Ensembles on Random Patches

Gilles Louppe and Pierre Geurts

Dept. of EE & CS, & GIGA-R
University of Liège, Belgium

Abstract. In this paper, we consider supervised learning under the assumption that the available memory is small compared to the dataset size. This general framework is relevant in the context of big data, distributed databases and embedded systems. We investigate a very simple, yet effective, ensemble framework that builds each individual model of the ensemble from a random patch of data obtained by drawing random subsets of *both* instances and features from the whole dataset. We carry out an extensive and systematic evaluation of this method on 29 datasets, using decision tree-based estimators. With respect to popular ensemble methods, these experiments show that the proposed method provides on par performance in terms of accuracy while simultaneously lowering the memory needs, and attains significantly better performance when memory is severely constrained.

1 Motivation

Within the past few years, big data has become a popular trend among many scientific fields. In life sciences, computer vision, Internet search or finance, to cite a few, quantities of data have grown so large that it is increasingly difficult to process, analyze or visualize. In many cases, single computers are no longer fit for big data and distributed environments need to be considered to handle it. Although research is very active in this area, machine learning is no exception to this new paradigm. Much still needs to be done and methods and algorithms have to be reinvented to take this constraint into account.

In this context, we consider supervised learning problems for which the dataset is so large that it cannot be loaded into memory. In [1], Breiman proposed the Pasting method to tackle this problem by learning an ensemble of estimators individually built on random subsets of the training examples, hence alleviating the memory requirements since the base estimators would be built on only small parts of the whole dataset. Earlier, Ho proposed in [2] to learn an ensemble of estimators individually built on random subspaces (i.e., on random subsets of the features). While the first motivation of the Random Subspace method was to increase the diversity within the estimators of the ensemble, it can actually also be seen as way to reduce the memory requirements of building individual models. In this work, we propose to combine and leverage both approaches at the same time: learn an ensemble of estimators on *random patches*, i.e., on random subsets of the samples *and* of the features. Through an extensive empirical

study, we show that this approach (1) improves or preserves comparable accuracy with respect to other ensemble approaches which build base estimators on the whole dataset while (2) drastically lowering the memory requirements and hence allowing an equivalent reduction of the global computing time.

The rest of this paper is organized as follows. Section 2 describes and then compares the Random Patches method with popular ensemble algorithms. In Section 3, we investigate experimentally the performance of the method on an extensive list of datasets and then draw some first conclusions. We then study in Section 4 the benefits of our algorithm under memory constraints and show that, in that context, it appears to be significantly better than other ensemble methods. We conclude and discuss future work directions in Section 5.

2 Random Patches

In this section, we formally describe our method, briefly introduce standard base estimators that have been considered in this work, and then discuss how our algorithm relates with popular ensemble methods.

2.1 Description

The Random Patches algorithm proposed in this work (further referred to as RP) is a wrapper ensemble method that can be described in the following terms. Let $R(p_s, p_f, D)$ be the set of all random patches of size $p_s N_s \times p_f N_f$ than can be drawn from the dataset D, where N_s (resp. N_f) is the number of samples in D (resp. the number of features in D) and where $p_s \in [0, 1]$ (resp. p_f) is an hyper-parameter that controls the number of samples in a patch (resp. the number of features). That is, $R(p_s, p_f, D)$ is the set of all possible subsets containing $p_s N_s$ samples (among N_s) with $p_f N_f$ features (among N_f). The method then works as follows:

1. Draw a patch $r \sim U(R(p_s, p_f, D))$ uniformly at random.
2. Build an estimator on the selected patch r.
3. Repeat 1-2 for a preassigned number T of estimators.
4. Aggregate the predictions by voting (in case of classifiers) or averaging (in case of regressors) the predictions of the T estimators.

2.2 Tree-based methods

While the RP algorithm can exploit any kind of base estimators, we consider in this work only tree-based estimators. We first describe standard classification and regression trees and ensemble methods and then the two specific base learners we have considered in our experiments.

Classification and regression trees. A standard classification/regression tree [3] is an input-output model represented by a tree. Internal nodes of the tree are labeled with a (usually binary) test based on one input feature. Leaves are

labeled with a value of the output (discrete or continuous). The predicted output for a new instance is determined as the output associated to the leaf reached by the instance when it is propagated through the tree. A decision tree is built using a recursive procedure which identifies at each node the test that leads to a split of the node sample into two subsamples that are as pure as possible in terms of their output values, as measured by a so-called score measure. The construction of the tree then stops when some stopping criterion is met.

Ensemble of randomized trees. Single decision trees typically suffer from high variance, which makes them not competitive in terms of accuracy. A very efficient and simple way to address this flaw is to use them in the context of randomization-based ensemble methods. Specifically, the core principle is to introduce random perturbations into the learning procedure in order to produce several different decision trees from a single learning set. For example, in Bagging [4], trees are built on randomly drawn bootstrap copies of the original data, hence producing different decision trees. In Random Forests [5] (RF), Bagging is extended and combined with a randomization of the input features that are used when considering candidates to split internal nodes. In particular, instead of looking for the best split among all features, the algorithm selects, at each node, a random subset of K features and then determines the best test over these features only. In Extremely Randomized Trees [6] (ET), randomization goes even one step further: discretization threholds are also drawn at random and the best test is chosen among the K randomly drawn cut-points. Unlike in RF though, the trees in ET are not built on bootstrap copies of the input data.

Base estimators. We consider and evaluate two base estimators within the RP algorithm: standard classification trees and (single) extremely randomized trees. Unless otherwise stated, trees are unpruned and grown using Gini entropy as the main scoring criterion for node splitting. The parameter K of extremely randomized trees within RP is set to its maximum value $K = p_f N_f$ (i.e., corresponding to no further random selection of features).

2.3 Related work

The first benefit of RP is that it generalizes both the Pasting Rvotes (P) method [1] (and its extensions [7, 8]) and the Random Subspace (RS) algorithm [2]. Both are indeed merely particular cases of RP: setting $p_s = 1.0$ yields RS while setting $p_f = 1.0$ yields P. As such, it is expected that when both hyper-parameters p_s and p_f are tuned, RP should be at least as good as the best of the two methods, provided there is no overfitting associated with this tuning.

When the base estimators are standard decision trees (resp. extremely randomized trees with $K = p_f N_f$), interesting parallels can also be drawn between RP and the RF algorithm (resp. ET). For $p_s = 1.0$, the value of $p_f N_f$ is indeed nearly equivalent to the number K of features randomly considered when splitting a node. A major difference remains though. In RP, the subset of features

is selected globally once and for all, prior to the construction of the tree. By contrast, in RF (resp. in ET) subsets of features are drawn locally at each node. Clearly, the former approach already appears to be more attractive when dealing with large databases. Non-selected features indeed do not need to be considered at all, hence lowering the memory requirements for building a single tree. Another interesting parallel can be made when bootstrap samples are used like in RF: it nearly amounts to set $p_s = 0.632$, i.e. the average proportion of unique samples in a bootstrap sample. Differences are that in a bootstrap sample, the number of unique training samples varies from one to another (while it would be fixed to $0.632N_s$ in RP), and that samples are not all equally weighted.

In addition, RP also closely relates to the SubBag algorithm [9] which combines Bagging and RS for constructing ensembles. Using N_s bootstrapped samples (i.e., nearly equivalent to $p_s = 0.632$) and setting $p_f = 0.75$, Panov et al showed that SubBag has comparable performance to that of RF. An added advantage of SubBag, and hence of RP, is that it is applicable to any base estimator without the need to randomize the latter.

3 On Accuracy

Our validation of the RP algorithm is carried out in two steps. In this section, we first investigate how RP compares with other popular tree-based ensemble methods in terms of accuracy. In the next section, we then focus on its memory requirements for achieving optimal accuracy and its capability to handle strong memory constraints, again in comparison with other ensemble methods.

Considering accuracy only, our main objective is to investigate whether the additional degrees of freedom brought by p_s and p_f significantly improve, or degrade, the performance of RP. Additionally, our goal is also to see whether sampling features once globally, instead of locally at each node, impairs performance, as this is the main difference between RP and state-of-the-art methods such as RF or ET.

3.1 Protocol

We compare our method with P and RS, as well as with RF and ET. For RP, P and RS, two variants have been considered, one using standard decision trees (suffixed below with '-DT') as base estimators, and the other using extremely randomized trees (suffixed below with '-ET') as base estimators. Overall, 8 methods are compared: RP-DT, RP-ET, P-DT, P-ET, RS-DT, RS-ET, RF and ET.

We evaluate the accuracy of the methods on an extensive list of both artificial and real classification problems. For each dataset, three random partitions were drawn: the first and larger (50% of the original dataset) to be used as the training set, the second (25%) as validation set and the third (25%) as test set. For all methods, the hyper-parameters p_s and p_f were tuned on the validation set with a grid-search procedure, using the grid $\{0.01, 0.1, ..., 0.9, 1.0\}$ for both p_s and p_f. All other hyper-parameters were set to default values. In RF and

ET, the number K of features randomly selected at each node was tuned using the grid $p_f N_f$. For all ensembles, 250 fully developed trees were generated and the generalization accuracy was estimated on the test set. Unless otherwise mentioned, for all methods and for all datasets, that procedure was repeated 50 times, using the same 50 random partitions between all methods, and all scores reported below are averages over those 50 runs. All algorithms and experiments have been implemented in Python, using Scikit-Learn [10] as base framework.

3.2 Small datasets

Before diving into heavily computational experiments, we first wanted to validate our approach on small to medium datasets. To that end, experiments were carried out on a sample[1] of 16 well-known and publicly available datasets (see Table 1) from the UCI machine learning repository [11]. Overall, these datasets cover a wide range of conditions, with the sample sizes ranging from 208 to 20000 and the number of features varying from 6 to 168. Detailed average performances of the 8 methods for all 16 datasets using the protocol described above are reported in Table 1 of the supplementary materials[2]. Below, we analyze general trends by performing various statistical tests.

Following recommendations in [12], we first performed a Friedman test that rejected the hypothesis that all algorithms are equivalent at a significance level $\alpha = 0.05$. We then proceeded with a post-hoc Nemenyi test for a pairwise comparison of the average ranks of all 8 methods. According to this test, the performance of two classifiers is significantly different (at $\alpha = 0.05$) if their average ranks differ by at least the critical difference $CD = 2.6249$ (See [12] for further details). The diagram of Figure 1 summarizes these comparisons. The top line in the diagram is the axis along which the average rank R_m of each method m is plotted, from the highest ranks (worst methods) on the left to the lowest ranks (best methods) on the right. Groups of methods that are not statistically different from each other are connected. The critical difference CD is shown above the graph. To further support these rank comparisons, we also compare the 50 accuracy values obtained over each dataset split for each pair of methods by using a paired t-test (with $\alpha = 0.01$). The results of these comparisons are summarized in Table 2 in terms of "Win-Draw-Loss" statuses of all pairs of methods; the three values at the intersection of row i and column j of this table respectively indicate on how many datasets method i is significantly better/not significantly different/significantly worse than method j.

Since all methods are variants of ensembles of decision trees, average accuracies are not strikingly different from one method to another (see Table 1 of the supplementary materials). Yet, significant trends appear when looking at Figure 1 and Table 2. First, all ET-based methods are ranked before DT-based methods, including the popular Random Forest algorithm. Overall, the original ET algorithm is ranked first ($R_{ET} = 2.125$), then come RS-ET and RP-ET at close positions ($R_{RS-ET} = 2.8125$ and $R_{RP-ET} = 2.9375$) while P-ET is a bit behind

[1] These datasets were chosen a priori and independently of the results obtained.

[2] `http://www.montefiore.ulg.ac.be/~glouppe/pdf/ecml12-suppl.pdf`

Table 1: Small datasets

Dataset	N_s	N_f
DIABETES	768	8
DIG44	18000	16
IONOSPHERE	351	34
PENDIGITS	10992	16
LETTER	20000	16
LIVER	345	6
MUSK2	6598	168
RING-NORM	10000	20
SATELLITE	6435	36
SEGMENT	2310	19
SONAR	208	60
SPAMBASE	4601	57
TWO-NORM	9999	20
VEHICLE	1692	18
VOWEL	990	10
WAVEFORM	5000	21

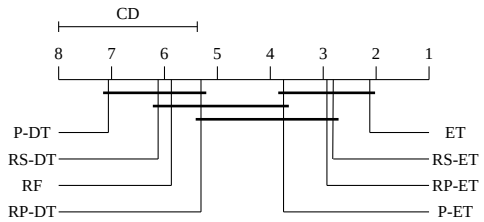


Fig. 1: Average ranks of all methods

Table 2: Pairwise t-test comparisons

	RF	ET	P-DT	P-ET	RS-DT	RS-ET	RP-DT	RP-ET
RF	—	1/2/13	12/4/0	1/7/8	4/7/5	2/2/12	1/10/5	0/4/12
ET	13/2/1	—	14/1/1	10/5/1	13/3/0	4/11/1	12/2/2	5/10/1
P-DT	0/4/12	1/1/14	—	0/4/12	2/3/11	2/1/13	0/4/12	0/4/12
P-ET	8/7/1	1/5/10	12/4/0	—	9/6/1	2/6/8	9/6/1	0/11/5
RS-DT	5/7/4	0/3/13	11/3/2	1/6/9	—	0/2/14	1/11/4	0/4/12
RS-ET	12/2/2	1/11/4	13/1/2	8/6/2	14/2/0	—	11/4/1	1/13/2
RP-DT	5/10/1	2/2/12	12/4/0	1/6/9	4/11/1	1/4/11	—	0/6/10
RP-ET	12/4/0	1/10/5	12/4/0	5/11/0	12/4/0	2/13/1	10/6/0	—

($R_{P-ET} = 3.75$). According to Figure 1, only ET is ranked significantly higher than all DT-based method but looking at Table 2, the worse ET-based variant (P-ET) is still 9 times significantly better (w.r.t. the 50 runs over each set) and only 1 times significantly worse than the best DT-based variant (RP-DT). The separation between these two families of algorithm thus appears quite significant. This observation clearly suggests that using random split thresholds, instead of optimized ones like in decision trees, pays off in terms of generalization.

Among ET-based methods, RP-ET is better than P-ET but it is superseded by ET and RS-ET in terms of average rank. Since RS-ET is a particular case of RP-ET, this suggests that we are slightly overfitting when tuning the additional parameter p_s. And indeed RP-ET is better ranked than RS-ET in average on the validation set (results not shown). Table 2 however indicates otherwise and makes RP-ET appear as slightly better than RS-ET (2/13/1). Regarding ET over RP-ET, the better performance of the former (5/10/1) is probably due to the fact that in ET subsets of features are redrawn locally at each node when building trees and not once and for all prior to their construction. This gives less chances to generate improper trees because of a bad initial choice of features and thus leads to a lower bias and a better accuracy.

Among DT-based methods, RP-DT now comes first (mean rank of 5.3125), then RF ($R_{RF} = 5.875$), RS-DT ($R_{RS-DT} = 6.125$) and then P-DT in last ($R_{P-DT} = 7.0625$). RP is only significantly worse than another DT-based variant on 1 dataset. The extra-randomization brought by the random choices of both samples and features seems to be beneficial with decision trees that do

not benefit from the randomization of discretization thresholds. The fact that RF samples features locally does not appear here anymore as an advantage over RP (RF is significantly worse on 5 problems and better on only one), probably because the decrease of bias that it provides does not exceed the increase of variance with respect to global feature selection.

3.3 Larger datasets

While the former experiments revealed promising results, it is fair to ask whether the conclusions that have been drawn would hold on and generalize to larger problems, for example when dealing with a few relevant features buried into hundreds or thousands of not important features (e.g., in genomic data), or when dealing with many correlated features (e.g., in images). To investigate this question, a second bench of experiments was carried out on 13 larger datasets (see Table 3). All but MADELON are real data. In terms of dimensions, these datasets are far bigger, ranging from a few hundreds of samples and thousands of features, to thousands of samples but hundreds of features. As such, the complexity of the problems is expected to be greater. We adopted the exact same protocol as for smaller datasets. However, to lower computing times, for datasets marked with $*$, the methods were run using 100 trees instead of 250 and the minimum number of samples required in an internal node was set to 10 in order to control complexity. Detailed results are provided in Table 2 of the supplementary materials and are summarized in Figure 2 and Table 4, respectively in terms of average rank (the critical difference at $\alpha = 0.05$ is now 2.9120) and Win/Draw/Loss statuses obtained with paired t-tests. A Friedman test (at $\alpha = 0.05$) still indicates that some methods are significantly different from the others.

As it may be observed from Figure 2, the average ranks of the methods are closer to each other than in the previous experiments, now ranging from 2.38 to 6.61, while they were previously ranging from 2.12 to 7. Methods are more connected by critical difference bars. This suggests that overall they behave more similarly to each other than before. General trends are nevertheless comparable to what we observed earlier. ET-based methods still seem to be the front-runners. From Figure 2, RS-ET, ET and RP-ET are in the top 4, while P-DT, RF and RP-DT remain in the second half of the ranking. Surprisingly however, RS-DT now comes right after RS-ET and ET and just before RP-ET whereas it ranked penultimate on the smaller datasets. Table 4 however suggests that RS-DT performs actually a little worse against RP-ET (1/10/2). All in all, it thus still seems beneficial to randomize split thresholds on the larger datasets.

Comparing ET-based variants, ET is no longer the best method on average, but RS-ET is (with 4/9/0 for RS-ET versus ET). This suggests than on larger datasets, picking features globally at random prior to the construction of the trees is as good, or even beat picking them locally at each node. Due to the quantitatively larger number of samples in a patch, and also to the larger number of redundant features expected in some large datasets (e.g., in CIFAR10 or MNIST), it is indeed less likely to build improper trees with strong biases. As a result, variance can be further decreased by sampling globally. In support

Table 3: Large datasets

Dataset	N_s	N_f
CIFAR10*	60000	3072
MNIST3VS8	13966	784
MNIST4VS9	13782	784
MNIST*	70000	784
ISOLET	7797	617
ARCENE	900	10000
BREAST2	295	24496
MADELON	4400	500
MARTI0	500	1024
REGED0	500	999
SECOM	1567	591
TIS	13375	927
SIDO0*	12678	4932

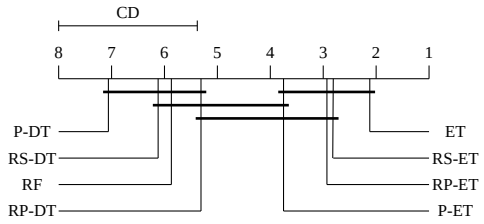


Fig. 2: Average ranks of all methods

Table 4: Pairwise t-test comparisons

	RF	ET	P-DT	P-ET	RS-DT	RS-ET	RP-DT	RP-ET
RF	—	1/5/7	8/3/2	2/6/5	0/6/7	0/5/8	0/6/7	0/6/7
ET	7/5/1	—	9/2/2	7/6/0	3/7/3	0/9/4	5/6/2	1/11/1
P-DT	2/3/8	2/2/9	—	1/5/7	0/3/10	0/3/10	1/3/9	0/4/9
P-ET	5/6/2	0/6/7	7/5/1	—	0/6/7	0/5/8	2/5/6	1/5/7
RS-DT	7/6/0	3/7/3	10/3/0	7/6/0	—	1/8/4	2/11/0	1/10/2
RS-ET	8/5/0	4/9/0	10/3/0	8/5/0	4/8/1	—	4/8/1	0/13/0
RP-DT	7/6/0	2/6/5	9/3/1	6/5/2	0/11/2	1/8/4	—	1/9/3
RP-ET	7/6/0	1/11/1	9/4/0	7/5/1	2/10/1	0/13/0	3/9/1	—

of this claim, on a few problems such as ARCENE, BREAST2, or MADELON that contain many irrelevant features, ET remains the best method. In that case, it is indeed more likely to sample globally improper random patches, and hence to build improper trees. The average rank of RP-ET suggests that it performs worse than RS-ET and thus that there is some potential overfitting when tuning p_s in addition to p_f. This difference is however not confirmed in Table 4 where the accuracies of these two methods are shown to be never significantly different (0/13/0). RP-ET is also on a perfect par with ET (1/11/1). Among DT-based variants, RP-DT, which was the best performer on small datasets, is still ranked above RF and P-DT, but it is now ranked below RS-DT with a win/draw/loss of 0/11/2. This is again due to some overfitting.

While less conclusive than before, the results on larger datasets are consistent with what we observed earlier. In particular, they indicate that the Random Patches method (with ET) remains competitive with the best performers.

3.4 Conclusions

Overall, this extensive experimental study reveals many interesting results. The first and foremost result is that ensembles of randomized trees nearly always beat ensembles of standard decision trees. As off-the-shelf methods, we advocate that ensembles of such trees should be preferred to ensembles of decision trees. In particular, these results show that the well-known Random Forest algorithm does not compete with the best performers. Far more important to our concern though, this study validates our RP approach. Building ensembles (of ET) on random patches of data is competitive in terms of accuracy. Overall, there is no strong statistical evidence that the method performs less well, but there is also no

conclusive evidence that it significantly improves performance. Yet, results show that RP is often as good as the very best methods. Regarding the shape of the random patches, the strategy behind Pasting (i.e., p_s free and $p_f = 1.0$) proved to be (very) ineffective on many datasets while the Random Subspace algorithm (i.e., $p_s = 1.0$ and p_f free) always ranked among the very best performers. On average, RS indeed came in second on the small datasets and in first on the larger datasets, which tends to indicate that sampling features is crucial in terms of accuracy. As for patches of freely adjustable size (i.e., using RP), they showed to be slightly sensitive to overfitting but proved to remain closely competitive with the very best methods. In addition, these results also suggest that sampling features globally, once and for all, prior to the construction of a (randomized) decision tree, does not actually impair performance. For instance, RS-ET or RP-ET are indeed not strongly worse, nor better, than ET, in which candidates features are re-sampled locally at each node.

4 On Memory

Section 3 reveals that building an ensemble of base estimators on random patches, instead of the whole data, is a competitive strategy. In the context of big data, that is when the size of the dataset is far bigger than the available memory, this suggests that using random parts of the data of the appropriate size to build each base estimator would likely result in an ensemble which is actually as good as if the whole data could have been loaded and used.

Formally, we assume a general framework where the number of data units that can be loaded at once into memory is constrained to be lower than a given threshold M_{max}. Not considering on-line algorithms within the scope of this study, M_{max} can hence be viewed as the total units of data allowed to be used to build a single base estimator. In the context of our sampling methods, the amount of memory required for a patch is given by $(p_s N_s)(p_f N_f)$ and thus constraining memory by M_{max} is equivalent to constraining the relative patch size $p_s p_f$ to be lower than $M'_{max} = M_{max}/(N_s N_f)$. While simplistic[3], this framework has the advantage of clearly addressing one of the main difficulties behind big data, that is the lack of fast memory. Yet, it is also relevant in other contexts, for example when data is costly to access (e.g., on remote locations) or when algorithms are run on embedded systems with strong memory constraints.

In Section 4.1, we first study the effects of p_s and p_f on the accuracy of the resulting ensemble and show that it is problem and base estimator dependent. Second, we show that the memory requirements, i.e., the relative size $p_s p_f$ of the random patches, can often be drastically reduced without significantly degrading the performance of the ensemble (Section 4.2). Third, because the sensitivity of the ensemble to p_s and p_f is problem and base estimator specific, we show that under very strong memory constraints adjusting both parameters at the same time, as RP does, is no longer merely as good but actually significantly better than other ensemble methods (Section 4.3).

[3] e.g., the quantity of memory used by the estimator itself is not taken into account.

4.1 Sensitivity to p_s and p_f

Let us first consider and analyze the sets $\{(p_s, p_f, Acc_D(p_s, p_f))|\forall p_s, p_f\}$ for various problems, where $Acc_D(p_s, p_f)$ is the average test accuracy of an ensemble built on random patches of size $p_s p_f$ (using the same protocol as previously) on the dataset D.

As Figure 3 illustrates for six datasets, the surfaces defined by these sets vary significantly from one problem to another. We observed four main trends. In Figures 3a, and 3b (resp. 3c), accuracy increases with p_s (resp. p_f) while adjusting p_f (resp. p_s) has no or limited impact. In Figure 3d, the best strategy is to increase both p_s and p_f. Finally, in Figures 3e and 3f, the surface features plateaus, which means that beyond some threshold, increasing p_s or p_f does not yield any significant improvement. Interestingly, in most of the cases, the optimum corresponds to a value $p_s p_f$ much smaller than 1.

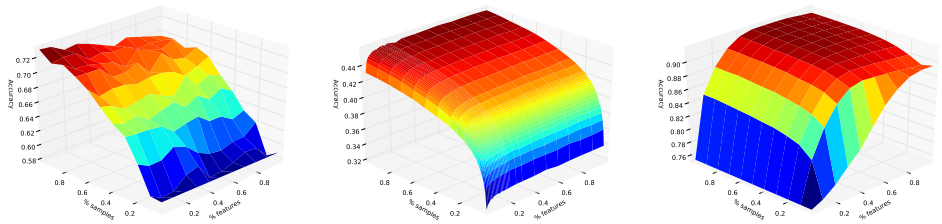
The choice of the base estimators does not have a strong impact on the aspect of the curves (compare the 1^{st} and 3^{rd} rows of sub-figures in Figure 3 with those in the 2^{nd} and 4^{th} rows). The only difference is the decrease of the accuracy of RP-DT when p_s and p_f grow towards 1.0. Indeed, since the only source of randomization in RP-DT is patch selection, it yields in this case ensembles of identical trees and therefore amounts to build a single tree on the whole dataset. By contrast, because of the extra-randomization of the split thresholds in ET, there is typically no drop of accuracy for RP-ET when p_s and p_f grow to 1.0.

Overall, this analysis suggests that not only the best pair $p_s p_f$ depends on the problem, but also that the sensitivity of the ensemble to changes to the size of a random patch is both problem and base estimator specific. As a result, these observations advocate for a method that could favor p_s, p_f or both, and do so appropriately given the base estimator.

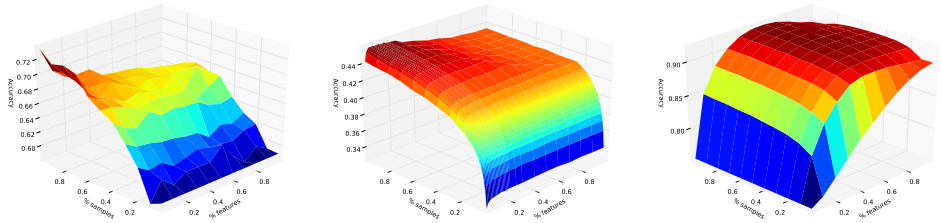
4.2 Memory reduction, without significant loss

We proceed to study in this section the actual size of the random patches when the values of p_s and p_f are tuned using an independent validation set. Our results are summarized in Figure 4a. Each ellipse corresponds to one of the 29 datasets of our benchmark, whose center is located at $(\overline{p_s}, \overline{p_f})$ (i.e., the average parameter values over the 50 runs) and whose semi-axes correspond to the standard deviations of p_s and p_f. Any point in the plot corresponds to a pair (p_s, p_f) and thus to a relative consumption $M' = p_s p_f$ of memory. To ease readability, level curves are plotted for $M' = 0.01, 0.1, ..., 0.9$. In the right part of the figure, the histogram counts the number of datasets such that $\overline{p_s} \cdot \overline{p_f}$ falls in the corresponding level set.

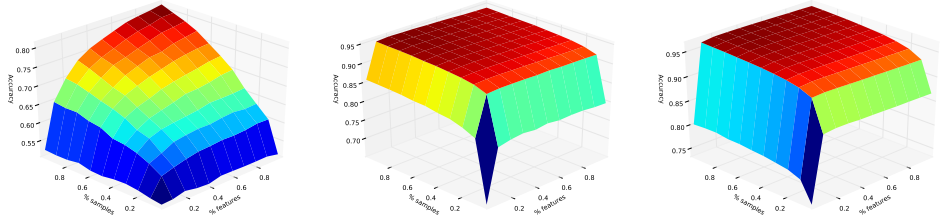
Figure 4a corroborates our previous discussion. On some datasets, it is better to favor p_s while on some other increasing p_f is a better strategy. The various sizes of the ellipses also confirm that the sensitivity to variations of p_s and p_f is indeed problem-specific. The figure also clearly highlights the fact that, even under no memory constraint, the optimal patches rarely consume the whole memory. A majority of ellipses indeed lie below the level set $M' = 0.5$ and only



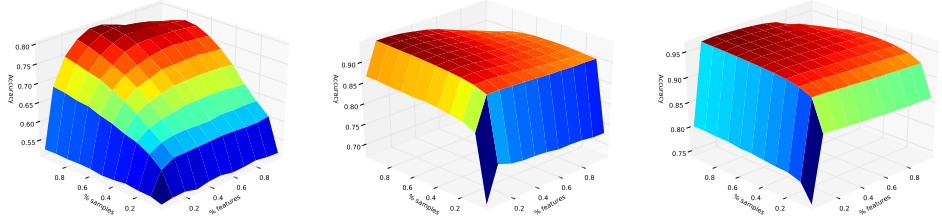
(a) ARCENE (RP-ET) (b) CIFAR10 (RP-ET) (c) TIS (RP-ET)



(a) ARCENE (RP-DT) (b) CIFAR10 (RP-DT) (c) TIS (RP-DT)

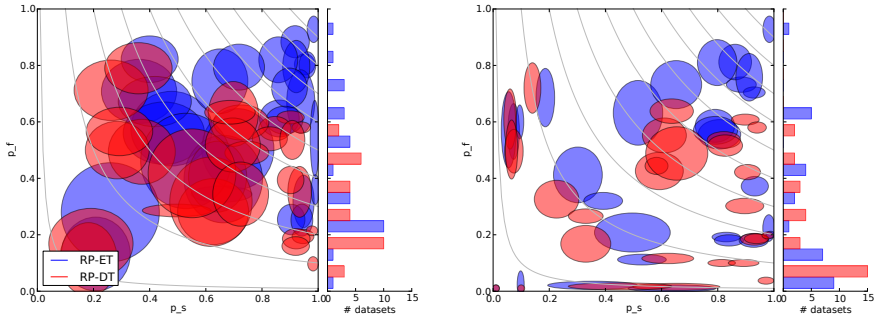


(d) MADELON (RP-ET) (e) ISOLET (RP-ET) (f) MNIST3VS8 (RP-ET)



(d) MADELON (RP-DT) (e) ISOLET (RP-DT) (f) MNIST3VS8 (RP-DT)

Fig. 3: Learning surfaces.



(a) Original patches (b) Reduced patches

Fig. 4: Optimal sizes of the random patches on our benchmark.

a couple of them are above $M' = 0.75$. With respect to ET or RF for which the base estimators are all built on the whole dataset, this means that ensembles of patches are not only as competitive but also less memory greedy. In addition, the figure also points out the difference between RP-ET and RP-DT as discussed in the previous section. To ensure diversity, RP-DT is constrained to use smaller patches than RP-ET, hence explaining why the ellipses in red are on average below those in blue. While RP-DT proved to be a bit less competitive in terms of accuracy, this indicates on the other hand that RP-DT may actually be more interesting from a memory consumption point of view.

In Section 4.1, we observed plateaus or very gentle slopes around the optimal pair (p_s, p_f). From a memory point of view, this suggests that the random patches are likely to be reducible without actually degrading the accuracy of the resulting ensemble. Put otherwise, our interest is to find the smallest size $p_s p_f$ such that the accuracy of the resulting ensemble is not significantly worse than an ensemble built without such constraint. To that end, we study the extent at which the constraint $p_s p_f < M'_{max}$ can be strengthened without any significant drop in accuracy. If M'_{max} can be reduced significantly then it would indeed mean that even when only small parts of the data are actually used to build single base estimators, competitive performance can still be achieved.

Figure 4b summarizes our results. For all datasets, M'_{max} was set to the lowest value such that it cannot be statistically detected that the average accuracy of the resulting ensemble is different from the average accuracy of an ensemble built with no memory constraint (at $\alpha = 0.05$). With regard to Figure 4a, the shift of most ellipses to lower memory level sets confirm our first intuition. In many cases, the size of the random patches can indeed be reduced, often drastically, without significant decrease of accuracy. For more than half of the datasets, memory can indeed be decreased to $M' = 0.1$ or $M' = 0.2$. In other words, building trees on small parts of the data (i.e., 10% or 20% of the original dataset) is, for more than half of the datasets, enough to reach competitive accuracy. Also, the sensitivity to p_s and p_f is now even more patent. Some ensembles use very few samples ($p_s < 0.1$) but with many features, while other uses many samples

with few features ($p_f < 0.1$). Again, from a memory point of view, RP-DT appears to be more interesting than RP-ET. The memory reduction is larger, as the histogram indicates. Optimized splits in the decision trees may indeed lead to a better exploitation of the data, hence to a potentially larger reduction of memory. In conclusion, while not detecting significant differences in accuracy does not allow to conclude that the performances are truly similar, these figures at least illustrate that memory requirements can be drastically reduced without apparent loss in accuracy.

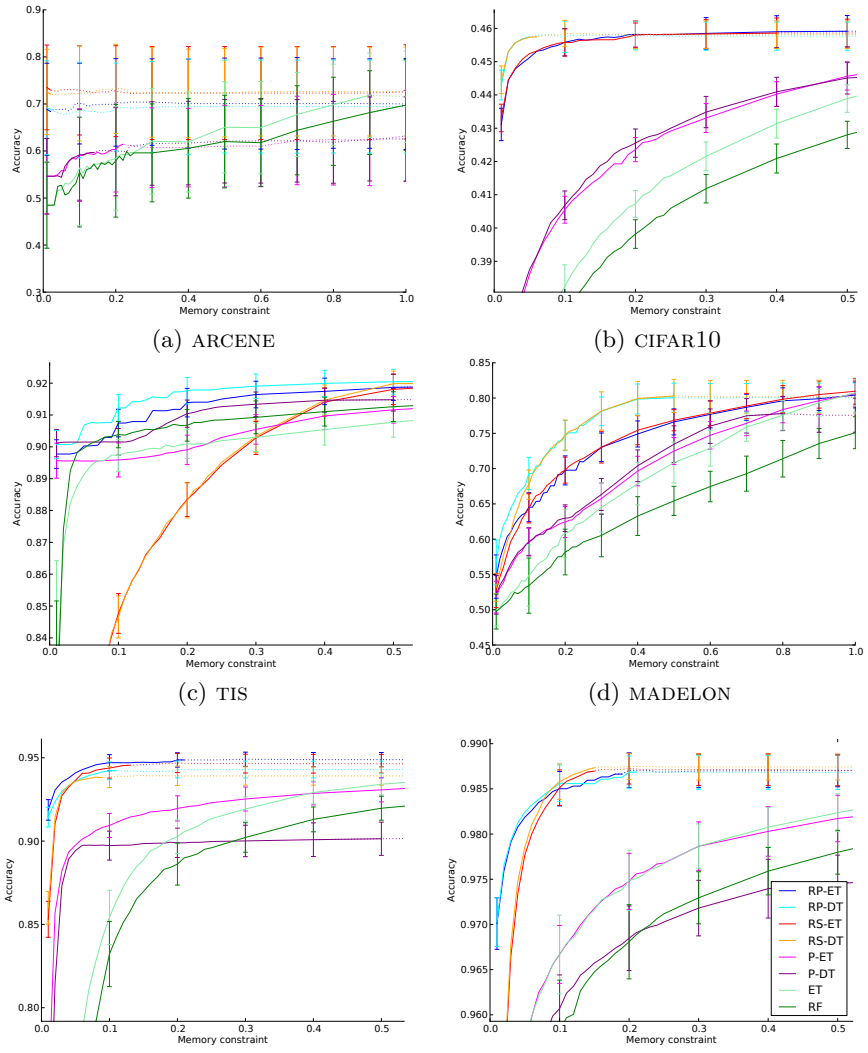
4.3 Memory reduction, with loss

The previous section has shown that the memory consumption can be reduced up to some threshold M'_{max} with no significant loss in accuracy. In this section we now look at the accuracy of the resulting ensemble when M'_{max} is further decreased. We argue that with severe constraints, and because datasets have all a different sensitivity, it is even more crucial to better exploit data and thus to find the right trade-off between both p_s and p_f, as only RP can.

To illustrate our point, Figure 5 compares for 6 representative datasets the accuracy of the methods with respect to the memory constraint $p_s p_f < M'_{max}$. A plain line indicates that the generalization error of the best resulting ensemble under memory constraint M'_{max} is significantly (at $\alpha = 0.05$) worse on the test sets than when there is no constraint (i.e., $M'_{max} = 1$). A dotted line indicates that on average, on the test set, the ensemble is not significantly less accurate.

As the figure shows, when M'_{max} is low, RP-based ensembles often achieve the best accuracy. Only on ARCENE (Figure 5a), RS seems to be a better strategy, suggesting some overfitting in setting p_s in RP. On all 5 other example datasets, RP is equivalent or better than RS and P for low values of M'_{max}, with the largest gaps appearing on ISOLET (Figure 5e) and MNIST3VS8 (Figure 5f). As already observed in the previous section, although RP-DT is not the best strategy when memory is unconstrained, its curve dominates the curve of RP-ET for small values of M'_{max} in Figures 5b, 5c, and 5d. Because split thresholds are not randomized in RP-DT, this method is more resistant than RP-ET to the strong randomization induced by a very low M'_{max} threshold.

For comparison, Figure 5 also features the learning curves of both ET and RF (with K optimized on the validation set), in which the trees have all been built on the *same* training sample of $M'_{max} N_s$ instances, with all features. These results are representative of the use of a straightforward sub-sampling of the instances to handle the memory constraint. On all datasets, this setting yields very poor performance when M'_{max} is low. Building base estimators on re-sampled random patches thus brings a clear advantage to RP, RS and P and hence confirms the conclusions of Basilico et al who showed in [8] that using more data indeed produces more accurate models than learning from a single subsample. This latter experiment furthermore shows that the good performances of RP cannot be trivially attributed to the fact that our datasets contain so many instances that only processing a subsample of them would be enough. On most problems, the slopes of the learning curves of RF and ET indeed suggest that convergence



(a) ARCENE (b) CIFAR10

(c) TIS (d) MADELON

(e) ISOLET (f) MNIST3VS8

Fig. 5: Accuracy under memory constraint.

has not yet been reached on these datasets. Yet, important improvement are gained by sub-sampling random patches. Overall, these results thus indicate that building an ensemble on random patches is not only a good strategy when data is abundant and redundant but also that it works even for scarce datasets with limited information regarding the problem.

4.4 Conclusion

We have shown in this section that the memory requirements of sampling-based ensembles are intrinsically low. Better, we have shown that they can often be drastically decreased without significant loss in accuracy. When the size of the dataset is far bigger than the available memory, we have also demonstrated that sampling data along both samples and features, as RP does, not only competes with other ensemble algorithms but also significantly improves the accuracy of the resulting ensemble. It also brings a significant improvement over a straightforward sub-sampling of the instances.

5 Conclusions and future work

The main contribution of this paper is to explore a new framework for supervised learning in the context of very strong memory constraints or, equivalently, very large datasets. To address such problems, we proposed the Random Patches ensemble method that builds each individual model of the ensemble from a random patch of the dataset obtained by drawing random subsets of both samples and features from the whole dataset. Through extensive experiments with tree-based estimators, we have shown that this strategy works as well as other popular randomization schemes in terms of accuracy (Section 3), at the same time reduces very significantly the memory requirements to build each individual model (Section 4.2), and, given its flexibility, attains significantly better accuracy than other methods when memory is severely constrained (Section 4.3). Since all models are built independently of each other, the approach is furthermore trivial to parallelize. All in all, we believe that the paradigm of our method highlights a very promising direction of research to address supervised learning on big data.

There remain several open questions and limitations to our approach that we would like to address in the future. First, this study motivates our interest in experimenting with truly large-scale problems (of giga-scale and higher). Since RP already appears advantageous for small to medium datasets, the potential benefits on very large-scale data indeed look very promising.

Second, the conclusions drawn in sections 3 and 4 are all based on the optimal values of the parameters p_s and p_f tuned through an exhaustive grid search on the validation set. Our analyses did not account for the memory and time required for tuning these two parameters. In practice, hyper-parameter tuning can not be avoided as we have shown that the optimal trade-off between p_f and p_s was problem dependent. It would therefore be interesting to design an efficient strategy to automatically find and adjust the values of p_s and p_f, taking

into account the global memory constraint. Our simplistic framework also only accounts for the memory required to store the training set in memory and not for the total memory required to actually build the ensemble.

We have only explored uniform sampling of patches of fixed size in our experiments. In the context of the Pasting approach, Breiman proposed an iterative instance weighting scheme that proved to be more efficient than uniform sampling [1]. It would be interesting to extend this approach when sampling both instances and features. Yet, parallelization would not be trivial anymore, although probably still possible in the line of the work in [7].

Finally, our analysis of RP is mostly empirical. In the future, we would like to strengthen these results with a more theoretical analysis. A starting point could be the work in [13] that studies a scheme similar to the Pasting method applied to linear models trained through parallel stochastic gradient descent. The extension of this work to non parametric tree-based estimators does not appear trivial however, since these latter are not well characterized theoretically.

Acknowledgements. The authors would like to thank Raphaël Marée and the reviewers for their helpful feedback. GL and PG are respectively research fellow and research associate of the FNRS, Belgium. This work is supported by PASCAL2 and the IUAP DYSCO, initiated by the Belgian State, Science Policy Office.

References

1. Breiman, L.: Pasting small votes for classification in large databases and on-line. Machine Learning **36**(1) (1999) 85–103
2. Ho, T.: The random subspace method for constructing decision forests. Pattern Analysis and Machine Intelligence, IEEE Transactions on **20**(8) (1998) 832–844
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and regression trees. (1984)
4. Breiman, L.: Bagging predictors. Machine learning **24**(2) (1996) 123–140
5. Breiman, L.: Random forests. Machine learning **45**(1) (2001) 5–32
6. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine Learning **63**(1) (2006) 3–42
7. Chawla, N.V., Hall, L.O., Bowyer, K.W., Kegelmeyer, W.P.: Learning ensembles from bites: A scalable and accurate approach. J. Mach. Learn. Res. **5** (December 2004) 421–451
8. Basilico, J., Munson, M., Kolda, T., Dixon, K., Kegelmeyer, W.: Comet: A recipe for learning and using large ensembles on massive data. In: Data Mining (ICDM), 2011 IEEE 11th International Conference on, IEEE (2011) 41–50
9. Panov, P., Džeroski, S.: Combining bagging and random subspaces to create better ensembles. Advances in intelligent data analysis VII (2007) 118–129
10. Pedregosa, F., et al.: Scikit-learn: Machine Learning in Python . Journal of Machine Learning Research **12** (2011) 2825–2830
11. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
12. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research **7** (2006) 1–30
13. Zinkevich, M., Weimer, M., Smola, A., Li, L.: Parallelized stochastic gradient descent. In Lafferty, J., Williams, C.K.I., Shawe-Taylor, J., Zemel, R., Culotta, A., eds.: Advances in Neural Information Processing Systems 23. (2010) 2595–2603