a $n \times p$ matrix, denoted by X, containing the x vectors (stacked to form the n rows of the matrix so that $X[i, j]$ is the jth component of the ith datapoint), and a column vector y of size n containing the corresponding labels. In such a representation, X is referred to as the learning matrix.

For instance, in the sepal-to-petal problem, the sepal information forms a $n \times 1$ matrix. In the species classification, the matrix is $n \times 2$ (sepal and petal length). In digit classification, the most direct representation consists in flattening the images into vectors of size 64 and thus forming a $n \times 64$ learning matrix.

Several algorithms we will discuss in this chapter expect this form. Section 3.7 will elaborate on how to adapt data which does not fall exactly in this setting (notably the case of discrete input parameters). Consequently, this chapter will tacitly expect this form of inputs.

3.1 Linear regression and its extensions

In linear regression, the hypothesis space is the set of linear (actually affine) functions.

Definition 3.1.1 (Linear hypothesis space (regression))**.**

$$\mathbb{H} = h : x \to w^T x + b \qquad \textit{with } w \in \mathbb{R}^p, b \in \mathbb{R} \tag{3.1}$$

3.1.1 Ordinary least square linear regression

Definition 3.1.2 (Ordinary least square (OLS))**.** *The ordinary least square linear regression is the following program:*

$$\begin{aligned} w_{OLS}, b_{OLS} &= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} \left(w^T x_i + b - y_i\right)^2 && (3.2)\\ &= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} w^{(j)} x_i^{(j)} + b - y_i\right)^2 && (3.3)\\ &= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} ||Xw + \mathbf{1}_n b - y||_2^2 && (3.4)\end{aligned}$$

where $\mathbf{1}_n$ is the vector containing n ones.

OLS amounts to the empirical risk minimization (ERM) under the least square ℓ_2 loss function. The matrix form encourages to consider the relationship between n and p.

The limit case. When $n = p$, X is a square matrix, w_{OLS} can be found by solving the linear system of equations

$$\begin{aligned} Xw_{\text{OLS}} + \mathbf{1}_n b = y && (3.5)\\ \iff w_{\text{OLS}} = X^{-1}(y - \mathbf{1}_n b) && (3.6)\end{aligned}$$

which is only feasible if X is full-rank. The solution thus computed would result in a zero resubstitution error. It should also be clear that, in this case, the selection of the hyperplane is quite susceptible to the drawing of the learning set (*i.e.* the learning algorithm will portray high variance).

The common case. Usually, we expect to have more datapoints to learn from than input features: $n > p$. This is also the situation advocated by the learning theory (since p relates to the VC dimension). From the linear system perspective, it means there are more constraints than unknowns. Unless the matrix contains redundancy, not all constraints can be met simultaneously, and finding a middle ground makes sense. This is the setting we will examine when solving the linear regression in Section 3.1.1.1.

The troublesome case. Also known as the small n, large p problem, the case where $n < p$ portrays fewer constraints than unknowns. Consequently, there is more than one solution to match all the constraints and get a null re-substitution error; the risk of overfitting is high. One way to restore the uniqueness of the solution is to solve a constrained (*i.e.* regularized) alternative. This will be the topic of Section 3.1.2.

Rank and redundancy. Even in situations where $n \geq p$, we might end up in the troublesome case if the matrix contains redundancy (*i.e.* the rank of X is lower than p). This is more than a purely theoretical issue as it might well happen that different features relate to a same physical quantity in a linear fashion without being obvious. Even if the relationship is not perfectly linear, for instance, due to some noise, working with such matrices is known to be numerically unstable. Removing the redundant features, when possible, is probably the best option. Turning to a regularized variant of the least square is another alternative.

3.1.1.1 Solving the ordinary least square regression

In this section, we will assume that $n \geq p$ and X is of rank at least p. Let us denote by F the objective function:

$$F = ||Xw + \mathbf{1}_n b - y||^2 \tag{3.7}$$

Furthermore, we will say that the learning matrix is centered if $\sum_{i=1}^{n} x_i = 0$.

Proposition 3.1.1 (Ordinary least square solution (centered case))**.** *The OLS solution for a centered X is given by*

$$\begin{cases} b_{OLS} = \bar{y} \\ w_{OLS} = \hat{\Sigma}^{-1} X^T (y - \mathbf{1}_n \bar{y}) \end{cases} \tag{3.8}$$

where

$$\bar{y} \triangleq \frac{1}{n} \sum_{i=1}^{n} y_i \tag{3.9}$$

$$\hat{\Sigma} \triangleq X^T X \tag{3.10}$$

Proof. The first-order necessary condition for optimality states

$$\begin{cases} \frac{\partial F}{\partial b} = 0 \iff b = \bar{y} - w^T \bar{x} \\ \nabla_w F = 0 \iff w = \left(X^T X\right)^{-1} X^T (y - \mathbf{1}_n b) \end{cases} \tag{3.11}$$

where $\bar{x} \triangleq \frac{1}{n} \sum_{i=1}^{n} x_i$ and $\nabla_w F$ is the gradient of F with respect to w. The right-most conditions are obtained by differentiating F and then solving for the appropriate variable.

Since X is centered, $\bar{x} = 0$, forcing $b = \bar{y}$. Note that F being convex, this is also a sufficient condition. □

The quantities involved in the OLS solution are insightful and deserve a close inspection.

The intercept. The intercept $b = \bar{y}$ is simply the best (in the sense of ℓ_2) constant which can approximate the output. The role of w is to account for the deviation from the mean.

The covariance matrix. $\hat{\Sigma} = \left(X^T X\right)$ corresponds to the (empirical) covariance matrix:

$$\Sigma = \begin{bmatrix} \hat{\sigma}^2_{x^{(1)}} & \hat{\sigma}_{x^{(1)},x^{(2)}} & \cdots & \hat{\sigma}_{x^{(1)},x^{(p)}} \\ \hat{\sigma}_{x^{(1)},x^{(2)}} & \hat{\sigma}^2_{x^{(2)}} & \cdots & \hat{\sigma}_{x^{(2)},x^{(p)}} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{\sigma}_{x^{(1)},x^{(p)}} & \hat{\sigma}_{x^{(2)},x^{(p)}} & \cdots & \hat{\sigma}^2_{x^{(p)}} \end{bmatrix} \tag{3.12}$$

$\hat{\sigma}^2_{x^{(j)}}$ is the empirical variance of feature j and $\hat{\sigma}_{x^{(j)},x^{(k)}}$ is the empirical covariance between feature j and k.

The matrix is positive semi-definite (positive definite if X has rank p). In the case where $n = p$ and X has full-rank, X is a square, invertible matrix and we fall back to Eq. 3.6.

If the input variables are independent, $\hat{\Sigma}$ is a diagonal matrix and w can easily be computed dimension-wise

$$w^{(j)} = \frac{\hat{\sigma}_{x^{(j)},y}}{\hat{\sigma}^2_{x^{(j)}}} = \hat{\rho}_{x^{(j)},y}\, \hat{\sigma}^2_y \tag{3.13}$$

where $\hat{\rho}_{x^{(j)},y}$ is the empirical correlation between the jth feature and the output. Intuitively, the slope in the jth dimension depends on how the jth feature and the output behaves with respect to one another, as well as how y naturally varies.

This view offers an alternative way of computing the solution by first orthogonalizing X (such as with the Gram-Schmidt algorithm) and then computing w component-wise over this new space.

Non-centered case. The OLS solution derived is valid for the case when X is centered. When this is not the case, a degree of freedom remains and the optimal solution is not unique. Nonetheless, one can always center the inputs as a pre-processing step. Owing to the linearity of the problem, this does not bear any consequences to the quality of the solution.

Computational considerations. Note that the analytical solution we have developed might not be the one implemented for computational reasons (time complexity, numerical stability, etc).

3.1.2 Regularized least-square linear regression

Regularized variants of the least square linear regression—also known as shrinkage methods—offer to restore uniqueness of the solution compared to ordinary least square. They also implement the structural risk minimization principle and might therefore prevent overfitting, especially in cases where n is small and p is large. We will look at three such methods: the ridge regression (Section 3.1.2.1), the lasso and the elastic net (both the latter in Section 3.1.2.2).

3.1.2.1 Ridge regression

Definition 3.1.3 (Ridge regression)**.** *The ridge regression (Hoerl and Kennard, 1970) amounts to solving the least square regression while applying a L_2 penalty on the weights:*

$$\begin{aligned} w_{ridge}, b_{ridge} &= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} \left(w^T x_i + b - y_i\right)^2 + \mu \sum_{j=1}^{p} \left(w^{(j)}\right)^2 && (3.14) \\ &= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} \; ||Xw + \mathbf{1}_n b - y||^2 + \mu||w||^2 && (3.15) \end{aligned}$$

This definition can be viewed as the Lagragian of a constrained OLS problem:

Definition 3.1.4 (Ridge regression (constrained version))**.**

$$\begin{aligned} \min_{(w,b)\in\mathbb{R}^{p+1}} \quad & ||Xw + \mathbf{1}_n b - y||^2 \\ \textit{subject to} \quad & ||w||^2 \leq t_\mu \end{aligned} \qquad (3.16)$$

where the threshold t_μ decreases as μ increases.

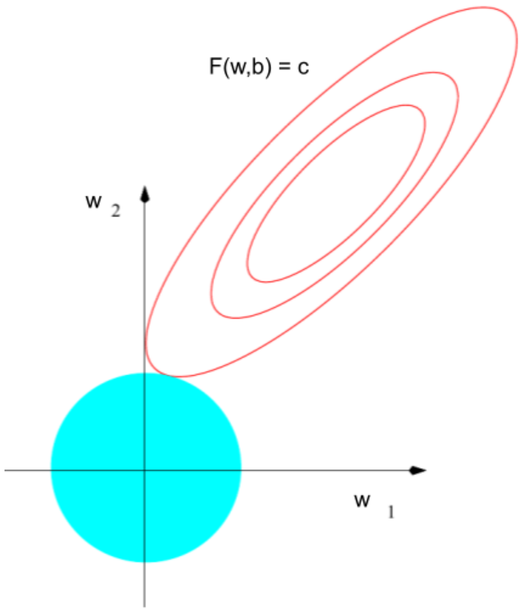
Using the first-order optimality condition, as in the OLS case, the following analytical solution can be derived.

Proposition 3.1.2 (Ridge solution)**.** *The solution to the ridge regression (for a centered learning matrix) is given by*

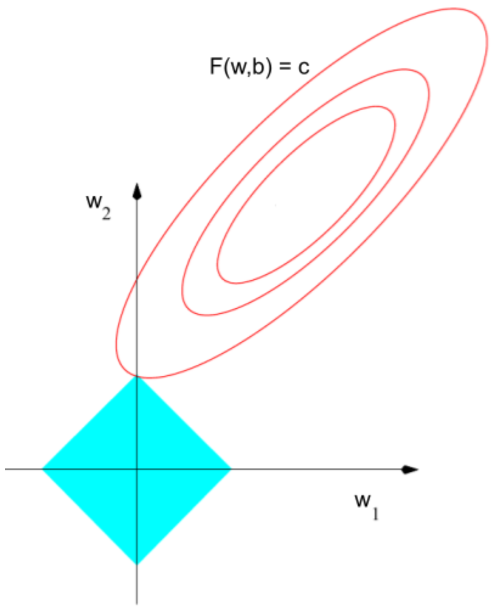
$$\begin{cases} b_{ridge} & = \bar{y} \\ w_{ridge} & = \left(\hat{\Sigma} + \mu I_p\right)^{-1} X^T(y - \mathbf{1}_n\bar{y}) \end{cases} \qquad (3.17)$$

where I_p is the p-dimensional identity matrix. Note that when $\mu = 0$ this amounts to the OLS solutions.

A geometrical interpretation of the ridge regression is given in Figure 3.1a. In the absence of a single best solution for the OLS, the penalty will force the selection of the w vector lying at the intersection of the feasible set and the innermost contour. Figure 3.2a illustrates how the weights vary as μ changes on a selected problem.



(A) Ridge regression. Even though multiple solutions may exist for the OLS, the ridge solution corresponds to the intersection of the ellipsis relating to the lowest contour line and the feasible set.



(B) Lasso. The solution usually involves a sparse weight vector because the most likely intersection point between the contour line of lowest value and the feasible set is at a "corner".

FIGURE 3.1: Geometrical interpretation of the regularized least square in a 2D space (adapted from Friedman, Hastie, and Tibshirani (2001a)). The light blue surface corresponds to the feasible set of weight vectors. The red ellipses correspond to contour lines of the objective function.

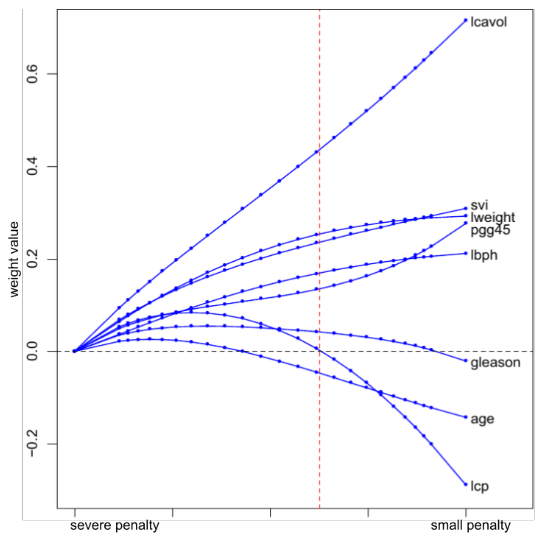
3.1.2.2 Lasso

Definition 3.1.5 (Lasso)**.** *The lasso regression (Tibshirani, 1996) imposes a L_1 penalty of the weights:*

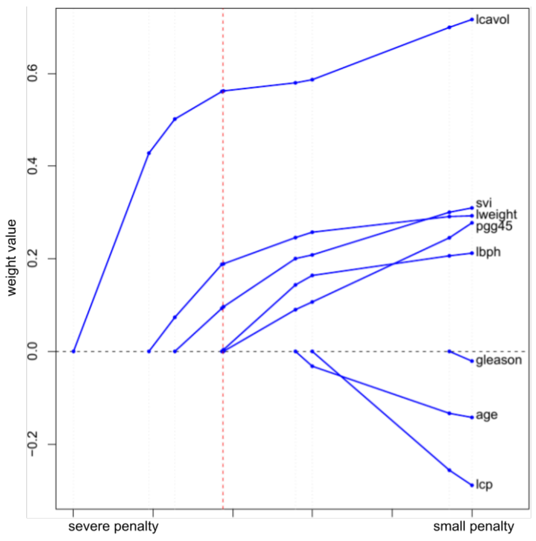
$$w_{lasso}, b_{lasso} = \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} \left(w^T x_i + b - y_i\right)^2 + \mu \sum_{j=1}^{p} \left|w^{(j)}\right| \tag{3.18}$$

Imposing a L_1 penalty leads to an interesting property of the solution w_{lasso}: sparsity. This is illustrated by Figure 3.2b. Varying the severity of the penalty changes the values of weights (as with the ridge regression) but also how many of them are non-zero. Relying on a subspace effectively decreases the VC dimension of the hypothesis space, which is a faithful implementation of the structural risk minimization principle. Sparsity also offers other advantages: faster computation (although the increase is mild in this case, unless many input features are dropped), less costly data acquisition (as some variables need not be monitored), etc.

A geometrical interpretation for how sparsity is encouraged by a L_1 penalty



(A) Ridge regression. As the penalty decreases, the magnitude of the weight vector increases. Note that this does not translate to an increase for all the components and some even change sign.



(B) Lasso. As the penalty decreases, more and more variables are added to the final model.

FIGURE 3.2: Relationship between the selected weight vector w and the penalty parameter μ. The figures are adapted from Friedman, Hastie, and Tibshirani (2001a) and the problem relates to diagnosing severe prostate cancer according to some clinical measures, whose weights are plotted. See Friedman, Hastie, and Tibshirani (2001a) for more details. The red dotted line corresponds to the optimal penalty as found by cross-validation.

is given in Figure 3.1b. In general, the contour lines of the objective function are more likely to hit the feasible set at a point of sparsity (*i.e.* on one axis).

The lasso regression does not offer an analytical solution but can be solved numerically by a quadratic program solver.

Elastic net. It is possible to combine a L_1 and L_2 penalty. This is known as the elastic net (Zou and Hastie, 2005).

Definition 3.1.6 (Elastic net).

$$w_{el.net}, b_{el.net} = \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} \left(w^T x_i + b - y_i\right)^2 + \mu \sum_{j=1}^{p} \left((1-\alpha)\left|w^{(j)}\right| + \alpha \left(w^{(j)}\right)^2\right) \tag{3.19}$$

In the elastic net, α weighs the penalties. A low α implies that the penalty is mostly L_1. This is usually the case recommended to enforce sparsity. Having a small L_2 penalty is useful for dealing with redundant variables. Indeed, in the case of duplicate variables, the contour of the objective function will run parallel to the L_1 feasible set and therefore nullify the sparsity effect.

3.2 Logistic regression

Despite what the name suggests, logistic regression is actually a classification algorithm. What is meant by linear in the context of classification is that the decision boundary is a hyperplane. Therefore, the hypothesis space for a binary problem is composed of functions of the following form.

Definition 3.2.1 (Linear hypothesis space (binary classification))**.** *Encoding one class as* 1 *and the other as* 0*, the hypothesis space for linear binary classification is*

$$\mathbb{H} = \{\text{sign}(w^T x + b) | (w, b) \in \mathbb{R}^{p+1}\} \tag{3.20}$$

with

$$\text{sign}(z) = \begin{cases} 1, & \textit{if } z \geq 0 \\ 0, & \textit{otherwise} \end{cases} \tag{3.21}$$

As mentioned in Chapter 2, working directly with a discrete output space (reflected by the sign function) is not convenient. Linear methods differ on the intermediate output space representation they adopt and, consequently, on the actual problem they solve to end up building a hypothesis of the appropriate form. Logistic regression operates by relaxing the discrete nature of the sign function. Section 3.2.1 discusses the binary case in some length and Section 3.2.2 extends logistic regression to the multi-class setting.

3.2.1 Binary logistic regression

Label encoding: binary logistic regression

For the binary logistic regression, the labels are assumed to be encoded so that $y \in \{0, 1\}$.

3.2.1.1 Logistic regression viewed as relaxation

Binary logistic regression uses the family of logistic functions as a smooth alternative to the sign function.

Definition 3.2.2 (Logistic functions)**.** *The β (≥ 0) logistic function $s_\beta : \mathbb{R} \to [0, 1]$ is the function*

$$s_\beta(z) = \frac{1}{1 + e^{-\beta z}} \tag{3.22}$$

where β is a parameter controlling the slope. In particular, the slope at the origin is $\frac{1}{4}\beta$. When $\beta = 0$, the function is constant. As $\beta \to \infty$, the logistic function tends to the sign *function.*

When $\beta = 1$, the function is known as the sigmoid function.

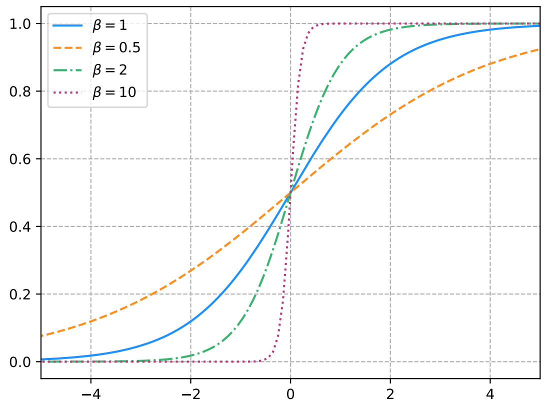


FIGURE 3.3: Examples of the logistic family of functions

The logistic family is illustrated in Figure 3.3. Since the slope of the logistic curve does not influence the final decision, we will default to using the sigmoid ($\beta = 1$).

The logistic functions have some remarkable properties among which are the following three:

$$s_\beta(z) = \frac{1}{1+e^{-\beta z}} = \frac{e^{\beta z}}{1+e^{\beta z}} \tag{3.23}$$

$$s_\beta(1-z) = 1 - s_\beta(z) \tag{3.24}$$

$$\frac{s_\beta(z)}{1-s_\beta(z)} = e^{\beta z} \tag{3.25}$$

Going back to the original problem, we can now describe the relevant hypothesis space.

Definition 3.2.3 (Hypothesis space for logistic regression (binary case))**.**

$$\mathbb{H} = \{\hat{p}(x; w, b) = s(w^T x + b) | (w, b) \in \mathbb{R}^{p+1}\} \tag{3.26}$$

Using the logistic functions, the final decision is such that

$$\begin{cases} 1, & \text{if } \hat{p}(x; w, b) \geq 0.5 \iff w^T x + b \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.27}$$

3.2.1.2 Logistic regression viewed as modeling probabilities

The choice of logistic functions has been motivated from the perspective of a relaxation of the sign function. From the forms of the logistic family, it also appears as a good candidate to model a probability distribution of the form $\mathbb{P}_{\mathcal{Y}|\mathcal{X}}$. From the law of total probability, the following derivation makes the connection apparent.

$$\begin{aligned}
\mathbb{P}(\mathcal{Y}=1|x) &= \frac{\mathbb{P}(\mathcal{Y}=1|x)}{\mathbb{P}(\mathcal{Y}=1|x)+\mathbb{P}(\mathcal{Y}=0|x)} && (3.28)\\
&= \frac{1}{1+\frac{\mathbb{P}(\mathcal{Y}=0|x)}{\mathbb{P}(\mathcal{Y}=1|x)}} && (3.29)\\
&= \frac{1}{1+e^{-\beta\frac{1}{\beta}\log\left(\frac{\mathbb{P}(\mathcal{Y}=1|x)}{\mathbb{P}(\mathcal{Y}=0|x)}\right)}} && (3.30)\\
&= \frac{1}{1+e^{-\beta LL(x)}} && (3.31)
\end{aligned}$$

In this view, the linear function models the log-likelihood ratio of the class given x, also called log-odds or logits. The decision boundary $\{x \in \mathbb{R}^p\,|w^T x + b = 0\}$ is the set of points where, according to the model, x is as likely to be from either class.

This view also motivates the choice of the $\hat{p}$ notation (suggesting we are approximating a probability) and the use of the cross-entropy loss function for choosing the best hypothesis (see next section). The symmetry which consists in inverting the label 0 and 1 would result in $-w_{\log}$ and $-b_{\log}$ being optimal.

3.2.1.3 The logistic program

Under the cross-entropy loss ℓ_{CE}, implementing the ERM principle over the logistic hypothesis space amounts to solving the following program.

Definition 3.2.4 (Binary logistic regression). *The logistic regression program is*

$$\begin{aligned}
w_{\log}, b_{\log} &= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min}\; -\sum_{i=1}^{n}\left(y_i \log \hat{p}(x;w,b) + (1-y_i)\log(1-\hat{p}(x;w,b))\right) && (3.32)\\
&= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min}\; -\sum_{i=1}^{n}\left(y_i(w^T x_i + b) - \log\left(1+e^{w^T x_i + b}\right)\right) && (3.33)\\
&= \underset{(w,b)\in\mathbb{R}^{p+1}}{\arg\min}\; L_{\log}(w,b) && (3.34)
\end{aligned}$$

Using the first order conditions for optimality, we can derive some constraints the optimal parameters must fulfill:

$$\left\{\begin{array}{l}
\frac{\partial L}{\partial b} = \sum_{i=1}^{n}(y_i - \hat{p}(x_i;w,b)) = 0\\
\frac{\partial L}{\partial w^{(j)}} = \sum_{i=1}^{n} x_i^{(j)}(y_i - \hat{p}(x_i;w,b)) = 0
\end{array}\right. \tag{3.35}$$

Contrary to linear regression, it is not possible to deduce an analytical solution from the first-order conditions. Nonetheless, the problem remains convex and is easily solved numerically. The most traditional approach is to use a second-order method since the Hessian is easily computable. We will

Algorithm 1: Stochastic gradient descent (SGD) algorithm for logistic regression.

```
Data: learning set $LS$, initial values $(w_{[0]}, b_{[0]})$, batch size $m$, learning
      rate $\lambda$, number of iterations $K$.
Result: $(w_{[K]}, b_{[K]})$
1 for $k \leftarrow 1$ to $K$ do
2     Draw $S = \{(x_i, y_i)\}_{i=1}^{m}$ randomly from $LS$
3     $w \leftarrow w - \lambda \nabla_w L_S(w, b)$
4     $b \leftarrow b - \lambda \frac{\partial}{\partial b} L_S(w, b)$
```

instead look at a simpler (*i.e.* first-order) iterative method: stochastic gradient descent (SGD). SGD will be the star of Section 3.6, but let us first illustrate it on this simpler problem.

Stochastic gradient descent. Stochastic gradient descent is a randomized variant of the steepest gradient method which, applied to the logistic case, produces a sequence of solutions $(w_{[0]}, b_{[0]}), (w_{[1]}, b_{[1]}), \ldots$ such that, for $k \geq 1$

$$\begin{cases} w_{[k+1]} = w_{[k]} - \lambda \nabla_w L\left(w_{[k]}, b_{[k]}\right) \\ b_{[k+1]} = b_{[k]} - \lambda \frac{\partial}{\partial b} L\left(w_{[k]}, b_{[k]}\right) \end{cases} \tag{3.36}$$

where $\lambda \leq 1$ is called the learning rate and controls the rate of the parameter updates.

In stochastic gradient descent, rather than updating the weights according to the derivatives of L, samples are randomly selected, forming a batch, and the update is performed on that batch, which is viewed as an estimate of L. Both the batch size and the learning rate are viewed as hyper-parameters of the method, fixed at the start once and for all (although extensions of SGD allow for varying λ and even individual learning rate per parameter).

The full SGD algorithm is detailed in Algorithm 1.

Initial values. One degree of freedom with iterative methods, such as SGD, lies in the initial values $w_{[0]}$ and $b_{[0]}$. When standardizing the learning matrix X, the null vector is a good default choice for $w_{[0]}$.

An interesting property of the first order optimality condition with the linear regression was that the intercept encoded the best constant approximation. In the case of logistic regression, we can deduce that the best constant approximation is given when $b = \log \frac{\pi}{1-\pi}$, where $\pi = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the proportion of the positive class. Due to the non-linearity, however, there is no guarantee that $b_{\log} = \log \frac{\pi}{1-\pi}$ is optimal when $w_{\log} \neq 0$. That said, it should be the default value for $b_{[0]}$ when $w_{[0]} = 0$ and X is centered.

Regularized variant. As with linear methods, it is possible to define a regularized version of the logistic regression.

3.2.2 Multiclass logistic regression

3.2.2.1 From two to several classes

A general-purpose, method-agnostic way of extending a binary classification scheme to a multi-class setting is to build a model for any pairs of classes. This one-versus-one approach has the drawback of requiring $K(K-1)/2$ decision boundaries for a K-classes problem.

In the case of logistic regression, it is possible to circumvent the quadratic burden. Firstly, we need to slightly adapt the encoding of the labels.

Label encoding: multi-class logistic regression

For the K-class logistic regression, the labels are assumed such that datapoint belonging to class k is encoded by a one-hot K-dimensional binary vector $y \in \{0,1\}^K$:

$$y^{(j)} = \begin{cases} 1, & \text{if } j = k \\ 0, & \text{otherwise} \end{cases} \tag{3.37}$$

Secondly, we need to replace the sigmoid with a multi-class alternative: the softmax.

Definition 3.2.5 (Softmax function). *For $k = 1, \dots, K$*

$$\text{softmax}(z)^{(k)} = \frac{e^{z^{(k)}}}{\sum_{l=1}^{K} e^{z^{(l)}}} = \frac{1}{1 + \sum_{l \neq k} e^{z^{(l)} - z^{(k)}}} \tag{3.38}$$

where z is a K-dimensional vector.

To obtain a linear boundary in the input space, the K-dimensional z vector must be computed as

$$z = \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(K)} \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_K^T \end{bmatrix} x + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix} = Wx + b \tag{3.39}$$

From there we can define the multi-class hypothesis space for logistic regression.

Definition 3.2.6 (Hypothesis space for logistic regression (multi-class)).

$$\mathbb{H} = \{\hat{p}(x; W, b) = \text{softmax}(Wx + b) \mid W \in \mathbb{R}^{K \times p}, b \in \mathbb{R}^K\} \tag{3.40}$$

The final class decision for a hard classification is $\arg\max_k \text{softmax}^{(k)}(z)$.

Solving the multi-class logistic regression. Let us first state the multi-class program.

Definition 3.2.7 (Multi-class logistic regression). *The multi-class logistic regression (MLR) program is*

$$\begin{aligned}
W_{mlr}, b_{mlr} &= \operatorname*{arg\,min}_{W\in\mathbb{R}^{K\times p}, b\in\mathbb{R}^K} \sum_{i=1}^{n} \ell_{CE}\left(\mathrm{softmax}(Wx+b), y\right) && (3.41)\\
&= \operatorname*{arg\,min}_{W\in\mathbb{R}^{K\times p}, b\in\mathbb{R}^K} -\sum_{i=1}^{n}\sum_{k=1}^{K} y_i^{(k)} \log \hat{p}^{(k)}(x_i; W, b) && (3.42)\\
&= \operatorname*{arg\,min}_{W\in\mathbb{R}^{K\times p}, b\in\mathbb{R}^K} -\sum_{i=1}^{n}\sum_{k=1}^{K} y_i^{(k)} \log \frac{e^{w_k^T x_i + b_k}}{\sum_{l=1}^{K} e^{w_l^T x_i + b_l}} && (3.43)\\
&= \operatorname*{arg\,min}_{W\in\mathbb{R}^{K\times p}, b\in\mathbb{R}^K} L_{mlr}(W, b) && (3.44)
\end{aligned}$$

The same techniques as for the binary logistic regression can be used to solve the multi-class case. Noting that

$$\begin{aligned}
\frac{\partial}{\partial z^{(l)}} \ell_{SCE}\left(z, y\right) &= \frac{\partial}{\partial z^{(l)}} \ell_{CE}\left(\mathrm{softmax}(z), y\right) && (3.45)\\
&= \frac{\partial}{\partial z^{(l)}} \sum_{k=1}^{K} -y^{(k)} \log \frac{e^{z^{(k)}}}{\sum_{j=1}^{K} e^{z^{(j)}}} && (3.46)\\
&= \left(\frac{e^{z^{(l)}}}{\sum_{j=1}^{K} e^{z^{(j)}}} - y^{(l)}\right) = \left(\hat{p}(x)^{(l)} - y^{(l)}\right) && (3.47)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial b_l} \ell_{SCE}(z(x), y) &= \sum_{k=1}^{K} \frac{\partial}{\partial z^{(k)}(x)} \ell_{SCE}(z(x), y) \frac{\partial z^{(k)}(x)}{\partial b_l} && (3.48)\\
&= \frac{\partial}{\partial z^{(l)}(x)} \ell_{SCE}(z(x), y) && (3.49)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial w_l^{(j)}} \ell_{SCE}(z(x), y) &= \sum_{k=1}^{K} \frac{\partial}{\partial z^{(k)}(x)} \ell_{SCE}(z(x), y) \frac{\partial z^{(k)}(x)}{\partial w_l^{(j)}} && (3.50)\\
&= \frac{\partial}{\partial z^{(l)}(x)} \ell_{SCE}(z(x), y) x^{(j)} && (3.51)
\end{aligned}$$

the partial derivatives are (by linearity)

$$\begin{cases}
\frac{\partial}{\partial b_l} L_{mlr} = \sum_{i=1}^{n} \left(\hat{p}^{(l)}(x_i) - y_i^{(l)}\right) \\
\frac{\partial}{\partial w_l^{(j)}} L_{mlr} = \sum_{i=1}^{n} \left(\hat{p}^{(l)}(x_i) - y_i^{(l)}\right) x_i^{(j)}
\end{cases} \qquad (3.52)$$

Figure 3.4 illustrates the result of the softmax regression for classification of three isotropic and homoscedastic Gaussians ($\sigma = 1$) with means placed on an equilateral triangle on the unit circle and twice as many dots as diamonds or stars. We will refer to this as the "3G" problem. It will be helpful at building some insight which will be called in Chapter 7.

Note that, due to the exponential and the normalization, the softmax is translation invariant. As a result, the linear boundaries in Figure 3.4 can be translated without changing anything, provided all boundaries are shifted by the same amount. Along the same lines, multiplying the magnitudes of the vectors defining the hyper-planes by a given amount would change the *softmax probabilities* but not the actual *hard classification*. As such, softmax regression might benefit from being regularized, especially for linearly separable problems.

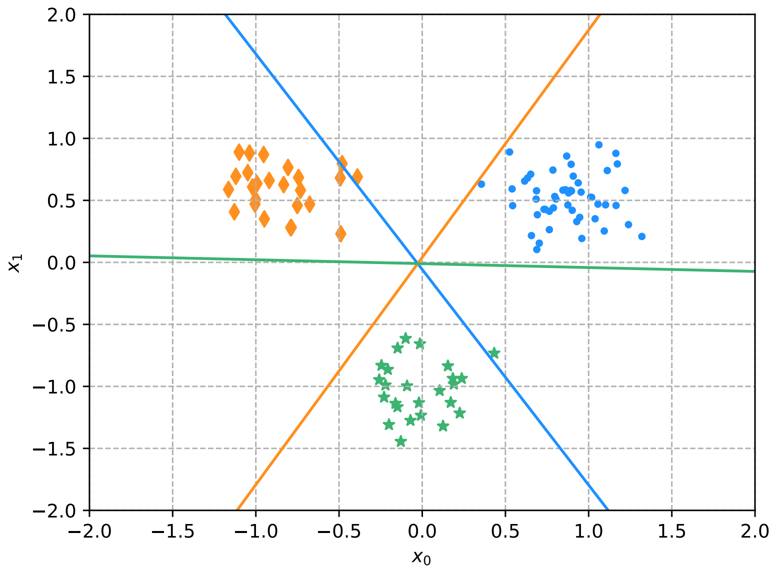


FIGURE 3.4: Result of the softmax regression for the classification of three isotropic and homoscedastic Gaussians ($\sigma = 1$) with means placed on an equilateral triangle on the unit circle and twice as many dots as diamonds or stars (aka. the "3G" problem).

3.2.2.2 Interpreting the softmax regression

Geometrical interpretation. For a pair (x, y), gradient descent implies the $w_l' = w_l - \lambda \nabla_{w_l} \ell_{CE}(\hat{p}(x), y)$ update . Figure 3.5 illustrates the effect of such an update. The gradient for the pair (x, y) follows the direction of the gradient linking the origin to x and the origin, implying the gradient will always point either towards (incorrect classification) or away from the origin (correction classification). Its magnitude is the one of x weighted by how well is the prediction there $(\hat{p}^{(l)}(x) - y^{(l)})$ (times the learning rate). The effect is twofold. Firstly, this update tends to rotate the hyper-plane so as to align it with x (orthogonal contribution). Secondly, the norm of the gradient vectors is magnified for correct classifications and lessened otherwise. Where the hyper-planes sit with respect to the origin is determined by the intercepts so that the model predictions reflect the class balance of the learning rate.

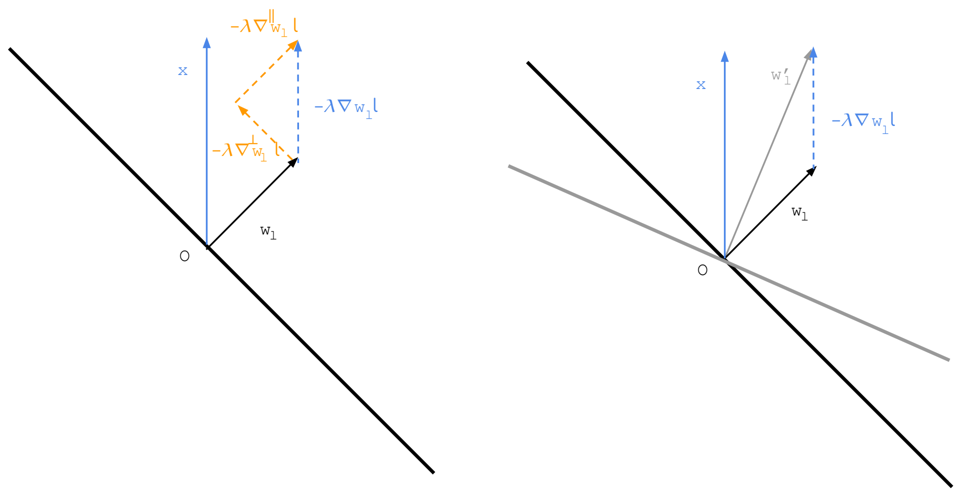

$-\lambda\nabla^{\parallel}_{w_1} l$
x
$-\lambda\nabla w_1 l$
$-\lambda\nabla^{\perp}_{w_1} l$
w_1
O
x
w'_1
$-\lambda\nabla w_1 l$
w_1
O

FIGURE 3.5: Softmax regression: effect of a gradient descent update for x for the hyper-plane w_l of the corresponding class. The update can be decomposed into an orthogonal component, responsible for rotating the boundary, and a parallel component, responsible for increasing or decreasing the magnitude of the hyper-plane in case of correct or incorrect classification respectively (left). Result of the update $w'_l = w_l - \lambda \nabla_{w_l} \ell$ (right).

An easily-missed subtlety regarding the update is that all samples contribute to settling the boundary: samples belonging to the class, as well as samples from the other classes. This implies that there are always classes whose hyper-planes are determined more by samples from the other classes than by samples from theirs. This is the case of all classes in the common setting where classes are balanced and is amplified as the number of classes increases. This is illustrated at Figure 3.6.

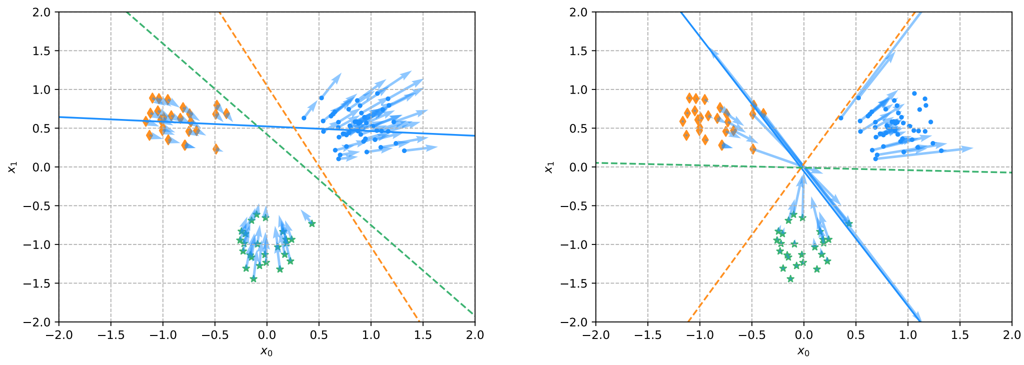
The struggle between the classification quality and the norm of a sample, illustrated at Figure 3.6, is better visualized in Figure 3.7. As is apparent, the exponential in the softmax rapidly wipes away the effect of the magnitude of the sample. How these two effects couple is also illustrated in more detail in Figure 3.7. Overall, the softmax regression will focus much more on misclassification than on samples far from the boundary.

Softmax interpretation of hyper-planes. There is a slight difference between how the hyper-planes are defined in the softmax variant compared to the sigmoid. The log-likelihood ratio for a pair k-l of classes is

$$\log \frac{\hat{p}^{(k)}(x)}{\hat{p}^{(l)}(x)} = \log \frac{e^{z^{(k)}}}{e^{z^{(k)}}} = (w_k^T - w_l^T)x + (b_k - b_l) \tag{3.53}$$

$$= \Delta w_{k,l}^T x + \Delta b_{k,l} \tag{3.54}$$

Therefore the one-versus-one hyperplanes separating the classes ($\Delta w_{k,l}$) are not the ones appearing in the softmax (w_k and w_l) but can be derived from



(A) Random Hyper-planes (e.g. start of training).

(B) Well-fitting hyper-planes.

FIGURE 3.6: Gradient update of the training instances with respect to the hyper-plane corresponding to the dotted class. Note that the scale of the gradients is different between the two figures. Is illustrated the centripetal/centrifugal nature of the gradient updates, the fact that only the magnitudes (taken loosely enough to include the sense of the vector) change during learning, the fact that a large magnitude is associated with misclassifications, and, finally, that correctly classified points far away from the boundary contribute little to the optimization.

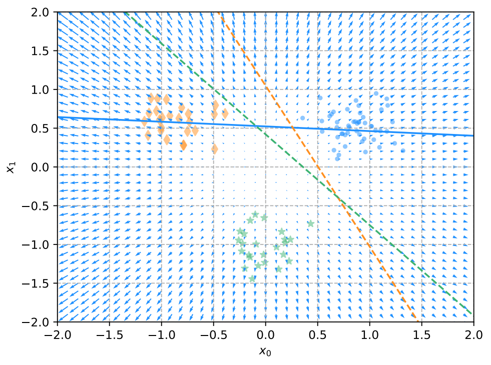
them. This allows to keep track of only $O(K)$ planes instead of $O(K^2)$. The k-versus-l decision boundary still is the set of points x where the likelihood of belonging to k or l is the same.

Interpreting the meaning of the kth softmax hyper-plane from a standalone perspective is more tricky. A bit of algebra leads to

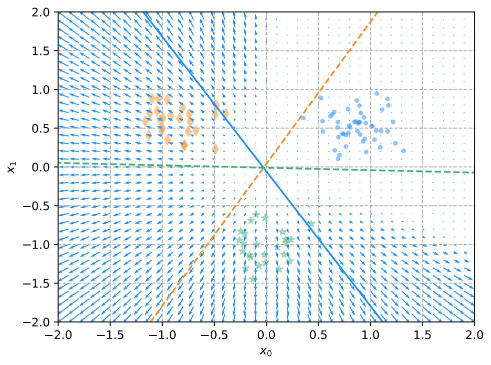
$$\begin{aligned} w_k^T x + b_k &= \log \frac{\hat{p}^{(k)}(x)}{1-\hat{p}^{(k)}(x)} + \log \sum_{l \neq k} e^{w_l^T x + b_l} && (3.55) \\ &= \log \frac{\hat{p}^{(k)}(x)}{1-\hat{p}^{(k)}(x)} + \underset{l \neq k}{\mathrm{LSE}} \{w_l^T x + b_l\} && (3.56) \\ &= \log \frac{\hat{p}^{(k)}(x)}{1-\hat{p}^{(k)}(x)} + \max_{l \neq k} \{w_l^T x + b_l\} + O(\log(K-1)) && (3.57) \end{aligned}$$

where $\max\{u_1, \ldots, u_m\} < \mathrm{LSE}\{u_1, \ldots, u_m\} \leq \max\{u_1, \ldots, u_m\} + \log m$ is called the logsumexp; a smooth approximation of the maximum. LSE approaches its lower bound when the maximum is much greater than the other values, and its upper bound when all the values are almost equals.

After optimization, where the kth hyper-planes sits is controlled by (i) where the kth class lies in the $\mathbb{X}$ space (log-likelihood term), (ii) how close the closest class is (the maximum term), and (iii) some safety margin when many classes are close in the $\mathbb{X}$ space (last term).



(A) Random Hyper-planes. When the hyper-planes are random, only the samples with a small norm have little impact.



(B) Well-fitting hyper-planes. When the hyper-planes are well aligned with the classes, the contribution of correctly classified, far away points is small compared to how a misclassification would impact the boundary.

FIGURE 3.7: Gradient field of how a given point labeled as a dot would contribute to changing the corresponding hyper-planes. Note that the scale of the gradients is different between the two figures.

3.3 Decision trees

Decision trees (Breiman et al., 1984) operate on a very different paradigm than linear methods. The latter form a single linear boundary (between two classes) with a combination of weighted features by solving a global optimization problem over a parametric hypothesis space of relatively low expressiveness. Decision trees recursively partition the input space by thresholding the features in a greedy fashion with a non-parametric hypothesis space of learning-set-size-dependent expressiveness.

We will assume the same setting as for linear models, that is $\mathbb{X} = \mathbb{R}^p$ and $\mathbb{Y}$ is either discrete (classification) or continuous (regression). In the former case, we will assume, without loss of generality, that classes are encoded so that $\mathbb{Y} = 1, 2, \ldots, K$. Decision trees can be used with discrete input features as well, which will be further discussed in Section 3.7.

In this section, we will first establish what a decision tree is and how it is used in the context of classification and regression (Section 3.3.1). We will then describe how a decision tree is learned from a training set (Section 3.3.2). Finally, we will discuss how to control the expressiveness of decision trees (Section 3.3.3).

3.3.1 Inference

Before discussing how a decision tree is selected according to the data, we need to be clear on what a decision tree is and how it is used as a hypothesis. To do so, we will go over a few definitions.

Definition 3.3.1 (Full binary tree)**.** *A full binary tree is either a* leaf *or an* internal node*. An internal node has exactly two children (the left one and the right one), which are full binary trees. The internal node is said to be the parent of its children. A full binary tree with no parent is called the root.*

Definition 3.3.2 (Decision node)**.** *A decision node is an internal node associated with a predicate over $\mathbb{X}$.*

Definition 3.3.3 (Splitting node)**.** *A splitting (or thresholding) node $\tau(\cdot; j, t)$ ($1 \leq j \leq p$, $t \in \mathbb{R}$) is a decision node whose predicate is of the form*

$$\tau(x; j, t) = \mathbb{I}\left(x^{(j)} \leq t\right) \tag{3.58}$$

Definition 3.3.4 (Classification leaf)**.** *A classification leaf is a leaf associated with either a class or a probability vector. The leaf prediction is the class (former case) or the class with maximum probability (latter case).*

Definition 3.3.5 (Regression leaf)**.** *A regression leaf is a leaf associated with either a value $y \in \mathbb{Y}$ or a Gaussian distribution $\mathcal{N}(\hat{\mu}_y, \hat{\sigma}_y)$. The leaf prediction is the value y (former case) or $\hat{\mu}_y$ (latter case).*

Definition 3.3.6 (Classification tree)**.** *A classification tree is a binary full tree whose internal nodes (if any) are decision nodes and whose leaves are classification leaves.*

Definition 3.3.7 (Regression tree)**.** *A regression tree is a binary full tree whose internal nodes (if any) are decision nodes and whose leaves are regression leaves.*

Definition 3.3.8 (Decision tree)**.** *A decision tree is either a classification or a regression tree.*

Decision trees are used as hypotheses $\mathbb{X} \rightarrow \mathbb{Y}$ by following a branch according to the predicates of the internal nodes and offering the prediction held at the leaf. This implies that the hypotheses are piece-wise constant. The full inference process is detailed in Algorithm 2. A classification tree and its corresponding boundary are depicted in Figure 3.8. Only the class or the value y is of interest so far as inference is concerned. Having access to the full probability vector or standard deviation is useful in many situations, however.

3.3.2 Induction

Decision trees, contrary to linear models, are not parametric; learning the tree is not akin to selecting the best parameters: the structure, as well as the splits and the leaf decisions, must be optimized. This does not lead to a continuous optimization program. Instead, decision trees are built sequentially in a greedy fashion.

Intuitively, a tree is developed by expanding its leaves, turning them into internal nodes. At each leaf, the question of which split (*i.e.* feature and threshold) will reduce the uncertainty over the prediction is answered by an

Algorithm 2: Decision tree inference algorithm.

```
Input: Decision tree $T$, input vector $x$
Output: decision (class or output)
Function TreeInference($T$, $x$):
    if $T$ is a leaf then
        return $T.decision$
    else
        $j \leftarrow T.feature\_index$
        $t \leftarrow T.threshold$
        if $x_j \leq t$ then
            return TreeInference($T$, $x.left$)
        else
            return TreeInference($T$, $x.right$)
```

exhaustive search. The reduction is such that the combined uncertainty in the newly created leaves should be less than in their parent node. The process is repeated until uncertainty cannot be further reduced (*i.e.* the node is pure), or some other criteria are met. The prediction associated with a leaf is established based on the ERM principle to offer the best constant predictor. In classification (from the ℓ_{0-1} perspective), this corresponds to the most represented class of the learning sub-sample reaching that leaf. In regression, it corresponds to the mean value (from the ℓ_2 perspective).

Uncertainty is at the core of the induction process. To be of any use, this metric must follow some properties. A splitting node $\tau(\cdot; j, t)$ is such that it partitions a sample in two. Let us denote its subsets.

Definition 3.3.9 (Right and left subsets)**.** *Let $S = \{(x_i, y_i)\}_{i=1}^n$ be a sample.*

$$L_{\tau(\cdot;j,t)}(S) = \{(x_i, y_i) \subseteq S | x_i^{(j)} \leq t\} \tag{3.59}$$

$$R_{\tau(\cdot;j,t)}(S) = S \setminus L_{\tau(\cdot;j,t)}(S) \tag{3.60}$$

For readability, we will shorten the notation to L_τ and R_τ.

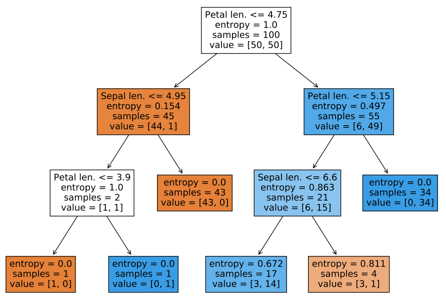
Definition 3.3.10 (Uncertainty)**.** *U is an uncertainty measure over sample sets. It is such that*

$$U(S) \geq 0 \tag{3.61}$$

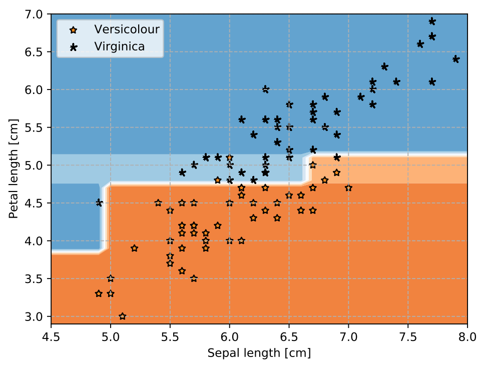
$$\Delta_\tau U(S) \geq 0 \tag{3.62}$$

where

$$\begin{aligned} \Delta_\tau U(S) &= U(S) - U(S|\tau) && (3.63) \\ &= U(S) - \left(\frac{|L_\tau(S)|}{|S|} U\left(L_\tau(S)\right) + \frac{|R_\tau(S)|}{|S|} U\left(R_\tau(S)\right) \right) && (3.64) \end{aligned}$$



(A) Structure of a classification tree. The first line in the box corresponds to the split (absent in leaves). The second line is the entropy of the sample in terms of class distribution. The third line is the number of samples and the following line is the distribution in terms of classes.



(B) A classification boundary. The boundary is piece-wise and axis-aligned, a manifestation of the path along splitting nodes.

FIGURE 3.8: A decision (classification) tree and its boundary for the iris species classification.

Moreover, when $U(S)$ is small (resp. big) the uncertainty is low (resp. high) and we say that S is pure when $U(S) = 0$.

Note that uncertainty only ever relates to the conditional output distribution at a node. The second condition imposes that the uncertainty of the parent node be higher than the weighted combination (by the proportion of samples) of the uncertainties at the children nodes.

The full top-down decision tree induction is covered in Algorithm 3. Note that this is a high-level view and implementation details may vary, most notably for efficiency reasons. Sections 3.3.2.1 and 3.3.2.2 propose measures of uncertainty for classification and regression, respectively.

3.3.2.1 Uncertainty in classification

In classification, uncertainty relates to the empirical distribution of classes, conditional to a node. Let K be the number of classes and

$$p_k(S) = \frac{|\{(x,y) \in S | y = k\}|}{|S|} \tag{3.65}$$

be the proportion of objects of class k ($k = 1, \dots, K$) in sample S, and let $p(S) = [p_1(S) \dots p_K(S)]^T$ be the corresponding class proportion vector. Uncertainty measures are defined in terms of $p(S)$, with two common choices being the Shannon entropy and the Gini index.

Entropy. A common choice of uncertainty score in classification is the empirical Shannon entropy H.

Algorithm 3: Decision tree induction algorithm.

```
Input: A learning set $LS = \{(x_i, y_i)\}_{i=1}^n$
Output: A decision tree $T$
Function TreeInduction($LS$):
    if $U(LS) = 0$ (or some other stopping criterion is met) then
        return MakeLeaf($LS$)
    /* Internal node:  find the best split with an exhaustive search.
       */
    $j^*, t^*, u^* \leftarrow 0, 0, +\infty$
    for $j \leftarrow 1$ to $p$ do
        for $i \leftarrow 1$ to $n$ do
            $t \leftarrow x_i^{(j)}$
            $L \leftarrow \{(x_i, y_i) \subseteq S | x_i^{(j)} \leq t\}$
            $R \leftarrow S \setminus L$
            $\alpha \leftarrow |L|/|S|$
            $u \leftarrow \alpha U(L) + (1 - \alpha)U(R)$
            if $u < u^*$ then
                $j^*, t^*, u^* \leftarrow j, t, u$
    if $u^* = U(LS)$ then
        return MakeLeaf($LS$)
    /* Create internal node with the best split.                    */
    $T \leftarrow$ MakeEmptyTree()
    $T.feature\_index \leftarrow j^*$
    $T.threshold \leftarrow t^*$
    /* Recursively developed the children.                          */
    $L \leftarrow \{(x_i, y_i) \subseteq S | x_i^{(j^*)} \leq t^*\}$
    $R \leftarrow S \setminus L$
    $T.left \leftarrow$ TreeInduction($L$)
    $T.right \leftarrow$ TreeInduction($R$)
    return $T$
/* MakeLeaf($S$) creates a leaf and sets the prediction value according
   to $S$ to be the best constant prediction.                       */
```

Definition 3.3.11 (Empirical Shannon entropy)**.** *The empirical Shannon entropy $H(S)$ is*

$$U_H(S) = H(p(S)) = -\sum_{k=1}^{K} p_k(S) \log_2 p_k(S) \tag{3.66}$$

In this context, we say that $\Delta_\tau U_H(S)$ is the information brought by the split.

Gini index. Another popular choice for uncertainty is the Gini index:

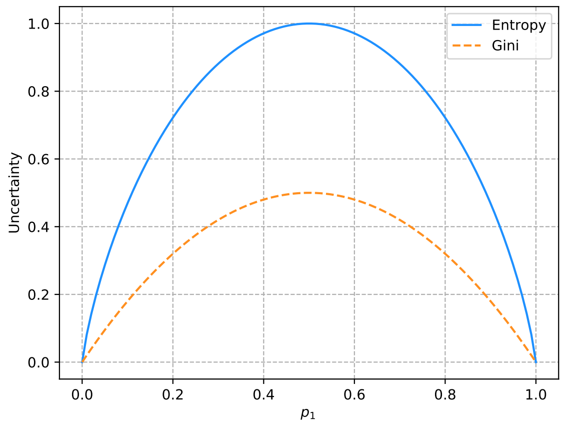


FIGURE 3.9: Entropy and Gini index as uncertainty measures for binary classification. The uncertainty is highest when $p_1 = 0.5$ and lowest when $p_1 = 0$ or $p_1 = 1$. Both measures can be rescaled to have a maximum value of 1 irrespective of the number of classes.

Definition 3.3.12 (Gini index)**.**

$$U_{Gi} = Gi(p(S)) = \sum_{k=1}^{K} p_k(S)(1 - p_k(S)) \tag{3.67}$$

A comparison between entropy and the Gini index in the case of binary classification is given in Figure 3.9. Both measures have their maximum at $p_k = 1/K\,\forall k$.

The fact that $\Delta_\tau U(S) \geq 0$ is a direct consequence of partitioning and the concavity. Indeed, let $|L_\tau(S)| = \alpha|S|$. It follows that $p(S) = \alpha p(L_\tau(S)) + (1-\alpha)p(R_\tau(S))$. Thus re-stating the uncertainty reduction in term of p rather than S leads to concavity.

3.3.2.2 Uncertainty in regression

The most common criterion in regression is the variance and the reduction of the uncertainty is a consequence of the law of total variance. Let

$$\mu_y(S) = \frac{1}{|S|} \sum_{(x,y)\in S} y \tag{3.68}$$

$$\sigma_y^2(S) = \frac{1}{|S|-1} \sum_{(x,y)\in S} (y - \mu_y(S))^2 \tag{3.69}$$

The law of total variance applied to the split τ state that

$$\begin{aligned}\sigma_y^2(S) = &\left(\frac{|L_\tau(S)|}{|S|}\sigma_y^2\left(L_\tau(S)\right) + \frac{|R_\tau(S)|}{|S|}\sigma_y^2\left(R_\tau(S)\right)\right) \\ &+ \left(\left(\mu_y(L_\tau(S)) - \mu_y(S)\right)^2 + \left(\mu_y(R_\tau(S)) - \mu_y(S)\right)^2\right)\end{aligned} \tag{3.70}$$

The first term is the weighted variance in the children (often called the "within" variance) and the second term reflects the distance between the predictions of the children (often called the "between" variance). Since the second term is always positive, the variance conforms to the uncertainty reduction, which measures how much of the variance is due to mixing the partition.

3.3.3 Expressiveness, stopping criteria and pruning

The most natural stopping criterion for the induction is to stop when the uncertainty can no longer be reduced. Nonetheless, providing other stopping criteria may be necessary. Although greedy in nature, the induction algorithm fully optimizes its local decisions: the splits. As the tree deepens, fewer and fewer datapoints are taken into account for this process. When the tree is fully developed, its leaves are pure, which results in highly confident predictions with a null re-substitution error. In short, overfitting is likely. Contrary to parametric hypothesis space, this state of affairs does not depend on the size of the learning set: whatever the size, fully-developed trees are likely, by design, to overfit.

Formulating a penalization, in this case, is not straightforward. What is, however, is limiting the growth of the tree by providing another stopping criterion. Many criteria are possible: limiting the depth, stopping when a certain threshold of uncertainty is reached, stopping when only a few learning samples reach a given node (or would reach one of its children), etc.

This process of stopping early the induction is known as pre-pruning. Alternatively, the tree can be fully built and then some branches can be cut off as a post-processing step. This is known as (post-)pruning. However pruning is carried, it lowers the variance and increases the bias of the learning algorithm.

Stopping criteria come with hyper-parameters: what maximum depth should be allowed? What uncertainty level should be reached? How many points are required? These hyper-parameters can be optimized by cross-validation to find the adequate expressiveness level for the task.

An altogether different solution to overfitting lies in what is called ensemble methods.

3.4 Ensemble methods: decision forests

Pruning decision trees plays on the bias-variance tradeoff. Shorter trees result in lower variance but higher bias. The ensemble methods we will investigate in this section aim at lowering the variance without raising the bias.

We will look at three such methods: bagging (Section 3.4.1), random forests (Section 3.4.2), and extremely randomized trees (Section 3.4.3).

3.4.1 Bagging

Bagging (**b**ootstrap **agg**regat**ing**) was introduced by Breiman (1996). It is a general technique, meaning it is not restricted to decision trees, but will be our stepping stone for the other methods. Let us first describe what it aims at achieving.

3.4.1.1 Variance reduction

Definition 3.4.1 (Ensemble model of independent hypotheses). *Let LS_1, $LS_2, \dots, LS_m$ be m learning samples drawn independently from $\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}$, and let $h_1 = \mathcal{L}_\phi(LS_1), \dots, h_m = \mathcal{L}_\phi(LS_m)$ be the hypotheses (aka. base models) selected by the learning algorithm $\mathcal{L}_\phi$ for each learning set.*

$$h_{ens}(x) = \frac{1}{m}\sum_{l=1}^{m} h_l(x) \tag{3.71}$$

is the ensemble over $h_1, \dots, h_m$. We will denote its distribution by $\mathcal{H}_{\mathcal{L}_\phi, \mathbb{P}^{(n\times m)}_{\mathcal{X},\mathcal{Y}}}$, which we will shorten as $\mathcal{H}_\phi^{(n\times m)}$ for ease of notation.

In classification, the ensemble operates on the intermediate representation (such as the probability vectors outputted by the models).

Note that we used ϕ to denote that all base models are learned with the same set of hyper-parameters. The learning algorithms are assumed to be fully deterministic.

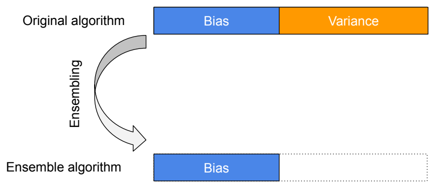
To understand how ensembling improves upon the base models, we will investigate its bias-variance decomposition in the case of regression, under the ℓ_2 loss. The expected risk of the ensemble algorithm is

$$\begin{aligned}
\mathcal{R}^{(n\times m)}_{\phi,\mathcal{Y}|x} &= \mathbb{E}_{\mathcal{H}_\phi^{(n\times m)}}\{\mathbb{E}_{\mathcal{Y}|x}\{(h(x)-y)^2\}\} && && (3.72)\\
&= \mathbb{E}_{\mathcal{Y}|x}\{(y-h_B(x))^2\} && \text{noise}(x) \\
&\quad + (h_B(x) - \mathbb{E}_{\mathcal{H}_\phi^{(n\times m)}}\{h_{ens}(x)\})^2 && \text{bias}^2(x) \\
&\quad + \mathbb{V}_{\mathcal{H}_\phi^{(n\times m)}}\{h_{ens}(x)\} && \text{variance}(x) && (3.73)
\end{aligned}$$

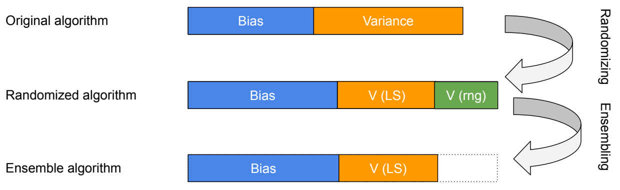
The noise term remains, obviously, unchanged. By linearity the average of the ensemble is also the average over the base models:

$$\mathbb{E}_{\mathcal{H}_\phi^{(n\times m)}}\{h_{ens}(x)\} = \mathbb{E}_{\mathcal{H}_\phi^{(n)}}\{h(x)\} \tag{3.74}$$

Therefore, the bias is unaffected by ensembling, as well.



(A) Variance reduction due to ensembling models on different learning sets.



(B) Variance reduction due to ensembling a randomized variant of a base algorithm (adapted from (Geurts, Ernst, and Wehenkel, 2006)). $V(rng)$ is the part of the variance due to the randomization while $V(LS)$ is the part of the variance due to the learning set.

FIGURE 3.10: Schematic illustration of variance reduction in the asymptotic case ($m \to +\infty$).

The variance, on the other hand, is reduced thanks to the base models being independent (from Eq. 3.75 to 3.77):

$$\mathbb{V}_{\mathcal{H}_\phi^{(n\times m)}}\{h_{ens}(x)\} = \mathbb{V}_{\mathcal{H}_\phi^{(n\times m)}}\left\{\frac{1}{m}\sum_{l=1}^{m} h_l(x)\right\} \tag{3.75}$$

$$= \frac{1}{m^2}\sum_{l=1}^{m} \mathbb{V}_{\mathcal{H}_\phi^{(n)}}\{h(x)\} \tag{3.76}$$

$$= \frac{1}{m}\mathbb{V}_{\mathcal{H}_\phi^{(n)}}\{h(x)\} \tag{3.77}$$

Variance reduction is illustrated in Figure 3.10a.

3.4.1.2 Bootstrap

As presented so far, the variance reduction method is purely theoretical. In practice, there is only one learning sample. Splitting it would reduce each chunk's size to n/t, which influences negatively both the bias and variance. Thus we cannot say that the variance is simply divided by t and the bias unchanged compared to the base models. Bagging tempers this issue by creating a bootstrap sample instead of dividing the learning sample.

Bootstrap samples are obtained by drawing with replacement from the learning set so that $|LS_1| = \ldots = |LS_t| = |LS|$. Bootstrap is illustrated in Figure 3.11. Drawing with replacement bears a consequence on the distribution as well. The probability of not selecting a given datapoint in a bootstrap sample is $\left(1 - \frac{1}{n}\right)^n$. As $n \to \infty$, this probability converges to $\frac{1}{e} \approx 0.37$. Thus, for a large dataset, about $1/3$ of the data will not be seen when learning a base model, the voids being filled with some objects appearing twice or more. In effect, the bootstrap learning sets are worth only about $2/3$ of n.

On the positive side, the missing datapoints (called the out-of-bag examples) can be used as test samples to assess the base model error.

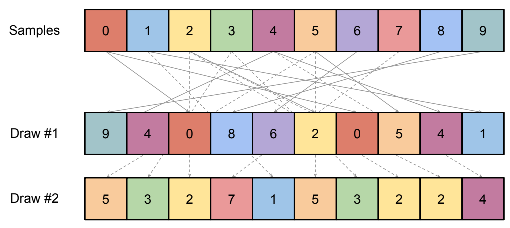


FIGURE 3.11: Bootstrap technique to simulate several learning sets.

3.4.2 Random forests

Random forests, also introduced by Breiman (2001), capitalize on the bagging technique and introduce more randomness in the specific case of decision trees. Increasing stochasticity has several implications. Firstly, the learning algorithm for the base model is changed and the bias-variance decomposition is impacted further than a reduced number of samples: the decomposition can no longer be related to the vanilla base models. This simple change increases the bias of the modified base learning algorithm compared to the original. On the other hand, this increases the amount of variance which can be reduced when averaging, compared to bagging. Overall, the tradeoff is usually in favor of random forests.

The added randomness in the case of the algorithm proposed in (Breiman, 2001) concerns the splits. Rather than looking exhaustively for the optimal split, only a subset of $1 \leq p' \leq p$ features are considered (line 5 in Algorithm 3). Note that the threshold is still fully optimized. p' controls the randomization level: when $p = p'$, random forest defaults to bagging; when $p = 1$, the splitting feature is chosen fully at random.

3.4.3 Extremely randomized trees

Geurts, Ernst, and Wehenkel (2006) introduce another ensembling technique revolving around decision trees: extremely randomized trees (extra-trees for short). Compared to random forests, the authors propose to circumvent the bootstrapping part and further increase the randomness instead. More specifically, they propose to select also the threshold (line 6 in Algorithm 3) randomly between the maximum and minimum value of the attribute (at that node). Therefore, only $1 \leq p' \leq p$ are considered at any node.

Removing the bootstrapping part in favor of other randomization mechanisms gives rise to a different bias-variance decomposition.

Definition 3.4.2 (Ensemble model of randomized hypotheses)**.** *Let LS be a learning sample drawn from $\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}$, and let $h_{\phi_1} = \mathcal{L}_{\phi_1}(LS), \ldots, h_{\phi_t} = \mathcal{L}_{\phi_t}(LS)$ be* *the hypotheses selected independently by the randomized learning algorithm.*

$$h_{ens}(x) = \frac{1}{m}\sum_{l=1}^{m} h_{\phi_l}(x) \tag{3.78}$$

is the ensemble over $h_{\phi_1}, \ldots, h_{\phi_t}$. *We will denote its distribution by* $\mathcal{H}_{\mathcal{L}_{\mathfrak{F}^m}, \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}$ *and shorten it as* $\mathcal{H}^{(n)}_{\mathfrak{F}^m}$.

In regression (under the ℓ_2 loss), the bias-variance decomposition of the ensemble is given by

$$\mathcal{R}^{(n)}_{\mathfrak{F}^m,\mathcal{Y}|x} = \mathbb{E}_{\mathcal{H}^{(n)}_{\mathfrak{F}^m}}\{\mathbb{E}_{\mathcal{Y}|x}\{(h(x)-y)^2\} \tag{3.79}$$

$$\begin{aligned} = \quad & \mathbb{E}_{\mathcal{Y}|x}\{(y-h_B(x))^2\} && \text{noise}(x) \\ & + (h_B(x) - \mathbb{E}_{\mathcal{H}^{(n)}_{\mathfrak{F}^m}}\{h_{ens}(x)\})^2 && \text{bias}^2(x) \\ & + \mathbb{V}_{\mathfrak{F}^m, \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{h_{ens}(x)\} && \text{variance}(x) \end{aligned} \tag{3.80}$$

Once more, the noise is unaffected (it does not depend on the algorithm). The bias is also unchanged compared to the bias of the base (randomized) algorithm:

$$bias(x) = h_B(x) - \mathbb{E}_{\mathcal{H}^{(n)}_{\mathfrak{F}}}\{h_\phi x)\} \tag{3.81}$$

The variance can be further decomposed (using the law of total variance) as

$$\mathbb{V}_{\mathfrak{F}^m, \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{h_{ens}(x)\} = \mathbb{V}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{E}_{\mathfrak{F}^m|LS}\{h_{ens}(x)\}\} + \mathbb{E}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{V}_{\mathfrak{F}^m|LS}\{h_{ens}(x)\}\} \tag{3.82}$$

$$= \mathbb{V}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{E}_{\mathfrak{F}|LS}\{h_\phi(x)\}\} + \frac{1}{m}\mathbb{E}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{V}_{\mathfrak{F}|LS}\{h_\phi(x)\}\} \tag{3.83}$$

where LS is the realization of $\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}$. To understand how it relates to the base (randomized) algorithm, we can compare the variances. For the base algorithms, the variance is

$$\mathbb{V}_{\mathfrak{F}, \mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{h_\phi(x)\} = \mathbb{V}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{E}_{\mathfrak{F}|LS}\{h_\phi(x)\}\} + \mathbb{E}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{V}_{\mathfrak{F}|LS}\{h_\phi(x)\}\} \tag{3.84}$$

The first terms in Equations 3.83 and 3.84 are the same. It is the variability of the average model due to the learning set. The second term accounts for the average variability due to the randomization of the algorithm. The ensemble is able to divide this amount by a factor of m. This second aspect of the variance can be made arbitrarily small by increasing the size of the ensemble. Since this is the only effect of ensembling, a monotonous decrease of the error is expected when increasing m.

3.4.3.1 On randomized algorithms

The previous bias-variance decomposition informs us there is always a gain to ensembling over the base models when they are issued from a *randomized* algorithm. Either the algorithm is stochastic by design or some randomization is added to a deterministic algorithm. Extra-trees fall into the latter category, therefore we can distinguish between the decision tree induction algorithm and the extra-tree induction algorithm. References to the base models or the base algorithm in the previous bias-variance decomposition pointed to the extra-tree induction algorithm (or induced models). Consequently, there is always a gain—*compared to a single extra-tree*—to form an ensemble. But is it always advantageous over using a sole, regular decision tree?

Ensembling randomized variants only make sense when the overall variance reduction is greater than the added bias and variance brought by randomizing the algorithm.

It is expected that the bias of a (model consisting in a single) extra-tree, which is also the bias of the ensemble, will be higher than the bias of the regular decision tree induction. This is because randomization blurs somewhat the effectiveness of the hypothesis selection mechanism.

Therefore, there is a gain only if $\mathbb{V}_{\mathbb{P}^n_{\mathcal{X},\mathcal{Y}}}\{\mathbb{E}_{\mathfrak{F}|LS}\{h_\phi(x)\}\}$ is lower (by at least the same amount as the bias increases) than the variance of the base, non-randomized algorithm. Intuitively, this happens because randomization forces the optimization to depart somewhat from the learning set. The overall effect is schematically summarized by Figure 3.10b. There is no guarantee that ensembling a randomized variant of a given algorithm will actually improve over the vanilla algorithm; it might well be that the direct reduction of variance brought by randomizing does not compensate for the bias increase.

3.5 Boosting

Whereas bagging and random forests aim at reducing variance, boosting is a method that tackles bias. A highly-biased learning algorithm is called a weak learner. The idea behind boosting is to combine weak learners in a sequential fashion under an additive model with an increasing number of terms whose goals are to fit what is not yet captured by the first terms (usually applying some learning rate for regularization purposes).

A generic procedure implementing the idea of boosting, called forward stagewise additive modeling (Friedman, 2001a) is described in Algorithm 4. In the remaining, we will restrict the discussion to weak learners being shallow decision trees. Such trees split the input space into a handful of coarse regions. As a result, the predictions tend to be highly biased and portraying low variance. The extreme case is when there is only a single splitting node and two leaves, which is called a stump.

Section 3.5.1 discusses one way boosting can be implemented for regression, while Section 3.5.2 discusses the case of classification.

Algorithm 4: Forward stagewise additive modeling (boosting).

```
Input: A learning set $LS = \{(x_i, y_i)\}_{i=1}^n$, a hypothesis space $\mathbb{H}$, the
        number of terms $t$, the learning rate $\lambda$
Output: A generalized linear model $h_*(x) = w_1 h_1(x) + \ldots + w_t h_t(x)$
1 Function FSAM($LS$, $\mathbb{H}$, $t$):
2     $h_{[0]} \leftarrow 0$
      /* Or the best constant wrt LS */
3     for $l \leftarrow 1$ to $m$ do
4         compute
              $(w_l, h_l) \leftarrow \arg\min_{(w,h)\in\mathbb{R}\times\mathbb{H}} \sum_{i=1}^n \ell(y_i, h_{[l-1]}(x_i) + wh(x_i))$
5         $h_{[l]}(\cdot) \leftarrow h_{[l-1]}(\cdot) + \lambda w_l h_l(\cdot)$
6     return $h_{[t]}$
```

3.5.1 Least square boosting

Specializing Algorithm 4 for regression is relatively straightforward (Friedman, 2001a). Using the ℓ_2 loss, the stagewise optimization becomes

$$\min_{(w,h)\in\mathbb{R}\times\mathbb{H}} \sum_{i=1}^n (y_i - h_{l-1}(x_i) - wh(x_i))^2 \tag{3.85}$$

We note that $r_i^{(l)} \triangleq y_i - h_{l-1}(x_i)$ is the residual of the previous iteration for the ith object. By feeding the regression tree induction algorithm a learning set $LS_l = \{(x_i, r_i^{(l)})\}_{i=1}^n$, the returned tree will solve the optimization for $w = 1$. Therefore, we obtain a model of the form

$$h(x) = h_0(x) + h_1(x) + \ldots + h_t(x) \tag{3.86}$$

Since, under the ℓ_2 loss, the best constant is the average, we can opt for $h_0(x) = 1/t \sum_{i=1}^n y_i$. As in the general case, a learning rate can be applied to the terms to regularize learning.

3.5.2 Adaboost

We will first discuss binary classification with Adaboost (Freund and Schapire, 1995). Adaboost can be seen as an instantiation of the forward stagewise additive modeling with the exponential loss. The labels follow a different encoding from the logistic case.

Label encoding: binary Adaboost

In Adaboost, $\mathbb{Y} = \{-1, 1\}$ so that one class (the "negative" class) receives the label -1 and the other (the "positive" class) receives the label 1.

Definition 3.5.1 (Exponential loss (binary classification)). *With the $\{-1,1\}$-encoding the exponential loss is*

$$\ell_{\exp}(y, z) = e^{-yz} \tag{3.87}$$

The $-yz$ product is called the margin.

Friedman, Hastie, Tibshirani, et al. (2000) showed that

$$h_B(x) = \arg\min_{\hat{y}} \mathbb{E}_{y|x}\{e^{-y\hat{y}(x)}\} = \frac{1}{2}\log\frac{\mathbb{P}(Y=1|x)}{1-\mathbb{P}(Y=1|x)} \tag{3.88}$$

Thus, optimizing under the exponential loss results in the same solution as with the cross-entropy. Moreover, this allow us to interpret the model in probabilistic terms:

$$h_B(x) = \frac{1}{2}\log\frac{\mathbb{P}(Y=1|x)}{1-\mathbb{P}(Y=1|x)} \tag{3.89}$$

$$\iff \mathbb{P}(Y=1|x) = \frac{1}{1+e^{-2h_B(x)}} = \frac{e^{h_B(x)}}{e^{h_B(x)}+e^{-h_B(x)}} \tag{3.90}$$

Note the resemblance; this justify the use of the softmax to output probabilities from the selected hypothesis:

$$\hat{p}(x;\hat{y}) = \text{softmax}(\hat{y}(x)) \tag{3.91}$$

Exponential loss versus cross-entropy. Adaboost is similar in many regards to a logistic regression over the space span by the basis functions $h_1, \dots, h_t$. It only differs on two points. Firstly, the basis function are learned and not given a priori, which leads to the stagewise optimization. Secondly, Adaboost relies on the exponential loss rather than the cross-entropy. Interpreting h as logits, the cross-entropy for a pair (x, y) is

$$\ell_{CE}(y, \hat{p}(x;\hat{y})) = y\log\hat{p}(x;\hat{y}) + (1-y)\log\left(1-\hat{p}(x;\hat{y})\right) \tag{3.92}$$

$$= \log\left(1+e^{-2y\hat{y}(x)}\right) \tag{3.93}$$

The exponential loss $e^{-y\hat{y}(x)}$ and the cross-entropy are represented at Figure 3.12. They both operate over $y\hat{y}(x)$—the margin—and encourage the boundary to be on the appropriate side and as far away from x as possible. For a point x close to the boundary or on the correct side they appear quite similar. Misclassified points by a large margin, are much more penalized by

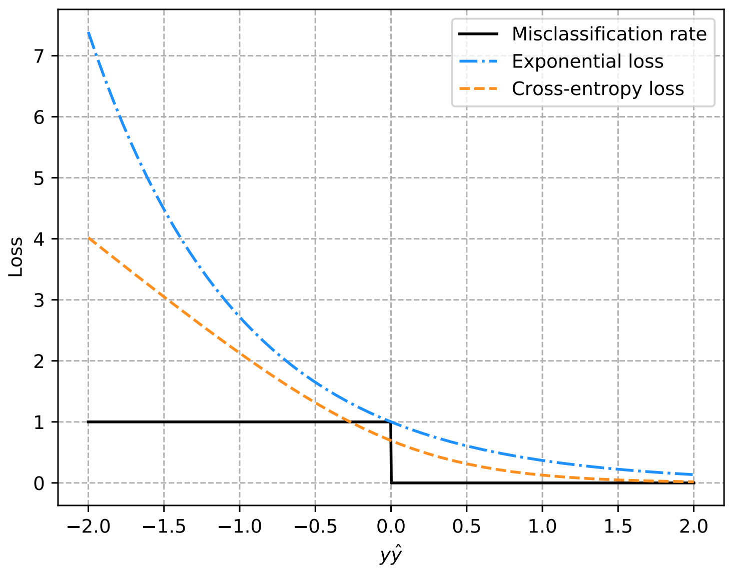


FIGURE 3.12: A comparison of losses with respect to the margin. Ground truth is encoded as $y = \pm 1$. Model outputs $\hat{y}$ is real-valued. Adapted from Friedman, Hastie, and Tibshirani (2001a)

the exponential loss than by the cross-entropy, which appears almost linear in that setting.

Having an exponentially growing penalty means that misclassified points close to the boundary will be completely overlooked so long as there is one misclassified point far away from it. This might prevent any learning when the amount of noise is high and such points are natural occurrences.

Despite leading to the same solution yet being less robust than the cross-entropy, the exponential loss finds its uses because it leads to closed-form optimization. When solving the optimization program once, the exponential loss might not be advisable. When solving it repeatedly as part of a, say, stagewise process, the burden of solving several cross-entropy-based optimizations might not be worth the robustness it brings, hence justifying the exponential loss.

Adaboost as instance reweighing. Historically, Adaboost was not proposed as an instance of forward stagewise additive modeling but rather as an algorithm reweighing the learning instances. Most algorithms offer the possibility to give more or less weight to each (x_i, y_i) pair. For global optimization problems, this is done by multiplying the corresponding loss term by α_i. In the case of decision trees, this is done by multiplying by α_i the uncertainty relating to the ith learning point.

To make the connection with instance reweighing more concrete, consider the following program corresponding to stage t:

$$\min_{(w,\hat{y})\in\mathbb{R}\times\mathbb{H}} \sum_{i=1}^{n} e^{-y_i(\hat{y}_{:t}(x_i)+w\hat{y}(x_i))} \tag{3.94}$$

$$= \min_{(w,\hat{y})\in\mathbb{R}\times\mathbb{H}} \sum_{i=1}^{n} \alpha_i^{(t)} e^{-y_i w\hat{y}(x_i)} \tag{3.95}$$

where $\alpha_i^{(t)} = e^{-y_i\hat{y}_{:t}(x_i)}$ is the weight of the ith learning instance at stage t and $\hat{y}_{:t}(x) = \sum_{l=1}^{m} w_l\hat{y}_l(x)$ is the stagewise model at t. Solving the program is done in two phases. Firstly, $\hat{y}$ is chosen to minimize the weighted error

$$err^{(t)} = \frac{\sum_{i=1}^{n}\alpha_i^{(t)}\mathbb{I}(y_i \neq \hat{y}(x_i))}{\sum_{i=1}^{n}\alpha_i} \tag{3.96}$$

This is done by learning a model according to the $\alpha_i^{(t)}$ weights. Then w_t is optimized to solve the program, leading to new weights

$$\alpha_i^{(t+1)} = \alpha_i^{(t)} e^{-y_i w\hat{y}(x_i)} \tag{3.97}$$

Multi-class Adaboost. An extension to the multi-class setting was proposed by Zhu et al. (2009) relying on a generalization of the exponential loss to more than two classes. For multi-class Adaboost, the labels must be encoded in the following manner.

Label encoding: multi-class Adaboost

A datapoint belonging to class k ($k = 1, \ldots, K$) is encoded as

$$y^{(j)} = \begin{cases} 1, & \text{if } j = k \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases} \tag{3.98}$$

Under this encoding, which generalizes the binary case, the exponential loss can also be generalized.

Definition 3.5.2 (Exponential loss (multi-class))**.**

$$\ell_{\text{exp}}(y,\hat{y}) = e^{-\frac{1}{K}\sum_{k=1}^{K} y^{(k)}\hat{y}^{(k)}} \tag{3.99}$$

where y and $\hat{y}$ both follow the multi-class encoding.

The authors showed the following result.

Proposition 3.5.1. *The program*

$$\begin{aligned}\arg\min_{\hat{y}} \; \mathbb{E}_{\mathcal{Y}|x} \exp\left(-\frac{1}{K}\sum_{k=1}^{K} y^{(k)}\hat{y}^{(k)}(x)\right) \\ s.t. \sum_{k=1}^{K} h^{(k)}(x) = 0 \end{aligned} \tag{3.100}$$

has for solution

$$h_B(x)^{(k)} = (K-1)\left(\log \mathbb{P}(\mathcal{Y}=k|x) - \frac{1}{K}\sum_{l=1}^{K}\log \mathbb{P}(\mathcal{Y}=l|x)\right) \tag{3.101}$$

Taking advantage of the zero-sum constraint, this implies

$$\mathbb{P}(\mathcal{Y}=k|x) = \frac{e^{\frac{1}{K-1}h_B^{(k)}(x)}}{\sum_{l=1}^{K} e^{\frac{1}{K-1}h_B^{(l)}(x)}} = \overset{(k)}{\text{softmax}}\left(\frac{1}{K-1}h_B(x)\right) \tag{3.102}$$

Given a new term satisfying Eq. 3.101, new instance weights can be derived following Eq. 3.96 and the general Adaboost procedure. It is easy to check that everything falls back to the binary exponential loss when $K=2$.

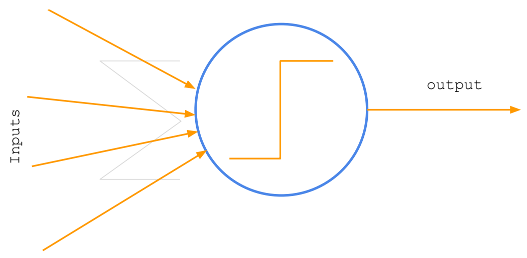
3.6 Deep learning

Deep learning, multi-layer perceptron (MLP), (artificial) neural network (ANN), feed-forward network, in this section we go over this class of models, what they are, how they can make predictions and how they can be learned from data.

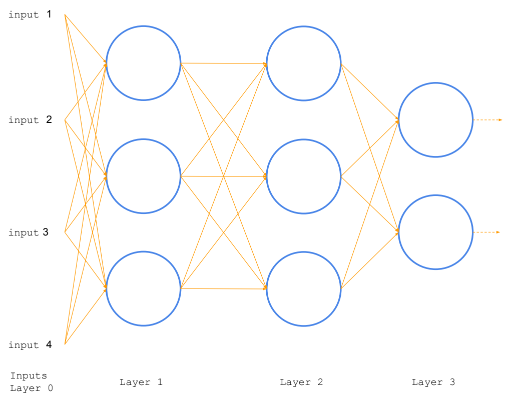
3.6.1 Structure and inference

Figure 3.13a depicts a neuron. Mathematically, a neuron is very close to the logistic regression: inputs travel along axons to reach the core; under certain conditions, the neuron fires a signal which can be transmitted to other neurons via the synapses. Typically, this is implemented by combining the inputs linearly—the weights simulating the strength of neural connectivity—and running this combination into an activation function, historically a continuous equivalent of a sign function.

Artificial neurons transcend the logistic regression by being connected together, forming an artificial neural network. Many graph topologies have been studied but the most common ones are feed-forward networks. In such networks, information flows only in one direction (at least at inference time), owing to the directed acyclic graph nature of the network. Neurons are usually organized into layers, with connection only (or mostly) between two successive layers. When, for all layers, all neurons are connected to all neurons



(A) Schematic representation of a neuron.



(B) Start of a feed-forward, fully-connected neural network.

FIGURE 3.13: An artificial neuron and a network of such neurons.

of the following layer, the network is termed fully-connected. An example of a fully-connected, feed-forward network is given in Figure 3.13b.

Historically, the first neural model, containing a single neuron, was proposed by Rosenblatt (1958) under the name perceptron. Therefore, feed-forward networks are also sometimes referred to as multi-layer perceptrons.

There is no need *a priori* for the activation functions to be the same in all neurons, nor is it mandatory to have linear combinations.

Moving away from the biological analogy, a feed-forward neural network can be formalized in the following fashion.

Definition 3.6.1 (Feed-forward neural network (general case))**.** *A feed-forward neural network of L layers is a function of the form*

$$\hat{y}(\cdot,\Theta) = f_L(\cdot;\theta_L) \circ \ldots \circ f_1(\cdot;\theta_1) : \mathbb{X} \to \mathbb{R}^K \tag{3.103}$$

with $\Theta = [\theta_1, \ldots, \theta_L]$ the set of all trainable network parameters.

Note that the structure is fixed at learning time, therefore, the hypothesis space $\mathbb{H}_\Theta$ is the set of all networks of a given structure. The structure can be assimilated to a hyper-parameter. When repeating patterns of layers occur in a network, it is not uncommon to see them as indivisible blocks.

Furthermore, let $z_l(x;\Theta_l)$ be the feature vector of layer l ($1 \leq l \leq L$) for an input x:

$$z_l(x;\Theta_l) = (f_l(\cdot;\theta_l) \circ \ldots \circ f_1(\cdot;\theta_1))\,(x), \tag{3.104}$$

with $\Theta_l = [\theta_1, \ldots, \theta_l]$ the parameters of the first l layers. This definition entails $z_L(x;\Theta_L) = \hat{y}(x,\Theta)$. For convenience, let us extend the definition so that $z_0(x) = x$.

With this view of a neural network, inference is straightforward as a model is just a composition of parametrized functions. In the simplest case, this correspond to a linear function followed by an non-linear activation $s(\cdot)$:

$$z_l = s\left(Wz_{l-1} + b\right) \tag{3.105}$$

In regression, we can use the neural network thus defined directly as hypothesis. In classification, an ultimate softmax layer is usually appended to the network to derive probability vectors:

$$\hat{p}(x;\Theta) = \text{softmax}\left(\hat{y}(x;\Theta)\right) \tag{3.106}$$

From there, a hard classification can be made by choosing the most probable class, as usual. Note that these last steps do not contain any learnable parameters.

Note that, for hard classification, the softmax layer is only really needed during training. For inference, the network can be cut at the logits. Classification proceeds by putting forth the class whose logit is the highest.

Building blocks. Designing architectures and tailoring them to a particular problem is a challenging task. Fortunately, the community has come to provide standard building blocks as well as dedicated architectures for several types of tasks. In this thesis, we will mainly focus on two tasks: unstructured problems (Chapter 9) and image classification (Chapters 7 and 8). In the former case, we will use fully-connected networks: trainable parameters will be the coefficients of the linear maps from one layer to the next. The specificities of images classification will be discussed in Section 3.6.3.

3.6.2 Learning: the backpropagation algorithm

In regression, formulating the learning program is straightforward.

Definition 3.6.2 (Neural network program (regression)). *Under the squared loss, the neural network optimization is*

$$\min_{\Theta} \sum_{i=1}^{n} \ell_2\left(\hat{y}(x_i, \Theta), y_i\right) + \mu \sum_{l=1}^{L} ||\theta_l||_2^2 \tag{3.107}$$

$$\min_{\Theta} \sum_{i=1}^{n} \left(y_i - \hat{y}(x_i, \Theta)\right)^2 + \mu \sum_{l=1}^{L} ||\theta_l||_2^2 \tag{3.108}$$

In classification, neural networks are learned via the cross-entropy loss. This requires the following label encoding.

Label encoding: neural networks

For the K-class neural network, the labels are encoded as one-hot vectors such that:

$$y^{(j)} = \begin{cases} 1, & \text{if } j = k \\ 0, & \text{otherwise} \end{cases} \tag{3.109}$$

Definition 3.6.3 (Neural network program (classification))**.** *Under the cross-entropy loss, the neural network optimization is*

$$\min_{\Theta} \sum_{i=1}^{n} \ell_{CE}\left(\hat{p}(x_i;\Theta), y_i\right) + \mu \sum_{l=1}^{L} ||\theta_l||_2^2 \tag{3.110}$$

$$\min_{\Theta} \sum_{i=1}^{n} \sum_{j=1}^{K} y_i^{(j)} \log \hat{p}(x_i;\Theta) + \mu \sum_{l=1}^{L} ||\theta_l||_2^2 \tag{3.111}$$

In both classification and regression, the regularization term—called the weight decay in this context—is usually accompanied with a small penalty coefficient $\mu = 5 \times 10^{-4}$.

In the following, we focus on the more involved classification case.

Neural networks are most usually trained via a first-order gradient descent with the loss estimated on mini-batches of the training data. The most simple instance of this scheme is the stochastic gradient (SGD) introduced in Algorithm 1 (Section 3.2). Contrary to logistic regression, we need to fit the parameters of every layer. Since the network is a composition of functions, the necessary gradient can be computed via the chain rule, a process also known as *backpropagation*.

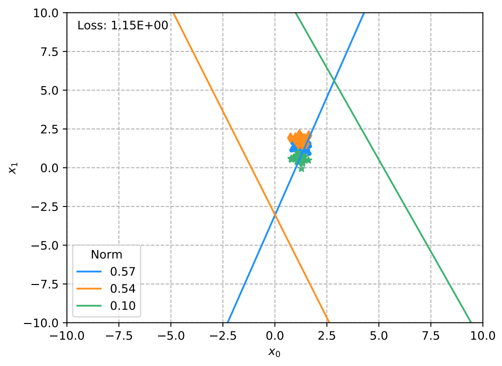
In full generality, the component of the gradient of layer l $(1 \leq l \leq L)$ relating to the pair (x, y) is given by

$$\begin{aligned} \nabla_{\theta_l} \ell_{CE}\left(\hat{p}(x;\Theta), y\right) &= \left(\prod_{k=l}^{L} J_k^T(x)\right) \nabla_{z_L} \ell_{CE}\left(\hat{p}(x;\Theta), y\right) && (3.112) \\ &= \left(\prod_{k=l}^{L} J_k^T(x)\right) \left(\hat{p}(x;\Theta) - y\right) && (3.113) \\ &= J_{k:L}^T(x) \Delta_y \hat{p}(x) && (3.114) \end{aligned}$$

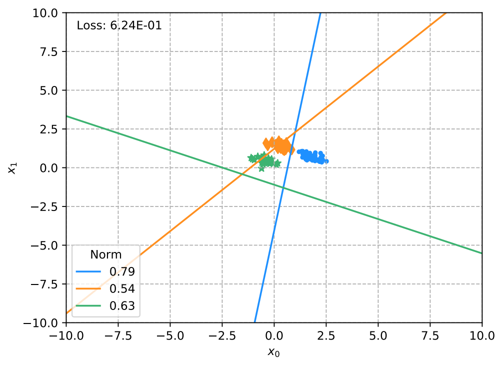
where J_k is the Jacobian of the kth layer. Note that in the multi-class setting $\hat{p}$ and y refer to vectors (with a one-hot encoding for y).

Thus the backpropagation consists of a product of Jacobians multiplied by the error/learning signal $\Delta\hat{p}(x)$. The following two sections will explore issues linked to backpropagation which have held back deep learning for some time, namely the vanishing gradient (Section 3.6.2.2) and the covariate shift (Section 3.6.2.3) problems.

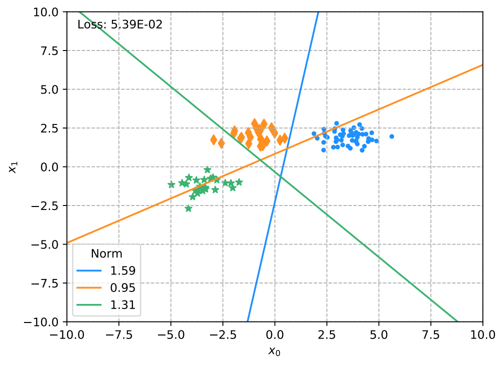
3.6.2.1 First-order interpretation: from softmax regression to neural networks



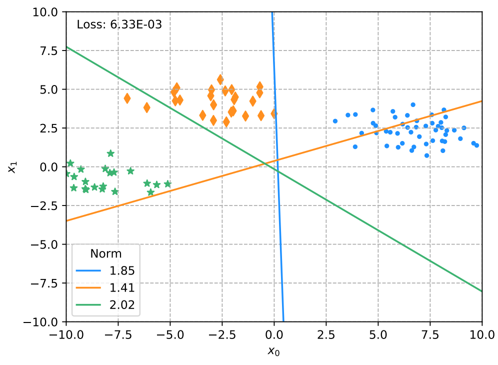
(A) Random initialization. The classes are not linearly separable.



(B) After 10 iterations. The classes start being linearly separable even though the classification hyper-planes are not yet perfect.



(C) After 15 iterations. The problem is solved: classes are articulated around the origin of the latent space, are linearly separable and the hyper-planes are correctly placed.



(D) At convergence. The further loss reduction is due to pushing the classes further away, resulting in a more confident network. The increase in the hyper-plane-defining vectors is relatively mild compared to how much the latent vectors have been pushed from the origin.

FIGURE 3.14: Evolution of the feature learning process for the latent space and its classification hyper-planes on a toy problem at different stages of learning. The problem consists of three 3D Gaussians, co-linear in the two first dimensions and more easily separable in the third (to simulate how learning features might provide a better space for separability). The network is composed of two fully-connected feature extraction layers initialized as the identity over the two first dimensions and ignoring the third, all with null bias and ReLU activations. The network is trained with a small weight decay of 5×10^{-4}.

Most neural networks in classification do not have an activation function just before the softmax, where a linear layer transforms the latent vectors into logits, resulting in the following structure:

$$\hat{p}(\cdot, \Theta) = \underbrace{\text{softmax}(\cdot) \circ f_L(\cdot; [W, b])}_{\text{softmax classifier}} \circ \underbrace{f_{L-1}(\cdot; \theta_{L-1}) \circ \ldots \circ f_1(\cdot; \theta_1)}_{\text{feature extractor}} \tag{3.115}$$

The feature extractor is also called a representation learner. This dichotomic view of the network structure highlights the fact that optimizing the model amounts to jointly learning a softmax classifier and an ideal feature representation for this classifier. So what would be ideal from the point of view of the softmax?

The first step in reducing the loss is to avoid misclassification. With a softmax classifier, this encourages the representation learning to place the classes in linearly separable sub-regions, such as in the 3G problem (Section 3.2.2). When the dimensionality of the pre-logit, latent space is larger than the number of classes (the most common situation in practice), the network has the flexibility to spread the classes in different sub-spaces. This is further encouraged when weight decay is applied, since using the whole dimensionality is less penalizing than requiring larger coefficients for the hyper-planes vectors—either of which is needed to lower the loss the most.

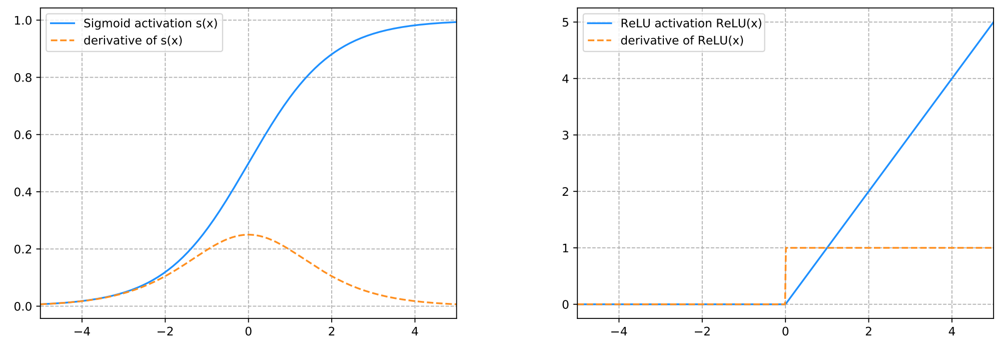
Once the classes are linearly separable, or alternatively, once the hyper-planes are set (*i.e.* rotated and translated) to provide welcoming regions, aligning the latent vectors $z_L(x)$ with the class-corresponding hyper-planes is the optimal way to boost the network confidence (thus lowering the error signal $\Delta\hat{p}$) for a fixed norm $\|z_L(x)\|$.

The next step is to further boost confidence. This can either be done by increasing the norm $\|z_L(x)\|$ of the latent vector $z_L(x)$, driving them further away from the center, or by directly increasing the norm of W. The former offers more flexibility, such as pushing away points which align less well with the hyper-planes. On the other hand, increasing the norm W affects all points at once.

Adding some weight decay will more directly limit the growth of W and will limit, to a small extent, the confidence of the network by moving away from the sole equilibrium over the error signal.

This whole process is illustrated by Figure 3.14.

This geometric interpretation of feature learning may provide some insight into why slow-down regularization is helpful in the case of neural networks. If the network quickly goes into separating roughly the classes and then increases the norm of each hyper-plane it will quickly get a low resubstitution error. In the event where the separation is perfect on the training set but not as adequate in general, the model will have overfitted the training set. By slowing down learning, the latent space gets more time to optimize its regions, ending up with a better generalization.



(A) A sigmoid activation function and its derivative. Note how the peak of the derivative is much less than one and how it decreases towards zero.

(B) A ReLU (Nair and Hinton, 2010) activation function and its derivative. Note how the derivative does not decrease for positive inputs.

FIGURE 3.15: A comparison of the behavior of two activation functions (sigmoid on the left, ReLU on the right). Note the difference in scale of the y-axis. The derivative of the sigmoid will dampen the learning signal, whereas the ReLU will backpropagate all the information relating to positive inputs.

3.6.2.2 Vanishing gradient

For brevity, let us denote by $\nabla_{\theta_l} \ell$ the loss gradient and $\Delta\hat{p}$ the learning signal. A frequent problem in learning deep networks (*i.e.* neural networks with many layers) is that, as it gets propagated, the error signal decreases in magnitude to the point of little information reaching the early layers:

$$\left\|\nabla_{\theta_l} \ell\right\| \ll \|\Delta\hat{p}\| \tag{3.116}$$

Considering how the former is computed, this dampening effect is due to the product of Jacobians. Note that the opposite effect—exploding gradient—is sometimes observed, although more rarely since the optimization forces the gradient towards zero, which can happen if the error signal is null (the goal actually pursued) or because the gradient vanishes along the layers.

Unbounded activation functions. Figure 3.15 depicts two activation functions together with their derivative. Note how the sigmoid derivative peaks below 0.3 and decreases away from a null input. Applied to all values in the Jacobian matrices, this will slowly decrease the magnitude of the learning signal. Using other logistic curves (which peak at 1 or higher) can only partially solve the problem; the shape of the derivative still encourages the dampening effect. On the other hand, peaking too high would favor exploding gradient situations.

The second activation function is called a rectified linear unit (Nair and Hinton, 2010), or ReLU for short. The basic variant is of the form

$$ReLU(x) = \begin{cases} x \text{ if } x \geq 0, \\ 0 \text{ otherwise.} \end{cases} \tag{3.117}$$

Thus negative activations get turned off. In a fully-connected, feed-forward network the layers are functions of the overall form[1]

$$z_l = ReLU\left(Wz_{l-1} + b\right) \tag{3.118}$$

By being unbounded, The ReLU function (and its variants) solves (partially) the dampening effect by avoiding the otherwise necessary slow down in the activation. Therefore, the decrease of magnitude in the propagated error signal due to the negative inputs can be compensated by higher-value positive inputs. In practice, ReLU offers a strong first step in avoiding the vanishing gradient problem.

Note that the derivative of the ReLU is technically not defined at zero. In practice, however, since our goal is to optimize the network, any subgradient can be chosen at this point, would it chance to happen.

Interestingly, ReLU captures the idea of neurons firing at a much coarser level, moving away from the biological inspiration. However, the non-linearity they provide and the adaptability of (deep) artificial networks seem to suffice to learn good models.

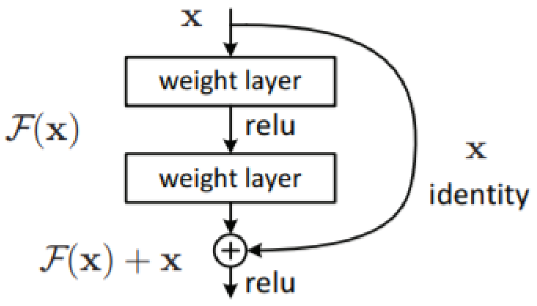
Skip connections. Using ReLU allowed to explore networks deeper than a couple of layers. Training networks with tens of layers remained challenging, and hundreds virtually impossible, however. Introducing skip connections solved this issue. Skip connections come in many flavors but all share the idea of information running in parallel in the network, with one way bypassing several layers. This skip allows for a pathway where the learning signal can travel without being amortized. Figure 3.16 illustrates two types of skip connections.

3.6.2.3 Covariate shift

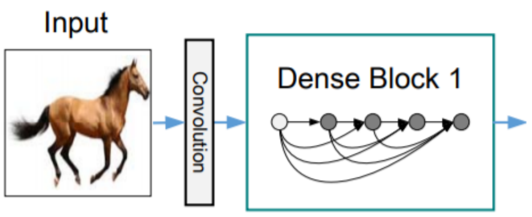
Another problem when learning networks is that changing the weights of layers $1 \leq k \leq l$ changes the distribution of inputs for layer l—a phenomenon known as covariate shift. When updating all layers at once, this means that an update of previous layers can render the update of a latter layer obsolete or inconsistent.

One workaround is to train the network layer by layer, updating only one layer at a time. This, however, has been recognized as inefficient and a more traditional solution nowadays is to use batch-normalization layers (Ioffe and

[1]Activation functions are usually defined over scalars and tacitly extended to matrices/tensors as element-wise operators.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(A) Residual connection (He et al., 2016): the value of x is added to the learned $\mathcal{F}$ function.



(B) Densely-connected blocks(Huang et al., 2017): all the features of the previous layers within a block are concatenated and fed to subsequent nodes.

FIGURE 3.16: Two examples of skip connections.

Szegedy, 2015). We will look at how those layers are implemented more specifically in the case of image classification (in Section 3.6.3.4).

3.6.2.4 Miscellaneous

Initialization. The parameter initialization in iterative algorithms is also an important question. Most notably, a bad initialization in deep learning might render the model hard to train with bad conditioning due to symmetry or bad gradient flow.

Higher order methods. As mentioned, there exists a higher-order method for solving the logistic regression and exposing the first order, gradient method was mainly a pedagogical choice. Higher-order methods are not advocated for deep learning, however (Bottou and Bousquet, 2007).

Instead, the community has proposed other strategies to speed up the learning, such as improvement on SGD (such as Nesterov momentum, individualized learning rate, and so on; see (Duchi, Hazan, and Singer, 2011; Kingma and Ba, 2015, e.g.)) and scheduling: the idea that the learning rate can be divided by a factor at some points during the learning, ideally when the training stagnates.

Non-convex optimization. Finally, it is important to note that the objective functions to optimize in deep learning are usually non-convex. Therefore, there is a chance to end up in a local minimum (or close to one) which is far from the actual global minimum.

3.6.3 Image classification

Dedicated layers have been presented over the years which both serve to decrease the number of learnable parameters and to better exploit the structure of the inputs. Indeed, images are known to be highly spatially correlated,

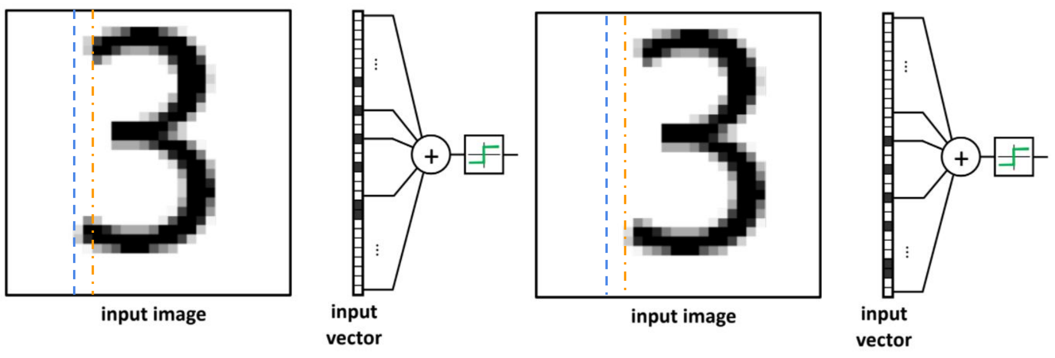


FIGURE 3.17: The semantic content of an image does not change with a translation. Left: original image and how it would be encoded as a one-dimensional vector to be fed in the network. Right: the same with a slight horizontal translation. Notice how the encoding is different, losing the spatial information. From Bronstein et al. (2017).

contain structures, provide information at several scales and somewhat spatially stationary (Figure 3.17). By designing layers to leverage these, the community has been able to provide relevant inductive biases to the hypothesis spaces of neural networks for image classification.

3.6.3.1 Tensors and feature maps

The major specificity when working with images in the context of neural networks is to keep the spatial structure of the data. To do so, the standard view of vector spaces will be generalized, at the very least conceptually.

The input space of the network is $\mathbb{X} = \mathbb{R}^{C \times H \times W}$ where C is the number of channels, H is the height and W is the width of the images. Thus $x^{(c,h,w))}$ is the value of the pixel of the cth channel ($1 \leq c \leq C$), at position (h, w) ($1 \leq h \leq H, 1 \leq w \leq W$).

The layer functions $z_l(\cdot; \Theta_l)$ take as inputs tensors—called feature maps—from $\mathbb{R}^{C_l \times H_l \times W_l}$ and outputs feature maps in $\mathbb{R}^{C_{l+1} \times H_{l+1} \times W_{l+1}}$. The functions are such that the spatial structure is maintained (see Section 3.6.3.2 for an example).

The feature extraction phase usually flattens the feature maps of the last latent space in the form of a vector so as to feed a softmax classifier. On rarer (or older) occasions, the feature maps are flattened upstream and traditional layers (such as rectified fully-connected) are appended before the classifier. See Figure 3.20 for a schematic of a modern network for image classification.

3.6.3.2 Convolution

Convolutions are such a fundamental basic block for image-based neural networks that these are usually termed Convolutional neural networks (CNN). Convolution aims at providing some spatial invariance by imposing some restrictions over the fully-connected layers. They work by convoluting a

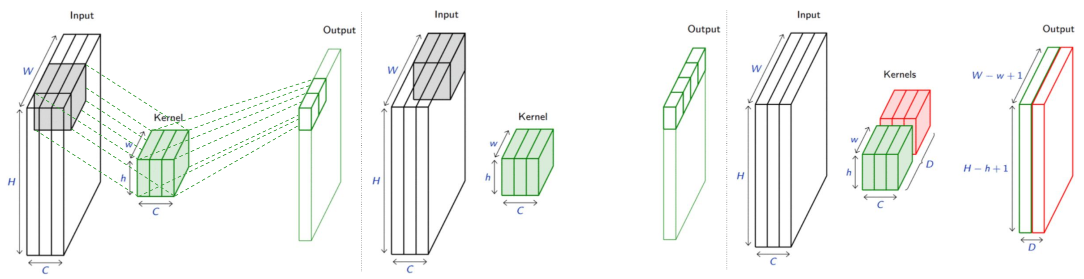


FIGURE 3.18: An illustration of convolution in CNN. Left: the output feature map value at a given position is computed by applying the kernel on the input feature map. Center: the kernel has moved at the value is computed at another location with the same kernel but a different part of the input feature map. Right: another channel of the output feature map is computed thanks to a different kernel. Adapted from Fleuret (2021).

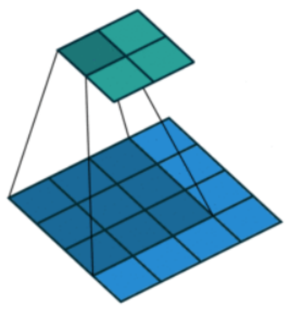
linear kernel which is moved over the image. Let $z_l \in \mathbb{R}^{(C_l, H_l, W_l)}, z_{l+1} \in \mathbb{R}^{(C_{l+1}, H_{l+1}, W_{l+1})}$ be the input and output feature maps of a convolution layer f_l. Assuming squared-base kernels K of size $C_l \times k \times k$ (k being odd) and a stride of 1 with no padding (see *infra*) The input-output relationship between them can be described as

$$z_{l+1}^{(c_o, h_o, w_o)} = b_{l,c_o} + \sum_{c=1}^{C_l} \sum_{i=1}^{k} \sum_{j=1}^{k} z_l^{(c, h_o+i-1, w_o+j-1)} \times K_{l,c_o}^{(c,i,j)} \tag{3.119}$$

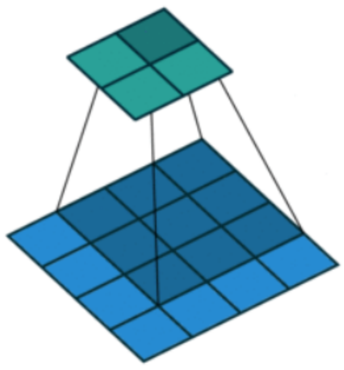
for $1 \leq c_o \leq C_{l+1}$, $1 \leq h_o \leq H_{l+1} - (k-1)$ and $1 \leq w_o \leq W_{l+1} - (k-1)$.

Kernels are usually visualized as moving across the input feature maps, centering on a location and computing the local linear function (see Figure 3.18). The above formula shows that the output feature map is smaller by a constant factor (depending on the kernel size) along each spatial dimension. Padding can be used to change this constant (at a fixed kernel size). With a padding of $\lfloor k/2 \rfloor$, the kernel is (spatially) centered at the same location on the input and output feature maps. This implies that, while moving, the kernel might partially overflow the input feature map. Several strategies exist to cope with this situation (such as considering a constant value outside the feature map).

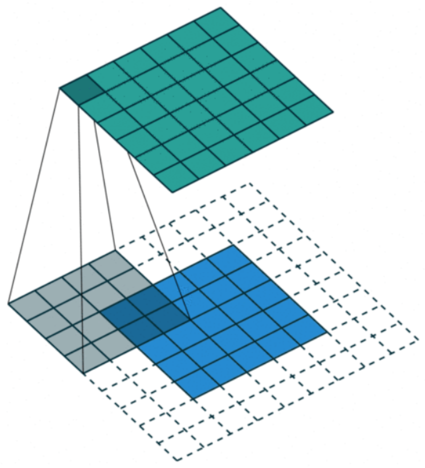
Rather than reducing the spatial dimension by a constant factor, strides can be used to divide the spatial dimension. The most frequent case is to use a stride of 2: as the kernel moves, it sidesteps every other location, resulting in an output feature map roughly twice as small. The effect of the stride and padding are illustrated in Figure 3.19.



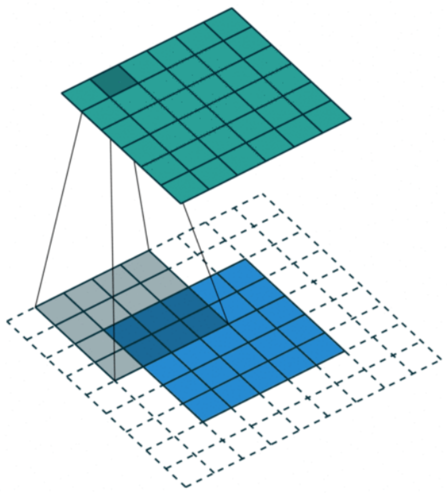
(A) 3×3 convolution with a stride of 1 and no padding (location 1).



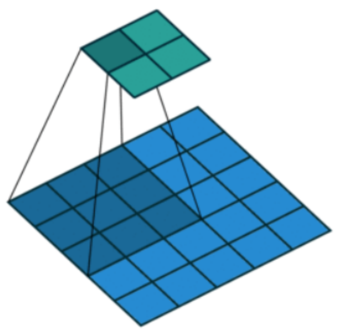
(B) 3×3 convolution with a stride of 1 and no padding (location 2).



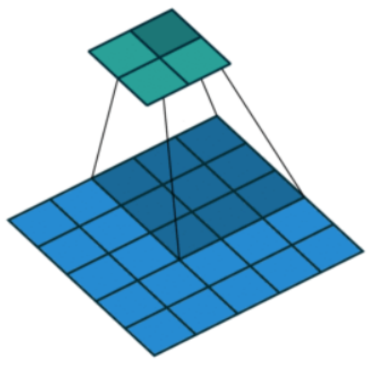
(C) 5×5 convolution with a stride of 1 and a padding of 2 (location 1).



(D) 5×5 convolution with a stride of 1 and a padding of 2 (location 2).



(E) 3×3 convolution with a stride of 2 and no padding (location 1).



(F) 3×3 convolution with a stride of 2 and no padding (location 2).

FIGURE 3.19: Effects of padding and stride in convolution (illustrated for a single channel). The blue image (at the bottom) represents the input feature map and the green (top) one the output feature map. Note how the feature map is reduced by or not depending on the parameters. From Dumoulin and Visin (2016).

3.6.3.3 Pooling

Pooling can be seen as a non-linear generalization of convolution, where an arbitrary function is applied to the moving window. Pooling layers using the maximum function (max-pooling layers) are often used with a stride greater than one. The strides serve to reduce the feature map spatial dimensions and the maximum was motivated by spatial invariance, overall producing a coarser representation of the input feature map.

Nowadays, max-pooling layers tend to be less used as they force the gradient to flow only through one component of the kernel. Traditional convolution layers with strides greater than one are now used instead. There is however a pooling operation which has become the norm: global average pooling (Lin, Chen, and Yan, 2013). This simply consists in outputting the spatial mean of a feature map. In modern architectures, this is how the network changes from the feature map representation used for feature extraction to the vector representation which is used for the classification.

Whether pooling is used together with convolutions or not, they both stem from the idea of producing spatial invariance and, when used in cascade, scale invariance as well.

3.6.3.4 Batch-normalization

As discussed in Section 3.6.2.3 changing the parameters of layer l will render the changes of layer $l+1$ obsolete, as its input distribution changes as well. Batch normalization layers (Ioffe and Szegedy, 2015) try to circumvent this phenomenon. This layer operates in two steps. First, it standardizes the input batch with respect to some estimated statistics (Eq. 3.120). Then it applies a linear transformation (Eq. 3.121). Let B be the set of layer indices corresponding to the batchnorm layers. Then for all $l \in B$,

$$y_l^{(c,w,h)} = \frac{z_{l-1}^{(c,w,h)} - \mu_l^{(c)}}{\sqrt{\left(\left[\sigma_l^{(c)}\right]^2 + \epsilon\right)}} \quad \forall c, w, h \tag{3.120}$$

$$z_l^{(c,w,h)} = y_l^{(c,w,h)} \times \gamma_l^{(c)} + \beta_l^{(c)} \quad \forall c, w, h \tag{3.121}$$

with C_l ($1 \leq c \leq C_l$), W_l ($1 \leq c \leq W_l$) and H_l ($1 \leq c \leq H_l$), standing respectively for the number of channels, the width and the height of the input feature maps at layer l.

The two stages follow somewhat antagonist goals. The standardization forces the input distribution of the $l+1$ layers to resemble the one before updating the previous $1, \ldots, l$ layers. On the other hand, the linear output transformation allows to deviate from that goal to ensure updates are actually accomplishing something.

Batch-normalization layers are not specific to image processing but the formulation holds some particularities. Note how only two parameters ($\mu_l^{(c)}$

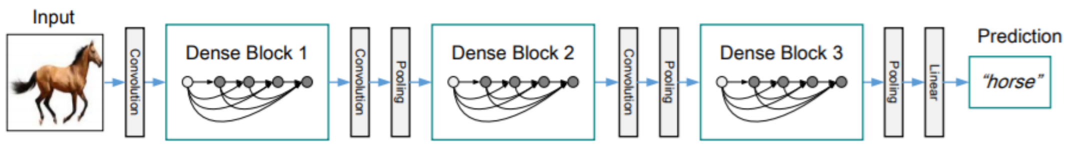


FIGURE 3.20: A schematic view of a DenseNet. The input image is fed to a first convolution layer. The feature maps then run through several dense blocks (containing batch normalization, convolutions and ReLU activations) followed by further convolutions and a max-pooling layer which reduces the spatial dimension of the feature maps. The last pooling layer consists in a average pooling to deliver a feature vector for the softmax classifier (embodied by the linear layer). From Huang et al. (2017).

and $\sigma_l^{(c)}$) are estimated per channel for the normalization. The authors justify this choice by symmetry with the convolution layers: "For convolutional layers, we additionally want the normalization to obey the convolutional property – so that different elements of the same feature map, at different locations, are normalized in the same way" (Ioffe and Szegedy, 2015).

This view of all locations being somehow from the same distribution dictates to estimate the *total* variance, in contrast, for instance, to averaging over all locations the variance across each instance (*within* variance where groups are the locations). The estimation of all the necessary quantities is done via a running average and $0 < \epsilon \ll 1$ is added in the standardization for numerical stability.

Figure 3.20 illustrates a modern architecture.

3.7 Tools and tricks

To conclude this tour of supervised learning algorithms, let us discuss some important tools and tricks.

3.7.1 Risk management

Class balancing. In many classification problems, some classes may be more represented than others in the learning set. This might be due to the phenomenon, where some classes naturally appear more frequently than others (e.g. more healthy people than ill ones), or to the data collection process (bias, difficulty to record some classes, etc.). Depending on the context, this imbalance might pose some issues. Methods have been proposed to deal with this (Guo et al., 2017b, see, for instance).

Risk assessment: symmetric losses under asymmetric issues. Sometimes all misclassifications are equivalently troublesome, and sometimes missing or confusing some classes are more problematic than others. For instance, missing some signs of cancer is a far greater problem than diagnosing one

TABLE 3.1: Confusion matrix for a binary problem. P stands for positive, N for negative, TP for true positive, FN for false negative, FP for false positive, TN for true negative. Confusion matrices are contingency table crossing the number of truly positive and negative samples against how a given method classes them.

	Prediction		
Actual class	Positive	Negative	Total
Positive	TP	FN	P
Negative	FP	TN	N

when there is not (in which case latter examination will expose the false positive). Likewise, it is better to catch a suspicious bank transaction and ask for a more secure validation rather than miss a fraud.

Even if both types of errors are equally penalizing, relying on, say, the accuracy in contexts where the classes themselves are *imbalanced* might obfuscate more than it reveals. For instance, in a binary classification between the "positive" and "negative" samples, if the positive class represents $x\%$ (e.g. 1%) of the data a model will be exposed to, constantly predicting the negative class will result in an accuracy of $x\%$. As is evident, the accuracy might appear good while the method is actually worthless.

Yet another problem with accuracy arises with some classifiers relying on recognizing some patterns in data and might miss them sometimes (*i.e.* produce a false negative) but rarely detect them when there is not (*i.e.* produce a false positive), or the other way around.

So far, all the losses we have encountered in classification have been symmetrical: confusing class i for class j bore the same weight for all $i, j (i \neq j)$. They are, consequently, not well suited to deal with those situations. Other metrics are needed to better assess the model under those circumstances.

Risk assessment: the confusion matrix. The simplest way to manage asymmetry is to avoid aggregating the information in a single number and use a confusion matrix instead. A confusion matrix CM is a matrix such that $CM_{i,j}$ is the number of examples from class i which have been classified as j. Table 3.1 shows a confusion matrix for a binary problem, with one class labeled as positive and the other as negative. This case is so important in practice that each entry in the matrix has a dedicated name: True Positive for positive samples which have been recognized as such, False Negative for positive samples which have been missed, False Positive for negative samples which have mistakenly been predicted as positive and True Negative for correctly recognized negative samples. The elements off the diagonal are called the risks.

The accuracy can be computed from the confusion matrix by summing the diagonal and dividing by the total number of samples. Other interesting metrics can be computed—especially in the case of binary classification.

Definition 3.7.1 (Sensitivity)**.** *Sensitivity, or recall, or true positive rate (TPR) is defined as*

$$\frac{TP}{P} \tag{3.122}$$

It represents the proportion of positive samples which have been correctly identified and indicates how much a model is capable of actually finding the positive samples; a model with high sensitivity is good at detecting positives.

Definition 3.7.2 (Specificity)**.** *Specificity, or true negative rate (TNR), is defined as*

$$\frac{TN}{N} = 1 - \frac{FP}{N} \tag{3.123}$$

It represents the proportion of negative samples which have been correctly identified and indicates how much a model is capable of actually rejecting negative samples; a model with high specificity is good at not being fooled by negative samples.

Definition 3.7.3 (Precision)**.** *Precision is defined as*

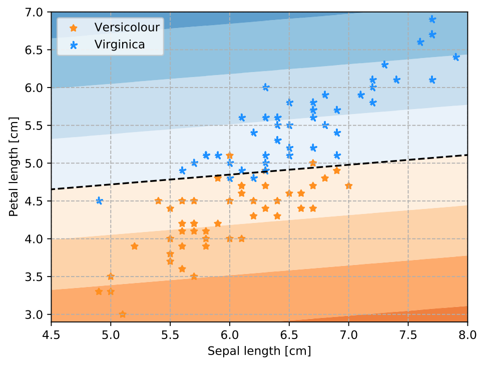
$$\frac{TP}{TP + FP} \tag{3.124}$$

It represents the proportion of truly positive samples among the samples which have been predicted as positive. A model with high precision can be trusted when it predicts the positive class.

Managing the risks is about *balancing* them, rather than blindly optimizing a given metric. It is easy to make up situations where a given metric would fail to expose meaningful information. Predicting always the positive class would yield a sensitivity of 1. Predicting always the negative class would result in a perfect specificity. A situation of class imbalance similar to the one discussed about accuracy would also impact the precision.

Actual risk management. In the presence of asymmetry, be it expressly aimed for or a consequence of the problem's nature, aiming for a good *accuracy* might actually not be the best approach. One possibility is to change the loss function to reflect the asymmetry. Another possibility is to re-calibrate the model afterward. This is typically done by changing the hard classification threshold on the model continuous output. For example, with a classifier giving a probability of belonging to the positive class $\hat{p}$, the Bayes classifier under symmetric risks consists in associating the positive class as soon as $\hat{p} > 0.5$. Instead, the hard classification could be $\hat{p} > 0.75$, this would change some TP to FN and some FP to TN, increasing the specificity of the model while lowering its sensitivity.

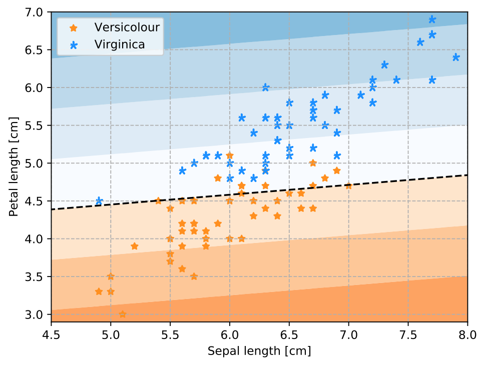
Figure 3.21 illustrates how calibrating the hard classification based on the probability outputted by a logistic regression can affect the risk balance.



(A) Boundary minimizing the risks symmetrically.

	Prediction		
Truth	Pos.	Neg.	Total
P	47	3	50
N	3	47	50

(B) Confusion table for the symmetric risks.



(C) Boundary minimizing the false positive risk.

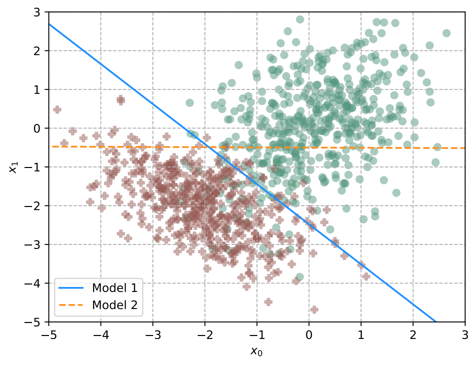
	Prediction		
Truth	Pos.	Neg.	Total
P	39	11	50
N	0	50	50

(D) Confusion table for the false positive risk.

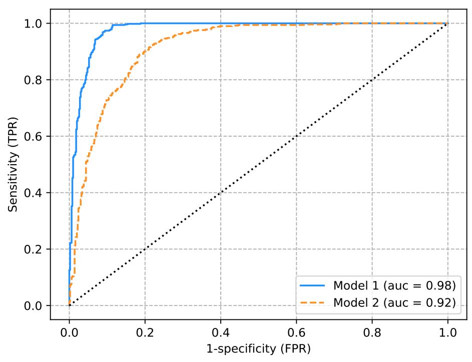
FIGURE 3.21: Two linear models for the iris classification problems (left column) and the corresponding confusion matrices (right column). On top, the model has been trained under symmetric risks (reflected by the symmetries in the corresponding confusion matrix). On the bottom, the decision boundary has been translated to minimize the false positive rate.

Area under the curves. Post-calibration changes the confusion matrix. In other words, the confusion matrix depends on the (quality of the) calibration. Sometimes, it is interesting to evaluate how a model performs independently of the calibration. This conveys an idea of how good the model is in general, rather than when tailored for a specific risk balancing, and consequently indicates how the model can be useful under other risk tradeoffs. It might also reveal that a model is actually good but gifted with a skewed calibration, or conversely that a model seems good with respect to a given risk but something is amiss. Additionally, this offers new—and more robust—ways to summarize all the information contained in a confusion matrix in a single number.

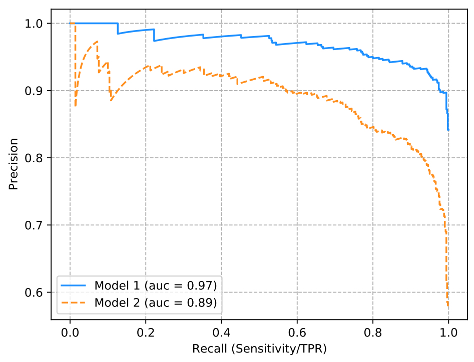
The receiver operating characteristic (ROC) curve consists in varying the calibration to generate all the possible confusion matrices and to plot, for



(A) Two linear models on a binary classification problem with two Gaussians. The first model is more appropriate than the second.



(B) Receiver Operating Characteristic (ROC) curve for the above problem with two linear models. The first model has a higher area under the curve (AUC) than the second.



(C) Precision-Recall (PR) curve for the above problem with two linear models. The first model has a higher area under the curve (AUC) than the second.

FIGURE 3.22: ROC and Precision-recall curves for two linear models on the problem above.

each increasing value of the false positive rate the corresponding true positive rate (sensitivity). Since the measures involved are normalized with the number of positive and negative samples, the ROC curves are not affected by the class (im)balance.

The precision-recall (PR) curve consists in varying the calibration to generate all the possible confusion matrices and to plot for each value increasing value of the recall (sensitivity) the corresponding precision. The precision-recall curve therefore only looks at the samples predicted as positive.

In both cases, the area under the curves (AUC) can be used as a summary value ($0 \leq \text{auc} \leq 1$) for the appropriateness of the model. Figure 3.22 illustrates the ROC and PR plots and their area under the curve.

3.7.2 Miscellaneous

Computational considerations. It is important to note that for efficiency, memory or numerical reasons and tradeoffs, the algorithm implementations might not follow rigorously the definition given in this chapter. Moreover, the success of deep learning is due in part to the advent of better hardware, namely graphical processing units (GPUs), without which the training of deep neural networks would be unbearably long for most.

Languages and libraries. The success of machine learning as a whole is due to the availability of many great libraries, such as Scikit-learn (Pedregosa et al., 2011) for machine learning in general and a others dedicated to deep learning. These especially require to handle tensor, automatic differentiation, and nowadays GPUs. Automatic differentiation allows to build complex neural networks easily, leaving to the computer to figure out what the gradients are. Similarly, handling GPUs seamlessly allows the users to focus on the machine learning rather than on hardware details. One of the first of such libraries was Theano (Al-Rfou et al., 2016) developed at the University of Montreal. Theano has become deprecated as alternative have emerged with more dedicated teams: Keras and TensorFlow from Google (Abadi et al., 2015), PyTorch from Facebook (Paszke et al., 2017) and more recently JAX (Bradbury et al., 2018), another product from Google.

These libraries are developed for the Python programming language, one of the most popular choices for machine learning.

Input features. Most methods discussed in this chapter rely on the input features being real-valued. A general method to work with categorical variables is to use a one-hot encoding, introducing k dummy variables for a variable which has k modalities. This might not be necessary for binary variables whose value should be encoded as -1 and 1 for symmetry.

Although decision trees and forests are able to cope with categorical variables in principle, the implementation might not cover this case, encouraging the one-hot encoding as well.

Algorithms which cannot deal with categorical variables usually expect the training to be standardized, or at least centered, especially when some kind of algebra operates on the data (and mixes features).

Data augmentation. A simple yet effective strategy consists in artificially enlarging the learning set. Provided the mechanisms by which this augmentation operates make sense (for instance by adding the horizontal flip of natural images, taking advantage of natural symmetries), they should improve the learning due to the sheer fact of leveraging more data. Additionally, they can help the learning in other ways. For instance, adding small rotations improves the model by being more robust to those non-semantic changes.

3.8 Conclusion

This chapter introduced some of the supervised learning algorithms. It started with linear methods in both regression (Section 3.1) and classification (Section 3.2). Regularized methods have been introduced in the context of regression in Section 3.1.2. The methods presented therein, in particular, the lasso 3.1.2.2 and its variant will play a role in Chapters 6 and 9.

Section 3.3 presented the decision tree algorithm which served as a basis for the ensemble methods (Section 3.4) and boosting (Section 3.5). These methods will be the focus of Chapters 6 and 9.

The main motivation for our choice of linear classifiers was to offer a stepping stone to introduce neural networks in Section 3.6. Some emphasis was put on image processing, which will serve as context in Chapters 7 and 8. More traditional networks will be used in Chapter 9 as well.

Finally, Section 3.7 discussed a few miscellaneous topics for better learning, as well as the important questions of risk management (possibly in class-imbalanced problems) which will be important for Chapter 7.

Chapter

4 On machine learning and philosophy

Chapter overview

In this chapter, we address a few issues of supervised learning and discuss its relationship with more philosophical questions.

This chapter is divided in three sections. Section 4.1.1, discusses where machine learning stands within the study of knowledge. In particular, Section 4.1.2 discusses quickly the problem of induction and contributions of machine learning in the matter. Section 4.2 then quickly shows how machine learning can help better define a philosophical principle known as Occam's razor. Finally, Section 4.3 serves as a disclaimer regarding causality and the questions we will address in this thesis.

As Henry Poincaré famously said

mathematics is the art of giving the same name to different things

(Poincaré, 1914)

Maybe, one definition of philosophy could be "the art of spotting when different things have the same name".

4.1 On knowledge and induction

Machine learning is, as the name suggests, about learning—and a particular case at that: learning happens by generalizing from data, a process known as induction. But what is learning if not the production of knowledge? And what is knowledge then? This section investigates these issues. It should be forewarned that what follows is by no means a detailed account on the matter.

4.1.1 Knowledge

Defining knowledge is a harder task than might seem at first glance. The most widely accepted definition is the following.

Definition 4.1.1 (Knowledge)**.** *Knowledge is a true and justified belief.*

This definition falls however a bit short on two accounts. Firstly, the ambiguity surrounding the notion of justification leads to some counter-intuitive paradoxes (see Gettier, 1963). It seems that justification is a necessary component of knowledge but is not sufficient or well-enough defined; not all justifications, even when obeying the laws of logic, manage to qualify truth as knowledge.

Secondly, truth is not always assessable. For instance, nobody knows the *true* laws of the universe. Yet we speak about scientific "knowledge". Admittedly this so-called "knowledge" is well "justified". It surely is made of rational beliefs. But true?

Truth. Philosophers who have thought on this issue have come up with a couple of solutions. A useful dichotomy, introduced by Immanuel Kant, is that of analytical and synthetic propositions[1]. The former is true by definition. For instance, `all squares are quadrilaterals` is always true. On the other hand, a synthetic statement is only true because (or when) it holds for the world we live in. For instance, `there is a cube on the desk` would be true if a cube was actually on the desk and false otherwise. It does not follow from the definition of the cube that it must be on the table referred to.

Western philosophy has a long tradition of suspicion towards synthetic statements. Plato's theory of forms had little room for the true world. Radical skepticism, such as Pyrrho's, completely disregarded reality as untrustable. Even Descartes, with his demon, illustrates how synthetic propositions have been regarded with caution.

Analytical statements are true in all realities, but what if we want to *know* something about the reality we live in? Epistemic (*i.e.* relating to knowledge) caution is surely an option. Another one is to actually examine what is meant by true. One meaning is the *correspondence theory of truth*: is true what is in accordance with reality. For instance, the paradigm shift of switching from Newton's theory of gravitation to Einstein's general relativity is justified by the fact the latter is in better accordance with reality. For instance, the former is not able to fully account for the orbit of Mercury.

This conception of truth is not the only one. For instance, Pythagoreans justified some of their beliefs by invoking aesthetic arguments. Sophists cared little for truth-as-correspondence. Their view resembled the *consensus theory of truth*: is true what is agreed upon. Actually, the concept which has probably been the most well-accepted throughout the ages is the *coherence theory of truth*, which states that a theory should not contradict itself and be absent of paradoxes.

[1]Other dichotomies with or without slight technical changes exist, such as *a priori*/*a posteriori* knowledge, (intra-)/extra-linguistic knowledge, or the relation between ideas/matters of facts of David Hume. We will stick to Kant's for simplicity.

No truce for truth: a digression. The ambivalence of the notion of truth has profound implications: changing the epistemic frame (*i.e.* changing what counts as truth and proper justifications) changes what is deemed as knowledge. Consider the following statements:

1. you should eat fruit;
2. you should eat fruit to be healthy;
3. most people think they should eat fruit;
4. fruits contain a fragment of the Earth's soul.

Within the correspondence theory of truth, and disregarding the imprecision regarding "healthy" and "most", Statement 1, by virtue of being normative, has no value. It requires another epistemic frame, such as the consensus theory of truth, for it to (possibly) belong to the realm of knowledge. In contrast, Statement 2 and 3 can actually be qualified of epistemic within the correspondence-as-truth frame. Statement 4 is also epistemic, in the sense that it can correspond to reality. It would usually be qualified as metaphysical, since, in our acceptance of the word "soul", no conformable or refutable statements can be derived from it to provide justifications.

Which notions of truth should prevail? Answering this question amounts to producing knowledge about knowledge—or motivating a position for which truth only exists in some epistemic frames. As far as supervised learning is concerned, it falls right in the correspondence theory of truth: the expected risk is the embodiment of the correspondence, with data standing for reality. Is that view ubiquitous?

Empiricism strikes back. Today's view on truth and justifications is a more complex patchwork, weaved through many turns of the wheel of time, that might appear at first glance. Paradox-free theories which model well reality are sought after but those which have some aesthetic properties, such as simplicity, continuity, symmetry and so on are examined more than others. Moreover, in the presence of competing theories (a problem known as epistemic uncertainty, see also Section 7.2.2) one usually dominates the other simply by being privileged by many.

Nonetheless, there is an undeniable focus on accordance with reality, a legacy of the philosophical movement of empiricism (or logical positivism). It somewhat justifies the use of expressions such as "scientific knowledge". Although what is held as knowledge at the moment might not be the actual laws of the universe, they are in accordance—or at best can be for now—with reality.

From this conception of truth derives the issue of establishing accordance with reality. This goes through a series of problems such as deriving observational facts from theories, moving from direct observations subjected to perceptual illusion to measurements only interpretable through the lenses of a theory, working against biases, and so on. One of the main problems, however, is how to form theories from observations—a problem known as induction.

4.1.2 Induction

David Hume, the Scottish philosopher of the 19th century, is usually invoked when discussing the problem of induction, although criticisms date at least back to the Greek philosopher Sextus Empiricus. The heart of the problem with induction is that it does not seem valid to infer a general law from a few (consequently not all) instances. Bertrand Russell gives a compelling example of the problem:

> The man who has fed the chicken every day throughout its life at last wrings its neck instead, showing that more refined views as to the uniformity of nature would have been useful to the chicken.
>
> (Russell, 2001, Chapter 4)

By inducing from observations that the man will take care of him, the chicken comes to a wrong conclusion.

Where stands supervised learning—an inductive process—in the face of all this? The following sections will discuss this issue under the following assumption. Let $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$ be a noise-free binary classification problem. That is, the problem is fully characterized by the pair $(\mathbb{P}_{\mathcal{X}}, f)$ where $f(x)$ is a binary function[2]. Given this setting, the expected risk of a hypothesis h will be rewritten as

$$\mathcal{R}_{\mathcal{X},f}(h) = \mathbb{E}_X\{\ell_{0-1}(h(x), f(x))\} \tag{4.1}$$

4.1.2.1 Distribution shift

One might understand the philosopher's doubts as questioning how present regularities will be affected beyond the present horizon. In Russell's example, the chicken is not wrong in trusting the man, up to the point where a sharp—hence unpredictable—discontinuity occurs.

One way of formalizing all this is through distribution shift[3].

Proposition 4.1.1 (Error bound under distribution shift). *Let $(\mathbb{P}_{\mathcal{X}_1}, f_1)$ and $(\mathbb{P}_{\mathcal{X}_2}, f_2)$ be the problem on which learning happened and the shifted problem, respectively. Furthermore, let h be the hypothesis learned on the original problem. How h fares in the new setting can be bounded by*

$$\begin{aligned}\mathcal{R}_{\mathcal{X}_2,f_2}(h) &\leq \mathcal{R}_{\mathcal{X}_1,f_1}(h) + \mathcal{R}_{\mathcal{X}_1,f_1}(f_2) + \left|\mathcal{R}_{\mathcal{X}_2,f_2}(h) - \mathcal{R}_{\mathcal{X}_1,f_2}(h)\right| && (4.2)\\ &= \mathcal{R}_{\mathcal{X}_1,f_1}(h) + \mathcal{R}_{\mathcal{X}_1,f_1}(f_2) + d_{f_2,h}\left(\mathbb{P}_{\mathcal{X}_1}, \mathbb{P}_{\mathcal{X}_2}\right) && (4.3)\end{aligned}$$

[2]The general setting can be recovered by posing $\mathbb{P}_{\mathcal{Y}|x}(y|x) = \mathbb{I}(y = f(x))$.

[3]It should be noted that distributions can vary in other ways than the one presented here. See Section 7.2.1.2 for other examples.

Proof. The proof is a variant of the proof given by Ben-David et al. (2010) where f_2 takes the place of $h*$ (notation theirs).

$$\mathcal{R}_{\mathcal{X}_2,f_2}(h) \leq \mathcal{R}_{\mathcal{X}_1,f_2}(h) + \left|\mathcal{R}_{\mathcal{X}_2,f_2}(h) - \mathcal{R}_{\mathcal{X}_1,f_2}(h)\right| \tag{4.4}$$

$$\leq \mathcal{R}_{\mathcal{X}_1,f_1}(h) + \mathcal{R}_{\mathcal{X}_1,f_1}(f_2) + \left|\mathcal{R}_{\mathcal{X}_2,f_2}(h) - \mathcal{R}_{\mathcal{X}_1,f_2}(h)\right| \tag{4.5}$$

The second step is due to the triangle inequality and the symmetry of the loss. As such, the conclusion is valid for any metric-based loss. □

The first term in Eq. 4.3 is the risk of h on the original problem. The second reflects how the two tasks relate, or, more precisely, how the shifted labeling function would fare on the original problem (this term is null if $f_1 = f_2$, *i.e.* no concept shift). The third term reflects the distance between the densities (*i.e.* null if there is no covariate shift).

In essence, the result is saying that induction might be robust to small shifts (either in concept or density). Conversely, in the presence of a large shift, the traditional guarantees of statistical learning collapse.

The discontinuity needs not have any time-bound quality. The key element is that a model is used in conditions different than when learned. Examples include applying a physical law at different scales, facing a biased data-collection process (see Section 4.2 for an example), trying to influence the output by intervening on an input variable (see Section 4.3 for a concrete example) and doing extrapolation in general.

Disqualifying distribution shift amounts to requiring post-training data to be independently and identically distributed (iid). Relying on such an assumption range from common and implicit to doubtful or hard to notice. For instance, physicists usually assume that the laws of nature do not change every other day. Therefore, a model learned at some point in time would still be adequate at another. On the other hand, when making extrapolation, scientists may proceed with caution—assuming of course that they realize it is an extrapolation.

Arguably, distribution shift is only one interpretation of the philosophers' doubts. Another way to look at Russell's chicken is to consider the distribution has not changed but there was actually no ground to trust the regularity in the farmer's patterns. This is formalized by the upcoming section.

4.1.2.2 No free lunch theorems

The no free lunch (NFL) theorems refer to the work of Wolpert (1995; 1996; 2002). It is established in the case of finite (possibly uncountable) $\mathbb{X}$ and regards off-training set (OTS) error: "[the] generalization error for test sets that contain no overlap with the training set" (Wolpert, 1995). We will denote by

$$\check{\mathcal{R}}^{(n)}_{\mathcal{X},f}(\mathcal{L})$$

the off-training set expected risk of a learning algorithm $\mathcal{L}$.

The practical implication of NFL is the following statement.

Definition 4.1.2 (No Free Lunch). *For any two machine learning algorithms $\mathcal{L}_1$ and $\mathcal{L}_2$, the no free lunch theorems imply that*

$$\text{uniformly over all } f, \quad \mathbb{E}_f\{\check{\mathcal{R}}^{(n)}_{\mathcal{X},f}(\mathcal{L}_1) - \check{\mathcal{R}}^{(n)}_{\mathcal{X},f}(\mathcal{L}_2)\} = 0 \tag{4.6}$$

In words, NFL implies that, on average over all problems, no learning algorithm is better than the other.

What NFL implies, is that for any problem on which a learning algorithm produces a (the?) good hypothesis, there exists a problem for which it would produce a bad (*i.e.* (at least) worse than another algorithm's) hypothesis. Consequently, there is no win-them-all learning algorithm.

At first glance, it might appear that NLF only commands caution regarding the generalizability of the results of an algorithm on a few problem instances. In this sense, it is more a result about meta-induction. There is no reason for the shadow of meta-induction to loom over induction. After all, once the hypothesis is learned, it is possible to estimate its risk via an unseen set and thus decide to keep or not.

The problem with this line of reasoning is that model selection is a wrapping learning algorithm, thus subjected to NFL as well. Even without making any decision based on any post-training risk estimate, the NFL still holds. As Wolpert (2002) argues, a good way to understand NFL is to see it backward. Imagine that data (certainly the training set but possibly also the test set) are first generated according to $\mathcal{X}$, then the hypothesis h is chosen to offer pseudo-ground-truth. A function f consistent with h on the data (including the test set) can be chosen afterward. Therefore there is no reason to believe that h and f will agree on anything beyond the data.

Another incorrect objection concerns the finitude of $\mathbb{X}$. Firstly, it does not constitute a real restriction, since any data must be quantized and encoded to be processed. Secondly, we can argue that NFL only applies to the off-training risk. However, we usually expect the OTS to be much larger than the training set. Assuming otherwise would mean we are mostly dealing with memorization (a much easier task) rather than generalization. Has NFL dealt a deadly blow to induction?

Puzzling is the scenario where a model would be learned on a training set, perform well on an independent test set, and then fail elsewhere. Making it to the test set means one of two things: either the learning algorithm has been able to pick up some regularities, or it is just chance. The probability of the latter happening can be quantified by statistical learning theory. With a large test set, it would require a very unlikely draw to be so unlucky for the test set to be fitted as well by sheer chance. Additionally, this is not really the point NFL is trying to make, which leaves us with the idea that the regularities which have been picked up are not representative (enough) of the (whole) phenomenon.

This point is made by NFL in the uniform averaging. On that matter, Wolpert (2002) argues that, since the distribution over f is unknown, making any other assumption amounts to blindly favoring some learning algorithms (hence some hypothesis spaces). Yet making some assumptions might be more natural than might seem: *e.g.* using convolutional neural networks on natural images, using regularizer when redundancy among variables is suspected, using simpler hypothesis spaces on simpler problems, and so on. By tailoring the hypothesis space, induction bias is introduced. At a high level, this amounts to choosing the kind of regularities which must be learned, narrowing down the search for a good hypothesis.

4.1.2.3 The importance of assumptions

The results of the previous two sections contrast—or seem to—with the theory developed in Chapter 2. Yet the results obtained therein (bias-variance decomposition, statistical learning bounds) were already saying induction could fail if the learning algorithm was not appropriate. Too expressive a hypothesis space and no guarantee could be given (overfitting). Too restrictive and there was a high chance that no decent model could be found (underfitting). Even in the appropriate regime, a "likely good" bound is synonymous with "unlikely bad"; it is still possible to end up with a model which only performs wells on the training set.

Nonetheless, it seemed that resorting to an estimate of the risk could cast all doubts aside. Both the distribution shift and NFL brush that hope aside. In the first instance, the estimate may become hopelessly (and unquantifiably so) biased. The second says a function which would fit well every data and disagree with the chosen hypothesis elsewhere exists.

One conclusion is that induction is built on sand. Another, more interesting, is that induction requires additional (not always fully voiced) assumptions. In a sense, it is the meta regularities (*e.g.* the stationarity of data) which allow for leveraging regularities from the data. Provided these assumptions hold, induction works (in a probabilistic sense).

As such, induction is no different from any other synthetic knowledge. For instance, Euclidean geometry is, in itself, analytical knowledge. Applying it to the world goes through the assumptions that reality conforms to its axioms, though.

Without assumptions, it would seem that the form of knowledge to which we can inspire is quite restrictive. It might feel that humanity has not progressed much on the problem of induction. This might not be true. Pinpointing the crucial assumptions has paved the way to better direct effort towards tackling the issue, and machine learning might yet provide a tool to sidestep it in the form of recognizing when induction would fail—but this will have to wait for Chapter 7.

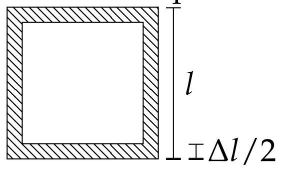
4.1.3 The curse of dimensionality

Not all knowledge is equally easy to learn. One important characteristic in this regard is dimensionality p. We have already mentioned that the cardinality of a complete finite hypothesis space grows exponentially with the dimension (Section 2.3) and that the VC dimension of the set of all linear indicator functions is $p + 1$ (Section 2.6.2). Given how these quantities are involved in the bounds over the generalization gaps (Section 2.6), it is clear that picking up a hypothesis from a higher-dimensional space is a harder challenge. The fact that learning in high-dimensional regimes is more difficult is known as the *curse of dimensionality*. It is supported by other considerations.

The first one is the evident computational burden which comes with more dimensions: a larger matrix to invert for linear regressions, more components in the gradient vectors (*e.g.* logistic regression), more dimensions along which to compute uncertainty reduction in decision trees, and so on.

The second consideration has to do with sampling. Since the volume grows exponentially with the dimensionality, getting representative samples soon becomes infeasible. For instance, sampling the unit (hyper-)cube over a grid spaced by 0.1 requires 10^p points. Large datasets contain less than 10^{10} sample points whereas they have a dimensionality $p \gg 10$. On the bright side, data is usually believed to lay in a much lower-dimensional manifold—yet another assumption.

A final consideration has to do with how the points are distributed in higher dimensions. Imagine an hypercube with side-length l. Suppose we leave out the outer-most strip of size $\Delta l/2$. The ratio of volume the strip represents compared to the original cube is



$$\frac{V_{\Delta l}}{V_l} = 1 - \left(1 - \frac{\Delta l}{l}\right)^p \tag{4.7}$$

As the dimensionality increases, the strip will occupy more and more of the total volume. For instance, a 10% strip ($\Delta l/l = 0.1$) occupies 27% of the volume in three dimension. In ten dimensions, a 10% strip covers 65% of the total volume. Consequently, an increasing fraction of data (possibly) populates the outskirts of the space. Making predictions there is harder because it means extrapolating, rather than interpolating.

4.1.4 Bayesianism

We would be remiss not to mention Bayesianism in this chapter. As an epistemic movement, Bayesianism proposes to use the theory of probability to model beliefs. This comes with many advantages.

Firstly the quantification of belief is useful in addressing questions such as how much a belief should be updated in the presence of evidence (done via Bayes rule). Secondly, this unifies under the same framework knowledge and believes, the former being a special of the latter (provided the belief is true).

It also allows modeling the fact that different agents may have different views on a matter even while having witnessed the same evidence (via the prior). This feature, which echoes the philosophical debates about internalism versus externalism knowledge, is as much an advantage as a drawback, and Bayesianism has long been shunned on the ground of its subjectivity—as if knowledge was not already riddled with it (*e.g.* choice of epistemic frame and aesthetic criteria).

Finally, requiring to explicit the event space helps in avoiding many pitfalls of informal reasoning, especially by keeping perspective of all the alternatives. On the downside, this non-commitment comes with a heavy computation burden.

Bayesian machine learning. Bayesianism can be implemented in machine learning by assigning prior to each hypothesis. Learning then consists in updating the believes based on data (the evidence). This is done through Bayes theorem (illustrated here on a parametric space and assuming iid data):

$$\begin{aligned} \mathbb{P}\left(\theta|LS\right) &\propto \mathbb{P}\left(LS|\theta\right)\mathbb{P}(\theta) && (4.8)\\ &\propto \mathbb{P}(\theta)\prod_{i=1}^{n}\mathbb{P}\left((x_i,y_i)|\theta\right) && (4.9)\end{aligned}$$

Inference is done by marginalization:

$$\mathbb{P}(\mathcal{Y}=y|x,LS)=\int_{\theta}\mathbb{P}(\mathcal{Y}=y|x,\theta)\,\mathbb{P}(\theta|LS)d\theta \qquad (4.10)$$

Following the Bayesian recipe to the letter with a large enough parameter space is cumbersome, which is why it is usually approximated (*e.g.* Wilson and Izmailov, 2020).

4.2 Occam's razor

Occam's razor, also known as the parsimony principle (or the law of parsimony), states that "Entities should not be multiplied beyond necessity." (Tornay, 1938). As an epistemic statement, Occam's razor is quite normative. Is it supported by anything other than aesthetics?

Within the field of machine learning, Occam's razor has been the main topic of several articles (*e.g.* Blumer et al., 1987; Domingos, 1999; Rasmussen and Ghahramani, 2001). The emphasis has been to discuss what the multiplication of "entities" meant so that the principled would turn outright. This should be reminiscent of Chapter 2 and more precisely of Section 2.6. Therein, we saw that a particularly fruitful notion of expressiveness was the VC dimension. It was also apparent that the expressiveness had little to say on the matter of the right hypothesis: the bounds we discussed establish the risk of overfitting; with high expressiveness comes high risks.

Discriminating between two hypotheses *a posteriori* based on the principle makes little sense. This would entail discarding the hypothesis coming from the more expressive space, or the most complex one for another (less well-motivated?) interpretation of "multiplied entities" (such as the number of parameters, or the minimum description length). Ironically, the no free lunch theorems imply the exact opposite: for some problems the good choice would be to favor the more "complex" hypothesis. Imagine, for instance, you record the temperature outside everyday *at noon*. We expect a single periodic function with an annual period would fit well the data. It would not model the true phenomenon well, however, as there is also a night-and-day cycle and another periodic function is necessary for this higher frequency. From the data the two models are indistinguishable. Leaving to Occam's razor to choose leads to the incorrect solution.

Cutting oneself, the danger of Occam's razor: a digression. Machine learning might have given an edge to Occam's razor that goes beyond justifying the principle within the field. The VC dimension interpretation of expressiveness sheds some light on why explanation length might not be a relevant factor in informal use of Occam's razor. To consider only one example, think of conspiracy theories. Almost anything can fit within a conspiracy theory, even though the explanation ("it is because of the conspiracy") is quite short[4]. On the other hand, almost anything can be accounted for with this theory. As a consequence, the informal equivalence of the VC dimension would be very large, and the risk of over-interpreting high. Once more, that is not to say that a conspiracy could never happen because there will always be a simpler explanation (in the sense of the informal VC dimension). It is an interesting consideration when weighing the hypotheses, however.

Of course, the parallel is fragile since we talk here about an interpretative subject (by opposition to predictive), in which case establishing the accordance with reality is not straightforward. But this should be taken as further motivation for caution.

4.3 Causality

In several instances in this thesis we (will) use expressions such as "understanding the input-output relationship", "gleaning knowledge from the data", "capturing the underlying phenomenon". These, and the whole of Chapter 9, should be taken in a correlative/predictive sense, rather than a causal one. The kind of information which is gathered relates to how a prediction about the output can be made when facing a new datapoint drawn from the same data source. This does not imply that the techniques developed in this thesis allow us to say much about how meddling with data would influence the output in the real world.

To take a concrete example, suppose you collect observations about children, in particular how time is spent watching TV and the result to some standard tests. Let us further imagine that there is a negative correlation between the two: more TV and less good results go together. Is increasing the

[4]Arguably, "conspiracy" acts as a shortcut and what is actually meant should be substituted in the explanation when measuring its length. It is doubtful this would invalidate the conclusion, however. On the one hand, it should be established this is actually how Occam's razor would be interpreted. One argument against this is the difficulty of defining precisely such highly expressive (in the VC sense) theories. On the other hand, substitution would add a constant to the length, whereas a conspiracy can account for so many decisions that a competing theory would soon form a conjunction of simpler hypotheses, each of which would also require context. Ultimately, the conspiracy theory would still require a shorter explanation.

scores as simple as decreasing television time? Possibly—if there is a *causal* connection between the two. For the sake of the argument, let us imagine that there is a confounding variable, for instance the socio-economic background, which influences both the time spent watching TV and the score. Maybe, less fortunate people have fewer activities to propose to their children, who live in conditions where preparing for tests is more challenging. Cutting down television time does not improve the study conditions, resulting in little improvement overall.

Gleaning from data the knowledge that there is a correlation between the two variables, or simply being able to extrapolate by asking a well-calibrated model what it would predict for a given input does not imply that acting accordingly to those insights would deliver the expected result in the real world. Nonetheless, getting some sense of how entities co-vary is a first step in the direction of causality.

The study of causality is a whole field, in and of itself, and we will not address such questions in this thesis. The interested reader can refer to Pearl et al. (2000) for an in-depth look at the matter.

4.4 Conclusion

In this chapter, we discussed some of the issues which lie at the intersection of philosophy and machine learning. Section 4.1 contextualized machine learning in the broader scope of epistemology, the study of knowledge. After a very quick and timid dip in the more-than-two-millennia discussion on knowledge (Section 4.1.1), we ended up on the problem of induction. Section 4.1.2 delved into how machine learning looks at the problem. More specifically, we discussed the problem of distribution shift (4.1.2.1) and the so-called no free lunch theorems (4.1.2.2). We concluded that induction did work provided additional assumptions. Since we were on the topic of knowledge, we briefly commented on learning difficulties in high dimensions (Section 4.1.3) and introduced Bayesianism and its machine learning variant (Section 4.1.4).

The remainder of the chapter was dedicated to two other questions of philosophical nature. Section 4.2 discussed Occam's razor, a principle which advocates the use of the simplest explanation. The field of machine learning has contributed to this question by better defining the notions involved so that it would go beyond being a mere aesthetic statement. Section 4.3 clarified some vocabulary which hinted at causality when, in truth, our endeavor is much more modest.

This chapter provided an opportunity to finish our tour of machine learning. Starting from the next chapter, the core of the thesis begins with a discussion about constraints.

Supervised learning under constraints

Part II

Chapter

5 Machine learning under constraints

Chapter overview

This chapter firstly discusses constraints in supervised learning: what is meant by constraints (Section 5.1.1), why they matter (Section 5.1.2) and how they can be categorized (Section 5.1.3). Then the constraints on which our contributions are based are broached at a high level in Section 5.2 (subsequent chapters will delve into each in more detail).
The second half of this chapter more clearly highlights which contributions of this thesis tackle which constraints (Section 5.3).

5.1 On constraints

5.1.1 Definition

In an ideal situation, enough foresight would be available to tailor *a priori* the hypothesis space, with good prior feature engineering, to the tackled problem, for which—if not unlimited, at least—plentiful of data is readily available to select the best hypothesis without concern for any memory or computational costs, at both training and inference time.

Reality is far from ideal, however. Models need to be fast, data is costly to collect and label, problems are unknown, accuracy is not the sole goal. All these practical considerations, forcing a departure from the ideal situation, constitute constraints which must be met on top of traditional machine learning.

To be more formal, remember how we framed the goal of supervised learning:

> Given data drawn from $\mathbb{P}_{\mathcal{X},\mathcal{Y}}$, a loss function ℓ, and a hypothesis space $\mathbb{H}$, the goal of supervised learning is to minimize the expected risk [...] under a reasonable time.
>
> Definition 2.2.16

Definition 5.1.1 (Constraint)**.** *According to how the goal of supervised learning was stated (Definition 2.2.16), a* constraint *is anything (i) not intrinsic to the problem which (ii) conditions the choice of some components or limit the extent to which the goal is fulfilled.*

5.1.2 Motivation

Concerns for constraints are not new. For instance, work on semi-supervised learning, whose goal is to overcome the shortage of labeled data, is half a century old (Scudder, 1965). The idea of compressing/pruning neural networks dates at least back to the 90's (*e.g.* LeCun, Denker, and Solla, 1989; Hassibi, Stork, and Wolff, 1993).

Lately, working under constraints has gained much interest, to the point where applied machine learning has almost become a field on its own, parallel to more traditional works. This recent increase of interest for practical considerations has two sources.

Firstly, recent successes of the field have led to much enthusiasm, which, in turn, has led to supervised learning being used in more and more contexts—not always close to the ideal situation. For instance, advances in natural language processing and computer vision, among others, have teased at wondrous new technologies which, by design, must operate in constrained environments.

The second source for practical considerations is the discovery of vulnerabilities or properties which are at odds with modern morals. With models being increasingly deployed and relied upon, they must prove how reliable they actually are. A company cannot stay idle while its model is indulging into non-ethical biases and might just be lawsuit away from closing if data-leakage is readily feasible through an easily exploitable model. Undoubtedly, this second source is also channeled by the massive enthusiasm for machine learning.

Irrespective of the source, modern practices fate machine learning to face constraints—or fade.

5.1.3 Typologies of constraints

This section discusses several categorizations of constraints. Section 5.1.3.1 structures constraints based on which component (*i.e.* which part of the supervised learning definition) is constrained. Section 5.1.3.2 is about the origin of the constraint, while Section 5.1.3.3 discusses when the constraints appeared in the process. In other words, the following sections discuss the what, why and when of constraints, respectively.

5.1.3.1 Component-based categorization

Given our definition of constraints, a straightforward categorization would be based on the components involved (*e.g.* hypothesis space, data, time, etc.). Since some share a common nature, they can be grouped together.

Resource constraints. The first type of constraints relate to resources: data and time. These constraints usually act as a limiting factor in reaching the optimum: data scarcity can lead to overfitting and training time might force to stop early the search for the best hypothesis. In regard to the latter, Bottou and Bousquet (2007) qualify problems whose main limit is training time as large-scale learning problems.

Another important resource is memory, which may, indirectly (see Section 5.1.3.2), constrain several other components.

Formulation constraints. Other constraints act on how the problem is formulated in terms of loss function, hypothesis space, learning algorithm and sometimes how the learning problem itself is formulated.

The component most often affected is the hypothesis space (examples will follow). Large-scale learning in the context of deep learning, by resorting to stochastic gradient descent rather than more advanced algorithms, offers an example where the training algorithm is chosen to browse through more datapoints in a limited time. Overcoming data scarcity is sometimes tackled through changing the learning problem as a semi-supervised or transfer learning problem (see Section 2.9.1).

Objective constraints. Sometimes, the overall formulation is left unchanged but the actual goal is changed so that the objective is no longer to aim for the best generalizer, but rather find a compromise with another goal. Using a lasso penalty to bias the search towards a more interpretable model falls under the objective constraint category.

5.1.3.2 Source-based categorization

Whereas looking at which component is impacted is useful to organize solutions, looking at the source is useful to discuss issues.

Cascading effect: source versus resulting constraints. Consider the two following scenarios: (i) real-time inference is needed, (ii) training data of good quality is expensive to collect. Possibly, both of these can be dealt with by choosing an appropriate hypothesis space. In the former case, restricting to fast models solve the problem by design. In the latter case, a less expressive hypothesis space reduces the risk of overfitting.

In the real-time inference case, real-time only constitutes a constraint to the extent that it conditions the choice of hypothesis space. In contrast, in the data-scarcity case, there is a cascade from the resource constraint (*i.e.* the source) to a formulation constraint (the resulting constraint).

Implicit cascading effects. Cascading effects are sometimes more subtle than in the previous examples. For instance, the decision tree induction algorithm limits the tree depth based on the number of training instances. Thus the hypothesis space is actually implicitly constrained by the resources.

Another case relates to the example involving the lasso given above. By penalizing the objective (formulation constraint) a (soft) constraint is imposed once more on the hypothesis space.

5.1.3.3 Chronology-based categorization

A final consideration to get a full(er) picture of constraints relates to when they are raised and dealt with.

Design versus post-training constraints. When constraints are anticipated prior to learning, they can be tackled as part of the overall design of the solution—design constraints. Sometimes, a constraint is dealt with after training; a post-training constraint (PTC). For instance, one might want to equip a trained model with some form of robustness, or try to explain the predictions of a black-box model.

PTCs may be made apparent once the model is already deployed, or be the results of a choice to delay enforcement. A general solution is to restart learning with the constraint in mind, tipping over a new design constraint. This might not always be possible, however.

When it is not possible to simply restart learning, it is particularly interesting to look at how the constraint is dealt with. A first alternative is to alter the model. An example of this is simplifying a large, trained model (*e.g.* post-pruning of decision trees/forests, low-rank approximation, quantization; see Chapters 6 and 8, respectively). Simplifying blindly is a risky undertaking. Consequently, model alteration techniques usually benefit from some form of (re-)learning, leaving them somewhat on the fence between training and post-training approaches.

On the other hand, the model might be left untouched. In such a case, it might become embedded in some other component (*e.g.* a filtering mechanism to discard bad inputs; see Chapter 7) or the constraint might be met with some additional computations (*e.g.* extracting interpretations; see Chapter 9).

5.2 Examples of constraints

The upcoming sections will look more closely at a few constraints related to this thesis, namely (organized by source)

- fast training;
- small models;
- robustness;

- data scarcity;
- interpretability.

These constraints will be discussed in general terms in the present chapter. Chapter 6 to Chapter 9 will explore them in more depth in their respective context. Problem-specific references will be given in those upcoming chapters.

5.2.1 Fast training

Even long before artificial intelligence, computer science has always been critical towards what could be achieved and what could be achieved in a timely manner. In the context of supervised learning, this has led to the design of fast algorithms. That is, polynomial (ideally less than quadratic) in time with respect to the number of samples and the dimensionality of the problem.

With time, more complex problems have been tackled, which requires more instances to learn from. To give an order of magnitude, ImageNet (Deng et al., 2009), an image classification dataset, contains more than 10 millions images spread over 1000 classes. Moreover, today's datasets are sometimes split across several sites for a number of reasons (size constraints, privacy, robustness, and so on). As such, learning must now account for communication cost as well.

On the other hand, more complex methods—literally requiring more computation steps for a given problem—have also been introduced. We currently see deep networks with tens or even hundreds of layers (He et al., 2016; Zagoruyko and Komodakis, 2016; Huang et al., 2017, e.g.). Since the work of Krizhevsky, Sutskever, and Hinton (2012), it has been recognized that even shallower models can no longer be trained on traditional hardware. Frameworks such as Theano (Al-Rfou et al., 2016), TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2017) and JAX (Bradbury et al., 2018) have allowed to keep the technical overhead low and thus the development time as well. The high cost of the equipment (either to buy or rent), however, remains a problem for many, further freezing the market in an oligopoly, with academic research lagging behind industrial research.

More complex methods usually come with more hyper-parameters to tune, which results in heavier cross-validation loops for already long training. Overall, we have moved to more data, higher-dimensional problems, more cumbersome models and additional communication cost. In this context, offering decent and fast solutions might be of more practical value than beating state-of-the-art records by a small margin.

Contributions. This emphasis for fast solutions is scattered throughout our contributions and usually takes up the form of a design constraint. In Chapter 6, we propose a pre-pruning method for decision forests to avoid building the whole forest as a pre-processing step. In Chapter 7, although the context does not allow for much learning to happen, we propose out-of-distribution indicators with little extra computational cost. Finally, in Chapter 8, we propose to circumvent computation-intensive methods by leveraging additional data to transfer the knowledge from one network to another.

5.2.2 Small models

As the previous section highlights, we have turned to more complex problems and methods. Often this implies models with a larger memory footprint, either by design (deep networks) or due to the availability of more data (for non-parametric models such as decision trees and forests).

On the other hand, new practices have led to machine learning being squeezed into embedded devices to offer ubiquitous intelligent agents. Machine learning must run on mobile phones and low-memory devices. When communication with large computing infrastructures can be guaranteed, predictions can be discretely outsourced (with all the privacy issues that might come along), but this is not always possible: communication can be disabled (*e.g.* no internet connection, flight mode) or not responsive enough.

Sometimes, the bottleneck is not exactly the memory footprint but having a small model solves the problem just as well. Smaller model is often synonymous with less computation steps, a requirement for fast predictions (such as required for real-time inference) and the lesser energy consumption required by battery-powered devices. Less energy might in turn expand the durability of the hardware operating the model.

Interestingly, rooting for small models is also an implicit form of regularization which can help prevent overfitting. In contrast, if the model is selected from a hypothesis subclass not expressive enough, the model might also end up being too simple due to representational bias. Therefore, regularization and the quest for small models may not pursue the same goal.

Contribution. Looking for small models or compressing bigger ones will be the topic of Chapter 6 for decision forests and Chapter 8 for deep networks. Each chapter will discuss the specificities of model compression, in particular why it is feasible in the first place, how it is tackled and how well it performs.

5.2.3 Robustness

Robustness is the overarching idea that a model should be able to withstand some unsuitable usage and still be able to function correctly. Unsuitable usage can come in many flavors: improper use, faulty equipment feeding abnormal inputs, user mistakes, intentional attack, and so on. Robustness is a vast topic, ranging from preventing small perturbations of the input to fool the model (aka. adversarial attacks (Goodfellow, Shlens, and Szegedy, 2015)), to preventing data leakage and privacy breaches (*e.g.* Fredrikson, Jha, and Ristenpart, 2015), to avoiding discrimination bias from creeping up in policy-enabling models.

Ideally, robustness should be a design concern, apparent from the start. In practice, robustness can come as an afterthought, especially with model shipped "as is". Admittedly, it might not be easy to know against what a model should guard before the model is deployed. This is even more so when the people responsible for building the model are not those using it. Even if robustness was a design priority, new vulnerabilities are bound to be discovered, forcing a patch approach of robustness.

Contribution. Chapter 7 will look at the problem of out-of-distribution detection: finding when a given input does not belong to the same distribution as the one used during training. We will look at how to implement it on an already deployed model, even when training data is no longer available.

5.2.4 Data scarcity

Data is a prime component of induction: without it selecting a hypothesis is like flipping millions of coins at once and hoping for them all to turn head. Sadly, collecting data takes time and money. In certain areas, data is plentiful, but labels scarce. Sometimes, data are incomplete with partially missing records. Due to the nature of the phenomenon or how it is collected, the training distribution might be imbalanced or different from what will be used at inference time.

Even when data is available for training, storing them for long term use is costly. It might even not be legally feasible for privacy constraints with medical or personal data. Additionally, when learning and operating the model are decoupled, teams might be reluctant to make data available to potential competitors. And—of course—data can mistakenly be lost.

Overall, having trained with quality data offers little guarantee that it will still be available later, say, to enforce robustness or derive a smaller model.

Contribution. Enforcing a form of robustness without data will be at the heart of Chapter 7, where our goal will be to do out-of-distribution detection for an already-deployed model (post-training constraint). Chapter 8 will look at how it is possible to transfer the knowledge of a big deep learning model to a smaller one when using the original training data is no longer an option.

5.2.5 Interpretability

Interpretability refers to gleaning some understanding of the underlying relationship between inputs and output of a given phenomenon. Interpretability can be instantiated in many ways but somewhat interacts with the traditional goal of supervised learning. On the one hand, a model needs to be accurate to provide trustworthy insight. On the other, too complex a model does little to offer intelligible knowledge from a human perspective. Although this might be reminiscent of the bias-variance tradeoff, the equilibrium for accuracy and interpretability might not (and often will not) be the same, resulting in a conflict between the two objectives.

In some area, interpretability is at most an appreciated bonus. In other, it is the prime goal and the learned hypothesis is of little use beyond that. Most situations fall in some middle ground and learning more on the studied phenomenon usually brings many advantages such as targeting a more suitable hypothesis space or discovering uninformative (or marginally informative) variables which can be discarded with little negative impact, to mention a few.

Contribution. Chapter 9 is dedicated to the topic of interpretability. Section 9.2 will investigate how we can re-use the core algorithm of Chapter 6 to design the search for intrinsically interpretable models. Section 9.3 will turn to the problem of measuring which variables are most important.

5.3 Overview of the following chapters

The previous sections have delved into some constraints and, for each, the relationship with our contributions have been mentioned. In this section and for the sake of clarity, we do the opposite: for each contribution we explain under which constraints they operate.

Chapter 6 is dedicated to building small decision forests with a technique known as Globally Induced Forests (GIFs). GIFs allow to produce a small model by enforcing a hard constraint on the number of nodes in the forest (a design constraint). Framing the algorithm as a pre-pruning method ensures it also economical during training.

Chapter 7 focuses on robustness and in particular of out-of-distribution (OOD) detection. This is examined as a post-training constraint where data is also scarce, a context we introduced as data-free OOD detection.

Chapter 8 comes back to the problem of small models, in the context of deep networks this time. The problem is tackled as a post-training constraint where data scarcity is overcome thanks to the presence of a collection of relevant unlabeled data. As with GIFs, size is controlled through a hard constraint as the final model architecture is a prior choice of the method.

Chapter 9 discusses the topic of interpretability. The first half employs GIFs to build rule sets, an interpretable model. The second half poses interpretability as a post-training constraint where the goal is to analyze which variables are most crucial to predict the output.

6 Chapter Globally Induced Forests

Chapter overview

This chapter tackles the (pre-)pruning of decision forests: how to produce lightweight-yet-accurate models under a node budget. To accomplish this delicate balance, we use a mix of global (to reduce redundancy) and local (to regularize) optimizations.

This chapter is based on our publication "**G**lobally **I**nduced **F**orests" (Begon, Joly, and Geurts, 2017) presented at the 34th International Conference of Machine Learning (ICML).

GIF is implemented as a Python package available at `https://github.com/jm-begon/globally-induced-forest`. It is implemented on top of the Scikit-Learn library (Pedregosa et al., 2011).

The chapter is divided into five sections, the first of which (Section 6.1) exposes our ambitions: the goal pursued, our contribution and the motivation behind our work. Section 6.2 discusses compression of decision forests in general: why it is feasible (Section 6.2.1), how it can be formulated (Section 6.2.2), how it has been tackled so far (Section 6.2.3) and how our method differ from other works (Section 6.2.4).
The third section describes the GIF algorithm (Section 6.3.1), and how it is instantiated for both regression and classification (Sections 6.3.2 and 6.3.3). Finally, Section 6.3.4 justifies why GIF can be seen as a pre-pruning algorithm.
The fourth section is concerned with the empirical analysis: how GIFs fare (Section 6.4.1), how their hyper-parameters interact (Section 6.4.2) and how GIFs compare to other methods (Sections 6.4.3 to 6.4.5).
Section 6.5 concludes and proposes avenues for future improvements.

Compared to the original article, the background has been expanded with the discussions on feasibility and problem formulation. Related works have also been updated. Section 6.3.3 goes into more details on the closed-form solution for classification and its interpretation. The comparison with post-pruning method (Section 6.4.5) is new. Finally, GIFs are also the topic of the first half of Chapter 9, all new material.

6.1 Ambitions

6.1.1 Goal and contribution

Our goal is to propose a *pre-pruning* algorithm for decision forests, which we named Globally Induced Forest (GIF). By pre-pruning, we mean that it is not required to build the whole forest and then cut off branches or trees. The motivation to opt for a pre-pruning algorithm will be discussed in Section 6.2.4.

Contribution. In light of these, our contributions can be summarized as follows:

- we propose the GIF algorithm for regression and classification, a fast method to produce lightweight yet accurate models thanks to a combination of local and global optimizations;
- we conduct an empirical study to (i) validate the method, and (ii) give insights regarding the hyper-parameters introduced by the algorithm.

6.1.2 Motivation

Decision forests, such as Random Forest (Breiman, 2001) and Extremely Randomized Trees (Geurts, Ernst, and Wehenkel, 2006), are popular methods, offering overall good accuracy, relative ease-of-use, short learning/prediction time and interpretability. Their non-parametric nature allows them to adapt to the complexity of the data and consequently portray low (representational) bias, while high variance is somewhat kept at bay thanks to their randomized-and-averaged nature (see Section 3.4)

However, as mentioned in Section 5.2.1 datasets have become bigger and bigger over the past decade and with big data come big constraints. The number of instances n has increased and the community has turned to very high-dimensional learning problems. The former has led to bigger trees, as the number of nodes in a tree is $O(n)$. The latter, on the other hand, tends to steer towards larger forests. Indeed, the variance of individual trees tends to increase with the dimensionality p of the problem: more variables, more possible splits. Therefore, the adequate number of trees t increases with the dimensionality. Overall, this change of focus might render tree-based ensemble techniques impractical memory-wise, as the total footprint is $O(n \times t_p)$.

As Section 5.2.2 argues, this increase in size is somewhat in opposition to modern practice which would like intelligent systems to run on low-memory devices, such as mobile phones and embedded environments so as to become ubiquitous.

Besides memory considerations, smaller models offer many additional advantages: faster inference, lower energy consumption, more durability, implicit regularization, better interpretability (Section 5.2.2). In the case of decision trees, the number of computations required for a prediction is $O(d)$ where d is the depth of the tree. In the case of a forest, all the m trees of the forest must make prediction before being aggregated to the final prediction. Arguably, the tree predictions can be computed in parallel. It is dubious that low memory devices would be equipped with such capabilities, however. Therefore, small models will require less computation for inference (less and shallower trees), resulting *de facto* in faster inference and lower energy consumption.

All in all, tree-based models might benefit from a lighter memory footprint in many different ways. There remains to design an algorithm which produces light-yet-accurate forests—provided this is even feasible.

6.2 Decision forest compression

This section reviews several considerations related to the compression of decision forests. Section 6.2.1 delves into why compression is feasible in the first place. Section 6.2.2 shows how the problem can be formulated at a high level. Section 6.2.3 discusses related works. The specificity of our work is highlighted in Section 6.2.4.

6.2.1 Feasibility of decision forest compression

One might wonder to what extent a forest can be compressed while retaining a sufficiently low error. The first thing to note is that bigger is not always better when it comes to tree-based aggregating ensembles. It is true that the more trees, better the model (see Section 3.4). However, adding trees serves to decrease the variance of the learning algorithm. Once this reduceable variance is dealt with, more trees do not bring any more advantages but overload the ensemble substantially.

In addition, deeper trees do not necessarily imply a better forest. Trees are usually fully grown so that (representational) bias is lowest at the expense of variance, which can—in part—be reduced. This might not be the optimal tradeoff as the left-over variance might exceed the gain in bias.

Besides these considerations, decision forests portray a large amount of redundancy. Within a single tree, it is not unexpected to find very similar splits along different branches. For separable, axis-aligned problems, this might even be the norm. Split inversion along two branches, correlated variables which allow for similar splits and some robustness to the exact threshold values are further reasons encouraging structural redundancy. The prime source of redundancy is between trees, though. Building trees independently offers no mechanism to exclude redundancy, besides sheer unlikeliness due to randomization. When the number of trees is large, similar splits may well crop up in different trees, especially near the top. Near the leaves, redundancy might be predictive rather than structural. Fully-developed trees will agree on the training set (or the common instances they received). What set each tree apart is how it behaves locally to those points. Since they are based on the same (or boostrapped) samples, they have much incentive to behave closely as well.

The number of trees, depth and redundancy are levers for compression but they interplay, as well as with randomization, in a complex manner, resulting in tricky problems. For instance, increasing randomization will reduce redundancy but will increase bias. The added stochasticity can be counter-balanced with more trees and the bias with deeper trees. On the other hand, reducing randomization might render the forest only marginally better than a single tree. Ideally, randomization should be tuned according to the problem to offer the best bias-(reduced-)variance trade-off and the compression technique should be flexible enough to leverage the best way to reduce memory while keeping the good performance of the forest.

6.2.2 Problem formulations

Forest compression is counter-balanced by the need for a low error. The three compression levers (redundancy, forest size and tree depth) offer some hope of substantially reducing the memory footprint of the forest while impacting moderately the error. How much accuracy can be sacrificed is context- and problem-dependent. Methods should therefore provide a knob to adjust the compression-versus-accuracy balance. From a high level perspective, such adaptability can be formulated in several manners:

- learn the smallest forest which does not exceed a given critical error level;
- learn the forest with lowest error which does not exceed a given overall size;
- learn the forest which achieves a given size-accuracy tradeoff.

The first two options have a clear constraint which make practical sense. A critical error level might be demanded for a task and reducing the forest footprint may be accompanied by a lesser cost for the material running it. As Section 6.2.3 will highlight, this is usually not how compression is tackled in practice. Supervised learning is all about minimizing the error level, rather than minimizing some other quantity at a given error level.

The second formulation is practical: given some hardware limitations, find the best forest. Falling back to the first formulation is also straightforward by increasing the memory limit if the critical error level is not met. If the goal is to minimize equipment cost (the motivation for the first formulation), this increase-and-retry approach goes well with the discrete nature of hardware capabilities. Using the first formulation to solve the second requires more effort, though, since the link between the error level and meeting some hardware requirement is fuzzier.

The third formulation is more remote from practicality but can be used as proxy for the other two formulations by playing with the tradeoff. This more direct weighing of the two quantities of interest demands to cast them on comparable ground, which expectedly will result in one (or both) of the component(s) being only indirectly tuneable. For instance, decreasing the sparsity constraints which are weighed against the error will result in a larger forest with a smaller error. How much larger and how much better is unknown *a priori*, however, rendering this blind-eye approach quite cumbersome to use.

Problem formulations are not equivalent from the perspective of comparisons. Two methods following the first or second formulations can be compared over their optimized parameter (size and error respectively) for given constraints. Admittedly, the constraint might not be exactly met (possibly more frequently for the first formulation) but the third formulation makes it much harder to enforce some fix points at which performances between several methods can be evaluated.

Measuring model footprint. However the memory constraint manifests itself (hard design constraint, soft objective constraint, *etc.*) the footprint must be measured. A straightforward option is to measure the actual memory the model occupies on the disk (or in the RAM) in, say, megabytes or gigabytes. For instance, a forest of 1000 fully-developed extra-trees (see Section 3.4.3) learned on the MNIST dataset (LeCun et al., 1998a) with Scikit-Learn occupies of the order of $2.5Gb$ on disk[1].

This would mean the measure is tied to the actual implementation of the data structures representing the nodes, the trees, and the forest. Although of practical interest this is usually not how the footprint is envisioned. Instead, a more implementation-independent space complexity is used.

An internal node can, in principle, have an $O(1)$ (*i.e.* constant) space complexity since only the split variable and the threshold must be stored. A leaf has an $O(K)$ space complexity, where K is the dimensionality of the prediction. $K = 1$ in (single-output) regression. In classification, K represents the number of classes, provided a class vector is stored in the leaf (and not just the majority class). Therefore, the total space complexity for one tree with n_i internal nodes and n_l leaves is $O(n_i + n_l K)$.

For a full (*i.e.* internal nodes have exactly two children) binary tree, the numbers of internal nodes and leaves are linked by $n_l = n_i + 1$. Therefore, the complexity is fully characterized (once K is known) by n_l, n_i or the total number of nodes ($n_t = n_i + n_l$). This allows for straightforward comparisons between trees.

For a non-full tree (as we will deal with in this chapter), resorting to only one metric (n_i, n_l or n_t) will result in an approximation of the space complexity. Since we expect K to be some order of magnitude smaller (in particular $K = 1$ for regression and $K = 2$ for binary classification) than the number of nodes, we will use the total number of nodes n_t as a metric in this chapter, to better reflect cases where the tree is far from being full.

A note on inference time. Although the main purpose of this chapter is to produce lightweight models, one possible goal behind searching for small(er) models might be to reduce the inference time. In the case of (traditional) decision forests, the inference is usually fast. It is upper bounded by $O(n\,m)$,

[1]It should be noted that this implementation is not optimal memory-wise. In particular for classification, an internal node keeps information about class distribution and has thus a memory complexity of $O(K)$.

where n is the size of the learning set and m is the number of trees. The linearity with respect to n is in the worst-case, where the tree is fully developed and degenerates into a single branch. Assuming a (more) balanced tree, the bounds tightens to $O(m \log n)$. Additionally, each tree can be evaluated independently. Provided hardware permits, the total inference time can be divided with parallelization.

Consequently, decision forests provide fast inference. We will therefore not focus on speeding up inference in the remainder of this chapter and will simply make the following observation: for a given accuracy level, designing small models and designing fast ones might require different choices. One such example is given by De Vleeschouwer et al. (2015), who note that increasing the depth of a tree by one level will double (in the worst-case) the number of nodes while having little effect on the inference time.

6.2.3 Related works

Memory constraints of tree-based ensemble methods has a long tradition and has been tackled from various perspectives, which can be partitioned into tree-agnostic and tree-aware methods.

Tree-agnostic methods. The former set of techniques are general-purpose methods which can deal with any ensembles. We can further distinguish between re-learning algorithms and ensemble pruning methods. Re-learning methods (*e.g.* Domingos, 1997; Menke and Martinez, 2009), try to come up with a smaller, equivalent models by teaching a student model to perform as well as a teacher model. This is also known as teacher-student transfer, or distillation (Buciluǎ, Caruana, and Niculescu-Mizil, 2006; Hinton, Vinyals, and Dean, 2015). When performance is measured in terms of error, the transfer data is the same as the training data used by the teacher and the student comes for a subset of the teacher's hypothesis space, it is unclear why not learning directly the student would work as well as this teaching approach.

Ensemble pruning (*e.g.* Tsoumakas, Partalas, and Vlahavas, 2008; Rokach, 2016) try to eliminate some of the base models (*i.e.* trees in this case) constituting the ensemble. They do not attempt to reduce the complexity of the individual models. Developing sophisticated techniques under this paradigm is somewhat at odds with the forest induction mechanism. There is *a priori* no reason why a single tree would be more useful in the forest than another and the redundancy between the trees is expected to be spread uniformly. As a consequence, one can expect to reach comparable result by dropping trees at random. Moreover, these techniques only play on one compression lever, the number of trees, and therefore good results are only expected so long as the number of trees remains in the variance-reduction spectrum.

Tree-aware methods. Contrary to tree-agnostic methods, tree-aware methods take into account the structure of the base models. As such they are more amenable to use all the compression levers. Several families have been proposed.

For instance, Breiman (1999) learns the forest with a subsample of the training data. Since the training set of each tree is smaller, so is the maximum depth. Some authors have proposed to exploit the redundancy in the forest to relax it into a directed acyclic graph *a posteriori* at first (*e.g.* Peterson and Martinez, 2009) and more recently *a priori* (Shotton et al., 2013).

Along the same lines, techniques working on the whole dataset and yielding ensemble of *trees* (rather than more general graphs) can be partitioned into pre- and post-pruning methods. As mentioned, pre-pruning methods aim at stopping the development of uninteresting branches in the top down induction procedure. On the other hand, the goal of post-pruning methods is to discard *a posteriori* subtrees which do not provide significant accuracy improvements, possibly in comparison to the redundancy they bring.

Originally, pruning methods were introduced to control the model complexity and avoid overfitting. The advent of ensemble methods somewhat cast aside those techniques as the averaging mechanism became responsible for reducing the variance and rendered pruning mostly unnecessary from the point of view of accuracy. Nonetheless, a few ensemble-wise, post-pruning methods have emerged with a focus on memory minimization. In both Meinshausen et al. (2009) and Joly et al. (2012), the compression is formulated as a slightly different global constrained optimization problem. For instance, the latter re-optimizes the forest globally while imposing a L1-based penalty to enforce sparsity in the tree structure. In Ren et al. (2015), compression is undertaken with a sequential optimization approach by removing iteratively the least interesting leaves. In De Vleeschouwer et al. (2015), the authors alleviate the leaves' memory requirements by clustering their conditional distributions. After computing a wavelet coefficient for each node, Elisha and Dekel (2016) discard all the nodes which are not on the path to a node of sufficient coefficient. All these methods are able to retain almost the full forest accuracy while offering a significant memory improvement, leaving their requirement for building the whole forest first, and consequently the high temporary memory and computational costs, as their only major drawbacks.

Lossless compression. Note that all these works focus on lossy compression. Interestingly, Painsky and Rosset (2019) proposed a lossless compression scheme exploiting the tree structure and the independence of model learning. By design, lossless methods offer limited compression guarantees and important gains can only be observed for largely redundant forests—in which case reducing the number of trees might result in the same effect. As a side note, the authors also proposed a lossy scheme by limiting the number of bits needed for their encoding. Both their lossy and lossless variants suffer from the same drawback though: the need to decompress the tree on the fly at inference time. Owing to the branching structure of the tree, this decompression is efficient. Nevertheless, it is dubious that the extra cost *per inference* would be worth the compression gain for an inference-intensive task. Finally, let us note that a lossless compression scheme can always be applied on top of other compression mechanisms and would only be useless if all the redundancy has already been removed.

Recent works. Since the publication of our article, a couple of new works have been published on the topic of decision forest compression (or related). Nie et al. (2017) proposed a method very similar to Joly et al. (2012)'s. Souad and Abdelkader (2019) proposed an ensemble pruning method relying on a diversity measures for the class-conditional predictions. Nakamura and Sakurada (2019) proposed to find a number of internal nodes whose split thresholds can be altered without impacting many of training set decision paths so that more trees would share the same nodes—a first step to facilitate (a no longer) lossless compression.

Tree Alternating Optimization (TAO) was proposed by Carreira-Perpiñán and Tavallali (2018) in the context of classification and later extended to regression (Tavallali, Tavallali, and Singhal, 2019; Zharmagambetov and Carreira-Perpiñán, 2020) and ensembles (Carreira-Perpiñán and Zharmagambetov, 2020; Zharmagambetov and Carreira-Perpiñán, 2020). These works deviate from pruning because the structure of the tree(s) is fixed *a priori* rather than obtained in the classical top-down fashion. The structure is then fitted by solving a classification problem sequentially for each node, usually in the form of a regularized logistic regression which acts as split for that node. This process is repeated until convergence, hence the name "alternating". Tree(s) can be shorter than initialized as a by-product of the optimization process, rather than by design. Bagging is used to create ensembles. Overall, the method fixes the structure, learns the split of a single tree globally but trees independently, making it quite different from other techniques. It is however able to produce short tree-like structures portraying comparable performances to forests, at the cost of solving many logistic regressions. Oddly, TAOs in an ensemble are not learned globally, meaning that substantial redundancy must creep into such ensembles. Finally note that the use of linear splits and predictions prevent from comparing the memory footprints of TAOs and more traditional trees on a per-node basis. Overall, TAOs end up quite different than what is traditionally meant by decision trees. With so many differences, it is legitimate to envision changing the hypothesis space altogether.

6.2.4 GIF versus other techniques

As it stands, GIF is a rare case of pre-pruning method, that is a method which (i) does not build the whole forest, and (ii) actually operates and delivers on trees.

The motivation to opt for a pre-pruning algorithm is two fold. Firstly, since it does not necessitate the building of the whole forest, it is economical in term of both time and memory. Admittedly, these resources might be available at *training* time and one might argue that a method taking more information into account (*i.e.* the whole, fully-grown forest) would result in a better compression. As usual in supervised learning, this may also lead to higher variance and overall higher error.

The second motivation was that, compared to the existing sophisticated post-pruning methods, it allows for a more direct control of the forest's size.

Indeed, prior methods usually relied on the compression-error tradeoff formulation (the third one in Section 6.2.2), while GIF is formulated as a lowest-error-at-fixed-size optimization (the second formulation in Section 6.2.2). The fixed size is given in the form of a node budget. Since the full tree constraint is relaxed, this budget accounts for all nodes (leaves included).

Interestingly, GIF is able to play on the three compression levers (redundancy, forest size and tree depth) thanks to part of its optimization being global (*i.e.* forest-wise). The algorithm may decide to add a redundant split to start deepening a new tree, for instance. How the algorithm weighs the levers is implicitly controlled by a handful of hyper-parameters. Hopefully, the empirical study in Section 6.4 sheds some light on how to set those hyper-parameters.

Due to how GIFs are built, they share many similarities with (gradient) boosting methods (Friedman, 2001b), which fit additively tree ensembles based on a global criterion and are also able to build accurate yet small models. Whereas most boosting methods only explore ensembles of fixed-size trees, GIF does not put any prior complexity constraint on the individual trees but instead adapts their shape greedily. It shares this property with Johnson and Zhang (2014)'s regularized greedy forests (RGF), a method proposed to overcome several limitations of standard gradient boosting. Other recent close works include those of Dawer, Guo, and Barbu (2020), which adds a tree from a pool of varying-size trees, and of Zuo and Drummond (2020), which formulate the boosting objective differently.

6.3 The GIF algorithm

In this section, we present the GIF algorithm. It is first given in a general form in Section 6.3.1. It is then instantiated for regression and classification in Sections 6.3.2 and 6.3.3, respectively. Section 6.3.4 defends the point of view that GIF is a pre-pruning algorithm.

6.3.1 General algorithm

GIFs rely on the view of an m-trees forest as a linear model in the "forest space", a binary M-dimensional space, where M is the total number of nodes in the whole forest (Joly et al., 2012; Vens and Costa, 2011):

$$\hat{y}(x) = \sum_{j=1}^{M} w_j z_j(x), \tag{6.1}$$

where the indicator function $z_j(x)$ is 1 if x reaches node j and 0 otherwise, and w_j is $\frac{1}{m}$ times the prediction at a node j if j is a leaf and 0 otherwise. In regression, the leaf prediction would be the average value of the subset of outputs reaching leaf j. In classification, $w_j \in \mathbb{R}^K$ is a vector of dimension K, where $w_j^{(k)}$ ($k = 1, \ldots, K$) is $\frac{1}{m}$ times the probability associated to class k,

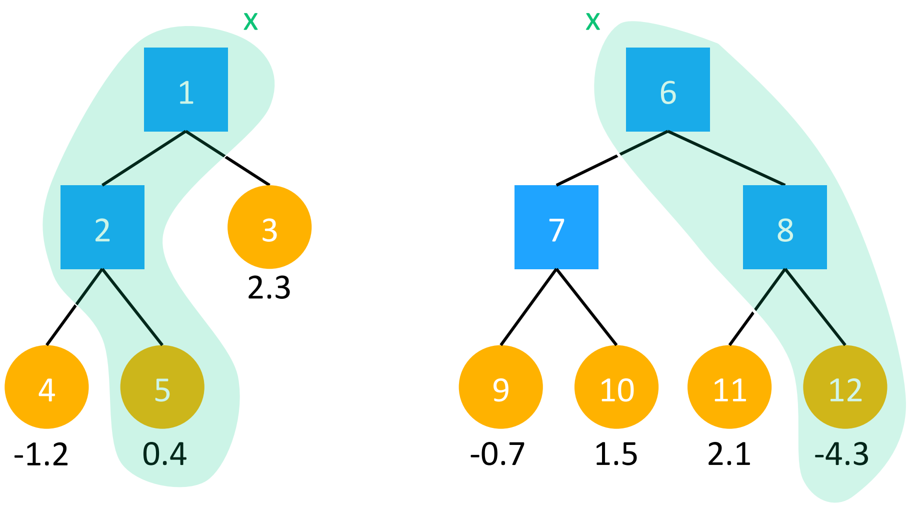


FIGURE 6.1: Forest space: how to represent a forest as a linear model. Blue rectangles correspond to split nodes while orange circles correspond to leaves. For instance x propagated in this small forest, the node indicators are such that $z_j(x) = 1$ for $i = \{1, 2, 5, 6, 8, 12\}$ and 0 otherwise. Since the weights associated to internal nodes are null, the overall prediction is $\hat{y}(x) = 0.4 - 4.3 = -3.9$.

typically estimated by the proportion of samples of this class falling into leaf j. Figure 6.1 illustrates the notion of forest space in the regression case.

The idea behind GIF is to develop the trees gradually by following a standard forest learning algorithm but pausing after each node addition to optimize its associated weight globally. The next added node is then the one which improves the forest the best. By optimizing which node to add on a global scale, GIF mitigates the redundancy: it can decide whether to add a whole new tree or to deepen an existing one, thus playing with the compression levers.

Algorithm 5 describes the GIF training algorithm in more technical details. A visual illustration is given in Figure 6.2. Starting from a constant model (Step 8), GIF builds an additive model in the form of Equation 6.1 by incrementally adding new node indicator functions in a stagewise fashion in order to grow the forest. At each step, a subset of candidate nodes $C_{[t]}$ is drawn uniformly at random from the total candidate list C (Step 11). For each of those nodes, the weight is optimized globally according to some loss function ℓ. The node j^* among those of $C_{[t]}$ which contributes the most to a decrease of the loss is selected (Step 12) and introduced in the model via its indicator function z_{j^*} and its optimal weight w_j^* tempered by some learning rate λ (Steps 13 and 14). This node is then split locally according to the reference tree growing strategy $\mathcal{T}$ (Step 16) and replaced by its two children in the candidate list (Step 17). The process is stopped when the node budget B is reached. Note that the root nodes are only accounted for when one of its

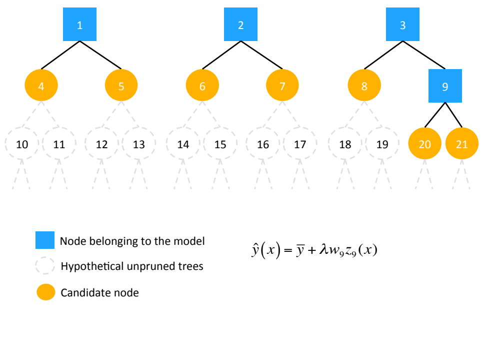
Algorithm 5: The GIF learning algorithm.

```
Input: $LS = \{(x_i, y_i)\}_{i=1}^{n}$, a learning set; $\mathcal{T}$, the tree learning
       algorithm; $\ell$, the loss function; $B$, the node budget; $m$, the
       number of trees; $CW$, the candidate window size; $\lambda$, the
       learning rate.
Output: An ensemble $S$ of $B$ tree nodes with their corresponding
        weights.
Function GIF($LS$, $\mathcal{T}$, $\ell$, $B$, $m$, $CW$, $\lambda$):
    $S \leftarrow \emptyset$                                /* the set of produced nodes */

    $C \leftarrow \emptyset$                                /* the set of candidate nodes */

    $t \leftarrow 1$
    $\hat{y}_{[0]}(.) \leftarrow \arg\min_y \sum_{i=1}^{n} \ell(y_i, y)$     /* best constant model */

    grow $m$ stumps with $\mathcal{T}$ on $LS$ and add the left and right
      successors of all stumps to $C$
    repeat
        $C_{[t]} \leftarrow$ draw a subset of size $\min\{CW, |C|\}$ from $C$ chosen
          uniformly at random
        compute
            $$(j^*, w_j^*) \leftarrow \arg\min_{j \in C_{[t]}, w} \sum_{i=1}^{n} \ell\left(y_i, \hat{y}_{[t-1]}(x_i) + w z_j(x_i)\right)$$
                     /* get the best node $j^*$ among $C_{[t]}$ and its weight $w_j^*$ */
        $S \leftarrow S \cup \{(j^*, w_j^*)\}$
        $\hat{y}_{[t]}(.) \leftarrow \hat{y}_{[t-1]}(.) + \lambda w_j^* z_{j^*}(.)$
        $C \leftarrow C \setminus \{j^*\}$
        split $j^*$ using $\mathcal{T}$ to obtain children $j_l$ and $j_r$
        $C \leftarrow C \cup \{j_l, j_r\}$
        $t \leftarrow t + 1$
    until budget $B$ is met
    return $S$
```

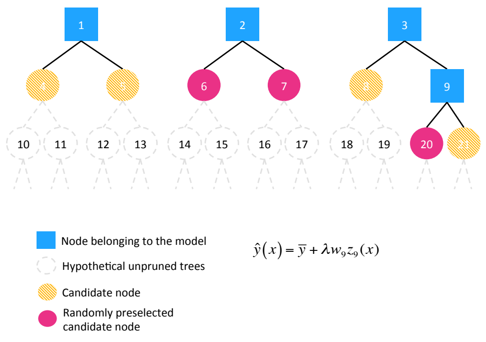
children is taken into the model (hence the condition on the budget rather than directly on t).

Contrary to Equation 6.1, each node has a non-zero weight, since it was optimized. Nonetheless, as soon as both its children are inserted, the parent node's weight can be removed from the sum by pushing its weight to its successors.

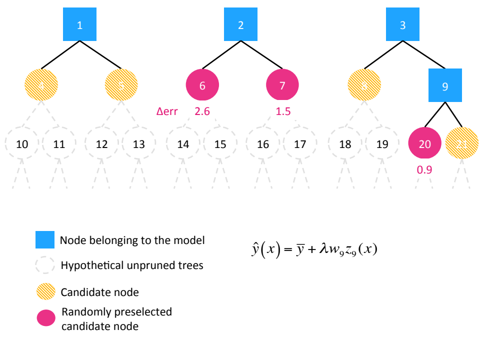
Node selection and weight optimization. Step 12 of Algorithm 5 can be decomposed into two parts. First, the optimal weight for a given candidate


Node belonging to the model
Hypothetical unpruned trees
Candidate node
$\hat{y}(x) = \bar{y} + \lambda w_9 z_9(x)$

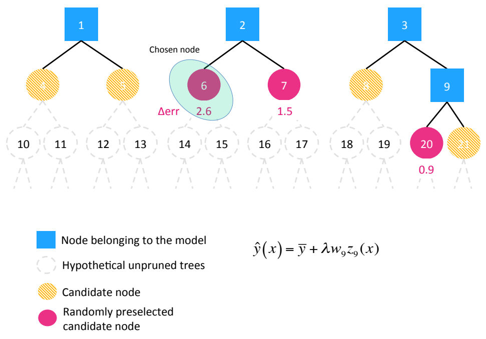
(A) Current forest at time t


Node belonging to the model
Hypothetical unpruned trees
Candidate node
Randomly preselected candidate node
$\hat{y}(x) = \bar{y} + \lambda w_9 z_9(x)$

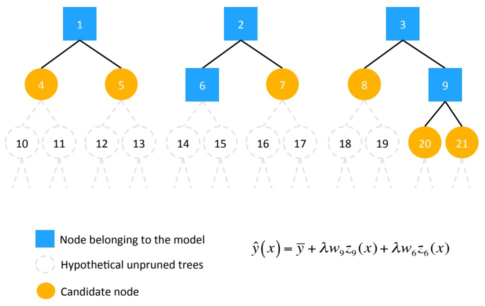
(B) A subset of candidates C_t is drawn uniformely at random from the set of candidates C (Step 11)


Δerr
2.6
1.5
0.9
Node belonging to the model
Hypothetical unpruned trees
Candidate node
Randomly preselected candidate node
$\hat{y}(x) = \bar{y} + \lambda w_9 z_9(x)$

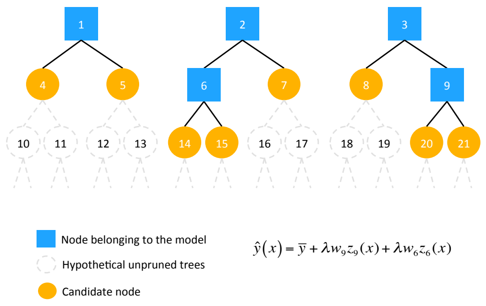
(C) The error reduction is computed for all candidates of C_t (Step 12)


Chosen node
Δerr
2.6
1.5
0.9
Node belonging to the model
Hypothetical unpruned trees
Candidate node
Randomly preselected candidate node
$\hat{y}(x) = \bar{y} + \lambda w_9 z_9(x)$

(D) The best node (highest error reduction) is selected (Step 12)


Node belonging to the model
Hypothetical unpruned trees
Candidate node
$\hat{y}(x) = \bar{y} + \lambda w_9 z_9(x) + \lambda w_6 z_6(x)$

(E) The chosen node is introduced in the model (Steps 13 and 14)


Node belonging to the model
Hypothetical unpruned trees
Candidate node
$\hat{y}(x) = \bar{y} + \lambda w_9 z_9(x) + \lambda w_6 z_6(x)$

(F) A split is determined for the chosen node (Step 16) and its children are added to the candidate list (Step 17)

FIGURE 6.2: Illustration of the GIF regression building algorithm ($m = 3$, $CW = 3$)

node at stage t is computed using:

$$w_j = \arg\min_{w \in \mathbb{R}^K} \sum_{i=1}^{N} \ell\left(y_i, \hat{y}_{[t-1]}(x_i) + w z_j(x_i)\right) \tag{6.2}$$

Closed-form formulas for optimal weights are derived in Sections 6.3.2 and 6.3.3 for two losses.

Second, the optimal node—the one which reduces the loss the most—is selected with exhaustive search. Computing the loss gain associated to a candidate node j can be done efficiently as it requires to go only over the instances reaching that node j. Indeed, finding the optimal node j^* at stage t requires to compute:

$$j^* = \arg\min_{j \in C_{[t]}} \sum_{i=1}^{N} \mathrm{err}_{j,i}^{[t]} = \arg\max_{j \in C_{[t]}} \sum_{i=1}^{N} \left(\mathrm{err}_{i}^{[t-1]} - \mathrm{err}_{j,i}^{[t]} \right) \tag{6.3}$$

where

$$\mathrm{err}_{j,i}^{[t]} \triangleq \ell(y_i, \hat{y}_{[t-1]}(x_i) + w_j^* z_j(x_i)) \tag{6.4}$$

$$\mathrm{err}_{i}^{[t-1]} \triangleq \ell(y_i, \hat{y}_{[t-1]}(x_i)) \tag{6.5}$$

in words, the former is the loss value for the ith instance after the addition of node j while the latter is the same loss prior to the addition.

Given that $z_j(x_i) \neq 0$ only for the instances reaching node j, Equation 6.3 can be simplified into:

$$j^* = \arg\max_{j \in C_{[t]}} \sum_{i \in Z_j} (\mathrm{err}_{i}^{[t-1]} - \mathrm{err}_{j,i}^{[t]}) \tag{6.6}$$

where $Z_j = \{1 \leq i \leq n | z_j(x_i) = 1\}$ is the set of instances reaching node j. Due to the partitioning induced by the tree, at each iteration, computing the optimal weights for all the nodes of a given tree is at most $O(n)$; each instance reaches only one candidate node per tree. Assuming a single weight optimization runs in linear time in the number of instances reaching that node. Consequently, the asymptotic complexity of the induction algorithm is the same as the classical forest. That is why having closed-form optimization allows to keep the GIF overhead over the tree learning algorithm mild.

Note that, since the optimization is global, the candidate node weights must be recomputed at each iteration as the addition of the chosen node impacts the optimal weights of all the candidates it is sharing learning instances with, leaving scarce room for memoization.

Arguably, the minimization of a global loss prevent from building the trees in parallel. The search for the best candidate could, however, be run in parallel, as could the search for the best split.

Tree learning algorithm. The tree learning algorithm is responsible for splitting the data reaching a node. This choice is made locally, meaning that it disregards the current global predictions of the model. As a consequence, the tree nodes that are selected by GIF are exactly a subset of the nodes that would be obtained using algorithm $\mathcal{T}$ to build a full ensemble. The motivation for not optimizing these splits globally is threefold: (i) our algorithm can be framed as a pre-pruning technique for any forest training algorithm, (ii) it introduces some natural regularization, and (iii) it leads to a very efficient algorithm as the splits in the candidate list do not have to be re-optimized at each iteration. Although any tree learning method can be used, in our experiments, we will use the Extremely randomized trees's splitting rule (Geurts, Ernst, and Wehenkel, 2006): m out of p features are selected uniformly at random and, for each feature, a cut point is chosen uniformly at random between the current minimum and maximum value of this feature.

Hyper-parameters. Of all the hyper-parameters of the GIF algorithm, only a handful are really new. The choice of the tree learning algorithm is prior to the idea of pruning and the budget is given by the context. Ultimately, the candidate window CW and the learning rate λ are specific to GIFs. The former was mostly introduced to limit the computation time taken by step 12 but can also serves as regularizer. The goal of the learning rate is to regularize the model. Since both of those parameters impacts the node selection (either directly or via how the global model is optimized), they balance the compression levers and influence the overall forest shape.

Forest shape. Three parameters interact to influence the shape of the forest: the number of trees m, the candidate window size CW and the learning rate λ.

On the one hand, $CW = 1$ means that the forest shape is predetermined and solely governed by the number of trees. Few trees impose a development in depth of the forest, while many trees encourage in-breadth growth. Since the selection is uniform over the candidates, it also implies that well-developed trees are more likely to get developed further, as choosing a node means replacing it in the candidate list by its two children (unless it is a leaf). This aggregation effect should somewhat be slowed down when increasing the number of trees (in-breadth development). Note that subsampling the candidates (*i.e.* small value of CW) also acts as a regularization mechanism and reduces the computing time.

On the other hand, $CW = +\infty$ means that the algorithm takes the time to optimize completely the node it chooses, giving it full rein to adapt the forest shape to the problem at hand. The relationship between the depth of a node and the number of instances that reach it interplays with the forest shape. A node with many instances has the potential to reduce the empirical loss significantly since it can reduce it on many examples. Conversely, the further down the tree, the easier it is to find a weight which suits the remaining instances because of how the tree is built. Therefore, the per-instance reduction is greater while affecting less examples. How the two factors balance each other out is hard to foresee. Easier to discuss is which node will be selected next given the previous choice(s) and the learning rate. If the latter is low, the previous node will not be fully exploited and the algorithm will look for similar nodes at subsequent stages. In contrast, if the learning rate is high, the node will be fully exploited and the algorithm will turn to different nodes. As similar nodes tend to be located roughly at the same level in trees, low (resp.

high) learning rate will encourage in breadth (resp. in depth) development. This is backed up by experimental evidence (Figure 6.3b).

6.3.2 Regression

For regression under the squared loss, the optimization 6.2 at stage t becomes

$$w_j = \arg\min_{w \in \mathbb{R}} \sum_{i=1}^{N} \ell_2 \left(y_i, \hat{y}_{[t-1]}(x_i) + w z_j(x_i) \right) \tag{6.7}$$

$$= \arg\min_{w \in \mathbb{R}} \sum_{i=1}^{N} \left(\hat{y}_{[t-1]}(x_i) + w z_j(x_i) - y_i \right)^2 \tag{6.8}$$

Optimizing the weight is readily feasible thanks to a closed-formula solution.

Proposition 6.3.1 (GIF's weight optimization (squared loss))**.** *Under the squared loss the optimal weight for node candidate node j at stage t is given by*

$$w_j = \frac{1}{|Z_j|} \sum_{i \in Z_j} r_i^{[t]} \tag{6.9}$$

$$r_i^{[t]} \triangleq y_i - \hat{y}_{[t-1]}(x_i) \tag{6.10}$$

where $Z_j = \{1 \leq i \leq N | z_j(x_i) = 1\}$ is the subset of instances reaching node j. $r_i^{[t]}$ is the residual for the ith training instance.

Proof. This a direct consequence of the first order optimality condition and the nature of the indicator function z_j. Let us denote

$$G_{\text{reg}}(w) = \sum_{i=1}^{N} \left(\hat{y}(x_i) + w z_j(x_i) - y_i \right)^2 \tag{6.11}$$

The first order optimality condition states that the optimum is reached at

$$\begin{aligned}
& \frac{dG_{\text{reg}}}{dw}(w) = 0 \\
\iff & 2 \sum_{i=1}^{N} \left(\hat{y}(x_i) + w z_j(x_i) - y_i \right) z_j(x_i) = 0 \\
\iff & \sum_{i \in Z_j} \left(\hat{y}(x_i) + w - y_i \right) = 0 \\
\iff & w = \frac{1}{|Z_j|} \sum_{i \in Z_j} \left(y_i - \hat{y}(x_i) \right)
\end{aligned}$$

□

Interestingly, extending to the multi-output case is straightforward: one only needs to fit a weight independently for each output. The loss becomes the sum of the individual losses over each output.

6.3.3 Classification

In classification, GIFs will output a K-dimensional vector, where K is the number of classes. Along the same lines as logistic regressions, this is a raw output, not a probability vector. To do so, GIFs will use K-dimensional weight vectors w as well, which will be learned under the exponential loss.

Classification GIFs differ from multi-class Adaboost (Section 3.5.2) in that the basis for the former are indicator functions, leaving the multidimensionality to the learnable weights. As such, the derivation of the closed-form solution is slightly more involved.

Label encoding: GIF

GIF relies on the exponential loss and therefore follows the label encoding adopted by the multi-class Adaboost (Section 3.5.2). A datapoint belonging to class k ($k = 1, \dots, K$) is encoded as

$$y^{(j)} = \begin{cases} 1, & \text{if } j = k \\ -\frac{1}{K-1}, & \text{otherwise} \end{cases} \tag{6.12}$$

At stage $t+1$, optimization 6.2 becomes

$$\begin{aligned} \min_{w \in \mathbb{R}^K} \quad & \sum_{i=1}^{n} \exp\left(-\frac{1}{K} y_i^T \left(\hat{y}_{[t]}(x_i) + w z_j(x_i)\right)\right) \\ \text{subject to} \quad & \sum_{l=1}^{K} \hat{y}_{[t]}^{(l)}(x_i) + w^{(l)} z_j(x_i) = 0 \end{aligned} \tag{6.13}$$

The purpose of the condition is twofold. Firstly, it ensures that the predictions sum to zero at each stage of the algorithm. This is consistent with how the true labels are encoded and it prevents the algorithm from decreasing the loss by focusing on well classified instances too much. Secondly, it ensures uniqueness of the solution and ease some computation.

For instance, by linearity

$$\sum_{l=1}^{K} \hat{y}_{[t]}^{(l)}(x_i) = \sum_{l=1}^{K} \hat{y}_{[t+1]}^{(l)}(x_i) = 0 = \sum_{l=1}^{K} w_j^{(l)} \tag{6.14}$$

and

$$-\frac{1}{K} y_i^T w = -\frac{1}{K}\left(w^{(k)} - \frac{1}{K-1}\sum_{l \neq k} w^{(l)}\right) \tag{6.15}$$

$$= -\frac{1}{K}\left(w^{(k)} + \frac{1}{K-1} w^{(k)}\right) \tag{6.16}$$

$$= -\frac{1}{K-1} w^{(k)} \tag{6.17}$$

where k is the class of instance i. The first step is due to the label encoding and the second to the constraint. A similar reasoning can be made for $\hat{y}_{[t]}$.

This latter property gives a shortcut to compute the loss by only looking at the class the instance belongs to (say k):

$$\ell_{\exp}\left(y_i, \hat{y}_{[t]}(x_i)\right) = \exp\left(-\frac{1}{K}\sum_{l=1}^{K} y_i^{(l)} \hat{y}_{[t]}^{(l)}(x_i)\right) = \exp\left(-\frac{1}{K-1}\hat{y}_{[t]}^{(k)}(x_i)\right) \tag{6.18}$$

Proposition 6.3.2 (GIF's weight optimization (multi-exponential loss))**.** *For a candidate node j at stage t, the solution of Eq. 6.13 is given by*

$$w_j^{(k)} = \frac{K-1}{K}\sum_{l=1}^{K}\log\frac{\varepsilon_j^{[t](k)}}{\varepsilon_j^{[t](l)}} \tag{6.19}$$

where

$$\varepsilon_j^{[t](l)} \triangleq \sum_{i \in Z_j^l} \ell_{\exp}\left(y_i, \hat{y}_{[t]}(x_i)\right) \tag{6.20}$$

$$Z_l^k \triangleq \{1 \le i \le n | z_j(x_i) = 1 \wedge y_i^{(l)} = 1\} \tag{6.21}$$

Z_j^k is the set of instances reaching node j and belonging to class l. $\varepsilon_j^{[t](l)}$ relates to the loss values of examples reaching node j and the weight is optimal

Proof. Let us denote $G_{\text{cls}}(w)$ the Lagrangian of problem 6.13 (where ζ is the Lagrange multiplier):

$$G_{\text{cls}}(w, \zeta) = \sum_{i=1}^{n}\exp\left(-\frac{1}{K}y_i^T\left(\hat{y}_{[t]}(x_i) + wz_j(x_i)\right)\right) - \zeta\left(\sum_{l=1}^{K} w^{(k)}\right) \tag{6.22}$$

Note that the constraint is formulated in terms of w and not $\hat{y}$ for ease. The first order optimality condition is such that

$$\left\{\begin{array}{l} \frac{\partial}{\partial w^{(k)}} G_{\text{cls}}(w, \zeta) = 0 \\ \frac{\partial}{\partial \zeta} G_{\text{cls}}(w, \zeta) = 0 = \sum_{l=1}^{K} w^{(k)} \end{array}\right. \tag{6.23}$$

Examining more closely the first partial derivative yields

$$\frac{\partial}{\partial w^{(k)}} G_{\text{cls}}(w, \zeta) \tag{6.24}$$

$$= \sum_{i=1}^{n}\left(\exp\left(-\frac{1}{K}y_i^T \hat{y}_{[t]}(x_i)\right)\exp\left(-\frac{1}{K}y_i^T w z_j(x_i)\right)\left(-\frac{1}{K}y_i^{(k)} z_j(x_i)\right)\right) - \zeta \tag{6.25}$$

$$= -\frac{1}{K}\sum_{i \in Z_j}\left(\ell_{\exp}\left(y_i, \hat{y}_{[t]}(x_i)\right)\exp\left(-\frac{1}{K}y_i^T w\right) y_i^{(k)}\right) - \zeta \tag{6.26}$$

$$= -\frac{1}{K}\left(\varepsilon_j^{[t](k)} e^{-\frac{1}{K-1}w^{(k)}} - \frac{1}{K-1}\sum_{l=1,l\neg k}^{K} \varepsilon_j^{[t](l)} e^{-\frac{1}{K-1}w^{(l)}}\right) - \zeta \tag{6.27}$$

$$= -\frac{1}{K}\left(\frac{K}{K-1}\varepsilon_j^{[t](k)} e^{-\frac{1}{K-1}w^{(k)}} - \frac{1}{K-1}\sum_{l=1}^{K} \varepsilon_j^{[t](l)} e^{-\frac{1}{K-1}w^{(l)}}\right) - \zeta \tag{6.28}$$

The first step is direct application of the partial differentiation coupled with a factorization of the exponential. The second step uses the definition of z_j to focus only on samples reaching node j. The third decomposes the sum by classes while using the definition of y_i and the zero-sum properties mentioned earlier. Finally, the last step consists in re-introducing the class k in the second term.

For w to be optimal, it must fullfil

$$\frac{\partial}{\partial w^{(k)}} G_{\text{cls}}(w,\zeta) = 0 \tag{6.29}$$

$$\iff \varepsilon_j^{[t](k)} e^{-\frac{1}{K-1}w^{(k)}} = \frac{1}{K}\sum_{l=1}^{K} \varepsilon_j^{[t](l)} e^{-\frac{1}{K-1}w^{(l)}} - (K-1)\zeta \tag{6.30}$$

This is true for all $k = 1, \ldots, K$, meaning that all $\varepsilon_j^{[t](k)} e^{-\frac{1}{K-1}w^{(k)}}$ are equal to the same quantity, irrespective of k. Therefore, they are all equal to each other which comes down to

$$\varepsilon_j^{[t](k)} e^{-\frac{1}{K-1}w^{(k)}} = \varepsilon_j^{[t](l)} e^{-\frac{1}{K-1}w^{(l)}} \qquad 1 \le k,l \le K \tag{6.31}$$

$$\iff w_j^{(k)} = \frac{K-1}{K}\sum_{l=1}^{K} \log \frac{\varepsilon_j^{[t](k)}}{\varepsilon_j^{[t](l)}} \tag{6.32}$$

$\square$

Trimmed exponential loss. Equation 6.19 glosses over a potential pitfall: what happens when some classes are not represented, that is $Z_j^k = \emptyset$ for some k? Given the nature of the underlying trees, it is quite possible that an internal node would be able to partition the classes. After all, it is its encouraged behavior. This is less of a problem in binary classification since, as soon as all the two classes are separated, the induction is stopped for that branch. Therefore, GIF could be restricted to operate on the internal nodes. This would not patch the multi-classification case however.

To circumvent this problem, we propose to approximate the optimal weight (Equation 6.19) in the following fashion:

$$w_j^{(k)} = \frac{K-1}{K} \sum_{l=1}^{K} \tau_\theta \left(\varepsilon_j^{[t](k)}, \varepsilon_j^{[t](l)} \right) \tag{6.33}$$

$$\tau_\theta(x_1, x_2) \triangleq \begin{cases} \theta, & \text{if } x_2 = 0 \text{ or } \frac{x_1}{x_2} > e^\theta \\ -\theta, & \text{if } x_1 = 0 \text{ or } \frac{x_2}{x_1} > e^\theta \\ \log \frac{x_1}{x_2}, & \text{otherwise} \end{cases} \tag{6.34}$$

The thresholding function τ_θ acts as an implicit regularization mechanism: it prevents some class errors from weighing too much in the final solution by imposing, through the parameter θ, a maximum order of magnitude between the class errors. For instance, a saturation $\theta = 3$ means that the class errors imbalance is not allowed to count for more than $e^3 \approx 20$.

6.3.3.1 Interpreting the GIF algorithm

In regression, the weights learned by GIFs directly relate to the current residual. A similar situation arise in classification, although it is not as straightforward.

Let us denote by $\mu_i = \frac{1}{K} y^T \hat{y}$ the (hyper-)margin of instance i. The larger it is, the lower the error. Equivalently, a high negative margin $-\mu_i$ reflects a high error.

Log loss. Now let us consider what $\log \varepsilon_j^{(k)}$ represents. Owing to the definition of the multi-exponential loss, this comes down to a log-sum-exp over $-\mu_i$ of instances reaching node j. Consequently, we have

$$\max_{i \in Z_j^k} \{-\mu_i\} < \log \varepsilon_j^{(k)} = \log \sum_{i \in Z_j^k} \exp^{-\mu_i} \leq \max_{i \in Z_j^k} \{-\mu_i\} + \log \left| Z_j^k \right| \tag{6.35}$$

The lower bound is neared when one negative margin (roughly speaking one "misclassification") dominates the other. This echoes the tendency of the exponential loss to discard most samples in the presence of a badly classified one.

On the other hand, the upper bound is reached when all the margins are similar. In such a case, the size of the node (as measured by the number of instances reaching it) may or may not play a role, which is also influenced by its position. High in the tree (*i.e.* near the root), $\left| Z_j^k \right|$ is supposedly large and its influence will extend to a larger range of margin values. On a relatively balanced tree (which is encouraged by its induction relying on concave uncertainty measures), the size of the node will diminish exponentially fast with the depth, suggesting that the log of the size will quickly vanish when descending down the tree.

Overall, $\log \varepsilon_j^{(k)}$ is tightly bounded by the maximum margin, except on a few occasions such as near the root node.

The weight. The weight can be rewritten as

$$w_j^{(k)} = \frac{K-1}{K} \sum_{l=1}^{K} \log \frac{\varepsilon_j^{[t](k)}}{\varepsilon_j^{[t](l)}} \tag{6.36}$$

$$= (K-1) \left(\log \varepsilon_j^{(k)} - \frac{1}{K} \sum_{l=1}^{K} \log \varepsilon_j^{(l)} \right) \tag{6.37}$$

$$= (K-1) \left(\frac{1}{K} \sum_{l}^{K} \mu^{(l)} - \mu^{(k)} \right) + O\left(|Z_j|\right) \tag{6.38}$$

where $\mu^{(k)} = \min_{i \in Z_j^k} \{\mu_i\}$.

Given our previous observations regarding the value of $\log \varepsilon_j^{(k)}$, we can conclude that, bar a few situations, the weight is determined as a constant (*i.e.* $K-1$) times the maximum margin along each dimension, centered beforehand (*i.e.* $O\left(|Z_j|\right)$ is negligible).

The margin. Therefore, the margin for instances (say i) of class k reaching node j, after its addition to the model, is

$$\frac{1}{K} y_i^T \left(\hat{y}(x_i) + w_j\right) = \mu_i + \frac{1}{K-1} w_j^{(k)} \tag{6.39}$$

$$= \mu_i + \left(\left(\frac{1}{K} \sum_{l}^{K} \mu^{(l)} - \mu^{(k)} \right) + O\left(|Z_j|\right) \right) \tag{6.40}$$

$$\approx \left(\mu_i - \mu^{(k)} \right) + \frac{1}{K} \sum_{l}^{K} \mu^{(l)} \tag{6.41}$$

Recall that the margin is negative for a misclassified sample. In the context of this discussion, we therefore expect $\mu^{(l)}$ ($l = 1, \ldots, K$) to be negative. Overall, the margin of instances belonging to classes whose worst margin are worse than the average worst margin will have their margin widened, while the other will end up with smaller margins. This is a consequence of the exponential loss giving more and more weight the more an instance is misclassified. In conclusion, the loss is reduced by spreading the margin to suit the exponential loss.

Loss reduction. In the end, the loss is reduced by an amount of

$$\sum_{i=1}^{n} e^{-\frac{1}{K}y_i^T\left(\hat{y}(x_i)+w_j z_j(x_i)\right)} - \sum_{i=1}^{n} e^{-\frac{1}{K}y_i^T \hat{y}(x_i)} \tag{6.42}$$

$$= \sum_{k=1}^{K} \sum_{i \in Z_j^l} \left(e^{-\frac{1}{K-1}w_j^{(k)}} - 1\right) \ell_{\exp}\left(y_i, \hat{y}(x_i)\right) \tag{6.43}$$

$$= \sum_{k=1}^{K} \sum_{i \in Z_j^l} \left(\frac{\left(\prod_{l=1}^{K} \varepsilon_j^{(l)}\right)^{\frac{1}{K}}}{\varepsilon_j^{(k)}} - 1 \right) \ell_{\exp}\left(y_i, \hat{y}(x_i)\right) \tag{6.44}$$

$$= \sum_{k=1}^{K} \sum_{i \in Z_j^l} \frac{\ell_{\exp}\left(y_i, \hat{y}(x_i)\right) \left(\left(\prod_{l=1}^{K} \varepsilon_j^{(l)}\right)^{\frac{1}{K}} - \varepsilon_j^{(k)} \right)}{\sum_{i \in Z_j^l} \ell_{\exp}\left(y_i, \hat{y}(x_i)\right)} \tag{6.45}$$

Thus, the reduction (for a class k) is a weighted sum of how $\varepsilon_j^{(k)}$ differs from the geometrical averages of ε_j, and the weights correspond to the current loss for each class.

6.3.3.2 From GIFs to probabilities

Probabilities can be derived through a softmax. More precisely, we have the following proposition.

Proposition 6.3.3. *Posterior probabilities of an example x belonging to class k can be derived as*

$$\hat{p}_{[t]}^{(k)}(x) = \frac{\exp\left(\frac{1}{K-1}\hat{y}_{[t]}^{(k)}(x)\right)}{\sum_{l=1}^{K} \exp\left(\frac{1}{K-1}\hat{y}_{[t]}^{(l)}(x)\right)} \tag{6.46}$$

$$= \overset{(k)}{\text{softmax}}\left(\frac{1}{K-1}\hat{y}_{[t]}(x)\right) \tag{6.47}$$

Proof. As with Adaboost, this is motivated by looking at the population minimizer $\mathcal{R}_{\mathcal{X},\mathcal{Y}}(\hat{y}; \ell_{\exp}) = \mathbb{E}_{\mathcal{X},\mathcal{Y}}\{\ell_{\exp}(y, \hat{y}(x))\}$. Firstly, notice that $\hat{y}$ can minimize the pointwise expected risk $\mathcal{R}_{\mathcal{Y}|x} = \mathbb{E}_{\mathcal{Y}|x}\{\ell_{\exp}(y, \hat{y}(x))\}$ at each x. This comes down to

$$\mathcal{R}_{\mathcal{Y}|x} = \sum_{l=1}^{K} \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \exp\left(-\frac{1}{K}y^T\hat{y}(x)\right) \tag{6.48}$$

$$\frac{\partial}{\partial \hat{y}^{(k)}} \mathcal{R}_{\mathcal{Y}|x} = 0 \tag{6.49}$$

$$= \sum_{l=1}^{K} \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \exp\left(-\frac{1}{K} y^T \hat{y}(x)\right) \left(-\frac{1}{K} y^{(k)}\right) \tag{6.50}$$

$$= \sum_{l=1}^{K} \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(l)}(x)\right) \left(-\frac{1}{K} y^{(k)}\right) \tag{6.51}$$

$$= -\frac{1}{K} \mathbb{P}(\mathcal{Y}^{(k)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(k)}(x)\right) + \frac{1}{K(K-1)} \sum_{l=1, l\neq k}^{K} \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(l)}(x)\right) \tag{6.52}$$

$$= -\frac{1}{K-1} \mathbb{P}(\mathcal{Y}^{(k)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(k)}(x)\right) + \frac{1}{K(K-1)} \sum_{l=1}^{K} \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(l)}(x)\right) \tag{6.53}$$

$$\iff \text{cst} = \frac{1}{K-1} \mathbb{P}(\mathcal{Y}^{(k)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(k)}(x)\right) \tag{6.54}$$

The derivation follows step by step the derivation of the optimal solution. Since the same reasoning applies for all k, all right-hand-side equivalents of the last lines are equal to the same constant, leading to

$$\mathbb{P}(\mathcal{Y}^{(k)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(k)}(x)\right) = \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \exp\left(-\frac{1}{K-1} \hat{y}^{(l)}(x)\right) \tag{6.55}$$

$$\sum_{l=1}^{K} \mathbb{P}(\mathcal{Y}^{(k)} = 1|x) \frac{\exp\left(-\frac{1}{K-1} \hat{y}^{(k)}(x)\right)}{\exp\left(-\frac{1}{K-1} \hat{y}^{(l)}(x)\right)} = \sum_{l=1}^{K} \mathbb{P}(\mathcal{Y}^{(l)} = 1|x) \tag{6.56}$$

$$\mathbb{P}(\mathcal{Y}^{(k)} = 1|x) \sum_{l=1}^{K} \frac{\exp\left(\frac{1}{K-1} \hat{y}^{(l)}(x)\right)}{\exp\left(\frac{1}{K-1} \hat{y}^{(k)}(x)\right)} = 1 \tag{6.57}$$

$$\frac{\mathbb{P}(\mathcal{Y}^{(k)} = 1|x)}{\exp\left(\frac{1}{K-1} \hat{y}^{(k)}(x)\right)} \sum_{l=1}^{K} \exp\left(\frac{1}{K-1} \hat{y}^{(l)}(x)\right) = 1 \tag{6.58}$$

$$\iff \mathbb{P}(\mathcal{Y}^{(k)} = 1|x) = \frac{\exp\left(\frac{1}{K-1} \hat{y}^{(k)}(x)\right)}{\sum_{l=1}^{K} \exp\left(\frac{1}{K-1} \hat{y}^{(l)}(x)\right)} \tag{6.59}$$

□

In the case of a unit learning rate ($\lambda = 1$) and a single tree ($T = 1$), the probabilities thus derived coincide with the ones the underlying tree would provide (see Section 6.3.4.2).

6.3.4 GIF with a single tree

In the case of a single tree ($T = 1$) and a unit learning rate ($\lambda = 1$), GIF predictions coincide with the ones the underlying tree would provide. This is due to the fact that, when examining the weight to give to node j at stage t, the prediction of stage $t-1$ relates solely to the parent node π_j of j. It is thus independent of t and is also the same for all instances reaching that node. This is no longer the case with multiple trees because of the global optimization which is necessary to take advantage of the inter-tree redundancies.

We will adopt the following slight change in notation:

$$\hat{y}_j = \hat{y}_{(\pi_j)} + w_j \tag{6.60}$$

meaning that the prediction associated to any object reaching node j is the weight of j plus the prediction associated to its parent π_j. For the root, we set $\hat{y}_{(\pi_1)} = 0$ (null prediction for its non-existent parent).

6.3.4.1 Regression

In regression (under the ℓ_2 norm), the tree prediction Tr_j of any leaf j is the average of the learning set's outputs reaching that node: $Tr_j = \frac{1}{|Z_j|} \sum_{i \in Z_j} y_i$. With a single tree, a GIF would make the same prediction as the tree. More formally, we have the following proposition.

Proposition 6.3.4. *In regression (under the ℓ_2 norm) and with a single tree, the prediction of a sample x reaching node j in a GIF is*

$$\hat{y}_j(x) = \frac{1}{|Z_j|} \sum_{i \in Z_j} y_i \tag{6.61}$$

Proof. The prediction of any sample reaching node j is

$$\begin{aligned} \hat{y}_j(x) &= \hat{y}_{(\pi_j)}(x) + w_j && (6.62)\\ &= \hat{y}_{(\pi_j)}(x) + \frac{1}{|Z_j|} \sum_{i \in Z_j} \left(y_i - \hat{y}_{(\pi_j)}(x) \right) && (6.63)\\ &= \hat{y}_{(\pi_j)}(x) + \frac{1}{|Z_j|} \sum_{i \in Z_j} (y_i) - \hat{y}_{(\pi_j)}(x) && (6.64)\\ &= \frac{1}{|Z_j|} \sum_{i \in Z_j} y_i && (6.65) \end{aligned}$$

The first step is how the additive model is built. The second is the optimal weight value of node j derived in Equation 6.9, the third step is due to the fact that the prediction at π_j is constant since there is only one tree. □

6.3.4.2 Classification

In classification, a decision tree associates to a sample x reaching a node j the vector of probability representing the relative frequencies of sample from each class reaching that node. In order to have the same prediction as the underlying tree, we must demonstrate that the probability of being in class k associated to node j will be $\frac{\left|Z_j^{(k)}\right|}{|Z_j|}$.

Proposition 6.3.5. *In classification and with a single tree, the probabilistic prediction of a sample x reaching node j in a GIF is*

$$\hat{p}_j^{(k)}(x) = \frac{\left|Z_j^{(k)}\right|}{|Z_j|} \tag{6.66}$$

Proof. Under the zero-sum constraint, we have for node j

$$\exp\left(\frac{1}{K-1} w_j^{(k)}\right) = \frac{1}{c_k} \varepsilon^{(k))}_{(\pi_j} \tag{6.67}$$

$$= \frac{1}{c_k} \sum_{i \in Z_j^{(k)}} \exp\left(-\frac{1}{K-1} \hat{y}^{(k)}_{(\pi_j)}\right) \tag{6.68}$$

$$= \frac{1}{c_j} \left|Z_j^{(k)}\right| \exp\left(-\frac{1}{K-1} \hat{y}^{(k)}_{(\pi_j)}\right) \tag{6.69}$$

$$\exp\left(\frac{1}{K-1} \hat{y}_j^{(k)}\right) = \exp\left(\frac{1}{K-1} \hat{y}^{(k)}_{(\pi_j)}\right) \exp\left(\frac{1}{K-1} w_j^{(k)}\right) \tag{6.70}$$

$$= \frac{1}{c_j} \left|Z_j^{(k)}\right| \tag{6.71}$$

$$\hat{p}_j^{(k)} = \frac{\exp\left(\frac{1}{K-1} \hat{y}_j^{(k)}\right)}{\sum_{k=1}^{K} \exp\left(\frac{1}{K-1} \hat{y}_j^{(k)}\right)} = \frac{\left|Z_j^{(k)}\right|}{|Z_j|} \tag{6.72}$$

where $c_k = \left(\prod_{l=1}^{K} \varepsilon_j^{(l)}\right)^{\frac{1}{K}}$ is a constant. The first equality is a consequence of the value of $w_j^{(l)}$ (Equation 6.19). The second is a due to the definition of $\varepsilon^{(k)}$ (Equation 6.20). The third is a consequence of having a single tree: the prediction of the parent is the same for all instances. The fourth line uses the definition of $\hat{y}$. The rest follows from Proposition 6.3.3. □

Notice that, in both regression and classification, the equivalence also holds for an internal node: the prediction is the one the tree would have yielded were that node a leaf.

6.4 Empirical analysis

This section is concerned with the empirical validation of GIFs. Section 6.4.1 portrays how GIF fare in regression and classification. How the hyper-parameters influence the results is the topic of Section 6.4.2. Comparisons then investigated with local methods (6.4.3), boosting (6.4.4) and post-pruning methods (6.4.5).

6.4.1 Regression and classification

In this section, we explore how GIFs behave empirically. All results presented in this section were obtained by repeated holdout evaluation and are averaged over ten different learning sample/testing sample splits.

The experiment are conducted with several regression and (binary and multi-class) classifications datasets whose main characteristics are summarized in Table 6.1. Abalone, CT slice, California data housing (Cadata), Musk2, Vowel and Letter come from the UCI Machine Learning Repository (Blake and Merz, 1998). Ringnorm, Twonorm and Waveform are described in Breiman et al. (1998). Hwang F5 comes from the DELVE repository [2]. The noise parameter of the Friedman1 dataset (Friedman, 1991) has been set to 1. Hastie is described in Friedman, Hastie, and Tibshirani (2001b). Out of the 500 features of Madelon (Guyon et al., 2004), 20 are informative and 50 are redundant; the others are noise. MNIST8vs9 is the MNIST dataset (LeCun et al., 1998b) of which only the 8 and 9 digits have been kept. Binary versions of the MNIST, Letter and Vowel datasets have been created as well by grouping the first half and second half classes together. All the dataset are accessible via the GIF repository.

Our first experiment was to test the GIF against the Extremely randomized trees (ET). To get an estimate of the average number of nodes per tree, we first computed ten forests of 1000 fully-developed ET. We then examined how GIF compared to ET for 1% and 10% of the original budget. For GIF, these values were directly used as budget constraints. For ET, we built forests of 10 ($\text{ET}_{1\%}$) and 100 ($\text{ET}_{10\%}$) trees. The supplementary materials include further comparisons with three other local pre-pruning baselines, focusing more on the top of the trees. As these baselines tend to perform poorly, we focus our comparison below to the $\text{ET}_{1\%}$ and $\text{ET}_{10\%}$ baselines.

The extremely randomized trees were computed with version 0.18 of Scikit-Learn (Pedregosa et al., 2011) with the default parameters proposed in Geurts, Ernst, and Wehenkel (2006). In particular, the trees are fully-developed and the number of features examined at each split is $\sqrt{p}$ in classification and p in regression, where p is the initial number of features. For GIF, we started with $T = 1000$ stumps, a learning rate of $\lambda = 10^{-1.5}$ and $CW = 1$. The underlying tree building algorithm is ET with no restriction regarding the depth and $\sqrt{p}$ features are examined for each split, in both classification and regression. We will refer to this parameter setting as the default one.

[2]https://www.cs.toronto.edu/~delve/

TABLE 6.1: Characteristics of the datasets. n is the learning sample size, TS stands for testing set, and p is the number of features. $\overline{\text{nodes}}$ corresponds to the average number of nodes of a fully-developed extra-tree; when two numbers are present, the second corresponds to the binary version of the dataset.

DATASET	n	$\|TS\|$	p	# CLASSES	$\overline{\text{NODES}}$
FRIEDMAN1	300	2000	10	-	535
ABALONE	2506	1671	10	-	3811
CT SLICE	2000	51500	385	-	3995
HWANG F5	2000	11600	2	-	3996
CADATA	12384	8256	8	-	24038
RINGNORM	300	7100	20	2	167
TWONORM	300	7100	10	2	164
HASTIE	2000	10000	10	2	1606
MUSK2	2000	4598	166	2	449
MADELON	2200	2200	500	2	1082
MNIST8VS9	11800	1983	784	2	1375
WAVEFORM	3500	1500	40	3	2222
VOWEL	495	495	10	11	456/223
MNIST	50000	10000	784	10	20467/13678
LETTER	16000	4000	8	26	8219/5453

TABLE 6.2: Average mean square error at 1% and 10% budgets ($p_e = \sqrt{p}$, $\lambda = 10^{-1.5}$, $m = 1000$, $CW = 1$).

DATASET	$\text{ET}_{100\%}$	$\text{ET}_{10\%}$	$\text{GIF}_{10\%}$	$\text{ET}_{1\%}$	$\text{GIF}_{1\%}$
FRIEDMAN1	4.89 ± 0.23	5.02 ± 0.22	2.37 ± 0.24	5.87 ± 0.27	3.26 ± 0.29
ABALONE	4.83 ± 0.21	4.87 ± 0.21	5.20 ± 0.21	5.29 ± 0.27	4.74 ± 0.23
CT SLICE	19.32 ± 1.69	19.62 ± 1.69	19.31 ± 0.61	23.84 ± 1.85	36.48 ± 1.32
HWANG F5 $\times 10^{-2}$	8.20 ± 0.11	8.25 ± 0.11	8.58 ± 0.10	8.67 ± 0.12	6.91 ± 0.04
CADATA $\times 10^{-2}$	25.45 ± 0.65	25.71 ± 0.62	21.76 ± 0.66	28.39 ± 0.97	24.08 ± 0.65

Regression was handled with the square loss. For classification, we tested two methods. The first one is a one-vs-rest approach by allocating one output per class with the square loss. The second method was to use the trimmed exponential loss with a saturation $\theta = 3$. The results are reported in Tables 6.2 and 6.3.

Regression. As we can see from Table 6.2, this default set of parameters performs quite well under heavy memory constraint (*i.e.* a budget of 1%). $\text{GIF}_{1\%}$ outperforms significantly $\text{ET}_{1\%}$ four times out of five. Moreover, on those four datasets, $\text{GIF}_{1\%}$ is able to beat the original forest with only 1% of its node budget. The mild constraint case (*i.e.* a budget of 10%) is more contrasted. On Friedman1, California data housing and CT Slice, $\text{GIF}_{10\%}$ outperforms $\text{ET}_{10\%}$. For both Abalone and Hwang, $\text{GIF}_{10\%}$ overfits; in both cases the errors of $\text{GIF}_{1\%}$ were better than at 10% and, as mentioned, better than $\text{ET}_{100\%}$.

TABLE 6.3: Error rate (%) at 1% and 10% budgets ($p_e = \sqrt{p}$, $\lambda = 10^{-1.5}$, $m = 1000$, $CW = 1$). $\text{GIF}_{SQ,\cdot}$ relates to the multi-output square loss. $\text{GIF}_{TE,\cdot}$ relates to the trimmed exponential loss with $\theta = 3$. The six first datasets are binary classification. The last three are multiclass. The three in the middle are their binary versions.

DATASET	$\text{ET}_{10\%}$	$\text{GIF}_{SQ,10\%}$	$\text{GIF}_{TE,10\%}$	$\text{ET}_{100\%}$
RINGNORM	3.28 ± 0.41	4.05 ± 0.45	3.17 ± 0.34	2.91 ± 0.40
TWONORM	3.54 ± 0.18	3.50 ± 0.24	3.35 ± 0.22	3.13 ± 0.13
HASTIE	11.78 ± 0.56	10.33 ± 0.41	7.38 ± 0.29	10.30 ± 0.46
MUSK2	3.70 ± 0.37	3.41 ± 0.34	3.14 ± 0.34	3.65 ± 0.40
MADELON	12.43 ± 0.77	9.18 ± 0.83	8.03 ± 0.60	9.75 ± 0.75
MNIST8VS9	1.06 ± 0.23	0.86 ± 0.24	0.76 ± 0.16	0.99 ± 0.23
BIN. VOWEL	2.28 ± 1.20	2.81 ± 1.17	2.24 ± 1.19	1.96 ± 1.04
BIN. MNIST	2.04 ± 0.21	1.76 ± 0.15	1.59 ± 0.15	1.92 ± 0.16
BIN. LETTER	2.00 ± 0.17	2.44 ± 0.25	2.28 ± 0.19	1.80 ± 0.20
WAVEFORM	14.47 ± 0.93	14.17 ± 0.62	14.51 ± 0.67	13.95 ± 0.58
VOWEL	6.08 ± 1.13	7.31 ± 1.18	15.90 ± 1.35	5.92 ± 1.29
MNIST	2.87 ± 0.19	2.26 ± 0.17	4.05 ± 0.25	2.63 ± 0.18
LETTER	2.75 ± 0.17	2.82 ± 0.19	9.07 ± 0.53	2.53 ± 0.16
	$\text{ET}_{1\%}$	$\text{GIF}_{SQ,1\%}$	$\text{GIF}_{TE,1\%}$	
RINGNORM	7.43 ± 0.55	5.35 ± 0.65	4.30 ± 0.51	
TWONORM	8.00 ± 0.57	3.91 ± 0.39	3.92 ± 0.31	
HASTIE	20.38 ± 0.56	7.64 ± 0.50	6.76 ± 0.42	
MUSK2	4.22 ± 0.37	7.40 ± 0.38	6.65 ± 0.28	
MADELON	23.91 ± 1.17	12.55 ± 0.83	12.40 ± 0.76	
MNIST8VS9	1.58 ± 0.31	2.10 ± 0.35	1.53 ± 0.31	
BIN. VOWEL	4.18 ± 1.70	12.28 ± 2.00	11.92 ± 2.03	
BIN. MNIST	3.37 ± 0.17	3.24 ± 0.20	2.76 ± 0.18	
BIN. LETTER	3.59 ± 0.35	7.57 ± 0.38	6.65 ± 0.24	
WAVEFORM	19.11 ± 0.57	13.26 ± 0.56	14.78 ± 0.81	
VOWEL	11.74 ± 1.71	22.91 ± 2.03	36.30 ± 2.62	
MNIST	4.94 ± 0.21	3.92 ± 0.25	5.68 ± 0.31	
LETTER	5.34 ± 0.27	8.10 ± 0.55	19.87 ± 0.77	

Classification. An interesting conclusion can be reached from Table 6.3: the number of classes should guide the choice of loss. In the binary case, the trimmed exponential works well. At 1%, it loses on Musk2, and the binarized version of Vowel and Letter to $\text{ET}_{1\%}$. At 10%, it only loses on binary Vowel, where it closes the gap somewhat.

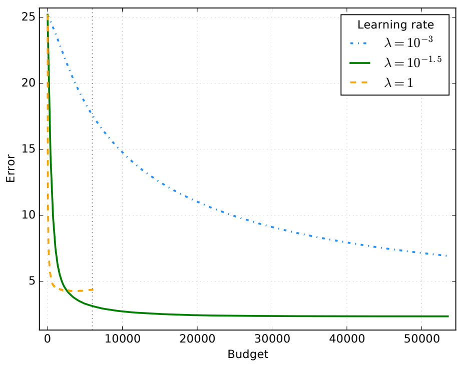
When it comes to multiclassification, however, the trimmed exponential seems to suffer. The multi-output square loss version is sometimes able to outperform the ET version. This is the case of both Waveform and MNIST at 1% and of MNIST at 10%.

The binary versions of Vowel, and MNIST indicate that GIF at 10% struggles much more with the number of classes than with the the dimensionality of the problem and/or the learning sample size.

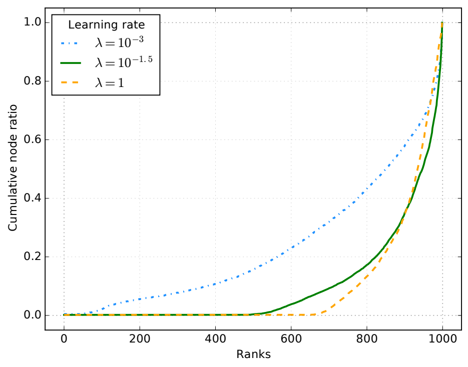
Interestingly, GIF's performance on Madelon with both losses are better than the base ET version. This suggests that GIF is well capable of handling irrelevant features.

Needless to say that this default parameter setting, although performing well on average, is not optimal for all datasets. For instance, on CT slice at 1%, we can reach 20.54 ± 0.76 by enlarging the candidate window size to 10. For the trimmed exponential loss, with $\lambda = 10^{-1}$ at 1%, we can reach 3.74 ± 0.31 on Twonorm and 3.54 ± 0.3 on Musk2.

6.4.2 Influence of the hyper-parameters



(A) Average test set error with respect to the budget B ($CW = 1$, $p_e = \sqrt{10}$, $m = 1000$).



(B) Cumulative node distribution with respect to the size-ranks ($CW = \infty$, m=$\sqrt{10}$, $T = 1000$, $B = 10\%$).

FIGURE 6.3: Friedman1: effect of some hyper-parameters

Learning rate. Figure 6.3a depicts a typical evolution of the error with the budget for different learning rates in the case of Friedman1 (the budget maxes out at 59900 nodes, corresponding to 10%). A unit learning rate will usually decrease the test set error rapidly but will then either saturate or overfit. Too small a learning rate (*e.g.* 10^{-3}) will prevent the model from reaching its minimum in the alloted budget. The learning rate also influences the forest shape, provided the candidate window size is large enough. Figure 6.3b portrays the cumulative node distribution with respect to the size-ranks of the trees for $CW = \infty$, meaning that f(x) is the ratio of nodes of the x/T smallest trees. We can see that, for the smallest learning rate, 80% of the smallest trees account for approximately 43% of the nodes. At the same stage, only 17% and 13% of the nodes are covered for the average and biggest learning rates, respectively.

Number of features. Table 6.4 shows how the error varies at 10% for $CW = 1$ with respect to both the learning rate λ and m, the number of features examined for a split, in the case of CT slice and Musk2, two datasets with many features. Interestingly, the error tends to vary continuously over those two parameters. On both datasets, it appears that the choice of learning rate (global parameter) is more critical than the number of features (local parameter). The optimal number of features remains problem-dependent, though.

TABLE 6.4: Average test set error with respect to p_e and λ ($CW = 1$, $m = 1000$, $B = 10\%$; $\theta = 3$). In bold is $p_e = \sqrt{p}$.

CT slice: mean square error

$p_e \setminus \lambda$	$10^{-2.5}$	10^{-2}	$10^{-1.5}$	10^{-1}	$10^{-0.5}$
19	27.28	20.34	19.31	21.97	29.82
38	25.78	19.51	18.63	20.88	27.62
96	25.53	19.74	18.79	20.68	26.64
192	26.55	20.96	19.92	21.62	26.87
288	28.20	22.43	20.91	22.31	27.64
385	31.42	25.04	23.11	24.17	29.56

Musk2: error rate (%)

m \ λ	$10^{-2.5}$	10^{-2}	$10^{-1.5}$	10^{-1}	$10^{-0.5}$
12	5.13	3.74	3.14	2.90	2.86
16	5.00	3.67	3.11	2.91	2.85
41	4.50	3.39	3.00	2.93	2.93
83	4.24	3.26	2.92	2.88	2.90
124	4.11	3.20	2.89	2.79	2.75
166	4.11	3.19	2.94	2.84	2.86

TABLE 6.5: Error rate (%) for the trimmed exponential loss ($\theta = 3$, $\lambda = 10^{-1.5}$, $p_e = \sqrt{10}$, $m = 1000$, $B = 10\%$)

DATASET	CW=1	CW=10
WAVEFORM	14.51 ± 0.67	14.05 ± 0.82
VOWEL	15.90 ± 1.35	10.87 ± 1.61
MNIST	4.05 ± 0.25	3.66 ± 0.31
LETTER	9.07 ± 0.53	5.88 ± 0.32

Candidate window size. Figure 6.4 illustrates the influence of the candidate window size on both the error and the fitting time for several datasets with $\lambda = 10^{-1.5}$, $p_e = \sqrt{p}$, $m = 1000$ and a budget=10%. Firstly, the linear dependence of the window size on the building time is clearly visible. More interestingly, the smaller window size ($CW = 1$) performs best on all four datasets. All in all, this seems to be a good regularization mechanism, allowing for a dramatic decrease of computing times while ensuring better predictions.

Although this is representative of the regression and binary classification problems, this is not exactly the case of multiclassification, where increasing CW over 1 might improve performance slightly (see Table 6.5).

Number of trees. The initial number of trees is an intricate parameter, as it impacts model predictions, the fitting time and the shape of the forest.

Table 6.6 focuses on the errors with $m = \sqrt{p}$ and $\lambda = 10^{-1.5}$. Unsurprisingly, the models perform badly when it has only 10 trees at its disposal; this leaves only little room for the learning algorithm to optimize globally. The more trees is not always better, however. When the candidate window is infinitely large, this might be due to overfitting: there are so many candidates

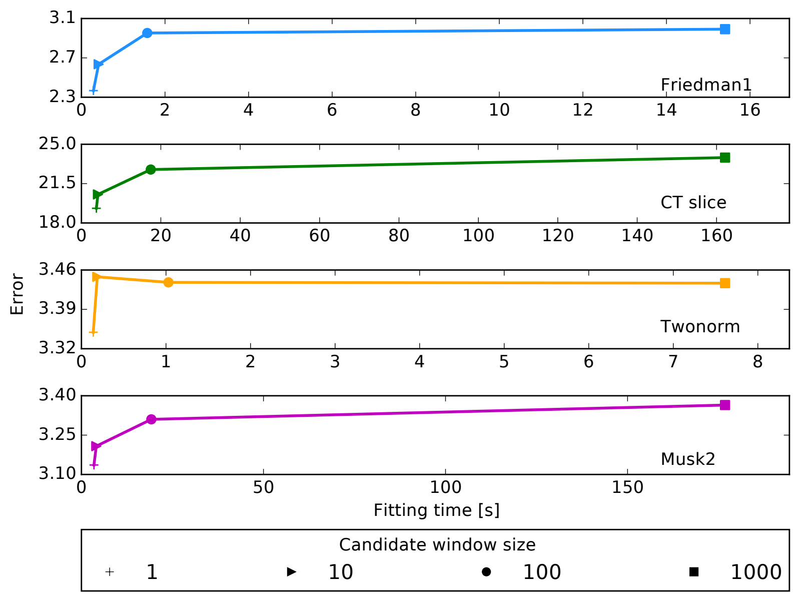


FIGURE 6.4: Average test set error (MSE for Friedman1 and CT slice, error rate (%) for Twonorm and Musk2) and fitting time with respect to CW ($\lambda = 10^{-1.5}$, $p_e = \sqrt{p}$, $m = 1000$, $B = 10\%$; $\theta = 3$).

TABLE 6.6: Test set error with respect to the initial number of trees m ($p_e = \sqrt{p}$, $\lambda = 10^{-1.5}$, same budget $B = 10\%$; $\theta = 3$).

Friedman1: mean square error

	Friedman1		Twonorm	
	Mean square error		Misclassification rate (%)	
T	CW=1	CW=∞	CW=1	CW=∞
10	7.88 ± 0.64	7.62 ± 0.71	7.47 ± 0.73	7.05 ± 0.29
100	3.31 ± 0.41	3.60 ± 0.35	3.44 ± 0.16	3.52 ± 0.13
1000	2.37 ± 0.24	3.05 ± 0.29	3.35 ± 0.22	3.43 ± 0.23
10000	2.26 ± 0.20	3.18 ± 0.28	3.53 ± 0.25	3.87 ± 0.32

to choose from that over-optimization hurts the model. When the window size is 1, this is more directly linked to the forest shape.

Table 6.7 holds the normalized entropy of the node distribution across trees for Friedman1. By "normalized", we mean that the entropy was divided by its maximal possible value $\log_2 T$ and then multiplied by 100. Only one value is reported for the case $CW = 1$ as the forest has always the same shape, whatever the learning rate λ. The evolution of the entropy for a fix number of trees when $CW = \infty$ has already been commented on (see Figure 6.3b). It is rendered more obvious when the initial number of trees is larger, however, meaning that GIF is able to exploit the greater freedom offered by the additional trees. When $CW = 1$, the distribution is much closer to being uniform (entropy close to 100) than when the learning algorithm can adapt the forest shape. If this shape does not agree with the data, the model might perform less well. Nevertheless, as we saw, $CW = 1$ yields better result on all but the multiclass problems, and $T = 1000$ seems to be adequate in average.

TABLE 6.7: Friedman1: average normalized node distribution entropy with respect to m and λ ($p_e = \sqrt{p}$, same budget $B = 10\%$).

	CW=1	CW=∞		
T \ λ	*	10^{-3}	$10^{-1.5}$	1
100	99.89	99.84	99.24	98.48
1000	98.15	94.49	87.32	83.72
10000	97.20	89.12	76.23	68.99

TABLE 6.8: Friedman1: fitting time (seconds) with respect to m and λ ($p_e = \sqrt{p}$, same budget $B = 10\%$).

	CW=1	CW=∞	
T \ λ	*	10^{-3}	1
100	0.34 ± 0.07	0.35 ± 0.07	0.32 ± 0.07
1000	0.59 ± 0.12	3.84 ± 0.18	2.78 ± 0.54
10000	1.55 ± 0.02	25.95 ± 1.05	20.69 ± 2.92

The number of trees also impacts the learning time, as depicted by Table 6.8. The linear increase in computing time in the case of $CW = \infty$ is due to the global optimization of the chosen node that must run through all the candidates. In the case of $CW = 1$, the computing time is almost not burdened by the number of trees. The slight increase is actually related to the forest shape: since the distribution of node tends to be more uniform, the algorithm must run through more examples while optimizing the weights (higher part of the trees).

6.4.3 Comparison with local baseline algorithms

We have tested three deepening algorithm for decision forest relying on non-global metrics, meaning that the choice of the best candidate is not made according to how well the forest, as a whole, performs. These algorithms share that the final model is exactly a sub-forest of the un-pruned forest: contrary to GIF, no internal weights are fitted and the predictions at the leaves are the usual tree predictions.

Breadth first deepening. This variant consist in adding the nodes level after level, from left to right, producing a heaped forest. As a consequence, all trees have the same (order of) height, implying that the forest can be quite wide but usually shallow.

Random deepening. This variant consist in first choosing a tree and then choosing one of its leaves to transform to a decision nodes. Both choices are made uniformly at random so that the trees are expected to have approximately the same number of nodes. The depth, however, might vary significantly.

TABLE 6.9: Average mean square error for local baselines at 1% and 10% budgets ($m = 1000$, $p_e = p$).

DATASET	BREADTH FIRST$_{10\%}$	RANDOM$_{10\%}$	BEST FIRST$_{10\%}$
FRIEDMAN1	6.02 ± 0.28	6.80 ± 0.34	15.00 ± 0.39
ABALONE	4.72 ± 0.23	4.77 ± 0.23	6.82 ± 0.33
CT SLICE	30.39 ± 1.90	36.19 ± 1.84	310.87 ± 4.79
HWANG F5 $\times 10^{-2}$	6.73 ± 0.07	6.83 ± 0.06	56.57 ± 6.03
CADATA $\times 10^{-2}$	29.24 ± 0.73	31.08 ± 0.74	75.23 ± 0.95
	BREADTH FIRST$_{1\%}$	RANDOM$_{1\%}$	BEST FIRST$_{1\%}$
FRIEDMAN1	11.73 ± 0.46	12.52 ± 0.47	15.29 ± 0.42
ABALONE	5.42 ± 0.27	5.55 ± 0.27	6.82 ± 0.33
CT SLICE	82.19 ± 2.41	97.24 ± 1.90	313.84 ± 4.64
HWANG F5 $\times 10^{-2}$	8.52 ± 0.24	13.17 ± 0.44	56.60 ± 6.07
CADATA $\times 10^{-2}$	43.40 ± 1.18	47.47 ± 1.02	75.48 ± 0.95

Best first deepening. This variant consist in choosing, among all leaves which could be turned into a internal node, the one which reduces the local uncertainty (as defined in 3.3.10) the most.

Since the fraction of learning instances reaching the candidate is accounted for in the reduction of impurity, this approach will naturally favor higher nodes in the trees.

Experiment. We conducted the same experiment as for GIF: the three algorithms were tested on ten folds with different learning sample/testing sample splits and were subjected to the 1% and 10% node constraints. We started with a pool of $T = 1000$ roots and no restriction was imposed regarding the depth. All of the $p_e = p$ the features were examined in regression and $p_e = \sqrt{p}$ in classification, as suggested in Geurts, Ernst, and Wehenkel, 2006. Table 6.9 holds the average mean square error for the five regression problems and Table 6.10 holds the average misclassification rate for the classification problems.

Regression. The trend is quite clear: both at 1% and 10%, the breadth first algorithm is the best and the best first is (largely) the worst. There are two instances where the local baselines are able to beat GIF: on Abalone and Hwang F5 at 10%. Interestingly, these are the same cases on which GIF was beaten by a small forest of Extremely randomized trees. The 10% Hwang F5 case aside, the local baselines always underperform the smaller fully-developed forest. Overall, such variants do not seem adequate for regression.

Classification. In classification, the breadth first and random baselines tend to perform similarly, one beating the other on some problems. Once again, the best first approach seems to be lagging behind on some datasets. At 10%, the local baselines cannot rival with the other methods. Only on Waveform are they able to reach the other performances—a setting where all methods seems to produce close results. At 1%, the breadth first and/or the random methods surpass the ET$_{10\%}$ on Twonorm, Hastie, Madelon and Waveform.

TABLE 6.10: Error rate (%) for local baselines at 1% and 10% budgets ($m = 1000$, $p_e = \sqrt{p}$). The six first datasets are binary classification. The last three are multiclass. The three in the middle are their binary versions.

DATASET	BREADTH FIRST$_{10\%}$	RANDOM$_{10\%}$	BEST FIRST$_{10\%}$
RINGNORM	4.25 ± 1.24	4.08 ± 1.12	8.38 ± 6.94
TWONORM	3.51 ± 0.26	3.53 ± 0.30	5.59 ± 1.85
HASTIE	11.30 ± 1.20	11.18 ± 1.16	21.24 ± 7.11
MUSK2	7.01 ± 0.40	7.63 ± 0.43	15.42 ± 0.23
MADELON	11.68 ± 0.67	11.92 ± 0.65	19.12 ± 1.94
MNIST8VS9	2.20 ± 0.38	2.37 ± 0.39	6.17 ± 0.73
BIN. VOWEL	8.99 ± 1.96	8.85 ± 2.03	16.57 ± 3.02
BIN. MNIST	4.46 ± 0.25	4.91 ± 0.27	21.71 ± 0.30
BIN. LETTER	5.91 ± 0.43	5.71 ± 0.40	26.16 ± 0.86
WAVEFORM	14.74 ± 0.63	14.83 ± 0.76	20.25 ± 2.22
VOWEL	14.26 ± 2.41	13.21 ± 2.33	41.49 ± 5.45
MNIST	4.63 ± 0.27	4.96 ± 0.26	28.54 ± 0.59
LETTER	7.06 ± 0.29	6.39 ± 0.20	36.92 ± 1.80
	BREADTH FIRST$_{1\%}$	RANDOM$_{1\%}$	BEST FIRST$_{1\%}$
RINGNORM	8.94 ± 7.45	8.53 ± 7.04	8.94 ± 7.41
TWONORM	5.91 ± 3.03	6.52 ± 4.28	7.28 ± 4.34
HASTIE	13.92 ± 2.93	14.29 ± 3.20	21.24 ± 7.12
MUSK2	15.42 ± 0.23	15.42 ± 0.23	15.42 ± 0.23
MADELON	16.26 ± 0.97	16.70 ± 1.07	20.14 ± 2.41
MNIST8VS9	4.53 ± 0.48	4.84 ± 0.51	6.67 ± 0.69
BIN. VOWEL	18.73 ± 3.08	19.90 ± 3.71	21.80 ± 4.38
BIN. MNIST	10.09 ± 0.25	11.78 ± 0.32	22.50 ± 0.35
BIN. LETTER	17.91 ± 0.77	18.05 ± 0.78	26.19 ± 0.88
WAVEFORM	16.75 ± 1.26	17.13 ± 1.25	20.45 ± 2.21
VOWEL	42.40 ± 4.33	40.28 ± 4.62	50.44 ± 5.81
MNIST	8.60 ± 0.35	9.76 ± 0.31	29.72 ± 0.61
LETTER	22.11 ± 0.59	20.90 ± 0.55	37.27 ± 1.78

Those datasets correspond to cases where ET was under-performing significantly compared to GIF. All in all, the local baselines are never able to beat GIF, even in the multiclass setting, which is particularly defavorable for GIF. Once again, the conclusion is against the purely local baselines.

We believed the poor performances of the baselines are due to the building mechanism of traditional ensemble methods. Although the trees are built independently and with randomization, there remains an important redundancy between them, which is especially unfavorable to pruning. A global approach is better able to avoid redundancy and can thus better exploit the node budget. This would also explain why the best first variant performs worst in both regression and classification: it is prone at picking redundant nodes, which will usually offer the same kind of impurity reduction.

6.4.4 A preliminary comparison with Boosting

In this section, we carry out a first comparison of GIF with Boosting. To submit Boosting to the budget constraint, we have used stumps as base learners and have made as many trees as were necessary to meet the constraint. We

TABLE 6.11: Test set error (MSE/error rate (%)) for stump least-sqaure Boosting/Adaboost under budget constraints ($\lambda = 10^{-1.5}$).

DATASETS	$B = 10\%$	$B = 1\%$
FRIEDMAN1	4.53 ± 0.23	3.86 ± 0.10
ABALONE	5.17 ± 0.20	4.83 ± 0.20
CT SLICE	82.44 ± 3.80	68.73 ± 1.92
HWANG $\times 10{-}2$	97.88 ± 2.33	88.62 ± 1.73
RINGNORM	5.48 ± 0.55	6.71 ± 0.99
TWONORM	5.09 ± 0.56	5.98 ± 0.47
HASTIE	5.65 ± 0.34	7.10 ± 0.41
MUSK2	2.70 ± 0.37	4.20 ± 0.28
MADELON	11.30 ± 0.68	11.33 ± 0.69

TABLE 6.12: Musk2: fitting/prediction times (seconds). Stump Adaboost versus GIF (trimmed loss with $\theta = 3$, $m = 1000$, $p_e = \sqrt{p}$, $CW = 1$) for $B = 10\%$ and $\lambda = 10^{-1.5}$.

	ADABOOST	GIF
FITTING	399.17 ± 60.91	1.53 ± 0.04
PREDICTION	28.39 ± 5.43	0.31 ± 0.07

have used the same learning rate as for GIF in Table 6.2. Regression has been tackled with least square Boosting (Friedman, Hastie, and Tibshirani, 2001b) and classification with Adaboost (Freund and Schapire, 1995), so that the same losses are used for GIF and Boosting. Scikit-Learn was used as Boosting implementation.

Table 6.11 holds the errors for Boosting at 1% and 10%. In the default setting, GIF beats Boosting on all regression datasets except Abalone where it performs slightly less well. Interestingly, Boosting also overfits on Abalone and Hwang. The situation is more contrasted in classification, where Boosting outperforms GIF on Hastie and Musk2 for both budget constraints. Notice that stumps are not optimal for Hwang and CT slice, where a depth of 2 would yield lower errors of 11.09 ± 0.25 and 8.40 ± 0.19 at 10% and 1% respectively for Hwang and 33.53 ± 1.65 and 36.67 ± 1.36 at 10% and 1% respectively for CT slice. However, this does not change the conclusions regarding the comparison with GIF.

GIF (with $CW = 1$) is faster in both learning and prediction than Boosting, as confirmed in Table 6.12. Firstly, Boosting's base learners are traditional decision trees, which are slower to fit than ET for a given structure. Secondly, Boosting's base learners are shallow and they can thus take less advantage of the partitioning induced by the trees.

Overall, the performances of Boosting and GIF in terms of errors are somewhat similar. Sometimes GIF's extra-layers of regularization, combined with a greater variety of depths pays off and sometimes not. However, GIF is faster in both learning and prediction.

6.4.5 Comparison with post-pruning

In this section, we compare GIFs to a post-pruning method, namely the L1-based compression of Joly et al. (2012). Their method consists in learning the forest with fully-developed trees and then refitting the linear model corresponding to the forest space with a L1 penalty to induce sparsity. More precisely they solve the following program

$$\min_{w} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{M} w_j z_j(x_i) \right)^2 + \mu \sum_{j=1}^{M} |w_j| \tag{6.73}$$

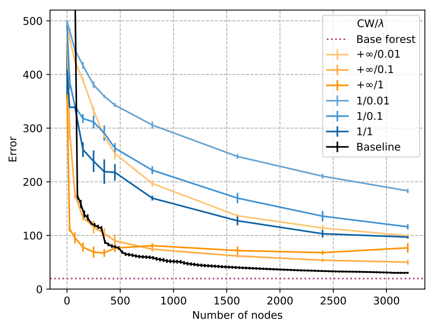
where w_j are the weights associated to the M nodes of the forest and μ is the tunable hyper-parameter controlling the sparsity of the model. The retained nodes are those leading to a node of weight $w_j > 0$. Therefore, the final list of nodes is a superset of the nodes which actually receive a strictly positive weight. μ is a high level knob to play on the compression/accuracy tradeoff which makes it hard to attain the desired compression level without hyper-parameter tuning. Fortunately, there exist efficient algorithms to compute the whole regularization path.

Experimental setup. Since the problem is formulated with the squared loss, we only looked at regression problems. For each dataset, we built a fully-developed forest of 1000 extra-trees with the default hyper-parameters and then computed the whole regularization path of the L1-compression. At each stage, we extracted the number of nodes kept by the algorithm in order to cover the same spectrum with GIFs.

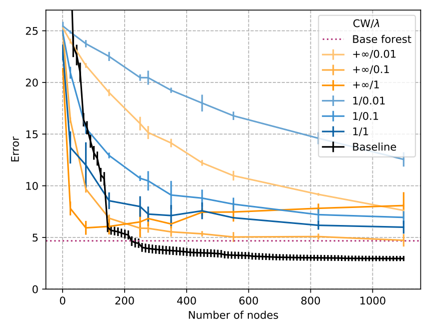
GIFs were used in the exact same conditions (same hyper-parameters for the underlying forest, same train/test splits). We used several pairs of candidate window sizes and learning rate. Our expectations were that for severely constrained cases, more optimization would be needed to take full advantage of the somewhat small node budget, that is a large candidate window size and high learning rate.

Results and discussion. Figure 6.5 displays how GIFs compare to the L1-based post-pruning baseline. The results are unanimous: highly optimized GIFs ($CW = +\infty$, $\lambda = 1$) excel when the budget is very tight, typically under a couple hundreds nodes. With more permissive budget, the ability to select nodes further down the tree via post-pruning seems to provide a crucial advantage.

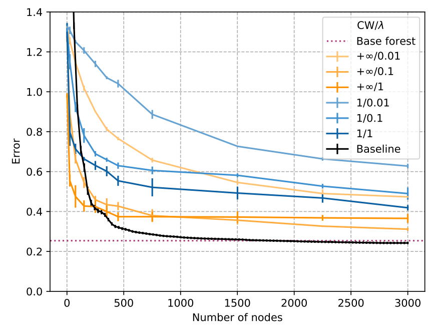
In the longer run, a slightly smaller learning rate ($\lambda = 0.1$) usually proves better. On Abalone, this is absolutely mandatory as we can see overfitting soaring. Interestingly, the lines corresponding to a lesser optimization have not yet plateaued by the end of the budget. This is consistent with our results of the previous section and sheds further light on how to choose hyper-parameters with respect to the node budget.



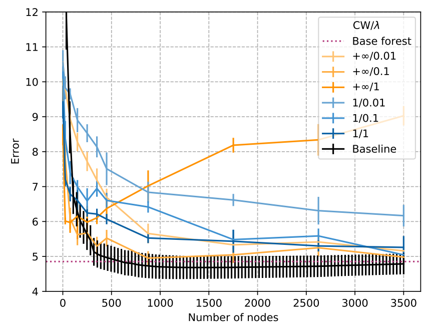
(A) CT_SLICE



(B) FRIEDMAN1



(C) CADATA



(D) ABALONE

FIGURE 6.5: GIFs versus L1-based post-pruning. Base forest correspond to the error of the fully-developed and unpruned forest. Baseline refers to L1-based post-pruning. The other lines correspond to GIFs with different candidate window size (CW) and learning rates (λ).

6.5 Conclusion and perspectives

This chapter discussed the problem of pruning decision forests. Several takes on the problem as well as levers to accomplish it have been discussed and used as lenses to discuss existing work on the matter.

Globally Induced Forest (GIF) were then introduced and discussed at some length. Their goal is to produce lightweight yet accurate tree-based ensemble models by sequentially adding nodes to the model. Being a tree-aware technique, GIFs are able to leverage all levers of compression (redundancy, forest size and tree depth). Contrary to most tree-aware techniques, our method is framed as a pre-pruning method that does not require the *a priori* building of the whole forest.

Several hyper-parameters govern the learning algorithm. We have proposed a set of default parameters which seems to work quite well in average, beating the baselines, under mild and severe memory constraints. Under extreme conditions (Section 6.4.5), the default values should be adapted to allow the learning algorithm to optimize more fully the few nodes it receives.

Needless to say that these parameters can be further fine-tuned if necessary, although this goes against the philosophy of building directly the pruned forest. To counter-balance this, the empirical study was designed to shed some light as to how the hyper-parameters can be tailored to specific needs without resorting to cross-validation.

Another interesting finding of this study is the conclusion that it is usually better not to optimize the choice of nodes by keeping the candidate window small, except in the most extreme setting. In other words, letting the algorithm optimize the forest shape is—surprisingly—harmful. Although it complicates the choice of the initial number of trees, this makes the algorithm extremely fast. Under the most dire budget, increasing the candidate window size bears little impact on the overall running time since the budget is so small.

Overall, the real downside of GIFs are there relatively less convincing results on multi-class problems. Further investigation would need to be conducted to understand exactly what is the cause of the problem. The fact that the problem arises on multi-class problems hints at the proposed trimmed exponential loss, which might be refined to better handle class disappearance down the trees.

A few tweaks to the algorithm could also be investigated. For instance, one could consider introducing both children at the same time at each iteration, or allow for the refitting of the already chosen nodes by leaving them in the candidate list. Alternatively, the final model could refitted globally, possible with the addition of some post-pruning regularization for even smaller models.

A drawback GIFs have compared to post-pruning method is that GIFs do not have the foresight to expand a branch to get an interesting node several level further down. This might be improved upon by adding partially developed subtree to the candidate list instead of leaves. As we saw, however, this is not required with fewer than a couple hundreds nodes.

More theoretical works could also investigate how GIFs impact the bias-variance compared to the underlying forest. Intuitively, optimizing in part globally should further the bias while increasing the variance. Forests are also considered interpretable models, either when small (as a low node budget would enforce) or via their ability to detect important features and discard irrelevant ones. How GIFs impact the latter is yet unknown.

Finally, note that GIFs will be re-contextualized as an interpretable method in Chapter 9.

7 Chapter Sample-free out-of-distribution

Chapter overview

This chapter tackles the problem of out-of-distribution detection (*i.e.* recognizing samples not originating from the training distribution) in a sample-free setting. It is investigated in the context of image classification with deep neural networks. We proposed a set of indicators (based on optimality conditions, batch-normalization layers and a mix of both) to find the irrelevant samples. Overall the results are promising on all but the most challenging tasks.

This chapter is based on our publication "Sample-free white-box out-of-distribution detection for deep learning" (Begon and Geurts, 2021) presented at the fair, data efficient and trusted computer vision workshop of the conference of Computer Vision and Pattern Recognition (CVPR) in 2021.

The code relating to this contribution is available as a Python package at `https://github.com/jm-begon/ood_samplefree`. It is implemented on top of PyTorch (Paszke et al., 2017).

This chapter is organized in eight sections, the first of which (Section 7.1) exposes our ambitions: the goal pursued, our contribution and the motivation behind our work.
Section 7.2 analyzes under close scrutiny the problem of out-of-distribution (OOD) detection in general, while Section 7.3 focuses on the sample-free setting. Section 7.4 describes our solution based on OOD indicators, which are then empirically studied in Section 7.5. Section 7.6 proposes a scheme to aggregate the indicator in the absence of data. Section 7.7 considers how the indicators can be used in practice before reaching the conclusion in Section 7.8.

Departing from the article, a few changes and additions have made it to this manuscript. The most important one has been a lengthier discussion of the topic of OOD (Section 7.2). Materials from the appendix of the original paper have been partially integrated into the core of this chapter (Section 7.5.3). The conclusion (Section 7.8) has been expanded upon to reflect these changes.

7.1 Ambitions

7.1.1 Goal and contribution

Our goal is to provide some robustness, in the form of out-of-distribution (OOD) resilience, to an already deployed model, a situation where training data may no longer be available. We looked at the specific case of deep networks for which we suppose a white-box access is available. By that, we mean that the architecture and the trainable weights are known. Our experiments are conducted on image classification problems (although this is not a strict requirement).

Contribution. In light of these, our contributions can be summarized as follows:

- we present a detailed introduction to the complex domain of OOD detection (Section 7.2);
- we introduce the sample-free setting for OOD detection (Section 7.3);
- we review which prior works provide indicators falling into our setting and propose/adapt several new ones (Section 7.4);
- we conduct an extensive empirical analysis of these indicators on classical benchmark datasets (Section 7.5);
- we introduce a summary indicator to serve as an ultimate criterion for OOD detection, which is shown to perform well in comparison to the individual indicators (Section 7.6);

7.1.2 Motivation

Imagine you are running some medical tests to determine whether you have cancer or not, but erroneous data are fed to the machine learning (ML) model in charge of establishing the diagnosis. Would you prefer to get a positive or a negative answer? Or would you rather the model refrained from making a prediction and instead alerted the operator—something traditional models cannot do?

Out-of-distribution (OOD) detection (Hendrycks and Gimpel, 2017) precisely aims at detecting samples which come from a different distribution than the one used to train the model (Figure 7.1). There are many reasons why a model would be fed such OOD inputs: faulty equipment, user mistake, malicious intent, etc. A more insidious situation arises when inference is made on samples semantically belonging to the training distribution but in previously unobserved conditions such as different angles, lighting conditions, colors or shapes; a problem known as non-semantic shift (Hsu et al., 2020). And then there are all the philosophical issues raised in Section 4.1.2.

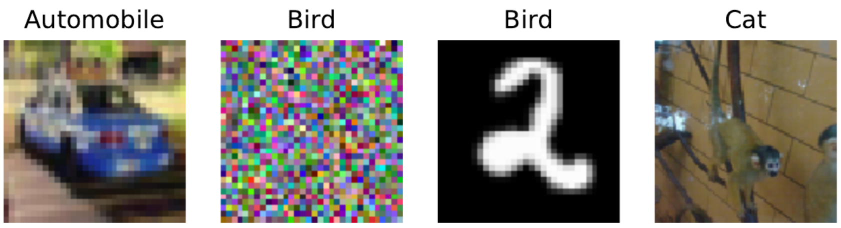


FIGURE 7.1: Examples of OOD samples. The original task is CIFAR 10 (Krizhevsky, Hinton, et al., 2009) whose label space includes cars, birds, and cats. From left to right: (i) an ID sample, (ii) an OOD noise sample interpreted as bird, (iii) an OOD hand-written digit sample interpreted as bird, and (iv) a misclassified ID sample.

Whether intentional or not, not being able for an ML model to detect when it receives an OOD sample to prevent senseless inference raises legitimate concerns about the reliability of systems built upon such models, especially in critical applications.

The sample-free setting. As more and more ML models are being deployed, addressing this issue becomes a very pressing matter. Unfortunately, as Sections 5.2.3 argues, such concerns might not have been anticipated. As a consequence, enforcing trustworthiness must be done at a further stage.

Delaying the implementation of robustness comes with many challenges, not the least of which is incurring the risk of the original, "in-distribution" (ID) data not being available, as argued in Section 5.2.4, a setting referred to as sample-free or data-free. This bears several implications, the first and most obvious one being that data is not available to learn from. Additionally, this implies that even if white-box access to the model is granted (which we assume), altering it (fine-tuning included) is off the charts: there is no way to control how changes would degrade performances.

On the brighter side, this approach comes along with a few free advantages. Sample-free OOD detection (i) will provide, by construction, data-efficient solutions, (ii) is relevant for other data-free paradigms, such as zero-shot distillation (*e.g.* Cai et al., 2020a; Cai et al., 2020b; Chen et al., 2019; Nayak et al., 2019, and the topic of Chapter 8), (iii) analyzing its limits will allow understanding how much information about the "in-distribution" is buried within a network trained in a standard way, and (iv) contrary to optimization-impacting approaches, it does not downplay the model accuracy. Actually, as Section 7.5.3.3 shows, sample-free methods can be used to anticipate misclassification, thus improving operational accuracy.

Since this chapter is concerned with two constraints (robustness in the form of OOD detection and the lack of data), Section 7.2 will first delve into the complex problem of out-of-distribution in general. We will come back to the sample-free setting in Section 7.3.

7.2 Out-of-distribution detection in general

This section discusses the problem of out-of-distribution detection in general terms. Section 7.2.1 is spent on the formulation. In particular, we highlight the importance of balancing the risks between missing OOD samples and falsely rejecting good ones (Section 7.2.1.3). We then look at what can be meant by out-of-distribution (Section 7.2.1.1). Section 7.2.1.2 concludes the discussion on formulation by looking at how out-of-distribution can relate to "good" data.

7.2.1 Problem formulation

The term "out-of-distribution" was recently coined by Hendrycks and Gimpel (2017), drawing buzz and fuzz alike. Its generality (and lack of formal definition) is responsible for some significant overlap with other problems, or techniques for solving them. Evidently, the concern for robustness and related topics is not new. In an old review on outlier detection, Beckman and Cook (1983) suggest that attempting to discard irrelevant data dates at least back to Bernoulli.

To better understand how different works and areas relate to each other, we will first formalize the setting. Let $\mathbb{X}$ be the input space, $\mathbb{P}_{\mathcal{I}}$ a density with support in $\mathbb{X}$, and $h(\cdot)$ be a hypothesis selected on $\mathbb{P}_{\mathcal{I}}$. We will refer to $\mathbb{P}_{\mathcal{I}}$ as the $\mathcal{I}$-distribution (or the in-distribution, possibly simply the training distribution). The term "out-of-distribution" suggests there is another distribution, $\mathcal{O}$ (possibly a mixture of several more compact distributions), with density $\mathbb{P}_{\mathcal{O}}$ supported in $\mathbb{X}$ as well. We will refer to $\mathbb{P}_{\mathcal{O}}$ as the OO-distribution. From there, OOD detection aims at deciding whether a sample x is a valid input for h depending on its relationship to $\mathcal{I}$ and $\mathcal{O}$.

Discussing further OOD detection amounts to answering the following questions:

- when should a sample be declared OOD?
- what is the relationship between $\mathcal{I}$ and $\mathcal{O}$?
- how should risk be balanced?
- what information (data or assumption) is available to help OOD detection?

The first three questions aim at clarifying the goal actually pursued, while the last one is about the means. The next sections will delve into the formers and the question of the available information is deferred to Section 7.2.4, where it will help articulate the related works.

7.2.1.1 Interpreting what is meant by out-of-distribution

In this section, we discuss what can be meant by out-of-distribution (OOD) detection. We describe four interpretations of OODness (*i.e.* what it means to be out-of-distribution), the first three of which are illustrated in Figure 7.2. The descriptions are followed by a discussion motivating the different views.

In-support interpretation. When looking at the problem only from the perspective of $\mathcal{I}$, we naturally arrive to the following interpretation.

Definition 7.2.1 (OODness under the in-support interpretation)**.** *A sample x is "in-distribution" (ID) if and only if $\mathbb{P}_{\mathcal{I}}(x) > 0$. A sample not ID is OOD.*

This interpretation simply says that if a sample *might* have come from $\mathcal{I}$ (*i.e.* it is in its support) it should be regarded as ID.

$\mathcal{I}$-membership-assessment interpretation. The $\mathcal{I}$-membership-assessment interpretation consists in regarding a sample as ID if and only if it is sufficiently likely that it comes from $\mathcal{I}$. It is a relaxation compared to the in-support interpretation. Typically, this will be formulated in terms of density.

Definition 7.2.2 (OODness under the $\mathcal{I}$-membership-assessment interpretation)**.** *A sample x is ID if and only if $\mathbb{P}_{\mathcal{I}}(x) > t$, where t is a predefined minimum density threshold. A sample not ID is OOD.*

Source-identification interpretation. By explicitly stating the existence of $\mathcal{O}$, we might naturally come to the following interpretation.

Definition 7.2.3 (OODness under the source-identification interpretation)**.** *A sample x is in-distribution (ID) if it is drawn from $\mathcal{I}$. It is OOD if it is drawn from $\mathcal{O}$.*

This interpretation simply uses the actual source distribution of a sample to designate it as ID or OOD. Whereas the in-support interpretation asks "could this x come from $\mathcal{I}$?", the source-identification interpretation asks "does this x come from $\mathcal{I}$?"

Membership-ratio-assessment interpretation. The membership-ratio-assessment interpretation consists in considering a sample as ID if it is (much) more likely to have come from $\mathcal{I}$ rather than $\mathcal{O}$. Typically, this will be formulated in terms of density ratio.

Definition 7.2.4 (OODness under the membership-ratio-assessment interpretation)**.** *A sample x is ID if and only if*

$$\mathbb{P}_{\mathcal{O}}(x) = 0 \textit{ and } \mathbb{P}_{\mathcal{I}}(x) > 0\,, \textit{or } \frac{\mathbb{P}_{\mathcal{I}}(x)}{\mathbb{P}_{\mathcal{O}}(x)} > t \tag{7.1}$$

where t is a predefined minimum density threshold. A sample not ID is OOD.

Why several interpretations? The need for several interpretations stems from the fact that there is some discrepancy between what the name suggests and the goal actually pursued. Indeed, Hendrycks and Gimpel (2017) opened their article by saying

> When machine learning classifiers are employed in real-world tasks, they tend to fail when the training and test distributions differ.
>
> Hendrycks and Gimpel (2017)

clearly indicating concerns about robustness and suggesting the goal is to prevent such situations. Their definition of out-of-distribution does not elaborate on that concern, however:

> [...] in- and out-of-distribution detection: can we predict whether a test example is from a different distribution from the training data; can we predict if it is from within the same distribution?
>
> Hendrycks and Gimpel (2017)

This leaves some gray area as to how samples should be considered when $\mathcal{I}$ and $\mathcal{O}$ share some support. On the one hand, the goal (*i.e.* rejecting invalid samples) leans towards the in-support interpretation. On the other hand, the name "out-of-distribution detection" is more aligned with the source-identification interpretation. As a consequence, paradoxical situations might arise, which the membership-assessment interpretations try to solve.

Interpretation comparison. When there is an overlap between distributions, the source-identification interpretation leads to situations where a sample x issued from $\mathcal{O}$ would be classified as OOD while still being likely to originate from $\mathcal{I}$. In such circumstances, a perfect (source-identification) OOD detector would prevent the model from making a prediction on x. It is, after all, the whole point of OOD detection. Yet, letting the model make a prediction for x in this case is totally legitimate. Therefore, we end up hampering the normal usage of the model. This is not acceptable.

In the in-support interpretation, the situation described above does not arise but makes way for two new issues. Firstly, everything would be ID with infinite-support $\mathcal{I}$-distributions. One could argue that it might not make sense to talk about OOD detection in such cases. This brings us to the second issue. Whether the support is infinite or not, we might end up with a sample x belonging to the support but much more likely (say several orders of magnitude more likely) to originate from $\mathcal{O}$. In all generality, it is unclear whether we would indeed like x to be considered as ID.

The $\mathcal{I}$-membership-assessment interpretation patches the previous problem by requiring that samples belong to areas of sufficient density. Besides introducing a parameter, this interpretation comes with a caveat, which is the opposite issue to source identification: samples from the tail of the $\mathcal{I}$-distribution can be rejected while being OOD is even less likely. In such a case, it would make sense to accept those samples. Arguably, those samples should be rare and would bear little influence overall.

The membership-ratio assessment (theoretically) solves all the previous problems, leaving its parametric nature as the prevalent issue. At first glance, being able to set the density ratio threshold on a per-problem basis might appear like an advantage. It is, however, somewhat redundant with risk management (but not equivalent, see Section 7.2.1.3). Ultimately, it feels odd not to have an absolute ground truth. Admittedly, one could formulate the problem of OOD detection as predicting accurately the density ratio. This still poses a practical problem (estimating the densities) on which we will come back in the next paragraph. Before that, let us mention another peculiarity (shared with the source-identification interpretation): the ground truth is also parametrized by the OO-distribution. That is, two different OO-distribution would lead to different ground truths. Envisioned solely as a classification problem, this seems normal. In the case of OOD detection, it means that a same sample $x \sim \mathcal{I}$ can be accepted in one context and rejected in another, depending on the OOD context. Whether a model should perform an inference on x does not seem like a contextual question, however: either making the prediction is legitimate, or it is not.

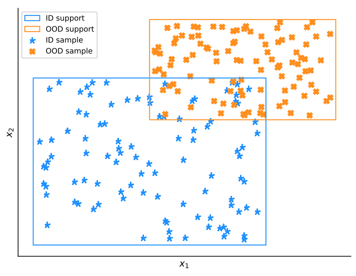
Establishing the ground truth. Both the membership-assessment interpretations are faced with a practical challenge. Unless querying the densities can be done directly (in which case the problem would mostly be solved anyway), establishing the ground truth requires estimating those densities from data. As such, what is held as ground truth could be quite noisy due to the estimation process. What is more, the noise would not be equally spread. Low-density areas will suffer the most since there would be less data to estimate precisely the density over those regions.

The in-support interpretation faces a special case of density estimation: bounding the support. For reasons similar to those mentioned above, we expect that the boundary can only be established with large imprecision.

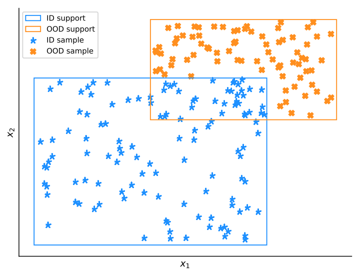
Only the source-identification interpretation is not hard-pressed when it comes to establishing the ground truth.

Which interpretation to choose? As we have argued, there is no perfect solution when it comes to defining what an OOD sample exactly is. The most promising interpretations are either $\mathcal{I}$-membership assessment or membership-ratio-assessment, depending on whether the definition of ID should be dependent on the OOD context. Only the network operator can answer definitively this question.

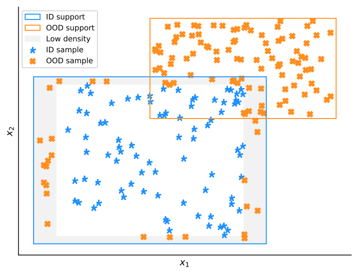
From a practical perspective, however, the source-identification interpretation is the only which offers an easily accessible ground truth. As way of consequence, it is the dominant interpretation as far as evaluation purpose is concerned. Of course, all interpretations converge when there is little overlap between the $\mathcal{I}$ and $\mathcal{O}$ distributions (and the thresholds of the membership assessments are kept low). Whether there is overlap or not depends on how close $\mathcal{I}$ and $\mathcal{O}$ are.



(A) OOD detection as source identification: where the data actually comes from constitutes the ground truth. The overlapping area between the distributions may contain both ID and OOD samples, resulting in some irreducible noise (the placement of the decision boundary will affect the risk-balancing).



(B) OOD detection as an in-support problem: all samples within the support of the $\mathcal{I}$-distribution—thus including the overlapping area—are ID samples.



(C) OOD detection as a membership assessment problem: all samples within the majority of the mass $\mathcal{I}$-distribution are ID samples, all others are OOD. Note that the low-density region has been exaggerated for visual purposes.

FIGURE 7.2: Different interpretations of OOD detection. The rectangles reflect the (finite) support of the distribution, while the markers indicate ground truth. Depending on how the problem is viewed, the ground truth of ID samples may change.

7.2.1.2 Relationship between data sources

This section attempts to explain how $\mathcal{I}$ and $\mathcal{O}$ may differ. We will look at different categories of problems

Gross statistical differences. The easiest case for OOD detection is when the out-of distribution bears little overlap with the training distribution. Examples of these are given in Figure 7.1 (images (ii) and (iii)) and include detecting white noise or images coming from a vastly different colorimetric spectrum. Although this task is easy, it might not necessarily constitute a case where rejecting such inputs is actually pursued, as with covariate shifts.

Dataset shifts (covariate and semantic). The concept of dataset shifts is well-defined and can go a long way in better understanding and describing what OOD detection is seeking to do. Let $\mathcal{M}$ be the mixture of $\mathcal{I}$ and $\mathcal{O}$ the

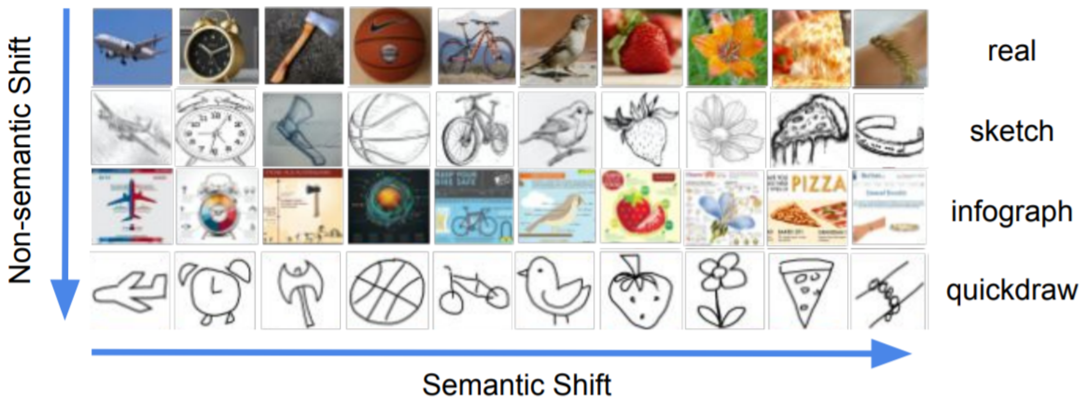


FIGURE 7.3: The ID/OOD shift can either be semantic, *i.e.* the label spaces are different, or non-semantic, *i.e.* the same concepts are presented in different modalities. From Hsu et al. (2020).

hypothesis will be exposed to once deployed and let $\mathbb{Y}_\mathcal{I}$ and $\mathbb{Y}_\mathcal{M}$ be the label spaces corresponding to training and mixture distributions respectively.

Definition 7.2.5 (Covariate/non-semantic shift)**.** *Covariate shift (Moreno-Torres et al., 2012), or non-semantic shift (Hsu et al., 2020) is when the problem remains unchanged but the input distribution changes:*

$$\begin{cases} \mathbb{Y}_\mathcal{I} = \mathbb{Y}_\mathcal{M} \\ \mathbb{P}_\mathcal{I}(\mathcal{Y}|x) = \mathbb{P}_\mathcal{M}(\mathcal{Y}|x) \\ \mathbb{P}_\mathcal{I}(x) \neq \mathbb{P}_\mathcal{M}(x) \end{cases} \tag{7.2}$$

Definition 7.2.6 (Semantic shift)**.** *Semantic shift (Hsu et al., 2020) occurs when data presented to the hypothesis does not belong to the label space the hypothesis was trained on:*

$$\mathbb{Y}_\mathcal{I} \neq \mathbb{Y}_\mathcal{M} \tag{7.3}$$

An illustration of semantic and non-semantic shifts are given in Figure 7.3. More common examples of covariate shifts include changes in lighting conditions, or more generally different modes of acquisition for the same information. Such changes might lead to gross statistical differences while keeping the same semantic structure. Inversely, gross statistical differences can fall under semantic shift (*e.g.* feeding a model white noise).

Ideally, OOD detection should be able to detect semantic shifts while a model should be robust to non-semantic shifts (*i.e.* would still correctly classify instances). In practice, and as suggested by Figure 7.3, non-semantic shift might result in a more impactful statistical shift and might thus be easier to capture. Detecting non-semantic shifts might not be such a bad idea when not doing so would lead the model, not trained to recognize the classes in those new forms, would commit many mistakes. As a result, one might hope that a model would be robust to slight covariate shifts and detecting large ones would only be required when the statistical disturbance is so large that

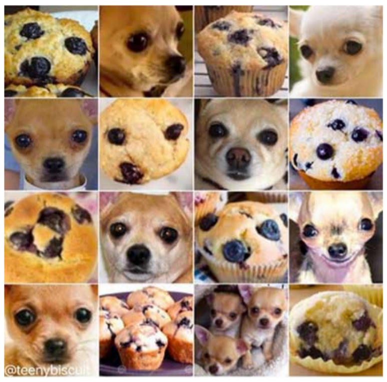


FIGURE 7.4: Semantic shift does not imply detecting OOD is trivial, as samples from very different label spaces can still be statistically (and visually) close. From Togootogtokh and Amartuvshin (2018).

little overlap remains between the $\mathcal{I}$-distribution and the covariately-shifted distributions, resulting in an easier problem.

Depending on the context, a semantic shift might be enough to guarantee that $\mathcal{I}$ and $\mathcal{O}$ do not overlap at a conceptual level, even if capturing the delineation in high-dimension remains challenging (as Figure 7.4 can testify). In other contexts, there is less assurance of being overlap-free. For instance, (poorly) handwritten letters can easily be mistaken for handwritten digits, other letters or letters from a different alphabet.

Note that other forms of dataset shifts (prior probability shift, concept shift/drift) have been defined but are not relevant to OOD detection (at least in the form we are addressing here; *e.g.* we will not assume that the definition of the $\mathcal{I}$-distribution will change over time).

Adversarial examples. Adversarial examples are samples engineered to fool a model (Goodfellow, Shlens, and Szegedy, 2015, see also Figure 7.5). Typically, this consists in, given a model, applying some well-chosen—and perceptually insignificant—noise on top of an ID instance so that the model will end up making a different prediction to the one it would have settled on (supposedly the correct one) in the absence of perturbation. In the case of neural networks, this is done by making one or several loss gradient ascent steps on the input variables, rather than on the weights (which is naturally handled by backpropagation).

Adversarial samples constitute a major reliability breach and detecting them is, by virtue of being such a targeted attack, a hard challenge. Adversarial samples might not qualify as OOD, however. By nature, the added noise must be kept low and the attack is only successful if the model departs from the prediction it would make with the original sample. Since the true class membership of the adversarial sample is regarded as unchanged (hence the model being fooled), nothing qualifies adversarial samples as either semantic or non-semantic shifts. Since $\mathbb{Y}_{\mathcal{I}} = \mathbb{Y}_{\mathcal{M}}$, $\mathbb{P}_{\mathcal{I}}(\mathcal{Y}|x) = \mathbb{P}_{\mathcal{M}}(\mathcal{Y}|x)$ and

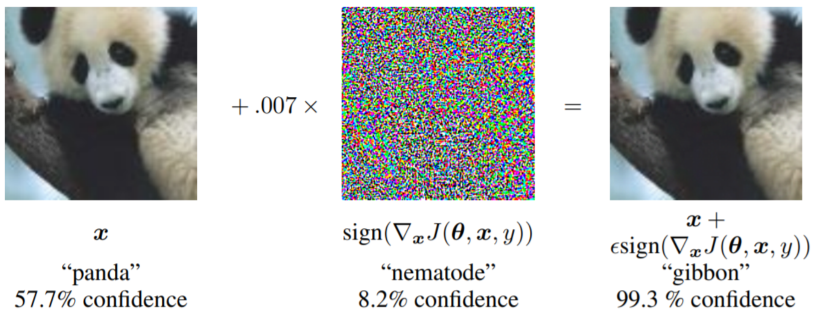

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

FIGURE 7.5: An adversarial sample, fooling a network to mistake a panda for a gibbon with high confidence, even though the change is barely noticeable (Goodfellow, Shlens, and Szegedy, 2015).

$\mathbb{P}_{\mathcal{I}}(x) = \mathbb{P}_{\mathcal{M}}(x)$ (where $\mathcal{M}$ represents the mixture over normal and adversarial samples here) we must conclude the problem is actually unchanged.

This is not to say that investigating how methods developed for OOD detection perform on other problems, such as adversarial sample detection, is not an interesting endeavor.

Unknown. In many (most?) cases, the relationship between the ID and OOD source is unknown. A model might receive semantically-shifted samples, covariately-shifted samples or a mix of both. Additionally, the proportion of OOD samples might be small or large—little is known *a priori*. This will steer methods towards ID-centric approaches.

7.2.1.3 Balancing the risks

OOD detection is one of those situations where risk management is important. Table 7.1 shows the confusion matrix associated to OOD detection (formulated as such, we assimilate the "positive" class to OOD). The two types of errors which can be committed in regards to OOD detection are (i) accepting OOD samples (false negatives) and (ii) rejecting ID samples (false positives). Normalizing the former by the number of OOD samples yields the OOD-acceptance rate, while normalizing the latter by the number of ID samples leads to the ID-rejection rate. Risk balancing is the notion of choosing which one of those rates to raise in order to lower the other (see Section 3.7.1 for a general discussion on the matter).

There are two main reasons for balancing the risks, rather than aiming for the equilibrium. When deciding whether a sample is ID or OOD, it often happens that all errors are not identical, with some worse than others. For instance, in a medical diagnosis application, it is preferable to reject an ID sample, in which case the medical examination can be redone, rather than make a prediction on irrelevant data. Even if both types of errors are equally penalizing, OOD detection is (hopefully) an imbalanced problem where there is a prevalence of ID samples. As discussed in Section 3.7.1, treating errors equally in such a situation might not lead to interesting solutions.

TABLE 7.1: Confusion matrix for OOD detection (OOD taken as the positive class). P stands for positive, N for negative, TP for true positive, FN for false negative, FP for false positive, TN for true negative.

	Prediction		
Actual class	OOD	ID	Total
OOD	detected OOD (TP)	missed OOD (FN)	P
ID	rejected ID (FP)	acknowledged ID (TN)	N

Whether risks need to be intrinsically balanced or simply should be because OOD is a rare event, OOD detectors should provide a way to balance the risks to tailor to a given application.

Note that, on the conceptual level at least, there is a great difference between changing the definition of OODness, possibly through some continuously varying parameter (*cf* Section 7.2.1.1), and doing risk management. Whereas the former is about the ground truth (*i.e.* the row marginals of the confusion matrix), the latter is about the model predictions (*i.e.* the column marginals of the confusion matrix).

7.2.2 Related problems

As previously mentioned, the concern for robustness is not new. Now that we have a better grasp of what out-of-distribution (OOD) detection is like, we can investigate how it compares to other related problems (see Section 7.2.3).

7.2.2.1 Open set recognition

The closed-world assumption is the idea that everything an agent will need to know will be presented at training time. Open set recognition works under the idea that there are unknowns at training time and the model will, sooner or later, face new classes and needs to handle gracefully this situation (Scheirer et al., 2012).

A variant of open set recognition (OSR) in the domain of dialog systems (intelligent answering machines) is out-of-scope (OOS) query detection, where OOS queries are "queries that users may reasonably make, but fall outside of the scope of the system-supported intents" (Larson et al., 2019).

Conceptually, OOD detection differs from open set recognition in the sense that the latter does not suggest a second source of data. Rather all data belong to a unique mixture with only a subset of classes known at training time. The overarching goals also differ. In OOD detection, the goal is to uncover problematic samples and reject them. Open set recognition ships with the idea of recognizing unknown classes so as to later learn from them and incorporate them into a more encyclopedic model.

Mathematically, however, open set recognition is akin to a semantic shift for which we may expect the distributions to be statistically close and more balanced. As such, much of the same techniques can be used to tackle both, barring one detail: open-world assumption is a design-driving principle rather than a posterior robustness overlay. As a consequence, open set recognition techniques are not squeamish about designing custom architectures (see Section 7.2.3.6). This contrasts to OOD detection, where the hypothesis space is dictated first by the base task. Also, this somewhat restricts OSR to training-time methods, in which case samples for the known class (*i.e.* ID data) are available.

The interested reader can refer to the work of Scheirer et al. (2012) for a review on open set recognition.

7.2.2.2 Anomaly detection

Anomaly detection (Chandola, Banerjee, and Kumar, 2009) aims at identifying samples which portray unexpected behaviors. Some amount of fuzziness surrounds the notion of anomaly. In some contexts, it might be taken in a statistical sense, in which case anomaly is synonymous of outlier, although belonging rightly to the $\mathcal{I}$-distribution (and thus often present in the learning sample). In others, anomalies are referred to as novelties, which lean conspicuously close to open set recognition. For some authors, OOD samples are a kind of anomaly. For others, there is the notion of a model trained on the original task which can be leveraged to help detect samples.

Overall, the demarcation between OOD and anomaly detection is shrouded in some epistemic blur. Even in the case of outlier detection—where there is implicitly no notion of an OOD source—developed methods might be recycled for OOD detection.

The interested reader can learn more in the relevant surveys (*e.g.* Chandola, Banerjee, and Kumar, 2009; Thudumu et al., 2020; Aldweesh, Derhab, and Emam, 2020).

7.2.2.3 Uncertainty modeling

The (estimation of the) expected risk reflects how a model is performing in general. In many applications, we would like to know how the model fares on a given input (not in average); we would like to know the *local* or *pointwise* uncertainty. Uncertainty is usually decomposed into data uncertainty (or irreducible/aleatoric uncertainty, also sometimes called ontic vagueness Worboys and Duckham (2004)), which is just another name for noise (as in the bias-variance decomposition, *cf.* Section 2.5), and model (or reducible/epistemic) uncertainty, which is the idea that several models can have the same consistency level over a finite dataset (Figure 7.6). In the absence of other discriminating criteria (*e.g.* weight decay, margin minimization), this situation leads to arbitrary choices of hypotheses. Although not strictly equivalent with variance (which relates to the models obtained with *different* datasets), a high model uncertainty at some location x will also usually involve a high variance at x.

Data uncertainty might be ill-suited for helping in some OOD detection tasks, such as with gross statistical differences. Informally, we might expect the uncertainty to be high for those OOD samples. The definition of

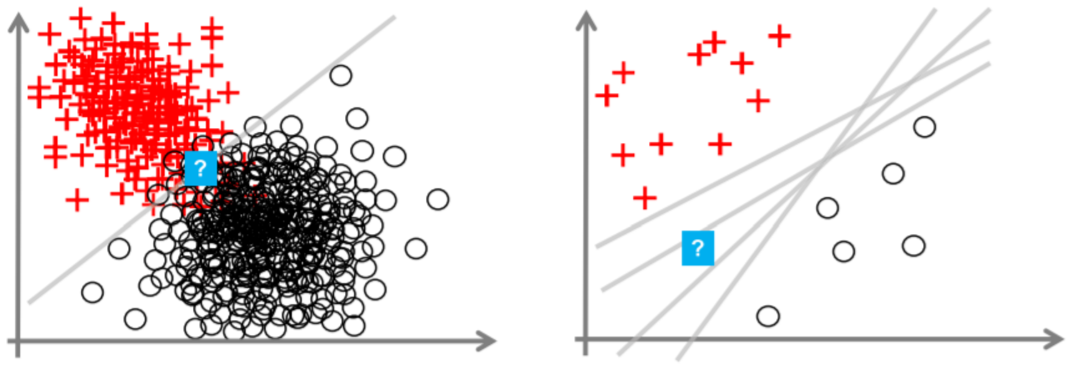


FIGURE 7.6: Aleatoric versus epistemic uncertainty. Left: even with the Bayes' decision boundary the class label (red cross or black circle) at the question mark is uncertain; an instance of aleatoric uncertainty due to a noisy problem. Right: several hypotheses are equally good for the problem; an instance of epistemic uncertainty. From Hüllermeier and Waegeman (2021)

the Bayes' model (on which the noise is based) actually says little as to how the model should behave outside of the scope of the problem. For instance, in an almost-linearly-separable problem, an OOD sample far from the Bayes boundary would actually have a close-to-zero noise level (see Figure 7.9 for such an example).

Model uncertainty might be more grounded for helping in OOD detection: this kind of uncertainty is expected to be high in regions where few instances have been observed, since consistency has not been encouraged there. This provides an interesting surrogate measure of low density for detecting OOD samples. For similar reasons, variance due to training sets might also be reflective of low-density areas: fewer samples would come from there, resulting in higher variance over those regions.

Conversely, OOD detection is an important question for uncertainty modeling as it might allow discerning when the model is extrapolating. In such instances, we expect the model to portray low confidence, since samples over those regions were absent during training. OOD detection may provide a shortcut for exhibiting reserve over those areas.

For more information on model uncertainty, see *e.g.* Hüllermeier and Waegeman (2021) and Abdar et al. (2021).

7.2.2.4 Pointwise versus samplewise methods

So far, the discussion has been on "pointwise" decisions: deciding whether a given datapoint was in- or out-of-distribution. In some context, it is possible to identify a bunch of data coming from the same source, in which case the question applies to many samples at once, providing richer information for the decision. This can happen, for instance, when a dataset shift occurs (Rabanser, Günnemann, and Lipton, 2019), in which case the data is an unstructured collection of points. Another instance is when data forms a structure, such as with time series. In this case, temporal components of the signal might appear individually normal, whereas the signal as a whole is not. Chandola, Banerjee, and Kumar (2009) encompass such cases in their survey.

In the remainder, we will focus on pointwise approaches.

7.2.2.5 Other paradigms

The aforementioned problems come in contact with other learning paradigms whose goal is to *adapt* to the new data encountered. For instance, in continual learning (*e.g.* Parisi et al., 2019), the goal is to keep learning even when the model is deployed. In zero-shot learning (*e.g.* Xian et al., 2019) the goal is to correctly label samples from unseen classes by leveraging available information of another nature (compared to the usual training samples) for these new classes. Domain adaptation (*e.g.* Jiang, 2008) aims at using data from one domain to learn a model applicable on another. This covers cases such as covariate shifts. More generally, transfer learning (*e.g.* Pan and Yang, 2009, see also Section 2.9.1) aims at providing knowledge learned in one situation for another.

7.2.3 Families of methods

If the previous section(s) discussed the "what", this section discusses the "how". Indeed, there is no one-to-one correspondence between the exact problem which is tackled (*i.e.* whether it is out-of-distribution detection, open set recognition, anomaly detection, and so on) and the method used to solve it. In other words, a method designed for uncertainty modeling might well be useful in the context of OOD detection, and *vice versa*. Fortunately, methods can be grouped into a few general families. This section inspects those while Section 7.2.4 will specifically target works which portray themselves explicitly as OOD detection.

7.2.3.1 Classification-based approaches

The most straightforward scheme, so long as the positive samples (*i.e.* OOD, anomaly) are well identified (and some labeled instances are available) is to frame the problem as a classification problem. The interesting question becomes the choice of input features. The first possibility is to work in the original input space, but working within some feature-engineered, or some latent space—whether it is learned for the task or inherited from the original task—might prove more adequate.

This scheme is common in fault detection systems, where the goal is to detect defects or predict when some system is about to fail (or is failing). See the review of Lei et al. (2020) for more.

7.2.3.2 Distribution-based approaches

Density-based approaches. Aside from classification methods, the other straightforward approach consists in learning the density of the $\mathcal{I}$-distribution and filtering out the tail of the distribution. There is a plethora of methods for density estimation, ranging from simple, fixed-bin width histograms to kernel density estimation (Parzen, 1962), to much more elaborate techniques such as normalizing flow (Kobyzev, Prince, and Brubaker, 2020).

Mass-based approaches. Density-based approaches come with a caveat: it is unclear at which value the density should be thresholded. Depending on the shape of the distribution, regions with a density lower than t might constitute only the tail of the distribution, or most of it. To circumvent this problem, it might be easier to work directly on the distribution mass.

The coverage problem (Geifman and El-Yaniv, 2019) consists in finding a subset of the input space which encompasses a given percentage of the input distribution mass. Samples originating from this subspace are then deemed normal and samples coming from outside can be rejected.

Admittedly, several subspaces might enclose the given mass. The problem of learning the minimum volume set (*e.g.* Polonik, 1997; Scott and Nowak, 2005) aims at finding the smallest coverage possible. A well-known representative method is the one-class support vector machine (Schölkopf et al., 1999).

7.2.3.3 Information-theoretic approaches

Both density- and mass-based approaches focus on the $\mathcal{I}$-distribution and work toward rejecting the tail of the distribution. Another take on the problem is to develop a representation of the samples wherein most of them can be compactly described by exploiting regularities in the data. As a consequence, samples having a lengthy description do not portray those regularities and are deemed suspicious.

An example of such a method is isolation forest (Liu, Ting, and Zhou, 2008), which consists in growing a variant of decision forest where anomalies are assimilated to samples found at abnormal depths.

Compression-based approaches. A variant of the information-theoretic approach is to truncate the representation and see how well the samples are reconstructed from the limited information.

For instance, Collin and Vleeschouwer (2020) proposed to use an auto-encoder to detect anomalies in images. The decoder part is trained to reconstruct anomaly-free images. Among the metrics studied, the distance between the original and reconstructed images performs best.

7.2.3.4 Proximity-based approaches

An altogether different approach than working on the distribution is to consider the input (or a latent) space and assume that samples close to the training data are normal. When the distance is established with respect to some modes of the distribution, this can be interpreted as an extreme case of compression, where the bases are the modes and the overall distance represents the reconstruction error.

For instance, Bolton and Hand (2001) proposed to use the distance between a sample and its k nearest neighbors from the training set to judge whether a sample is abnormal or not.

7.2.3.5 Confidence-based approaches

Many methods look into how confident a model is (or should be) of its prediction for an input sample and use that metric as a surrogate for detecting abnormal samples. This is either grounded in uncertainty modeling, or more simply in the fact that the model is encouraged to be (over)confident on the training distribution (see Section 3.6.2.1 for a more in-depth discussion).

The first approach when assessing the confidence of a neural network is to look at the probability vector it outputs. As shown by Figure 7.5 this might be misleading, though. Admittedly, adversarial samples constitute an extreme case. Nonetheless, Guo et al. (2017a) argues this is a general phenomenon. Even when looking at rightful samples, modern architectures tend to be overconfident: misclassifying some samples with utmost certainty. The authors propose to use the validation set to re-calibrate the network's confidence so that the network would be correct x% of the time for samples predicted with x% of confidence.

Methods inspired by uncertainty modeling include conformal prediction (Shafer and Vovk, 2008), which aims at giving confidence interval together with the prediction, bayesian learning (*e.g.* Gal and Ghahramani, 2016, and Section 4.1.4), where learning yields a distribution over models (rather than a single model) which can then be used to assess how uncertain a prediction is, or ensembling (*e.g.* Lakshminarayanan, Pritzel, and Blundell, 2017), where the same idea can be used, although with a very different justification.

7.2.3.6 Design-altering implementations

Parallel to those runs the question of how it should be implemented. Most methods employ disconnected apparatus, such as learning a new model for the task. In a few instances (most notably in open set recognition), though, the task has been forethought and dedicated models (or hypothesis spaces) are designed with the goal of implementing one of the above schemes in some embedded fashion.

For instance, Bendale and Boult (2016) propose to replace the softmax layer with an openmax layer which computes an elaborately-normalized distance to class centers in the logit space, highlighting samples which do not seem to belong to any classes. This is an instance of a proximity-based method implemented by adapting the architecture. Shu, Xu, and Liu (2017) instead advocate using sigmoid on a K-headed network, coming back to a 1-vs-rest approach on the ground that avoiding the normalization due to the softmax offers a better criterion to detect samples belonging to no class in particular. Yet another approach is to dedicate a head to detecting anomalous samples. For instance, Geifman and El-Yaniv (2019) propose to learn the head to reject a small percentage of the training distribution (coverage problem).

7.2.4 Literature on OOD detection methods

In the few years since the term was coined, undoubtedly aided by pre-existing related problems, out-of-distribution (OOD) has become a popular research topic. Since most works assume the availability of some data, they are not directly relevant to the sample-free setting we will target from Section 7.3 onwards. Consequently, this section will focus (i) on proposed methods (rather than comparison papers, (such as Shafaei, Schmidt, and Little, 2018; Rabanser, Günnemann, and Lipton, 2019; Roady et al., 2019)), (ii) which involve a model learned on $\mathcal{I}$ to perform the original task (rather than models learned solely for rejecting OOD samples (such as Vyas et al., 2018; Golan and El-Yaniv, 2018; Che et al., 2021; Kumar et al., 2021; Sehwag, Chiang, and Mittal, 2021).

To navigate through those works, we will first describe two baselines methods (Section 7.2.4.1) and then group methods by the assumptions they make regarding the available data (Section 7.2.4.2).

7.2.4.1 Baselines

Hendrycks and Gimpel (2017) pioneered out-of-distribution by proposing to use the maximum softmax probability to detect OOD samples; a confidence-based method. As discussed in Section 7.2.3.5, it has since been recognized that network confidence tends to be unreasonably high and might not reflect well uncertainty in a sense relevant to OOD detection (Ovadia et al., 2019).

ODIN (Liang, Li, and Srikant, 2018) improves upon this baseline by relying on two mechanisms. Firstly, instances are adversarially perturbed to increase the loss. According to the authors, this impacts ID and OOD samples in different ways, allowing for easier detection of the latters. The second idea introduced is to use a modified softmax (see Section 7.4.1 for more details).

7.2.4.2 OOD detection by available data

In this section, we look at methods depending on the type of data they need. Although publications usually showcase a method in a particular setting, the method is not necessarily confined in it. Figure 7.7 describes the generality structure over settings. The most general setting is the one requiring the least information. As such, we will proceed from the least to the most general setting, adding constraints as we go.

Both ID and OOD, labeled. The ideal situation, referred to as supervised, is when both ID and OOD data are available, with clear labels. Under the source identification view of OOD detection, it becomes a straightforward classification problem whose only remaining question is what input features should be used.

Aigrain and Detyniecki (2019) proposed to train a classifier on the logit vectors, while Quintanilha et al. (2018) derived some features based on the batch-normalization layers (Ioffe and Szegedy, 2015) of a trained network.

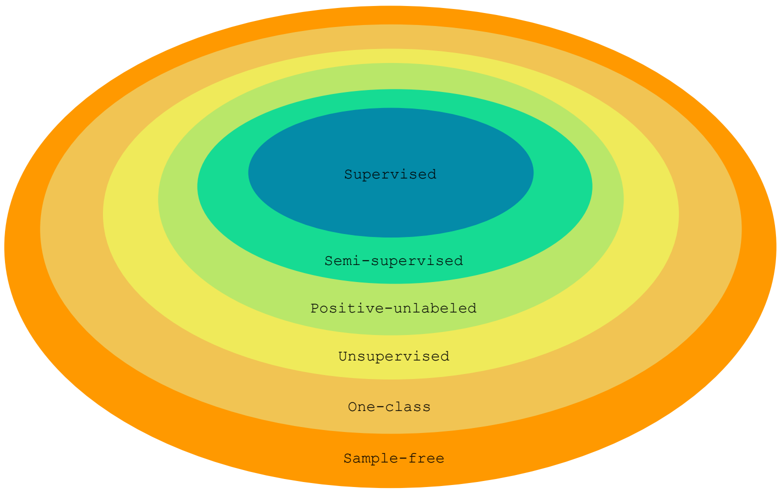


FIGURE 7.7: Generality structure over the available information. Supervised correspond to the setting where both ID and OOD data are available and labeled. When only some of those data are labeled, the setting is called semi-supervised. When labels are only available for ID samples, the setting is called positive-unlabeled. When no labels are available, the setting is called unsupervised. When only one type of data is available, the setting is called one-class. When no data are available, the setting is sample-free.

Using the network as a feature extractor, a dataset of ID and OOD samples can thus be created, then fed to any classification algorithm. The authors also investigate the case of having only ID data.

Lee and AlRegib (2020) proposed to use the gradients of the loss as a measure of uncertainty which can then be fed to a classifier. As a pseudo ground truth for the loss, they consider that samples may belong to all classes. This is motivated by the idea that large loss components will only incur, on ID samples, for the parameters towards the end of the network, whereas all the relevant feature extractors (beginning of the network) are already learned. In contrast, the loss components will be uniformly high for OOD samples.

Some authors have voiced their skepticism regarding the availability of OOD data, especially when labeled (*e.g.* Shafaei, Schmidt, and Little, 2018; Kardan, Sharma, and Stanley, 2021). Although such an amount of information is not necessarily unrealistic, the lack of justification has led the community to propose evaluation protocols where the OOD data available at training are considered as proxies rather than the embodiment of the actual OO-distribution the model will face. This is further discussed below (see "Only ID and proxy OOD" below).

Both ID and OOD, unlabeled except for some ID samples. In a variant of the previous setting, a large pool of unlabeled samples is available together with some identified as ID. This positive-unlabeled[1] setting differs from semi-supervised in that labels are only available for ID samples. This setting presents itself when the model is deployed but the learning set is still available, for instance. The ID/OOD balance of the unlabeled set usually weighs heavily on the expected performance.

Yu and Aizawa (2019) proposed to use neural networks with two classification heads—typically a modern architecture serving a feature extractor and a fully-connected layer per head. In a first phase, the neural network is trained in a classical fashion on the base task, the only difference between the heads being the initialization. In a second, fine-tuning stage, unlabeled samples are used to force a discrepancy (measure as the difference in softmax entropy) between the heads, while label, ID samples are used to avoid forgetting the base task. The underlying idea is that the discrepancy between the heads should be more pronounced on OOD samples, rather than on ID data for which the network is encouraged to yield the correct class with reasonable confidence. The discrepancy can thus be used as a measure of OODness.

Mohseni et al. (2020) also proposed two-headed networks for the positive-unlabeled setting. In this case, one head is classically dedicated to the base classification task while the other is a multi-class rejection head, allowing for more flexibility than a binary rejection head. The second head is trained solely with unlabeled samples using a cross-entropy loss with self-predicted labels from the previous epoch. The sum of the softmax prediction of the rejection head is used as detection signal.

Only ID and proxy OOD. Since many authors have shown their skepticism of the supervised setting, judging the availability of labeled OOD samples too strong an assumption, many works have looked at using easily-available proxy data to serve as positive samples. Whether this is a good idea or not depends on the relationship between the proxy and true OOD data—see Figure 7.8—which cannot be asserted due to the lack of data. Therefore, such methods must be used with much caution when strong priors regarding the OO-distribution are lacking. Aside from those considerations, this setting can be implemented so long as ID data are available and could, in principle, be coupled with other settings such as the positive-unlabeled or semi-supervised, propelling back the methods into the supervised realm.

Together with the maximum softmax probability, Hendrycks and Gimpel (2017) propose to append an "abnormality module" to a network. This consists of (i) a decoder mapping from the latent space back to the input space, which is trained jointly as the regular network during training, and (ii) a sigmoidal head, fitted after regular training on normal and noisy versions of the original data. Overall, this falls into the compression-based approach with a design-alternating implementation and the proxy data is noise over ID samples.

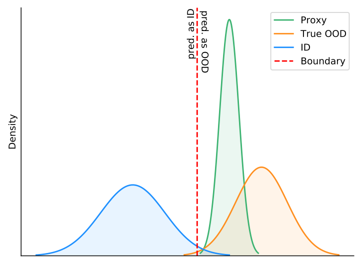
[1]negative-unlabeled would be more consistent since we consider OOD as the positive class but we will stick to the denomination in use.

Lee et al. (2018a) propose to train a network to jointly perform well on the ID task and detect OOD samples. To achieve this, they propose to jointly train a GAN (Goodfellow et al., 2014) and the network. Traditional GANs are made of a generator, whose goal is to produce samples as close as possible to the training data, and a discriminator, whose goal is to distinguish between training data and generated samples. In this case, the generator is also encouraged to deliver samples close to the boundary. The antagonist nature of the two goals the generator pursues is supposed to encourage sampling from the tail of the $\mathcal{I}$-distribution.

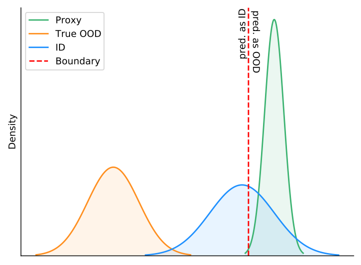
Hendrycks, Mazeika, and Dietterich (2019) follow the same general approach of training jointly a network on the original task and for OOD detection. To circumvent the need for a fully-fledge GAN scheme, including the choice of architecture, the choice of optimization-balancing hyper-parameters, and the lengthy training time, they advocate using OOD training cases coming from other datasets. Without the generator and discriminator, the loss simplifies considerably, leaving only one network and two loss components: a traditional cross-entropy and a term whose purpose is to lower the network confidence on OOD data (in the form of a cross-entropy between the softmax probabilities and a uniform distribution in the case of classification). The tradeoff between performance on the base task and OOD detection is managed through a single hyper-parameter.

Working on genomics sequences, Ren et al. (2019) hypothesize that OOD examples are hard to detect because they have the same background noise as ID samples, the difference between the two being the presence of interesting structure in ID data. They propose to generate OOD samples by breaking the structure of ID samples. Rather than learn a discriminative model between the two, they work with two generative models: one for assessing the likelihood of being ID in general, and one for assessing the likelihood of having the same background noise. By taking the ratio between the two, the authors estimate they capture the likelihood with respect to the structure of interest.

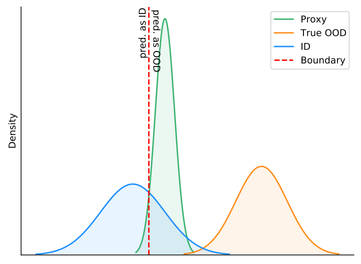
When faced with a large collection of auxiliary data, Li and Vasconcelos (2020) proposed to sample mini-batches to form an OOD dataset. Rather than sample uniformly, they proposed to weigh the auxiliary samples according to how non-uniformly they would be predicted, a scheme they termed as adversarial resampling. The network is then trained to jointly perform well on the base task and produce low confidence on OOD data thanks to a mix-objective similar to the one proposed by Hendrycks, Mazeika, and Dietterich (2019).



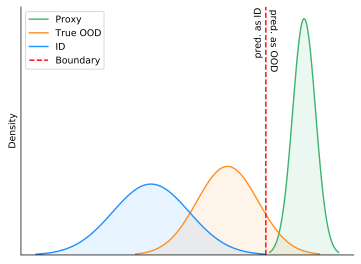
(A) Ideal situation: the proxy data is a relevant surrogate.



(B) Worst case: the proxy data is so irrelevant that the OOD detector will be useless.



(C) The proxy data is relevant but balancing the risks is difficult. If the goal is to be cautious of possible OOD samples, most of the $\mathcal{I}$-distribution might be rejected even though accepting it would be safe with respect to the actual OO-distribution.



(D) Second worst case: the proxy data is irrelevant. It promises an almost perfect accuracy which is not reflective of what would happen with true OOD samples (especially in the source-identification view).

FIGURE 7.8: Difficulty in placing the decision boundary for risk management depending on the relationship between proxy OOD data and true OO-distribution. The density does not relate to a specific input variable and proper engineering of the features, as well as working in higher-dimension spaces might help favor the best case. The lack of true OOD data makes it hard to assess, though. Note that using proxy data such a noise tends to fall into the gross statistical differences embodied by the bottom right plot.

Only ID. The most common case might be to have only ID data available; leftovers from training. If we disregard the possibility of using proxy data for OOD (discussed in the previous paragraph), this restricts the methods to low-density filtering and related.

DeVries and Taylor (2018) proposed an uncertainty modeling approach which consists in appending a confidence head at the end of the network. The two network heads are jointly trained so that the network can decide not to be punished for a prediction on which it is not confident but must be as confident as possible. After optimization, the confidence score can be used to decide whether an input should be discarded as OOD or not.

Lee et al. (2018b) proposed a proximity-based approach operating in the latent space(s) of a network: they use the Mahalanobis distance between a prediction and its closest class center as a criterion to accept or reject samples. The estimation of class centers and Mahalanobis parameters (means and covariance matrices) require ID samples. The authors advocate to either use the logit space or train a classifier of distance vectors from several latent spaces, in which case OOD or proxy samples, might be required.

Sastry and Oore (2019) followed along the same lines and proposed to use statistics based on the whole feature maps extracted at each layer. To use them in a one-class fashion, they use the sum of the deviations from the means for each statistic as ultimate criterion. By working with normalized scores, they alleviate the need for OOD data to tune how to weigh the per-layer features when aggregating them.

Ahmed and Courville (2020) proposed to leverage auxiliary tasks (such as predicting the orientation of objects) while learning to build models which are more resilient to semantic anomalies. Interestingly, learning along several related tasks also improves the model accuracy-wise.

Lee, Yu, and Yu (2020) proposed multi-class data description (MCDD), a multi one-class approach where spherical decision boundaries are learned to encompass most of the learning instances without overlapping. They then relax their formulation to model the hyperspheres as isotropic Gaussian distributions, allowing for the density to the closest Gaussian as OOD metric.

Hsu et al. (2020) noticed that when trained only on ID samples, the probabilities the model outputs are always implicitly conditioned to the sample being ID. This probability can be factorized into two components, one of which being the probability of a sample being ID. Based on this observation, they proposed to train two functions of the logits to model these two components, although why this precise decomposition should end up being the one enforced remains fuzzy. Since this method modifies the architecture, it needs ID samples to be learned. Input pre-processing also requires the availability of ID data.

Antonello and Garner (2020) noticed that the linear boundary associated with fully-connected-and-softmax layers is not suited for OOD samples ending up far away from the boundaries since the model will be extremely confident of itself. To mitigate this issue, they proposed to replace the softmax layer with a so-called t-softmax layer. This layer integrates a quadratic term with respect to the latent space, better able to capture OOD samples far away from the decision boundaries.

Cheng and Vasconcelos (2021) proposed another framework for novelty detection using the Mahalanobis distance. Their contribution is to note that Mahalanobis distance really makes sense when the latent, class-conditional vectors follow a multi-dimensional Gaussian distribution, which is at best a rough approximation in practice. Consequently, they introduce a new term in the loss to enforce gaussianity during training so that thresholding the distance ends up making sense.

Note that some of these methods employ hyper-parameters which benefit greatly from tuning with OOD data, especially once they are used with data not related to the benchmarks on which they are showcased.

No data (sample-free setting). Another setting is the utter lack of data of any kind, which is the setting adopted in the chapter and will be examined in more detail in Section 7.3. As far as related works are concerned, the maximum softmax probability of Hendrycks and Gimpel (2017) and ODIN (Liang, Li, and Srikant, 2018) (provided we use the default hyper-parameters), match our setting, where (i) the model cannot be altered (since controlling the alteration without ID samples is challenging), and (ii) no ID data is available. Another recent work which falls into this category was proposed by Liu et al. (2020). Motivated by the energy-based model, the authors proposed to use the log-sum-exp of the logits (in other words, the denominator appearing in the softmax).

The emergence of zero-shot sampler methods (*e.g.* Chen et al., 2019; Nayak et al., 2019; Haroush et al., 2020; Micaelli and Storkey, 2019; Choi et al., 2020), whose goal is to (try to) sample from $\mathcal{I}$-distribution by solely looking at a model learned from it, is a promising research venue, offering to craft ID samples (or relevant-enough ones) to then apply an only-ID method. Actually, most of those works can be seen as doing the opposite: using a sample-free OOD detection method to guide how training samples could be crafted. Chapter 8 proposes a method in that vein.

Other settings. Other settings, as depicted by Figure 7.7, exist but do not seem to have been tackled yet (in the sense that existing works fall better in another category).

In semi-supervised OOD detection, are available samples which are clearly identified as ID or OOD, as well as a collection of unlabeled samples, being either ID or OOD. In this setting, labeled and unlabeled OOD samples might come from the same distribution, a different distribution, or a mix of both. When there is only one OO-distribution, this resembles the supervised approach, whereas having completely different distributions leans in the direction of proxy data.

Another setting is where only unlabeled data, containing both ID and OOD samples, is available. This unsupervised setting resembles the sample-free setting quite closely. Note that Yu and Aizawa (2019) actually proposed a method for the unsupervised setting but built a completely separate detector from scratch. This setting will also be invoked in Section 7.7. An interesting question with this setting when a model is available is whether it was trained on pure data or the training set already contained OOD samples of some form (such as might be the case in outlier detection).

Finally, a last—and amusing—setting is the one where only OOD samples are available. Supposing these are from the OO-distribution of interest, for instance samples collected while running the model for being recognized as abnormal, one-class methods might be as useful as if the data was ID. This would have the advantage to offer more direct means of controlling the (usually) worst risk: the OOD-rejection rate.

7.2.4.3 Conclusion

This rather long introduction has shed some light on the domain of out-of-distribution (OOD) detection. We have seen that it is both important and tricky to formulate, with several different interpretations of what is meant by OOD, an intrinsically contextual definition of what the OO-distribution is compared to the $\mathcal{I}$-distribution and the need to balance both types of risks (accepting OOD/rejecting ID samples). As concerns for robustness are not new, OOD detection closely resembles or interconnects with other problems, such as open set recognition, anomaly detection, and uncertainty modeling. Even in situations where problems can be distinguished, methods developed in one context tend to be applicable in another. As such, OOD detection (under some assumptions, *e.g.* the source-identification interpretation) offers a simple testbed for other paradigms as well.

Many approaches have been proposed over the years and can be partitioned either based on how they operate or on what kinds of data they required. For works grounded in the OOD literature, the two most common settings are relying either only on ID samples or on ID samples and some proxy data. In the latter case, an important issue is how the proxy data relates to both the in- and out-of-distributions. Overall, the most successful venue—or, at least, the one which has convinced the most—is the proximity-based approach (*e.g.* Hendrycks and Gimpel, 2017; Liang, Li, and Srikant, 2018; Lee et al., 2018b; Sastry and Oore, 2019; Liu et al., 2020; Lee and AlRegib, 2020), possibly with some learning-time tweaking (*e.g.* Hendrycks and Gimpel, 2017; Hendrycks, Mazeika, and Dietterich, 2019; Lee, Yu, and Yu, 2020; Antonello and Garner, 2020; Cheng and Vasconcelos, 2021). Note that some of these approaches might not have initially been motivated as proximity-based. For instance, the maximum softmax probability was motivated as a confidence-based approach and only falls into the proximity-based realm due to how the softmax operates.

Some of these works can be seen as inspirations (or natural convergences) of our proposed method, where assumptions and knowledge on how training networks works replace what is usually gathered by the available data. For instance, some of the indicators discussed in Section 7.4.1 aims at achieving the same goal as the Mahalanobis-distance technique of Lee et al. (2018b), where knowledge of how a trained network should behave is used as a surrogate for the actual location of the class centers. As Sastry and Oore (2019), we also use normalized indicators to form an aggregated OOD signal, although in our case we use a proxy distribution instead of ID data for the normalization. Many authors share the idea of using proxy data, although in our case it is more a surrogate for ID than OOD data.

7.3 The sample-free setting

Hereon we focus on out-of-distribution (OOD) detection in a sample-free setting, as motivated in Section 7.1.2. In this short section, we first ask the question of whether OOD detection is feasible in such a constrained setting (Section 7.3.1) before explaining how we envision pursuing our goal in general terms (Section 7.3.2). Section 7.4 will then delve into the details.

7.3.1 Feasibility

Can anything actually be done in the total absence of data? In comparison, the availability of ID and OOD data hints at learning an OOD classifier. Even when only ID data is at hand, it is possible to estimate the density of $\mathcal{I}$.

To be a viable option, the sample-free setting must leverage what knowledge is already buried within the trained model. Intuitively, it feels that a learned hypothesis would have been selected so that the decision boundaries are meaningful with respect to the (conditional) densities. To see how relevant this is, remember that the hypothesis $\hat{p}$ is selected so that

$$\hat{p}^{(k)}(x) \approx \mathbb{P}_{\mathcal{I}}(\mathcal{Y}=k|x) = \frac{\mathbb{P}_{\mathcal{I}}(x|\mathcal{Y}=k)\,\mathbb{P}_{\mathcal{I}}(\mathcal{Y}=k)}{\sum_j^K \mathbb{P}_{\mathcal{I}}(x|\mathcal{Y}=j)\,\mathbb{P}_{\mathcal{I}}(\mathcal{Y}=j)} \tag{7.4}$$

In a sense, learning the classifier amounts to estimating the class-conditional densities and the class marginals, the latter of which is trivial. Marginalizing on the class, we get (law of total probability)

$$\mathbb{P}_{\mathcal{I}}(x) = \sum_j^K \mathbb{P}_{\mathcal{I}}(x|\mathcal{Y}=j)\,\mathbb{P}_{\mathcal{I}}(\mathcal{Y}=j) \tag{7.5}$$

Therefore, the selected hypothesis, if any good, must somehow contains traces of information about the density $\mathbb{P}_{\mathcal{I}}(x)$.

The next section describes in more detail what kind of information we are seeking. Section 7.4 will examine several ways to leverage traces of such information.

7.3.2 Indicators

We propose to construct functions $g : \mathbb{X} \to \mathbb{R}$, called indicators, that allow discriminating as well as possible ID from OOD samples, with respect to the given neural network.

We will design indicators that take low values for ID samples and large values for OOD samples. In practice, a test example x can thus be rejected as soon as $g(x) > t$, where t is a risk-balancing threshold that can be set to minimize a given error type, taking into account the needs of the application.

Depending on how out-of-distribution is to be interpreted, the ideal indicator is $\frac{\mathbb{P}_{\mathcal{O}}}{\mathbb{P}_{\mathcal{I}}}$ (membership-ratio assessment; source identification) or $\frac{1}{\mathbb{P}_{\mathcal{I}}}$ (in-support interpretation; $\mathcal{I}$-membership assessment).

No sample from $\mathcal{I}$ or $\mathcal{O}$ are available but we assume a white-box access to the neural network, which allows us to investigate candidate indicator functions of the following general form:

$$g(x) = G\left(x, \Theta, z_1(x;\theta_1), \ldots, z_L(x;\theta_L)\right). \tag{7.6}$$

where $z_l(x;\theta_l)$ represents the latent feature vector of the lth layer corresponding to x (see Section 3.6 for more details).

Indicators can thus be defined from features computed anywhere in the network, as well as from network parameters. Given that $\mathcal{O}$ is unknown, our main way of making sure that $g(x)$ is low for $x \sim \mathcal{I}$ is to take into account the way the neural network was trained.

7.4 Sample-free white-box OOD indicators

In this section, we introduce a number of indicators for OOD detection. An indicator assesses how unlikely it is for a sample to be ID (Section 7.3). We describe two categories of such indicators: (i) optimality-based indicators (Section 7.4.1), and (ii) batch-normalization-based indicators (Section 7.4.2).

7.4.1 Optimality-based indicators

Hopefully, a deployed network should be well trained on the original task, resulting in an (near-)optimal network over the $\mathcal{I}$-distribution. Sections 3.2.2.2 and 3.6.2.1 discussed how the network proceeds to reach optimality, which can be summarized as

- avoiding misclassification by learning a latent space where classes are linearly separable;
- boosting the network confidence by
  - pushing points in the latent space far away from the boundaries, ideally perpendicular to the hyper-plane for maximum efficiency;
  - increasing the norm of the hyper-plane vectors.

All these effects result in $\hat{p}_j(x)$ converging to y_j $(1 \leq j \leq K)$.

The indicators developed in this section are based on the assumption that those conditions are met for ID samples but might not be met for OOD ones. As noted in Section 7.2.4.3, this is also the motivation behind several, non-necessarily sample-free methods.

In the event where the network has reached convergence but still performs poorly, we expect the optimality-based indicators to fail. This would be the case if the network is substantially overfitting or if it performs poorly in general (the loss gradient is low but the error is still high, for instance with a network of limited capacity). In both cases, the consequences of optimality would not be observed on the $\mathcal{I}$-distribution, invalidating the proposed indicators. It can reasonably be expected, however, that a deployed model would be any good at generalizing. Consequently, the developed indicators are reasonable in practice. We discuss this further in Appendix B.2 when comparing CIFAR 10 and CIFAR 100. We additionally shed some more light on the matter in Section 7.5.3.2, where we discuss the impact of the model quality, and in Section 7.5.3.3, where we discuss the related problem of misclassification detection.

Baselines. Two common baseline indicators which derive directly from the optimality condition are

$$\textsc{mp}(x) = 1 - \max_{1 \leq j \leq K} p_j(x) \tag{7.7}$$

$$\textsc{h}(x) = - \sum_{j=1}^{K} p_j(x) \log p_j(x) \tag{7.8}$$

Using the maximum probability was proposed by Hendrycks and Gimpel (2017) when introducing the topic of OOD detection. When the probabilities given by the network for the minority classes are uniform and close to zero, the entropy H should behave like MP. The entropy might convey a little more information than the maximum probability when the uniformity constraint is not satisfied.

ODIN. ODIN was introduced by Liang, Li, and Srikant (2018) and is popular in the OOD context. It relies on two ideas. First, some adversarial noise (Goodfellow, Shlens, and Szegedy, 2015) is added to the input x. Then, the softmax probability vector given by the network is computed using a temperature T of 1000 in the softmax:

$$x' = x + \epsilon \text{ sign} \left(\nabla_x \ell_{CE}(\hat{p}(x), d_k) \right) \tag{7.9}$$

$$\hat{p}_T^{(j)}(x) = \frac{e^{z_L^{(j)}(x)/T}}{\sum_{k=1}^{K} e^{z_L^{(k)}(x)/T}} \tag{7.10}$$

$$\text{T1000}(x) = 1 - \max_{1 \leq j \leq K} \hat{p}_{1000}^{(j)}(x) \tag{7.11}$$

$$\text{ODIN}(x) = \text{T1000}(x') \tag{7.12}$$

where d_k is the one-hot vector whose sole non-zero element is a one in the kth component, and where k is the class of x predicted by the network. The rationale is that the adversarial perturbation will have different effects on

ID/OOD samples. Additionally, it can be shown (Appendix B.1), that

$$\hat{p}_T^{(j)} \approx \frac{c}{K} + \frac{1}{T\,K} z_L^{(k)} \tag{7.13}$$

so long as $z_k \ll T$. As such, using T1000 is a way of normalizing the logit of the predicted class in the range $0 \ll \text{T1000}(x) \leq 1 - 1/K$.

When $\epsilon = 0$, ODIN reduces to T1000 and the expensive cost of computing the adversarial perturbation is avoided. Tuning ϵ in a sample-free setting is not trivial. Arguably though, the magnitude of the perturbation might not vary much due to the sign function. How much the direction influences the results of ODIN in the general case is not clear. In any case, we will use the default value of the noise magnitude proposed in the original paper ($\epsilon = 8 \times 10^{-4}$). Considering it was established on CIFAR 10(0) as well, it should constitute a strong baseline anyway.

Latent space indicators. Let $u = z_{L-1}$ be the latent pre-linear vector and $z_L = Wu + b$ be the logit vector, with $\theta_{L-1} = [W, b]$.

Since $w_k^T u + b_k = ||w_k||\,||u|| \cos \alpha_{u,k} + b_k$, the optimality conditions imply that:

1. $||u||$ is high (*i.e.* the point is far away from the center in the pre-linear latent space);
2. $\cos \alpha_{u,k}$ is close to 1 (the point is well aligned with the hyper-plane).

From them, we can derive the following indicators:

$$\textsc{norm}(x) = -||u|| \tag{7.14}$$

$$\textsc{ang}(x) = 1 - \cos \alpha_{u,k} = 1 - \frac{w_k^T u}{||w_k||\,||u||} \tag{7.15}$$

$$\textsc{proj}(x) = -||u|| \cos \alpha_{u,k} = -\frac{w_k^T u}{||w_k||} \tag{7.16}$$

$$\textsc{act}(x) = -w_k^T u \tag{7.17}$$

The NORM indicator should not be sufficient by itself, as a high norm possibly benefits all the logits. ANG stands for angularity and is the cosine distance between u and w_k. Compared to the logit (close to ACT), it will favor more samples which align well with the hyperplanes and will favor fewer samples which just have a high latent norm. The PROJ indicator combines the information from both ANG and NORM. When $||w_j||$ and b_j are relatively constant with respect to j (a situation which can be expected in the absence of asymmetry, such as class imbalance; see also Table 7.2), PROJ is expected to be closely related to the logit. ACT is the logit without the b_j part and should also be close to the logit.

Positivity. ReLU-based architectures, which include most modern ones in image classification, end the feature extraction phase with a ReLU activation,

TABLE 7.2: Statistics of the latent space parameters. The parameters have been extracted from trained network on CIFAR 10/100. See Section 7.5.1.1 for more information about the networks, the datasets and the learning protocol.

	ORDER OF STD(W) / STD(b)		% OF POSITIVE COMPONENTS OF W	
	CIFAR 10	CIFAR 100	CIFAR 10	CIFAR 100
RESNET 50	10^{-2} / 10^{-1}	10^{-2} / 10^{-2}	37.5	35.8
WIDERESNET	10^{-1} / 10^{-1}	10^{-1} / 10^{-2}	41.1	40.1
DENSENET 121	10^{-1} / 10^{-1}	10^{-2} / 10^{-2}	42.1	43.1

possibly followed by max or average pooling. As a result, the latent vectors are non-negative, whereas most components of the hyper-plane weights are negative (see the discussion in Section 3.2.2.2 and Table 7.2) and are used to bid *against* the other classes, rather than *for* the predicted one. This suggests that it might be worth looking at the positive and negative parts of the previous indicators separately.

We define three new indicators NORM+, ANG++ and ACT+ that are obtained by reducing the vectors w_k and u to the components with positive weights in w_k in the definitions of NORM (Eq. 7.14), ANG (Eq. 7.15), and ACT (Eq. 7.17) respectively. In other words, we only consider the positive subspace of w_k.

More precisely, let $\mathcal{P}(w) = \{1 \leq i \leq p_{L-1} | w^{(i)} \geq 0\}$ (where p_{L-1} is the dimensionality of the pre-linear latent space) be the set of indices of the non-negative components of w. Suppose the predicted class for x is k and let

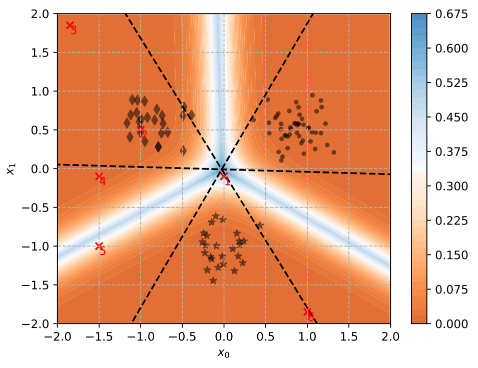
$$||\cdot||_j^+ = \sqrt{\sum_{i \in \mathcal{P}(w_j)} \left(\cdot^{(i)}\right)^2} \tag{7.18}$$

$$\text{NORM+} = -||u||_k^+ \tag{7.19}$$

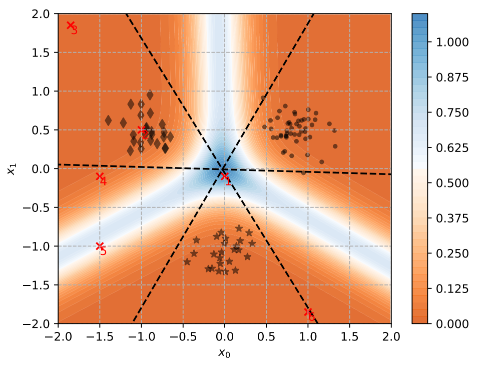
$$\text{ACT+}(x) = -\sum_{i \in \mathcal{P}(k)} w_k^{(i)} u^{(i)} \tag{7.20}$$

$$\text{ANG++}(x) = 1 + \frac{\text{ACT+}(x)}{||w_k||_k^+ \, ||u||_k^+} \tag{7.21}$$

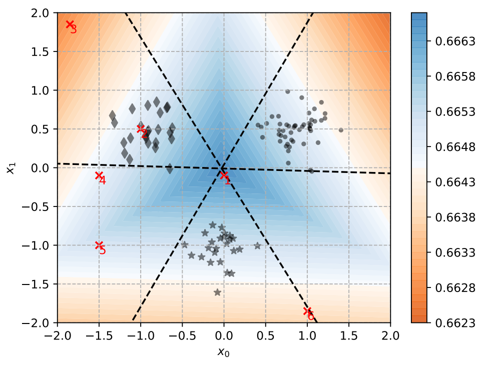
The rationale for using the positive indicators, instead of their counterpart, is to reject OOD samples whose high probability would be due to being unlikely to come from any other classes than the predicted one, rather than appearing to belong to the predicted class. Note that positivity also implies ANG cannot be zero.



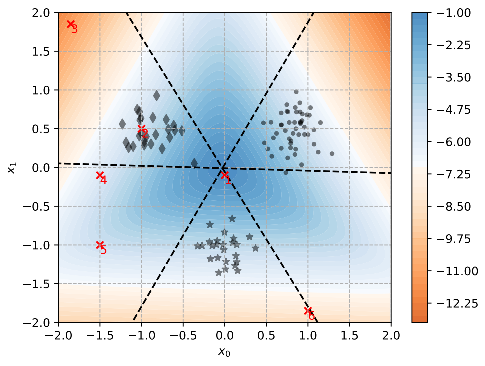
(A) MP



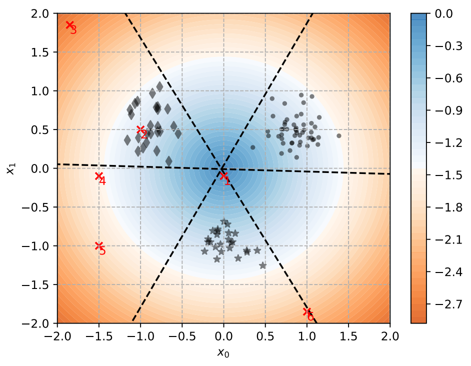
(B) H



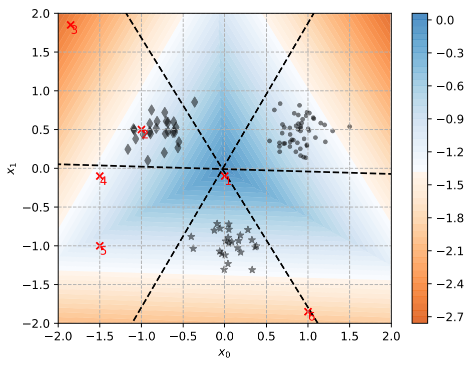
(C) T1000



(D) *LSE* (Liu et al., 2020)

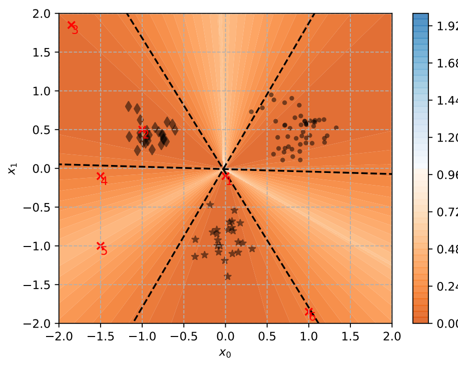


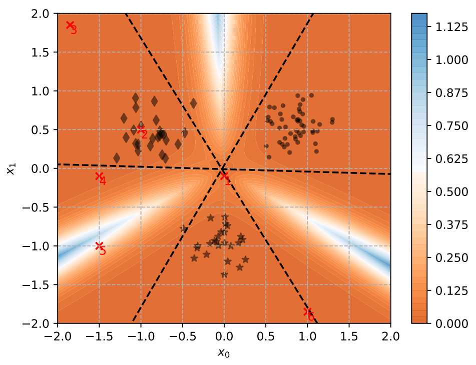
(E) NORM



(F) PROJ

FIGURE 7.9: Visual comparison of several indicators in the case of a softmax classifier for the 3G problem (Section 3.2.2)—first part. Notice that the color scales depend on the indicators and are not directly comparable; what matters is how OOD samples would be ranked in comparison with the displayed ID classes.


(A) ANG


(B) Loss gradient magnitude inspired by (Lee and AlRegib, 2020)

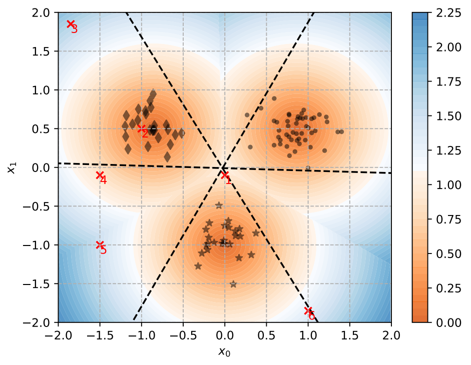

(C) Distance to class centers somewhat similar to (Lee et al., 2018b)

FIGURE 7.10: Visual comparison of several indicators in the case of a softmax classifier for the 3G problem (Section 3.2.2)—second part.

Comparison of the optimality-based indicators. Figures 7.9 and 7.10 illustrate some of the optimality-based indicators. As is apparent, MP and H are similar. T1000, PROJ and LSE (Liu et al. 2020; see Paragraph "No data (sample-free setting)") also look alike. The closeness between T1000 and PROJ is due to the hyper-planes having similar norms. The resemblance with LSE is due to the log-sum-exp being a smooth approximation of the maximum, which is the logit in this case: the contour lines are parallel to the hyper-planes except where there is an ambivalence regarding the class to predict. Overall, LSE is similar to the other two and will not be included for comparison. From this, PROJ is not expected to be significantly different from T1000 for symmetric problems.

Six sites have been marked on the plots: near the center (site 1), in the midst of a class (site 2), in the direction of a class but further away (site 3), in a class-less region near a boundary (site 4), remote from all classes but at a distance comparable (slightly greater) to the class center (site 5), and on the boundary but further away than the class centers (site 6). The optimality conditions suggest that ID samples cannot appear at sites 1 and 5 (otherwise the model would not be good) and are unlikely at sites 3 and 6 (the gradient would die out before the samples are pushed that far; *cf.* Eq. 3.52 and 3.113). It is hard to say where OOD samples may appear in practice, however. Most works implicitly assume they would be found on the 1-5-sites diagonal (or any high-dimension equivalent), usually more closely to site 1. Whether sites 3 and 6 are realistic OOD cases is unclear. Capturing OOD samples which fall near an ID mode (such as site 2) seems impossible.

Interestingly, different methods capture different sites. Site 1 is the most often captured site, although ANG and the loss gradient magnitude (LGM; see Lee and AlRegib 2020 and Paragraph "Both ID and OOD, labeled") might fail. Site 3 and 6 are the hardest to capture; only a method based on the distance to the class centers seems to be working. The isotropic nature of NORM seems off for this problem. Site 5 is easily flagged out by MP, H, ANG and somewhat LGM the distance-to-center indicators but is easily missed by T1000 and its group.

Some caution must be exerted regarding this discussion, however. For once, the intuition gathered with the 2D case might fail to translate to high dimension problems (which most practical ones are). Secondly, this is illustrated on a well-behaved toy problem. As Section 3.6.2.1 showed classes might not be so well distributed in the latent space—although that case must also be subjected to the high-dimension reservations.

7.4.2 Batchnorm-based indicators

Beyond optimality conditions, the presence of batch normalization layers (Ioffe and Szegedy, 2015) offers the opportunity to define additional indicators. Indeed, those layers are based on statistical parameters directly estimated on the training data, promising a direct route to ID statistical information.

Using batch-normalization-derived features for OOD detection has been proposed by (Quintanilha et al., 2018), however in the context of one-class and supervised OOD detection. Here we propose *indicators* based on them. Such features also tend to be used more and more in the context of data-free compression (Cai et al., 2020b; Yin et al., 2020), which basically relies on the definition of OOD losses.

Batch-normalization has been discussed in Section 3.6.3.4. For the purpose of OOD detection, the interesting part is that it estimates the $\mu_l^{(c)}$ and $\sigma_l^{(c)}$ parameters for each channel $1 \leq c \leq C_l$ of each batch-normalization layer $l \in \mathbb{B}$ ($\mathbb{B} \subset \{1, \ldots, L\}$ being the set of batch-normalization layers). Since those parameters are specific to ID samples, we can hope to use them for OOD rejection. More precisely, defining,

$$M_l^{(c)} = \frac{1}{H_l \times W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} y_l^{(c,w,h)} \tag{7.22}$$

$$S_l^{(c)} = \sqrt{\frac{1}{(H_l \times W_l) - 1} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \left[y_l^{(c,w,h)} - M_l^{(c)} \right]^2} \tag{7.23}$$

$$V_l^{(c)} = (H_l \times W_l - 1) \left[y_l^{(c)} \right]^2 \tag{7.24}$$

where $y_l^{(c,w,h)}$ is $z_{l-1}^{(c,w,h)}$ standardized with $\mu_l^{(c)}$ and $\sigma_l^{(c)}$, we can expect that

$$\mathbb{E}_{\mathcal{I}}\{y_l^{(c)}\} = 0 \tag{7.25}$$

$$\mathbb{E}_{\mathcal{I}}\{M_l^{(c)}\} = 0 \tag{7.26}$$

$$\mathbb{E}_{\mathcal{I}}\{S_l^{(c)}\} = 1 \tag{7.27}$$

$$\mathbb{E}_{\mathcal{I}}\{V_l^{(c)}\} \sim \chi^2_{(H_l \times W_l - 1)} \tag{7.28}$$

Given these conditions, we propose to derive the following indicators (where $\mathbb{B}_s \subseteq \mathbb{B}$ is a subset of batchnorm layers):

$$\text{DMS}_{\mathbb{B}_s} = \frac{1}{C_{\mathbb{B}_s}} \sum_{l \in \mathbb{B}_s} \sum_{c=1}^{C_l} \left(M_l^{(c)} \right)^2 \tag{7.29}$$

$$\text{DMS-AOS}_{\mathbb{B}_s} = \frac{1}{C_{\mathbb{B}_s}} \sum_{l \in \mathbb{B}_s} \sum_{c=1}^{C_l} \frac{1}{H_l \times W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \left(y_l^{(c)} \right)^2 \tag{7.30}$$

$$\text{DSS}_{\mathbb{B}_s} = \frac{1}{C_{\mathbb{B}_s}} \sum_{l \in \mathbb{B}_s} \sum_{c=1}^{C_l} \left(y_l^{(c)} - 1 \right)^2 \tag{7.31}$$

$$\text{DSS-EXT}_{\mathbb{B}_s} = \frac{1}{C_{\mathbb{B}_s}} \sum_{l \in \mathbb{B}_s} \sum_{c=1}^{C_l} \mathbb{I} \left[\text{ext}^{(\alpha)}_{\chi^2_{(H_l \times W_l - 1)}} \left(V_l^{(c)} \right) \right] \tag{7.32}$$

where $C_{\mathbb{B}_s} = \sum_{l \in \mathbb{B}_s} C_l$ is the total number of channels in the considered set,

TABLE 7.3: Summary of sample-free indicators and their bounds, when available.

$0 \leq$ MP $\leq 1 - 1/K$	$0 \leq$ ANG++ ≤ 1
$0 \leq$ H $\leq \log K$	$0 \leq$ IN-DMS
$0 \ll$ T1000 $\leq 1 - 1/K$	$0 \leq$ IN-DMS-AOS
$0 \ll$ ODIN $\leq 1 - 1/K$	$0 \leq$ IN-DSS
NORM	$0 \leq$ IN-DSS-EXT ≤ 1
$0 \leq$ ANG ≤ 1	$0 \leq$ DMS
PROJ	$0 \leq$ DMS-AOS
ACT	$0 \leq$ DSS
ACT+	$0 \leq$ DSS-EXT ≤ 1

and $\text{ext}_{\mathcal{A}}^{(\alpha)}$ is true only when its argument has a (bilateral) p-value according to law $\mathcal{A}$ below the significance level α.

DMS/DSS stands for **d**eparture from the **m**ean/**s**tandard deviation **s**tandardization, and AOS stands for **a**verage **o**f sum. In the remainder we will assume $\alpha = 0.1$ for DSS-EXT.

The intuition behind DMS, DMS-AOS and DSS is that they should produce small values on $\mathcal{I}$. Note however that if $\mathbb{B}_s$ contains many layers, the value might rise quickly since inter-channel correlations are expected. The intuition behind DSS-EXT is that the variance of $y_l^{(c,w,h)}$ should not produce extreme values too often on $\mathcal{I}$.

Relevant subsets. Since the input vectors of the network must also be standardized, we treat the preprocessing as a pseudo-batchnorm layer. Some subsets of batchnorm layers might work better than others. However, in the absence of data, we cannot hope to learn which one is the best. In consequence, we propose to focus on two sets: (i) the input pseudo-batchnorm layer, and (ii) all the layers. We denote by the prefix IN- all indicators relating solely on the input normalization so that IN-DMS $=$ DMS$_{\{1\}}$. We will refer to those as the IN- indicators. We also drop $\mathbb{B}_s$ from the notation when $\mathbb{B}_s = \mathbb{B}$.

7.4.3 Summary

Table 7.3 summarizes the sample-free indicators and their bounds, when available. All indicators are such that ID samples should portray small values. Although interpretable, probability-based indicators (MP, H, T1000, ODIN) are not necessarily easier to bound (See Table 7.4 for some statistics about the indicator distributions). Unbounded indicators are *de facto* harder to use in a sample-free setting.

7.5 Empirical study

In this section, we evaluate how the proposed indicators perform individually. After detailing our methodology (Section 7.5.1.1), our main results are

TABLE 7.4: Percentiles of the indicator distributions. The indicators were extracted from a DenseNet 121 learned on CIFAR 10. See Section 7.5.1.1 for more information about the network and the dataset.

	CIFAR 10 (TEST SET)			TINY IMAGENET		
	P25	P50	P75	P25	P50	P75
MP	0.000	0.000	0.003	0.014	0.112	0.339
H	0.000	0.004	0.036	0.123	0.628	1.233
T1000	0.898	0.898	0.899	0.899	0.899	0.899
ODIN	0.898	0.898	0.898	0.899	0.899	0.899
NORM	−6.97	−6.34	−5.78	−6.63	−6.02	−5.46
ANG	0.267	0.314	0.391	0.479	0.577	0.657
PROJ	−4.87	−4.28	−3.61	−3.28	−2.55	−1.96
ACT+	−12.9	−11.5	−10.1	−10.2	−8.78	−7.54
ANG+	0.231	0.266	0.319	0.361	0.424	0.477
IN-DMS	0.405	0.677	1.065	0.430	0.735	1.154
DMS	9.377	10.36	11.49	8.722	9.634	10.78
IN-DSS	0.236	0.423	0.669	0.267	0.450	0.670
DSS	7.690	8.116	8.661	7.850	8.385	9.112
1C-SUM	-153.4	-138.5	-123.0	-120.6	-96.9	-79.2

discussed in Section 7.5.1.2. A more thorough examination of the problem of semantic and covariate shifs is conducted in Section 7.5.2.

Section 7.5.3 goes over additional findings. Namely, Section 7.5.3.1 discusses the redundancy and complementarity between indicators. Since some indicators are based on the optimality assumption, Section 7.5.3.2 examines how the indicators fare when this assumption is not fully met. Finally, Section 7.5.3.3 is concerned with how the indicators can be used to detect misclassified samples and tackled both this task and OOD detection.

7.5.1 Main experiment

7.5.1.1 Protocol

ID tasks. In order to evaluate the indicator performances, we have trained three networks on three image classification tasks to serve as ID datasets, namely, we used CIFAR 10, CIFAR 100 (Krizhevsky, Hinton, et al., 2009) and ImageNet (Deng et al., 2009). Experiments were all carried out with PyTorch (Paszke et al., 2017).

CIFAR datasets consist in 60000 32×32 RGB images. There is a standard train/test split of respectively 50000 and 10000 images. CIFAR 10 has 10 classes, while CIFAR 100 has 100. The datasets are balanced class-wise (across both train and test sets). In the case of ImageNet, we followed the standard procedure to rescale the RGB images and extract centered 224×224 crops on the 100000 test images spread among 1000 classes.

The networks are a ResNet 50 (He et al., 2016), a WideResNet-40 (Zagoruyko and Komodakis, 2016) and a DenseNet 121 (Huang et al., 2017). All three architectures are ReLU-based and output non-negative latent vectors. They also include batch-normalization layers.

TABLE 7.5: Model Performance (in %) on the ID task.

	ACCURACY		IMAGENET	
	CIFAR 10	CIFAR 100	TOP-1 ERROR	TOP-5 ERROR
RESNET 50	94.11 ± 0.25	77.48 ± 0.23	23.85	7.13
WIDERESNET	94.18 ± 0.31	74.17 ± 0.72	21.49	5.91
DENSENET 121	94.30 ± 0.31	77.89 ± 0.04	25.35	7.83

TABLE 7.6: OOD dataset characteristics.

GAUSSIAN	$32 \times 32 \times 3$	$\mu = 0.5, \sigma = 0.25$ CLIPPED ON $[0, 1]$
SVHN	$32 \times 32 \times 3$	NETZER ET AL. (2011)
MNIST	28×28	LECUN ET AL. (1998A)
FASHION MNIST	28×28	XIAO, RASUL, AND VOLLGRAF (2017)
TINY IMAGENET	$64 \times 64 \times 3$	DENG ET AL. (2009)
LSUN	$256 \times 256 \times 3$	YU ET AL. (2015)[2]
CIFAR 10/100	$32 \times 32 \times 3$	KRIZHEVSKY, HINTON, ET AL. (2009)

On CIFARs, they expect input of size 32×32 and were trained for 450 epochs by stochastic gradient descent (batches of size 128, weight decay of 5×10^{-4} and momentum of 0.9). The learning rate was initialized at 0.1. It was decreased by a factor 10 after 150 epochs and again at epoch 300. Each decrease was accompanied by a restart from the best model according to the validation accuracy. Horizontal flip and random cropping (with a padding of 4) were used as data augmentation.

On ImageNet, we used the pre-trained networks available in PyTorch which expect 244×244 RGB images as input.

Table 7.5 gathers the accuracy of each model.

OOD datasets. For each ID dataset, we will consider multiple OOD datasets (see Table 7.6 for the details) whose proximity with the ID data will vary, offering a broad spectrum of cases to assess on which tasks each indicator is effective.

To be more precise, Gaussian is a dataset of generated images. Although there exists a chance to produce any images with this distribution, the probability to produce one (or enough to skew the results) images which could be confounded with ID samples is as good as nonexistent. MNIST and fashion MNIST are gray-level images and fall into the "gross statistical differences" category, as well as having different label spaces. Compared to the synthetic Gaussian dataset, structures are still present in the images. SVHN (Street View House Numbers) is a clear semantic shift and is expected to have an input distribution somewhat different from the natural images contained in the other dataset. Although the label space of LSUN is technically disjoint of the ID dataset, objects in the scene might fall into one of their classes. The overlap is more pronounced for the CIFAR 10(0) and (Tiny)Imagenet datasets.

In all cases, the ground truth was established under the source identification formulation. All images were resized and cast to RGB when needed, then rescaled in the range $[0, 1]$ and normalized channel-wise according to the ID dataset input statistics, as is expected when operating the network.

ID/OOD balance. Except in the case of the supervised approach (see main text), the whole ID test set and the whole OOD dataset are used to assess the indicator performances. As a consequence, the classification task is quite unbalanced (as might be the case in a real setting, although we might expect a much higher proportion of ID samples). For artificial datasets, we generated 50000 samples.

Metric. We tackle the problem from the OOD rejection perspective. This means we consider OOD samples as *positive* and use the dataset origin as ground truth (source identification). We use the test sets of CIFAR 10, CIFAR 100 or ImageNet as *negative* (ID) samples. Those have never been seen during training.

We report the area under the ROC curve (auroc) for each indicator used to discriminate between positive (OOD) and negative (ID) samples. Most papers in the domain also report the OOD rejection rate for a fixed ID acceptance rate. In our setting, ID samples are not available, and setting the threshold at a given acceptance rate is a challenge in itself. Besides, risk-balancing is application-dependent, therefore privileging a risk-independent metric should be more helpful in general. Contrary to precision-recall curves, ROC curves are fully independent of the—typically unknown—proportion of ID/OOD samples. We therefore feel auroc is the most relevant metric.

Pre-processing. Prior to running through the network, all images (ID and OOD) are resized to fit the network expected size, transformed to RGB if necessary, rescaled in the range $[0, 1]$ and then normalized channel-wise according to the ID dataset input statistics (see next paragraph). Artifacts due to resizing may help detect OOD samples. In the case of artificial datasets, images are generated with the appropriate size.

Input normalization. For CIFARs, we estimated the channel mean/standard deviation on the training set. Regarding the standard deviation, we computed the square root of the *total* variance, in accordance with PyTorch's batchnorm implementation. For some reasons, available statistics usually used the average *intra-image* variance, disregarding the *inter-image* variance. Admittedly, the difference is slight.

For ImageNet, we re-used the pre-trained network and thus conformed to using the same statistics as were used for training (based on intra-image variance).

Variability. On CIFARs, results are established on three random initializations of the network's parameters and is, with batch sampling, the only sources of randomness; artificially-generated datasets are the same throughout the experiments. Since we re-used pre-trained models for ImageNet, there is only one experiment per network (there is only one set of weights available per architecture). Note that the IN- indicators are independent of the network; they only depend on the input, channel-wise statistics of the ID datasets. As such, they are not subject to randomness.

Supervised results. We also include supervised results. In that case, half of the ID testing set and half of the OOD data are used to build a linear SVM (Cortes and Vapnik, 1995). The remaining half is used to evaluate the indicators. This means that the training and testing OOD samples are from the *same* distribution—the most favorable case, totally outside of our setting. These results are only reported for comparison purposes. It is worth noting that the supervised approach performs almost perfectly on the easy tasks and is almost always best on the hard ones.

7.5.1.2 Results

Table 7.7 shows areas under the ROC curves (auroc) for OOD detection with CIFAR 10 as the ID set on ResNet 50. Detailed tables for the other ID sets and networks are present in Appendix B.2. Table 7.8 summarizes the average rank (over all the OOD datasets) of each indicator for all settings. Note that the ranking is sensitive to the choice of OOD datasets, although major trends seem stable. For the purpose of this section, the last line can be ignored.

Baseline indicators. ODIN performs well in the case of ImageNet. On CIFARs, it is less clear whether the cost of the backward pass is worth it compared to simply using T1000. As envisioned in the previous section, H is slightly better than MP, although ODIN and T1000 are better suited as single indicators.

Batchnorm indicators. They do not work consistently. For these indicators, the OOD dataset has a high impact on the ranking and results are better understood by looking individually at the datasets (*e.g.* Table 7.7). They are intuitive, however. Indicators based on the input normalization work only on grey-level datasets. When input statistics are close to the ID's (Tiny ImageNet, LSUN), those indicators do not work better than random. They also fail on the noisy Gaussian dataset, which has individual pixel statistics that are close to ID's. It would be easy to reject such samples if inter-channel information were available, as demonstrates the indicators based on all batchnorm layers for which such information is made available thanks to the convolutions. Overall, it is clear that, in our setting, batchnorm indicators can only discriminate specific OOD sets.

It is interesting to note that these performances are roughly equivalent on CIFAR 10 and CIFAR 100, implying these features do not rely on the assumption the model performs well at its original task.

TABLE 7.7: Area under the ROC curve for OOD detection with CIFAR 10 as ID on ResNet 50. Shading highlights the 50% best scores per column (darker is better). The scores are averaged over three runs (*i.e.* network initializations). Note that IN- indicators are independent of the network, hence the single value.

	GAUSSIAN	SVHN	MNIST	FASHION MNIST	TINY IMAGENET	LSUN (TEST SET)
ODIN	91.36 ± 5.42	90.22 ± 4.03	96.88 ± 0.70	95.89 ± 0.75	87.22 ± 2.12	92.38 ± 1.56
T1000	83.17 ± 9.00	93.14 ± 3.05	94.81 ± 0.78	95.43 ± 0.62	88.70 ± 1.23	92.66 ± 1.04
MP	89.27 ± 4.90	91.89 ± 1.30	90.76 ± 0.65	91.97 ± 0.47	87.05 ± 0.61	90.08 ± 0.60
H	89.05 ± 5.03	92.51 ± 1.46	91.40 ± 0.62	92.71 ± 0.58	87.52 ± 0.67	90.62 ± 0.59
NORM	53.96 ± 33.02	85.46 ± 10.89	92.28 ± 4.92	89.52 ± 4.00	80.19 ± 4.27	82.50 ± 4.93
NORM+	54.99 ± 28.60	87.17 ± 9.12	94.61 ± 2.09	92.92 ± 1.85	85.00 ± 2.61	88.87 ± 2.82
ACT	83.34 ± 9.02	93.32 ± 2.95	94.90 ± 0.70	95.47 ± 0.59	88.77 ± 1.18	92.50 ± 1.08
ACT+	87.68 ± 9.18	94.23 ± 3.50	96.03 ± 1.44	95.93 ± 0.72	88.05 ± 1.53	91.68 ± 1.38
PROJ	85.53 ± 8.09	94.01 ± 2.42	95.61 ± 0.40	95.47 ± 0.58	88.61 ± 1.26	92.05 ± 1.21
ANG	91.78 ± 2.79	93.41 ± 0.09	94.15 ± 0.60	94.76 ± 1.02	88.35 ± 0.51	91.98 ± 0.58
ANG++	99.89 ± 0.12	97.26 ± 0.17	94.25 ± 1.22	93.41 ± 1.70	86.05 ± 0.88	88.43 ± 0.75
IN-DMS	7.85	60.46	98.59	71.94	52.89	49.26
IN-DMS-AOS	52.79	30.41	99.68	96.02	52.55	54.91
IN-DSS	5.13	85.99	36.16	58.53	52.03	42.94
DMS	100.00 ± 0.00	80.29 ± 8.30	93.97 ± 2.47	69.39 ± 6.49	34.21 ± 5.54	22.67 ± 5.33
DMS-AOS	99.25 ± 0.48	4.72 ± 2.26	81.12 ± 9.04	59.42 ± 9.53	25.25 ± 2.65	23.78 ± 2.66
DSS	99.86 ± 0.14	96.51 ± 0.60	70.33 ± 15.30	62.22 ± 3.53	55.01 ± 1.90	47.40 ± 4.40
DSS-EXT	98.24 ± 0.61	97.70 ± 0.34	66.93 ± 1.88	67.64 ± 1.67	66.84 ± 0.88	62.94 ± 1.38
SUPERVISED	100.00 ± 0.00	99.75 ± 0.05	100.00 ± 0.00	99.70 ± 0.03	90.82 ± 0.45	96.14 ± 0.19
1C-SUM	97.84 ± 2.70	97.83 ± 0.95	96.47 ± 1.58	95.86 ± 0.63	88.86 ± 0.79	91.61 ± 0.90

Latent space indicators. As expected, NORM and NORM+ do not convey the appropriate information. The remaining indicators rank well, however. On ImageNet, positive-only indicators seem to work better, while this is not as clear for the other ID tasks. In particular, ANG++ performs better than ANG on ImageNet but ANG works better in the other settings (except for ResNet 50 on CIFAR 100).

Once again, the OOD dataset has an impact on the ranking: ACT/ACT+ tend to struggle with (fashion) MNIST on CIFAR 100 and ImageNet (Appendix C.1), while, with ImageNet as ID task, ANG++ comes way ahead of the other indicators against CIFARs as OOD but underperforms on LSUN. On the hardest cases with CIFARs as ID tasks (*i.e.* rejecting Tiny ImageNet/LSUN samples) ODIN does not perform better than T1000.

Discussion. Batchnorm indicators can capture gross statistical differences but fail on more challenging tasks. For those, optimality-based indicators are more appropriate. In a few instances, ANG/ANG++ perform extremely well. ODIN is also a strong baseline *if* the cost of the backward pass can be paid. Note however that the gap between ODIN and ANG++ is usually wider when the former underperforms (*e.g.* Gaussian and SVHN on Table 7.7), suggesting that ANG++ is more robust besides being faster to compute.

Since PROJ and ACT/ACT+ are harder to bound, they are also harder to use in a sample-free context. On that matter, Table 7.4 displays some statistics about a few indicators. As can be seen, pinpointing where the threshold should be placed is not easy on challenging tasks, at least without data. This will be discussed further in Section 7.7.

7.5.2 Semantic and covariate closeness

In this section, we take a closer look at how the indicators perform with respect to semantic and covariate shifts. We derive two sets of partition from CIFAR 10 illustrated in Table 7.9 (top part). The first partition correspond to the first five classes (`h1`) and the remaining ones (`h2`). The second partition splits the animals (`h3`) from the vehicles (`h4`). Note that the all of the base task are balanced (roughly 4500 images per class in all cases).

Following the protocol of the previous section, ResNet 50 models have been trained on each sub-dataset. As can be gleaned from Table 7.9, the accuracies are roughly in the same range, except on `h3` where it is significantly smaller, possibly an effect on having more classes. The effect of the model quality is discussed in Section 7.5.3.2.

The `h1`/`h2` partition is both semantically close (having both animals and vehicles) and statistically close, at least at the level of the input statistics (Table 7.9 bottom three rows). On the other hand, the `h3`/`h4` partition is more varied both semantically and (consequently) statistically. We would like to insist on the fact that the OOD images have been normalized according to the input statistics of the ID task.

Table 7.10 gathers the results for the two partitions using one of the subsets as ID base task and the other as OOD. The first thing to note is that the

TABLE 7.8: Average indicator rank (lower is better). These are the average across datasets of the indicator rank per dataset. R50, W and D121 stand for ResNet 50, WideResNet and DenseNet 121, respectively. Shading highlights the 50% best (*i.e.* topmost) scores per column (darker is better).

	CIFAR 10			CIFAR 100			IMAGENET		
	R50	W	D121	R50	W	D121	R50	W	D121
ODIN	7.30	8.20	10.70	7.20	7.30	6.30	3.40	4.60	4.90
T1000	7.80	7.30	9.30	7.50	8.00	6.70	9.30	9.90	10.30
MP	11.80	11.80	8.80	9.80	13.80	10.00	12.60	11.60	13.40
H	11.00	10.20	7.30	12.50	10.30	11.20	8.90	8.60	9.60
NORM	14.50	13.20	18.00	14.70	15.70	14.70	13.70	18.30	14.90
NORM+	12.30	10.30	15.20	12.80	14.20	13.30	12.30	17.00	13.10
ACT	7.00	6.30	8.50	7.20	8.00	6.70	9.40	10.00	10.10
ACT+	7.00	7.70	12.80	8.00	8.00	8.20	6.70	9.60	9.00
PROJ	7.20	6.00	7.70	9.30	7.30	8.80	9.70	9.90	9.10
ANG	8.20	9.00	4.50	7.80	8.30	7.00	11.30	8.40	10.90
ANG++	8.50	13.70	7.30	4.30	10.70	8.80	4.70	3.10	5.10
IN-DMS	14.50	14.80	14.50	14.80	15.00	14.80	17.00	15.60	16.40
IN-DMS-AOS	12.30	12.20	11.70	11.00	12.70	11.30	14.70	14.00	14.10
IN-DSS	18.50	18.50	17.70	18.00	15.50	17.80	17.60	17.00	17.40
DMS	14.50	12.00	11.00	16.30	9.50	16.20	8.30	10.40	12.60
DMS-AOS	16.70	16.80	17.20	13.30	16.70	13.80	18.30	16.00	17.70
DSS	12.80	12.00	11.30	19.50	7.80	19.30	11.40	9.30	6.70
DSS-EXT	12.20	14.30	10.80	9.20	13.30	9.20	14.30	11.40	9.30
SUPERVISED	1.00	1.00	1.00	1.30	1.80	1.70	1.00	1.00	1.00
1C-SUM	4.80	4.70	4.70	5.30	6.00	4.20	5.40	4.40	4.30

results are not symmetric, at least in the case of the h1/h2 partition. Training a network on h1, it is much more difficult to detect h2 samples than the other way around. This is not true for the h3/h4 partition. Interestingly, this does not seem to originate from the quality of the model as the worst models are those of h3. Where does the asymmetry stem from remains an open question.

Without surprise, the batchnorm indicators are struggling with the statistical closeness. Only DSS and DSS-EXT seem helpful in the h3/h4 case, assuredly due the wider gap in input statistics.

Overall, the results are better on the h3/h4 partition than on h1/h2. This is unsurprising since the latter pair is much closer semantically and closer statistically as well.

As far as individual indicators are concerned, the situation is quite different between partitions. On h1/h2, T1000, MP and H perform best, with ACT and PROJ tagging along, whereas ANG is lagging behind. This is an interesting piece of geometrical trivia since looking at the norm of the latent vector (*cf.* NORM) or its orientation (*cf.* ANG) is not as good as looking at both at the same time (*e.g.* PROJ). On h3/h4, the situation is more standard.

Once again, the supervised models work best and bring a significant increase in auroc scores, suggesting the sample-free indicators convey meaningful information.

TABLE 7.9: Partitions over CIFAR 10. The accuracy (expressed in percent) is established on a ResNet 50 network. Stat. represents the channel-wise averages and standard deviations in the input space.

		H1	H2	H3	H4
CLASSES		AIRPLANE	DOG	BIRD	AIRPLANE
		AUTOMOBILE	FROG	CAT	AUTOMOBILE
		BIRD	HORSE	DEER	SHIP
		CAT	SHIP	DOG	TRUCK
		DEER	TRUCK	FROG	-
		-	-	HORSE	-
ACCURACY		96.05 ± 0.30	97.10 ± 0.25	93.17 ± 0.56	96.36 ± 0.16
STAT.	R	0.49 ± 0.25	0.49 ± 0.25	0.49 ± 0.24	0.50 ± 0.26
	G	0.49 ± 0.24	0.48 ± 0.25	0.47 ± 0.23	0.51 ± 0.26
	B	0.45 ± 0.26	0.44 ± 0.26	0.40 ± 0.24	0.52 ± 0.27

Conclusion. It is clear that the closer the OOD set (both semantically and statistically) the harder to tell it apart it is. On h3/h4 the overall detection performances are convincing. On the much closer partition, however, detection suffers severely. It should come as no surprise that tackling such a difficult OOD task with no data is so challenging—even the over-optimistic supervised method suffers.

Interestingly, it has also been shown that OOD detection is not always symmetric.

7.5.3 Additional results

This section investigates several related topics. In Section 7.5.3.1 the redundancy between indicators is analyzed. Section 7.5.3.2 investigates how the quality of the model impacts the performance of the indicators. Finally, Section 7.5.3.3 takes a look at the task of detecting misclassification in the network prediction, as well as the joint task of misclassification and OOD detections.

Unless otherwise specified, the protocol follows the one of Section 7.5.1.1.

7.5.3.1 Complementarity/redundancy

We used principal component analysis (PCA) to assess the (linear) redundancy or complementarity of the proposed indicators. For each dataset independently, we created one unsupervised $n \times p$ matrix M_d. $M_d[i, j]$ is value of the jth indicator for the ith sample from dataset d, standardized according to the mean/standard deviation of indicator j on d. We then computed the PCA of each matrix (Figure 7.11).

Figure 7.11a shows how the ratio of explained variance evolves with respect to the number of principal components. The first component accounts for 50% of the variance and roughly half of the components are needed to account for the majority (*i.e.* $> 95\%$) of the variance.

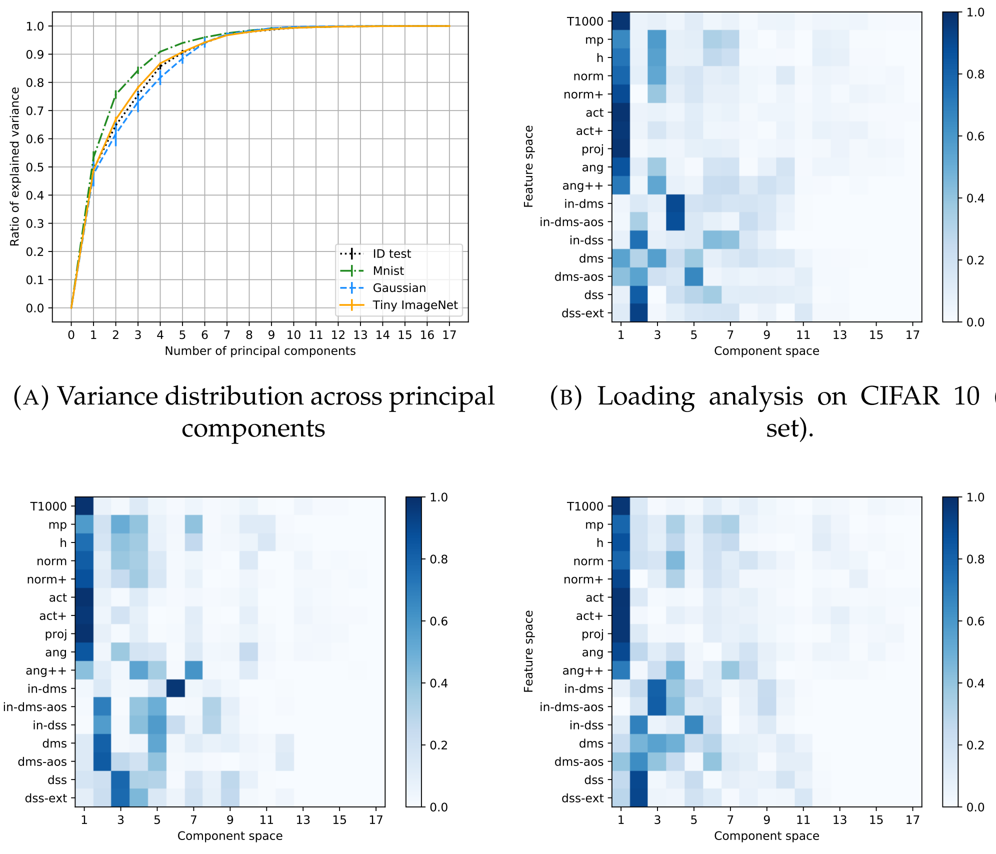
TABLE 7.10: Area under the ROC curve for OOD detection with the h1/h2 and h3/h4 partitions on ResNet 50. *The first subset corresponds to the ID base task and the second to the OOD dataset.* Shading highlights the 50% best scores per column (darker is better). The scores are averaged over three runs (*i.e.* network initializations). Note that IN- indicators are independent of the network, hence the single value. 1C-Sum* refers to the summary indicator that includes no batchnorm indicators.

INDICATOR	H1 (ID) / H2 (OOD)	H2 / H1	H3 / H4	H4 / H3
T1000	78.02 ± 0.69	85.48 ± 0.11	90.89 ± 0.60	90.12 ± 1.33
MP	78.37 ± 0.45	85.25 ± 0.31	88.16 ± 0.18	88.08 ± 0.91
H	78.43 ± 0.45	85.38 ± 0.27	88.53 ± 0.19	88.33 ± 1.00
NORM	71.55 ± 2.48	81.92 ± 1.34	87.76 ± 1.50	89.54 ± 1.74
NORM+	74.40 ± 1.79	83.46 ± 0.86	89.75 ± 1.14	89.66 ± 1.51
ACT	77.94 ± 0.72	85.25 ± 0.15	91.03 ± 0.61	91.10 ± 1.26
ACT+	75.95 ± 1.42	84.52 ± 0.19	91.38 ± 0.74	91.75 ± 1.36
PROJ	77.20 ± 0.75	85.04 ± 0.16	91.01 ± 0.63	91.30 ± 1.19
ANG	75.79 ± 0.97	83.42 ± 0.16	89.20 ± 0.64	90.05 ± 1.22
ANG++	76.06 ± 0.86	78.75 ± 0.36	84.69 ± 2.54	83.22 ± 3.02
IN-NOTA	45.12	55.28	54.17	52.69
IN-DMS	43.87	56.43	58.77	56.95
IN-DMS-AOS	53.58	46.71	66.88	41.47
IN-DSS	41.47	59.04	37.64	65.19
NOTA	53.14 ± 4.43	49.67 ± 2.39	47.79 ± 2.73	52.71 ± 8.25
DMS	24.92 ± 1.64	25.42 ± 0.50	32.60 ± 1.85	31.60 ± 3.32
DMS-AOS	33.61 ± 1.99	27.36 ± 0.87	29.85 ± 4.53	15.63 ± 1.72
DSS	51.49 ± 1.75	58.34 ± 2.09	62.28 ± 4.29	75.58 ± 4.76
DSS-EXT	57.63 ± 1.55	62.69 ± 1.62	72.98 ± 1.71	80.04 ± 1.55
1C-SUM	73.52 ± 2.10	83.82 ± 0.49	92.04 ± 0.58	91.97 ± 0.54
1C-SUM*	77.33 ± 0.77	85.06 ± 0.18	91.00 ± 0.68	90.99 ± 1.27
SUPERVISED	83.04 ± 0.43	87.60 ± 0.80	95.06 ± 0.52	94.90 ± 0.52

Figures 7.11b-7.11d show how the components relate to the original indicators. As can be seen, the first component mainly focuses on the optimality-based indicators. The large quantity of variance it explains is somewhat misleading since more than half of the indicators are so highly correlated themselves. Interestingly, ANG++, H, MP, NORM and NORM+ correlations are spread among two components, mainly, suggesting those might be complementary to the other optimality-based indicators.

To a lesser extent, the second component focuses on the batchnorm indicators. However, several components are needed to fully capture all the batchnorm information. In particular, IN-DMS(-AOS) stand apart from the other indicators. As for IN-DSS, it tends to share its variance with two components. Of the remaining indicators, DSS and DSS-EXT are well correlated and are the main focus of one component. On natural images, DMS and DMS-AOS correlate with several components. On Gaussian noise, though—where they perform well—they stand apart as the second component.

Overall, we observe three main clusters of indicators: (i) the optimality-based ones, (ii) IN- indicators, and (iii) the remaining batchnorm ones. Nevertheless, more than three components are needed to summarize (most of)



(A) Variance distribution across principal components

(B) Loading analysis on CIFAR 10 (ID set).

(C) Loading analysis on Gaussian noise

(D) Loading analysis on Tiny ImageNet.

FIGURE 7.11: PCA analysis of redundancy on CIFAR 10 with ResNet 50. In the loading analyses, pixel on row *i* and column *j* expresses the absolute value of the correlation between the *i*th indicator and the *j*th component.

the variance, since some indicators spread their variance across several components. Although partially redundant, the indicators remain complementary and might help catch different OOD samples.

This analysis suggests that the first few principal components could be used as new indicators that summarize the information contained in the proposed ones. Nevertheless, using the principal components in practice is not trivial, owing to the by-dataset standardization. One could use a proxy dataset on which to perform the PCA reduction. We are concerned, however, that this would further bias the detector to perform well mostly on closer OOD datasets to the one used as reference, and degrade the performances in the other cases.

7.5.3.2 Model quality

In this section, we would like to investigate how much the quality of the model impacts the quality of OOD detection. To do so, we trained a ResNet 50 on CIFAR 10 and paused the learning at several stages to compute the proposed indicators. We used the accuracy on the training set as criterion to snapshot the performances. Table 7.11 holds the results with a selection of indicators for several datasets, as well as the test set accuracies.

We can distinguish between three phases. At first, the model is training but not yet overfitting. Around 95% of training accuracy, we see the first evidence of overfitting occurring. At 99%, overfitting is no longer mild. At this point, the gain of accuracy is small and the network decreases its training loss mainly by becoming more confident. A few observations are worth mentioning.

Optimality-based indicators. Without surprise, optimality-based indicators suffer from a sub-optimal network. At the first stage, the proportion of misclassified test samples is relatively high. Since these indicators all rely in some way or another on the predicted class, poor performances are to be expected. Once reaching 95% of training accuracy, the proportion of misclassified test samples remains stable, and the performances vary less, up to the point where the model becomes overconfident. It does indeed seem easy for the network to push all samples far away from the decision planes in the last latent space, with a regrettable side effect for OOD detection. In this regime, the variance of the results increases as well.

Batchnorm indicators. Firstly, note we did not include the IN- indicators because they do not depend on the model. On datasets where they are useful (Gaussian, SVHN), the other batchnorm indicators reach noticeable performances early in the training but keep being refined up till the end. Overconfidence is not a problem for those indicators. They are quite unstable on MNIST and overall worthless against Tiny ImageNet.

Supervised method. When supervision is applicable, the model quality plays a much less important role. Except on Tiny ImageNet, the indicator performances with the model trained at 75% of accuracy are already close to their best. Even at the first stage, the linear SVM model is able to discard useless indicators (as against Gaussian, where individual optimality-based indicators perform randomly). As showcases MNIST, the supervised models also go beyond picking up the best individual indicator. This approach is also more robust with respect to overconfidence.

Discussion. As expected, optimality-based indicators suffer from a sub-optimal model. They also suffer from an over-confident network. Batchnorm indicators are less predictable but seem to be also impacted by the model quality when they are useful. Some supervision can compensate for the lack of training.

7.5.3.3 Misclassification detection

In this section, we investigate whether wrongly rejected ID samples correspond to misclassified ones. We performed two experiments.

Error detection. The first experiment consists in using the proposed indicators to detect misclassifications: we only look at the training set of the base task and label as positive the samples for which the network makes a classification error; samples for which the model is correct are labeled as negative. Table 7.12 shows the area under the ROC curve for detecting these positive samples. As can be seen, not all indicators are equal in this respect. Indicator appropriateness is stable across architectures and datasets.

Optimality-based indicators are clearly best suited: a well-optimized network should lower its confidence when making a mistake. Among those, MP and H stand out, then comes ANG, followed by ACT, PROJ and T1000. For this task, the positive variant are less adequate. T1000 always outperforms ODIN; the adversarial perturbation will lower the network confidence blurring the separation between positive and negative samples.

Batchnorm indicators are not suited for the task, suggesting that the network mistakes are not due to statistical outliers.

Even though the networks make more mistakes on CIFAR 100, detecting them is a harder challenge. Whether this is caused by a less well-performing model, or by other factors (such as the number of classes which might spread the predictions more across classes) is not clear.

Joint OOD and misclassification detection. In the second experiment, we are considering both OOD samples and misclassification as the positive class. In other words, are considered negative samples only those of the base task for which the network predicts the correct class. Although misclassifications cannot count as OOD samples *per se*, it might be more interesting in practice to reject those as well.

Table 7.13 shows the average (over the OOD datasets—the same as for the other experiments) improvement in auroc when tackling the joint task rather than OOD detection only. It is confirmed that wrongly rejected ID samples are partly due to misclassified ones, in the case of optimality-based indicators. Indeed, they benefit from a raise of auroc, which is more pronounced with CIFAR 100, where there are much more classification errors.

We also see that this is not true of batchnorm indicators, albeit IN-DMS and IN-DSS see a small improvement. This suggests that misclassified samples are not necessarily statistically off compared to other ID samples.

The previous analyses highlighted that indicators good at detecting misclassifications might differ from those best at OOD detection. However, when the network performs well, misclassified samples should be negligible. This is confirmed by table 7.14, which displays the average (across OOD datasets—the same as for the other experiments) top rank for each indicator at this joint task. On CIFAR 10, the relative order of indicators is mostly unchanged. On CIFAR 100, MP and H are better positioned in the ranking, although the number of mistakes might be too low for them to outperform the best indicators.

Overall, it does seem there is an overlap between some of the wrongly rejected ID samples and the misclassified ones. On the other hand, the fact that indicators best at each task are not the same suggests misclassified and OOD samples might follow different patterns.

TABLE 7.11: Model quality and its impact on OOD detection. A ResNet 50 was trained on CIFAR 10 and paused when reaching some training set accuracy threshold (first row) to examine how the features perform. The metric is the area under the ROC curve. Coloring reflects the 50% of *overall* best results per dataset.

	TRAIN ACC. (%)	85	95	99	FULL
	TEST ACC. (%)	84.99 ± 0.36	92.41 ± 0.49	93.58 ± 0.06	94.11 ± 0.25
GAUSSIAN	T1000	89.71 ± 2.95	96.95 ± 3.31	95.41 ± 4.59	83.17 ± 9.00
	MP	84.84 ± 5.74	95.06 ± 5.27	93.31 ± 5.37	89.27 ± 4.90
	H	87.59 ± 4.08	95.65 ± 4.80	94.09 ± 5.33	89.05 ± 5.03
	NORM	83.74 ± 1.46	96.57 ± 3.76	94.61 ± 5.47	53.96 ± 33.02
	ACT	89.51 ± 3.40	96.87 ± 3.43	95.48 ± 4.50	83.34 ± 9.02
	PROJ	89.69 ± 4.11	96.57 ± 3.97	95.63 ± 4.02	85.53 ± 8.09
	ANG	89.41 ± 5.93	95.85 ± 4.38	94.73 ± 3.90	91.78 ± 2.79
	DMS-AOS	85.95 ± 4.30	86.66 ± 4.07	95.01 ± 2.37	99.25 ± 0.48
	DSS-EXT	96.60 ± 0.61	98.64 ± 0.55	98.17 ± 0.79	98.24 ± 0.61
	SUPERVISED	100.00 ± 0.00	100.00 ± 0.01	99.99 ± 0.01	100.00 ± 0.00
	1C-SUM	99.54 ± 0.13	99.88 ± 0.16	99.81 ± 0.21	97.84 ± 2.70
SVHN	T1000	94.68 ± 1.68	97.17 ± 1.05	96.49 ± 0.32	93.14 ± 3.05
	MP	92.60 ± 1.17	94.89 ± 1.59	94.32 ± 1.27	91.89 ± 1.30
	H	94.36 ± 1.35	96.36 ± 1.33	95.19 ± 1.14	92.51 ± 1.46
	NORM	95.52 ± 2.72	97.27 ± 1.90	94.55 ± 1.83	85.46 ± 10.89
	ACT	94.94 ± 1.61	97.24 ± 1.05	96.58 ± 0.30	93.32 ± 2.95
	PROJ	94.84 ± 1.35	97.34 ± 0.96	96.68 ± 0.29	94.01 ± 2.42
	ANG	90.84 ± 0.33	95.32 ± 0.29	94.36 ± 1.19	93.41 ± 0.09
	DMS-AOS	11.38 ± 10.15	2.72 ± 1.27	3.92 ± 1.70	4.72 ± 2.26
	DSS-EXT	97.80 ± 0.45	98.40 ± 0.22	97.88 ± 0.18	97.70 ± 0.34
	SUPERVISED	99.50 ± 0.23	99.72 ± 0.01	99.65 ± 0.04	99.75 ± 0.05
	1C-SUM	98.49 ± 0.59	99.00 ± 0.22	98.71 ± 0.17	97.83 ± 0.95
MNIST	T1000	89.35 ± 1.44	94.70 ± 0.18	94.88 ± 2.18	94.81 ± 0.78
	MP	82.44 ± 1.47	90.03 ± 0.77	89.47 ± 2.48	90.76 ± 0.65
	H	85.59 ± 1.33	91.76 ± 0.64	90.41 ± 2.67	91.40 ± 0.62
	NORM	88.82 ± 5.56	94.92 ± 1.60	97.36 ± 0.93	92.28 ± 4.92
	ACT	89.44 ± 1.53	94.73 ± 0.19	94.93 ± 2.19	94.90 ± 0.70
	PROJ	90.42 ± 1.73	95.30 ± 0.22	95.59 ± 1.75	95.61 ± 0.40
	ANG	88.18 ± 0.57	93.65 ± 1.14	91.93 ± 2.90	94.15 ± 0.60
	DMS-AOS	64.11 ± 10.43	57.98 ± 2.28	65.48 ± 5.40	81.12 ± 9.04
	DSS-EXT	68.02 ± 3.52	73.17 ± 1.54	68.64 ± 3.40	66.93 ± 1.88
	SUPERVISED	99.99 ± 0.01	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
	1C-SUM	93.76 ± 1.89	96.37 ± 0.40	96.53 ± 1.73	96.47 ± 1.58
TINY IMAGENET	T1000	83.33 ± 1.67	89.57 ± 0.05	89.46 ± 0.36	88.70 ± 1.23
	MP	80.37 ± 1.76	87.05 ± 0.17	87.20 ± 0.26	87.05 ± 0.61
	H	82.07 ± 1.73	88.23 ± 0.12	87.81 ± 0.28	87.52 ± 0.67
	NORM	80.70 ± 2.21	87.55 ± 0.17	83.57 ± 1.55	80.19 ± 4.27
	ACT	83.09 ± 1.79	89.47 ± 0.09	89.47 ± 0.38	88.77 ± 1.18
	PROJ	82.91 ± 1.80	89.34 ± 0.10	89.30 ± 0.38	88.61 ± 1.26
	ANG	81.02 ± 1.16	87.99 ± 0.15	88.47 ± 0.23	88.35 ± 0.51
	DMS-AOS	29.05 ± 2.07	22.83 ± 1.20	25.31 ± 1.19	25.25 ± 2.65
	DSS-EXT	65.97 ± 1.83	68.01 ± 0.55	67.92 ± 0.39	66.84 ± 0.88
	SUPERVISED	85.22 ± 1.50	90.71 ± 0.27	90.56 ± 0.17	90.82 ± 0.45
	1C-SUM	82.84 ± 2.39	89.33 ± 0.27	88.92 ± 0.78	88.86 ± 0.79

TABLE 7.12: Error detection. Indicator performance (area under the ROC curve) for misclassification prediction.

	CIFAR 10		
	ResNet 50	WideResNet	DenseNet 121
ODIN	85.63 ± 2.92	83.38 ± 3.84	75.17 ± 1.77
T1000	88.99 ± 1.39	88.60 ± 0.35	85.70 ± 1.85
MP	93.13 ± 0.59	92.96 ± 0.39	92.58 ± 0.28
H	93.12 ± 0.59	92.88 ± 0.38	92.47 ± 0.27
NORM	74.79 ± 5.41	70.38 ± 7.18	50.37 ± 3.27
NORM+	81.55 ± 3.41	80.04 ± 3.75	66.54 ± 3.13
ACT	89.11 ± 1.33	88.63 ± 0.40	85.86 ± 1.80
ACT+	87.35 ± 1.99	84.90 ± 2.30	78.52 ± 2.92
PROJ	88.97 ± 1.37	88.74 ± 0.12	86.09 ± 1.80
ANG	90.73 ± 0.52	91.79 ± 0.25	92.03 ± 0.58
ANG++	90.48 ± 0.77	86.99 ± 0.73	86.96 ± 2.15
DMS	29.77 ± 4.79	37.70 ± 1.88	34.21 ± 1.89
DMS-AOS	31.05 ± 1.91	38.47 ± 1.51	33.11 ± 0.97
DSS	40.39 ± 0.34	41.14 ± 2.19	45.40 ± 1.12
DSS-EXT	53.46 ± 0.63	49.95 ± 1.84	56.76 ± 0.91
1C-SUM	89.42 ± 2.64	86.24 ± 2.97	85.02 ± 1.87
	CIFAR 100		
	ResNet 50	WideResNet	DenseNet 121
ODIN	76.86 ± 0.14	78.66 ± 1.13	78.55 ± 1.08
T1000	79.13 ± 0.32	79.78 ± 0.67	79.96 ± 0.27
MP	86.54 ± 0.47	86.47 ± 0.19	87.32 ± 0.35
H	86.19 ± 0.52	86.29 ± 0.17	86.87 ± 0.27
NORM	52.18 ± 0.72	62.44 ± 2.72	59.72 ± 3.18
NORM+	65.77 ± 0.37	70.33 ± 1.86	69.47 ± 1.64
ACT	79.18 ± 0.32	79.80 ± 0.65	80.02 ± 0.28
ACT+	74.06 ± 0.43	76.83 ± 1.17	76.94 ± 1.15
PROJ	80.56 ± 0.28	79.74 ± 0.56	80.80 ± 0.13
ANG	84.06 ± 0.35	81.45 ± 0.20	82.14 ± 1.21
ANG++	82.24 ± 0.29	76.84 ± 0.62	77.35 ± 0.60
DMS	38.76 ± 0.53	39.46 ± 0.64	39.69 ± 0.97
DMS-AOS	39.47 ± 0.61	41.90 ± 0.29	41.12 ± 0.75
DSS	43.63 ± 1.06	44.46 ± 0.34	41.59 ± 1.29
DSS-EXT	49.13 ± 0.27	49.57 ± 0.38	47.81 ± 0.39
1C-SUM	78.95 ± 1.03	79.51 ± 0.89	81.82 ± 0.62

TABLE 7.13: Average auroc score improvement when tackling the joint task of OOD and misclassification detection.

	CIFAR 10		
	ResNet 50	WideResNet	DenseNet 121
ODIN	1.30 ± 0.68	0.87 ± 0.51	0.60 ± 0.79
T1000	1.71 ± 0.93	1.22 ± 0.60	1.29 ± 0.51
MP	2.64 ± 0.60	2.22 ± 0.59	2.15 ± 0.43
H	2.47 ± 0.65	1.96 ± 0.66	1.99 ± 0.47
NORM	1.05 ± 1.20	0.21 ± 0.80	0.02 ± 0.79
NORM+	1.40 ± 1.16	0.65 ± 0.70	0.75 ± 0.71
ACT	1.71 ± 0.94	1.20 ± 0.61	1.29 ± 0.52
ACT+	1.33 ± 0.87	0.82 ± 0.69	0.99 ± 0.67
PROJ	1.64 ± 0.89	1.21 ± 0.62	1.29 ± 0.55
ANG	1.81 ± 0.47	1.83 ± 0.51	1.40 ± 0.56
ANG++	1.25 ± 0.92	1.63 ± 0.80	1.19 ± 0.92
DMS	-1.48 ± 1.16	-1.12 ± 1.05	-1.18 ± 1.03
DMS-AOS	-1.38 ± 1.22	-1.10 ± 1.09	-1.40 ± 1.00
DSS	-1.08 ± 1.02	-0.96 ± 1.02	-0.71 ± 0.96
DSS-EXT	-0.41 ± 0.91	-0.52 ± 0.83	-0.19 ± 0.88
1C-SUM	1.20 ± 0.93	0.73 ± 0.81	0.85 ± 0.87
	CIFAR 100		
	ResNet 50	WideResNet	DenseNet 121
ODIN	4.39 ± 2.00	4.63 ± 2.53	4.49 ± 2.22
T1000	5.06 ± 2.74	4.99 ± 2.97	4.88 ± 2.59
MP	8.01 ± 2.35	8.10 ± 3.16	8.42 ± 2.91
H	9.01 ± 2.29	6.73 ± 3.05	9.25 ± 2.62
NORM	-0.07 ± 2.62	2.19 ± 3.60	1.62 ± 2.94
NORM+	2.78 ± 2.84	3.47 ± 3.26	3.40 ± 3.23
ACT	5.06 ± 2.71	4.99 ± 2.95	4.90 ± 2.58
ACT+	3.21 ± 2.74	3.63 ± 3.37	3.85 ± 2.70
PROJ	5.67 ± 2.76	4.99 ± 2.99	5.31 ± 2.62
ANG	6.48 ± 2.87	5.75 ± 3.29	5.61 ± 2.86
ANG++	4.61 ± 3.53	4.26 ± 3.36	4.10 ± 3.08
DMS	-1.27 ± 4.83	-5.49 ± 4.35	-1.77 ± 4.21
DMS-AOS	-3.82 ± 3.53	-3.74 ± 3.87	-3.39 ± 3.44
DSS	2.56 ± 2.98	-4.27 ± 4.50	1.69 ± 2.48
DSS-EXT	-1.81 ± 3.18	-1.20 ± 2.80	-1.77 ± 3.37
1C-SUM	3.25 ± 3.56	3.37 ± 4.09	4.01 ± 3.45

TABLE 7.14: Average top ranking for the joint task of misclassification and OOD detection.

	CIFAR 10		
	ResNet 50	WideResNet	DenseNet 121
ODIN	6.83	7.00	9.67
T1000	6.83	5.33	8.33
MP	10.17	8.83	5.67
H	9.00	7.33	4.67
NORM	13.67	12.83	16.83
NORM+	12.17	10.17	14.17
ACT	5.83	4.00	7.33
ACT+	6.67	6.83	11.67
PROJ	6.00	4.00	6.33
ANG	6.17	6.83	2.17
ANG++	7.00	11.83	5.83
DMS	14.33	13.83	12.33
DMS-AOS	16.17	17.67	17.50
DSS	11.83	13.00	11.67
DSS-EXT	11.33	13.17	10.83
1C-SUM	2.67	3.83	3.83
	CIFAR 100		
	ResNet 50	WideResNet	DenseNet 121
ODIN	5.83	5.50	5.83
T1000	6.33	6.33	6.00
MP	6.67	8.17	7.00
H	9.17	6.17	9.33
NORM	13.67	14.83	13.67
NORM+	12.50	12.83	12.50
ACT	6.00	6.33	5.17
ACT+	7.83	7.00	8.33
PROJ	6.67	5.83	6.67
ANG	5.33	5.67	3.33
ANG++	3.00	8.83	8.00
DMS	17.00	13.83	16.83
DMS-AOS	14.67	17.33	14.33
DSS	18.67	11.33	18.50
DSS-EXT	10.33	13.33	9.50
1C-SUM	3.83	3.67	3.00

7.6 Summary indicators

The previous section concluded that there is no one-fits-all indicator. In this section, we attempt to remedy this by proposing a summary indicator and evaluating its performance against the other individual indicators.

Whereas combining indicators when ID and OOD data are available is as straightforward as learning a model, it is not an easy task in a sample-free setting. Accordingly, we propose a simple aggregation scheme that consists in summing (a subset of) the previously-introduced indicators. The simple intuition behind this sum is that it will allow benefiting both from the redundancy and complementarity of the individual indicators. We called this aggregation scheme the 1-class sum (1C-Sum).

Since all indicators are such that their values are low (resp. high) for ID (resp. OOD) samples, this is also the case of their sum. However, since indicators have very different distributions (see Table 7.4), directly summing them would give them largely uneven weights in the aggregated indicator. We thus propose to first rescale their distributions to comparable ranges by standardizing them. In principle, this requires estimating the mean and variance of these indicators on ID samples, which are unavailable. We propose instead to estimate statistics of the indicators on randomly generated data. Arguably, random data could lead to poor estimates of ID means and variances but will, hopefully, nevertheless allow rescaling the indicators to more comparable ranges.

As normalizing data, we chose uniform noise $\mathcal{U}(0,1)$, matching the network input size. Samples drawn from this distribution are then standardized according to the ID statistics, as usual. Let h_i, $i = 1, \dots, N$ be the collection of indicators, $\mu_i = \mathbb{E}_{x \sim \mathcal{U}}\{h_i(x)\}$ be the expectation of the ith indicator under the uniform distribution, and $\sigma_i^2 = \mathbb{V}_{x \sim \mathcal{U}}\{h_i(x)\}$ its variance. The summary indicator H is the following sum:

$$H(x) = \sum_{i=1}^{N} \frac{h_i(x) - \mu_i}{\sigma_i}. \tag{7.33}$$

We introduced in this sum all indicators, except the IN- indicators and ODIN. The former performed poorly on the Gaussian dataset and are thus expected to result in unsuitable standardization under uniform noise. ODIN was excluded to avoid its costly backward pass and keep the complexity of the 1C-sum as low as possible.

Empirical analysis: OOD detection. To validate 1C-Sum, we tested it in the same experimental conditions as for the other indicators (see Section 7.5 for more details). From Table 7.8, one can see that 1C-Sum performs extremely well, being almost always the second-best in terms of average ranking (after the supervised approach which is not realistic in our setting). On the few instances where it does not come second, it has a rank close to its challengers (ANG++, ODIN). The contrary cannot be said: ANG++ and ODIN can have far worse rank than 1C-Sum. This is because when 1C-Sum is beaten by an indicator, it is never by far. Overall, 1C-Sum is quite stable.

Empirical analysis: semantic and covariate closeness. On the statistically and semantically closest task (`h1/h2`) 1C-Sum suffers from including any batchnorm indicators at all (*i.e.* even the non-IN- ones). By removing them, the summary is able to reach performances much closer to the best indicator.

Interestingly, on the other partition 1C-Sum works best when including some of the batchnorm indicators (as usual). In that case, it is the best performing indicator (disregarding the supervised one).

Overall, 1C-Sum is a good candidate but care about its components should be taken when the OO-distribution is assumed to be very close.

Empirical analysis: model quality. Going back to Table 7.11, we see that by combining indicators, it is possible to achieve good performances with a less well-trained model (85% of training accuracy, a stage more on Tiny ImageNet). At that stage, 1C-Sum is already performing better than most other (sample-free) indicators at their peak. Unlike the supervised method, this approach suffers somewhat from overconfidence.

Empirical analysis: misclassification detection. It appears that 1C-Sum performs adequately for detecting misclassification (Table 7.12) , although simpler indicators work better. This is unsurprising since some of the incorporated indicators perform less well on that task.

Interestingly, 1C-Sum benefits slightly from the joint task of detecting OOD samples and misclassified ones, even though it incorporates batchnorm indicators (Table 7.13.

Overall 1C-Sum remains the best bet to tackle OOD detection in a sample-free setting (Table 7.8, possibly jointly with misclassification rejection (Table 7.14).

7.7 Real-world setting

The empirical analysis highlighted several indicators as adequate, in the sense that they provide a thresholdable quantity capable of separating well ID and OOD samples. The analysis was conducted through the lens of the area under the ROC curve (auroc), a threshold-agnostic metric. In practice, however, a cut point for the indicator must be chosen in order to automatically reject samples. Although some indicators are more interpretable than others, it remains challenging to set a threshold in a sample-free, and also architecture-independent, fashion (See Table 7.4 and box-plots in Appendix B.3).

The approach we advocate is to collect a few samples while the model is deployed in real conditions to adapt the threshold (and possibly fine-tune the weights of 1C-Sum). We sketch below a few of such solutions.

Test samples can be labeled. If some (human) effort can be dedicated to labeling observed samples as ID or OOD, setting a threshold is straightforward. Because of the univariate nature of our indicators, we expect that only a few samples would be needed to converge to a stable threshold, although it depends on the expected proportion of OOD samples. Obviously, if many labeled samples become available, the problem will stop being sample-free and one could consider supervised approaches. Our experiments show that excellent results can be reached by fitting a simple linear model on all our indicators.

No labeling is possible. Addressing the problem of setting a threshold fully automatically and without any labeling is only possible in our opinion if some assumptions can be made on the OOD data. Let us consider a few examples.

First, if a good guess could be made regarding the expected proportion of observed OOD samples, one could simply set the threshold so as to isolate that proportion of samples in the stream of data.

Second, if the OO-distribution is stable and far away from the $\mathcal{I}$-distribution in the indicator space, it is possible to isolate both parts of the mix distribution by minimizing the intra-variance along the indicator in an unsupervised way. More generally, if the number of statistical modes is known for the indicator space, any clustering algorithm could isolate the ID samples.

Third, if we assume knowledge of what the OO-distribution should be but cannot access it, we can turn to proxy data. Provided the data is well chosen to lie between the in- and true out-of-distributions, isolating the proxy data would also isolate the OOD samples.

Fourth, zero-shot samplers (see the discussion about the sample-free setting in Section 7.2.4) allow sampling data, hopefully close enough to ID samples, to set the threshold(s).

Remember that identifying the ID cluster is made easy in all these scenarios because the indicators have been designed to take low values for ID samples. It is especially straightforward with 1C-Sum, where a single threshold must be set.

7.8 Conclusion

In this chapter we discussed the problem of out-of-distribution (OOD) detection (by opposition to "in-distribution" (ID) samples; samples drawn from the training $\mathcal{I}$-distribution): what careful considerations went into its formulation (risk balancing, defining OODness and how OOD data could relate to the $\mathcal{I}$-distribution), how it relates to other well studied problems and how it can, and has been tackled, depending on the data available (Section 7.2). We then proposed the sample-free setting resulting in the question of how could OOD samples be detected if we only have the network available (Section 7.3)?

Several indicators were proposed, either based on the optimality conditions (Section 7.4.1) or on the batch-normalization layers (Section 7.4.2). A last one, 1C-Sum, was proposed as a summary indicator by aggregating the other indicators (Section 7.6). Aggregating is a simple matter when data is available but a challenge in our setting. Indicators are crafted to display low value for ID samples and hopefully high value for OOD samples, offering simple thresholdable quantities to distinguish between ID and OOD data.

The indicators were then evaluated (Section 7.5) on a variety of OOD tasks, ranging from easy ones (due to the large statistical differences) to much harder ones (very close statistically). The indicators perform well. In particular, they cover three cases. Some batchnorm indicators are efficient at detecting gross channel-wise statistical differences, while others are good at filtering out noise. On harder tasks, optimality-based indicators were found to be more appropriate. The summary indicator is a good default choice and can be further fine-tuned if data become available. When there is more supervision, the indicators can be combined to offer near-perfect results.

Further investigation yielded insightful results. Firstly, dimensionality reduction allowed recovering the three groups of indicators discussed in the previous paragraph (Section 7.5.3.1). Secondly, Section 7.5.3.2 highlighted that the optimality-based indicators both suffer from a low accuracy model—unsurprisingly—as well as overconfidence. Contrary to expectations, the batch-normalization indicators are also impacted by the model quality, although the pattern is erratic. 1C-Sum is more robust to the model quality and supervision can compensate for the lack of quality. Finally, we briefly looked at the problem of detecting classification errors with the indicators (Section 7.5.3.3). Interestingly, the best-performing indicators for this task are not necessarily the same as for OOD detection, suggesting different patterns in the latent space for OOD and misclassified samples. Without surprise, batch-normalization indicators are useless for this. As a consequence of all these, when tackling the joint task of misclassification and OOD detection (the most practical setting?), some indicators come out better than others compared to tackling the sole task of OOD detection. On a well-performing network, misclassification should play a relatively negligible role, however. Once more, the best sample-free option is to turn to 1C-Sum.

Finally, we proposed several ways to use these indicators in practical settings, depending on the information that can be gathered when the model is deployed and the assumptions that can be made about the nature of the OOD data (Section 7.7). Hopefully, the simplicity of the indicators should render that phase efficient data-wise.

This contribution raises new questions. First of all, there is the question of whether it is possible to craft sample-free indicators relying on yet another paradigm besides optimality conditions and batch-normalization layers.

Secondly, Figures 7.9 and 7.10 portraying the behavior of several indicators in the pre-logit space spark a few questions. Is it possible, in a sample-free setting, to take a good guess at where the modes lie? This would offer more and better ways to detect OOD samples. Alternatively, is it possible to design a simpler summary indicator? It feels that combining ANG and NORM should yield interesting results as it both tells whether the samples align well with a hyper-plane and whether it is close to the center or not. Arguably this is what PROJ and T1000 are doing but how both are integrated does not lead to astounding results.

Thirdly, 1C-Sum being computed thanks to generated noise, we had to drop the IN- indicators. Would another proxy set, for instance one which could be assumed to be closer to the $\mathcal{I}$-distribution, enable the use of those features as well?

Finally, all the experiments were conducted with a class-balanced base task. It would be interesting to see how OOD detection performs in imbalanced settings. An imbalanced base task might (i) challenge our assumption about the optimality conditions and (ii) result in some classes being badly predicted. How this asymmetry would impact detection is yet another open question.

Chapter

8 Distillation from heterogeneous unlabeled collections

Chapter overview

This chapter tackles the compression of deep networks towards a smaller, given architecture without training data in the context of image classification. To overcome this limitation, we rely on a large collection of unlabeled data, assumed to contain some samples relevant for the transfer. Our solution consists in biasing sampling towards seemingly relevant samples while maximizing the transfer with some attention mechanism.

This chapter is based on our paper "Distillation from heterogeneous unlabeled collections" currently under review.

The code relating to this contribution is available as a Python package at `https://github.com/jm-begon/distill_by_transfer`. It is implemented on top of PyTorch (Paszke et al., 2017).

This chapter is organized in six sections, the first of which (Section 8.1) exposes our ambitions: the goal pursued, our contribution and the motivation behind our work. Section 8.2 discusses the problem of coming up with small networks in general, while Section 8.3 focuses on the data-free compression. Our solution is detailed in Section 8.4 and evaluated in Section 8.5. Section 8.6 concludes.

The present chapter is faithful to the original article. Sections 8.1.2 to 8.2.3 have been added to provide more context and notations have been uniformized.

8.1 Ambitions

8.1.1 Goal and contribution

This work consists in tackling the task of compressing a large neural network into a smaller one without the original dataset. More precisely, we suppose the availability of (i) a teacher network which has been learned and performs well on a target task, and (ii) an unlabeled collection containing "relevant" data, which is not expected to be or to contain the original data used to train the teacher.

Solutions which fit our setting have been proposed (see Section 8.3) but come with a large additional computation cost compared to what would be required to learn the small network directly, were the data available. We would like to leverage the unlabeled collection of data to propose a (much) faster alternative.

We restrain our investigation to image classification where the availability of unlabeled samples might not consist a hindrance.

Contribution. In light of these, our contributions can be summarized as follows:

- we propose a method to focus on relevant samples from a collection of unlabeled data;
- we propose a fast solution to tackle distillation (transferring the knowledge from a trained model to a new one) when original data is missing but such a collection is available;
- we conduct an extensive empirical study of the proposed solution and show how and when it is called for.

8.1.2 Motivation

If one thing can be said regarding the last decade in the world of machine learning, it is that we have witnessed a massive enthusiasm for what has come to be known as deep learning. After struggling for decades, neural networks have achieved a quick succession of astounding progress. The cause behind this revolution is many-fold but the success has attracted more researchers which has led to more developments that have drawn more attention and so on, up to the point where dedicated hardware and libraries to use them seamlessly have been developed. As hinted in Section 5.2.2, these have sparked the hope of leveraging the astounding performances of machine learning in constrained settings, thus democratizing the use of such models and offering a world of endless possibilities on end-devices, far from huge computing servers and free of communication costs (and the associated privacy issues). This is even more true of deep learning which has established itself as a landmark in areas such as computer vision, speech recognition, translation and so on—areas which are deemed important for artificial general intelligence and can benefit from being used at a much wider scale.

Similarly to decision forests (Chapter 6), smaller models come with a handful of advantages: fast inference, low energy consumption, more durability, and implicit regularization—all beneficial for embedded situations. Interpretability is less clear in this case compared to decision forests. Fast inference, on the other hand, is more crucial than ever. If training network models without GPU is hard to imagine nowadays, inference might get away without, provided the network is fast enough. Therefore fast inference is accompanied by lower costs due to more standard, cheaper equipment.

Note that compression is employed here in the broad sense of ending up with "small" models, however it is achieved, thus encompassing a myriad of methods spanning many paradigms (further discussed in Section 8.2.3).

The sample-free setting. As Section 5.2.4 elaborates, the learning set might not make it past the training stage. Therefore, no data might be leverageable when developing small models is a post-training concern. There are a number of reasons why compression could be an afterthought: developing a new product, moving on to cheaper equipment, massively deploying a solution, targeting a new share of customers, and so on.

Since this chapter is concerned with two constraints (model size and the lack of data), Section 8.2 will first delve into the general problem of compressing deep networks. We will come back to the sample-free setting in Section 8.3.

8.2 Deep learning compression

This section discusses the problem of coming up with small neural network models in general terms. Section 8.2.1 discusses why the problem is feasible in the first place. Section 8.2.2 illustrates how the problem can be formulated with some attention to the metrics used when discussing the size (or speed) of neural network models. Section 8.2.3 then proceeds onto categorizing the methods of compression: designing small architectures (Section 8.2.3.1), pruning (Section 8.2.3.2), low-rank approximation (Section 8.2.3.3), quantization (Section 8.2.3.4), and distillation (Section 8.2.3.5).

8.2.1 Feasibility of neural network compression

Is compression even feasible for neural networks?

Neural networks (with at least one hidden layer) are reputed to be universal approximators (*e.g.* Hornik, Stinchcombe, and White, 1989; Zhou, 2018; Heinecke, Ho, and Hwang, 2020). This is not to say that *any* architecture can approximate any function with an arbitrary level of precision. The results are established for networks with unbounded capacity, such as being able to increase the "width" of the feature extraction layer(s) to match the wanted precision. Therefore, it cannot be expected that a network can be compressed without impacting the accuracy, as the size of the network is an important parameter: too small and the Bayes model might not figure in the hypothesis space (representation bias), too big and the risk of overfitting is high. The necessity to adapt the architecture to both the problem and the available data has long been an inherent hindrance with neural networks—until recently.

The width-depth dilemma. Half a decade ago the neural network community was divided on the question of whether neural networks should go wide or deep. The fact that the domain has been re-baptized "deep learning" gives away the outcome but widists had solid argument: arbitrary large networks could be universal approximators and deep models were hard to train.

Two papers from that era best exemplified the dilemma. With a telltale article entitled "Do Deep Nets Really Need to be Deep?", Ba and Caruana (2014) looked at how wide-and-shallow networks could approximate deeper[1] ones. In response, Romero et al. (2015) published a paper doing the exact opposite: learn to approximate a network with a thin-and-deep one. Interestingly, they had to introduce several mechanisms to help the learning, the most prominent being the hints, a form of attention mechanism (see Section 8.4.3 for more on attention mechanism), thus illustrating the harder training faced by those networks.

In the end, the match was won by the depthists thanks to neural network portraying a form of combinatorics with depth (Poole et al., 2016; Raghu et al., 2017). As Raghu et al. (2017) summed it up "the complexity of the computed function grows exponentially with depth", where complexity relates to the number of piece-wise linear regions in the output space corresponding to the input manifold. With depth come easier—and more economical—combinations of patterns. In the case of convolutional neural networks (CNN), depth also allows enforcing spatial invariance (patterns detected anywhere in an image will end up being processed at a later stage) and multi-resolution analysis (thanks to there being more pooling along the way).

The advent of deep learning. When rooting for deeper rather than wider, the community has faced the challenge of training these architectures. As discussed in Section 3.6.2.2, one of the major problems to overcome was to enable the learning signal to back-propagate in the network. A challenge successfully tackled by the community using a combination of techniques (unbounded activation functions, skip connections, etc.)

With networks of high capacity, now trainable, the remaining issue was to avoid overfitting. Several factors have contributed to reducing this risk: better initialization (*e.g.* Glorot and Bengio, 2010), better regularization, such as dropout (Srivastava et al., 2014), possibly better optimizers (*e.g.* Duchi, Hazan, and Singer, 2011; Kingma and Ba, 2015) and more appropriate inductive bias. Another major contributor has been the availability of very large datasets and the interesting fact that a significant part of learning is transferable. For instance, it is possible to learn the feature extractor in a domain where data is plentiful and use it (possibly with some fine-tuning) in others (Yosinski et al., 2014; Donahue et al., 2014; Mormont, Geurts, and Marée, 2018). A fact rendered even more useful by the availability of standard networks trained on nowadays's large datasets.

Armed with the capability of learning very high-capacity networks with moderate risks of overfitting, the current practice is to rely on over-sized hypothesis spaces rather than incur the risk of missing the Bayes model by a large margin.

[1]"Deep" must be understood in the context of the epoch; as of today's standard, the studied networks would be termed deep very timidly.

The era of over-parametrization. Having over-parametrized architectures to train while overfitting can be kept at bay has several advantages. Firstly, those architectures portray better learning capabilities (Du et al., 2019; Allen-Zhu, Li, and Song, 2019; Sankararaman et al., 2020). According to Allen-Zhu, Li, and Song (2019) “with the help of over-parameterization, near the GD/SGD training trajectory, there is no local minima and the objective is semi-smooth”. In essence, over-parametrization ensures that there is always a direction to lower the error barring at the global minimum.

The amount of over-parametrization is such that Denil et al. (2013) showed it was possible to recover up to 95% of the network weights without loss of accuracy.

On the other hand, Frankle and Carbin (2019) argue that well-performing models are in part due to the presence of fortunately-initialized sub-networks. In some sense, the more parameters, the more likely it is to find sub-parts which perform well on their own.

According to Arora et al. (2018) the layers in a trained network are quite resilient to noise compared to a randomly initialized one. More precisely, they observe the noise is quickly amortized while being forwarded in the network. They suggest this might be an explanation as to why the networks, even though potentially capable of memorizing the training set due to being over-parametrized, generalize well nonetheless. The network ends up being much smoother than it might have been, never falling near the worst-case scenario the theoretical bounds address. This smoothness is a regularity which can be exploited to compress a neural network.

Designing over-parametrized architectures apparently come with many benefits in addition to the insurance that the network has sufficient capacity to learn the problem. It also means there is a margin for lossless compression in (current) deep learning. Reaching it might be difficult or computationally costly, however, as it would mean doing without the training benefits of over-parametrization. Once past this lossless compression threshold, a realm of lossy compression awaits, with the challenging question of how to best compress a model to limit the accuracy loss the most.

8.2.2 Problem formulation

As with decision forests (Section 6.2.2), how much accuracy can be sacrificed is problem-dependent and methods should provide a way to select the appropriate accuracy/compression tradeoff. At a high level, this can be formulated in three ways:

- select the smallest network meeting a minimal accuracy level;
- select the best network not exceeding a given overall size;
- select the network which achieves the best size-accuracy tradeoff.

Note that “select the network” can mean (i) choose the architecture, (ii) train the model, or (iii) both, in this context. For instance, when designing small networks (Section 8.2.3.1), the architecture is fixed prior to training and

TABLE 8.1: Model "size" measures. A reference value is provided for a ResNet 50 (He et al., 2016) network from PyTorch (Paszke et al., 2017) expecting 224×224 RGB images as input and 1000 outputs. No measure is provided for latency as it does not depend solely on the implementation of the model.

MEMORY	TIME
ON-DISK FOOTPRINT ($\approx 90Mb$)	LATENCY (*i.e.* INFERENCE TIME)
NUMBER OF PARAMETERS ($\approx$ 24 MILLIONS)	NUMBER OF FLOPS ($\approx 4 \times 10^9$)

only the parameters are learned. On the other hand, pruning methods (Section 8.2.3.2) start with a larger architecture than they end with; the architecture is thus partially learned. Providing time permits, the architecture can nonetheless be optimized by a traditional model selection method.

Whatever the formulation, models must be comparable in terms of accuracy (which is straightforward) and in terms of size, which is the topic of the upcoming section.

8.2.2.1 Model size: a note on measures

Depending on the motivation behind the compression, several metrics can be used to measure how small a model is.

Memory measures. If compression is done for the sake of memory, the most straightforward measure is the on-disk memory. Table 8.1 (row 1, left column) indicates how much memory a given network is worth. As with decision forests (see Section 6.2.2), this metric is dependent on how the model is stored, although admittedly to a lesser extent since the most part of this footprint is due to the learnable parameters for which there are relatively few choices in terms of representation.

Nonetheless, it is usually beneficial to abstract away the representation. The space complexity of a model is linearly dependent on the number of parameters, as well as the number of layers of each type. Since the former usually makes up most of the footprint, it is common to only report the number of parameters (Table 8.1 row 2, left column). Focusing solely on the number of parameters has the advantage to facilitate the comparison between architectures.

Runtime and downtime memory. All the measures in Table 8.1 (left part) relate to the memory footprint of the network. However, as the model makes an inference, the memory for the intermediate tensors (or the largest one) must be accounted for. For instance, the first convolutional layer of a ResNet 50 expecting 224×224 RGB images will produce a tensor of size $64 \times 112 \times 112 \approx 803k$; much less than the number of parameters but still not insignificant.

Good memory management is necessary to keep the additional memory for inference as low as possible.

Time measures. The most straightforward measure for inference cost is inference duration, also known as latency. The inference duration being much more dependent on the hardware than memory (*e.g.* can everything fit in the RAM or will there be many memory loading operations? Is there a GPU? How are convolutions implemented?), one might prefer a hardware-agnostic measure—or not. It actually depends on the part the hardware plays. If the hardware is pre-selected and the goal is to compress a network to go below a certain inference time, it makes little sense to turn to metrics independent of the device. Think of real-time inference, for instance. On the other hand, such a focus ties any analysis to a given hardware which limits the generality of the related discussions.

The time complexity can be measured in a hardware-agnostic way by counting the number of floating-point operations (FLOPs[2]). Table 8.1 (row 2, right part) illustrates the number of FLOPs for a ResNet 50.

Conventional wisdom dictates that the actual running time should correlate well with the number of FLOPs. This is certainly true of older architectures in unconstrained conditions (Canziani, Paszke, and Culurciello, 2016). When looking at more modern and varied architectures of comparable numbers of FLOPs, it is not rare to observe a significant gap in running times. Ma et al. (2018) explain this by two factors: the memory access cost (some layers put more stress on the memory than others) and the amount of parallelism.

Whatever the measure, how it is used depends on the compression method.

8.2.3 Method overview

The goal of the present section is to briefly brush the available methods for compressing (in a broad sense) neural networks to get a feeling of how it can be achieved. This is by no means a complete tour of the literature on the topic and more specific reviews will be pointed out when available. Works more related to our setting will be addressed in Section 8.3.

8.2.3.1 Designing small architectures

A first alternative to end up with a small network is to just *start* with a small architecture and call it a day. Over the years, many strategies have been investigated to design layers with fewer parameters and overall architectures whose number of parameters does not explode with depth. Exploiting assumptions about the data being processed has allowed compensating the diminution of representational power (or the lessening of overparametrization) by providing better inductive biases; no need to search a giant haystack when the needle can be found in a small one. Corollarily, progress on small-yet-good architecture is limited to the type of data. We will investigate those relating to images.

[2]In FLOPs the "s" indicates the plural; it must not be confused with FLOPS which stands for floating-point operation per second, a measure of hardware speed.

Convolution. Convolution layers can only compute a fraction of what fully-connected layers can. The kernel size, stride and padding sparsify the extent of the connectivity between adjacent layers, whereas the kernel imposes a form of weight sharing. These two factors combine to offer a great reduction on what a fully-connected layer would have in terms of parameters to represent the exact same operation.

To be more quantitative, assuming a padding of $(k-1)/2$ and a stride of 1, a convolution layer with kernels of $k \times k$ transforming input of size $C_i \times H \times W$ into C_o features maps (consequently of spatial dimension $H \times W$) requires $C_o \times (C_i \times k \times k)$ parameters. Since the output if of total size $C_o \times H \times W$, a fully-connected layer would require $(C_i \times H \times W) \times (C_o \times H \times W)$ parameters the compression ratio is

$$\frac{(C_i \times H \times W) \times (C_o \times H \times W)}{C_o \times (C_i \times k \times k)} = \frac{H^2 \times W^2}{k \times k} \tag{8.1}$$

There is no gain with respect to the size of the intermediate tensor, however. In terms of operation count, the convolution amount to $O(C_o \times (H \times W) \times (C_i \times k \times k))$ FLOPs (for each output channel and at each location, the sum-product of the kernel must be computed).

Spatial reduction. Spatial reduction, whether with strided convolution or pooling, ensures the spatial dimensions of the tensors propagating down the network are smaller (typically divided by two along each spatial dimension). The effect varies with the type of subsequent layers. For instance, this does not affect the number of parameters needed for a convolution all things being equal (although the kernel might be chosen according to the spatial dimension of the feature maps, in which case it would impact the number of parameters). It will however reduce the size of the intermediate tensors and consequently the computing time.

Less fully-connected layers. Pre-modern architectures for image processing rely on several fully-connected layers after the convolutional/pooling part. For instance, the VGG16 network (Simonyan and Zisserman, 2015) used three fully-connected layers: two feature extractors outputting vectors of size 4096 and the softmax classifier. In such architectures, most of the parameters are located in those layers. For instance, roughly 90% of VGG parameters are linked to the fully-connected layers.

Even though fully-connected layers are costly memory-wise, it should be noted that most part of the computation time is spent in convolutional layers (Yang et al., 2015). What to reduce is thus dependent on what is sought after with compression.

Blocks and architectures. Besides the previous considerations, modern design practices focus on proposing groups of layers, aka. blocks, which factorizes what more traditional layers do. Architectures are then designed by composing such blocks. The following paragraphs review some of these blocks.

Depthwise separable convolution. Depthwise separable convolutions have been introduced by Sifre and Mallat (2014). The idea is to factorize a traditional 2D convolution as a sequence of two convolutions. The first convolution uses C_m kernels of dimension $1 \times k \times k$. This so-called depthwise convolution aims at processing the spatial information channel by channel. The second convolution uses C_o kernels of total size $C_m \times 1 \times 1$. It is referred to as a pointwise convolution and processes the channel information at each location.

Assuming input tensors of size $C_i \times H \times W$, a stride of 1 and paddings so that the output tensor has the same spatial size, the compression ratio compared to a full convolution is

$$\frac{C_o \times (C_i \times k \times k)}{(C_m \times (1 \times k \times k)) + (C_o(C_m \times 1 \times 1)} = \frac{C_i}{C_m} \frac{C_o \times (k \times k)}{C_o + (k \times k)} \tag{8.2}$$

Since the goal of the depthwise convolution is to process only the spatial information, $C_m = C_i$ is a natural choice: each input channel is only processed once.

The depthwise separable convolution is also less costly than traditional convolution in terms of operations: $O(C_o \times (H \times W) + C_m \times (H \times W) \times (k \times k))$ FLOPs.

Depthwise separable convolution is a block implemented with two layers to achieve the same goal as a single 2D convolution layer. With the presence of such blocks, comparing the number of layers in modern architectures might make little sense.

Group convolution. Group convolution (Krizhevsky, Sutskever, and Hinton, 2012; Xie et al., 2017; Zhang et al., 2017) factorizes a 2D convolution in a different way. Rather than separating the spatial and channel-wise convolution, the channels are divided into G groups of C_i/G channels and traditional 2D convolutions operate at the level of groups (with $G = 1$ implying the vanilla 2D convolution). Since the convolutions process fewer channels, the kernels are smaller in the channel dimension. The overall gain is

$$\frac{C_o \times (C_i \times k \times k)}{G \times \left(\frac{C_o}{G} \times \left(\frac{C_i}{G} \times (k \times k)\right)\right)} = G \tag{8.3}$$

where the denominator expresses that there are G groups, producing each C_o/G channels (so that the sum is C_o) by convolving kernels of size $C_i/G \times k \times k$.

The number of operations is also reduced: $O(C_o \times (H \times W) \times (C_i/G \times k \times k))$ FLOPs.

Zhang et al. (2017) note that grouping the channels limits drastically the representation power of the group convolution layer; concatenating such blocks is akin to having parallel pathways within the network operating on less information. To overcome this limitation, they propose the simple fix of shuffling the layers between group-convolution blocks in their ShuffleNet architecture.

Separable convolution with linear bottleneck. Sandler et al. (2018) proposed to use a pointwise convolution (followed by a non-linear activation function) which is fed into a depthwise convolution (also followed by a non-linear activation function) which in turn is fed into a pointwise convolution, without non-linear activation. The ordering of the layers and the linear ending allow accumulating the final tensor without explicitly computing the whole intermediate tensors. This is used to generate a large number of intermediate channels without increasing the runtime memory.

Architecture design versus architecture search. The blocks we have discussed in the previous paragraphs can be used as building material, along with others, to design lightweight and fast architecture, hopefully without decreasing the representation power too much. For instance, depthwise separable convolution are used in MobileNet (Howard et al., 2017). Deeper ResNet networks also need such convolution to prevent the number of parameters from exploding. As mentioned ShuffleNet (Zhang et al., 2017) and ShuffleNet V2 (Ma et al., 2018) use group convolutions. MobileNet V2 (Sandler et al., 2018) uses separable convolutions with linear bottlenecks.

It should be noted that a (computationally expensive) alternative to designing architectures from scratch is to learn them. For instance, NasNets (Zoph et al., 2018) are architectures which have been found by formulating the architecture search as a reinforcement learning problem.

8.2.3.2 Pruning

Whereas the previous section exposed solutions which require to design-and-learn a network, this section will focus on what can be done once the architecture is given as part of the problem. Two considerations are important to organize the works on pruning.

As with decision forests (Chapter 6), pruning can either be done on an already-trained network (post-pruning) or can be embedded into the learning procedure (pre-pruning). A fine-tuning step is usually required even with post-pruning methods.

The second consideration relates to what is actually pruned in the network. Mao et al. (2017) propose four levels of granularity in CNNs. The fine-grained level corresponds to removing any parameters of the convolution kernels. The vector level corresponds to removing parameters along the spatial dimensions of the kernel. The kernel level corresponds to removing input channels, while the filter level corresponds to removing output channels. Note that what is meant by removing an input channel is that the channel will not be taken into account when computing one of the output channels.

In comparison, when removing an output channel, it is not available for any computation of the subsequent layer.

Most of the works focus on either the fine-grained or the filter levels and will be referred to as unstructured and structured pruning, respectively. This dichotomy also translates to fully-connected networks where unstructured refers to pruning the connections while structured pruning targets the neurons (and consequently all the incoming and outgoing connections). Nothing prevents a method from working at both levels, though.

The distinction between structured and unstructured is important on two accounts. On the one hand, the more unstructured the pruning, the more flexible the methods, a fact well reported by Mao et al. (2017) (See Figure 8.1). On the other hand, this kind of sparsity might not be quite amenable to actually leveraging memory/speed gains on standard hardware. Indeed, removing all but one connection to a hidden neuron prevents removing that neuron altogether and masks (*i.e.* forcing connection to have a null weight) can only serve to showcase a theoretical memory gain. In comparison, structure pruning forces the removal of a whole channel (in CNN) or neuron (in traditional networks) and by-pass the problem of having a very sparse yet not truly actionable architecture.

Note that contrary to unstructured pruning, the choice of which component to remove (*i.e.* a whole convolution filter, a whole neuron) is not time/memory-independent: removing a $3 \times 7 \times 7$ convolution kernel is different from removing a $512 \times 3 \times 3$ kernel. These methods have thus the flexibility of choosing how to balance the accuracy drop with the memory/time gain. When the purpose of compression is to reduce the memory footprint, weighting by the number of parameters offers a straightforward and compelling solution. When the aim relates to inference time, the metrics to use is less clear (as discussed in Section 8.2.2.1). Ideally, real-time measurements should be used but this is costly to assess due to the combinatory nature of the problem. A possible workaround, proposed by Yang et al. (2018), consists in using lookup tables of real-time measurement per blocks and assume that the total inference time is simply the sum of the time for all blocks. Moreover, the measurement for similar blocks can be assessed once, thus reducing the overall complexity of the problem.

Since the pre- and post-pruning dichotomy influences the problem formulation the most, this criterion will be used to articulate a selection of proposed methods.

Pre-pruning methods. Pre-pruning methods usually start with a given architecture. Training is then formulated as both learning the weights of the architecture and sparsifying it at the same time.

Han et al. (2015) proposed to apply regularization on the connections and remove all below a certain threshold; an example of unstructured pruning. Their approach consists of two learning phases: one with regularization to prune the structure and a re-training without regularization phase to learn

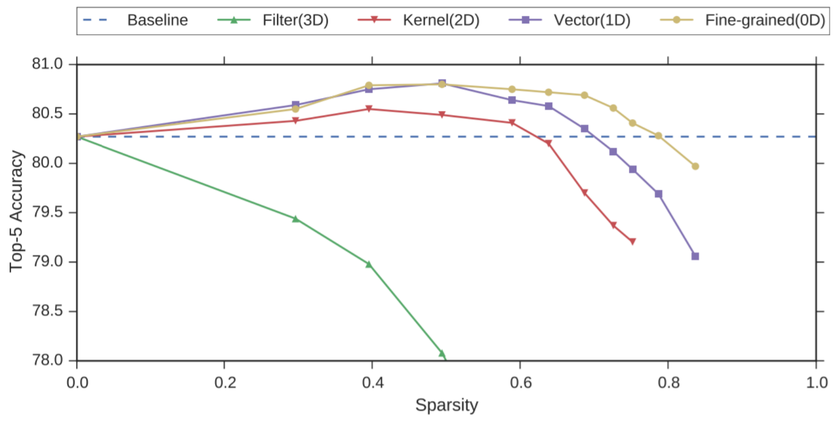


FIGURE 8.1: Accuracy/compression tradeoffs: the case of AlexNet (Krizhevsky, Sutskever, and Hinton, 2012) on the ImageNet (Deng et al., 2009). The more flexible the method (*i.e.* the more fine-grained the pruning) the better it is at retaining accuracy for a given compression level—at least with magnitude group-weight pruning. From Mao et al. (2017).

the final parameters of the network. They found that although a $L1$ regularization worked best to induce the sparsity without re-training, $L2$ regularization coupled better with the re-training phase.

Wen et al. (2016) proposed to apply group lasso to the parameters of CNNs. Group lasso consists in applying a sparsity-inducing regularization to groups of parameters. They review several grouping strategies: channel-wise (to reduce the number of input channels) filter-wise (to reduce the number of output channels), as well as groups resulting in unstructured pruning.

Alvarez and Salzmann (2016) and Scardapane et al. (2017) both proposed to apply sparse group lasso, which consists in enforcing sparsity both at a group level (*e.g.* all the weights for a given neuron) as well as on individual parameters, resulting in a mix of structured and unstructured pruning.

Liu et al. (2017) proposed to enforce an $L1$ penalty on the scaling factor of the linear output map of batch-normalization layers (the γ parameters in Equation 3.121). Since each parameter relates to a channel, this constitutes a form of structured pruning.

Yang et al. (2019) proposed to associate masks with all layers of the networks, working as elementwise gates for each parameter. The masks are initialized to work as if they were absent but a $L1$ regularization is applied to them during training. Neurons or filters whose mask averages are below a given threshold are removed. The remaining masks are merged into the network, resulting in a mix of structured and unstructured pruning.

Post-pruning methods. Post-pruning methods are based on the premise that an already-trained network is available. They usually rely on a saliency measure which indicates how much a (group of) parameter(s) is important. Parameters with low saliency are sacrificeable. To attain good accuracy with high compression rate, post-pruning methods usually operate on a greedily-iterative scheme: some parameters are removed, then the network is re-trained to update the saliencies before removing more parameters, and so on.

LeCun, Denker, and Solla (1989) proposed to mark for removal the weights which influence the objective function the least. To do so, they applied the Taylor decomposition of the loss function, using some approximating assumptions to reduce the computation burden. They suggest updating the measure every so often and fine-tuning the obtained networks.

Hassibi, Stork, and Wolff (1993) proposed to avoid the simplifying assumptions of the previous method so that the saliency measure better reflects the effect of removing a node. Rather than retraining the network to update the saliencies, they proposed to use the Hessian matrix (needed anyway) to directly update the objective. They still advocated for a final fine-tuning. It should be noted that there is a fast algorithm to compute the Hessian matrix when it is involved in a Hessian-vector product, as is often the case (Pearlmutter, 1994).

Molchanov et al. (2017) revisited the idea behind using the Taylor expansion as saliency measure for (more) modern architectures with slightly different approximations. Theis et al. (2018) formulated a similar criterion for structured pruning.

Zhu and Gupta (2018) proposed to use the weight magnitudes as saliency and introduced gradual pruning, where the sparsity is only fully enforced after several steps to let the network adapt to it during training. This work somewhat sits on the pre-/post-pruning fence.

Yu et al. (2018) proposed a saliency measure which captures how each neuron impacts the pre-linear latent space. For internal neurons, the score is computed by aggregating over the paths from that neuron to the last layer.

Ironically, Mittal et al. (2018) showed that good performances can be obtained by pruning the filters randomly and retraining the network. The saliency measures seem less important than might have been anticipated.

Frankle and Carbin (2019) achieved up to a compression factor of 10 with a simple scheme: iteratively (re-)training the network and filtering out a small percentage of the parameters. They use the weight magnitude as saliency measure. The core contribution of their work was to notice that re-training should start with the original weights of the network, thus ensuring to isolate a good sub-network; one which happens to be well initialized for the problem—the so-called lottery ticket.

The interested reader can consult the work of Blalock et al. (2020) for a more complete review on pruning.

8.2.3.3 Low-rank approximation

Most neural networks include linear maps (fully-connected layers, convolutions). The presence of linear components begs the question of whether it is possible to approximate them as a product of low-rank matrices. For instance, if a $p_l \times p_{l+1}$ W_l matrix can be decomposed $U_l V_l \approx W_l$, where U_l is

$p_l \times k$ and V_l is $k \times p_{l+1}$ the compression gain is

$$\frac{p_l \times p_{l+1}}{p_l \times k + k \times p_{l+1}} \tag{8.4}$$

Low-rank approximation is a hybrid case. When the decomposition happens during the design, they fall into the design-small-and-learn approach. Indeed, there is no need *a priori* to learn the map and then factor it when the factorization can directly be learned. In essence that is how convolutions, separable convolutions, group convolutions, and so on operate.

On the other hand, when the low-rank approximation is actually computed from a trained network, the whole process is more akin to structured pruning. Jaderberg, Vedaldi, and Zisserman (2014) addressed both cases without declaring a clear winner.

Low-rank as design. Sainath et al. (2013) proposed to factorize the largest fully-connected layers of an architecture as a $U_l V_l \approx W_l$ product and discussed the rank/accuracy compromise.

Osawa et al. (2017) proposed to flatten convolution layers into 2D matrices, decomposed along an SVD scheme and optimize the order of the multiplications.

Liu et al. (2015) proposed to factorize convolutions as convolutions of smaller kernels and imposed a sparsity-inducing regularization on the kernel parameters, the equivalent of a mix of structured and unstructured pruning.

Learning low-rank approximation. Denton et al. (2014) reviewed several factorizations based on SVD and applicable for CNNs. They advise fine-tuning the network if the factorization results in a very low rank.

Swaminathan et al. (2020) proposed to couple a truncated SVD decomposition with pruning to further sparsify the factorization. They explore weight and activation magnitudes as saliency measures, as well as the impact on the objective function. Overall, they end up with a structured-unstructured hybrid method.

Zhang et al. (2016) proposed to learn a factorization to reconstruct the feature maps rather than the weight matrices, admittedly a better-motivated goal. Along the same lines, Yu et al. (2017) proposed to reconstruct both the weight matrices and the feature maps, but decomposed the weight matrices as $W \approx L + S$ where L is low rank and S is sparse.

Lebedev et al. (2015) proposed to look directly at the equivalent of SVD for the 4D tensors involved in convolution, the canonical polyadic decomposition (CP-decomposition). Interestingly the kernel decomposition comes down to a succession of a pointwise convolution, two convolutions along one spatial dimension and another pointwise convolution. Once the 4D tensor corresponding to the kernel is factorized, they further fine-tune the network.

It should be noted that when the weight matrices present special structures, dedicated decompositions might work better than standard SVD. For instance, Chen et al. (2018) proposed a special decomposition for the word embedding layers in natural language processing networks.

8.2.3.4 Quantization

Quantization is the idea of encoding parameters with fewer bits. Having fewer bits results in faster computation provided the appropriate hardware is available. There is also a direct gain in memory. As with low-rank approximation, quantized architectures can be trained from scratch or quantization can be applied at a later stage (often accompanied with fine-tuning). The interested reader can consult the work of Gholami et al. (2021) for a review of quantization.

8.2.3.5 Teacher-student transfer

Teacher-student transfer is the idea of transferring what a model—the teacher—knows to another model—the student. For the purpose of this chapter, the teacher-student transfer we are interested in is when the teacher is a big (or slow) network and the student is a small (or fast) network. We will focus on works for image classification with deep learning, although the idea is generic and goes beyond neural networks.

When data (possibly unlabeled) is available, teacher-student transfer is straightforward: run the training data through the teacher and train the student to mimic the output. To be more precise, let $\hat{y}_t(\cdot, \Psi) : \mathbb{X} \to \mathbb{R}^K$ be the teacher network (outputting logit vectors). The goal of fully supervised transfer is to train a student network $\hat{y}_s(\cdot, \Theta) : \mathbb{X} \to \mathbb{R}^K$ so that

$$\Theta \approx \min_{\Theta'} \mathbb{E}_{x,y} \left\{ \ell_{trf}\big(\hat{y}_s(x;\Theta'), \hat{y}_t(x;\Psi)\big) + \kappa \ell_{sup}\big(\hat{y}_s(x;\Theta'), y\big) \right\} \tag{8.5}$$

where κ balances the transfer loss ℓ_{trf} and the supervised loss ℓ_{sup}.

Buciluă, Caruana, and Niculescu-Mizil (2006) proposed to use the distance to the teacher in the logit space as loss for training ($\ell_{trf} = \ell_2$) and no supervision loss ($\kappa = 0$). In their case, the teacher is a cumbersome ensemble of neural networks.

Hinton, Vinyals, and Dean (2015) proposed to use the cross-entropy between the (softened) softmax probabilities of both networks as transfer loss, a scheme called knowledge distillation. Since then, teacher-student transfer is commonly referred to as distillation.

Romero et al. (2015) proposed an attention mechanism (called hints) between the middle layers of the teacher and the student. An attention mechanism is an additional loss component whose goal is to force the student to output feature maps similar to those of the teacher; the student is not only mimicking the teacher's output but also other parts of the network.

Zagoruyko and Komodakis (2017) proposed to place systematic attention mechanisms between the teacher and student. In their case, a parallel between teacher and student architectures can always be drawn so that trying to mimic the teacher in each latent space makes sense.

Yim et al. (2017) also proposed the use of more systematic attention mechanisms. Once more, this relies on having similar (but different) structures for the teacher and student. The attention mechanisms they proposed encourage the student to have similar cross-channel correlations as the teacher.

Along the same lines, Wang et al. (2018a) proposed to train a student network of a comparable structure as the teacher's by distilling sequentially the building blocks.

Srinivas and Fleuret (2018) proposed to also match the Jacobian matrices of the teacher and student. Since those are costly to compute they proposed to only compute a subset of the matrices.

Zhou et al. (2018) proposed to train the teacher and student networks at the same time. The two networks share the start (*i.e.* the early layers) of the model and diverge at some points with a bigger end for the teacher network. The two networks are then jointly trained to perform correct classification and output similar logit vectors. This is somewhat akin to what happens in the Inception network (Szegedy et al., 2015), although the goal here is to extract the small network, rather than help train the bigger part.

Xu, Hsu, and Huang (2018) proposed to learn the transfer loss function via a GAN-like architecture. This seems interesting when the student network is very small and does not have the sufficient capacity to mimic the teacher. See the discussion about failing the latent mapping assumption in Section 8.5.5 for a similar phenomenon.

Lee, Kim, and Song (2018) proposed an attention mechanism between corresponding blocks of the teacher and student which forces similarity in a low-rank basis obtained via SVD.

Kim, Park, and Kwak (2018) proposed an attention mechanism working in two steps. Firstly, a so-called paraphraser autoencoder is trained in an unsupervised way to be able to reconstruct the feature maps of the teacher. Then the student is joined to the paraphraser through another network: the translator. The student is then trained with a usual, supervised cross-entropy loss as well as with attention so that the translator would mimic the paraphraser.

Heo et al. (2019b) argued that the actual values of the feature maps are less important than whether or not the neurons are activated. Accordingly, they proposed to re-use the hints of Romero et al. (2015) to build a loss function which treats differently the teacher's hints according to whether or not the neurons are activated.

Park et al. (2019) proposed to use a loss function based on the $L2$ norm, normalized over batches, as well as an angular loss based on triplets of points to encourage the student to learn the same structure as the teacher.

Mirzadeh et al. (2020) proposed to use one or more intermediate assistant networks of decreasing complexity to improve the performances of the transfer.

A quick overview of fully-supervised distillation. Table 8.2 contains the results of the discussed papers on several common benchmarks. When more than one result was available (such as for methods with hyper-parameters), only the best one is reported. This might provide an optimistic view of the methods, which should not be closely compared based on these results (this is not the point being made).

Besides the obvious fact that benchmarks have evolved over time and networks have improved, Table 8.2 highlights an important fact regarding teacher-student transfer: only on rare occasions does the transfer substantially improve over learning directly the student. In many cases the student network trained by supervision is already quite close to the teacher, preventing spectacular improvements by distillation. When the gap is more pronounced, the gain from teacher-student is usually quite mild. Actually, Yuan et al. (2020) argued that this kind of gain is not really due to leveraging much from the teacher but rather to the regularization afforded by learning from soft labels, which can be obtained without having a teacher. The fact that on rare occasions the student is able to surpass the teacher is further evidence of this. In the end, there are only a few instances in which the improvement seems truly substantial, one of which is (due to Wang et al. (2018a) on CIFAR 10; [8] in the table) portrays an uncharacteristically low accuracy of 72.41% for a mere simplification of the teacher model.

This is not to say that the gain should be discarded, but rather that the question of whether the accuracy boost is worth the trouble when data is available should be carefully examined. Moreover, the more spectacular gains are due to attention mechanisms, which require the student to have a comparable architecture to the teacher, another obstacle. Distillation, however, shines when there is no/few/more data, which is the topic of the next section.

Besides compression, teacher-student transfer has been used widely, as it is such a flexible scheme. For instance, Domingos (1997) proposed to transfer from a large ensemble to a much simpler model for the sake of interpretability. Vongkulbhisal, Vinayavekhin, and Scarzanella (2019) proposed to merge several classifiers trained on overlapping label spaces into a single model by using left-over data for the transfer. Shi et al. (2017) and Cioppa et al. (2019) proposed the opposite: transferring only a subset of the teacher's classification capacity to the student. Bachman, Alsharif, and Precup (2014) proposed to generate an ensemble based on a single method by perturbating it and then transferring back the robustness to the original architecture as a form of regularization.

The interested reader can consult the work of Wang and Yoon (2020) and Gou et al. (2021) for more complete reviews.

TABLE 8.2: Accuracy comparison of different teacher-student transfer methods. The teacher, student and transferred columns correspond to the accuracy of the teacher, the student when learning directly from the training set, and the student taught by transfer from the teacher with the training set, respectively. Only the most favorable cases per dataset are reported. Standard deviations are present when available. Tag is either ⋆ to indicate a true gain from transfer (*i.e.* student much closer to teacher with transfer), or † to indicate closeness between teacher and student. [1] is Hinton, Vinyals, and Dean (2015), [2] is Romero et al. (2015), [3] is Sau and Balasubramanian (2016), [4] is Zagoruyko and Komodakis (2017), [5] is Yim et al. (2017), [6] is Srinivas and Fleuret (2018), [7] is Kim, Park, and Kwak (2018), [8] is Wang et al. (2018a), [9] is Zhou et al. (2018), [10] is Xu, Hsu, and Huang (2018), [11] is Lee, Kim, and Song (2018), [12] is Park et al. (2019). [13] is Heo et al. (2019b), [14] is Heo et al. (2019a), [15] is Mirzadeh et al. (2020).

SOURCE (YEAR)	TEACHER	STUDENT	TRANSFERRED	TAG
MNIST				
[1] (2015)	99.33	98.54	99.26	⋆ ?
[2] (2015)	99.45	-	99.49	
[3] (2016)	99.32	99.03	99.13	†
SVHN				
[2] (2015)	99.76	-	99.76	
[3] (2016)	96.18	95.40	95.55	†
[3] (2016)	96.18	95.40	95.55	†
[9] (2018)	-	96.42	97.80	
CIFAR 10				
[2] (2015)	90.18	-	91.61	
[3] (2016)	91.60	78.06	81.04	
[4] (2017)	94.77	93.69	94.29	†
[5] (2017)	91.91	87.91	88.70	
[6] (2018)	90.94	84.56	87.29	⋆
[7] (2018)	95.56	94.01	95.35	†
[8] (2018)	86.61	72.41	83.56	⋆
[9] (2018)	92.73	91.31	92.15	†
[10] (2018)	95.81	92.54	93.91	
[13] (2019)	95.49	93.53	94.42	†
[14] (2019)	92.55	86.02	87.32	
[14] (2019)	92.55	90.16	91.23	† ?
[15] (2020)	-	88.52	88.98	
CIFAR 100				
[2] (2015)	63.54	-	64.96	
[5] (2017)	64.06	58.65	63.33	⋆
[6] (2018)	64.78	54.28	54.57	
[7] (2018)	73.09	71.96	74.38	
[9] (2018)	-	56.30	67.00	
[10] (2018)	79.38	71.48	74.25	⋆ ?
[11] (2018)	64.44	61.37	65.05	
[11] (2018)	66.58	64.00	65.43	†
[12] (2019)	77.76	71.26	72.97	
[15] (2020)	-	61.37	61.82	
TINY IMAGENET				
[12] (2019)	61.55	54.45	56.36	
[14] (2019)	56.10	50.68	52.99	

8.3 Data-constrained compression

The methods discussed in the previous section have in common that they assume training data is available. Even low-rank approximations of the learned weight matrices, theoretically possible without data, are often accompanied with fine-tuning to recover the loss of accuracy due to the factorization. Yet there remain scenarios for which compression must be handled once data is no longer available (as discussed in Section 8.1.2).

Before looking at how data scarcity has been tackled so far, let us mention an altogether different approach to somehow combat data scarcity when it is foreseeable data will be useful in the future yet storage is limited: compress the *data* itself. Dataset distillation (*e.g.* Wang et al., 2018b; Sucholutsky and Schonlau, 2019; Sucholutsky and Schonlau, 2019; Nguyen et al., 2021) aims at deriving datapoints (possibly in the form of generated samples and labels) from a training set and a learned network so that learning with the distilled set would have resulted in the best possible accuracy. At present, the results are not yet satisfactory, especially when facing architecture changes.

In semi-supervised distillation (*e.g.* Li et al., 2014; Gong et al., 2018; Tang et al., 2019), data scarcity is tackled by relying, in addition to some small learning sample, on some unlabeled collection. These methods leverage the capacity of the teacher to produce "ground truth" for the unlabeled samples, which are assumed to be drawn from the original distribution.

As early as 2006, Buciluǎ, Caruana, and Niculescu-Mizil (2006) addressed data scarcity by using some form of data-augmentation when training the student network, taking once more advantage of the teacher capacity to provide labels. Since then, more modern data-augmentation techniques have been proposed (*e.g.* Kimura et al., 2018; Wang et al., 2018c). Another way to compensate for the lack of data is to extract more information from the teacher than only the output activations (see Section 8.2.3.5). All those methods can be bootstrapped from a very small learning set.

Recently, there has been a body of work trying to tackle the challenging task of zero-short distillation: transferring the teacher knowledge without any data (from the original distribution). These methods share a common principle which is to derive from the teacher some out-of-distribution (OOD) (see Chapter 7) loss useful to craft informative samples. They differ on the actual loss which is used, as well as on how they craft these samples. For the latter, the two main techniques which have been investigated are to adversarially perturb pure noise samples to minimize the OOD loss (Heo et al., 2019a; Nayak et al., 2019; Cai et al., 2020a; Haroush et al., 2020), and to include a generative adversarial network (Goodfellow et al., 2014) in the loop (Chen et al., 2019; Micaelli and Storkey, 2019; Choi et al., 2020). In the case of teacher-student transfer, these zero-shot sampling methods do not need to provide faithful samples (*i.e.* samples actually coming from the original distribution), only informative ones for transfer. Although offering good performance and needing neither a labeled nor unlabeled learning set, these methods are very heavy computation-wise, requiring thousands of adversarial perturbations or learning a whole generator network. Both approaches also introduce new hyper-parameters, including the whole generator architecture in the case of GAN, which are very difficult to tune in a setting where no data is available. Current practices turn to generative architectures which have been shown to work well precisely on the distributions being transferred which might provide over-optimistic results, especially when moving away from such distributions—or worse, when changing the domain altogether.

Rather than manufacturing the inputs at great cost, our goal is to take advantage of the availability of a collection of unlabeled images. The closest to this setting is the work of Xu et al. (2019), where the authors similarly train the student network on unlabeled images drawn from a distribution which differs from the base-task distribution. They however assume the availability of a small set of samples from the original distribution, from which the most relevant samples of the collection are determined through a lengthy and sophisticated procedure. Our method does not require these label points besides being much faster.

8.4 Distilling from an unlabeled collection

After introducing the setting (Section 8.4.1), this section develops the two key components of our solution to tackle fast distillation from an unlabeled collection. Section 8.4.2 presents how we propose to bias the sampling to rely mostly on informative samples, while Section 8.4.3 describes how to take full advantage of the learning signal.

8.4.1 Setting

Let $\mathcal{I}$ be the distribution of interest (also called the target or original distribution) on which a neural network $\hat{y}_t(\cdot, \Psi) : \mathbb{X} \to \mathbb{R}^K$ was learned.

Our goal is to learn a student network parametrized by Θ (rather than Ψ), denoted $\hat{y}_s(\cdot, \Theta) : \mathbb{X} \to \mathbb{R}^{\mathbb{K}}$, so that

$$\Theta \approx \min_{\Theta'} \mathbb{E}_{x\sim\mathcal{I}} \left\{ \ell_{DSCE(T=1)}\big(\hat{y}_s(x;\Theta'), \hat{y}_t(x;\Psi)\big) \right\} \tag{8.6}$$

$$\approx \min_{\Theta'} \mathbb{E}_{x\sim\mathcal{I}} \left\{ \ell_{CE}\big(\hat{p}_s(x;\Theta'), \hat{p}_t(x;\Psi)\big) \right\} \tag{8.7}$$

where $\hat{p}_t$ and $\hat{p}_s$ are the softmax probabilities of the teacher and student respectively. $DSCE(T)$ stands for double softmax cross-entropy with a temperature of T.

We *do not* have access to a dataset of samples from $\mathcal{I}$ but we assume access to an unlabeled collection of samples $\mathbb{K} = \{x_i \in \mathbb{X}\}_{i=1}^{n}$, where $x_i \sim \mathcal{O}$ $(\neq \mathcal{I})$.

Choice of the student network. Our method implements the "select the best network not exceeding a given overall size" (Section 8.2.2), where the size threshold is hard-coded in the student architecture and select must be understood to mean "train". As such the method is fairly general and can be used under any constraint, be it the on-disk footprint, the number of learnable parameters, the overall memory during inference, the actual inference time or the number of FLOPs (Section 8.2.2.1). The constraint is simply enforced by choosing an appropriate architecture.

It should also be noted that our method works whether the student is a pruned or quantized version of the teacher, or whether it is an altogether new architecture. We will focus on the latter as it is more general (quantized and pruned network can be considered as a different architecture), and can readily be deployed on standard hardware.

8.4.2 Biased sampling

8.4.2.1 Computing the sampling probabilities

In order to learn the student network, the loss in Eq. 8.7 must be estimated. It can be rewritten as

$$\mathbb{E}_{x\sim\mathcal{I}}\left\{\ell\big(\hat{y}_s(x),\hat{y}_t(x)\big)\right\} = \mathbb{E}_{x\sim\mathcal{O}}\left\{\beta(x)\ell\big(\hat{y}_s(x),\hat{y}_t(x)\big)\right\}, \tag{8.8}$$

where

$$\beta(x) = \frac{\mathbb{P}_{\mathcal{I}(x)}}{\mathbb{P}_{\mathcal{O}(x)}} \tag{8.9}$$

is the density ratio with $\mathbb{P}_{\mathcal{I}}(x)$ (resp. $\mathbb{P}_{\mathcal{O}}(x)$) the density of x for distribution $\mathcal{I}$ (resp. $\mathcal{O}$). Reweighing the samples in the estimate is legitimate since they are supposed to be in the support of $\mathcal{O}$. If β was known, one could train the network by uniformly sampling examples from the collection $\mathbb{K}$ and using Eq. 8.8 as the training loss. We propose instead to bias the sampling of the datapoints from $\mathbb{K}$ proportionally to the $\beta(x)$ weights for efficiency reasons. This nevertheless requires estimating the density ratio.

Estimating the density ratio. Let us introduce a random binary variable s, with $s = i$ if x is drawn from $\mathcal{I}$ and $s = o$ if x is drawn from $\mathcal{O}$. We can rewrite the density ratio as (using Bayes' rule):

$$\begin{aligned}\beta(x) &= \frac{\mathbb{P}_{\mathcal{I}}(x)}{\mathbb{P}_{\mathcal{O}}(x)} = \frac{\mathbb{P}(x|s=i)}{\mathbb{P}(x|s=o)} = \frac{\mathbb{P}(s=o)}{\mathbb{P}(s=i)}\frac{\mathbb{P}(s=i|x)}{\mathbb{P}(s=o|x)} \\ &= \frac{1-\pi}{\pi}\frac{\mathbb{P}(s=i|x)}{1-\mathbb{P}(s=i|x)} = \frac{1-\pi}{\pi}e^{\lambda g(x)}\end{aligned} \tag{8.10}$$

with

$$g(x) = \frac{1}{\lambda}\log\frac{\mathbb{P}(s=i|x)}{1-\mathbb{P}(s=i|x)} \tag{8.11}$$

is proportional to the log-odds ratio. Assuming we have access to $g(x)$ (see "Characterizing score" below), let us discuss how all this can be turned this into sampling probabilities.

From density ratio to sampling. Since our goal is to sample from $\mathbb{K}$ with probabilities proportional to β, we can simply normalize them:

$$q_i = \frac{\beta(x_i)}{\sum_{x\in\mathbb{K}} \beta(x)} = \frac{\frac{1-\pi}{\pi} e^{\lambda g(x_i)}}{\frac{1-\pi}{\pi}\sum_{x\in\mathbb{K}} e^{\lambda g(x)}} \tag{8.12}$$

$$= \operatorname{softmax}(\lambda g(x_i)) \tag{8.13}$$

We will denote by $\mathcal{Q}(\mathbb{K}, g, \lambda)$ the distribution over $\mathbb{K}$ reflecting those probabilities (dropping the arguments when the context is clear). Note that π needs not be estimated as far as the sampling probabilities are concerned.

Controlling the diversity. To train our student, we will sample the examples constituting each training batch from $\mathbb{K}$ using distribution $\mathcal{Q}$. To ensure some diversity in the selected samples, we can adjust the λ parameter, which controls an exploitation-exploration tradeoff.

If $\lambda = 0$, the distribution is uniform, while as $\lambda \to \infty$ only the sample(s) appearing the most to come from $\mathcal{I}$ will be selected.

Optimizing the hyper-parameter λ cannot be done in the usual fashions (*e.g.* cross-validation) in our setting but choosing a value can be done in a number of ways. Here we propose an intuitive non-parametric solution. Let l (resp. h) be the index of the lth (resp. hth) sample of $\mathbb{K}$ is ascending order of value of $\hat{g}$, by choosing a value for q_h/q_l we can isolate λ from

$$\log \frac{q_h}{q_l} = \lambda(\hat{g}(x_h) - \hat{g}(x_l)). \tag{8.14}$$

We propose to select l (resp. h) to correspond to the first (resp. third) quartile and dub q_h/q_l the inter-quartile sampling probability ratio (IQPR). If IQPR $=$ 5, for instance, the sample corresponding to the third quartile is five times more likely to be selected than the sample at the first one. Note that when IQPR $= 1$, the distribution $\mathcal{Q}$ is uniform.

Characterizing score. One question remains: how is the log-odds ratio g computed? Ideally, we would have a sample

$$\mathbb{S} = \{(x_1, i), \dots, (x_{N_i}, i), (x_{N_i+1}, o), \dots (x_{N_i+N_o}, o)\}$$

where the N_i first samples are i.i.d. from $\mathcal{I}$ and the remaining from $\mathcal{O}$. g could then be learned by training a probabilistic classifier to discriminate i and o samples. As we have seen in Section 3.2, the logistic regression can be interpreted as modeling the log-odds with a linear boundary.

Unfortunately, no such sample is available in our setting. Therefore, we propose to use a proxy $\hat{g}$, henceforth called characterizing score, in the form of some OOD loss. In that regard, it seems that all indicators developed in Chapter 7 could serve as characterizing scores. We will focus on two such scores: T1000 and 1C-Sum (see Sections 7.4.1 and 7.6).

8.4.2.2 Discussion

In this section, we motivate the design choices which have led to the formalization of the biasing mechanism described in the previous section.

Robustness. The most questionable choice regarding the biasing mechanism is surely the proxy for the log-odds. Resorting to OOD indicators suggests we are at best modeling something related to $\mathbb{P}_{\mathcal{I}}$. The indicators are agnostic of $\mathcal{O}$ and talking about odds in these circumstances might not make much sense.

Our solution is somewhat resilient to a bad choice of characterizing score. On the one hand, $\hat{g}$ needs only be a linear transformation of a good approximation of g: the softmax is translation invariant and λ can accommodate a scaling factor.

On the other hand, even an inadequate choice of score might prove to be useful. Since we are sampling from the collection—not selecting samples once and for all—we can choose λ so that the distribution is not too skewed.

Computational cost. Biasing the sampling incurs an additional cost of one forward pass of the teacher per sample in the collection. This is admittedly much less than what is required to learn a good generator or adversarially transform pure noise into useful samples, as proposed in the context of zero-shot distillation (see Section 8.3). Note that we could have instead sampled mini-batches uniformly from $\mathbb{K}$ and then reweighed them in the loss as in Eq. 8.16 (like Kimura et al., 2018; Tang et al., 2019). We believe there is no need to spend too much time on uninformative samples, however.

The importance of biasing. Prior to running any experiments, what are our expectations regarding the proposed biasing mechanism? The first thing to consider is that enforcing the same decision boundary as the teacher's in low β areas is not an issue in itself. Rather, the problem would be if that prevented imitating the teacher in more relevant areas. The effect of biasing is thus subjected to the capacity of the student and the collection size. For a high-capacity student, biasing might have very little effect. Indeed, if the student is able to mimic the teacher for all samples of the collection, biasing is useless. Actually, biasing might even hurt learning as doing without it might constitute a regularization mechanism.

On the other hand, the lower the capacity, the more a student might benefit from biasing. However, if the capacity is too low the student might end up being poor in terms of performance.

Finally, if the biasing is too severe, it will mostly act as if only a subset of the collection was available, in which case the performances should be hurt whatever the student capacity.

8.4.3 Capturing the learning signal: fixed-linear distillation under latent mapping assumption

Traditional teacher-student transfer encourages the student to replicate the teacher outputs, either by imposing a L_2 norm on the logit (Buciluǎ, Caruana, and Niculescu-Mizil, 2006) or, more frequently, a (softened) cross entropy loss on the output probabilities (Hinton, Vinyals, and Dean, 2015). The latter will be referred to as classical (or vanilla) distillation. See Section 8.2.3.5 for more details.

However, many problems (including most practical ones) have a small number of outputs. As a consequence, trying to replicate only those is wasteful sample-wise; a better approach would be to extract more constraints per input from the teacher's inner functioning. Attention mechanisms, whose goal is to force other parts of the student to behave as the teacher, have been proposed to guide and accelerate student training (*e.g.* Romero et al., 2015; Yim et al., 2017; Zagoruyko and Komodakis, 2017; Chung et al., 2020; Li et al., 2020). Although well motivated when inner parts of the student and teacher networks can be matched, they can not be applied in our more general setting where the two architectures are not (necessarily) related.

We propose to take advantage of the only part where a mapping can be expected even for unrelated architectures, *i.e.*, at the end of the feature extraction phase. Recall from Section 3.6.2.1 that a network can be viewed as

$$\hat{y}(x;\Theta) = Wz_{L-1}(x;\theta_{L-1}) + b \tag{8.15}$$

where $z_{L-1}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}^{p_{L-1}}$ is a feature extractor. Our idea is to enforce the student to match the teacher's feature representation. Since $p_{L-1} \gg K$ (the dimensionality of the latent space is much greater than the number of classes), this will put more constraints during training. By doing that, we hope to be more efficient when learning from samples which are not from the target distribution.

To formalize this idea, let us denote by $u_s \in \mathbb{R}^{p_s}$ (resp. $u_t \in \mathbb{R}^{p_t}$) the student (resp. teacher) latent vector corresponding to some x, and use the corresponding subscript for W, b. Given p_s might be different from p_t, the learning problem is then defined as follows:

$$\begin{cases} W_s = PW_t \\ b_s = b_t \\ \min_{\theta,P} \mathbb{E}_{x\sim Q} \, ||Pu_s(x;\theta) - u_t(x;\psi)||_2^2 \end{cases} \tag{8.16}$$

that is, we fix the linear part of the student and learn the same feature extraction as the teacher's. We will refer to Eq. 8.16 as fixed-linear distillation.

P serves to project the student latent vector onto the teacher space when $p_S \neq p_t$ so that $W_t u_t = W_s u_s = W_t P u_s$. When $p_s = p_t$, one should take P as the identity matrix and drop it from the objective. In contrast to how traditional attention mechanisms are incorporated in the learning, the loss has only one component and therefore no weighing hyper-parameter needs to be tuned (which would be hard to do in our setting).

TABLE 8.3: Details of the collections used as proxy.

COLLECTION	INCLUDED	TRANSFER SIZE
ORI	CIFAR 10	55000
REL	TINY IMAGENET	100000
	STL 10	102500
IRREL	MNIST (x2)	128000
	FASHION MNIST	64000
	SVHN	91963

	RELATIVE SIZE (IN %)		
COLLECTION	ORI	REL	IRREL
ORI + REL	21.36	78.64	-
ORI + IRREL	16.23	-	83.77
REL + IRREL	-	41.63	58.37

Although it is possible to fine-tune the linear part of the student by classical distillation, we expect the gain to be small (that part has already been optimized on the teacher) compared to the risk of disrupting the model (using a poorly chosen fine-tuning learning rate, for instance).

Latent mapping assumption. Forcing the student to match the feature extraction of the teacher poses the implicit assumption that the student is actually capable of doing so. We will refer to this as the latent mapping assumption.

Whether this assumption is valid depends on whether the teacher's feature extractor lies in the student's hypothesis space (or if it contains a good enough approximation). Since the student is expected to be smaller, the main reason for violating the assumption would be for the student to lack capacity.

It is dubious that a student architecture failing this assumption would turn out to be amenable for teaching by transfer whatever the loss. That being said, vanilla distillation might be more appropriate in such a case, as it would leave more flexibility for the student to learn a good approximation of the teacher's boundary. Anyway, the question of whether it makes sense to use such a student should be raised.

We will investigate the violation of the latent mapping assumption in Section 8.5.5.3.

8.5 Empirical analysis

In this section, we analyze the role played by the collection (Section 8.5.2), followed by a discussion of the influence of biasing the sampling (Section 8.5.3) and a study of fixed-linear distillation (Section 8.5.4). We then cover a few additional questions naturally raised by our methods (Section 8.5.5). We start by describing our protocol.

TABLE 8.4: Test set accuracy (in %) of the teacher networks.

	CIFAR 10	KMNIST
RESNET 50	94.11 ± 0.25	98.85 ± 0.01
DENSENET 121	94.30 ± 0.31	-

8.5.1 Protocol

Tasks. We evaluate our methodology extensively on CIFAR 10 (Krizhevsky, Hinton, et al., 2009) and more briefly on KMNIST (Clanuwat et al., 2018), using their standard test sets to assess model performance.

To constitute the collection of unlabeled images, we used MNIST (LeCun et al., 1998a), Fashion MNIST (Xiao, Rasul, and Vollgraf, 2017), SVHN (Netzer et al., 2011), STL 10 (Coates, Ng, and Lee, 2011) (with all the unlabeled images) and tiny ImageNet (Le and Yang, 2015). We used both train and test sets for the collection, only keeping 10% of the train set as validation set to monitor the loss. When the collection is made up of several datasets, we concatenated those validation sets. When images from the original task appear in the collection, images from the test set are not included obviously. All images were resized and cast to RGB to fit the network expectations.

We grouped some datasets to form meaningful collections as shown in Table 8.3. `Ori` is simply the original task and is included for the sake of the discussion. `Rel` stands for relevant, *i.e.* datasets whose label-space intersects with the original task's. `Irrel` contains only irrelevant datasets. We will also consider three combinations of those collections. The second part of Table 8.3 holds the relative size of each sub-collection. Note that the most realistic collection is `rel + irrel`.

Teachers. ResNet 50 (He et al., 2016) and DenseNet 121 (Huang et al., 2017) were used as teachers. All networks expect RGB images. The networks were optimized during 450 epochs by stochastic gradient descent (batches of size 64, weight decay of 5×10^{-4} and momentum of 0.9). The learning rate was initialized at 0.1. It was decreased by a factor of 10 after 150 epochs and again at epoch 300. Each decrease was accompanied by a restart from the best model according to the validation accuracy. Horizontal flip and random cropping (with a padding of 4) were used as data augmentation. Teacher accuracies are shown in Table 8.4.

Students and distillation. We used MobileNet v2 (Sandler et al., 2018) and ShuffleNet v2 (Ma et al., 2018) as students, which expect RGB images as well. The students were optimized with vanilla distillation or according to Eq. 8.16 with different values of IQPR. They were trained for the equivalent of 150 epochs of the target task's training set so as to reach convergence. To reach fast convergence, we considered pseudo-epochs of 5000 samples (accounting for 1350 of such pseudo-epochs) drawn with replacement from the collection, and divided the learning rate (initialized at 0.01) by approximately 0.4 after no improvement was seen on the held-out validation set from the collection for 20 pseudo-epochs. We used the same data augmentation scheme as for the teachers (data augmentation is performed after bias sampling). The results are averaged over three random initialization of the teacher and student networks. Unless mentioned otherwise, experiment results are based on 1C-Sum with CIFAR 10 as target task. All experiments were carried out with PyTorch (Paszke et al., 2017).

To give an idea of the relative sizes, ResNet 50 has approximately 24 millions parameters, DenseNet approximately 7 millions, MobileNet v2 approximately 2 millions and ShuffleNet v2 approximately 1.3 millions parameters.

8.5.2 Collection analysis

Table 8.5 shows CIFAR 10 test set accuracy after full fixed-linear distillation when using different collections as transfer set and for several from/to architecture pairs and three values of IQRP. We defer the comparison with vanilla distillation to Section 8.5.4.

As a sanity check, let us note that the best performance (up to 94% of accuracy) is obtained by using the original dataset without biasing the sampling (last column), a setting which is supposed to be impossible in our context. Two more prominent observations can be made: highly biasing the sampling can be harmful (as we envisioned near the end of Section 8.4.2) and only using irrelevant data results in very poor accuracies (although one might be surprised at how high an accuracy is achievable with such unrelated data).

Excellent performances can usually be obtained when the original data is part of the transfer collection. Interestingly, we see that even in the worst-case situation (`ori + irrel`, where good data represents less than one-fifth of the collection) very decent accuracy can be reached with uniform sampling, although biasing the sampling is most useful in this situation. This suggests that student training is robust to such irrelevant data, probably because the network is not saturated.

When only relevant samples are available (`rel`), biasing makes little sense (IQPR=1 offers the best accuracy). However, in the most realistic setting, where relevant and irrelevant samples form the collection (`rel + irrel`), a small bias (IQPR=5) usually offers a slight edge.

TABLE 8.5: CIFAR 10 test set accuracy (in %) after distillation with respect to the collection used as transfer set. Coloring is by teacher-student pairs and is linear with accuracy.

IQPR	REL + IRREL	REL	IRREL	ORI + REL	ORI + IRREL	ORI
			RESNET 50 TO MOBILENET			
1	91.46 ± 0.27	92.17 ± 0.08	72.24 ± 4.69	93.90 ± 0.21	93.06 ± 0.29	94.15 ± 0.34
5	91.04 ± 0.15	90.99 ± 0.35	68.67 ± 2.22	93.83 ± 0.55	93.99 ± 0.46	93.73 ± 0.23
25	89.25 ± 0.46	74.10 ± 9.15	58.17 ± 2.12	93.38 ± 0.28	93.60 ± 0.26	89.44 ± 0.27
			RESNET 50 TO SHUFFLENET			
1	90.34 ± 0.17	91.47 ± 0.05	66.14 ± 2.25	93.36 ± 0.31	91.93 ± 0.20	93.69 ± 0.25
5	90.63 ± 0.01	90.59 ± 0.00	64.78 ± 1.85	93.47 ± 0.40	92.37 ± 0.65	92.77 ± 0.08
25	87.90 ± 0.36	80.93 ± 0.71	55.06 ± 1.54	89.48 ± 3.10	92.69 ± 0.20	88.34 ± 0.09
			DENSENET 121 TO MOBILENET			
1	91.27 ± 0.40	91.60 ± 0.35	76.43 ± 0.80	93.92 ± 0.26	93.06 ± 0.17	94.36 ± 0.25
5	91.89 ± 0.04	91.41 ± 0.61	75.89 ± 0.96	94.05 ± 0.13	93.93 ± 0.25	93.81 ± 0.14
25	91.62 ± 0.49	88.85 ± 0.33	73.01 ± 0.02	87.68 ± 6.78	93.68 ± 0.78	90.29 ± 0.12
			DENSENET 121 TO SHUFFLENET			
1	90.37 ± 0.01	91.50 ± 0.53	70.35 ± 0.59	93.33 ± 0.17	91.79 ± 0.20	93.71 ± 0.01
5	91.14 ± 0.16	91.24 ± 0.35	70.31 ± 1.09	93.83 ± 0.28	93.24 ± 0.15	92.51 ± 0.08
25	90.63 ± 0.11	83.46 ± 2.59	67.90 ± 1.89	93.70 ± 0.11	93.41 ± 0.02	87.67 ± 0.78

8.5.3 Sampling analysis

In this section, we investigate the effect of the biased sampling. Although the previous section suggested that biasing had little impact, we can see from Figure 8.2 that it tends to accelerate the convergence to the final accuracy. On `rel + irrel`, using an IQPR of 5 instead of 1 results in an average accuracy of 77.5% instead of 69.1% at 10% of the learning. The gap remains wide during the whole training on `ori + irrel`. When using the original data (`ori`), biasing downgrades the performance, by masking examples. We expect biasing to provide an advantage in this setting only if the dataset contains outliers.

Table 8.6 offers more insight into the sampling mechanism. Skip ratio represents the percentage of samples from the collection which never get selected. Uniformity is the entropy of the empirical selection distribution. It is rescaled in the range $[0, 1]$ (close to 1 means uniform, close to 0 means highly biased) and ignores samples which are never selected. Finally, irrel. prop. is the percentage of the samples that are used at least once during the training that comes from `irrel`.

When there is a slight bias (IQPR=5), only a small fraction of the data is ignored. This percentage is the smallest on `ori` where the scores used to compute the actual sampling probabilities are supposed to be more uniform. The proportion of irrelevant samples is drastically reduced in the case of `ori + irrel`. This suggests that we are able to select samples from the original data quite well and explains the good results in Table 8.5.2 and Figure 8.2.

When the bias is more severe (IQPR=25), we see that a great proportion (20 to 40 %) of the data are totally discarded. Even for the remaining samples, the selection departs largely from a uniform distribution. As a consequence, many datapoints are mostly ignored. It is clear that this strategy can only pay off when the collection is polluted with many easily-identified irrelevant samples. Since this is close to pre-selecting the samples, this scheme also rids us of the benefits of compensating a bad choice of characterizing score. In any case, this accounts for the bad performances on `rel` and `ori` (Table 8.5).

8.5.4 Fixed-linear distillation analysis

To assess the impact of fixed-linear distillation, we compare it to classical distillation. Since a projection is involved for our variant, we carried out two tests: the first one with the default version of the student architecture and the second one with a modified student network where the pre-linear latent space is updated to match the dimensionality of the teacher's (2048 for ResNet 50). To do so, we simply modify the number of feature maps produced by the last convolution layer of our MobileNet student. The last convolution is indeed followed by global average pooling, resulting in one latent feature per feature map. In this case, the size of the latent space is increased (from 1280 to 2048), resulting in more parameters for this variant of the student. The results are collected in Table 8.7. Fixed-lin. + proj. corresponds to Eq. 8.7. Fixed-lin. 2048 is our method on the modified student.

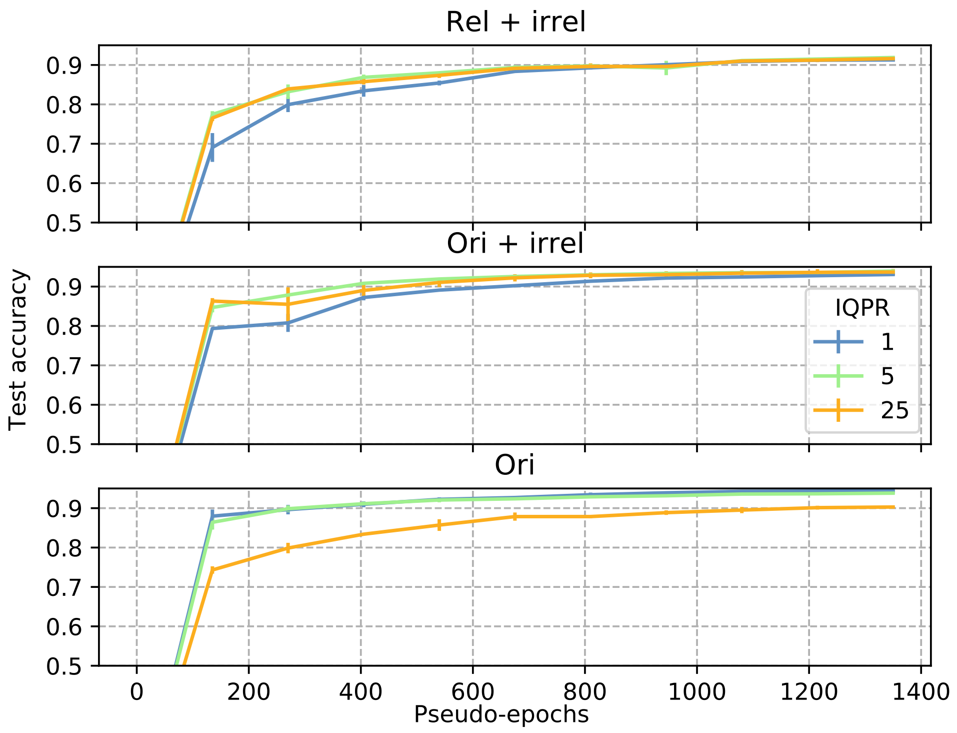


FIGURE 8.2: Convergence rate: CIFAR 10 test set accuracy with respect to the learning time and for several collections and IQPR values (based on DenseNet 121 to MobileNet).

Distill 2048 is vanilla distillation on the modified student, while Distill corresponds to vanilla distillation on the original student. We used a temperature of 2 for all vanilla distillations.

On the `rel + irrel` collection, there is a clear incentive to use the fixed-linear distillation throughout the whole learning procedure and irrespective of the IQPR. The projection, on the other hand, plays an insignificant role.

On the original data (`ori`), the usefulness of taking advantage of the latent space information clearly disappears, offering only a slight edge at the start. Interestingly, the projection is better able to keep up with the distillation than when the student's latent space is made to match the teacher's.

Overall, it appears more critical to exploit well the learning signal when using proxy data. The conclusion drawn regarding biased sampling seems to hold with classical distillation as well (*i.e.* useful on proxy data containing relevant samples to speed up convergence). By the end of learning, the modified student never significantly outperforms the original student and even tends to underperform (or at least is less stable) on the original data. As a consequence, there is no incentive to add more parameters to match the teacher.

TABLE 8.6: Biased sampling related metrics: skip ratio is the percentage of samples which are never selected; uniformity is the empirical entropy of the selected sample distribution; irrelevant proportion is the percentage of samples from `irrel` which are used at least once during training (based on transferring CIFAR 10 from DenseNet 121 to MobileNet). Cells marked with ⋆ have non-negligible standard deviation (see Supplementary for more details).

	SKIP RATIO		
IQPR	REL + IRREL	ORI + IRREL	ORI
1	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
5	3.53 ± 0.01	1.91 ± 0.01	0.37 ± 0.01
25	24.99 ± 0.15	40.09 ± 15.83	21.48 ± 7.36
	UNIFORMITY		
IQRP	REL + IRREL	ORI + IRREL	ORI
1	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
5	0.97 ± 0.00	0.96 ± 0.00	0.94 ± 0.00
25	0.95 ± 0.00	0.86 ± 0.07	0.60 ± 0.18
	IRRELEVANT PROPORTION		
IQPR	REL + IRREL	ORI + IRREL	ORI
1	58.36 ± 0.02	86.32 ± 0.00	-
5	42.12 ± 0.01	56.65 ± 0.01	-
25	38.84 ± 9.46	17.32 ± 12.81	-

8.5.5 Additional experiments

8.5.5.1 Influence of the characterizing score

All the previous experiments relied on the 1C-Sum score to bias the sampling. Table 8.8 revisits the transfer of CIFAR 10 from ResNet 50 to MobileNet with our method using the `rel + irrel` collection to highlight the effect of the characterizing score.

The T1000 score results in slightly worse accuracy than 1C-Sum on moderate bias (IQPR=5). The accuracy drops significantly when the bias increases, however. Interestingly, the score seems appropriate to detect the relevant samples. At IQPR=5, both scores skip about 7% of the data and T1000 slightly better rejects irrelevant samples. When IQPR=25, T1000 focuses on only a quarter of the samples but those are mostly relevant ones. In both cases, however, the results is a much less uniform distribution of samples compared to 1C-Sum.

This paradoxical situation is due to the fact that the characterizing score has two impacts on the learning (when IQPR > 1).

On the one hand, there is a direct effect: a bad choice of characterizing score would put forward irrelevant samples. In such a case, any IQPR value greater than one would result in a worse outcome than uniform sampling, since the student would be presented more often with "bad" samples.

On the other hand, the score distribution is important. Imagine the score is perfect with respect to a given collection, task and teacher (informally, all

TABLE 8.7: CIFAR 10 test set accuracy (in %) at 10% and 100% of learning with different distillation methods and collections. FL+P (Fixed-linear + projection) is our method; DS refers to the classical distillation; 2048 refers to modifying the student to match the latent space dimensionality of the teacher (based on ResNet 50 to MobileNet).

		IQPR = 1		IQPR = 5	
	METHOD	@ 10 %	@ 100 %	@ 10 %	@ 100 %
REL+IRREL	FL+P	70.33 ± 2.37	91.46 ± 0.27	75.77 ± 0.07	91.04 ± 0.15
	FL 2048	71.59 ± 2.65	91.34 ± 0.07	75.42 ± 0.74	91.18 ± 0.00
	DS 2048	63.52 ± 0.38	89.59 ± 0.46	68.45 ± 0.60	89.11 ± 0.38
	DS	64.02 ± 2.02	89.26 ± 0.97	69.03 ± 3.22	89.04 ± 0.75
ORI	FL+P	87.00 ± 0.37	94.15 ± 0.34	85.69 ± 1.03	93.73 ± 0.23
	FL 2048	87.96 ± 0.28	93.10 ± 1.37	85.92 ± 1.15	91.99 ± 1.18
	DS 2048	86.85 ± 0.13	95.01 ± 0.07	84.73 ± 0.59	94.03 ± 0.10
	DS	86.52 ± 0.05	94.97 ± 0.20	84.91 ± 0.19	94.12 ± 0.20

TABLE 8.8: Comparison of characterizing score (based on transferring CIFAR 10 from ResNet 50 to MobileNet). IP stands for irrevelant proportion.

IQPR	SCORE	ACCURACY (%)	SKIP RATIO	UNIFORMITY	IP
5	1C-SUM	91.04 ± 0.15	7.04	0.96	51.57
	T1000	89.57 ± 0.58	6.85	0.91	48.39
25	1C-SUM	89.25 ± 0.46	35.69	0.85	32.63
	T1000	74.69 ± 1.08	75.76	0.55	7.00

relevant samples of the collection for task score higher than irrelevant ones) and there is exactly one-quarter of irrelevant samples. The second quarter (containing only relevant samples) will almost never be selected if its components are much closer to the irrelevant samples than to the third quarter, for instance.

T1000 distribution is not suited for our problem, even though it might ultimately be a good characterizing score. A different scheme for transforming the score into a sampling probability might be more appropriate for T1000.

8.5.5.2 One collection to rule them all

An advantage of having a large collection of data from many sources is that the same collection can be used for different tasks. Using the same protocol as for the other experiments, we transferred by fixed-linear distillation a ResNet 50 teacher learned on KMNIST into a MobileNet student using the `rel + irrel` collection (IQPR=25). We reached an accuracy of 97.45% $\pm$ 0.80. We thus see that re-using a collection which performs well on CIFAR 10 leads also to close-to-teacher (98.85% $\pm$ 0.01, see Table 8.4) accuracy on an unrelated problem. This time, the proportion of what was considered as "irrelevant" samples for CIFAR 10 (MNIST, Fashion MNIST, SVHN) increases up to reaching 75.09% $\pm$ 4.04. Such samples are much more relevant with respect

TABLE 8.9: TwoConvNet: architecture details. DC stands for depthwise convolution, PC for pointwise convolution, BN for batch-normalization, GAP for global average pooling, FC for fully connected.

LAYER TYPE	OUTPUT SIZE	DETAILS
DC	$109 \times 109 \times 63$	KERNEL 7×7, STRIDE 2
PC	$109 \times 109 \times 512$	KERNEL 1×1
BN	$109 \times 109 \times 512$	
RELU	$109 \times 109 \times 512$	
DC	$53 \times 53 \times 1024$	KERNEL 5×5, STRIDE 2
PC	$53 \times 53 \times 2048$	KERNEL 1×1
BN	$53 \times 53 \times 2048$	
RELU	$53 \times 53 \times 2048$	
GAP	2048	
FC	10	

to KMNIST, illustrating that the characterizing score is indeed able to focus on the most relevant datapoints.

8.5.5.3 Failing the latent mapping assumption

When proposing the fixed-linear distillation, we assumed that some correspondence between the teacher's and student's latent space existed. This seems to be the case when transferring from ResNet 50/DenseNet 121 to MobileNet/ShuffleNet. When the architectures are widely different, the assumption might not hold, however.

To test this, we used as student a network composed of two depthwise-separable convolutions followed by a traditional linear part. This network will be referred to as TwoConvNet and is detailed in Table 8.9 (we do not expect the architectural details to bear much weight on the conclusions, however). Although the number of parameters of this network is of the same order as MobileNet's (*i.e.* roughly two millions parameters), TwoConvNet is far from being as deep. We learned a TwoConvNet by fixed-linear distillation and classical distillation, following the same protocol as for the other experiments. We set the IQPR to 1 as we only wanted to test the mapping assumption. We obtained an accuracy of 56.06 by fixed-linear distillation and of 58.46 by classical distillation. This suggests that fixed-linear distillation is inappropriate when the latent spaces from the student cannot emulate the teacher's. Admittedly, classical distillation does not work well either, since the network is too shallow for CIFAR 10.

8.6 Conclusion

In this chapter, we discussed the problem of neural network compression, an important problem as motivated in Section 8.1.2. There is notably room for compression due to the fact that neural networks are over-parametrized (see

Section 8.2.1) making them cumbersome, slow, and energy-inefficient. Section 8.2.2 and Section 8.2.3 discuss how the problem can be tackled at high- and lower-level, respectively. Section 8.3 then turns to how compression can be done in a data-constrained environment.

In Section 8.4 we tackle the challenging task of distilling a large teacher network into a smaller one—the student—in the absence of the original data. To do so, we proposed to leverage a collection of unlabeled samples which is supposed to contain "relevant" samples. We focused on image classification for which such data bank is likely to exist.

To fully take advantage of the available collection, we proposed to (i) bias the sampling to present more often data which appear relevant in the sense of some characterizing score (Section 8.4.2), and (ii) better exploit the learning signal via fixed-linear distillation (Section 8.4.3). To control the former, we introduced a simple hyper-parameter (IQPR). Contrary to related works, we are able to focus on relevant samples without requiring (a small part of) the original data and the whole learning runs in a time comparable to what would be required to directly learn the student, were the target dataset available.

We illustrated that good performances could indeed be reached when either the collection contained relevant samples or, unrealistically, the original data itself (Section 8.5.2). We observed that biasing the sampling could speed up, or even help, the learning when irrelevant data is part of the collection (Section 8.5.3). Biasing to the point where many datapoints are ignored might result in suboptimal performances, however. As for the fixed-linear distillation, we showed it was called for in our setting where the original data is missing (or we are on a tight training budget); using more information from the latent space significantly helps the learning (Section 8.5.4).

We then delved into some additional topics (Section 8.5.5). Firstly, we illustrated that a good characterizing score (in this context) must not only be able to highlight relevant samples but should also conform to some constraints on the distribution it produces. Then, we showed that a same collection can be useful for several tasks (possibly with different subsets accounting for the relevant samples) and that our fixed-linear distillation is subject to some latent mapping assumptions.

This work opens some avenues for future works. Firstly, the way the biasing mechanism is controlled could be improved. The simple scheme we proposed works well in tandem with 1C-Sum but is less successful with T1000 (Section 8.5.5). A first step might be to lower the biasing as learning progresses, so as to focus more on relevant samples at the start and reduce the instability when close to convergence. Scheduling the biasing would, however, introduce new hyper-parameters.

Further ideas relating to the sampling bias would be to ensure each class of the learning problem receives the same amount of samples (for a balanced problem) or to incorporate in the sampling probabilities the relative loss of each sample, so as to skip samples which will not affect the loss much anyway. How to weigh the latter so that it would not encourage uniformity by highlighting less-seen datapoints during the whole learning (thus neutralizing the intend of the bias) is unclear, however.

Besides the sampling mechanism, being able to transfer more from the teacher to the student might provide the final edge to close the gap between the two. Re-using proposed attention mechanisms (see Section 8.2.3.5) when possible, for instance with quantized or pruned students, is straightforward. In the more general case, the works of Park et al. (2019), which proposed a metric learning kind of loss, or the one of Xu, Hsu, and Huang (2018), which proposed to learn how to distill via a GAN-like architecture, might provide inspirations.

The sample-free setting is so constrained that it prevents controlling the accuracy of the final model, or even simply tuning hyper-parameters. An interesting question is whether the unlabeled collection could act as proxy for those as well. For instance, is there a high enough correlation between the true accuracy and the loss on the unlabeled collection to make informed-yet-imperfect decisions? Would it suffice to, say, choose between two roughly equivalent in terms of numbers of parameters (or FLOPs) but otherwise quite different student architectures? If yes to some extent, would it be better to create a validation set randomly from the unlabeled collection, or should be made by using the characterization score? Such questions required further analysis?

To conclude, let us note that the most critical aspect is to have access to relevant data. Building a larger collection might well prevail over building a better one or improving the distillation algorithm.

Chapter 9 Interpretable Machine Learning

Chapter overview

This chapter is about interpretability, the idea that a model can be dissected to get a better idea of its inner workings—or of the phenomenon it is modeling.

This chapter is divided into three main parts. Section 9.1 discusses interpretability in general. Then two cases of interpretability are scrutinized. Section 9.2 revisit the idea of GIFs (see Chapter 6) as a way to extract sufficiently small models to turn them into a list of interpretable rules. Section 9.3 discusses the problem of feature importance and examines how neural networks fare in this area compared to other tree-based state-of-the-art methods. Since the two parts are quite distinct, a more detailed overview of them is given therein. Finally, Section 9.4 concludes this chapter.

9.1 Interpretability

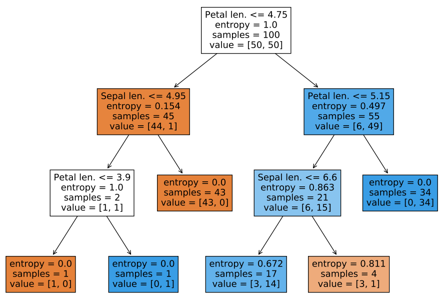
In this section, we investigate general notions relating to interpretability.

9.1.1 Motivation and high-level goals

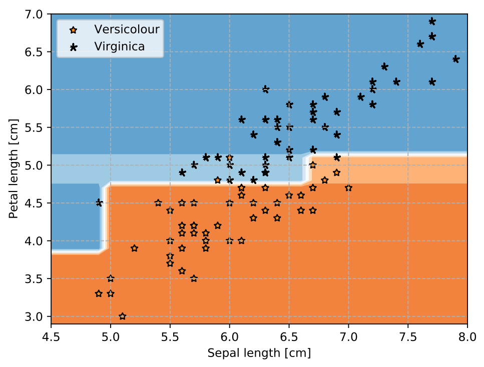
Can you trust your model? Is your model making decisions reflecting the discrimination within the data it was trained on? Can machine learning help us broaden our understanding of the wider world? Is there something to actually *learn* from the data beyond just memorizing it? Is your model learning the "right" thing? These questions dip into the broad field of interpretability, whose general goal is to provide insight into the input-output relationship the model is trying to capture.

From this somewhat vague definition stem many methods which can be dichotomized in *local* and *global* techniques.

Global interpretability. The goal of a global interpretability method is to shed some light on the input-output relationship in general, that is irrespective of any given samples. This can take many forms. For instance, one



(A) Structure of a classification tree. The first line in the box corresponds to the split (absent in leaves). The second line is the entropy of the sample in terms of class distribution. The third line is the number of samples and the following line is the distribution in terms of classes.



(B) A classification boundary. The boundary is piece-wise and axis-aligned, a manifestation of the paths along splitting nodes.

FIGURE 9.1: A decision (classification) tree and its boundary for the iris species classification (see Chapter 2 for more details).

might try to understand under which situation a natural phenomenon occurs. Someone else might be interested in seeing which of the input variables bear the most weight on the output. Conversely, one might want to detect irrelevant variables, or redundant ones.

An example of a model providing global interpretability was given in Chapter 3 with decision trees and is re-introduced here in Figure 9.1a. The simplicity of the tree ensures it is possible to get insight into the input-output relationship. A close look at the tree reveals the Virginia species entails longer petals, with long Versicolor iris being rather the exception than the rule. In this case, it is easy to check visually that this high-level interpretation makes sense (Figure 9.1).

Other examples of global methods include (sparse) linear models where the linear coefficients (associated with the standardized variables) inform on the input-output relationship. Further instances will be discussed in Section 9.2 in the case of rules sets and and in Section 9.3 when discussing input variable importances.

Local interpretability. Contrary to global methods, local techniques aim at deciphering why a given sample receives its output. How comes a patient received such a diagnosis? Can a medical doctor double-check it by looking at what part of a scan triggered the model? These kinds of questions are *local*; what happens outside of the investigated case is of no moment.

Once more, Figure 9.1a can serve for illustration purposes. Consider the local query of why an iris with a petal length of 4.9 and a sepal length of 6.5 is classified as Versicolor? Following the path the example takes, we end up on the right-most leaf of depth 3 and we see that a few training instances fell into that area.

As is evident from the example of the decision tree, local and global methods usually interact. Global knowledge is oftentimes relevant locally and can serve as a first analysis step. Conversely, local techniques can usually form the basis of a global one by aggregating information over a whole set of samples (see Section 9.3).

Intrinsic or extrinsic interpretability. In general, having access to the model (and possibly data) is not enough to glean much understanding of the input-output relationship, and neither is looking at how it responds to a given sample enough to fully grasp what are the key factors influencing its decision. Models not so easily interpretable, such as neural networks (and large and deep decision forests) are qualified of black-boxes[1]. In contrast, a model which is not a black-box is intrinsically interpretable.

Black-boxes might still be subjected to interpretability via additional computation steps, making them extrinsically interpretable. Such cases are often referred to as post-hoc interpretations (Molnar, 2020). Some methods might try to crack open the box while others are totally agnostic to the model.

The boundary between black-box and intrinsically interpretable models is rather vague. In reality, there is a spectrum of how much work is needed to get answers to interpretability queries. This overhead might also change—implying the intrinsic/extrinsic nature of the model as well—-with the type of queries.

Types of queries. The scale at which interpretability is investigated is not the sole factor influencing the nature of the queries one might ask a trained model, as the start of this chapter suggests. We will focus on two types of queries. Section 9.2 will look at getting insight into the input-output relationship (in a statistical sense rather than in a causal one; see Section 4.3). Section 9.3 will examine the question of which input variable influences the output the most (at a global scale).

Note that by considering different sub-problems (via the queries which can be formulated) as part of "interpretable machine learning", we conveniently sidestep the need for a clear definition of what interpretability (or more generally explainable artificial intelligence, XAI) stands for exactly; a known difficult issue. Rather, the next section will discuss a few desirable properties common to the sub-problems which typically fall under the notion of interpretability.

9.1.2 Feasibility and mid-level goals

Even though interpretability is a broad topic and might mean several things, there are general properties that could be expected. Here are a few of them.

[1]"Black-box" alludes to the idea of opacity, which comes in different flavors. In Chapter 7, we mentioned *white-box* access to a neural network. In this chapter, *black-box* is to be understood as meaning opaque in the sense that even when looking at its parameter (*i.e.* with white-box access) it is not easy to understand what the network is doing.

9.1.2.1 Desirable properties

Accuracy. The first thing we ought to expect from an interpretation is accuracy, which is the combination of two sub-questions. Firstly, do those interpretations actually reflect how the input-output relationship works (faithfulness)? Secondly, are all the important factors captured by the interpretations (completeness)?

The accuracy discussed in this paragraph relates to interpretability and is sometimes called *descriptive* accuracy (Murdoch et al., 2019). It must not be confused with *predictive* accuracy, which corresponds to how well the model is able to make predictions.

Simplicity. To be of use, interpretation must be concise (or *simple*) in order to fit the cognitive bandwidth of the interpreter. In this regard, Erasmus, Brunet, and Fisher (2020) felt the need to define interpretation as "[...] something that one does to an *explanation* to make it more *understandable*". For intrinsically interpretable models, this usually takes the form of a size constraint.

The requirement for getting an intelligible picture is counterbalanced by the ever-present need for an accurate one (in the predictive sense). Those two do not go hand in hand. Forcing the hypothesis space toward simple models is likely to increase the bias of the learning algorithm. Thus, too complex is hard to grasp but too simple and the full picture becomes the fool's: if the model is not accurate enough, how can the insights it brings be trusted?

This balance is of epistemic consequence as some problems might not offer the fragile compromise between simplicity and accuracy within the realm of intrinsically interpretable models.

Extrinsic methods might somewhat provide a go-between accurate models and simple interpretations but this raises an important question: how can one trust a simple interpretation derived from an accurate-yet-complex model when a simple model is not itself accurate enough? Of course, the question falls short if the extrinsic interpretations are proven to be accurate—not an easy feat in the context of extrinsic methods as we will argue in a couple of paragraphs.

Stability. Some authors (*e.g.* Murdoch et al., 2019; Yu and Kumbier, 2019; Bénard et al., 2021) also suggest that interpretations be stable. That is, interpretations derived from models (belonging to the same hypothesis space) inferred on different learning sets should roughly come up with the same interpretations. Although predictive stability should be expected (a sign that the model is not overfitting), descriptive stability might not follow.

For instance, a problem with redundant variables may swap those variables around without changing anything as far as predictions are concerned. Without going as far as redundant variables, dependency between the input features may well lead to several decision paths, equivalent so long as predictions are concerned.

Ideally, querying the model would uncover all those phenomena, in which case interpretation could be stable. In practice, these are difficult questions and partial answers might end up being significantly unstable. It should also be mentioned that uncovering natural redundancy of the phenomenon conflicts with the simplicity of interpretation.

Relevancy. Murdoch et al. (2019) also advocate for the notion of *relevancy*, *i.e.* the fact that interpretability is contextual and different operators might formulate different queries. For instance, a model might provide a good measure of which features bring information for the prediction but might not quite answer why a specific prediction is made. The same model might thus be interpretable in one context and not in another.

An fundamental dichotomy of queries is whether they relate to the model (*e.g.* "why is the model predicting y for input x?") or whether they relate to the underlying phenomenon (*e.g.* "why is y the ground truth for input x?"). The former is usually necessary to convince experts that a task can be automated by a model. This in turns implies some means of assessing (qualitatively at least) the model. In the latter case, the answer is unknown by design and thus interpretations are harder to assess.

As Watson and Floridi (2020) note, relevancy and stability interact: not all equally accurate (in the descriptive sense) explanations hold the same relevancy—the context determines what is relevant and what is not.

The interested reader can consult the works of Murdoch et al. (2019), Watson and Floridi (2020) and Zednik (2021) for a more in-depth discussion of the desirable properties.

9.1.2.2 Feasibility

Is reaping knowledge from data possible? The first thing to note in order to reply to this question is that the answer is contextual: relevancy and simplicity depend on the actual question being asked. The biggest obstacle to answering, however, might be relative to descriptive accuracy.

Whereas supervised learning benefits from a clear assessment methodology, interpretability, as a whole, does not. This is especially true of queries relating to the phenomenon for which, were the ground-truth known, the question would not arise. At best, the techniques can be evaluated on specific cases where the answer *is* known, either by other means (such as prior knowledge; see the ozone dataset in Section 9.2.1) or by design (see the benchmark datasets in Section 9.3.5). Applying such methods to other problems is an inductive leap which may demand strong theoretical guarantees to compensate for the lack of empirical control.

A possible counter-argument for the lack of assessment is that predictive accuracy can act as a surrogate for descriptive accuracy. As we have highlighted in Section 5.2.5, predictive accuracy seems indeed to be a necessary requirement[2]. The conclusion that it can serve as a surrogate measure does not follow from this premise, however. In the case of extrinsic interpretation, it is clear that a model could be good yet provide bad explanations, especially if the interpretation extraction mechanisms are not well-motivated by theory. Section 9.3 will provide an example of this. Whether having a model with high predictive accuracy is sufficient to trust intrinsic interpretability is unclear but the fact the same mechanisms are used for interpretability and prediction leans in the direction of the sufficient condition.

Despite these reservations, we believe interpretability is worth tackling and achievable to some extent. Easy problems offer accurate hypotheses, simple enough to be closely examined. Hard problems might not be fully dissectable, but the accuracy of the interpretations can be maintained by favoring faithfulness and foregoing completeness: the truth, nothing but the truth but maybe not the whole truth. Whether it be intrinsic or not, the method should, once more, come with a knob to adjust this faithfulness-completeness-conciseness tradeoff to the problem at hand.

Alternatively, one can turn to easier problems: going from global to focal (*e.g.* only look at one class or part of the prediction spectrum) to local, abstract more what interpretations are sought out, and so on.

Caution and epistemic modesty are called for, however, especially as we move from simple phenomena subjected to local and intrinsic queries to complex phenomena under global and extrinsic scrutiny. And this is even more true when interpretability is to be understood as causality—an endeavor far surpassing the modest goal of this chapter.

9.2 GIF as a rule extraction algorithm

Section overview

In this section, we come back to GIFs (Chapter 6). So far they have mainly been (and originally were) motivated by memory constraints, which are enforced by their node budgets. They are not etched in this sole context, however. Here they are re-contextualized for the sake of interpretability in the form of rule sets.

Section 9.2.1 presents the interpretable model of rule sets and shows how GIFs can be used to extract rules. Section 9.2.2 showcases that GIFs can actually serve the purpose of rule extraction. Finally, Section 9.2.3 quickly comes back to the topic of stability in the context of rule sets.

[2]To some extent, at the least. It could be that a model captures some of the underlying structure of the input-output relationship without capturing enough to reach high predictive accuracy. This limited understanding might still provide insightful knowledge but the model is clearly incapable of giving a full account of the underlying phenomenon.

9.2.1 Rule sets as an interpretable model

Rule set example. An example of rule set for the ozone LA dataset (Friedman, Hastie, and Tibshirani, 2001a) is given at Table 9.1. The dataset consists of 330 samples and the goal is to predict the maximum ozone concentration level (`ozone`) thanks to nine (meteorological) variables: the height the 500 mb constant pressure surface (`vh`), the wind speed (`wind`), the relative humidity (`humidity`), the temperature (`temp`), the inversion base height (`ibh`), the pressure gradient (`dpg`), the inversion base temperature (`ibt`), the visibility (`vis`) and the day of the year (`doy`).

Rules are of the form if-condition-then-prediction. From them, the output value for a given sample can be made by adding the intercept and all the predictions associated with the rules the sample falls into. When several rules with a common prefix match, only the more refined rule is followed. For instance, a sample with a `temp`=65 would match the second and seventh rules but not the first.

So far as interpretability is concerned, understanding how the ozone concentration relates to the other variables (a global interpretability query) is mainly done by looking at how a rule affects the output value (in relative terms). For instance, we see that the ozone concentration increases significantly when the temperature is more than 71.1 °F ($\approx$ 21.7 °C). The "prop." mention in the first column corresponds to the proportion of training instances meeting each rule. From them, we can deduce that, although the concentration level increases significantly when the temperature is high, the base height of the temperature inversion might play a more important role as it influences the prediction a bit less but plays a more frequent role. The proportion allows to filter out actual rules from important exceptions.

Fortunately, the rule set seems to conform to our understanding of tropospheric ozone. Sunlight plays a role in both the formation of ozone and temperature, while a closer-to-ground temperature inversion suggests stable weather which prevents the dispersion of ozone.

From GIFs to rule sets. Notice that the conditions are of the same nature as the splitting nodes in decision trees, or a conjunction of such splits. This should suggest a way of building rule sets: learn a decision tree and write down all paths. Since rule sets offer a mechanism for inference and fall into the category of intrinsic methods, the descriptive accuracy of the rules can be assimilated to the predictive accuracy of the model for the base, supervised task. Simplicity relates to the tree's size. Note however that a simple rule set must (i) contain few rules, and (ii) must have intelligible rules, which comes down to having shallow rules (*i.e.* with a small number of "ands").

With a single tree, there is a direct relationship between the number of leaves and the number of rules, leading to a rigid tradeoff between accuracy and simplicity. Rule sets can instead be built around decision forests, making sure the prediction of the rules reflects those of the forest.

GIFs come as a perfect fit for this problem. They naturally turn toward smaller forests. The complexity of the rule set can be specified via the node budget while the ideal shape is left for the algorithm to determine, possibly

TABLE 9.1: Example of a rule set for the ozone dataset isomorphic to a GIF ($m = 1000$, $CW = +\infty$, $\lambda = 1$).

		Intercept: 11.8			
prop. = 0.58	If	temp > 58.88	then	ozone =	1.11
prop. = 0.33	If	temp > 58.88 and temp ≤ 72.85	then	ozone =	2.21
prop. = 0.28	If	vh ≤ 5706.12	then	ozone =	-1.07
prop. = 0.48	If	ibh > 2244.43	then	ozone =	-5.80
prop. = 0.27	If	temp > 71.10	then	ozone =	9.74
prop. = 0.10	If	temp > 71.10 and ibt ≤ 227.83	then	ozone =	5.76
prop. = 0.40	If	temp > 65.20	then	ozone =	2.25
prop. = 0.41	If	ibh ≤ 1476.56	then	ozone =	-1.23
prop. = 0.05	If	ibt > 285.86	then	ozone =	3.58
prop. = 0.01	If	ibt > 285.86 and humidity ≤ 49.59	then	ozone =	-6.62
prop. = 0.27	If	doy > 266.85	then	ozone =	-0.82
prop. = 0.19	If	humidity ≤ 39.84	then	ozone =	-3.10
prop. = 0.47	If	ibh > 2386.16	then	ozone =	0.78

with hyper-parameters encouraging deeper or broader models. Moreover, the pre-pruning nature of the algorithm ensures that a very long branch will not get picked up early—an advantage for once.

Since we are aiming for a very low budget in order to ensure a simple rule set, GIFs will need to optimize their nodes as best as possible. Therefore, we expect a high learning rate and a large candidate window will be needed.

9.2.2 Empirical results

We closely follow the recent work of Bénard et al. (2021), introducing the SIRUS-R method, to evaluate how GIFs perform as rule extractors. The paper compares three methods and several baselines. RuleFit (Friedman and Popescu, 2008) is a boosting method topped with a lasso regularization to extract small rule sets. The number of rules and their depth are implicitly controlled by the penalty of the lasso constraint. Node Harvest (Meinshausen, 2010) is a random-forest-based method where a small forest is first learned in a traditional way before being globally refitted. The number of rules and their depth are decided by the parameters controlling the size of the forest. Finally, SIRUS-R learns a random forest restricting splits to q-quantile values (usually $q = 10$) and then keeps only the most frequent rules of the forest. The number of rules is governed by the minimum occurrence frequency. The depth of the rules is indirectly controlled in the same way since deeper rules are (exponentially) less likely to appear. The datasets used for the experiments are described in Table 9.2, with 10% of the samples dedicated to the test set.

GIFs hyper-parameters. Considering the goal of producing a small list of rules, GIF hyper-parameters have been chosen so that $\lambda = 1$, $CW = +\infty$ and $m = 1000$. Indeed, with a handful of rules, nodes need to be optimized as

TABLE 9.2: Dataset description.

DATASET	SIZE	DIM.	SOURCE
OZONE	330	9	FRIEDMAN, HASTIE, AND TIBSHIRANI (2001A)
MPG	398	7	BLAKE AND MERZ (1998)
HOUSING	506	13	HARRISON JR AND RUBINFELD (1978)
DIABETES	442	10	EFRON ET AL. (2004)
MACHINE	209	6	BLAKE AND MERZ (1998)
ABALONE	4177	8	BLAKE AND MERZ (1998)

TABLE 9.3: Number of rules extracted by method. Expanded from Bénard et al. (2021).

DATASET	DEC. TREE	RULEFIT	NODE HARVEST	SIRUS-R	GIF
OZONE	15	21	46	11	13.6 ± 0.9
MPG	15	40	43	9	10.9 ± 0.5
HOUSING	15	54	40	6	7.3 ± 0.5
DIABETES	12	25	42	12	12.9 ± 0.7
MACHINE	8	44	42	9	10.4 ± 0.5
ABALONE	20	58	35	6	8.3 ± 0.6

much as possible and the risk of overfitting is slight. The node budget, on the other hand, was chosen so as to be close to the number of rules of SIRUS-R (Table 9.3). Note that the number of rules extracted by GIF is slightly greater than for SIRUS-R. However, the rules of the latter contain a "else" clause which allows for a more accurate prediction on a per-rule basis.

Analysis. Our main result is Table 9.4 where the proportion of unexplained variance (*i.e.* the mean squared error divided by the variance of the output) is reported for several methods. Random forest and decision tree are included as a baseline. The results for RuleFit, Node Harvest, SIRUS-R and SIRUS-R (50) are directly taken from Bénard et al. (2021). In the case of GIFs, performances are reported over 10 random shuffling of the datasets. Extra-trees (see Section 3.4.3) are 1000 fully-developed trees with otherwise default parameters.

As can be seen, GIF holds its own in this context. On ozone and machine, it performs best on average. On the other datasets, it is on par with Node Harvest and SIRUS-R, with a slight edge on housing and abalone compared to SIRUS-R.

Node budget vs. number of rules. GIFs, as implemented, do not allow to specify the number of rules they should extract. As a rule of thumb, for a small list of rules, the number of rules is approximately two-thirds of the node budget (see Table 9.5). Indeed, a depth-1 rule accounts for two nodes: the root, which provides the condition and a leaf, which provides the prediction. Similarly, a depth-2 node with no sibling accounts for three nodes.

TABLE 9.4: Proportion of unexplained variance for several rule extractors. Expanded from Bénard et al. (2021). The uncolored rows are baselines. Coloring is linear by dataset.

METHOD	OZONE	MPG	HOUSING
EXTRA-TREES	0.25 ± 0.04	0.10 ± 0.02	0.14 ± 0.10
RANDOM FOREST	0.25	0.13	0.13
DECISION TREE	0.36	0.20	0.28
RULEFIT	0.36	0.15	0.16
NODE HARVEST	0.31	0.20	0.24
SIRUS-R	0.32	0.21	0.31
GIF	0.26 ± 0.06	0.21 ± 0.10	0.22 ± 0.10
	DIABETES	MACHINE	ABALONE
EXTRA-TREES	0.60 ± 0.12	0.21 ± 0.13	0.46 ± 0.03
RANDOM FOREST	0.55	0.13	0.44
DECISION TREE	0.67	0.39	0.56
RULEFIT	0.55	0.26	0.46
NODE HARVEST	0.58	0.29	0.61
SIRUS-R	0.56	0.29	0.66
GIF	0.58 ± 0.06	0.23 ± 0.17	0.60 ± 0.03

TABLE 9.5: Relationship between the node budget and the number of extracted rules.

DATASET	BUDGET	NUM. OF RULES
OZONE	22	13.6 ± 0.9
MPG	18	10.9 ± 0.5
HOUSING	12	7.3 ± 0.5
DIABETES	24	12.9 ± 0.7
HARDWARE	18	10.4 ± 0.5
ABALONE	12	8.3 ± 0.6

Intrinsic or extrinsic? So far the discussion has mainly been focused on GIFs as an intrinsically interpretable method (disregarding the straightforward extraction step). Being a pruning method, GIF can also be seen as an extrinsic method by pruning a larger forest before extracting the rules. The extra-trees line of Table 9.4 shows the unexplained variance of the unpruned forest corresponding to GIF: the extracted rules are the same.

9.2.3 A further digression about stability

SIRUS-R was motivated by the idea of rule stability: learning the model on different samples should yield similar rules. The authors measure rule stability by counting (and then normalizing) the number of exactly equivalent conditions which are obtained by cross-validation. To ensure that some splits actually appear several times, they restrict the random forest to split on the $q = 10$ quantiles of the splitting variable. This is also an important component of the method.

Owing to the authors' definition of descriptive stability, GIFs would fare badly in that regard. Indeed, extra-trees-based GIFs choose the splitting value at random, resulting in finding *exactly the same* condition twice unlikely. However, it is not expected that small shifts in splitting values would have an overall tremendous impact. At least not more than quantizing the input values. A more flexible definition of stability would be needed—provided it is a good endeavor in the first place.

For one thing, interpreting phenomena with small rule sets might not be consistent with descriptive stability. In the face of redundant variables, the size constraint will prevent giving a full account of the underlying interactions between variables. Different runs would then result in highlighting different variables. Consequently, the problem might be unstable by nature and decreasing accuracy in favor of stability might be altogether harmful.

The other problem with stability relates to how it can be achieved. If it goes through increasing the bias (as the q-quantile splits do), it might further restrict the class of problems on which rule sets may be applied. Indeed, the bias of rule sets is already high (on complex problems) since the model is severely constrained in size.

9.2.4 Conclusion

Re-contextualizing GIFs within the question of interpretability has shed some interesting results: GIFs also perform reasonably well in this setting. The small experimental study also gave insights as to how the GIF hyper-parameters should be set to obtain accurate-yet-intelligible rule sets.

A more thorough and systematic study would be needed to back these first results. The first step would be to see how GIF-based rule sets fare on classification tasks, especially in multi-class settings where GIFs have been shown to underperform. A fairer comparison with existing methods is also called for. Bénard et al. (2021) did not discuss in detail the hyper-parameters of the baselines, which might influence the conclusion.

A final question is how GIFs would handle irrelevant variables—a problem central to the next section.

9.3 Feature importances

Section overview

This section reviews and compares techniques for feature importances for decision trees and neural networks, which were the main models examined in this thesis.

This section is based on the publication "Nets versus trees for feature ranking and gene network inference" (Vecoven et al., 2020). My contribution to this work was to supervise N. Vecoven's master thesis on which the paper is based, discuss the deep learning methods, the methodology and the experimental protocol, as well as help write the paper.

The code relating to this contribution is available at `https://github.com/nvecoven/ann_fsl`. It is implemented by Nicolas Vecoven on top of TensorFlow 2.0 (Abadi et al., 2015) and Scikit-Learn (Pedregosa *et al.*, 2011).

Section 9.3.1.1 briefly sum up the contributions. Section 9.3.1.2 discusses the motivation behind the specific kind of interpretability measures this whole section investigates, while Section 9.3.2 discusses several closely related problems. Section 9.3.3 and Section 9.3.4 detail the importance measures which will be investigated for decision forests and neural networks, respectively. An empirical analysis on benchmark problems is conducted in Section 9.3.5, while Section 9.3.6 turns to the problem of Gene Regulatory Networks wherein feature importance measures are used to infer which genes regulate which.

The present section is mainly faithful to the original article. Sections 9.3.1.2 and 9.3.2 provide a bit more context, and notations have been uniformized.

9.3.1 Ambitions

9.3.1.1 Goal and contribution

The goal of this contribution was to analyze how embedded feature importance measures for artificial neural networks (ANN) compared to decision forest-based approaches, usually held as (among the) state-of-the-art methods. Our driving problem was the hard biological case of gene regulatory networks (detailed in Section 9.3.6).

Contribution. Our contribution was to

- propose a framework to analyze local feature importance measures in a quantitative way through the problem of feature selection;

- empirically compare the performances of feature importance measures for ANN and random forest (RF) on benchmark problems, which delivered a complex conclusion;
- combine the so-called selection layers (see Section 9.3.4.3) with other feature importance measures, which turn out to (i) yield a great improvement for feature importance, and (ii) be a good regularizer in the context of irrelevant/redundant variables;
- assess the neural network feature importance measures in the context of a challenging task in computational biology, namely the inference of gene regulatory networks (where RF are currently amongst the state-of-the-art approaches).

9.3.1.2 Motivation

What would happen if we included irrelevant input variables in a learning set? For instance, suppose the ozone dataset of the previous section had another variable which consists in independently generated Gaussian noise. Would this improve the prediction? Hardly. Actually, it might well hurt learning by providing a lever for overfitting. If there are relatively few instances compared with the expressiveness of the hypothesis space, the learning algorithm could easily pick up on what appear as structures in the irrelevant variables due to the finiteness of the data (see Figure 9.2).

Arguably, adding independently generated features to a dataset makes little sense but there are many reasons why the data might contain features which have no actual predictive power. For instance, it is not always easy *a priori* when engineering the features to decide which ones to include. Some features might seem useful but really are not.

The converse is true as well. Imagine the ozone dataset included a feature about the average mood of the citizen. At first glance, it might seem irrelevant to the ozone concentration but people's moods tend to be influenced by the weather conditions and smog will definitely turn a day gloomy. It might not be as informative as other features but it might still help in the prediction process. Is it worth the cost of monitoring such a variable?

Even when the features are *a priori* assumed to be relevant, some might eventually turn out to be irrelevant. Still on the ozone case, imagine one piece of equipment was faulty, resulting in random measurements. This would constitute a situation where an irrelevant variable is present in the dataset, even though there were good reasons not to suspect it.

Alternatively, the whole purpose of learning a model might be to discover which variables really *are* important. See Section 9.3.6 for such an example. As disclaimed in Section 4.3, this must be understood in a predictive, rather than causal, sense. Coming back to the example of the ozone level and general mood, it is clear that, although the latter might help predict the former, the mood is not a *cause* of the ozone level. When the acquisition is costly, focusing on relevant variables might go a long way toward creating a large learning sample at an affordable cost.

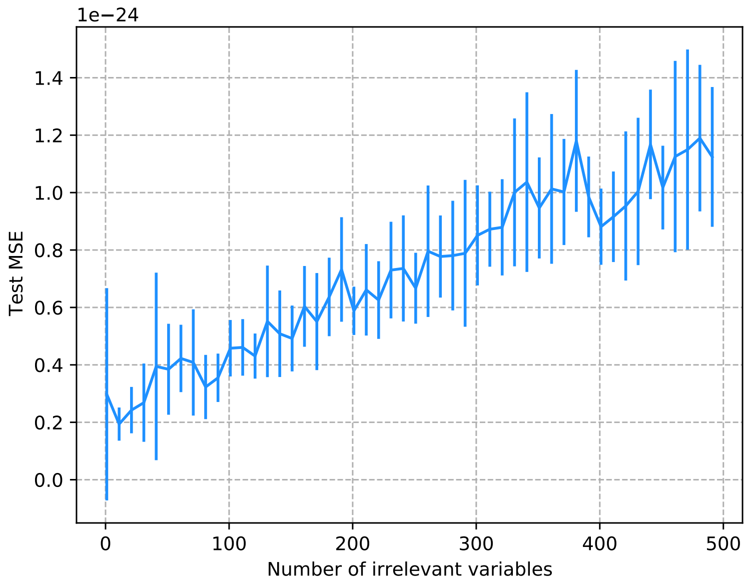


FIGURE 9.2: Degradation of the found hypothesis with respect to the number of irrelevant variables. The problem contains 25 relevant variables and is linear (see LC in Section 9.3.5.1 for the details). The learning algorithm is the logistic regression trained with 20000 datapoints. Even though the hypothesis space contains the true function, the performance of the model degrades substantially when the number of irrelevant variables increases.

Conversely, it is reassuring to see that a learning process is able to recover known characteristics, such as which features are truly important. This might be the missing ingredient for trusting an artificial agent.

Whatever the reason, machine learning benefits from being able to reflect back on which input variables are truly important in making predictions.

Optimality-relevancy agreement assumption. A word of caution. Even though it is intuitively compelling to believe there is a strong connection between optimality and the "relevance" of features, the relationship between those two is not trivial; the suitability of the hypothesis space must also be taken into account. Even in the (data-)asymptotic case and assuming no search bias, it is not true for all hypotheses space that adding an informative variable will result in a better model (see the "Optimal subset" paragraph below).

This should not be confused with the general comment on the tradeoff between accuracy and interpretability with respect to the expressiveness of the hypothesis space (Section 5.2.5). This is more related to the question of whether predictive accuracy can serve as a surrogate measure for descriptive accuracy (embodied here by the feature relevancy).

The conclusion is that "relevancy" (not just interpretability) and accuracy sometimes conflict with each other. In other words, enforcing the former might constitute a constraint for the latter.

Nonetheless, this does not mean that optimality and "relevancy" do not go along to some extent. A model would not be able to make an accurate prediction if it relied only on irrelevant variables. Therefore it is not ungrounded to use optimality as a proxy for "relevancy" (as for interpretability in general). The assumption that accuracy can be used instead of "relevancy" when making decisions will be referred to as the optimality-relevancy agreement assumption.

But what is "relevancy" exactly?

9.3.2 Problem formulation

9.3.2.1 Relevance

Let us first introduce the concept of relevant variables. Here we reproduce the definitions of Kohavi, John, et al. (1997) (in the form given by Sutera (2021)). As usual, let $\mathcal{X} = (\mathcal{X}_1, \ldots, \mathcal{X}_p)$ and $\mathcal{Y}$ be the input and output variables respectively. Let $\mathbb{V} = \{\mathcal{X}_j | 1 \leq j \leq p\}$ be the set of input variables[3]. Furthermore, let $\mathbb{V}_{\neg j} = \mathbb{V} \setminus \{\mathcal{X}_j\}$.

Definition 9.3.1 (Relevant variable)**.** *A variable $\mathcal{X}_j$ is relevant if and only if*

$$\exists \mathbb{B} \subseteq \mathbb{V}_{\neg j} \text{ such that } \mathcal{X}_j \not\perp\!\!\!\perp \mathcal{Y} | \mathbb{B} \tag{9.1}$$

where $\mathcal{X}_j \not\perp\!\!\!\perp \mathcal{Y} | \mathbb{B}$ means that variable $\mathcal{X}_j$ is not conditionally independent of $\mathcal{Y}$ given $\mathbb{B}$.

If a variable is not relevant, it is irrelevant.

Even if the conditional independence of two variables given some others could be easily evaluated, the existential condition on $\mathbb{B}$ might soon constitute a computational burden (there are $O(2^p)$ subsets to consider). One way out of this issue is to consider a stronger condition:

Definition 9.3.2 (Strongly relevant variable)**.** *A variable $\mathcal{X}_j$ is* strongly *relevant if and only if*

$$\mathcal{X}_j \not\perp\!\!\!\perp \mathcal{Y} | \mathbb{V}_{\neg j} \tag{9.2}$$

In words, $\mathcal{X}_j$ brings information to $\mathcal{Y}$ even in the presence of all the other variables.

Strong relevance might be too strong an assumption, however. Imagine that $\mathcal{Y} = f(\mathcal{X}_1)$ and $\mathcal{X}_1 = a\mathcal{X}_2 + b$, where f is a deterministic function and $p = 2$. Since $\mathcal{X}_1$ and $\mathcal{X}_2$ are isomorphic, they bring the same information to $\mathcal{Y}$. Yet, neither are strongly relevant even though the input-output relationship is deterministic. Such a case might seem academic but redundant variables

[3]Gathered in vectorial form elsewhere in this thesis.

are frequent in practice, especially in high-dimensional settings (think of the spatial redundancy in images). For this purpose, it makes sense to define the concept of marginal relevance.

Definition 9.3.3 (Marginally relevant variable)**.** *A variable $\mathcal{X}_j$ is* marginally *relevant if and only if*

$$\mathcal{X}_j \not\perp\!\!\!\perp \mathcal{Y} \tag{9.3}$$

In words, $\mathcal{X}_j$ is marginally relevant to $\mathcal{Y}$ if they are not independent.

It should be noted that a relevant variable is not necessarily marginally relevant: $\mathcal{X}_j$ could be independent of $\mathcal{Y}$ but provide information in the presence of a collider variable $\mathcal{X}_l$. Imagine that two coins are tossed, that $\mathcal{X}_j$ stands for whether the coin landed on tail, and that $\mathcal{Y}$ stands for the result of the second coin. Knowing the result of the first coin brings no information regarding the second on its own. If $\mathcal{X}_l$ indicates whether the two coins landed on the same side, knowledge of $\mathcal{X}_j$ and $\mathcal{X}_l$ fully determines the outcome of $\mathcal{Y}$.

Optimal subset. Finally, let us mention that Kohavi, John, et al. (1997) also discussed the notion of the optimal subset (the subset which allows minimizing the expected loss) and show with a couple of counter-examples that optimal subsets may not contain all relevant features—even not all strongly so. In particular, using a larger set of relevant variables to learn a hypothesis from might not amount to a better accuracy if no hypotheses can leverage the information of the additional feature. Additionally, irrelevant features, such as constants, might truly (*i.e.* without overfitting) help in predicting by improving the expressiveness of the hypothesis space.

9.3.2.2 Importance

Relevance is a rather coarse-grained measure. In some context, it might be preferable to have an idea of the degree of relevancy. For instance, it is reasonable to suspect that measuring the ozone level is a better predictor of the ozone level than whether or not the citizens are moody.

The most straightforward measure of the relevancy degree is surely the conditional mutual information. Since conditional independence is equivalent to having no conditional mutual information, an easy parallel can be drawn with the notions of relevance.

Assessing the mutual information might not be easy, however, as it would require getting a good estimate and possibly some combinatoric evaluation (*i.e.* Eq. 9.1). As such, it might be more convenient to work with surrogate measures which more or less relate to one of the concepts of relevancy described in the previous section. Such measures, henceforth called importance measures, are the topic of Sections 9.3.3 and 9.3.4.

9.3.2.3 Selection and ranking

A different, yet related, problem is feature selection: find a subset of variables which together bring some amount of information for the output.

A literal transcript of this, when we are looking for the most information, goes through the notions of Markov blankets and boundaries (Pearl, 1989).

Definition 9.3.4 (Markov blanket)**.** *A Markov blanket is a subset $\mathbb{B} \subseteq \mathbb{V}$*

$$\mathcal{Y} \perp\!\!\!\perp \mathbb{V} \setminus \mathbb{B} | \mathbb{B} \tag{9.4}$$

In words, a Markov blanket is a set of random variables which contains all the information about the output.

Definition 9.3.5 (Markov boundary)**.** *A Markov boundary is a Markov blanket of smallest cardinality.*

Markov boundaries (there might be more than one) relate to relevancy. Two interesting properties are that a Markov boundary must include all the strongly relevant variables and that no irrelevant variables can be included in the boundary (Tsamardinos and Aliferis, 2003).

From importance to ranking to selection. Even though the Markov boundary is the smallest blanket, it might still be a large set. Actually, it might well include all the variables. The remark we made about relevancy still holds, however: not all variables contribute in the same way. Actually, a Markov boundary might include many features which barely bring information.

To go past this issue, a general solution is to tackle feature selection via a ranking of the variables, most usually established via an importance measure. It goes like this: each variable receives a score, thanks to which a ranking is established. In the end, only the top k variables are kept. Provided the importance measure fulfills some additive property, k can be chosen to reflect some minimum total information threshold.

Although this places a bottom-up view on a pedestal, it should be noted that many methods actually compute the importance score in a top-down fashion by attributing to each variable some part of the total importance available.

From local to global methods. Although feature selection is a global problem (the selection is made irrespective of a given input), measuring the importance of a feature or ranking them might either be done at a global scale or at a more local one: which feature is important for predicting the output to input x?

The local information can be aggregated over a set of examples to lead to a global measure. The simplest scheme, the one which we will use, is simply to average the scores. This aggregation step renders the link between the final importance/ranking and the notions of relevancy and Markov blanket more difficult to establish.

9.3.3 Feature importance with random forests

A classical importance measure for random forests is the mean decrease of impurity (MDI) (Breiman, 2002). Recall from Section 3.3 that a decision tree propagates an example x down the tree towards a leaf by following a succession of splits $\tau(x; j, t)$ where j is a feature index and t is the threshold value on which the decision (passing through the left or right child) is made. While building the tree, the splits τ are chosen so that the uncertainty reduction $\Delta_\tau U(\mathbb{S})$ will be largest.

Definition 9.3.6 (Mean Decrease of Impurity (MDI))**.** *The MDI score for variable j computed from a m-trees forest is*

$$MDI(j; \mathbb{S}) = \frac{1}{m} \sum_{l=1}^{m} \sum_{(k,\tau(\cdot;j',t)) \in \mathbb{T}_l | j'=j} \frac{|\mathbb{S}_{l,k}|}{|\mathbb{S}|} \Delta_\tau U(\mathbb{S}_{l,k}) \tag{9.5}$$

where $\mathbb{T}_l$ is the set of pairs (node index, splits) the lth tree and $\mathbb{S}_{l,k} \subseteq \mathbb{S}$ is the subset of training instances reaching node k in the lth tree. Typically $\mathbb{S}$ is the learning set.

In words, the MDI score is the total weighted reduction of uncertainty (aka. impurity) brought by a feature j, averaged over all the trees constituting the forest.

Note that MDI can be used with any uncertainty measure U. In the case where U is the Shannon entropy, Louppe et al. (2013) proved that this score relates, in the asymptotic setting and with totally randomized trees, to the mutual information between variable j and the output, averaged over all conditionings. This is a compelling argument in favor of the MDI score.

9.3.4 Feature importance with neural networks

With great popularity comes great responsibilities. The advent of deep learning has nudged practitioners in embracing such models in many areas but one thing cannot be said about deep networks: that they are intrinsically interpretable. Consequently, the past few years have witnessed the proposition of many embedded methods to compute importance measures for input features.

These approaches usually provide *local* importance scores, measuring the relevance of each input feature for a given individual prediction. They can be broadly divided into two families: gradient-based methods (*e.g.* Simonyan, Vedaldi, and Zisserman, 2014; Baehrens et al., 2010), which compute the gradient of the output with respect to the input, and decomposition-based methods (*e.g.* Bach et al., 2015; Sundararajan, Taly, and Yan, 2017; Shrikumar, Greenside, and Kundaje, 2017), which decompose the output prediction (or the difference with respect to a baseline) into a sum of contributions from the different input features. Both gradient-based and decomposition-based methods are backpropagation approaches that propagate the importance signal from an output neuron to the input neurons through each layer of the network.

Since our goal is to study how ANN-based importance measures compare to those derived from decision forests (rather than comparing the existing

ANN-based measures among them) only one representative method from each family will be examined. For a detailed discussion and comparison of the different existing approaches for ANN, the reader can refer to Montavon, Samek, and Müller (2018), Ancona et al. (2018), and Kindermans et al. (2019).

As representative of the gradient-based methods, we chose to backpropagate the partial derivative of the absolute value of the derivative of the output with respect to each input feature (called GI in the following) (Leray and Gallinari, 1999; Simonyan, Vedaldi, and Zisserman, 2014). This will be discussed at more length in Section 9.3.4.1.

As representative of decomposition-based approaches, we chose the layer-wise relevance propagation (LI) technique (Bach et al., 2015). This will be discussed at more length in Section 9.3.4.2.

Both GI and LI can be used with any pre-trained network with an arbitrary feed-forward structure. As such they might provide useful insight with more structured inputs than those which we will investigate.

Another approach—the so-called selection layer method—which cannot be used directly with a trained model is the topic of Section 9.3.4.3 which is followed by a short discussion about hybrid approaches (Section 9.3.4.4).

On architecture and scope. For fair comparison, we wanted to focus on problems on which no learning algorithm was expected to perform better. Therefore, we focused on standard machine learning (rather than images) and turned to fully-connected feed-forward neural networks. The selected methods (GI and LI) are not restricted to those architectures, however. For instance, (some of the variants of) these methods have been previously discussed in the context of image classification, where they were shown to be able to identify the pixels that are useful for classifying a given image (Simonyan, Vedaldi, and Zisserman, 2014; Bach et al., 2015). It should also be noted that a third category of methods exists for image processing (*e.g.* Zeiler and Fergus, 2014; Springenberg et al., 2014; Kindermans et al., 2017). They work by identifying the input pattern which activates the neurons in the different layers for visualization purposes.

Since ReLU-based networks are ubiquitous nowadays, we will focus on them.

The architectures we will look at are thus of the form

$$\hat{y}(\cdot, \Theta) = f_L(\cdot; \theta_L) \circ \ldots \circ f_1(\cdot; \theta_1) : \mathbb{X} \to \mathbb{R}^K \tag{9.6}$$

where

$$f_l(x; \theta_l) = ReLU\left(W_l x + b_l\right) \tag{9.7}$$

with $\theta_l = (W_l, b_l)$. We conform to the notations of Section 3.6.1.

9.3.4.1 Gradient-based method (GI)

A standard importance measure for variable j for an input x is given by the absolute (or squared) value of the partial derivative for variable j at x (Leray and Gallinari, 1999; Simonyan, Vedaldi, and Zisserman, 2014). This importance score measures how much the network output for the sample x changes regarding an infinitesimal change in $x^{(j)}$, and can be efficiently computed using backpropagation. To obtain a global importance score, we extend this approach by simply taking the sum of the derivatives over all the instances $\mathbb{U} = \{x_i \sim \mathcal{X}\}_{i=1}^{n}$ and denote it by GI, which stands for Gradient Importance.

Definition 9.3.7 (Sensitivity (gradient importance) GI)**.**

$$GI\,(j;\mathbb{U}) = \sum_{x\in\mathbb{U}} \sum_{k=1}^{K} \left| \frac{\partial \hat{y}^{(k)}(x)}{\partial x^{(j)}} \right|, \tag{9.8}$$

$GI(j)$ is simply the sum over all instances of the $L1$ norm of the jth row vectors of the end-to-end Jacobian matrix.

With linear-and-rectified architectures, the partial derivatives come down to

$$\frac{\partial \hat{y}^{(k)}(x)}{\partial x^{(j)}} = \sum_{P\in\mathbb{G}_j(x)} \prod_{(l,i,o)\in P} W_l^{(o,i)} \tag{9.9}$$

where $\mathbb{G}_j(x)$ is the set of all non-blocked paths from the jth input neuron to the kth output for the example x and $P = ((1, i_1, o_1), \ldots (L-1, i_{L-1}, o_{L-1}))$ is a path (a $(L-1)$-tuple of triplets: the layer index, the index of the input neuron and the index of the output neuron). A path is said to be blocked if it passes through an inactive neuron (*i.e.* a ReLU neuron that has a null or negative input). In this view, the role of x is merely to indicate which paths to block.

9.3.4.2 Layer-wise relevance propagation (LI)

Although commonly used, the gradient method has the drawback that it does not explain the output of the network but rather how the output varies when the input is changed (Montavon, Samek, and Müller, 2018). Clearly, an input could be relevant even if Eq. 9.8 is (close to) zero at a given point. Moreover, linear-and-rectified architectures form piece-wise linear regions. Therefore, a gradient-related measure might not be quite so useful as knowing which variables entail more discontinuities. Finally, a model suffering from vanishing gradient would provide little information with such a method.

Several alternatives have been proposed to circumvent these limitations. As a representative of these methods, we use below a particular instance of the generic layer-wise relevance propagation (LRP) method proposed by Bach et al. (2015).

Recall that $z_{l+1}(x) = ReLU(W_{l+1}z_l(x) + b_{l+1})$. For simplicity, let us define the product of the element at position (j,k) of W_{l+1} and of $z_l^{(k)}$ by $u_{jk}^{(l+1)}$ so that $z_{l+1}^{(j)} = ReLU\left(\sum_k^{p_{l+1}} u_{jk}^{(l+1)} + b_l^{(j+1)}\right)$.

Definition 9.3.8 (Layer-wise Relevance)**.** *The layer-wise relevance $LR_l^{(j)}(x)$ of the jth neuron of the lth layer of network at x is*

$$LR_l^{(j)}(x) = \begin{cases} \left|z_L^{(j)}\right| & \text{if } l = L \\ \sum_{k=1}^{p_{l+1}} \frac{ReLU\left(u_{kj}^{(l+1)}\right)}{\sum_{m=1}^{p_{l+1}} ReLU\left(u_{km}^{(l+1)}\right)} LR_{l+1}^{(k)} & \text{otherwise} \end{cases} \tag{9.10}$$

This propagation rule corresponds to the $\alpha\beta$-LRP rule with $\alpha = 1$ and $\beta = 0$ (Bach et al., 2015).

In words, the layer-wise relevance propagation consists in distributing the relevance of the $l+1$th layer to the lth layer proportionally to the activations (numerator). The denominator serves to rescale the relevance so all the relevance signal is backpropagated to the input (conservation):

$$\sum_{j=1}^{p_1} LR_l^{(j)} = \sum_{j=1}^{p_L} LR_k^{(j)} \qquad \forall j, k \tag{9.11}$$

In the case of ReLU activations, it can be shown that applying this rule at a given layer can be viewed as a Taylor decomposition of the importance at that layer onto the lower layer (Montavon, Samek, and Müller, 2018).

Definition 9.3.9 (Layer-wise Importance LI)**.** *The importance $LI(j)$ of variable j computed over $\mathbb{U}$ is*

$$LI(j; \mathbb{U}) = \sum_{x \in \mathbb{U}} LR_0^{(j)}(x) \tag{9.12}$$

Sensitivity versus LRP. The gradient importance and the layer-wise importance follow a similar scheme of backpropagating through the network some importance score. They differ on several accounts. The sensitivity requires to compute the gradient

$$\nabla_x \hat{y}(x) = \left(\prod_{l=1}^{L} J_l^T(x) \right) \mathbf{1}_K \tag{9.13}$$

where J_l is the Jacobian matrix of the lth layer and $\mathbf{1}_K$ is the K-dimensional vector containing all ones.

In the case of a succession of fully-connected layers and ReLU, the Jacobians of two consecutive layers come down to

$$W_l^T \, B_{l+1}(x) \tag{9.14}$$

where W_l is the weight matrix of the lth layer and $B_{l+1}(x)$ is a binary matrix (due to the ReLU) reflecting which of the neurons were active (in the forward pass). Rewriting the full backpropagation, we end up with

$$\nabla_x \hat{y}(x) = \left(\prod_{l=1; l=l+2}^{L} W_l^T \, B_{l+1}(x) \right) \mathbf{1}_K \tag{9.15}$$

On the other hand, LRP can be rewritten in matrix form as

$$LRP_0(x) = \left(\prod_{l=1; l=l+2}^{L} N_l(x) \left(U_l^T(x) \odot B'_{l+1}(x) \right) \right) |\hat{y}(x)| \tag{9.16}$$

where $\odot$ represents the Kronecker product, $N_l(x)$ accounts for the normalization (the denominator in Eq. 9.10) and $B'_{l+1}(x)$ is a binary mask. Note that $B \neq B'$ due the oddly missing bias term in LRP.

The first disagreement between LRP and the sensitivity is what is backpropagated. In LRP, the goal is to re-attribute the importance measure to the input variable, hence the absolute value at the end of the network. In the gradient-based approach, a dummy signal is backpropagated, and the absolute value (embodying the notion of measure) is only taken over the imputed scores on the input variables. Conceptually, there is thus a wide gap between the two methods.

The second disagreement is about how the backpropagations operate. Firstly, the ReLUs are not used in the same way. Secondly, LRP includes a normalization factor to implement relevance conservation. Thirdly, the U matrices (which contain the u_{kj} entries) depend on the network weights but also on the latent vectors. In essence, while GI uses the samples to identify the (in)active paths, LI looks at the activations.

9.3.4.3 Selection layers

This method is inspired by sparse linear regression (see Section 3.1.2.2) and was proposed by Li, Chen, and Wasserman (2015). As illustrated in Figure 9.3, a one-to-one connected layer with linear activations and no bias, called here *selection layer* (SL), is introduced between the inputs and the first hidden layer of the network. Like all the other weights of the network, the weights of SL are initialized with random values drawn from a truncated normal distribution with 0 mean and 0.1 standard deviation, and the network is trained while penalizing them to ensure that only useful information goes through the network.

This penalization can be achieved through an elastic net program (Section 3.1.2.2), where the overall loss function is of the following form.

Definition 9.3.10 (Selection Layer optimization)**.**

$$\min_{\Theta, w_{sl}} \sum_{i=1}^{n} \ell\left(\hat{y}(x_i; \Theta, w_s l), y_i\right) + \frac{\alpha_1}{p} \sum_{j=1}^{p} |w_{sl}^{(j)}| + \frac{\alpha_2}{p} \sum_{j=1}^{p} \left(w_{sl}^{(j)}\right)^2 \tag{9.17}$$

where ℓ is a traditional loss, $\alpha_1 \geq 0$ and $\alpha_2 \geq 0$ are hyper-parameters balancing the penalty terms.

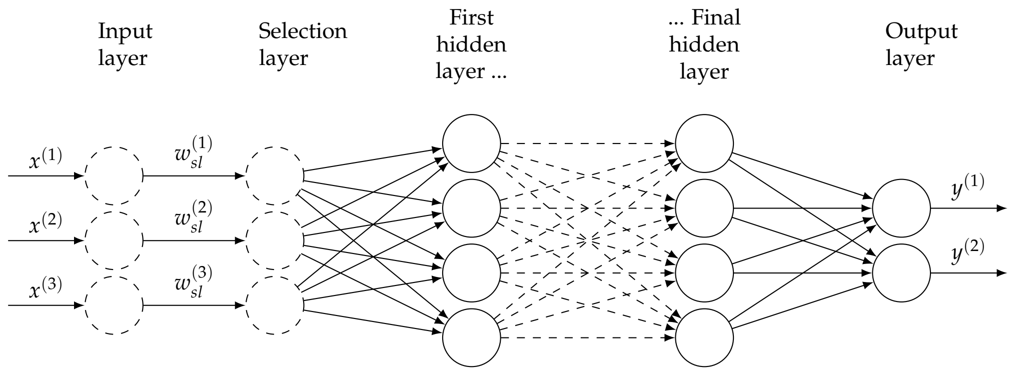


FIGURE 9.3: Example of selection layer architecture. The selection layer consists of a one-to-one connected layer between the input variables x_i and the first hidden layer of the network. Dashed (resp. plain) circles represent neurons with linear (resp. non-linear) activations.

This objective can be optimized as usual (*i.e.* with SGD). Unless otherwise mentioned, in our experiments we focus on a L1 penalty only, *i.e.* we set $\alpha_2 = 0$. Given this optimization program, a global importance score relating to the selection layer can be derived.

Definition 9.3.11 (Selection Layer Importance)**.** *The selection layer importance SL of feature j is*

$$SL(j) = \left| w_{sl}^{(j)} \right| \tag{9.18}$$

9.3.4.4 Hybrid approaches

Below, we also experiment with mixed strategies, called SL+GI and SL+LI, which train the network using the selection layer but compute the variable importances using the GI and LI techniques respectively. Since the selection layer is a one-to-one linear layer, this amounts to multiplying the importance score computed without the selection layer by the selection weight of the input feature. As we will see in Section 9.3.5, using a selection layer allows to strongly increase the performance on benchmark datasets.

When applied alone, GI and LI can be used with all trained architectures as extrinsic methods. Since they do not interact with the base objective, interpretation comes as a post-training constraint. The selection layer, on the other hand, changes the model which is learned. When used alone, the selection layer is intrinsically interpretable, since all it takes to get the information is a look at the selection weights.

Although MDI, GI, and LI all share the need for a set of instances, there is a large difference between them. In the case of MDI, the set must include labels. Moreover, MDI being a global measure, the sets are used quite differently. A consequence of this is that all the necessary information for computing MDI is traditionally kept within the tree structure. Keeping the information relative to GI or LI with the network seems somewhat less general since they are local methods.

9.3.5 Empirical analysis on benchmark datasets

We use datasets with a known ground-truth (*i.e.*. known relevant features) in order to assess and compare the five ANN-based approaches (GI, LI, SL, SL+GI, SL+LI) introduced in the previous sections. Since the relevant features are known, the variable rankings are assessed using the area under the precision-recall curve (AUPR). Given a threshold on the importance measure, the precision indicates the proportion of relevant variables captured by the method among all the variables declared as important, while the recall indicates the proportion of relevant variables captured by the method among all relevant variables. The AUPR aggregate the precision and recall over all possible thresholds. It will be equal to 1 if the ranking is perfect, *i.e.*, if all the relevant variables receive higher importance than the irrelevant ones, while the AUPR will be close to the proportion of relevant variables for a random ranking (see Section 3.7.1 for more details).

The baseline is the mean decrease impurity (MDI) score obtained from standard random forests (RF). RF models are composed of 1000 unpruned trees. They use the Gini index as uncertainty measure in classification and the variance in regression. The number p_e of examined features at each node of a decision tree is tuned from the set of values $\{\sqrt{p}, \log(p), p/3, p/2, p\}$ (where p is the number of inputs).

In all the experiments, unless otherwise stated, each ANN is composed of 3 hidden layers of respectively 300, 150, and 75 ReLU neurons, and is trained for 30000 steps on batches of size 50 using dropout (Srivastava et al., 2014) and AdamOptimiser (Kingma and Ba, 2015) with a learning rate of 10^{-3}. Values of the selection layer parameter α_1 are optimized in $\{10, 100, 1000\}$ on the validation set.

9.3.5.1 Simulated problems and protocol

We consider four different simulated problems.

Linear regression (LR). A linear regression problem generated using the *make_regression* function from the Scikit-Learn library (Pedregosa *et al.*, 2011). Output y is computed as $\sum_{i=j}^{25} w_j x^{(j)}$, where weights w_i are randomly and uniformly selected in $[0, 100]$, and inputs $x^{(j)}$ are $\mathcal{N}(0, 1)$ distributed.

Linear classification (LC). A linear, binary classification problem generated by thresholding the LR problem output so that the two classes are perfectly balanced.

TABLE 9.6: AUPR (in %) and Misclassification (MCR)/MSE for the four simulated problems, with 5000 variables in total in each problem. Values indicate means and standard deviations computed over 10 datasets. The best predictive results are indicated in bold. Coloring is linear.

	LC		NLC	
	MCR	AUPR	MCR	AUPR
SL+GI		90.20 ± 4.40		94.50 ± 2.60
SL+LI	**0.057±0.011**	88.10 ± 4.00	**0.049±0.007**	94.10 ± 2.50
SL		85.50 ± 4.80		89.60 ± 3.90
GI	0.364±0.006	73.00 ± 3.70	0.390±0.025	59.90 ± 7.40
LI		72.90 ± 3.80		60.40 ± 7.40
MDI	0.239±0.014	72.40 ± 3.80	0.186±0.021	99.60 ± 0.80

	LR		NLR	
	MSE	AUPR	MSE	AUPR
SL+GI		97.60 ± 2.60		86.00 ± 9.10
SL+LI	**0.007±0.003**	96.90 ± 2.80	**0.152±0.044**	86.00 ± 9.10
SL		96.70 ± 2.90		86.00 ± 9.10
GI	0.740±0.018	86.40 ± 6.40	0.862±0.010	80.00 ± 0.00
LI		86.00 ± 7.50		80.00 ± 0.00
MDI	0.618±0.018	81.50 ± 7.90	0.237±0.008	100.00 ± 0.00

Non-linear regression (NLR). A non-linear regression problem generated using the *make_friedman1* function from the Scikit-Learn library, which generates the following problem:

$$y(x) = 10sin(\pi x^{(1)}x^{(2)}) + 20(x^{(3)} - 0.5)^2 + 10x^{(4)} + 5x^{(5)} + 0.1\epsilon \qquad (9.19)$$

where ϵ is a $\mathcal{N}(0,1)$ noise and the inputs X_i are uniformly distributed in $[0,1]$.

Non-linear classification (NLC). A non-linear, binary classification problem generated using the *make_classification* function from the Scikit-Learn library with 25 relevant features. Briefly, one of the two classes is associated randomly to each vertex of a hypercube of dimension 25, and training examples of the corresponding class are generated in the neighborhood of each vertex by using a normal distribution centered on the vertex (with $\Sigma = I$).

Sizes and irrelevant variables. For each problem, we generate 10 datasets with 2000 training samples, 1000 validation samples, and 8000 test samples, and we add in each dataset a varying number of irrelevant features. These irrelevant features are generated using the same type of distribution as for the relevant features (i.e. $\mathcal{N}(0,1)$ for LR, LC, and NLC and $\mathcal{U}(0,1)$ for NLR).

9.3.5.2 Results and discussion

Table 9.6 reports the AUPR for all the methods on the four benchmark datasets with 4975 irrelevant variables for LC, LR, and NLC and 4995 for NLR (for a

TABLE 9.7: Misclassification rate (MCR) and AUPR (in %) results on the NLC problem with an increasing number of irrelevant features (from 25 to 9975 irrelevant features, in addition to the 25 relevant ones). Coloring is linear and global among all AUPR.

	50 FEATURES		2500 FEATURES	
	MCR	AUPR	MCR	AUPR
SL+GI	**0.039 ± 0.011**	99.80 ± 0.30	**0.051 ± 0.010**	96.00 ± 2.50
SL+LI		99.50 ± 0.50		95.50 ± 4.30
SL		98.60 ± 1.10		89.50 ± 0.00
GI	0.040 ± 0.005	99.90 ± 0.10	0.352 ± 0.025	59.50 ± 5.10
LI		100.00 ± 0.00		60.60 ± 5.20
MDI	0.094 ± 0.014	100.00 ± 0.00	0.171 ± 0.017	99.70 ± 0.50

	5000 FEATURES		10000 FEATURES	
	MCR	AUPR	MCR	AUPR
SL+GI	**0.049±0.007**	94.50 ± 2.60	**0.065 ± 0.016**	90.50 ± 5.20
SL+LI		94.10 ± 2.50		91.60 ± 5.20
SL		89.60 ± 3.90		89.70 ± 6.40
GI	0.390±0.025	59.90 ± 7.40	0.418 ± 0.027	60.30 ± 8.60
LI		60.40 ± 7.40		60.70 ± 8.50
MDI	0.186±0.021	99.60 ± 0.80	0.193 ± 0.037	98.80 ± 1.50

total of 5000 variables in each dataset). We also report the predictive performance of each model, *i.e.*, the misclassification rate (MCR) in classification and the mean squared error (MSE) in regression, computed on the independent test set.

The results clearly show the lack of robustness of standard neural networks (*i.e.*, without any selection layer) in the presence of a large number of irrelevant features. Without SL, ANNs are usually worse than RF along both MCR/MSE and AUPR, while adding SL allows to strongly increase the performance along both criteria in high-dimensional datasets. Compared to RF, ANN with SL yield higher performance in terms of MCR/MSE on all the problems, as well as higher performance in terms of AUPR on the linear problems (LC and LR). RF are better at highlighting the relevant variables on the non-linear problems, despite worse predictive performance. Among the three SL methods, SL+GI and SL+LI yield equivalent AUPR while SL returns inferior results, showing that the weights of SL are not enough to measure feature importances (see also Figure 9.4).

Table 9.7 shows the impact of the number of irrelevant variables on the NLC problem. It is clear that the MDI score of decision forests is much more stable with respect to the presence of irrelevant variables. Although the neural networks are more prone to being fooled, it should be noted they are able to withstand a few irrelevant variables. As far as the misclassification rate is concerned, we can see once more that the selection layer does a good job at preventing a radical drop in performance. In particular, with 25 irrelevant features, the networks with and without selection layer are comparable while there is one factor of magnitude with 10000 features. Interestingly, the misclassification rate of the networks with selection layers with 10000 features

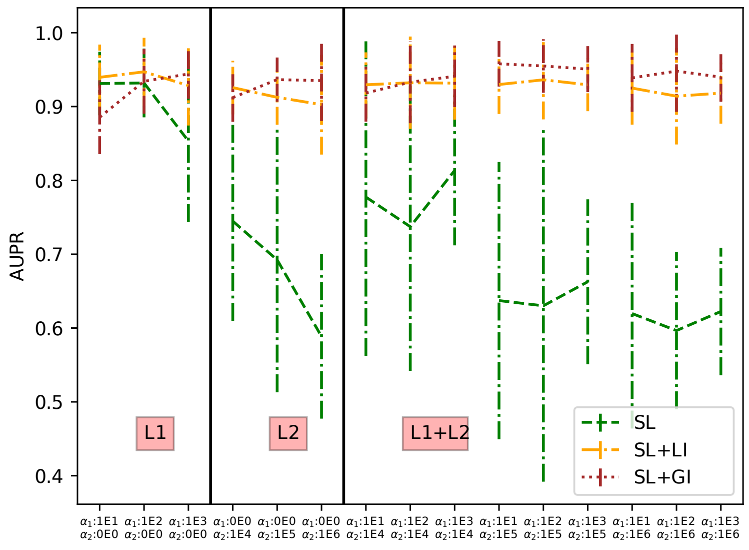


FIGURE 9.4: Impact of α_1 and α_2 for NLC with 4975 irrelevant variables. The figure plots the means and standard deviations of the AUPR over the ten different NLC datasets.

are better than the one of RF in the best scenario.

Overall, it is clear that the selection layer is a great benefit in the presence of irrelevant variables even though it might not constitute an optimal importance measure by itself. It is hard to provide a definitive answer as to which is better between GI and LI. The average edge GI has is not significant enough to conclude.

The results in Tables 9.6 and 9.7 were obtained using an L1 regularization (i.e., with α_2 in Equation 9.17 set to 0) on the weights of SL. Other regularization schemes could be used instead, such as an L2 regularization (with a corresponding regularization coefficient $\alpha_2 > 0$) or a combination of both. However, as shown in Figure 9.4, these other regularization schemes do not yield better results than L1 for SL+GI and SL+LI and return lower AUPR for SL.

The network architecture has also a great impact on the AUPR and MCR/MSE. For example, we observe in Figure 9.5 that networks with three or four hidden layers tend to yield the best results on the NLC problem (depending on the method). The figure also shows that although the MCR/MSE and the AUPR are not perfectly correlated, for a single dataset a lower MCR/MSE generally corresponds to a higher AUPR. This justifies the optimality-relevancy agreement assumption to some extent, which in turn justifies the use of MCR/MSE (on a validation set or by cross-validation) to tune the hyper-parameter values.

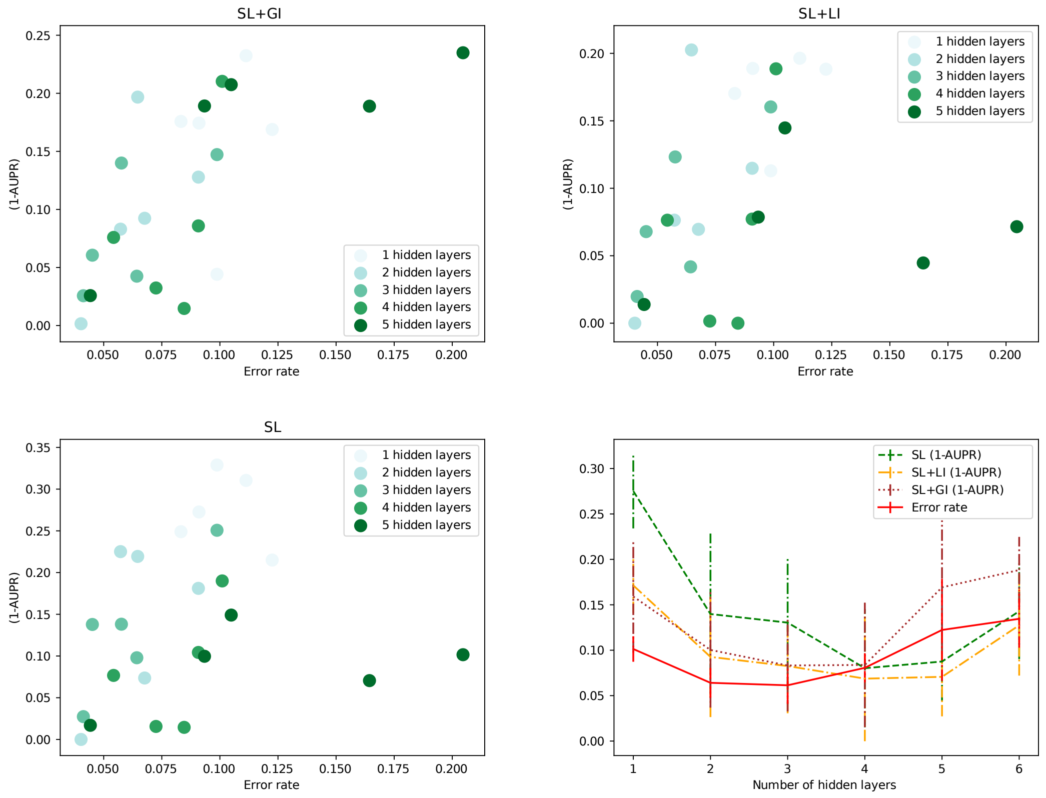


FIGURE 9.5: Impact of the number of hidden layers (150 neurons each) for the NLC dataset with 4975 irrelevant variables. Each scatter plot shows the performance metrics of one feature ranking method for five datasets (lower left is better). The bottom right figure shows the means and standard deviations over the five datasets.

9.3.6 Gene regulatory networks

9.3.6.1 Context

An open problem in computational biology is the reconstruction of gene regulatory networks (GRNs) from gene expression data. A GRN aims at explaining the joint variability in the expression levels of a group of genes through a directed graph. Each node of the graph represents a gene and an edge e_{ij} going from gene i to gene j indicates that i regulates the expression of j. We consider both ways of regulation: gene i can either increase the expression of gene j (activator) or decrease it (repressor). Often, the aim is to reconstruct a *weighted* network, where each edge is associated with the degree of confidence of the regulation. We focus on the latter. See the work of Mercatelli et al. (2020) for a review on GRN inference methods.

Gene expression level acquisition. Gene expression levels can be obtained in several ways. Here we consider multifactorial perturbation: "Multifactorial expression data are static steady-state measurements obtained by (slightly) perturbing all genes simultaneously. Multifactorial data might correspond for example to expression profiles obtained from different patients or biological replicates. " (Huynh-Thu et al., 2010). Such data are easier and cheaper to come by and hence more ubiquitous.

The data presents itself as a matrix E where the element at position (i, j) is the expression of gene j for the ith perturbation combination. From there, the goal is to infer to the regulatory network.

Inferring the gene regulatory network. One approach to infer weighted GRNs consists in solving one regression problem for each gene j in turn, with the expression of j as output variable and the expressions of the other genes as input variables. The variable importance score of gene i in the model predicting the expression of gene j is then used as weight for the edge e_{ij}. Using this framework, random forests are currently one of the state-of-the-art approaches for GRN inference (Huynh-Thu et al., 2010).

9.3.6.2 Empirical analysis

We use the ANN-based variable importance scores to reconstruct the five artificial networks of the DREAM4 multifactorial challenge and the real *Escherichia coli* network (Marbach et al., 2009) used in the DREAM5 challenge (Marbach et al., 2012).

Each DREAM4 network is composed of 100 genes, for which the simulated expressions in 100 samples are available. The *E. coli* dataset contains the expression levels of 4511 genes in 805 experimental conditions. For this dataset, we focus on 334 genes which are known to be transcription factors.

A gold standard network is available for each dataset, allowing the evaluation of a predicted ranking of edges in the form of an AUPR. Note that while the DREAM4 gold standard networks are the true (artificial) networks, the *E. coli* gold standard was built from experimentally confirmed interactions and is thus not perfect.

Protocol. Since one model must be learned per gene, fully tuning all the hyper-parameters per traditional cross-validation is computationally demanding. For the DREAM4 networks, we used the following cross-validation scheme. The genes were divided into five groups $G_1, \ldots, G_5$ and each group was associated with a fold of data $S_1, \ldots, S_5$. For each gene in subset G_i, the ANN was trained on the learning set composed of the four subsets $S_{j \neq i}$ and its MSE was evaluated using subset S_i. The MSE was used to choose the best hyper-parameters (optimality-relevancy agreement assumption) and a model was re-trained on all the data according to the selected hyper-parameters, which was then used to derive the feature importance score.

The number of hidden layers was optimized in {2,3}, the number of neurons per layer was optimized in {50,150} and the value of α_1 was optimized in {0,5,60,300,800,1500}. Each network was trained for 10000 steps on batches of size 35, with a learning rate of 10^{-4}.

The parameter p_e of RF was optimized in $\{\sqrt{p}, \log(p), p/3, p/2\}$, where p is the number of inputs (99 for DREAM4).

TABLE 9.8: AUPR (in %) for the five DREAM4 networks. The first row corresponds to the AUPR of a random ranking (not taken into account in the coloring). The three following rows indicate the results obtained when using a fixed ANN architecture (three layers of 75, 50 and 25 neurons respectively) and setting the regularization parameter $\alpha_1 = 10$. rows 5-8 indicate the results when α_1 and the architectures were optimized by cross-validation.

HP		NET 1	NET 2	NET 3	NET 4	NET 5
	RANDOM	1.8	2.5	2.0	2.1	2.0
FIXED	SL+GI	12.70	7.90	14.70	14.10	11.60
	SL+LI	14.80	9.50	16.90	15.50	14.20
	SL	13.70	8.70	14.50	13.40	11.40
TUNED	SL+GI	14.80	10.90	17.80	18.40	18.70
	SL+LI	14.30	10.10	19.30	17.20	18.00
	SL	12.60	12.10	19.10	19.20	16.60
	MDI	17.10	15.60	26.20	24.00	23.10

For the *E. coli* network, even this scheme was too computationally expensive. Therefore only one architecture is investigated.

Since the selection layer was crucial in providing good performance in terms of error and feature importance, we only investigate the SL, SL+GI, SL+FI methods.

Results. Table 9.8 shows the AUPR for the five DREAM4 networks. Results for ANN are shown for a fixed architecture and parameter α_1 (rows 2-4 of Table 9.8) and when both of them are tuned (rows 5-7). As can be seen, the cross-validation procedure allows improving the AUPR. The random forests appear nonetheless better on this problem, even though the ANN-based measures are far from random.

Figure 9.6 shows the result with a fixed architecture for the *E. coli* network. For this architecture and problem, we can see that SL+GI and SL+LI are competitive with the MDI. Once more SL is not good on its own.

9.3.7 Conclusion

In this section, we evaluated several feature ranking techniques based on artificial neural networks (ANN) and compared them on several problems to random forests (RF), chosen as a state-of-the-art reference. While the ANN importance measures can yield performances similar to the RF measures, they remain outperformed by RF on most problems we studied, despite having significantly better predictive performances.

Importantly, for datasets with a large number of irrelevant features, reaching good performance, both in terms of feature ranking and generalization error, comes at the cost of introducing a so-called selection layer within the

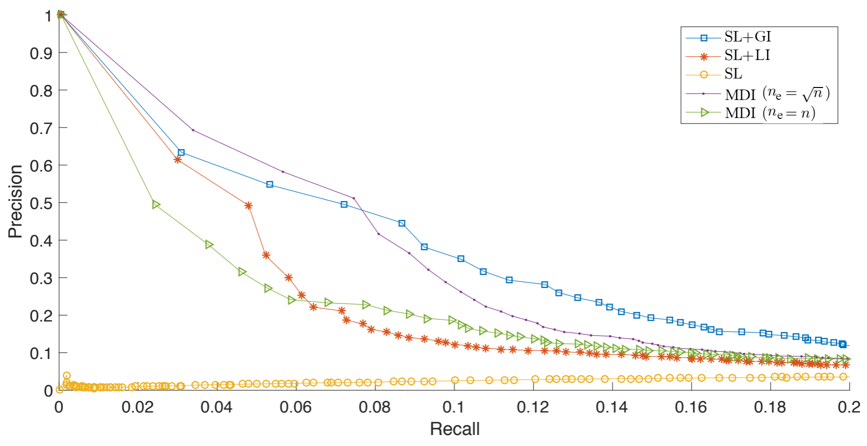


FIGURE 9.6: Precision-recall curves obtained for the *E. coli* network, when using an ANN architecture with 4 layers of 100 neurons each and $\alpha_1 = 10$. The RF performance is shown for two values of the parameter K ($p_e = \sqrt{n_{TF}}$ and $p_e = n_{TF}$, where $n_{TF} = 334$ is the number of transcription factors).

neural network architecture. This implies that interpretability must be anticipated and entails a great computational addition since the performance of the network is sensitive to the hyper-parameter relating to the selection layer.

Regarding the problem of gene network inference, ANN are competitive with RF on the real *E. coli* network, but are inferior on the artificial DREAM4 networks, even after an extensive tuning of the ANN architecture and regularization parameter.

The fact that the ANN approaches yield better predictive models than RF but not as good feature rankings suggests room for improvement in the latter area.

Another problem with ANN-based measures is the design question of the architecture. With unstructured data, deciding which architectures to investigate is a much more open question than in, say, image classification. This is even more true in the context of interpretability which imposes a further constraint on the choice of the architectures.

Since the selection layer is so important, it would be worth investigating whether it can be applied on an already-trained network with only a few fine-tuning steps. This would open the whole methodology for post-training use, possibly offering significant prediction boosts—provided some data is available.

Talking about data, it is hard to pass the question of what could be done regarding feature importance in a sample-free setting.

9.4 Conclusion

In this chapter, we looked at the problem of interpretability and how it interacts—mostly interferes—with traditional supervised learning. After discussing interpretability in some abstract and general way in Section 9.1, we delved into two ways of gleaning understanding from machine learning models.

Section 9.2 discussed the case of using GIFs (see Chapter 6) to build a list of rules, a global and intrinsic method of interpretability. A preliminary analysis showed that GIFs worked actually well for this purpose. A more in-depth study would be needed to draw definitive conclusions, however.

Section 9.3 discussed the problem of feature importance: how to score the input variables according to the information they bring to the output. Our goal was to assess how methods developed in the context of neural networks fared compared to state-of-the-art methods based on decision forests. In particular, we looked at the important biological application of gene regulatory networks (Section 9.3.6). The problem of feature importance also served to propose a quantitative way to assess local methods.

Overall, it was shown that regularization was a key aspect for neural networks to work well in the presence of irrelevant variables. The methods we investigated (with appropriate regularization) work reasonably well but are still outperformed by decision forest on most datasets, despite being much more capable of performing accurate predictions.

Conclusion

Part III

Chapter 10 Conclusion

This thesis has been about machine learning (Part I) and how its future goes through working with constraints (Part II).

10.1 Summary

After a short introduction (Chapter 1), we delved into supervised machine learning (ML) in Chapter 2. Supervised ML is a paradigm in Artificial Intelligence (AI) which consists in learning a hypothesis which best models a phenomenon from observations about it. Doing so naively usually results in a hypothesis which is only good on the observations used to select the hypothesis; a problem known as overfitting, and discussed at length.

Chapter 3 turned to supervised learning algorithms—how is learning a hypothesis done in practice? We reviewed the algorithms on which the latter chapters are based, notably decision forests and neural networks.

Part I ended with Chapter 4 which places supervised machine learning in the broader landscape of knowledge and how supervised machine learning (more precisely statistical learning theory) contributed back.

Part II opened on Chapter 5 which discusses the notion of constraints—the core of this thesis. A constraint is anything that stands in the way of supervised learning. We went through several examples of such constraints: training time, model size, robustness, data scarcity, and interpretability. Overcoming those constraints, or mix thereof, were the topic of the following chapters.

In Chapter 6, we turned to decision forests with the goal of exploiting all levers of compression (number of base models, depth of the base models, redundancy) to produce lightweight models, which we called GIFs. We developed an algorithm which could be flavored for either regression or classification and showed it worked well, especially in the former case. A subsequent study shed some light on how to set the hyper-parameters depending on the size of the model. Several improvements and research directions have been mentioned, the most prominent being the need to better handle multi-class prediction.

In Chapter 7, we looked at the problem of out-of-distribution (OOD) detection, a robustness issue where the goal is to detect when a model—typically a deep neural network—is fed inputs which do not come from the distribution on which it was trained. Robustness not being a new concern and OOD detection being quite broad, we spent some time refining our understanding of OOD detection and the related problems. Our contribution to this field was to investigate the challenging data-free setting, where assumptions about the learning process must replace knowledge otherwise harvested from data. In this regard, we developed a series of indicators based either on the optimality conditions or the so-called batch-normalization layers. We showed that different indicators were good at detecting different kinds of OOD objects. Since the indicators serve different purposes, we proposed a scheme to aggregate them which proved to be empirically stable. Overall, we were able to achieve impressive results given the severity of the setting—at least so long as the OOD samples do not come from a distribution too close to the base distribution. By the end of the chapter, several questions had naturally arisen, among which is the interrogation of whether it is possible to design an indicator relying on new principles (besides optimality and batch normalization layers).

In Chapter 8, we investigated the problem of compressing large neural networks. After discussing the various way in which this could be achieved, we proposed to leverage a collection of unlabeled samples, in which "relevant" samples are present, so as to achieve a fast knowledge transfer from a large network to a small one. It was shown to work well provided the collection *did* hold relevant samples. An important aspect of the method was to slightly bias the choice of samples from the collection in favor of those appearing most to come from the true distribution. We put to use the indicators proposed in Chapter 7 for this part. The second key aspect was to take as much as possible advantage of the learning signal. For that, we proposed an attention mechanism usable with any pair of trainer/trainee networks meeting some latent mapping assumption. This proved to be a crucial part in the absence of real training data. The most important part, however, is disposing of relevant data. Several research directions were proposed, mostly towards improving the two key aspects of our method: the biasing and the attention mechanisms.

In Chapter 9, we tackled interpretability: how can human-understandable knowledge be learned from complex high-dimensional mapping functions? After discussing the topic in some general fashion, we turned to two concrete problems.

Section 9.2 was dedicated to the problem of creating rule sets: short and directly interpretable models. For this, we re-used the GIFs of Chapter 6. A short experimental study showcased GIFs could actually serve that purpose quite well, although a more thorough and complete study is called for.

Section 9.3 was concerned with the problem of feature selection via feature importance measures. After some background exposition, we studied how neural networks fare in this regard compared to decision forests, held as state-of-the-art. It was shown neural networks had trouble dealing with irrelevant variables, although this could easily be remedied via some specific regularization mechanisms. With those, neural networks are able to equal decision forests on some problems but not all, despite being generally better at making accurate predictions. This contrasted situation carried to the important biological task of gene regulatory network inference. Needless to say, there is still room for improvement on this topic for neural networks.

10.2 Now what

This thesis tackled several tasks relating to constrained supervised learning and serves to illustrate, if it ever were necessary, that, even under those constraints, learning is doable.

The problems we tackled can hardly be deemed solved, however. In addition to the research directions we raised at the end of each chapter, there is definitely room to roam for improvement; the already good performances obtained are somewhat shadowed by the undismissable feeling that we are only at the tip of the iceberg.

This is especially true for data-free settings and interpretability. The former requires other sources of knowledge to browse from, which might ultimately lean towards the border of inductive learning. On the other hand, interpretability is a sweet promise of sleeping knowledge, waiting to be bucketed out. The convergence here is no coincidence, of course. Working with constraints requires using knowledge efficiently whether it is in infusing a model of small capacity with as much information as possible, or dissecting a trained model for traces of knowledge.

Beyond the few cases we studied, many more constrained settings need closer examination. Here are a few other problems which came up during this thesis and for which we are not aware of a definitive solution.

Sample-free post-pruning of decision forests. The observations made in the context of neural network compression carry over to decision forests, where the need for compression might also arise once the training set is no longer available. The relatively simpler nature of decision forests and the availability of a sound theoretical framework (the bias-variance decomposition) might allow reaching good compression rates with minimal loss of accuracy even without additional data. Painsky and Rosset (2019) proposed a lossless compression scheme which does not require data. It would be interesting to see how this baseline can be beaten, especially when other forms of information are available.

Other convergences. A natural question is whether more links can be created between the different topics we have broached; especially where help can be leveraged to overcome the constraints. For instance, it seems reasonable to expect that interpretability can help for out-of-distribution (OOD) detection: OOD examples can be expected to lead to odd (local) explanations. Likewise, it is not unreasonable to expect that pruning networks would help in detecting OOD samples: a smaller network would have to focus more on the distribution and OOD samples might portray more abnormal behaviors. Variable importance score metrices could also be exploited to identify less relevant model parts, be it for trees or neural networks, and lead to new pruning ideas.

Besides these ideas, other convergences might be worth exploring to better arm the community against constraints.

Covariate shift robustness. In Chapter 7 we have said that a model should ideally be robust to covariate shifts and OOD detection should focus on semantic shifts (Section 7.2.1). Ensuring robustness to covariate shifts might possibly be done seamlessly with the appropriate data. As we have advocated throughout this thesis, however, reliability concerns and data might not coincide. Moreover, collecting the appropriate data might be time-consuming or otherwise costly.

Raw ideas to boost covariate resilience without exhaustive data include more sophisticated data augmentation techniques (this already exists to some extent), being able to transfer this kind of robustness between models, or enforce it *a priori* or *a posteriori* with out-of-distribution data.

Mixing global and local interpretability. As we have said in Section 9.1, global and local interpretability interplay: global interpretations form a baseline for local ones and local ones sometimes can be aggregated at a global scale. Another approach would be to merge global and local interpretations more seamlessly.

GIFs are well suited for this purpose: the top part of the forest can be used to give a general idea of the phenomenon being analyzed while the bottom part can serve to give more fine-grained and local information. As we have seen, hyper-parameters for extremely shallow forests and larger ones must be set differently (higher learning rate and larger candidate window for the former). By changing hyper-parameters mid-course, GIFs might be able to cover the whole spectrum of interpretation scales.

The no free lunch of interpretability. As mentioned at the end of Section 9.1.2, interpretability is plagued with a few issues which oddly lean in the direction of the no free lunch (NFL; *cf.* Section 4.1.2.2).

Firstly, there is a meta-induction problem regarding the assessment of interpretability extraction methods, especially in the case of global ones. The fact that a method works well on a few problems is no guarantee it will work on others when no regularity between problems can be assumed.

Secondly, interpretability goes better with simplicity. NFL on the other hand argues that there is no assumption-free reason to be biased toward simplicity. There are problems for which interpretability will be harder—or so it feels.

In either case, it would be interesting to see whether a more formal argument could be pieced together to better define the limits of interpretability.

Bridging the gap between decision forests and deep learning. Overall, this thesis has focused its attention on two main learning algorithms: decision forests and deep learning. Decision forests are fast(er) to learn, portray low(er) latency for inference, come with built-in and convincing post-hoc interpretability. Neural networks, provided appropriate inductive bias is available, keep excelling prediction-wise and produce rich latent spaces, useful for transfer or other tasks.

Would not it be nice to be able to take the best of both worlds—or at least choose, on a per-problem basis, what to take from each? A few works (*e.g.* Biau, Scornet, and Welbl, 2016; Wang, Aggarwal, and Liu, 2017; Tanno et al., 2019) have investigated this question under different angles. The ideas include transforming a forest into a network, distilling a network into a forest, or designing hierarchical architectures for neural networks. So far, however, no definitive answer has been reached.

It may well be that endowing trees with feature learning capabilities would slow down training and inference, increase the memory footprint and hamper interpretability, but who knows what hidden gems will be found when looking for a new gold standard.

Despite conflicting with the traditional goal of supervised learning, it might so happen that the growing pressure of constrained settings will shed the light needed to dissipate the lingering shadows.

A Appendix Clustertools

Chapter overview

In this chapter, we review Clustertools, a tool which has been developed alongside this thesis to run all the necessary experiments, automating many of the involved sub-tasks and taking advantage of the parallelizable nature of the work.
The goal is to provide some motivations and quickly discuss how Clustertools works internally. A more detailed documentation and a rich set of examples covering how to use it in practice are provided in the repository.

Clustertools is available as a Python package at https://github.com/jm-begon/clustertools. The additional tools for analysis are available as Python package at https://github.com/jm-begon/clustertools-analytics.

Section A.1 exposes the general problem Clustertools is trying to solve. Section A.1.2 then discusses other benefits of Clustertools and Section A.1.3 showcase the package. Sections A.2 and A.3 present the concepts involved in Clustertools, for the processing and data sides respectively. Section A.5 promptly concludes this chapter.

A.1 Speeding up scientific computing

Scientific computing usually involves intensive number-crunching tasks. Moreover, the computations must usually be run several times: several parameters must be tested and variance must be estimated. In the case of machine learning, one usually wants to analyze how hyper-parameters affect the algorithm and how stable learning is with respect to the sampling of the learning set. Since there are several (hyper-)parameters to assess and one usually wants to investigate most of all the combinations, the number of computations is usually combinatorially large.

On the whole, scientists are faced with huge computation loads. Fortunately, they have access to computing resources, in the form of clusters of computing nodes, as well. The availability of such nodes allows to parallelize the work, rather than run everything sequentially.

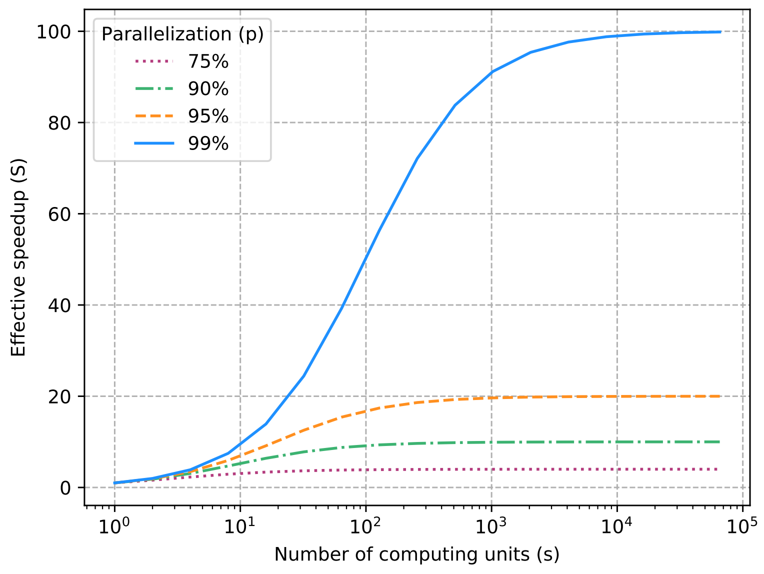


FIGURE A.1: Amdahl's law: the effective speedup S depends on the parallelization capacity s as well as the proportion of actually parallelized code p. Note the logarithmic abscissa.

If there are M processing units, the speedup is M right? Well, it depends.

A.1.1 Embarrassingly parallel code

There are many ways in which code can be parallelized: data parallelization (the same computation runs on different parts of data and the results are aggregated), task parallelization (different "subprocesses" run in parallel, doing different things), parallelization can also occur at lower (*i.e.* hardware parallelization) and higher (*i.e.* between unrelated computations) levels. Depending on the type of parallelization, the gain may vary. One way of formalizing this is through Amdahl's law (Rodgers, 1985).

Proposition A.1.1 (Amdahl's law)**.** *For a task whose latency (i.e. duration) is T and whose proportion of running time which is parallelizable is p, assuming constant workload, the speed up in latency is*

$$S(s) = \frac{T}{T(s)} = \frac{(1-p)T + pT}{(1-p)T + \frac{p}{s}T} = \frac{1}{1-p+\frac{p}{s}} \tag{A.1}$$

where s is the speed up due to parallelization. Note that

$$\lim_{s \to +\infty} S(s) = \frac{1}{1-p} \tag{A.2}$$

Figure A.1 shows the pessimistic truth behind Amdahl's law: the actual speedup is usually much less than could be expected at first. Moreover, the asymptotic nature of the law imposes a limit on the possible gain.

Amdahl's law might even happen to be more optimistic than reality. Parallelizing code usually entails costs absent in sequential computing, such as communication costs due to moving data around and synchronization latency, where some parts of the processing are stopped, waiting for another to complete.

A setting which escapes the pessimism of Amdahl's law is the so-called embarrassingly parallel code. In this setting, not only is the code easily to parallelize, but the *whole* running time can be parallelized (*i.e.* $p = 1$). Fortunately, this corresponds to the setting described above if parallelization is done at the level of (hyper-)parameters. That is how `Clustertools` proceeds.

A.1.2 Beyond parallelization

Motivated by the necessity to speed up computing, `Clustertools` brings many other advantages:

environment `Clustertools` allows to write code which runs seamlessly (*i.e.* without change) on personal computer or decentralized supercomputers (with `slurm` back-end, but easily extensible to other framework). No need for other scripts: everything can be done in pure Python and possibly in a single file. The environment is selected from the command line. A debug environment also exists to see what would be executed.

parameters `Clustertools` manages the combinatorial nature of the parameters to test. There are mechanisms to avoid some combinations and prioritize which ones to run first. Computation for each parameter tuples can be massively launched.

states A state system allows to monitor which tasks are done, running, pending or launchable. `Clustertools` will only launch computations which are launchable and information regarding computation time is saved.

analysis `Clustertools` has a mechanism to store results of the computations and then analyze them through an OLAP structure. A second package (`Clustertools-analytics`) offers additional functionality to create graphs and tables with coloring (such as the ones used in this manuscript; see *e.g.* Table 7.7).

command line utility `Clustertools` comes with a command-line utility which allows to easily monitor the state of computations, transfer experiments from one computer to another, see logs, and so on.

logging `Clustertools` comes with logging facilities (enabled with a simple command) which stores most of what is done. This is handy as a quick history.

A.1.3 Example

Code A.1 shows a simple one-file script using `Clustertools`. The computation part of the code goes into the `run` method, overloaded from the `Computation` class. In the `main` part, a command-line parser is instantiated and the list of parameters is used to create an `Experiment` instance. When running the script, the Cartesian product of all parameters will be used as parameters for the `run` method.

More detailed examples are given in the repository.

The following sections will delve into some more details about `Clustertools`. More precisely, Section A.2 will discuss the processing pipeline and the concepts involved, while Section A.3 will go over the data pipeline. Section A.4 will discuss the command line utility which ships with `Clustertools`.

A.2 The processing pipeline

In this section, the concepts involved in creating and running an experiment are introduced.

A.2.1 Computation

The `Computation` is the most fundamental concept in `Clustertools`. This class represents what needs to be computed. Once instantiated, it receives the parameter on which to run. Once done, it saves the results. Its stateful life-cycle is represented by Figure A.2.

The `State` instances which accompany the `Computation` object track the progress. From the `run` method, it is possible to update the stage at which the computation is. This allows deriving information such as the estimated time of arrival.

A.2.2 Experiment and parameters

In `Clustertools` terms, the `Experiment` represents all the computations which must be carried out. It is based on two concepts (Figure A.3). Firstly, a concrete instance of a `AbstractParameterSet` is responsible for generating the list of parameters that need to be supplied to a `Computation`. There are two base types of `AbstractParameterSet`: `ExplicitParameterSet` and `ParameterSet`. The former works like a container: it supplies back what is given. The latter generates the Cartesian product of all parameters. For the example of Code A.1, it would generate the tuples of parameters

```
(w=5, x=1, z=4)       (w=5, x=2, z=4)       (w=5, x=3, z=4)
(w=6, x=1, z=4)       (w=6, x=2, z=4)       (w=6, x=3, z=4).
```

The `AbstractParameterSet` is also responsible for making sure the code is reproducible by forcing a deterministic ordering of parameter tuples. In the case of an `ExplicitParameterSet` a simple FIFO policy is used to ensure this constraint. Consequently, parameter tuples can be added at a later stage without impact on the ordering.

`ParameterSet` uses a sorting mechanism to ensure consistency, similar to the `ParameterGrid` of the Scikit-learn library (Pedregosa *et al.*, 2011), which served as a based implementation. As it so happens, it is usually necessary

CODE A.1: example of `Clustertools` code

```python
from clustertools import Computation, CTParser, ParameterSet, \
    Experiment, set_stdout_logging

class MyComputation(Computation):
    def run(self, result, x, z, w, y=2, **parameters):
        result["multiply"] = x * y
        result["sum"] = z + w


if __name__ == "__main__":
    # Configure logging to debug on stdout
    # (the verbosity level can be adapted)
    set_stdout_logging()

    # Create the command line parser. It allows you to specify the backend,
    # as well as parameters related to the backend
    parser = CTParser()

    # Read from the command line and create an `Environment` to run the
    # code into
    environment, _ = parser.parse()

    # Define the parameter set: the domain each variable can take
    param_set = ParameterSet()
    param_set.add_parameters(x=[1, 2, 3], z=4, w=[5, 6])

    # Wrap it together as an experiment
    experiment = Experiment("BasicUsage", param_set, MyComputation)

    # Finally run the experiment
    environment.run(experiment)
```

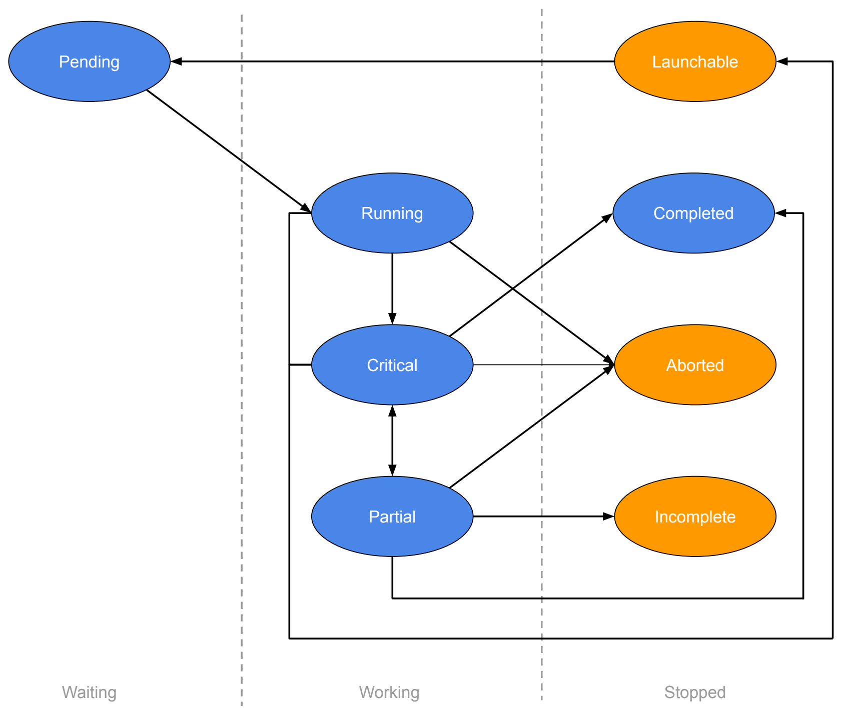


FIGURE A.2: State diagram for computations. Starting from launchable, a computation becomes pending, then running. It then enters the critical state when saving on disk. If it is the final or only writing, it goes to completed. Otherwise, it goes to the partial state and switches back and forth with critical until completion. If something goes wrong (*i.e.* `exception`) it goes to the aborted state. If it runs out of resources it goes to launchable, unless it was partial in which case it goes to incomplete.

to run a computation of more (values of the) parameters than originally envisioned. To avoid re-doing the previous computations yet maintain ordering consistency, a `add_separator` method is supplied to ensure a fixed ordering before the separator.

Given a concrete instance of a `AbstractParameterSet`, an `Experiment` object creates `Computations` thanks to a factory.

Constraints and prioritization. Not all computations are equally important. Sometimes, some values of parameters are examined for the sake of completeness, or a fine-grained idea of the variance is not mandatory at an earlier, more exploratory stage. On the other hand, not all combinations of parameters are worth investigating. For these, `Clustertools` offers the concepts of `PrioritizedParameterSet` and `ConstrainedParameterSet` (Figure A.4). They are implemented via the decorator pattern so that they can simply encapsulate another concrete instance of `AbstractParameterSet` and used directly instead. Code A.2 more specifically shows an example of how to use the `ConstrainedParameterSet`. More detailed examples can be found on the repository.

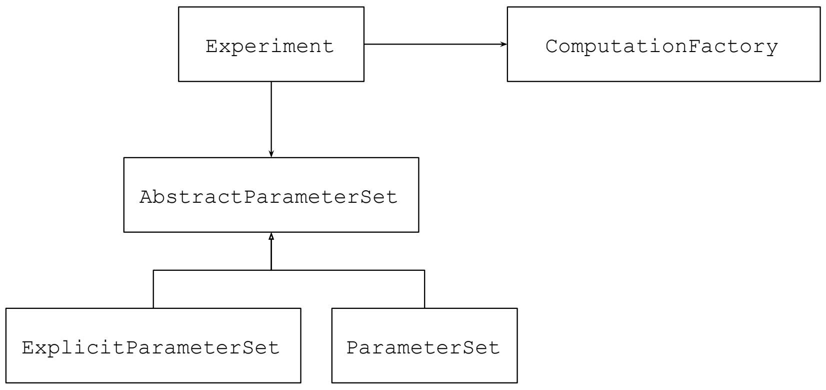


FIGURE A.3: An `Experiment` object uses a concrete instance of an `AbstractParameterSet` together with a `ComputationFactory` to instantiate `Computations`.

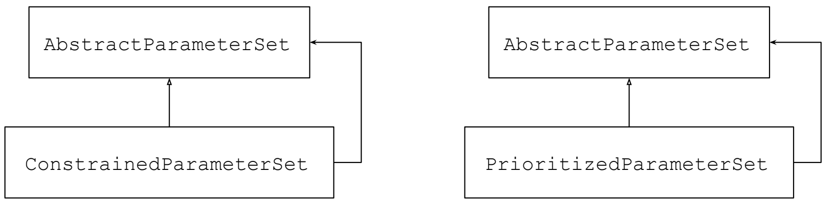


FIGURE A.4: Constrained and priorities are handled as decorators.

A.2.3 Environments: running experiments

In practice, the `Experiment` does not do much beyond naming the computations and articulating the aforementioned components. The object truly responsible for launching the computation is the `Environment`. Depending on the concrete class of the `Environment`, the computations can run sequentially in the current process, be submitted to bash (*i.e.* run concurrently as independent processes), or be submitted to a job scheduler (see Figure A.5). So far, only `slurm` is supported as scheduler but adding other schedulers amount to deriving other `Environments`. Along the same lines, an environment for *in situ* multi-processing using Python native capabilities can be implemented. Finally, there is a `DebugEnvironment` to see what would be run without actually launching any computations.

The parser. `Clustertools` provides a `CTParser` class whose job is to (i) select the `Environment` from the command line, (ii) supply environment-specific parameters (such as the duration or the memory requirement of the computations), and (iii) supply user-defined parameters (optional).

As such, the environment/back-end in which the computation is run is given from the command line. This allows to run the same code locally in

CODE A.2: example of using constraints over parameters in Clustertools

```
from clustertools import ConstrainedParameterSet
...
    # Define the parameter set: the domain each variable can take
    param_set = ParameterSet()
    param_set.add_parameters(x=[1, 2, 3], z=4, w=[5, 6])

        def not_x_eq_3_and_w_eq_6(x, w, **kwargs):
            # The predicate recieves the full parameter tuples and must
            # return False for all tuples which must not be run
            return not (x == 3 and w == 6)

    param_set = ConstrainedParameterSet(param_set)
    # We add the constrain
    param_set.add_constraints(not_x3_and_w6=not_x_eq_3_and_w_eq_6)

    experiment = Experiment('BasicUsage', param_set, MyComputation)
...
```

the current process for debugging purposes, or send the code running on computing clusters.

How is the code run. If the code is not run sequentially, the `Environment` instance uses the `Experiment` object to generate some of the computations (with the associated parameters). They are serialized and other processes are spawned (how depends on the actual implementation of the `Environment`) to deserialize and run the computations.

A slight drawback of the method is that it somewhat restricts the scope of the `Computations`. Concretely, package imports used in the `run` method must be made within the method, and using relative modules is impractical.

Code organization. In `Clustertools`, one of the most important design premises was to allow for one-file experiments (as shown with Code A.1). This conditioned the choice of the serialize/deserialize approach and the drawback we just mentioned. We felt, however, that it was a small price to pay for keeping all the code (computation and parameters) in a same place—an assuredly easier situation to manage, which only goes a small way in keeping a file system under control.

In some situations, keeping the parameters and computing code apart is sensible. When using other Python code, this is a simple matter of using the `run` method as a small stump for calling other code. Going through the command-line interface (because the called code is not Python or for modularity reasons) is, at the time of writing, a bit hackish. Better supports are provided (though not extensively tested) on the `dev` branch.

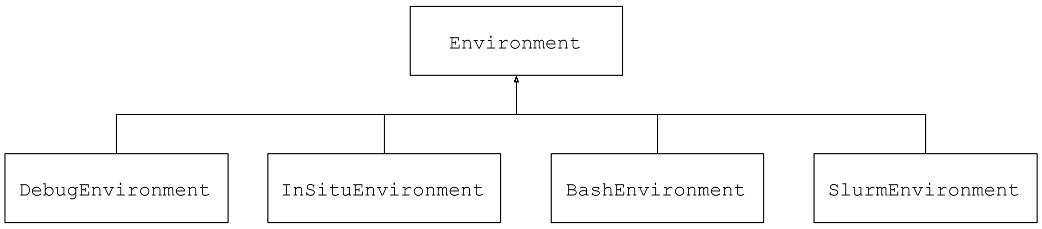


FIGURE A.5: `Environment` offered by `Clustertools`. `DebugEnvironment` only serves to show what would actually be run with another class instance. `InSituEnvironment` runs all the computation sequentially in the current process. `BashEnvironment` and `SlurmEnvironment` only create and serialize the computations. The former actually spawn a process per computation. The latter submits jobs to the scheduler. Other methods (parallelism with a pool of workers, or other schedulers) can be added by deriving other classes.

A.3 The data pipeline

The previous section gave an overview of the concepts involved in the processing pipeline and how they were articulated to produce and carry out the computations. In this section, we discuss how the results of all these computations are managed.

A.3.1 Result

The first argument of the `run` method in a `Computation` is a dictionary-like structure (see Code A.1) which is collected once the method completes. Each key of the dictionary is called in `Clustertools` jargon a metric. Any serializable object can be a metric but scalars, lists, and NumPy arrays are the most common types.

A `Storage` instance is responsible for saving the result object (see *infra*). During the saving period, the `Computation` instance enters the critical state. On normal completion, the state switches to completed.

It is possible to save the result manually during processing from the user code, in which case the critical state will be followed by the partial state. As of now, the support to continue an interrupted partial computation is limited but `Computation` instances in the partial state will not be overwritten when re-launching the `Experiment`.

`dev` branch

As of writing, the object transmitted to the `run` method in the `dev` branch is an instance of `Collector`, which has a few more responsibilities than the traditional result. It is the object managing the state, dealing with interrupts and it allows to monitor the duration of user-defined sub-tasks.

A.3.2 Storage

The `Storage` is responsible for committing everything to disk: results, states and logs. So far, only one concrete implementation exists in `Clustertools`: the `PickleStorage`. In principle, other classes could be derived as the relevant components receive a `Storage` factory in their constructor.

A `PickleStorage` object stores everything in the file system. General logs are saved in a folder that can be changed using the `CT_FOLDER` environment variable. By default, it points to a folder in the home directory. Along with the general logs, a folder is dedicated per `Experiment`. Within it, the logs, results, and states each have a dedicated folder with a file per `Computation`.

To render the writing to disk as atomic as possible, results are not overwritten. Instead, a rotation is organized so that files are committed to disk and then moved to the proper place.

A.3.3 Datacube

An OLAP structure, in the form of a hypercube, the `Datacube`, is provided by `Clustertools` to analyze the results. The `Datacube` is built from what is found in the disk. It is totally independent of how those were generated, and consequently from the other components.

A `Datacube` object provides a mapping from the parameter tuples to the space of metrics. Internally, all data are saved in a 1D array as is frequent with such structures. Sliced and diced views can be generated at low cost. Since the underlying buffer is not to be tempered with directly, all views share the same; there is no need for lengthy copies. There are also built-in facilities to iterate over the cubes.

Since constraints can prevent from running the `Computation` with the whole Cartesian space of parameters, the cube might contain holes. A couple of methods expose such holes.

Code A.3 shows a few manipulations of a `Datacube`. Note how the string representation is used for values to avoid issues with floating-point representation. See the examples for more.

A.3.4 Analytics

`Clustertools-analytics`, a separate but related package, offers means to facilitate and speed up the analysis of results. It is mainly composed of three elements.

`Accessor`. Formally, the accessing mechanisms offered by the `Datacube` class are sufficient. They might be quite cumbersome, however. To overcome this, the `Accessor` classes provide common shortcuts, such as for scalar metrics, series metrics, or for treating some parameters as special (*e.g.* aggregating easily over random seed to measure the variability).

Plots. Built on top of Matplotlib, a few classes are provided to quickly build common plots. They work with `Accessor` and allow to define `Conventions`

CODE A.3: example of using a Datacube (relative to Code A.1)

```python
from clustertools import build_datacube

# Creating the cube is easy: just give the name of the Experiment
cube = build_datacube("BasicUsage")
repr(cube)
# Output: Datacube(name='BasicUsage', metadata={'z': '4'}, \
#   parameters=['w', 'x'], domain={'w': ['5', '6'],  \
#   'x': ['1', '2', '3']}, metrics=['multiply', 'sum'], data='n/a')

# Dicing and slicing (x becomes a metadata)
cube(x=['1', '2'])
cube(x='1')

# Accessing a piece of result (sum is a metric)
cube("sum", x='1', w='5')

# Iterating over a parameter (iter_dimensions can take several)
# t_x is a singleton tuple taking all the values of x
# cube_i is the corresponding slice
for t_x, cube_i in cube.iter_dimensions("x"):
    pass
```

whose purpose is to decouple the data and its representation (which color, linestyle, etc.).

Tables. The last part of `Clustertools-analytics` concerns the rendering of tables. The two main goals are (i) to provide independence with the viewing back-end (*i.e.* same but one detail to generate tables as TSV, CSV or for LaTeX), and (ii) automatically color the tables, as was used throughout this thesis.

A.3.5 Logs

Logs have already been discussed. They are of two forms: `Clustertools` general logs (*i.e.* which `Experiment` is ran when) and `Computation` logs, which capture all that is printed to the standard and error output. The formers are saved at the root of the `Clustertools` folder, while the latters are saved within the `Experiment` folders.

A.4 Command-line manager

`Clustertools` ships with a command-line utility to manage some frequent tasks. The first argument is the name of the subprogram. It includes

`sync` transfers `Experiments`. It uses `rsync` to transfer the folder around.

`count` shows which `Computations` of a given `Experiment` are in which state. For the running jobs, the elapsed time, estimated time of completion, and estimated total duration are portrayed.

`launchable` sets a `Computation` (typically in aborted state) to launchable.

`reset` sets all the `Computations` as launchable.

`abr2lch` sets all the aborted `Computations` as launchable.

`display state` shows the state of a `Computation`;

`log` shows the logs of a `Computation`;

`info` shows information (parameters, duration, etc.) of a `Computation`.

`diagnose` shows the holes in the `Datacube` relating to an `Experiment`.

`list` lists `Computations` of an `Experiment` in a given state.

A.5 Conclusion

This chapter presented `Clustertools`, a Python package to automate scientific computation, taking advantage of the nature of the task to best parallelize lengthy computations, and in passing, automating many of the tasks involved.

When time permits, systematic coverage will be extended to the `dev` branch which will eventually be merged to the master branch.

B Appendix Out of distribution

B.1 Relationship between logit and T1000

The main result covered by this section is

$$p_{k|T=1000}(x) \approx \frac{c}{K} + \frac{1}{TK} z_k(x) \tag{B.1}$$

where k is the predicted class by the network, K is the number of class, $z(x)$ is the logit vector corresponding to input x and $T = 1000$ is the temperature. It holds so long as $||z|| \ll T$ and the network is trained long enough.

For shorthand, let $\hat{p}_k$ stands for softmax$_k$. From Taylor decomposition, it follows that

$$\begin{aligned}
p_{k|T=1000} &= \hat{p}_k\left(\frac{1}{T}z\right) && \text{(B.2)}\\
&= \hat{p}_k(0) + \frac{1}{T}\left(\nabla_z \hat{p}_k\left(0\right)\right)^T z + o\left(\frac{1}{T^2}||z||\right) && \text{(B.3)}\\
&\approx \frac{1}{K} + \frac{1}{T}\sum_{j=1}^{K}\left(\hat{p}_k(0)\left(\delta_{j,k} - \hat{p}_j(0)\right) z_j\right) && \text{(B.4)}\\
&= \frac{1}{K} + \frac{1}{T}\sum_{j=1}^{K}\left(\frac{1}{K}\left(\delta_{j,k} - \frac{1}{K}\right) z_j\right) && \text{(B.5)}\\
&= \frac{1}{K} + \frac{1}{TK}\left(z_k - \frac{1}{K}\sum_{j=1}^{K} z_j\right) && \text{(B.6)}\\
&= \frac{1}{K} + \frac{z_k - \bar{z}}{TK} && \text{(B.7)}
\end{aligned}$$

where $\delta_{j,k}$ is the Kronecker symbol.

By linearity, we have

$$
\begin{aligned}
\bar{z}(x) &= \frac{1}{K}\sum_j z_j(x) && \text{(B.8)}\\
&= \frac{1}{K}\sum_j \left(w_j^T x + b_j\right) && \text{(B.9)}\\
&= \overline{w}^T x + \bar{b} && \text{(B.10)}\\
\Delta z(x) &= z_k(x) - \bar{z}(x) && \text{(B.11)}\\
&= \left(w_k^T x + b_k\right) - \left(\overline{w}^T x + \bar{b}\right) && \text{(B.12)}\\
&= \Delta w_k^T x + \Delta b_k && \text{(B.13)}
\end{aligned}
$$

So we end up with a linear relationship between $p_{k|T=1000}$ and the Δ-logit Δz.

In addition, the average weight vector and bias tend to be close to null. Owing to the softmax translation invariance

$$
\frac{e^{z_k-(\overline{w}^T x+\hat{b})}}{\sum_j e^{z_j-(\overline{w}^T x+\hat{b})}} = \frac{e^{z_k}}{\sum_j e^{z_j}} \tag{B.14}
$$

the only incentive acting on the average weight vector and bias is the slight penalization which goes in the direction of $\overline{w} = 0$ and $\bar{b} = 0$. Indeed,

$$
\overline{w} = \frac{1}{K}\left(w_k + \sum_{j\neq k} w_j\right) = \frac{1}{K}\left(w_k + w_{\neg k}\right) \tag{B.15}
$$

Since $w_{\neg k}$ represents a hyperplane for rejecting class k, it would be wasteful for the network not enforcing $w_k = -w_{\neg k}$. As for $\bar{b}$, it does not depend on x and can be hidden away in the independent term.

Overall—provided the network was trained enough—we arrive at the conclusion

$$
\begin{aligned}
&\overline{w}^T x \ll w_k^T x && \text{(B.16)}\\
\implies\ & p_{k|T=1000} \approx \frac{c}{K} + \frac{z_k}{TK} && \text{(B.17)}
\end{aligned}
$$

For the purpose of ranking samples, we can further remove the constant term $\overline{\mathcal{Z}} = \bar{z} - \epsilon(z)$, where $\overline{\mathcal{Z}}$ is the expected average of logits over the input space and $\epsilon(z)$ is the deviation of the logit mean from its expectation. This

TABLE B.1: Mean values for the average weight vector and bias, as well as the logit of the predicted class (on the ID task). The first two tends to be very small, while the last one is several orders of magnitude higher. Although the logit is expected to be lower on an OOD task, orders of magnitude are equivalent. C. 10 and C. 100 stand for CIFAR 10 and CIFAR 100 respectively.

		$\|\|\overline{w}\|\|$	$\bar{b}$	z_k
C. 10	RESNET 50	$5\ 10^{-6} \pm 4\ 10^{-7}$	$1\ 10^{-7} \pm 3\ 10^{-7}$	11.3 ± 2.5
	WIDERESNET	$4\ 10^{-6} \pm 1\ 10^{-6}$	$-3\ 10^{-7} \pm 3\ 10^{-7}$	12.0 ± 3.4
	DENSENET 121	$3\ 10^{-6} \pm 1\ 10^{-7}$	$-2\ 10^{-7} \pm 2\ 10^{-7}$	10.2 ± 2.2
C. 100	RESNET 50	$2\ 10^{-6} \pm 6\ 10^{-8}$	$1\ 10^{-8} \pm 5\ 10^{-9}$	13.3 ± 3.8
	WIDERESNET	$4\ 10^{-6} \pm 4\ 10^{-6}$	$3\ 10^{-7} \pm 2\ 10^{-7}$	13.5 ± 4.4
	DENSENET 121	$7\ 10^{-7} \pm 2\ 10^{-7}$	$5\ 10^{-8} \pm 4\ 10^{-8}$	13.2 ± 4.1

leads to

$$= \frac{1}{K} + \frac{z_k - \left(\epsilon(z) + \overline{\mathcal{Z}}\right)}{T\,K} \tag{B.18}$$

$$= \frac{1}{K}\left(1 - \frac{\overline{\mathcal{Z}}}{T}\right) + \frac{z_k - \epsilon(z)}{T\,K} \tag{B.19}$$

$$= \frac{c'}{K} + \frac{z_k}{T\,K} - \frac{\epsilon(z)}{T\,K} \tag{B.20}$$

In this last relationship, $\epsilon(z)$ is of the order of magnitude of the standard deviation of $\overline{w}$ and $\bar{b}$, typically 8 order smaller than z_k (see Table B.1) and can be safely ignored.

B.2 Detailed results

B.2.1 Detailed auroc tables

Tables B.2-B.4 holds detailed results for CIFAR 10, CIFAR 100 and Imagenet as ID datasets, respectively.

Supervised approach Although not the focus of this work, we see that a supervised linear SVM (Cortes and Vapnik, 1995) established on the true ID/OOD mix distribution performs almost perfectly. ON CIFARs, it only struggles with Tiny ImageNet (TIN) and LSUN, where it still performs best, except on TIN with CIFAR 100. In that setting, the mean results of ACT, ANG and sometimes T1000 is slightly higher.

On ImageNet, it is perfect but for LSUN, where it comes first with a large margin.

Baselines ODIN and T1000 are strong baselines. For ImageNet, it would seem the additional perturbation provided by ODIN pays off, especially on grey images (fashion MNIST, MNIST). On CIFARs, the gap is much less present, except on MNIST for CIFAR 100 where 5 to 10 percent of auroc are lost. On harder tasks, T1000 may have a slight edge.

MP and H rarely yield remarkable results.

Batchnorm indicators The IN- family of indicators may work well at detecting grey images, although IN-DSS never really works. When input statistics are close to the ID's (Tiny ImageNet, LSUN), those indicators do not work better than random. They also fail on the noisy Gaussian dataset, which has individual pixel statistics that are close to ID's. It would be easy to reject such samples if inter-channel information were available.

On the other hand, indicators based on all batchnorm layers work extremely well on Gaussian since intermediate tensors contain inter-channel information, thanks to convolutions. Interestingly, DSS and/or DSS-EXT perform well on SVHN in all settings. Those indicators are much less robust to the network and its initialization, however. For instance, DMS achieves $82.73 \pm 0.56\%$ on DenseNet 121 for discriminating fashion MNIST against CIFAR 10, but it only achieves $69.39 \pm 6.49\%$ on ResNet 50 for the same task (note the high variance).

Quite often, batchnorm indicators have auroc much lower than 50%, indicating lower values for OOD samples. In our sample-free setting, we can only discard such indicators and conclude they can only discriminate specific OOD sets. However, in a supervised setting, such indicators might prove useful as the ordering condition we impose on indicators could be altogether ignored.

Latent space indicators As expected, NORM and NORM+ do not convey the appropriate information. The remaining indicators rank well, however. On ImageNet, positive-only indicators seem to work better, while this is not as clear for the other ID tasks. In particular, ANG++ performs better than ANG on ImageNet but ANG works better in the other settings (except for ResNet 50 on CIFAR 100). Once again, the OOD dataset has an impact on the ranking: ACT/ACT+ tend to struggle with (fashion) MNIST on CIFAR 100 and ImageNet, while, with ImageNet as ID task, ANG++ comes way ahead of the other indicators against CIFARs as OOD but underperforms on LSUN (except on WideResNet).

1C-Sum Overall, 1C-Sum results are good. Compared to individual indicators, it mainly lags behind on MNIST with CIFARs as ID sets and on LSUN. Hopefully, as soon as data become available, 1C-Sum can be turned into the supervised indicator to compensate for its initial weaknesses. More precisely, incorporating the IN- feature to better detect grey images and drop other batchnorm indicators for LSUN. Assumptions regarding the expected OOD distribution may also help tune the model weights.

CIFAR 10 vs. CIFAR 100 CIFAR 100 is a harder base task than CIFAR 10 and even well-optimized networks achieve more modest performances (Table 7.5). As we can see, the gap in accuracy is reflected in OOD detection as well, at least on optimality-based indicators, thus confirming that we also need the loss to be small for OOD detection.

TABLE B.2: Area under the ROC curve for OOD detection with CIFAR 10 as ID. TIN stands for Tiny ImageNet.

		GAUSSIAN	SVHN	MNIST	FASH. MNIST	TIN	LSUN
RESNET 50	ODIN	91.36 ± 5.42	90.22 ± 4.03	96.88 ± 0.70	95.89 ± 0.75	87.22 ± 2.12	92.38 ± 1.56
	T1000	83.17 ± 9.00	93.14 ± 3.05	94.81 ± 0.78	95.43 ± 0.62	88.70 ± 1.23	92.66 ± 1.04
	MP	89.27 ± 4.90	91.89 ± 1.30	90.76 ± 0.65	91.97 ± 0.47	87.05 ± 0.61	90.08 ± 0.60
	H	89.05 ± 5.03	92.51 ± 1.46	91.40 ± 0.62	92.71 ± 0.58	87.52 ± 0.67	90.62 ± 0.59
	NORM	53.96 ± 33.02	85.46 ± 10.89	92.28 ± 4.92	89.52 ± 4.00	80.19 ± 4.27	82.50 ± 4.93
	NORM+	54.99 ± 28.60	87.17 ± 9.12	94.61 ± 2.09	92.92 ± 1.85	85.00 ± 2.61	88.87 ± 2.82
	ACT	83.34 ± 9.02	93.32 ± 2.95	94.90 ± 0.70	95.47 ± 0.59	88.77 ± 1.18	92.50 ± 1.08
	ACT+	87.68 ± 9.18	94.23 ± 3.50	96.03 ± 1.44	95.93 ± 0.72	88.05 ± 1.53	91.68 ± 1.38
	PROJ	85.53 ± 8.09	94.01 ± 2.42	95.61 ± 0.40	95.47 ± 0.58	88.61 ± 1.26	92.05 ± 1.21
	ANG	91.78 ± 2.79	93.41 ± 0.09	94.15 ± 0.60	94.76 ± 1.02	88.35 ± 0.51	91.98 ± 0.58
	ANG++	99.89 ± 0.12	97.26 ± 0.17	94.25 ± 1.22	93.41 ± 1.70	86.05 ± 0.88	88.43 ± 0.75
	IN-DMS	7.85	60.46	98.59	71.94	52.89	49.26
	IN-DMS-AOS	52.79	30.41	99.68	96.02	52.55	54.91
	IN-DSS	5.13	85.99	36.16	58.53	52.03	42.94
	DMS	100.00 ± 0.00	80.29 ± 8.30	93.97 ± 2.47	69.39 ± 6.49	34.21 ± 5.54	22.67 ± 5.33
	DMS-AOS	99.25 ± 0.48	4.72 ± 2.26	81.12 ± 9.04	59.42 ± 9.53	25.25 ± 2.65	23.78 ± 2.66
	DSS	99.86 ± 0.14	96.51 ± 0.60	70.33 ± 15.30	62.22 ± 3.53	55.01 ± 1.90	47.40 ± 4.40
	DSS-EXT	98.24 ± 0.61	97.70 ± 0.34	66.93 ± 1.88	67.64 ± 1.67	66.84 ± 0.88	62.94 ± 1.38
	SUPERVISED	100.00 ± 0.00	99.75 ± 0.05	100.00 ± 0.00	99.70 ± 0.03	90.82 ± 0.45	96.14 ± 0.19
	1C-SUM	97.84 ± 2.70	97.83 ± 0.95	96.47 ± 1.58	95.86 ± 0.63	88.86 ± 0.79	91.61 ± 0.90
WIDERESNET	ODIN	99.73 ± 0.20	90.85 ± 5.11	94.11 ± 4.14	95.22 ± 1.16	84.31 ± 4.70	90.22 ± 2.87
	T1000	98.13 ± 1.37	95.20 ± 1.76	91.59 ± 4.71	94.88 ± 0.79	87.65 ± 2.06	91.49 ± 1.49
	MP	96.69 ± 1.60	93.34 ± 1.01	88.42 ± 3.64	92.01 ± 0.33	86.62 ± 1.04	89.69 ± 0.75
	H	97.41 ± 1.65	94.10 ± 1.25	89.08 ± 3.89	92.76 ± 0.38	87.09 ± 1.16	90.22 ± 0.85
	NORM	98.11 ± 2.24	92.21 ± 4.95	89.09 ± 10.60	87.96 ± 5.56	76.42 ± 7.85	79.48 ± 6.15
	NORM+	98.97 ± 0.90	93.73 ± 3.41	90.84 ± 7.45	92.89 ± 2.58	83.54 ± 4.64	87.58 ± 2.34
	ACT	98.32 ± 1.24	95.35 ± 1.71	91.96 ± 4.39	94.96 ± 0.77	87.73 ± 2.03	91.36 ± 1.50
	ACT+	99.17 ± 0.75	95.54 ± 2.36	92.45 ± 5.73	94.87 ± 1.46	85.57 ± 3.49	89.70 ± 2.72
	PROJ	98.29 ± 1.38	95.67 ± 1.46	92.88 ± 3.68	94.94 ± 0.69	87.74 ± 1.92	90.97 ± 1.64
	ANG	96.24 ± 1.90	92.63 ± 0.75	90.78 ± 1.20	93.49 ± 0.78	88.68 ± 0.50	91.61 ± 1.07
	ANG++	97.41 ± 2.15	92.46 ± 1.83	87.46 ± 3.74	86.36 ± 3.87	80.42 ± 0.15	81.92 ± 4.51
	IN-DMS	7.85	60.46	98.59	71.94	52.89	49.26
	IN-DMS-AOS	52.79	30.41	99.68	96.02	52.55	54.91
	IN-DSS	5.13	85.99	36.16	58.53	52.03	42.94
	DMS	100.00 ± 0.00	94.13 ± 0.87	98.26 ± 1.11	80.14 ± 3.04	48.94 ± 0.77	38.39 ± 0.75
	DMS-AOS	100.00 ± 0.00	4.10 ± 0.62	78.41 ± 2.76	53.01 ± 2.33	35.58 ± 1.63	40.37 ± 2.01
	DSS	100.00 ± 0.00	96.83 ± 0.27	83.13 ± 2.05	80.13 ± 2.09	54.11 ± 3.19	40.92 ± 4.76
	DSS-EXT	94.81 ± 1.16	97.77 ± 0.24	68.79 ± 2.13	71.78 ± 1.34	63.17 ± 2.63	53.89 ± 3.84
	SUPERVISED	100.00 ± 0.00	99.74 ± 0.04	100.00 ± 0.00	99.69 ± 0.02	90.64 ± 0.38	95.13 ± 0.64
	1C-SUM	100.00 ± 0.00	98.87 ± 0.19	94.98 ± 2.17	95.50 ± 0.73	87.49 ± 2.51	89.28 ± 2.89
DENSENET 121	ODIN	99.49 ± 0.37	82.99 ± 3.14	85.32 ± 8.29	88.69 ± 3.87	75.92 ± 1.61	82.35 ± 3.57
	T1000	96.93 ± 1.89	93.66 ± 1.47	84.19 ± 6.56	91.87 ± 1.89	83.51 ± 0.49	87.53 ± 2.03
	MP	96.49 ± 0.91	93.07 ± 1.30	85.76 ± 3.96	91.67 ± 0.71	85.48 ± 0.30	88.45 ± 1.00
	H	96.79 ± 1.22	93.62 ± 1.37	86.04 ± 4.10	92.14 ± 0.78	85.72 ± 0.35	88.75 ± 1.10
	NORM	51.24 ± 34.67	65.16 ± 6.79	44.44 ± 22.17	51.29 ± 15.42	46.78 ± 2.38	49.02 ± 4.55
	NORM+	65.46 ± 29.49	78.63 ± 4.36	59.79 ± 17.81	73.50 ± 11.20	66.09 ± 1.53	70.25 ± 3.43
	ACT	97.13 ± 1.78	93.86 ± 1.43	84.58 ± 6.46	91.94 ± 1.83	83.59 ± 0.49	87.32 ± 2.04
	ACT+	96.30 ± 3.39	90.53 ± 2.00	76.80 ± 11.96	85.04 ± 6.23	76.24 ± 0.83	80.38 ± 2.96
	PROJ	97.45 ± 1.61	94.39 ± 1.44	86.86 ± 5.04	92.07 ± 1.67	83.55 ± 0.42	86.22 ± 2.07
	ANG	98.75 ± 0.39	96.46 ± 0.79	94.65 ± 0.44	95.44 ± 0.88	89.68 ± 0.39	91.86 ± 0.71
	ANG++	99.85 ± 0.15	94.74 ± 1.24	95.67 ± 0.63	90.89 ± 1.15	83.01 ± 2.71	84.29 ± 2.09
	IN-DMS	7.85	60.46	98.59	71.94	52.89	49.26
	IN-DMS-AOS	52.79	30.41	99.68	96.02	52.55	54.91
	IN-DSS	5.13	85.99	36.16	58.53	52.03	42.94
	DMS	99.99 ± 0.00	94.61 ± 0.61	98.29 ± 0.36	82.73 ± 0.56	42.67 ± 1.58	34.95 ± 1.30
	DMS-AOS	98.79 ± 0.39	12.33 ± 1.91	74.75 ± 0.64	55.15 ± 1.00	27.51 ± 0.77	35.15 ± 0.31
	DSS	99.87 ± 0.10	97.09 ± 0.45	84.08 ± 1.36	78.19 ± 3.74	60.25 ± 2.33	42.50 ± 2.02
	DSS-EXT	99.06 ± 0.32	97.61 ± 0.29	79.54 ± 1.82	76.86 ± 1.93	70.61 ± 1.88	56.10 ± 2.18
	SUPERVISED	100.00 ± 0.00	99.78 ± 0.05	99.98 ± 0.00	99.74 ± 0.04	92.02 ± 0.34	95.64 ± 0.09
	1C-SUM	100.00 ± 0.00	97.89 ± 0.54	92.68 ± 3.43	93.69 ± 2.02	83.47 ± 1.09	83.76 ± 2.81

TABLE B.3: Area under the ROC curve for OOD detection with CIFAR 100 as ID. TIN stands for Tiny ImageNet.

		GAUSSIAN	SVHN	MNIST	FASH. MNIST	TIN	LSUN
RESNET 50	ODIN	95.76 ± 5.19	79.33 ± 3.38	81.32 ± 2.85	90.12 ± 0.41	76.40 ± 0.19	72.32 ± 0.91
	T1000	89.86 ± 10.20	84.64 ± 3.11	74.49 ± 2.97	88.95 ± 0.30	77.99 ± 0.10	72.58 ± 0.61
	MP	85.59 ± 9.02	76.65 ± 4.14	69.09 ± 2.66	83.93 ± 0.78	77.34 ± 0.07	73.46 ± 0.17
	H	83.18 ± 9.83	73.58 ± 4.21	64.83 ± 2.69	80.49 ± 0.59	72.86 ± 0.11	72.21 ± 2.45
	NORM	54.94 ± 33.83	71.68 ± 7.59	66.84 ± 4.26	64.45 ± 3.31	51.99 ± 1.12	53.81 ± 1.58
	NORM+	65.88 ± 35.79	78.49 ± 6.22	67.89 ± 2.16	79.23 ± 2.00	65.95 ± 0.60	62.63 ± 1.35
	ACT	89.98 ± 9.92	84.80 ± 3.08	74.66 ± 2.93	88.93 ± 0.30	78.00 ± 0.10	72.51 ± 0.63
	ACT+	90.95 ± 11.23	88.55 ± 2.64	79.42 ± 3.69	88.46 ± 0.29	74.54 ± 0.09	71.80 ± 0.86
	PROJ	89.53 ± 9.93	81.13 ± 4.19	74.02 ± 3.17	88.40 ± 0.28	78.04 ± 0.14	71.18 ± 0.56
	ANG	94.04 ± 2.86	75.70 ± 2.60	70.15 ± 2.99	88.44 ± 0.43	80.53 ± 0.16	72.73 ± 0.12
	ANG++	99.66 ± 0.14	81.42 ± 1.28	82.02 ± 4.09	89.17 ± 0.58	79.26 ± 0.18	75.66 ± 0.62
	IN-DMS	0.01	22.19	88.75	42.44	16.50	32.71
	IN-DMS-AOS	46.01	26.70	98.78	92.11	46.88	48.78
	IN-DSS	0.00	30.31	2.55	11.62	6.87	26.80
	DMS	99.94 ± 0.03	0.25 ± 0.12	13.40 ± 9.11	0.75 ± 0.55	0.46 ± 0.08	21.96 ± 15.56
	DMS-AOS	99.10 ± 0.82	7.03 ± 1.27	77.65 ± 5.84	54.39 ± 4.72	37.60 ± 0.79	40.94 ± 0.32
	DSS	0.00 ± 0.00	4.60 ± 1.34	0.00 ± 0.00	0.00 ± 0.00	0.31 ± 0.22	22.55 ± 15.79
	DSS-EXT	96.67 ± 1.36	95.98 ± 0.36	83.69 ± 2.62	74.09 ± 1.33	52.02 ± 0.68	43.52 ± 0.15
	SUPERVISED	100.00 ± 0.00	99.04 ± 0.14	99.98 ± 0.02	99.36 ± 0.24	79.18 ± 0.30	84.88 ± 1.16
	1C-SUM	99.85 ± 0.17	92.97 ± 0.84	84.14 ± 3.56	90.52 ± 0.45	76.93 ± 1.01	70.25 ± 1.43
WIDERESNET	ODIN	98.48 ± 1.06	81.28 ± 3.49	84.98 ± 2.82	91.10 ± 1.04	77.58 ± 0.42	73.20 ± 2.69
	T1000	94.49 ± 3.78	86.47 ± 2.43	78.51 ± 3.23	89.63 ± 0.92	78.99 ± 0.25	73.05 ± 2.43
	MP	92.03 ± 6.00	77.21 ± 2.06	70.71 ± 2.92	81.93 ± 0.97	76.17 ± 0.17	72.18 ± 1.08
	H	94.37 ± 4.14	80.54 ± 2.23	72.63 ± 3.04	85.29 ± 1.10	78.04 ± 0.11	78.03 ± 3.05
	NORM	67.44 ± 29.49	74.28 ± 5.66	75.02 ± 6.14	79.18 ± 2.71	63.00 ± 2.52	59.63 ± 2.45
	NORM+	80.19 ± 22.36	79.68 ± 4.77	74.43 ± 4.11	85.21 ± 1.60	71.13 ± 1.57	66.82 ± 2.56
	ACT	94.41 ± 3.84	86.74 ± 2.37	78.80 ± 3.17	89.58 ± 0.96	79.00 ± 0.24	72.94 ± 2.42
	ACT+	97.40 ± 2.04	89.09 ± 2.27	82.62 ± 3.83	89.68 ± 0.81	76.47 ± 0.72	70.52 ± 2.77
	PROJ	94.80 ± 3.59	85.51 ± 2.26	79.05 ± 3.61	89.69 ± 0.90	78.90 ± 0.23	73.06 ± 2.13
	ANG	96.22 ± 2.52	82.42 ± 1.30	75.17 ± 2.02	87.68 ± 1.09	79.93 ± 0.33	74.65 ± 1.28
	ANG++	97.02 ± 2.08	84.14 ± 1.29	83.48 ± 3.30	84.41 ± 0.77	75.10 ± 0.21	69.17 ± 1.05
	IN-DMS	0.01	22.19	88.75	42.44	16.50	32.71
	IN-DMS-AOS	46.01	26.70	98.78	92.11	46.88	48.78
	IN-DSS	0.00	30.31	2.55	11.92	6.87	26.80
	DMS	100.00 ± 0.00	84.65 ± 1.38	91.79 ± 1.54	62.49 ± 1.26	39.19 ± 0.54	75.94 ± 30.23
	DMS-AOS	100.00 ± 0.00	6.61 ± 0.81	62.58 ± 5.99	43.24 ± 4.88	40.56 ± 0.81	48.03 ± 0.36
	DSS	99.99 ± 0.00	94.85 ± 0.04	86.68 ± 1.64	82.52 ± 1.22	44.58 ± 0.61	77.31 ± 31.68
	DSS-EXT	85.25 ± 1.53	95.52 ± 0.14	78.13 ± 2.66	75.51 ± 1.68	51.30 ± 0.75	37.70 ± 0.94
	SUPERVISED	100.00 ± 0.00	99.22 ± 0.06	99.98 ± 0.02	99.20 ± 0.10	78.40 ± 0.05	81.15 ± 1.64
	1C-SUM	100.00 ± 0.00	95.44 ± 0.99	84.95 ± 4.30	91.06 ± 1.29	77.30 ± 0.32	68.55 ± 2.62
DENSENET 121	ODIN	97.79 ± 2.03	80.72 ± 1.10	75.12 ± 7.12	90.68 ± 1.28	79.22 ± 1.11	74.42 ± 1.55
	T1000	92.50 ± 5.48	87.45 ± 1.47	67.66 ± 6.00	89.77 ± 1.27	80.36 ± 0.78	73.99 ± 1.43
	MP	78.51 ± 13.57	82.17 ± 1.57	67.17 ± 1.57	83.52 ± 0.03	78.33 ± 0.04	74.09 ± 0.88
	H	76.84 ± 13.03	77.78 ± 1.55	61.64 ± 2.34	79.65 ± 0.32	75.32 ± 3.00	74.88 ± 0.96
	NORM	69.60 ± 27.50	58.29 ± 1.80	37.50 ± 14.69	64.73 ± 5.77	60.14 ± 3.49	60.96 ± 2.14
	NORM+	79.09 ± 26.05	69.80 ± 2.88	44.41 ± 14.49	78.80 ± 3.44	70.54 ± 2.26	66.05 ± 2.77
	ACT	92.38 ± 5.56	87.50 ± 1.48	67.77 ± 6.02	89.77 ± 1.25	80.39 ± 0.78	73.97 ± 1.43
	ACT+	93.93 ± 5.11	87.97 ± 0.99	66.48 ± 8.88	88.31 ± 2.40	77.85 ± 1.42	73.97 ± 1.70
	PROJ	92.36 ± 5.32	85.34 ± 2.32	67.46 ± 5.92	89.12 ± 1.12	80.23 ± 0.66	72.53 ± 1.31
	ANG	92.01 ± 7.24	86.73 ± 2.62	77.82 ± 0.67	89.85 ± 0.36	81.26 ± 0.10	72.25 ± 0.35
	ANG++	89.91 ± 13.68	87.72 ± 2.17	84.04 ± 3.14	85.70 ± 0.96	76.23 ± 0.09	71.79 ± 0.25
	IN-DMS	0.01	22.19	88.75	42.44	16.50	42.68
	IN-DMS-AOS	46.01	26.70	98.78	92.11	46.88	48.78
	IN-DSS	0.00	30.31	2.55	11.62	6.87	38.55
	DMS	100.00 ± 0.00	3.41 ± 1.67	6.56 ± 2.87	0.45 ± 0.19	13.92 ± 18.52	35.32 ± 0.33
	DMS-AOS	99.99 ± 0.01	8.99 ± 1.44	51.12 ± 4.77	33.48 ± 2.93	38.62 ± 0.81	50.00 ± 0.67
	DSS	1.84 ± 2.42	6.24 ± 1.04	0.00 ± 0.01	0.00 ± 0.00	14.04 ± 19.82	31.02 ± 1.71
	DSS-EXT	96.60 ± 0.74	95.39 ± 0.67	90.90 ± 0.63	84.26 ± 0.63	50.79 ± 0.32	33.29 ± 1.01
	SUPERVISED	100.00 ± 0.00	99.26 ± 0.08	99.98 ± 0.01	99.43 ± 0.09	80.30 ± 0.48	84.84 ± 0.30
	1C-SUM	99.40 ± 0.85	93.23 ± 1.61	79.50 ± 3.08	92.06 ± 1.39	80.10 ± 0.36	72.54 ± 1.78

TABLE B.4: Area under the ROC curve for OOD detection with ImageNet as ID.

		GAUSSIAN	SVHN	MNIST	FASH. MNIST	LSUN	CIFAR 10	CIFAR 100
RESNET 50	ODIN	99.96	99.82	99.73	94.16	80.38	87.66	89.98
	T1000	98.77	98.37	98.03	87.28	78.17	84.23	86.48
	MP	93.87	97.25	91.65	86.99	75.70	81.57	84.75
	H	97.55	98.36	96.16	89.32	77.47	84.36	87.25
	NORM	99.62	95.89	99.53	57.95	62.92	48.98	58.23
	NORM+	99.80	96.35	99.58	63.67	70.37	55.09	63.66
	ACT	98.74	98.37	98.05	87.29	78.15	84.18	86.46
	ACT+	99.94	99.58	99.77	87.29	77.36	83.00	86.89
	PROJ	98.77	98.21	98.43	88.37	77.04	83.59	86.31
	ANG	90.53	93.83	88.94	89.26	74.70	86.40	87.21
	ANG++	99.87	99.56	99.51	98.23	75.03	94.46	96.14
	IN-DMS	26.50	61.58	97.00	68.76	50.00	50.79	56.51
	IN-DMS-AOS	42.46	22.50	98.87	92.60	56.66	38.43	43.72
	IN-DSS	7.52	86.70	55.74	71.10	43.38	54.91	59.55
	DMS	99.92	97.10	99.18	95.69	38.20	86.99	89.04
	DMS-AOS	8.10	16.26	85.79	73.03	52.50	34.10	33.59
	DSS	100.00	98.15	98.19	88.37	31.08	80.06	84.03
	DSS-EXT	100.00	93.17	87.10	72.58	36.36	73.95	77.98
	SUPERVISED	100.00	99.98	100.00	99.79	85.18	99.13	99.09
	1C-SUM	100.00	99.43	99.62	93.28	61.76	87.51	91.55
WIDERESNET	ODIN	100.00	99.91	99.36	96.13	78.42	87.79	89.44
	T1000	99.77	96.59	96.26	88.87	77.00	83.20	85.16
	MP	99.70	95.25	92.17	87.41	77.51	82.89	85.41
	H	99.69	96.96	95.49	89.78	78.90	84.96	87.42
	NORM	80.56	53.96	74.21	40.52	44.73	27.21	31.31
	NORM+	92.88	59.47	80.81	45.66	53.56	32.13	36.98
	ACT	99.77	96.58	96.27	88.88	76.98	83.17	85.14
	ACT+	99.92	97.96	98.25	89.36	70.56	81.72	84.04
	PROJ	99.85	96.67	96.93	90.17	76.64	81.69	84.60
	ANG	98.93	96.16	93.74	92.55	79.65	89.14	90.46
	ANG++	99.98	99.70	99.51	99.15	79.40	96.50	97.12
	IN-DMS	26.50	61.58	97.00	68.76	50.00	50.79	56.51
	IN-DMS-AOS	42.46	22.50	98.87	92.60	56.66	38.43	43.72
	IN-DSS	7.52	86.70	55.74	71.10	43.38	54.91	59.55
	DMS	99.74	95.03	99.92	96.69	48.86	81.34	84.74
	DMS-AOS	4.70	12.07	99.67	83.86	58.68	23.12	26.21
	DSS	99.06	96.37	99.51	96.17	27.21	85.18	87.43
	DSS-EXT	99.99	96.36	92.57	87.23	32.86	84.58	86.41
	SUPERVISED	100.00	99.98	100.00	99.98	88.23	99.65	99.54
	1C-SUM	99.94	99.27	99.94	97.56	71.73	88.27	91.35
DENSENET 121	ODIN	100.00	99.54	98.08	92.86	81.90	86.44	88.10
	T1000	99.84	99.02	92.79	88.70	79.93	85.72	87.78
	MP	97.27	97.61	80.44	87.31	76.95	83.16	85.56
	H	99.87	98.79	86.31	90.04	79.01	86.01	88.13
	NORM	99.94	94.53	94.32	49.85	58.91	45.45	57.07
	NORM+	99.97	94.64	95.82	57.67	67.43	51.74	63.24
	ACT	99.85	99.03	92.88	88.71	79.90	85.68	87.76
	ACT+	99.98	99.59	97.33	88.40	77.65	81.84	86.33
	PROJ	99.93	98.94	95.15	88.97	77.85	85.73	87.97
	ANG	95.93	95.93	85.38	90.89	75.49	88.79	88.77
	ANG++	99.96	99.33	97.79	97.94	73.80	93.29	94.81
	IN-DMS	26.50	61.58	97.00	68.76	50.00	50.79	56.51
	IN-DMS-AOS	42.46	22.50	98.87	92.60	56.66	38.43	43.72
	IN-DSS	7.52	86.70	55.74	71.10	43.38	54.91	59.55
	DMS	99.17	90.45	97.55	91.63	52.45	79.22	82.03
	DMS-AOS	0.09	4.23	93.40	76.40	56.95	16.62	18.08
	DSS	100.00	98.92	99.86	97.76	30.66	90.02	92.35
	DSS-EXT	100.00	97.93	94.25	89.08	34.29	88.85	90.68
	SUPERVISED	100.00	99.95	99.99	99.87	87.13	98.84	98.69
	1C-SUM	99.99	99.74	99.78	97.63	63.98	92.15	95.00

B.3 Selected indicator distributions

In this section, we would like to share some distributions of the indicators. Figure B.1 displays the distribution for some of bounded indicators. In theory, those are the easiest to threshold without data. In practice, setting the cut points without data is challenging. Interestingly, in an application where rejecting ID samples is less of a problem, MP and H seem good candidates to minimize the OOD acceptance rate. This also explains why they benefit so much from rejecting misclassified samples: ID samples is their main source of mistakes—reducing the number of such samples improves the auroc score.

Figure B.2 displays some distributions for unbounded indicators. It does seem that pinpointing where to place the threshold is quite hard.

Figures B.3 and B.4 focuses on batchnorm indicators. As one can see, these indicators are of limited use in most cases.

Finally, Figure B.5 displays the distribution of 1C-Sum in various settings. Without any prior knowledge, placing the optimal threshold is, once more, challenging. The dependency on the network might be more important than the one on the ID task (ResNet 50 for ImageNet is slightly different than for CIFAR 10/100). We leave the evaluation of transferability/meta-learning of the threshold as future work.

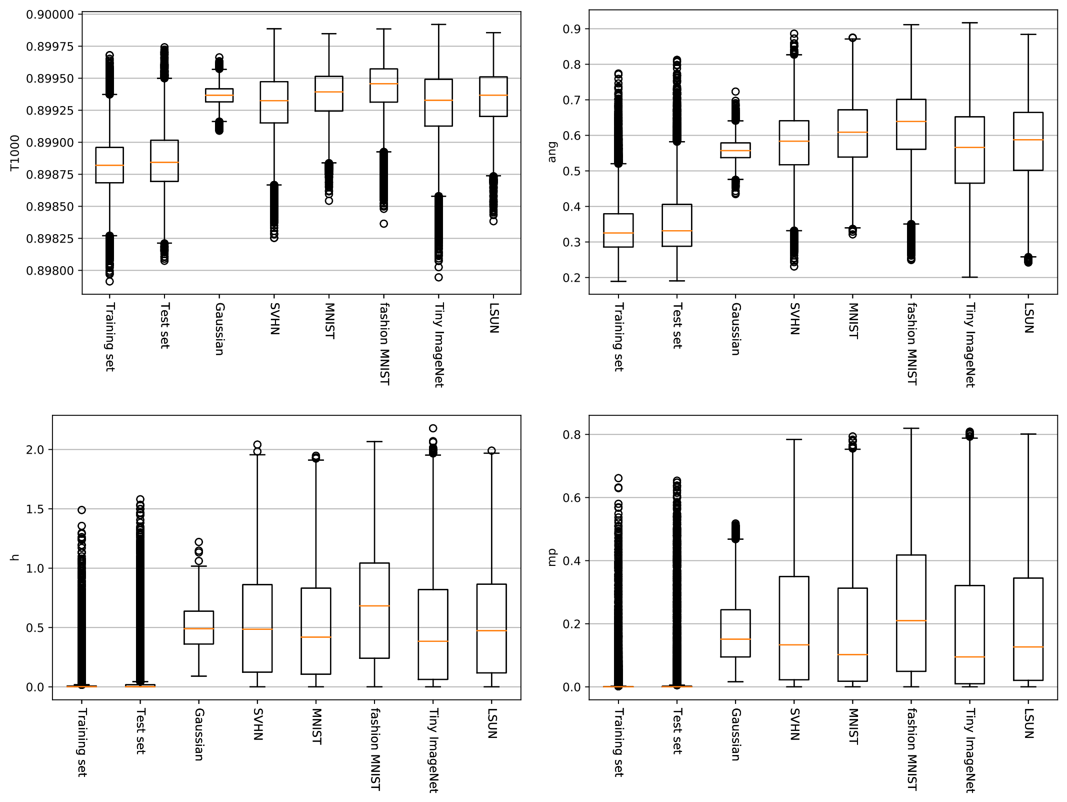


FIGURE B.1: Bounded indicator distributions established with CIFAR 10 as ID task on ResNet 50.

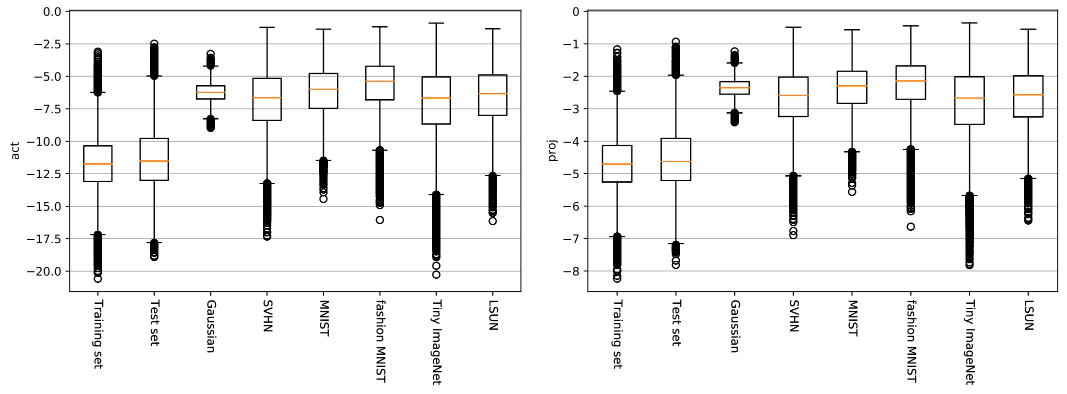


FIGURE B.2: Unbounded indicator distributions established with CIFAR 10 as ID task on ResNet 50.

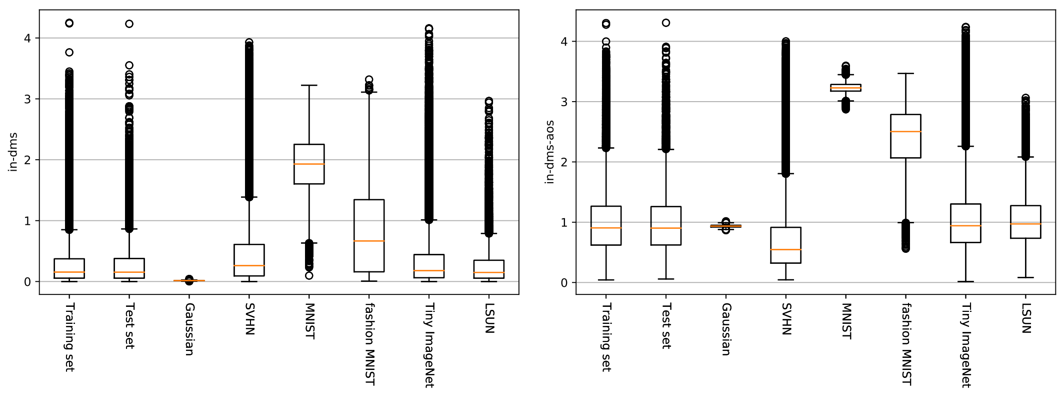


FIGURE B.3: IN- indicator distributions established with CIFAR 10 as ID task on ResNet 50.

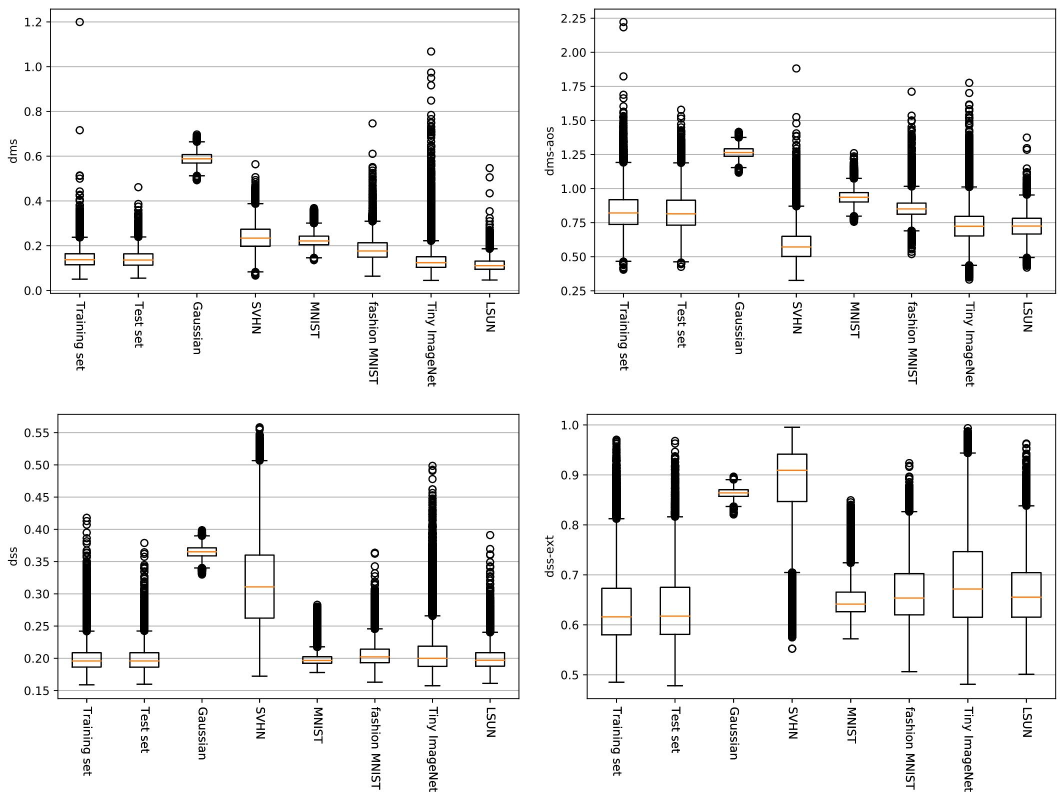
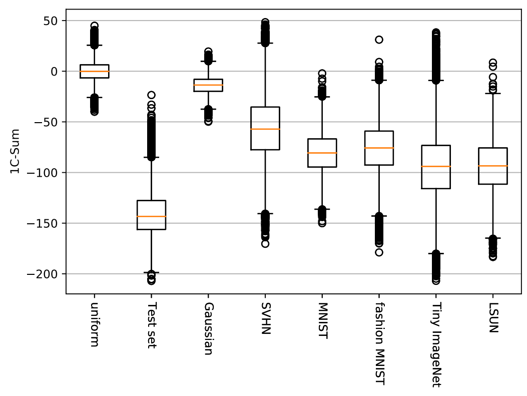
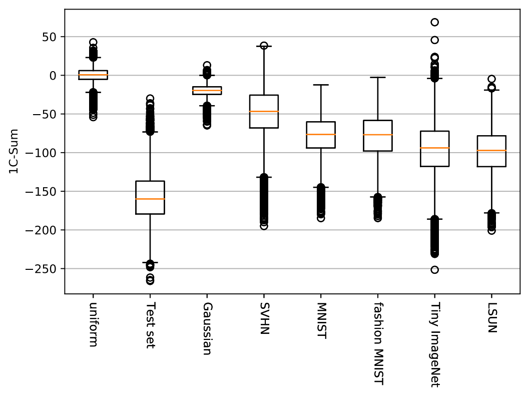


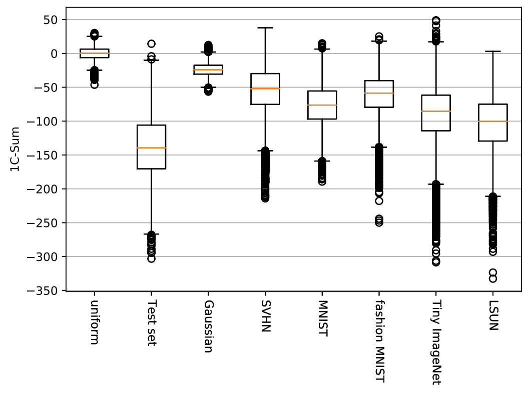
FIGURE B.4: Batchnorm indicator distributions established with CIFAR 10 as ID task on ResNet 50.



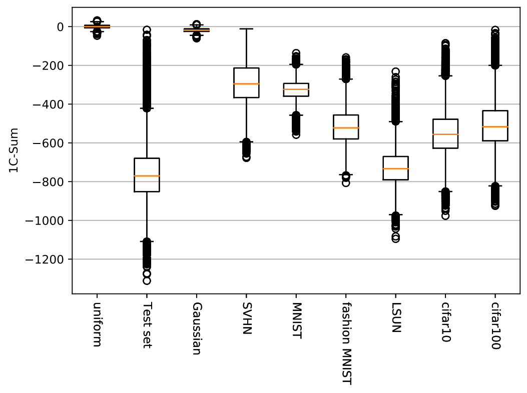
(A) CIFAR 10 as ID task on ResNet 50



(B) CIFAR 10 as ID task on WideResNet



(C) CIFAR 100 as ID task on ResNet 50



(D) ImageNet as ID task on ResNet 50

FIGURE B.5: 1C-Sum indicator distributions.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., et al. (2021). "A review of uncertainty quantification in deep learning: Techniques, applications and challenges". In: *Information Fusion*.

Ahmed, F. and Courville, A. C. (2020). "Detecting Semantic Anomalies". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, pp. 3154–3162. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5712.

Aigrain, J. and Detyniecki, M. (2019). "Detecting adversarial examples and other misclassifications in neural networks by introspection". In: *arXiv preprint arXiv:1905.09186*.

Al-Rfou, R. et al. (2016). "Theano: A Python framework for fast computation of mathematical expressions". In: *CoRR* abs/1605.02688. arXiv: 1605.02688. URL: http://arxiv.org/abs/1605.02688.

Aldweesh, A., Derhab, A., and Emam, A. Z. (2020). "Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues". In: *Knowl. Based Syst.* 189. DOI: 10.1016/j.knosys.2019.105124. URL: https://doi.org/10.1016/j.knosys.2019.105124.

Allen-Zhu, Z., Li, Y., and Song, Z. (2019). "A Convergence Theory for Deep Learning via Over-Parameterization". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 242–252. URL: http://proceedings.mlr.press/v97/allen-zhu19a.html.

Alvarez, J. M. and Salzmann, M. (2016). "Learning the Number of Neurons in Deep Networks". In: *Advances in Neural Information Processing Systems*

29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pp. 2262–2270. URL: https://proceedings.neurips.cc/paper/2016/hash/6e7d2da6d3953058db75714ac400b584-Abstract.html.

Ancona, M., Ceolini, E., Öztireli, C., and Gross, M. (2018). "Towards better understanding of gradient-based attribution methods for Deep Neural Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Sy21R9JAW.

Antonello, N. and Garner, P. N. (2020). "A t-Distribution Based Operator for Enhancing Out of Distribution Robustness of Neural Network Classifiers". In: *IEEE Signal Processing Letters* 27, pp. 1070–1074.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. (2018). "Stronger Generalization Bounds for Deep Nets via a Compression Approach". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 254–263. URL: http://proceedings.mlr.press/v80/arora18b.html.

Ba, J. and Caruana, R. (2014). "Do Deep Nets Really Need to be Deep?" In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 2654–2662. URL: https://proceedings.neurips.cc/paper/2014/hash/ea8fcd92d59581717e06eb187f10666d-Abstract.html.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PLoS ONE* 10.7, e0130140.

Bachman, P., Alsharif, O., and Precup, D. (2014). "Learning with Pseudo-Ensembles". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 3365–3373. URL: https://proceedings.neurips.cc/paper/2014/hash/66be31e4c40d676991f2405aaecc6934-Abstract.html.

Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., and Müller, K. (2010). "How to Explain Individual Classification Decisions". In: *The Journal of Machine Learning Research* 11, pp. 1803–1831. URL: http://portal.acm.org/citation.cfm?id=1859912.

Beckman, R. J. and Cook, R. D. (1983). "Outlier.......... s". In: *Technometrics* 25.2, pp. 119–149.

Begon, J.-M. and Geurts, P. (2021). "Sample-Free White-Box Out-of-Distribution Detection for Deep Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3290–3299.

Begon, J., Joly, A., and Geurts, P. (2017). “Globally Induced Forest: A Prepruning Compression Scheme”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 420–428. URL: http://proceedings.mlr.press/v70/begon17a.html.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). “A theory of learning from different domains”. In: *Machine learning* 79.1, pp. 151–175.

Bénard, C., Biau, G., Veiga, S. D., and Scornet, E. (2021). “Interpretable Random Forests via Rule Extraction”. In: *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 937–945. URL: http://proceedings.mlr.press/v130/benard21a.html.

Bendale, A. and Boult, T. E. (2016). “Towards open set deep networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1563–1572.

Biau, G., Scornet, E., and Welbl, J. (2016). “Neural Random Forests”. In: *CoRR* abs/1604.07143. arXiv: 1604.07143. URL: http://arxiv.org/abs/1604.07143.

Blake, C. and Merz, C. J. (1998). *{UCI} Repository of machine learning databases*. URL: https://archive.ics.uci.edu/ml/index.php.

Blalock, D. W., Ortiz, J. J. G., Frankle, J., and Guttag, J. V. (2020). “What is the State of Neural Network Pruning?” In: *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. Ed. by I. S. Dhillon, D. S. Papailiopoulos, and V. Sze. mlsys.org. URL: https://proceedings.mlsys.org/book/296.pdf.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1987). “Occam’s razor”. In: *Information processing letters* 24.6, pp. 377–380.

Bolton, R. J. and Hand, D. J. (2001). “Peer group analysis–local anomaly detection in longitudinal data”. In: *Technical Report*.

Bottou, L. and Bousquet, O. (2007). “The Tradeoffs of Large Scale Learning”. In: *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*. Ed. by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis. Curran Associates, Inc., pp. 161–168. URL: https://proceedings.neurips.cc/paper/2007/hash/0d3180d672e08b4c5312dcdafdf6e
Abstract.html.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. URL: http://github.com/google/jax.

Breiman, L. (1996). “Bagging Predictors”. In: *Mach. Learn.* 24.2, pp. 123–140. DOI: 10.1007/BF00058655. URL: https://doi.org/10.1007/BF00058655.

— (1999). “Pasting small votes for classification in large databases and online”. In: *Machine Learning* 36.1-2, pp. 85–103.

Breiman, L. (2001). "Random Forests". In: *Mach. Learn.* 45.1, pp. 5–32. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

— (2002). "Manual on setting up, using, and understanding random forests v3. 1". In: *Statistics Department University of California Berkeley, CA, USA* 1.58.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth. ISBN: 0-534-98053-8.

Breiman, L. et al. (1998). "Arcing classifier (with discussion and a rejoinder by the author)". In: *The annals of statistics* 26.3, pp. 801–849.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4, pp. 18–42.

Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020a). "ZeroQ: A Novel Zero Shot Quantization Framework". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, pp. 13166–13175. DOI: 10.1109/CVPR42600.2020.01318. URL: https://doi.org/10.1109/CVPR42600.2020.01318.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020b). "Zeroq: A novel zero shot quantization framework". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13169–13178.

Canziani, A., Paszke, A., and Culurciello, E. (2016). "An Analysis of Deep Neural Network Models for Practical Applications". In: *CoRR* abs/1605.07678. arXiv: 1605.07678. URL: http://arxiv.org/abs/1605.07678.

Carreira-Perpiñán, M. Á. and Tavallali, P. (2018). "Alternating optimization of decision trees, with application to learning sparse oblique trees". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 1219–1229. URL: https://proceedings.neurips.cc/paper/2018/hash/185c29dc24325934ee377cfda20e414c-Abstract.html.

Carreira-Perpiñán, M. Á. and Zharmagambetov, A. (2020). "Ensembles of Bagged TAO Trees Consistently Improve over Random Forests, AdaBoost and Gradient Boosting". In: *FODS '20: ACM-IMS Foundations of Data Science Conference, Virtual Event, USA, October 19-20, 2020*. Ed. by J. M. Wing and D. Madigan. ACM, pp. 35–46. DOI: 10.1145/3412815.3416882. URL: https://doi.org/10.1145/3412815.3416882.

Chandola, V., Banerjee, A., and Kumar, V. (2009). "Anomaly detection: A survey". In: *ACM Comput. Surv.* 41.3, 15:1–15:58. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

Chapelle, O., Scholkopf, B., and Zien, A. (2009). "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]". In: *IEEE Transactions on Neural Networks* 20.3, pp. 542–542.

Che, T., Liu, X., Li, S., Ge, Y., Zhang, R., Xiong, C., and Bengio, Y. (2021). "Deep Verifier Networks: Verification of Deep Discriminative Models with Deep Generative Models". In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, pp. 7002–7010. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16862.

Chen, H., Wang, Y., Xu, C., Yang, Z., Liu, C., Shi, B., Xu, C., Xu, C., and Tian, Q. (2019). "Data-free learning of student networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3514–3522.

Chen, P. H., Si, S., Li, Y., Chelba, C., and Hsieh, C. (2018). "GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 11011–11021. URL: https://proceedings.neurips.cc/paper/2018/hash/a2b8a85a29b2d64ad6f47275bf136
Abstract.html.

Cheng, J. and Vasconcelos, N. (2021). "Learning Deep Classifiers Consistent With Fine-Grained Novelty Detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1664–1673.

Choi, Y., Choi, J. P., El-Khamy, M., and Lee, J. (2020). "Data-Free Network Quantization With Adversarial Knowledge Distillation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. IEEE, pp. 3047–3057. DOI: 10.1109/CVPRW50498.2020.00363. URL: https://doi.org/10.1109/CVPRW50498.2020.00363.

Chung, I., Park, S., Kim, J., and Kwak, N. (2020). "Feature-map-level Online Adversarial Knowledge Distillation". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. III and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 2006–2015. URL: http://proceedings.mlr.press/v119/chung20a.html.

Cioppa, A., Deliege, A., Istasse, M., De Vleeschouwer, C., and Van Droogenbroeck, M. (2019). "Arthus: Adaptive real-time human segmentation in sports through online distillation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. (2018). "Deep Learning for Classical Japanese Literature". In: *CoRR* abs/1812.01718. arXiv: 1812.01718. URL: http://arxiv.org/abs/1812.01718.

Coates, A., Ng, A., and Lee, H. (2011). "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223.

Collin, A. and Vleeschouwer, C. D. (2020). “Improved anomaly detection by training an autoencoder with skip connections on images corrupted with Stain-shaped noise”. In: *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*. IEEE, pp. 7915–7922. DOI: 10.1109/ICPR48806.2021.9412842. URL: https://doi.org/10.1109/ICPR48806.2021.9412842.

Cortes, C. and Vapnik, V. (1995). “Support-vector networks”. In: *Machine learning* 20.3, pp. 273–297.

Dawer, G., Guo, Y., and Barbu, A. (2020). “Generating Compact Tree Ensembles via Annealing”. In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206593. URL: https://doi.org/10.1109/IJCNN48605.2020.9206593.

De Vleeschouwer, C., Legrand, A., Jacques, L., and Hebert, M. (2015). “Mitigating memory requirements for random trees/ferns”. In: *Image Processing (ICIP), 2015 IEEE International Conference on*. IEEE, pp. 227–231.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848. URL: https://doi.org/10.1109/CVPR.2009.5206848.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and Freitas, N. de (2013). “Predicting Parameters in Deep Learning”. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, pp. 2148–2156. URL: https://proceedings.neurips.cc/paper/2013/hash/7fec306d1e665bc9c748b5d2b99a6e97-Abstract.html.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). “Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 1269–1277. URL: https://proceedings.neurips.cc/paper/2014/hash/2afe4567e1bf64d32a5527244d104cea-Abstract.html.

DeVries, T. and Taylor, G. W. (2018). “Learning confidence for out-of-distribution detection in neural networks”. In: *arXiv preprint arXiv:1802.04865*.

Domingos, P. (1997). “Knowledge acquisition from examples via multiple models”. In: *Machine learning-international workshop then conference*. Morgan Kaufmann publishers, INC., pp. 98–106.

— (1999). “The role of Occam’s razor in knowledge discovery”. In: *Data mining and knowledge discovery* 3.4, pp. 409–425.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). “DeCAF: A Deep Convolutional Activation Feature for Generic

Visual Recognition". In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 647–655. URL: http://proceedings.mlr.press/v32/donahue14.html.

Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. (2019). "Gradient Descent Finds Global Minima of Deep Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 1675–1685. URL: http://proceedings.mlr.press/v97/du19c.html.

Duchi, J., Hazan, E., and Singer, Y. (2011). "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7.

Dumoulin, V. and Visin, F. (2016). "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints*. eprint: 1603.07285.

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al. (2004). "Least angle regression". In: *Annals of statistics* 32.2, pp. 407–499.

Elisha, O. and Dekel, S. (2016). "Wavelet decompositions of Random Forests-smoothness analysis, sparse approximation and applications". In: *Journal of Machine Learning Research* 17.198, pp. 1–38.

Erasmus, A., Brunet, T. D., and Fisher, E. (2020). "What is Interpretability?" In: *Philosophy & Technology*, pp. 1–30.

Fisher, R. A. (1936). "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2, pp. 179–188.

Fleuret, F. (2021). *EE559 Deep Learning, EPFL*. https://fleuret.org/dlc/. Accessed: 2021-08-06.

Frankle, J. and Carbin, M. (2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=rJl-b3RcF7.

Fredrikson, M., Jha, S., and Ristenpart, T. (2015). "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by I. Ray, N. Li, and C. Kruegel. ACM, pp. 1322–1333. DOI: 10.1145/2810103.2813677. URL: https://doi.org/10.1145/2810103.2813677.

Freund, Y. and Schapire, R. E. (1995). "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European conference on computational learning theory*. Springer, pp. 23–37.

Friedman, J., Hastie, T., and Tibshirani, R. (2001a). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

— (2001b). *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2000). "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)". In: *Annals of statistics* 28.2, pp. 337–407.

Friedman, J. H. (1991). "Multivariate adaptive regression splines". In: *The annals of statistics*, pp. 1–67.

— (2001a). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*, pp. 1189–1232.

— (2001b). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*, pp. 1189–1232.

Friedman, J. H. and Popescu, B. E. (2008). "Predictive learning via rule ensembles". In: *The Annals of Applied Statistics* 2.3, pp. 916–954.

Gal, Y. and Ghahramani, Z. (2016). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by M. Balcan and K. Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1050–1059. URL: http://proceedings.mlr.press/v48/gal16.html.

Geifman, Y. and El-Yaniv, R. (2019). "SelectiveNet: A Deep Neural Network with an Integrated Reject Option". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2151–2159. URL: http://proceedings.mlr.press/v97/geifman19a.html.

Gettier, E. L. (1963). "Is Justified True Belief Knowledge". In: *Analysis* 23(6), pp. 121–123. DOI: https://doi.org/10.2307/3326922.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). "Extremely randomized trees". In: *Machine learning* 63.1, pp. 3–42.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2021). "A Survey of Quantization Methods for Efficient Neural Network Inference". In: *CoRR* abs/2103.13630. arXiv: 2103.13630. URL: https://arxiv.org/abs/2103.13630.

Glorot, X. and Bengio, Y. (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Ed. by Y. W. Teh and D. M. Titterington. Vol. 9. JMLR Proceedings. JMLR.org, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Golan, I. and El-Yaniv, R. (2018). "Deep anomaly detection using geometric transformations". In: *Advances in Neural Information Processing Systems*, pp. 9758–9769.

Gong, C., Chang, X., Fang, M., and Yang, J. (2018). "Teaching Semi-Supervised Classifier via Generalized Distillation". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by J. Lang. ijcai.org, pp. 2156–2162. DOI: 10.24963/ijcai.2018/298. URL: https://doi.org/10.24963/ijcai.2018/298.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. Vol. 27. Curran

Associates, Inc., pp. 2672–2680. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). “Explaining and Harnessing Adversarial Examples”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1412.6572.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. (2021). “Knowledge Distillation: A Survey”. In: *Int. J. Comput. Vis.* 129.6, pp. 1789–1819. DOI: 10.1007/s11263-021-01453-z. URL: https://doi.org/10.1007/s11263-021-01453-z.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017a). “On calibration of modern neural networks”. In: *International Conference on Machine Learning*. PMLR, pp. 1321–1330.

Guo, H., Li, Y., Shang, J., Mingyun, G., Yuanyue, H., and Bing, G. (2017b). “Learning from class-imbalanced data: Review of methods and applications”. In: *Expert Syst. Appl.* 73, pp. 220–239. DOI: 10.1016/j.eswa.2016.12.035. URL: https://doi.org/10.1016/j.eswa.2016.12.035.

Guyon, I., Gunn, S. R., Ben-Hur, A., and Dror, G. (2004). “Result Analysis of the NIPS 2003 Feature Selection Challenge.” In: *NIPS*. Vol. 4, pp. 545–552.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). “Learning both Weights and Connections for Efficient Neural Networks”. In: *CoRR* abs/1506.02626. arXiv: 1506.02626. URL: http://arxiv.org/abs/1506.02626.

Haroush, M., Hubara, I., Hoffer, E., and Soudry, D. (2020). “The Knowledge Within: Methods for Data-Free Model Compression”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, pp. 8491–8499. DOI: 10.1109/CVPR42600.2020.00852. URL: https://doi.org/10.1109/CVPR42600.2020.00852.

Harrison Jr, D. and Rubinfeld, D. L. (1978). “Hedonic housing prices and the demand for clean air”. In: *Journal of environmental economics and management* 5.1, pp. 81–102.

Hassibi, B., Stork, D. G., and Wolff, G. J. (1993). “Optimal brain surgeon and general network pruning”. In: *IEEE international conference on neural networks*. IEEE, pp. 293–299.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Heinecke, A., Ho, J., and Hwang, W. (2020). “Refinement and Universal Approximation via Sparsely Connected ReLU Convolution Nets”. In: *IEEE Signal Process. Lett.* 27, pp. 1175–1179. DOI: 10.1109/LSP.2020.3005051. URL: https://doi.org/10.1109/LSP.2020.3005051.

Hendrycks, D. and Gimpel, K. (2017). “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Hkg4TI9xl.

Hendrycks, D., Mazeika, M., and Dietterich, T. G. (2019). “Deep Anomaly Detection with Outlier Exposure”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=HyxCxhRcY7.

Heo, B., Lee, M., Yun, S., and Choi, J. Y. (2019a). “Knowledge Distillation with Adversarial Samples Supporting Decision Boundary”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, pp. 3771–3778. DOI: 10.1609/aaai.v33i01.33013771. URL: https://doi.org/10.1609/aaai.v33i01.33013771.

— (2019b). “Knowledge Transfer via Distillation of Activation Boundaries Formed by Hidden Neurons”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, pp. 3779–3787. DOI: 10.1609/aaai.v33i01.33013779. URL: https://doi.org/10.1609/aaai.v33i01.33013779.

Hinton, G., Vinyals, O., and Dean, J. (2015). “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531*.

Hoerl, A. E. and Kennard, R. W. (1970). “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1, pp. 55–67.

Hornik, K., Stinchcombe, M. B., and White, H. (1989). “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5, pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8. URL: https://doi.org/10.1016/0893-6080(89)90020-8.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861. arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

Hsu, Y.-C., Shen, Y., Jin, H., and Kira, Z. (2020). “Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10951–10960.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708.

Hüllermeier, E. and Waegeman, W. (2021). “Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods”. In: *Machine Learning* 110.3, pp. 457–506.

Huynh-Thu, V. A., Irrthum, A., Wehenkel, L., and Geurts, P. (2010). “Inferring Regulatory Networks from Expression Data Using Tree-Based Methods”. In: *PLoS ONE* 5.9, e12776.

Ioffe, S. and Szegedy, C. (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings* *of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by F. R. Bach and D. M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). "Speeding up Convolutional Neural Networks with Low Rank Expansions". In: *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*. Ed. by M. F. Valstar, A. P. French, and T. P. Pridmore. BMVA Press. URL: http://www.bmva.org/bmvc/2014/papers/paper073/index.html.

Jiang, J. (2008). "A literature survey on domain adaptation of statistical classifiers". In: 3.1-12, p. 3.

Johnson, R. and Zhang, T. (2014). "Learning nonlinear functions using regularized greedy forest". In: *IEEE transactions on pattern analysis and machine intelligence* 36.5, pp. 942–954.

Joly, A., Schnitzler, F., Geurts, P., and Wehenkel, L. (2012). "L1-based compression of random forest models". In: *20th European Symposium on Artificial Neural Networks*.

Kardan, N., Sharma, A., and Stanley, K. O. (2021). "Towards Consistent Predictive Confidence through Fitted Ensembles". In: *CoRR* abs/2106.12070. arXiv: 2106.12070. URL: https://arxiv.org/abs/2106.12070.

Kim, J., Park, S., and Kwak, N. (2018). "Paraphrasing Complex Network: Network Compression via Factor Transfer". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 2765–2774. URL: https://proceedings.neurips.cc/paper/2018/hash/6d9cb7de5e8ac30bd5e8734bc96a35c1-Abstract.html.

Kimura, A., Ghahramani, Z., Takeuchi, K., Iwata, T., and Ueda, N. (2018). "Few-shot learning of neural networks from scratch by pseudo example optimization". In: *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*. BMVA Press, p. 105. URL: http://bmvc2018.org/contents/papers/0366.pdf.

Kindermans, P.-J., Hooker, S., Adebayo, J., Alber, M., Schütt, K. T., Dähne, S., Erhan, D., and Kim, B. (2019). "The (Un)reliability of Saliency Methods". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller. Springer International Publishing. Chap. 14, pp. 267–280.

Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R., Erhan, D., Kim, B., and Dähne, S. (2017). "Learning how to explain neural networks: PatternNet and PatternAttribution". In: *ArXiv e-prints*. eprint: 1705.05598.

Kingma, D. P. and Ba, J. (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1412.6980.

Kobyzev, I., Prince, S., and Brubaker, M. (2020). "Normalizing flows: An introduction and review of current methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kohavi, R., John, G. H., et al. (1997). "Wrappers for feature subset selection". In: *Artificial intelligence* 97.1-2, pp. 273–324.

Krizhevsky, A., Hinton, G., et al. (2009). "Learning multiple layers of features from tiny images". In:

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25, pp. 1097–1105.

Kumar, N., Hanfeld, P., Hecht, M., Bussmann, M., Gumhold, S., and Hoffmann, N. (2021). "InFlow: Robust outlier detection utilizing Normalizing Flows". In: *CoRR* abs/2106.12894. arXiv: 2106.12894. URL: https://arxiv.org/abs/2106.12894.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, pp. 6402–6413. URL: https://proceedings.neurips.cc/paper/2017/hash/9ef2ed4b7fd2c810847ffa5fa85bce Abstract.html.

Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J. K., Leach, K., Laurenzano, M. A., Tang, L., and Mars, J. (2019). "An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by K. Inui, J. Jiang, V. Ng, and X. Wan. Association for Computational Linguistics, pp. 1311–1316. DOI: 10.18653/v1/D19-1131. URL: https://doi.org/10.18653/v1/D19-1131.

Le, Y. and Yang, X. (2015). "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7, p. 3.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I. V., and Lempitsky, V. S. (2015). "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1412.6553.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

— (1998b). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Y., Denker, J. S., and Solla, S. A. (1989). "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*. Ed. by D. S. Touretzky. Morgan Kaufmann, pp. 598–605. URL: http://papers.nips.cc/paper/250-optimal-brain-damage.

Lee, D., Yu, S., and Yu, H. (2020). "Multi-Class Data Description for Out-of-distribution Detection". In: *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. Ed. by R. Gupta, Y. Liu, J. Tang, and B. A. Prakash. ACM, pp. 1362–1370. DOI: 10.1145/3394486.3403189. URL: https://doi.org/10.1145/3394486.3403189.

Lee, J. and AlRegib, G. (2020). "Gradients as a Measure of Uncertainty in Neural Networks". In: *IEEE International Conference on Image Processing, ICIP 2020, Abu Dhabi, United Arab Emirates, October 25-28, 2020*. IEEE, pp. 2416–2420. DOI: 10.1109/ICIP40778.2020.9190679. URL: https://doi.org/10.1109/ICIP40778.2020.9190679.

Lee, K., Lee, H., Lee, K., and Shin, J. (2018a). "Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=ryiAv2xAZ.

Lee, K., Lee, K., Lee, H., and Shin, J. (2018b). "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 7167–7177. URL: https://proceedings.neurips.cc/paper/2018/hash/abdeb6f575ac5c6676b747bca8d09cc2-Abstract.html.

Lee, S. H., Kim, D. H., and Song, B. C. (2018). "Self-supervised knowledge distillation using singular value decomposition". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 335–350.

Lei, Y., Yang, B., Jiang, X., Jia, F., Li, N., and Nandi, A. K. (2020). "Applications of machine learning to machine fault diagnosis: A review and roadmap". In: *Mechanical Systems and Signal Processing* 138, p. 106587.

Leray, P. and Gallinari, P. (1999). "Feature selection with neural networks". In: *Behaviormetrika* 26.1, pp. 145–166.

Li, J., Zhao, R., Huang, J., and Gong, Y. (2014). "Learning small-size DNN with output-distribution-based criteria". In: *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*. Ed. by H. Li, H. M. Meng, B. Ma, E. Chng, and L. Xie. ISCA, pp. 1910–1914. URL: http://www.isca-speech.org/archive/interspeech_2014/i14_1910.html.

Li, T., Li, J., Liu, Z., and Zhang, C. (2020). "Few Sample Knowledge Distillation for Efficient Network Compression". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, pp. 14627–14635. DOI: 10.1109/CVPR42600.2020.01465. URL: https://doi.org/10.1109/CVPR42600.2020.01465.

Li, Y., Chen, C.-Y., and Wasserman, W. W. (2015). "Deep Feature Selection: Theory and Application to Identify Enhancers and Promoters". In: *Proceedings of RECOMB2015*, pp. 205–217.

Li, Y. and Vasconcelos, N. (2020). "Background Data Resampling for Outlier-Aware Classification". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, pp. 13215–13224. DOI: 10.1109/CVPR42600.2020.01323. URL: https://doi.org/10.1109/CVPR42600.2020.01323.

Liang, S., Li, Y., and Srikant, R. (2018). "Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=H1VGkIxRZ.

Lin, M., Chen, Q., and Yan, S. (2013). "Network in network". In: *arXiv preprint arXiv:1312.4400*.

Liu, B., Wang, M., Foroosh, H., Tappen, M. F., and Pensky, M. (2015). "Sparse Convolutional Neural Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, pp. 806–814. DOI: 10.1109/CVPR.2015.7298681. URL: https://doi.org/10.1109/CVPR.2015.7298681.

Liu, F. T., Ting, K. M., and Zhou, Z. (2008). "Isolation Forest". In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*. IEEE Computer Society, pp. 413–422. DOI: 10.1109/ICDM.2008.17. URL: https://doi.org/10.1109/ICDM.2008.17.

Liu, W., Wang, X., Owens, J. D., and Li, Y. (2020). "Energy-based Out-of-distribution Detection". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/f5496252609c43eb8a3d147ab9b9c006-Abstract.html.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). "Learning Efficient Convolutional Networks through Network Slimming". In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, pp. 2755–2763. DOI: 10.1109/ICCV.2017.298. URL: https://doi.org/10.1109/ICCV.2017.298.

Louppe, G., Wehenkel, L., Sutera, A., and Geurts, P. (2013). "Understanding variable importances in forests of randomized trees". In: *Advances in Neural Information Processing Systems 26*, pp. 431–439.

Luxburg, U. von and Schölkopf, B. (2011). "Statistical Learning Theory: Models, Concepts, and Results". In: *Inductive Logic*. Ed. by D. M. Gabbay, S. Hartmann, and J. Woods. Vol. 10. Handbook of the History of Logic. Elsevier, pp. 651–706. DOI: 10.1016/B978-0-444-52936-7.50016-1. URL: https://doi.org/10.1016/B978-0-444-52936-7.50016-1.

Ma, N., Zhang, X., Zheng, H., and Sun, J. (2018). "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*. Ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Vol. 11218. Lecture Notes in Computer Science. Springer, pp. 122–138. DOI: 10.1007/978-3-030-01264-9_8. URL: https://doi.org/10.1007/978-3-030-01264-9_8.

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., and Dally, W. J. (2017). "Exploring the Granularity of Sparsity in Convolutional Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 1927–1934. DOI: 10.1109/CVPRW.2017.241. URL: https://doi.org/10.1109/CVPRW.2017.241.

Marbach, D., Costello, J. C., Küffner, R., Vega, N., Prill, R. J., Camacho, D. M., Allison, K. R., the DREAM5 Consortium, Kellis, M., Collins, J. J., and Stolovitzky, G. (2012). "Wisdom of crowds for robust gene network inference". In: *Nature Methods* 9.8, pp. 796–804.

Marbach, D., Schaffter, T., Mattiussi, C., and Floreano, D. (2009). "Generating Realistic In Silico Gene Networks for Performance Assessment of Reverse Engineering Methods." In: *Journal of Computational Biology* 16(2), pp. 229–239.

Meinshausen, N. (2010). "Node harvest". In: *The Annals of Applied Statistics*, pp. 2049–2072.

Meinshausen, N. et al. (2009). "Forest garrote". In: *Electronic Journal of Statistics* 3, pp. 1288–1304.

Menke, J. E. and Martinez, T. R. (2009). "Artificial neural network reduction through oracle learning". In: *Intelligent Data Analysis* 13.1, pp. 135–149.

Mercatelli, D., Scalambra, L., Triboli, L., Ray, F., and Giorgi, F. M. (2020). "Gene regulatory network inference resources: A practical overview". In: *Biochimica et Biophysica Acta (BBA)-Gene Regulatory Mechanisms* 1863.6, p. 194430.

Micaelli, P. and Storkey, A. J. (2019). "Zero-shot Knowledge Transfer via Adversarial Belief Matching". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, pp. 9547–9557. URL: https://proceedings.neurips.cc/paper/2019/hash/fe663a72b27bdc613873fbbb512f6f67-Abstract.html.

Mirzadeh, S., Farajtabar, M., Li, A., Levine, N., Matsukawa, A., and Ghasemzadeh, H. (2020). "Improved Knowledge Distillation via Teacher Assistant". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, pp. 5191–5198. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5963.

Mittal, D., Bhardwaj, S., Khapra, M. M., and Ravindran, B. (2018). "Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks". In: *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018*. IEEE Computer Society, pp. 848–857. DOI: 10.1109/WACV.2018.00098. URL: https://doi.org/10.1109/WACV.2018.00098.

Mohseni, S., Pitale, M., Yadawa, J. B. S., and Wang, Z. (2020). "Self-Supervised Learning for Generalizable Out-of-Distribution Detection". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, pp. 5216–5223. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5966.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). "Pruning Convolutional Neural Networks for Resource Efficient Inference". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SJGCiw5gl.

Molnar, C. (2020). *Interpretable machine learning*.

Montavon, G., Samek, W., and Müller, K.-R. (2018). "Methods for interpreting and understanding deep neural networks". In: *Digital Signal Processing* 73.Supplement C, pp. 1 –15.

Moreno-Torres, J. G., Raeder, T., Alaíz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). "A unifying view on dataset shift in classification". In: *Pattern Recognit.* 45.1, pp. 521–530. DOI: 10.1016/j.patcog.2011.06.019. URL: https://doi.org/10.1016/j.patcog.2011.06.019.

Mormont, R., Geurts, P., and Marée, R. (2018). "Comparison of Deep Transfer Learning Strategies for Digital Pathology". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 2262–2271. DOI: 10.1109/CVPRW.2018.00303. URL: http://openaccess.thecvf.com/content_cvpr_2018_workshops/w44/html/Mormont_Comparison_of_Deep_CVPR_2018_paper.html.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). "Definitions, methods, and applications in interpretable machine learning". In: *Proceedings of the National Academy of Sciences* 116.44, pp. 22071–22080.

Nair, V. and Hinton, G. E. (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. Ed. by J. Fürnkranz and T. Joachims. Omnipress, pp. 807–814. URL: https://icml.cc/Conferences/2010/papers/432.pdf.

Nakamura, A. and Sakurada, K. (2019). "An Algorithm for Reducing the Number of Distinct Branching Conditions in a Decision Forest". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*. Ed. by U. Brefeld, É. Fromont, A. Hotho, A. J. Knobbe, M. H. Maathuis, and C. Robardet. Vol. 11906. Lecture Notes in Computer Science. Springer, pp. 578–589. DOI: 10.1007/978-3-030-46150-8_34. URL: https://doi.org/10.1007/978-3-030-46150-8_34.

Nayak, G. K., Mopuri, K. R., Shaj, V., Radhakrishnan, V. B., and Chakraborty, A. (2019). "Zero-Shot Knowledge Distillation in Deep Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019,* *9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 4743–4751. URL: http://proceedings.mlr.press/v97/nayak19a.html.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). "Reading digits in natural images with unsupervised feature learning". In:

Nguyen, T., Novak, R., Xiao, L., and Lee, J. (2021). "Dataset Distillation with Infinitely Wide Convolutional Networks". In: *CoRR* abs/2107.13034. arXiv: 2107.13034. URL: https://arxiv.org/abs/2107.13034.

Nie, Z., Lin, B., Huang, S., Ramakrishnan, N., Fan, W., and Ye, J. (2017). "Pruning Decision Trees via Max-Heap Projection". In: *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017*. Ed. by N. V. Chawla and W. Wang. SIAM, pp. 10–18. DOI: 10.1137/1.9781611974973.2. URL: https://doi.org/10.1137/1.9781611974973.2.

Osawa, K., Sekiya, A., Naganuma, H., and Yokota, R. (2017). "Accelerating Matrix Multiplication in Deep Learning by Using Low-Rank Approximation". In: *2017 International Conference on High Performance Computing & Simulation, HPCS 2017, Genoa, Italy, July 17-21, 2017*. IEEE, pp. 186–192. DOI: 10.1109/HPCS.2017.37. URL: https://doi.org/10.1109/HPCS.2017.37.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). "Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift". In: *Advances in Neural Information Processing Systems*, pp. 13991–14002.

Painsky, A. and Rosset, S. (2019). "Lossless Compression of Random Forests". In: *J. Comput. Sci. Technol.* 34.2, pp. 494–506. DOI: 10.1007/s11390-019-1921-0. URL: https://doi.org/10.1007/s11390-019-1921-0.

Pan, S. J. and Yang, Q. (2009). "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10, pp. 1345–1359.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113, pp. 54–71.

Park, W., Kim, D., Lu, Y., and Cho, M. (2019). "Relational Knowledge Distillation". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, pp. 3967–3976. DOI: 10.1109/CVPR.2019.00409. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Park_Relational_Knowledge_Distillation_CVPR_2019_paper.html.

Parzen, E. (1962). "On estimation of a probability density function and mode". In: *The annals of mathematical statistics* 33.3, pp. 1065–1076.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). "Automatic differentiation in PyTorch". In:

Pearl, J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

Pearl, J. et al. (2000). "Models, reasoning and inference". In: *Cambridge, UK: CambridgeUniversityPress* 19.

Pearlmutter, B. A. (1994). "Fast Exact Multiplication by the Hessian". In: *Neural Comput.* 6.1, pp. 147–160. DOI: 10.1162/neco.1994.6.1.147. URL: https://doi.org/10.1162/neco.1994.6.1.147.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.Oct, pp. 2825–2830.

Pedregosa *et al.*, F. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Peterson, A. H. and Martinez, T. R. (2009). "Reducing Decision Tree Ensemble Size Using Parallel Decision DAGS". In: *International Journal on Artificial Intelligence Tools* 18.04, pp. 613–620.

Poincaré, H. (1914). "Science et méthode (1908)". In: *Book II* 2.

Polonik, W. (1997). "Minimum volume sets and generalized quantile processes". In: *Stochastic processes and their applications* 69.1, pp. 1–24.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. (2016). "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pp. 3360–3368. URL: https://proceedings.neurips.cc/paper/2016/hash/148510031349642de5ca0c544f31b2ef-Abstract.html.

Quintanilha, I. M., ME Filho, R. de, Lezama, J., Delbracio, M., and Nunes, L. O. (2018). "Detecting Out-Of-Distribution Samples Using Low-Order Deep Features Statistics". In:

Rabanser, S., Günnemann, S., and Lipton, Z. C. (2019). "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, pp. 1394–1406. URL: https://proceedings.neurips.cc/paper/2019/hash/846c260d715e5b854ffad5f70a516c88-Abstract.html.

Raghu, M., Poole, B., Kleinberg, J. M., Ganguli, S., and Sohl-Dickstein, J. (2017). "On the Expressive Power of Deep Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 2847–2854. URL: http://proceedings.mlr.press/v70/raghu17a.html.

Rasmussen, C. E. and Ghahramani, Z. (2001). "Occam's razor". In: *Advances in neural information processing systems*, pp. 294–300.

Ren, J., Liu, P. J., Fertig, E., Snoek, J., Poplin, R., DePristo, M. A., Dillon, J. V., and Lakshminarayanan, B. (2019). “Likelihood Ratios for Out-of-Distribution Detection”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, pp. 14680–14691. URL: https://proceedings.neurips.cc/paper/2019/hash/1e79596878b2320cac26dd792a6c51c9-Abstract.html.

Ren, S., Cao, X., Wei, Y., and Sun, J. (2015). “Global refinement of random forest”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 723–730.

Roady, R., Hayes, T. L., Kemker, R., Gonzales, A., and Kanan, C. (2019). “Are Out-of-Distribution Detection Methods Effective on Large-Scale Datasets?” In: *CoRR* abs/1910.14034. arXiv: 1910.14034. URL: http://arxiv.org/abs/1910.14034.

Rodgers, D. P. (1985). “Improvements in multiprocessor system design”. In: *ACM SIGARCH Computer Architecture News* 13.3, pp. 225–231.

Rokach, L. (2016). “Decision forest: Twenty years of research”. In: *Information Fusion* 27, pp. 111–125.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). “FitNets: Hints for Thin Deep Nets”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1412.6550.

Rosenblatt, F. (1958). “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6, p. 386.

Russell, B. (2001). *The problems of philosophy*. OUP Oxford.

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. (2013). “Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. IEEE, pp. 6655–6659. DOI: 10.1109/ICASSP.2013.6638949. URL: https://doi.org/10.1109/ICASSP.2013.6638949.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. (2018). “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.

Sankararaman, K. A., De, S., Xu, Z., Huang, W. R., and Goldstein, T. (2020). “The Impact of Neural Network Overparameterization on Gradient Confusion and Stochastic Gradient Descent”. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 8469–8479. URL: http://proceedings.mlr.press/v119/sankararaman20a.html.

Sastry, C. S. and Oore, S. (2019). “Detecting Out-of-Distribution Examples with In-distribution Examples and Gram Matrices”. In: *CoRR* abs/1912.12510. arXiv: 1912.12510. URL: http://arxiv.org/abs/1912.12510.

Sau, B. B. and Balasubramanian, V. N. (2016). “Deep model compression: Distilling knowledge from noisy teachers”. In: *arXiv preprint arXiv:1610.09650*.

Scardapane, S., Comminiello, D., Hussain, A., and Uncini, A. (2017). “Group sparse regularization for deep neural networks”. In: *Neurocomputing* 241, pp. 81–89. DOI: 10.1016/j.neucom.2017.02.029. URL: https://doi.org/10.1016/j.neucom.2017.02.029.

Scheirer, W. J., Rezende Rocha, A. de, Sapkota, A., and Boult, T. E. (2012). “Toward open set recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.7, pp. 1757–1772.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C. (1999). “Support Vector Method for Novelty Detection”. In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. The MIT Press, pp. 582–588. URL: http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.

Scott, C. D. and Nowak, R. D. (2005). “Learning Minimum Volume Sets”. In: *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pp. 1209–1216. URL: https://proceedings.neurips.cc/paper/2005/hash/d3d80b656929a5bc0fa34381bf42fbdd-Abstract.html.

Scudder, H. (1965). “Probability of error of some adaptive pattern-recognition machines”. In: *IEEE Transactions on Information Theory* 11.3, pp. 363–371. DOI: 10.1109/TIT.1965.1053799.

Sehwag, V., Chiang, M., and Mittal, P. (2021). “SSD: A Unified Framework for Self-Supervised Outlier Detection”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=v5gjXpmR8J.

Shafaei, A., Schmidt, M., and Little, J. J. (2018). “Does your model know the digit 6 is not a cat? A less biased evaluation of” outlier” detectors”. In: *arXiv preprint arXiv:1809.04729*.

Shafer, G. and Vovk, V. (2008). “A Tutorial on Conformal Prediction.” In: *Journal of Machine Learning Research* 9.3.

Shawe-Taylor, B. G. J. (2019). “A Primer on PAC-Bayesian Learning”. In:

Shi, M., Qin, F., Ye, Q., Han, Z., and Jiao, J. (2017). “A scalable convolutional neural network for task-specified scenarios via knowledge distillation”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*. IEEE, pp. 2467–2471. DOI: 10.1109/ICASSP.2017.7952600. URL: https://doi.org/10.1109/ICASSP.2017.7952600.

Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. (2013). “Decision jungles: Compact and rich models for classification”. In: *Advances in Neural Information Processing Systems*, pp. 234–242.

Shrikumar, A., Greenside, P., and Kundaje, A. (2017). “Learning Important Features through Propagating Activation Differences”. In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 3145–3153.

Shu, L., Xu, H., and Liu, B. (2017). “DOC: Deep Open Classification of Text Documents”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by M. Palmer, R. Hwa, and S. Riedel. Association for Computational Linguistics, pp. 2911–2916. DOI: 10.18653/v1/d17-1314. URL: https://doi.org/10.18653/v1/d17-1314.

Sifre, L. and Mallat, P. S. (2014). “Rigid-Motion Scattering For Image Classification Author”. In: *English. Supervisor: Prof. Stéphane Mallat. Ph. D. Thesis. Ecole Polytechnique*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *ArXiv e-prints*. arXiv: 1312.6034 [cs.CV].

Simonyan, K. and Zisserman, A. (2015). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1409.1556.

Souad, T. Z. and Abdelkader, A. (2019). “Pruning of Random Forests: a diversity-based heuristic measure to simplify a random forest ensemble”. In: *INFOCOMP: Journal of Computer Science* 18.1.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). “Striving for Simplicity: The All Convolutional Net”. In: *ArXiv e-prints*. eprint: 1412.6806.

Srinivas, S. and Fleuret, F. (2018). “Knowledge Transfer with Jacobian Matching”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4730–4738. URL: http://proceedings.mlr.press/v80/srinivas18a.html.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *J. Mach. Learn. Res.* 15.1, pp. 1929–1958. URL: http://dl.acm.org/citation.cfm?id=2670313.

Sucholutsky, I. and Schonlau, M. (2019). “Soft-label dataset distillation and text dataset distillation”. In: *arXiv preprint arXiv:1910.02551*.

Sundararajan, M., Taly, A., and Yan, Q. (2017). “Axiomatic Attribution for Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 3319–3328.

Sutera, A. (2021). “Importance measures derived from random forests: characterisation and extension”. In: *arXiv preprint arXiv:2106.09473*.

Swaminathan, S., Garg, D., Kannan, R., and Andrès, F. (2020). “Sparse low rank factorization for deep neural network compression”. In: *Neurocomputing* 398, pp. 185–196. DOI: 10.1016/j.neucom.2020.02.035. URL: https://doi.org/10.1016/j.neucom.2020.02.035.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). “Going deeper with convolutions”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594. URL: https://doi.org/10.1109/CVPR.2015.7298594.

Tang, S., Feng, L., Shao, W., Kuang, Z., Zhang, W., and Lu, Z. (2019). “Learning Efficient Detector with Semi-supervised Adaptive Distillation”. In: *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*. BMVA Press, p. 215. URL: https://bmvc2019.org/wp-content/uploads/papers/0145-paper.pdf.

Tanno, R., Arulkumaran, K., Alexander, D. C., Criminisi, A., and Nori, A. V. (2019). “Adaptive Neural Trees”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6166–6175. URL: http://proceedings.mlr.press/v97/tanno19a.html.

Tavallali, P., Tavallali, P., and Singhal, M. (2019). “Optimization of Hierarchical Regression Model with Application to Optimizing Multi-Response Regression K-ary Trees”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, pp. 5133–5142. DOI: 10.1609/aaai.v33i01.33015133. URL: https://doi.org/10.1609/aaai.v33i01.33015133.

Theis, L., Korshunova, I., Tejani, A., and Huszár, F. (2018). “Faster gaze prediction with dense networks and Fisher pruning”. In: *CoRR* abs/1801.05787. arXiv: 1801.05787. URL: http://arxiv.org/abs/1801.05787.

Thudumu, S., Branch, P., Jin, J., and Singh, J. J. (2020). “A comprehensive survey of anomaly detection techniques for high dimensional big data”. In: *Journal of Big Data* 7.1, pp. 1–30.

Tibshirani, R. (1996). “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1, pp. 267–288.

Togootogtokh, E. and Amartuvshin, A. (2018). “Deep learning approach for very similar objects recognition application on chihuahua and muffin problem”. In: *arXiv preprint arXiv:1801.09573*.

Tornay, S. C. (1938). “Ockham: Studies and selections”. In:

Tsamardinos, I. and Aliferis, C. F. (2003). “Towards Principled Feature Selection: Relevancy, Filters and Wrappers”. In: *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, AISTATS 2003, Key West, Florida, USA, January 3-6, 2003*. Ed. by C. M. Bishop and B. J. Frey. Society for Artificial Intelligence and Statistics. URL: http://research.microsoft.com/en-us/um/cambridge/events/aistats2003/proceedings/133.pdf.

Tsoumakas, G., Partalas, I., and Vlahavas, I. (2008). "A taxonomy and short review of ensemble selection". In: *ECAI 2008, workshop on supervised and unsupervised ensemble methods and their applications*, pp. 41–46.

Turing, A. (1950). "Computing Machinery and Intelligence". In: *Mind* 59.236, pp. 433–460.

Vapnik, V. (1998). *Statistical learning theory*. Wiley. ISBN: 978-0-471-03003-4.

Vecoven, N., Begon, J.-M., Sutera, A., Geurts, P., et al. (2020). "Nets versus trees for feature ranking and gene network inference". In: *International Conference on Discovery Science*. Springer, pp. 231–245.

Vens, C. and Costa, F. (2011). "Random forest based feature induction". In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, pp. 744–753.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nat.* 575.7782, pp. 350–354. DOI: `10.1038/s41586-019-1724-z`. URL: `https://doi.org/10.1038/s41586-019-1724-z`.

Vongkulbhisal, J., Vinayavekhin, P., and Scarzanella, M. V. (2019). "Unifying Heterogeneous Classifiers With Distillation". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, pp. 3175–3184. DOI: `10.1109/CVPR.2019.00329`. URL: `http://openaccess.thecvf.com/content\_CVPR\_2019/html/Vongkulbhisal\_Unifying\_Heterogeneous\_Classifiers\_With\_Distillation\_CVPR\_2019\_paper.html`.

Vyas, A., Jammalamadaka, N., Zhu, X., Das, D., Kaul, B., and Willke, T. L. (2018). "Out-of-distribution detection using an ensemble of self supervised leave-out classifiers". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 550–564.

Wang, H., Zhao, H., Li, X., and Tan, X. (2018a). "Progressive Blockwise Knowledge Distillation for Neural Network Acceleration". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by J. Lang. ijcai.org, pp. 2769–2775. DOI: `10.24963/ijcai.2018/384`. URL: `https://doi.org/10.24963/ijcai.2018/384`.

Wang, L. and Yoon, K. (2020). "Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks". In: *CoRR* abs/2004.05937. arXiv: `2004.05937`. URL: `https://arxiv.org/abs/2004.05937`.

Wang, S., Aggarwal, C. C., and Liu, H. (2017). "Using a Random Forest to Inspire a Neural Network and Improving on It". In: *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017*. Ed. by N. V. Chawla and W. Wang. SIAM, pp. 1–9. DOI:

10.1137/1.9781611974973.1. URL: https://doi.org/10.1137/1.9781611974973.1.

Wang, T., Zhu, J., Torralba, A., and Efros, A. A. (2018b). "Dataset Distillation". In: *CoRR* abs/1811.10959. arXiv: 1811.10959. URL: http://arxiv.org/abs/1811.10959.

Wang, X., Zhang, R., Sun, Y., and Qi, J. (2018c). "KDGAN: Knowledge Distillation with Generative Adversarial Networks." In: *NeurIPS*, pp. 783–794.

Watson, D. S. and Floridi, L. (2020). "The explanation game: a formal framework for interpretable machine learning". In: *Synthese*, pp. 1–32.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). "Learning Structured Sparsity in Deep Neural Networks". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pp. 2074–2082. URL: https://proceedings.neurips.cc/paper/2016/hash/41bfd20a38bb1b0bec75acf0845530a7-Abstract.html.

Wilson, A. G. and Izmailov, P. (2020). "Bayesian Deep Learning and a Probabilistic Perspective of Generalization". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/322f62469c5e3c7dc3e58f5a4d1ea399-Abstract.html.

Wolpert, D. H. (1995). "Off-training set error and a priori distinctions between learning algorithms". In: *Sante Fe Institute, Santa Fe, NM, USA, Tech. Rep. SFI-TR*, pp. 95–01.

Wolpert, D. H. (1996). "The Lack of A Priori Distinctions Between Learning Algorithms". In: *Neural Comput.* 8.7, pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341. URL: https://doi.org/10.1162/neco.1996.8.7.1341.

Wolpert, D. H. (2002). "The supervised learning no-free-lunch theorems". In: *Soft computing and industry*, pp. 25–42.

Worboys, M. F. and Duckham, M. (2004). *GIS: a computing perspective*. CRC press.

Xian, Y., Lampert, C. H., Schiele, B., and Akata, Z. (2019). "Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 41.9, pp. 2251–2265. DOI: 10.1109/TPAMI.2018.2857768. URL: https://doi.org/10.1109/TPAMI.2018.2857768.

Xiao, H., Rasul, K., and Vollgraf, R. (Aug. 28, 2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv: cs.LG/1708.07747 [cs.LG].

Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. (2017). "Aggregated Residual Transformations for Deep Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 5987–5995. DOI: 10.1109/CVPR.2017.634. URL: https://doi.org/10.1109/CVPR.2017.634.

Xu, Y., Wang, Y., Chen, H., Han, K., XU, C., Tao, D., and Xu, C. (2019). "Positive-Unlabeled Compression on the Cloud". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., pp. 2565–2574. URL: https://proceedings.neurips.cc/paper/2019/file/ac796a52db3f16bbdb6557d3d89d1c5a-Paper.pdf.

Xu, Z., Hsu, Y., and Huang, J. (2018). "Training Shallow and Thin Networks for Acceleration via Knowledge Distillation with Conditional Adversarial Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=BJbtuRRLM.

Yang, C., Yang, Z., Khattak, A. M., Yang, L., Zhang, W., Gao, W., and Wang, M. (2019). "Structured Pruning of Convolutional Neural Networks via L1 Regularization". In: *IEEE Access* 7, pp. 106385–106394. DOI: 10.1109/ACCESS.2019.2933032. URL: https://doi.org/10.1109/ACCESS.2019.2933032.

Yang, T., Howard, A. G., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. (2018). "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications". In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*. Ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Vol. 11214. Lecture Notes in Computer Science. Springer, pp. 289–304. DOI: 10.1007/978-3-030-01249-6_18. URL: https://doi.org/10.1007/978-3-030-01249-6_18.

Yang, Z., Moczulski, M., Denil, M., De Freitas, N., Smola, A., Song, L., and Wang, Z. (2015). "Deep fried convnets". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1476–1483.

Yim, J., Joo, D., Bae, J., and Kim, J. (2017). "A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 7130–7138. DOI: 10.1109/CVPR.2017.754. URL: https://doi.org/10.1109/CVPR.2017.754.

Yin, H., Molchanov, P., Alvarez, J. M., Li, Z., Mallya, A., Hoiem, D., Jha, N. K., and Kautz, J. (2020). "Dreaming to distill: Data-free knowledge transfer via DeepInversion". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8715–8724.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 3320–3328. URL: https://proceedings.neurips.cc/paper/2014/hash/375c71349b295fbe2dcdca9206f20a06-Abstract.html.

Yu, B. and Kumbier, K. (2019). “Three principles of data science: predictability, computability, and stability (PCS)”. In: *CoRR* abs/1901.08152. arXiv: 1901.08152. URL: http://arxiv.org/abs/1901.08152.

Yu, F., Zhang, Y., Song, S., Seff, A., and Xiao, J. (2015). “LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop”. In: *arXiv preprint arXiv:1506.03365*.

Yu, Q. and Aizawa, K. (2019). “Unsupervised Out-of-Distribution Detection by Maximum Classifier Discrepancy”. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, pp. 9517–9525. DOI: 10.1109/ICCV.2019.00961. URL: https://doi.org/10.1109/ICCV.2019.00961.

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., Gao, M., Lin, C., and Davis, L. S. (2018). “NISP: Pruning Networks Using Neuron Importance Score Propagation”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 9194–9203. DOI: 10.1109/CVPR.2018.00958. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Yu_NISP_Pruning_Networks_CVPR_2018_paper.html.

Yu, X., Liu, T., Wang, X., and Tao, D. (2017). “On Compressing Deep Models by Low Rank and Sparse Decomposition”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 67–76. DOI: 10.1109/CVPR.2017.15. URL: https://doi.org/10.1109/CVPR.2017.15.

Yuan, L., Tay, F. E. H., Li, G., Wang, T., and Feng, J. (2020). “Revisiting Knowledge Distillation via Label Smoothing Regularization”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, pp. 3902–3910. DOI: 10.1109/CVPR42600.2020.00396. URL: https://doi.org/10.1109/CVPR42600.2020.00396.

Zagoruyko, S. and Komodakis, N. (2016). “Wide Residual Networks”. In: *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. Ed. by R. C. Wilson, E. R. Hancock, and W. A. P. Smith. BMVA Press. URL: http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

— (2017). “Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Sks9_ajex.

Zednik, C. (2021). “Solving the black box problem: a normative framework for explainable artificial intelligence”. In: *Philosophy & Technology* 34.2, pp. 265–288.

Zeiler, M. D. and Fergus, R. (2014). “Visualizing and Understanding Convolutional Networks”. In: *Computer Vision – ECCV 2014*. Ed. by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars. Springer International Publishing, pp. 818–833.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2017). “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: *CoRR* abs/1707.01083. arXiv: 1707.01083. URL: http://arxiv.org/abs/1707.01083.

Zhang, X., Zou, J., He, K., and Sun, J. (2016). "Accelerating Very Deep Convolutional Networks for Classification and Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 38.10, pp. 1943–1955. DOI: 10.1109/TPAMI.2015.2502579. URL: https://doi.org/10.1109/TPAMI.2015.2502579.

Zharmagambetov, A. and Carreira-Perpiñán, M. Á. (2020). "Smaller, more accurate regression forests using tree alternating optimization". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 11398–11408. URL: http://proceedings.mlr.press/v119/zharmagambetov20a.html.

Zhou, D. (2018). "Universality of Deep Convolutional Neural Networks". In: *CoRR* abs/1805.10769. arXiv: 1805.10769. URL: http://arxiv.org/abs/1805.10769.

Zhou, G., Fan, Y., Cui, R., Bian, W., Zhu, X., and Gai, K. (2018). "Rocket Launching: A Universal and Efficient Framework for Training Well-Performing Light Net". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by S. A. McIlraith and K. Q. Weinberger. AAAI Press, pp. 4580–4587. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16090.

Zhu, J., Zou, H., Rosset, S., and Hastie, T. (2009). "Multi-class adaboost". In: *Statistics and its Interface* 2.3, pp. 349–360.

Zhu, M. and Gupta, S. (2018). "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Sy1iIDkPM.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2021). "A Comprehensive Survey on Transfer Learning". In: *Proc. IEEE* 109.1, pp. 43–76. DOI: 10.1109/JPROC.2020.3004555. URL: https://doi.org/10.1109/JPROC.2020.3004555.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). "Learning Transferable Architectures for Scalable Image Recognition". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 8697–8710. DOI: 10.1109/CVPR.2018.00907. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.html.

Zou, H. and Hastie, T. (2005). "Regularization and variable selection via the elastic net". In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2, pp. 301–320.

Zuo, Y. and Drummond, T. (2020). “Residual Likelihood Forests”. In: *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*. BMVA Press. URL: `https://www.bmvc2020-conference.com/assets/papers/0191.pdf`.