Ensembles of extremely randomized trees and some generic applications

Louis Wehenkel, Damien Ernst, Pierre Geurts
Department of Electrical Engineering and Computer Science
University of Liège - Sart-Tilman B28 - B-4000 Liège
Correspondence: L.Wehenkel@ulg.ac.be

Abstract - In this paper we present a new tree-based ensemble method called "Extra-Trees". This algorithm averages predictions of trees obtained by partitioning the input-space with randomly generated splits, leading to significant improvements of precision, and various algorithmic advantages, in particular reduced computational complexity and scalability. We also discuss two generic applications of this algorithm, namely for time-series classification and for the automatic inference of near-optimal sequential decision policies from experimental data.

Keywords - Automatic learning, robust supervised learning methods, time-series classification, learning of optimal control policies

1 INTRODUCTION

Classification and regression trees are well known methods used for data interpretation and statistical modeling. While these non-parametric and nonlinear methods are intrinsically robust to outliers, scalable to high-dimensional spaces and can cope with very large sample sizes, their main drawback is imprecision. The main factor of imprecision was found to be the high variance of these methods, which led, during the eighties and nineties, to the development of various variance reduction techniques [1, 2] and in particular so-called ensemble methods [3, 4], which consist of modeling the sought input-output relationship with an ensemble of trees whose predictions are aggregated by some voting scheme.

In this paper we present a new tree-based ensemble method called "Extra-Trees" (standing for extremely randomized trees). This algorithm averages predictions of trees obtained by partitioning the input-space with randomly generated splits [5], which leads to a significant improvement of precision, and has various algorithmic advantages, in particular a reduced computational complexity with respect to classical trees and other ensemble methods.

We explain how and why the method works, by describing the supervised learning algorithm and analytical characterizations of the models it produces. Then we discuss two generic applications of this algorithm of wide practical interest, respectively for time-series classification [6] and for the automatic inference of near-optimal sequential decision policies from experimental data [7].

To fix ideas, we start the paper with a quick review of what automatic learning and data mining are all about, introducing the main learning problems, protocols and terminology and reviewing the main results of research in the field while providing some pointers to the relevant literature. The rest of the paper is composed of three sections describing respectively the Extra-Trees method, and its use for time-series classification and learning of near optimal control strategies, respectively.

We have made efforts to make the paper self-contained. The reader already familiar with automatic learning, be it at an intuitive level, can skip section 2 and part of the introductory material of section 3 about standard tree-based methods.

2 AUTOMATIC LEARNING PER SE

Generally speaking, automatic learning aims at exploiting data gathered from observations (or simulations) of a system (or an environment), in order to build models explaining the behavior of the system and/or decision rules to interact in an appropriate way with it.

In what follows, we first describe the two automatic learning problems of interest in this paper, then we briefly discuss the relationship between automatic learning and classical statistical modeling.

2.1 *Types of automatic learning problems*

The three main types of automatic learning problems are so-called *supervised*, *reinforcement*, and *unsupervised* learning. We will focus on the two former in this paper and we will introduce first their probabilistic/statistical formalization and terminology. We refer the interested reader to more general textbooks for further information about automatic learning theory, its relation to other disciplines, and the precise description of the algorithms to which we refer in this paper [8, 9, 10, 11, 12].

2.1.1 *Supervised learning problem*

Given a sample $\{(x^i, y^i)\}_{i=1}^N$ of input-output pairs (where $x^i \in X$ and $y^i \in Y$)[1], a supervised learning algorithm aims at automatically building an input-output function (or a model, or a predictor) $f(x) : X \rightarrow Y$ to compute approximations of outputs as a function of inputs. Typically, a supervised learning algorithm searches in a (possibly very large, but restricted) set of candidate input-output functions, called the hypothesis space $\mathcal{H}$ (a subset of the space Y^X of all possible input-output functions). For example, decision trees, neural networks, and linear regression use different hypothesis spaces.

[1] We will focus on the case of a Euclidean input space ($X = \mathbb{R}^n$), and consider the cases where the output space $Y = \mathbb{R}$ for regression problems, and $Y = \{y_1, y_2, \ldots, y_m\}$ for (m-class) classification problems; however, supervised learning, in general, considers arbitrary input and output spaces.

Denoting the set of all finite size samples by

$$(X \times Y)^* = \bigcup_{N=1}^{\infty} (X \times Y)^N, \tag{1}$$

a *deterministic*[2] supervised learning algorithm A can thus formally be stated as a mapping

$$A : (X \times Y)^* \rightarrow \mathcal{H} \tag{2}$$

from $(X \times Y)^*$ into the hypothesis space $\mathcal{H}$. For a given sample $ls \in (X \times Y)^*$ we will denote by $A(ls)$ the model returned by the algorithm A.

The probabilistic formalization of supervised learning considers that $x : \Omega \rightarrow X$ and $y : \Omega \rightarrow Y$ are two random variables defined over some (unknown) probability space $(\Omega, \mathcal{E}, P)$. Let $P_{X,Y}$ denote their (unknown) joint probabiltiy distribution defined over $X \times Y$ and $\ell : Y \times Y \rightarrow I\!R^+$ be a non-negative loss function defined over $Y \times Y$, and for any $f \in \mathcal{H}$ let us denote by

$$L(f) = \int_{X \times Y} \ell(f(x), y) dP_{X,Y} \tag{3}$$

the inaccuracy (or average loss) of f. The goal of supervised learning is the derivation from a learning sample of a function $f \in \mathcal{H}$ which minimizes $L(f)$.

Assuming that samples $ls^N = \{(x^i, y^i)\}_{i=1}^N$ are drawn according to some sampling distribution $P_{(X,Y)^N}$, the sampling process and algorithm A induce a probability distribution over the hypothesis space and hence a probability distribution over inaccuracies $L(A(ls^N))$. Let us denote by

$$\overline{L}_A^N = \int_{(X,Y)^N} L(A(ls^N)) dP_{(X,Y)^N} \tag{4}$$

the expected average loss of A for fixed sample size N, by

$$L_{\mathcal{H}}^* = \inf_{f \in \mathcal{H}} L(f) \tag{5}$$

the lowest reachable average loss in $\mathcal{H}$, and by

$$L^* = \inf_{\mathcal{H} \subset Y^X} L_{\mathcal{H}}^* \tag{6}$$

the lowest possible average loss over all possible input-output functions for all possible hypothesis spaces $\mathcal{H}$.

Besides defining general conditions (on X, Y, $P_{X,Y}$, $P_{(X,Y)^N}$, ℓ, $\mathcal{H}$, A etc.) under which the above introduced quantities indeed exist, the objective of statistical learning theory is to study whether and how $\overline{L}_A^N$ and $L(A(ls^N))$ converge to $L_{\mathcal{H}}^*$ [13]. For i.i.d. sampling mechanisms[3], the essential result of this theory is that such convergence can be guaranteed independently of the unknown probability distribution $P_{X,Y}$, provided that $\mathcal{H}$ is not too large (i.e. of finite Vapnik-Chervonenkis-dimension) [10].

Consequently, the theoretical design of supervised learning algorithms also consists essentially of constructing sequences of growing hypothesis spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \ldots$ such that

$$\lim_{n \rightarrow \infty} L_{\mathcal{H}_n}^* = L^*, \tag{7}$$

and of defining associated learning algorithms A_i with good convergence properties, i.e. such that

$$\lim_{N \rightarrow \infty} L(A_i(ls^N)) = L_{\mathcal{H}_i}^*. \tag{8}$$

Examples of such hypothesis spaces are given by single-hidden-layer perceptrons with growing numbers of hidden neurons, or decision trees with growing numbers of nodes.

In practice, much of the research in supervised learning has focused on the design of algorithms scaling well in terms of computational requirements with the sample size and with the dimension of the input space X (and to a lesser extent that of the output space Y), and using "large" hypothesis spaces able to model complex non-linear input-output relations. From this research two broad classes of (closely related) algorithms have emerged during the last fifteen years, based respectively on kernels [14, 15] and on ensembles of trees [16, 5], which both automatically adapt the size of their hypothesis space to the sample size and input space dimensions. These two classes of methods are discussed in further detail in section 3 below.

2.1.2 *Reinforcement learning problem*

To avoid confusion with the input x of supervised learning, we denote by $s_t \in S$ the state of a dynamic system at time t, by $d_t \in D$ the control variable, and $r_t \in I\!R$ an instantaneous reward signal. Then, given a sample of N_T system trajectories

$$\{(s_0^i, d_0^i, r_0^i, s_1^i, \ldots s_{h_i-1}^i, d_{h_i-1}^i, r_{h_i-1}^i, s_{h_i}^i)\}_{i=1}^{N_T}, \tag{9}$$

reinforcement learning aims at deriving an approximation of an optimal decision strategy $\hat{d}^*(s,t)$ maximizing system performance in terms of a cumulated performance index over a certain horizon h, defined by

$$R_h = \sum_{t=0}^{h-1} \gamma^t r_t, \tag{10}$$

where $\gamma \in (0, 1]$ is a discount factor [17, 18].

From a theoretical point of view, reinforcement learning can be formalized within the stochastic dynamic programming framework. In particular, supposing that the system obeys to a time-invariant dynamics

$$s_{t+1} = f(s_t, d_t, w_t), \tag{11}$$

where w_t is a memoryless and time-invariant random process, and obtains a bounded and time-invariant reward signal

$$r_t = r(s_t, d_t, w_t), \tag{12}$$

[2] Below, in Section 3, we will also consider the more general case of so-called *randomized* learning algorithms which, instead of picking a particular hypothesis $A(ls)$, randomly sample hypotheses from an induced conditional distribution over $\mathcal{H}$ given a learning sample, $P_A(h|ls)$.

[3] Notice that while, originally in the late seventies and eighties, statistical learning theory was developed under the classical assumption of i.i.d. (independent and identically distributed) sampling according to the distribution $P_{X,Y}$, i.e. under the assumption that $P_{(X,Y)^N} = P_{X,Y}^N$, more recent work aims at weakening this assumption to cases where the samples are not independently distributed anymore [9].

over an infinite horizon ($h \rightarrow \infty$), one can show that the two following (Bellman) equations define an optimal decision strategy

$$Q(s,d) = E\{r(s,d,w) + \gamma \max_{d'} Q(f(s,d,w),d')\}, \quad (13)$$

$$d^*(s) = \arg\max_d Q(s,d). \quad (14)$$

Reinforcement learning can thus be tackled by developing algorithms to solve these equations (or their time-variant and finite horizon counterparts) approximately when the sole information available about the system dynamics and reward function are provided by a sample of system trajectories. The theoretical questions that have been studied in this context concern the statement of conditions on the sampling process and on the learning algorithm ensuring convergence to an optimal policy in asymptotic conditions (i.e., when $N_T \rightarrow \infty$ and/or $h_i \rightarrow \infty$)

Recent work in the field has allowed to take full advantage from state-of-the art supervised learning algorithms by defining appropriate frameworks to plug these algorithms in the reinforcement learning protocol. In particular, model based reinforcement learning methods use the sample to build approximations of the system dynamics and reward function and dynamic programming methods to derive from them an approximation of the optimal decision strategy. On the other hand, the Q-learning framework uses supervised learning in order to construct from the sample an approximation of the Q-function and derive from it the decision policy. While the first generation of Q-learning methods used parametric approximation techniques together with on-line gradient descent [19], the recently proposed fitted Q iteration method allows to fully exploit any parametric or non-parametric batch mode supervised learning algorithm in this context [7]. This latter method is further discussed below in Section 5.

2.2 Discussion

As it may be clear from this short overview, automatic learning tackles essentially classical modeling problems of statistics. However, while classical statistics has focused on the analytical study of parameter identification, by assuming that the functional forms of distributions are given, automatic learning has focused on the design of data driven algorithms, which are generally not exploiting any strong parametric assumptions and hence can in principle cope with larger classes of problems [20].

In automatic learning many algorithms have been originally designed in a heuristic way and were initially studied only empirically, by applying them to synthetic or real-life datasets and comparing their results with those of other methods. The developments in computer hardware, the availability of large databases and the good empirical performances of these algorithms made them become more and more popular in practice. During the last twenty years, statisticians and theoretical computer scientists became more strongly interested in this field and they drove significant theoretical research allowing to better understand the behavior of these algorithms, and even improve their design thanks to this new insight [9, 10, 13, 16].

Further work is focusing on developing tailored algorithms well suited to handle specific classes of practical problems, like time-series forecasting, image and text classification for instance, where the inputs (and/or the outputs) have specific properties [21, 6].

3 EXTREMELY RANDOMIZED TREES

For the sake of clarity, we will focus in the present section on (supervised) *regression* problems using a square error loss-function and assume that all input variables are numerical (i.e. $y^i \in I\!R$, $\ell(y,y') = (y-y')^2$, and $x^i = (x^i_1, x^i_2, \ldots, x^i_n) \in I\!R^n$). While our mathematical analysis is specific to this case, let us however stress the fact that the discussed methods and general ideas extend in a straighforward way to more general input and/or output spaces and to other loss functions (e.g., see [5, 22]).

3.1 Standard (single) tree-based regression

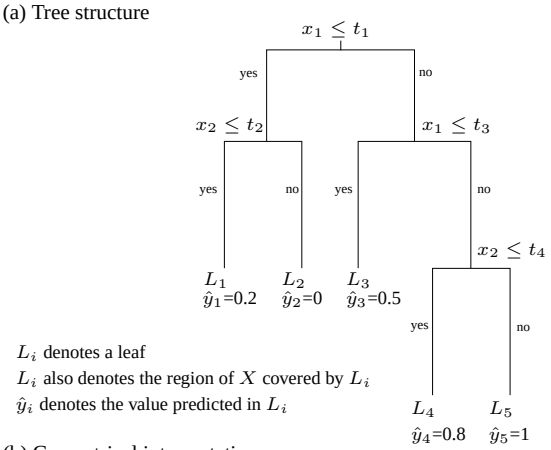
Standard regression trees [1] propose a solution to the supervised learning problem when the output space is the real axis, $Y = I\!R$, and the loss functions ℓ is the square error:

$$\ell(f(x),y) = (f(x)-y)^2. \quad (15)$$

The general idea of regression trees is to recursively split the learning sample with binary tests in the form $x_i \leq t$, where x_i is one of the input variables and t a threshold. Both are chosen by the tree growing procedure so as to reduce as much as possible the variance of y in the two subsamples resulting from that split. The splitting procedure is applied to the whole learning sample ls to define the root node of the tree and then recursively called on its successor nodes. It is stopped at terminal nodes either when the output is constant in this node or some other stopping criterion is met (e.g., the size of the local subsample goes below some threshold, or the split is deemed not significant according to some statistical test).

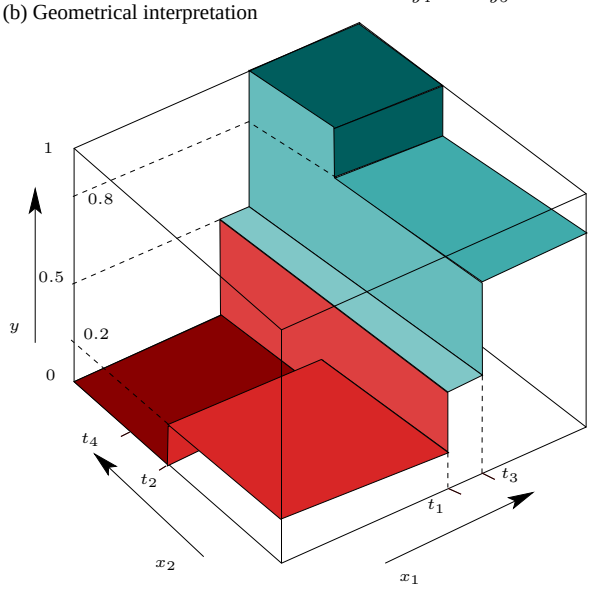
To fix ideas, Figure 1 illustrates this in a simple two-dimensional case, for a tree with four internal (test) nodes and five leaves. Figure 1a also illustrates the notation that we use subsequently, and the fact that the tree was grown in the top-down and left-to-right order is suggested by the numbering of the thresholds at its test nodes, from 1 to 4, and the numbering of its leaves, from 1 to 5. Obviously, such a tree may be used to predict an output for a given input vector $x = (x_1, x_2)$ by following the path from top to bottom along the edges selected by the different tests encountered, and by retrieving the value $\hat{y}$ attached to the reached leaf. The way the tree partitions the input space into a set of non-overlapping hyper-rectangular regions is illustrated on Figure 1b. Figure 1c gives its interpretation as a set of (mutually exclusive) decision rules, and Figure 1d provides its functional expression, in terms of an additive expansion of the indicator functions of the regions attached to its leaves.

(a) Tree structure



L_i denotes a leaf
L_i also denotes the region of X covered by L_i
$\hat{y}_i$ denotes the value predicted in L_i

(b) Geometrical interpretation



(c) Logical interpretation

$L_1 \equiv [x_1 \leq t_1] \wedge [x_2 \leq t_2] : x \in L_1 \Rightarrow f(x) = \hat{y}_1 = 0.2$
$L_2 \equiv [x_1 \leq t_1] \wedge [x_2 > t_2] : x \in L_2 \Rightarrow f(x) = \hat{y}_2 = 0$
$L_3 \equiv [x_1 > t_1] \wedge [x_1 \leq t_3] : x \in L_3 \Rightarrow f(x) = \hat{y}_3 = 0.5$
$L_4 \equiv [x_1 > t_3] \wedge [x_2 \leq t_4] : x \in L_4 \Rightarrow f(x) = \hat{y}_4 = 0.8$
$L_5 \equiv [x_1 > t_3] \wedge [x_2 > t_4] : x \in L_5 \Rightarrow f(x) = \hat{y}_5 = 1$

(d) Functional interpretation

$f(x) = \sum_{i=1}^{5} \hat{y}_i I(x \in L_i)$

Figure 1: Two-dimensional regression tree (inspired from [11])

3.1.1 *Node splitting procedure*

The score measure used to evaluate and select during tree growing a split $s = [x_i \leq t]$ at some node corresponding to a region $R \subset X$ is defined by

$$\text{Score}_R^s = \text{var}\{y|R\} - \frac{|R_{\leq}|}{|R|}\text{var}\{y|R_{\leq}\} - \frac{|R_{>}|}{|R|}\text{var}\{y|R_{>}\}, \quad (16)$$

where $R_{\leq}$ and $R_{>}$ are the subsets of R defined by s, and $|S|$ denotes the number of elements of the learning sample belonging to any $S \subset X$, defined by

$$|S| = \sum_{(x^i,y^i)\in ls} I(x^i \in S), \quad (17)$$

and where $\text{var}\{y|S\}$ denotes the empirical variance of the output variable y in S computed by

$$\text{var}\{y|S\} = |S|^{-1} \sum_{(x^i,y^i):x^i\in S} \left(y^i - |S|^{-1} \sum_{(x^i,y^i):x^i\in S} y^i \right)^2. \quad (18)$$

In the standard procedure, the nodes of the tree are developed in a greedy fashion, by searching at each test node the input variable x_* together with a threshold t_* maximizing the above score. This search is done exhaustively by looking at all possible variables and thresholds, and can be made in the order $n|R| \log |R|$ operations.

3.1.2 *Leaf labelling procedure*

Once the tree is grown, each leaf L_j is labelled with a prediction $\hat{y}_j$ defined as the local sample average of the output variable, given by

$$\hat{y}_j = \frac{1}{|L_j|} \sum_{(x^i,y^i):x^i\in L_j} y_i, \quad (19)$$

where $|L_j|$ is the number of learning cases that reach leaf L_j. Note that, for a given tree structure, these label-values actually minimize the average square prediction error over the learning sample, defined by

$$N^{-1} \sum_{(x^i,y^i)\in ls} (y^i - f(x^i))^2 = N^{-1} \sum_{L_j} \sum_{(x^i,y^i):x^i\in L_j} (y^i - \hat{y}_j)^2. \quad (20)$$

3.1.3 *Kernel interpretation of regression trees*

In general, *kernel based regression* consists of modeling the output using the following type of expansion

$$f_\alpha^k(x) = \sum_{(x^i,y^i)\in ls} \alpha_i k(x^i, x), \quad (21)$$

where $k(x, x')$ is a positive semidefinite kernel defined over the input space, i.e. a symmetric real-valued function defined over $X \times X$ and that is such that for any N and any sample $\{x^i\}_{i=1}^N$ of inputs, the Gram matrix K defined by $K_{i,j} = k(x^i, x^j)$ is positive semi-definite. Once a kernel is defined, the learning algorithm is used in order to determine the vector of coefficients $\alpha = (\alpha_1, \ldots, \alpha_N)$, typically by optimizing a penalized (or regularized) empirical least squares criterion, e.g. (see [11])

$$\alpha^* = \arg\min_\alpha \left\{ \lambda \alpha^T K \alpha + \sum_{(x^i,y^i)\in ls} (f_\alpha^k(x^i) - y^i)^2 \right\}, \quad (22)$$

and using a meta-optimization w.r.t. the choice of kernel k and regularization parameter λ where the generalization error is estimated by cross-validation.

As suggested by equation (21), kernel-based regression consists of using a hypothesis space $\mathcal{H}$ of linear combinations of the functions $\mathcal{H}_X = \{\phi_x(\cdot) = k(x, \cdot)\}_{x\in X}$. This (so-called) *feature-space* is actually a Hilbert space which scalar product, distance and norm, are obtained by the extension of the kernel from $\mathcal{H}_X$ to $\mathcal{H}$ by

$$\langle \sum_{x_i\in X} \lambda_i \phi_{x_i}, \sum_{x'_i\in X} \lambda'_i \phi_{x'_i} \rangle = \sum_i \sum_j \lambda_i \lambda'_j k(x_i, x'_j). \quad (23)$$

Depending on the structure of the input space X and its kernel k, this space may be finite-dimensional (e.g., if X is itself finite) or of infinite dimension[4]. However, for any finite learning sample of size N, only a subset of functions is reached by equation (21), included in the subspace (of dimension $\leq N$) spanned by

$$\{\phi_{x^i}(\cdot) = k(x^i, \cdot)\}_{i=1}^{N}. \tag{24}$$

Let us now analyze in this framework tree-based models. Considering a tree t over some input space X, with leaves $L_1, \ldots, L_K$, let us show that its input-output function can be expressed by the following expansion

$$f_t(x) = \sum_{(x^i,y^i)\in ls} \alpha_i k_t(x^i, x). \tag{25}$$

where the tree kernel k_t is defined by

$$k_t(x, x') = \sum_{i=1}^{K} \frac{I(x \in L_i)}{\sqrt{|L_i|}} \frac{I(x' \in L_i)}{\sqrt{|L_i|}}, \tag{26}$$

and where the parameters α_i are defined by

$$\alpha_i = y^i, \forall i = 1, \ldots, N. \tag{27}$$

Indeed, $k_t(x, x') = 0$ if x and x' reach different leaves of t and $k_t(x, x') = 1/|L_j|$ if they both reach a leaf L_j. Hence, expression (25) thus effectively computes the arithmetic average of the outputs of the learning subsample reaching the same node as x. Notice that the feature-space induced by the kernel (26) is spanned by the indicator functions $L_i(x)$; since these latter are orthogonal, the dimension of this feature-space is equal to the number of leaves of t. This dimension is upper bounded by the learning sample size, and typically grows linearly with it.

We can thus view regression tree induction as a process where the recursive partitioning step corresponds to the automatic construction of a kernel (and its feature space), and where the labelling step consists of computing the weights minimizing the empirical square error, accordining to equation (22) with $\lambda \to \infty$.

3.1.4 *Basic extensions*

The regression tree induction algorithm can be extended in various ways. In particular, in the case of multiple numerical outputs, i.e. $Y = I\!R^n$, with the loss function $\ell(f(x), y) = ||f(x) - y||^2$, where $||\cdot||$ denotes the Euclidean norm in $I\!R^n$, the variance computed in (16) and the predictions at leaf nodes are simply replaced respectively by

$$\text{var}\{y|S\} = |S|^{-1} \sum_{(x^i,y^i):x^i\in S} \left|\left| y_i - |S|^{-1} \sum_{(x^i,y^i):x^i\in S} y^i \right|\right|^2 \tag{28}$$

$$\hat{y}_j = |L_j|^{-1} \sum_{(x^i,y^i):x^i\in L_j} y^i. \tag{29}$$

The latter is the center of mass in the leaf and the former is the average in S of the squared Euclidean distance of output vectors to the center of mass.

The extension to categorical output variables (classification problems) is based on the use of an alternative loss-function (typically the classification error rate) and on different score functions to select splits (typically entropy reduction instead of variance reduction). Furthermore, in [22] we show how this algorithm may be generalized to arbitrary output spaces structured by a kernel, which allows to apply the method to the prediction of structured outputs, such as graphs, text, etc.

Another important extension consists of using a post-processing stage, where the tree is pruned on the basis of an independant sample, so as to remove those parts which *overfit* the learning sample.

3.1.5 *Main strengths and weaknesses*

The main characteristics of (single) tree-based regression are as follows:

Universal approximation/consistency: the method is in principle able to arbitrarily well regress any output variable, provided large enough training samples are provided.

Robustness to outliers: outliers tend to have a very local effect on the models induced, which makes the method intrinsically much more robust than parametric linear or non-linear least squares regression methods.

Computational complexity: tree induction complexity is typically on the order of $N \log N$ in the training sample size and linear in the number of input variables. In other words, the method can cope with very large scale data mining applications with thousands of input variables and millions of training samples.

Robustness w.r.t. irrelevant/redundant inputs: the tree growing procedure identifies relevant variables among a (posssible large) set of candidate ones comprising irrelevant variables, and is intrinsically robust w.r.t. to the presence of redundant variables.

Interpretability: Tree-based models highlight in a rather transparent way the importance of the input variables, and they are able to explain their reasoning.

High variance: compared to other regression methods, tree growing is very sensitive to the training samples, which means that for different samples of identical size drawn from a same distribution the models induced are quite different. This variance translates typically into a low accuracy which is certainly the most important drawback of the method. This variance is particularly detrimental when the information is spread over a large number of input variables. Tree pruning, while aiming at optimizing the bias/variance tradeoff is typically not able to reduce this variance significantly.

[4] In the latter case $\mathcal{H}$ consists of the functions $f(x) = \sum_{i=1}^{\infty} \lambda_i \phi_{x_i}$ of *bounded norm*, i.e. suchthat $\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \lambda_i \lambda_j k(x_i, x_j)$ is finite.

3.2 Ensemble methods

Ensemble-based supervised learning methods consist of using a base learning algorithm A and some perturbation mechanism, in order to derive from a learning sample an ensemble $F = \{f_1, \ldots, f_M\}$ of input-output functions (instead of a single one), and making a prediction at some point x by combining the elementary predictions $\{f_1(x), \ldots, f_M(x)\}$ in some appropriate fashion. In particular, in the case of regression, the final prediction is generally computed as a (generally weighted) average

$$f^M(x) = \sum_{k=1}^{M} w_k f_k(x) \tag{30}$$

where $\sum_k w_k = 1$. The functions f_i are intended to be complementary, in the sense that their prediction errors are not strongly correlated. The weights w_k are meant to be higher for those functions f_i which are believed to be more accurate. In particular, if all ensemble terms f_i are believed to be equivalent in terms of accuracy, then $w_k = 1/M, \forall k = 1, \ldots, M$.

Ensemble methods have been largely studied in the 90's, and are further investigated at present. The two following interpretations/motivations of this idea are of particular interest.

3.2.1 Variance reduction and bagging

Suppose that instead of a single learning sample of size N, we have at our disposal M independent samples $\{ls^k\}_{k=1}^M$ of size N, and let us denote by $f_k = A(ls^k)$ the hypothesis computed by a learning algorithm A using ls^k. One can show (e.g., see [5]) that the learning variance of the algorithm A^M which returns the model

$$f^M(x) = M^{-1} \sum_{k=1}^{M} f_k(x), \tag{31}$$

is reduced by a factor of M w.r.t. to the learning variance of A applied to ls^k, while its bias is left unchanged. In other words, if the variance of A is much higher than its bias, the new algorithm A^M will be much more accurate on the average than A applied to a sample of size N. Since the computational load to generate the model f^M grows linearly with M, depending on the computational complexity of A and the speed of its variance decrease with sample size, A^M may be a viable alternative the application of A to the union $\cup_k ls^k$.

The above analysis inspired the following *bagging* (bagging=bootstrap+aggregating) trick in order to improve a learning algorithm: given a single sample ls of size N, derive from it M *bootstrap* copies (each one is obtained by random sampling N times, with replacement), derive from these samples M models with algorithm A and build a final model by averaging. Because bootstrap sampling from a finite ls (imperfectly) mimics sampling from the population, bagging (imperfectly) allows to reduce variance of A, while (imperfectly) keeping bias unchanged. In particular, in the context of regression (and even more for decision) tree induction, bagging was empirically shown to outperform by far (and almost systematically) standard tree induction in terms of accuracy [3].

3.2.2 Bayesian motivation

Another way to motivate the model averaging of ensemble methods is based on the bayesian interpretation of statistics. In the context of supervised learning, this approach consists of defining a prior distribution $P(f)$ over the hypothesis space $\mathcal{H}$, and by then using the learning sample to compute an approximation of the posterior distribution $P(f|ls)$ and finally, by making predictions according to the following integral

$$f_{\mathcal{H}|ls}(x) = \int_{f \in \mathcal{H}} f(x) dP(f|ls). \tag{32}$$

This equation may be (finitely) approximated by

$$\hat{f}^M_{\mathcal{H}|ls}(x) = \sum_{f_i} M^{-1} f_i(x), \tag{33}$$

assuming that the functions $f_i(x)$ are obtained by sampling from the posterior distribution $P(f|ls)$. Notice that the convergence of this approximation may be improved by sampling preferentially models which have a high posterior probability, i.e. which fit rather well to the learning sample.

3.3 Ensembles of extremely randomized trees

Extra-Trees (see [5] for an in depth discussion and systematic empirical validation) are motivated by the above two ideas. They consist of averaging predictions of an ensemble of trees built in a randomized fashion. Each tree is grown by selecting at each node a number K of random splits (random choice of variable x_i, and random choice of threshold t) and keeping among these the one which maximizes the score. These trees are grown until all subsamples at all leaves are either pure in terms of outputs or contain less than $n_{\min}$ learning samples.

If the parameters are set respectively to $K = 1$ and $n_{\min} = 2$, these trees are totally random (in the sense that their structure does not depend on the outputs in the learning sample) and they perfectly fit the learning sample. Thus, in this case the Extra-Trees method can be viewed as an approach to sample in a neutral way from the set of all possible trees perfectly fitting the learning sample.

This tree sampling mechanism may be adjusted (biased) by using larger values of K, leading to trees preferentially using inputs with higher score values, and/or by using higher values of $n_{\min}$, leading to smaller trees, be it at the expense of a less good fit on the training sample. In practice, the optimal values of these two parameters are problem dependent and they may be adjusted automatically using a meta-optimization using cross-validation. Typically, large values of K are more appropriate if the problem has a large number of irrelevant variables, while small values correspond to assuming that the information is spread evenly over the input variables. On the other hand, larger values of $n_{\min}$ will be better suited if there is a lot of noise on the output variable.

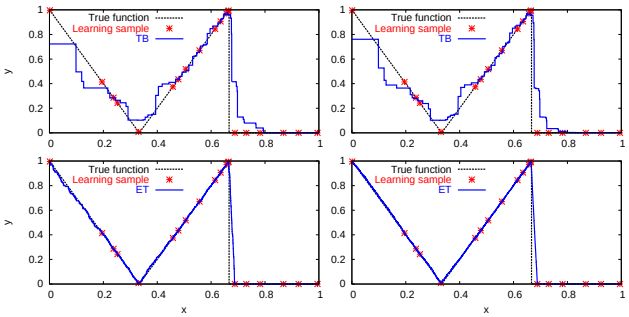


Figure 2: Tree Bagging (top), and Extra-Trees (bottom) on a one-dimensional piecewise linear problem ($N = 20$, $n_{\min} = 2$). Left with $M = 100$ trees, right with $M = 1000$ trees (adapted from [5])

3.3.1 *Geometric characterization*

While regular trees, as well as bagged ensembles of regular trees (even if $M \to \infty$) yield piecewise constant models, it is shown in [5] that Extra-Trees produce (in asymptotic conditions) a smoother and piecewise multilinear model. This is illustrated in the case of a one-dimensional input space, at Figure 2, showing that the Extra-Trees model rather quickly converges to a piece-wise linear model. Since most practical problems correspond to smooth input-output regressors, this explains why Extra-Trees are often more accurate than tree-bagging and other tree-based ensemble methods.

3.3.2 *Extra-Trees kernels*

As single tree models, ensembles of regression trees may also be interpreted in the kernel framework. Indeed, since their prediction is an average of predictions of tree-based models, straightforwardly their kernel is obtained by averaging of the individual tree kernels. In particular, for an ensemble $T = \{t_1, \ldots, t_M\}$ of M trees, the kernel-based model formulation becomes

$$f_T(x) = \sum_{(x^i, y^i) \in ls} y^i k_T(x^i, x), \tag{34}$$

where the ensemble kernel k_T is defined by

$$k_T(x, x') = \sum_{k=1}^{M} M^{-1} \sum_{i=1}^{|t_k|} \frac{I(x \in L_i^k) I(x' \in L_i^k)}{|L_i^k|}, \tag{35}$$

where L_i^k denotes the ith leaf of the kth tree and $|t_k|$ its number of leaves. This kernel essentially counts the proportion of trees in T in which the inputs x and x' reach the same terminal node. Notice that while single tree kernels are non-smooth (and take at most as many different values as there leaves in a tree), in the limit of $M \to \infty$, the Extra-Trees ensemble kernel becomes continuous.

The kernel-based reformulation shows that Extra-Trees are similar to nearest-neighbour types of methods, where the metric is automatically induced from the learning sample. Further work aims at characterizing other properties of these random tree-ensemble kernels (e.g., see [5] for some indications).

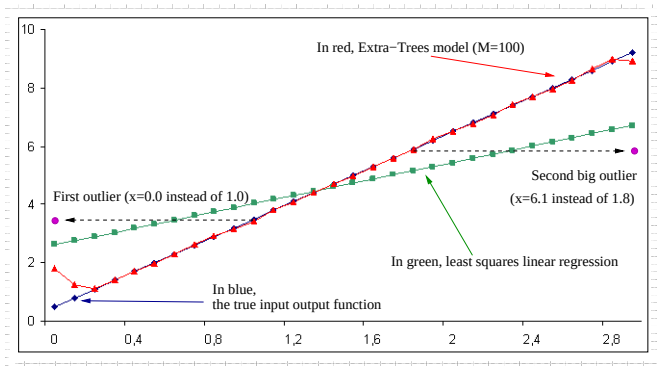


Figure 3: Robustness to outliers of Extra-Trees vs Linear regression

3.3.3 *Main properties of Extra-Trees*

The main characteristics of Extra-Trees based regression are as follows:

Universal approximation/consistency: it carries over directly from single-tree based methods.

Robustness to outliers: Figure 3 illustrates the robust behaviour of Extra-Trees w.r.t. gross errors. The underlying problem is a (one-dimensional) linear input-output relation drawn in blue on Figure 3. To train the models, a set of 30 data points was generated, evenly spread in the interval $x \in [0; 2.9]$, the outputs being computed by $y = 3x + 0.5$. As suggested by the pink points on the graphic, two outliers were introduced before training by corrupting two datapoints: the point $(1, 3.5)$ was replaced by $(0, 3.5)$ and $(1.8, 5.9)$ by $(6.8, 5.9)$. On this corrupted dataset, we trained both a linear regression (green curve) and an ensemble of 100 Extra-Trees (red curve, $n_{\min} = 4$ and $K = 1$). The graphic clearly highlights that, while the linear regression is strongly and globally affected by the outliers, the Extra-Trees model is only marginally affected, and only very locally in the regions where the outliers are located.

Computational complexity: the complexity is essentially proportional to $MKN \log N$, which may be better than that of single trees if the number n of input variables is very large compared to K. Note that accuracy always increases monotonically with M, but in a problem dependent way; typical values of M are in the range $[10; 100]$.

Robustness w.r.t. irrelevant/redundant inputs: the tree growing remains robust to irrelevant variables as long as K is sufficiently large w.r.t. n.

Interpretability: it is strongly reduced because the randomized trees are individually meaningless; nevertheless it is possible to compute the importance of input variables from a set of Extra-Trees.

Very low variance: compared to standard tree-based regression the variance of Extra-Trees is negligible.

Extensions/generalizations: those extensions that were discussed for standard trees all carry over to Extra-Trees, with minor adaptations.

4 TIME-SERIES CLASSIFICATION

In this section we consider a supervised learning problem, where inputs are multidimensional real-valued times-series (in discrete time) and the output is a discrete class label. Let us denote by

$$x[1\cdots t] = \begin{pmatrix} x_1[1] & \dots & x_1[t] \\ \vdots & & \vdots \\ x_n[1] & \dots & x_n[t] \end{pmatrix}, \qquad (36)$$

an observation of an n-dimensional time-series, where t denotes its duration and where the $x_i[1\cdots t] \in I\!R^{[1\cdots t]}$ are the elementary (temporal) input variables. We will denote by $t_{\min} \geq 1$ the minimal duration of any time-series considered in a particular learning problem. Let us consider a learning sample of classified time-series, i.e.

$$ls = \{(x^i[1\cdots t^i], y^i)\}_{i=1}^N, \qquad (37)$$

such that $y^i \in Y = \{y_1, \dots, y_m\}$ and $t^i \geq t_{\min}$. Notice that the time in a given time-series of the ls is interpreted here as relative time to its initial value, and that the durations of individual time-series may vary within the learning sample. Note also that $t_{\min}$, if not known a priori, can be upper bounded by $\min_{i=1}^N t^i$.

Supervised time-series classification consists of using an algorithm to derive from such a learning sample a time-series classifier, or in other words a function mapping any time-series $x[1\cdots t]$ of any duration $t \geq t_{\min}$ to a value $f(x[1\cdots t]) \in Y$. One-dimensional time-series *forecasting* would correspond to a similar problem, where $n = 1$ and $y = x[t+1]$.

4.1 Segment and Combine framework

The segment and combine algorithm [6] uses a standard (propositional) supervised learning algorithm to yield a time-series classifier from learning sample of time-series as in equation (37) . It works in the following way ($l \leq t_{min}$):

Subseries sampling. For $j = 1, \dots, N_s$ choose $i_j \in \{1, \dots, N\}$ randomly, then choose a subseries offset $t_0^j \in \{0, \dots, t^{i_j} - l\}$ randomly, and create an attribute vector of length $n \times l$

$$x^{j,l} = \Big(x_1^{i_j}[t_0^j+1], \dots, x_1^{i_j}[t_0^j+l], \dots \\ \dots, x_n^{i_j}[t_0^j+1], \dots, x_n^{i_j}[t_0^j+l]\Big)$$

concatenating the values of all n temporal attributes over the time interval $t_0^j + 1, \dots, t_0^j + l$. Collect the samples in a training set of subseries

$$ls_{N_s}^l = \left\{ (x^{j,l}, y^{i_j}) \middle| \, j = 1, \dots, N_s \right\}.$$

Classifier training. Use a (propositional) supervised base learner to build a subseries classifier $f^l(x[1\cdots l])$ from the subseries sample $ls_{N_s}^l$.

Notice that the "classifier" $f^l(x[1\cdots l])$ is supposed to return a class-probability vector (each component of which estimates the probability of one of the m classes $y_i \in Y$).

Time-series classification. For any new time-series $x[1\cdots t]$ extract systematically all its subseries of length l, $x[i+1\cdots i+l], \forall i \in \{0, \dots, t-l\}$. Use the learned model to estimate the probability that each subseries belongs to a signal of each class and classify the time-series by majority vote over the average probability estimates of its subseries, i.e. according to

$$f(x[1\cdots t]) \triangleq \arg\max_{y \in Y} \left\{ \sum_{i=0}^{t-l} f^l(x[i+1\cdots i+l]) \right\}.$$

Note that if the base learner returns $0/1$ class indicators, the aggregation step merely selects the class receiving the largest number of votes.

4.2 Tree-based times series classification

The above framework is easily combined with tree-based supervised learning by using in place of the base learner a tree-based method. In particular, reference [6] studies the use of Extra-Trees in this context and shows empirically, on a representative sample of non-trivial time-series classification tasks, that this combination provides quite interesting results. Within this context, it is useful to notice that the averaging effect of the segment and combine approach has itself some capability of variance reduction, which allows one to use rather small values of M (say $M \in [10; 50]$) in the Extra-Trees method, which is at the benefit of a reduced computational complexity.

4.3 Discussion

The subseries length l is a parameter of the segment and combine approach. It may in practice be adjusted to problem specifics by using a meta-optimization based on cross-validation. Note that the smaller the values of l, the more local and the more time-shift invariant the classifiers derived by the method are.

Along similar lines, the segment and combine framework combined with Extra-Tress is studied in [21] for image classification. Reference [23], on the other hand, provides a more general discussion of this framework for structured data classification, such as text, images and time-series.

5 APPLICATION TO OPTIMAL CONTROL

Let us first notice that the Bellman equation (13) introduced in Section 2 actually defines a contraction operator on the Banach space of bounded functions from $S \times D$ to $I\!R$. Therefore, this equation has a unique solution and, starting with an arbitrary initial Q-function guess Q_0, and iterating sufficiently many times in the following way

$$Q_{t+1}(s,d) = E\{r(s,d,w) + \gamma \max_{d'} Q_t(f(s,d,w),d')\} \qquad (38)$$

will necessarily yield a good approximation of this solution, from which a good approximation of an optimal infinite horizon (and stationary) policy can then be deduced by equation (14), easily (at least) if D is finite.

It is also useful to remark that if the iteration is started with $Q_0 \equiv 0$, the above iteration yields a sequence of Q_t-functions corresponding to finite horizon tail problems. In other words, the same iterative procedure, if carried out a fixed number h of times, will allow to determine an optimal (and time-varying) control policy for a finite horizon of length h. For the sake of simplicity, we nevertheless consider here the time-invariant infinite horizon case of the optimal control problem, and we refer the interested reader to [24] for the full presentation in the time-varying and finite horizon case.

The ADP (approximate dynamic programming) approach is based on the idea that, instead of computing these iterations exactly, they can be carried out in an approximate fashion by using an approximation architecture (i.e. a hypothesis space) to represent Q-functions, and by iterating over projections on this hypothesis space. Furthermore, in order to handle the computationally difficult problem of computing the expectation $E\{\cdot\}$ in equation (38), one may use Monte-Carlo simulation rather than numerical integration.

Putting these ideas together, and imposing the constraint that the only available information about the system to be controlled is contained in a set of trajectories, as defined in section 2, leads to the idea of Q-learning which aims at iterating equation (38) over a hypothesis space of candidate Q-functions, while computing expectations based only on the available sample of trajectories. The *fitted Q iteration* algorithm, introduced below, does this by exploiting in a generic way a batch-mode supervised learning algorithm in the inner loop of this iteration. We will see that, combined with tree-based supervised learning, this leads to a very robust and scalable approach to reinforcement learning as well as for simulation-based approximate dynamic programming in large state spaces.

5.1 *Fitted Q iteration framework*

Remind that the sole information used by reinforcement algorithms is given by a set of N_T observed system trajectories, as defined in equation (9). Before applying the fitted Q iteration algorithm this sample is flattened into a sample fts of $N = \sum_{i=1}^{N_T} h_i$ four-tuples, defined by

$$fts = \{(s^i_{t^i}, d^i_{t^i}, r^i_{t^i}, s^i_{t^i+1})\}_{i=1}^{N}. \tag{39}$$

Notice that in this operation the absolute time references are lost, but since we consider here a time-invariant system, actually no relevant information is lost in this way.

To exploit this sample of four-tuples, the fitted Q iteration uses batch-mode supervised learning to yield a sequence of approximate Q_t-functions from a sample of trajectories in the following way:

- Initialization: Set $t = 0$ and $\hat{Q}_0(s, d) \equiv 0$.
- Basic iteration:
  - Set $t = t + 1$
  - Create a learning sample $ls_t = \{(x^i, y^i)\}_{i=1}^{N}$ of input-output pairs, where $x^i = (s^i_{t^i}, d^i_{t^i})$ and $y^i = r^i_{t^i} + \gamma \max_d \hat{Q}_{t-1}(s^i_{t^i+1}, d)$.
  - Apply a supervised learning algorithm to build $\hat{Q}_t(x, u)$ from the learning sample ls_t.
- Finalization: if $\max_i |Q_t(s^i_{t^i}, d^i_{t^i}) - Q_{t-1}(s^i_{t^i}, d^i_{t^i})| \leq \epsilon$ or $t = t_{\max}$, extract the approximate optimal decision strategy $\hat{d}^*(\cdot)$ from the last function $\hat{Q}_t$ by

$$\hat{d}^*(s) = \arg\max_d \hat{Q}_t(s, d).$$

Note that the number of iterations necessary to yield convergence varies strongly with problem specifics and is often increasing with γ.

5.2 *Tree-based batch mode reinforcement learning*

If the state and decision spaces are finite and of relatively small size, the supervised learning step in the above algorithm can be carried out by using a tabular representation of the Q-functions and by simply averaging sample output values corresponding to inputs falling in each cell of the table. In the case of infinite (say continuous) or simply very large input and/or decision spaces, this idea can still be applied by discretizing these spaces a priori.

However, in order to yield robust estimates of the Q-functions, the number of cells has to remain smaller than the number N of samples, specially in the case of stochastic problems, which precludes the application of this idea to the case of high-dimensional state spaces and/or small samples. Indeed, the requirement that the number of observations per cell must be kept above a certain theshold, leads to a very rapid (exponentially fast) increase of the coarseness of the discretization with the dimension of the input/decision spaces, for any finite sample size. This phenomenon has been called the curse of dimensionality and has, in fact, hindered for many years (actually, since the early sixties) the application of dynamic programming to large scale real-life problems.

The tree-based batch mode reinforcement learning approach consist of using instead a tree-based regression algorithm in order to fit the Q-function. With respect to the use of a tabular approximation architecture this has the main advantage to adapt the discretization of $S \times U$ automatically to the problem at hand, and with a coarseness which is automatically linked to the size of the available sample. In this respect, ensemble-based methods offer the advantage of better accuracy than single trees, and within this category the Extra-Trees offer a further advantage of computational efficiency (remind that the supervised learning algorithm needs to be called several (tens, or hundreds of) times during the fitted Q-learning process). Furthermore, contrary to many other approximation architectures, e.g. linear and generalized linear regression, the tree-based methods have the additional feature of bounded input-output approximation (actually, their predictions are necessarily a convex combination of the sample values), which in the context of the fitted Q iteration algorithm yields the guarantee of non-divergence (and in some more specific conditions, of convergence) of the iterative fitting procedure.

5.3 Discussion

The above features, combined with the computational efficiency (and scalability) and the consistency property of tree-based supervised learning, yield a new very powerful framework of reinforcement learning, able to address complex and large scale applications, and to most efficiently exploit available samples of trajectories. We refer the interested reader to [7, 24] for further details and references to related work.

Notice that even when the system dynamics and reward functions are known (or can be simulated), the reinforcement learning framework may still be used as an alternative to direct optimization (e.g., dynamic programming or model predictive control), by extracting decision policies from samples generated automatically by Monte-Carlo simulation. In this context, the advantages of tree-based batch mode reinforcement learning are its capability to exploit efficiently large samples and cope with high-dimensional non-linear and stochastic problems.

6 CONCLUSION

In this paper, we have presented a new supervised learning method called Extra-Trees, based on averaging predictions by randomly generated trees. This method has been discussed in terms of its intrinsic properties, namely scalability, robustness, accuracy and flexibility, and we have analyzed it in the perspective of the so-called kernel-based methods. We have also discussed two particular frameworks wrapping this method, namely "segment and combine" for the classification of time-series, and "Tree-based batch mode reinforcement learning" for the inference of optimal control policies from sequential system performance recordings.

Due to limited space and time, we did not discuss actual real-world applications of this method, be it in the context of power systems or more generally. Nevertheless, we hope that this paper will foster many new applications in addition to the already existing ones.

ACKNOWLEDGMENTS

Damien Ernst and Pierre Geurts acknowledge the support of the Belgian FNRS (Fonds National de la Recherche Scientifique) where they are scientific research workers.

REFERENCES

[1] L. Breiman, J. Friedman, R. Olsen, and C. Stone, *Classification and Regression Trees*. Wadsworth International, 1984.

[2] L. Wehenkel, "Discretization of continuous attributes for supervised learning: variance evaluation and variance reduction," in *Proceedings of the International Fuzzy Systems Association World Congress*, 1997, pp. 381–388.

[3] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[4] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms : bagging, boosting, and variants," *Machine Learning*, vol. 36, pp. 105–139, 1999.

[5] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning Journal (advance access: DOI 10.1007/s10994-006-6226-1)*, pp. 1 – 40, 2006.

[6] P. Geurts and L. Wehenkel, "Segment and combine approach for non-parametric time-series classification," in *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, October 2005.

[7] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, April 2005.

[8] S. Russel and P. Norvig, *Artificial Intelligence: a Modern Approach*. Prentice Hall, 1994.

[9] M. Vidyasagar, *A Theory of Learning and Generalization: with Applications to Neural Networks and Control Systems*. Springer, 1997.

[10] V. Vapnik, *Statistical Learning Theory*. Wiley, New York, 1998.

[11] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[12] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. John Wiley & Sons, Inc., 2001.

[13] T. Poggio, R. Rifkin, S. Mukherjee, and P. Niyogi, "General conditions for predictivity in learning theory," *Nature*, vol. 428, pp. 419–422, 2004.

[14] B. Scholkopf, C. Burges, and A. Smola, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[15] C. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. MIT Press, Cambridge, MA, 2000.

[16] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[17] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[18] R. Sutton and A. Barto, *Reinforcement Learning. An Introduction*. MIT Press, 1998.

[19] C. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, Cambridge University, England, 1989.

[20] L. Breiman, "Statistical modeling: the two cultures," *Statistical Science*, vol. 16, no. 3, pp. 199–231, 2001.

[21] R. Marée, P. Geurts, J. Piater, and L. Wehenkel, "Random subwindows for robust image classification," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, CVPR 2005*, vol. 1, 2005, pp. 34–40.

[22] P. Geurts, L. Wehenkel, and F. d'Alché Buc, "Kernelizing the output of tree-based methods," in *Proc. of International Conference on Machine Learning, to appear*, 2006.

[23] P. Geurts, R. Marée, and L. Wehenkel, "Segment and combine: a generic approach for supervised learning of invariant classifiers from topologically structured data," in *Machine Learning Conference of Belgium and The Netherlands (Benelearn)*, 2006.

[24] L. Wehenkel, M. Glavic, P. Geurts, and D. Ernst, "Automatic learning of sequential decision strategies for dynamic security assessment and control," in *Proc. of IEEE PES General Meeting*, 2006, p. 6.