Mach Learn () :
DOI 10.1007/s10994-006-6226-1

Extremely randomized trees

Pierre Geurts · Damien Ernst · Louis Wehenkel

Received: 14 June 2005 / Revised: 29 October 2005 / Accepted: 15 November 2005 /
Published online: 2 March 2006

Abstract This paper proposes a new tree-based ensemble method for supervised classification and regression problems. It essentially consists of randomizing strongly both attribute and cut-point choice while splitting a tree node. In the extreme case, it builds totally randomized trees whose structures are independent of the output values of the learning sample. The strength of the randomization can be tuned to problem specifics by the appropriate choice of a parameter. We evaluate the robustness of the default choice of this parameter, and we also provide insight on how to adjust it in particular situations. Besides accuracy, the main strength of the resulting algorithm is computational efficiency. A bias/variance analysis of the Extra-Trees algorithm is also provided as well as a geometrical and a kernel characterization of the models induced.

Keywords Supervised learning · Decision and regression trees · Ensemble methods · Cut-point randomization · Bias/variance tradeoff · Kernel-based models

1. Introduction

In this article, we propose a new tree induction algorithm that selects splits, both attribute and cut-point, totally or partially at random.

The idea that randomized decision trees could perform as well as classical ones appeared in an experimental study published in the late eighties (Mingers, 1989), even if later it was

Editor: Johannes Fürnkranz

P. Geurts (✉) · D. Ernst · L. Wehenkel
Department of Electrical Engineering and Computer Science, University of Liège,
Liège, Sart-Tilman, B-28, B-4000 Belgium
e-mail: P.Geurts@ulg.ac.be

D. Ernst
e-mail: Dernst@ulg.ac.be

L. Wehenkel
e-mail: L.Wehenkel@ulg.ac.be

shown in a more carefully designed experiment that they were actually significantly less accurate than normal ones on many datasets (Buntine and Niblett, 1992).

During the early nineties, the statistical notions of *variance* and its companion, the *bias*, were studied more systematically by machine learning researchers (see for example, Dietterich and Kong, 1995; Breiman, 1996a; Friedman, 1997), and the high variance of tree-based models, like those induced by CART or C4.5, was acknowledged by the research community. Actually, because of this high variance, the models induced by these latter methods are to a large extent random, and also the splits, both attributes and cut-points, that are chosen at each internal node depend to a very large extent on the random nature of the learning sample (Wehenkel, 1997; Geurts and Wehenkel, 2000; Geurts, 2002). In particular, in (Wehenkel, 1997) it was shown empirically that the cut-point variance is very high, even for large sample sizes. More precisely, the optimal cut-point (the one maximizing, for a given problem and a given attribute, the score computed on the learning sample) was shown to depend very strongly on the particular learning sample used. Furthermore, this cut-point variance appeared to be responsible for a significant part of the error rates of tree-based methods (see, e.g., Geurts, 2002). Thus, in order to try to reduce cut-point variance, various kinds of cut-point smoothing or averaging approaches have been tried out, but, while allowing to reduce cut-point variance and hence improve interpretability, they did not translate into significantly better predictive accuracy (Geurts and Wehenkel, 2000).

During the same early nineties period, various ideas to combine multiple models emerged with the objective of reducing variance or bias in machine learning methods. For example, Bayesian averaging (Buntine and Weigend, 1991; Buntine, 1992) is essentially a variance reduction technique, whereas Stacking and Boosting (Wolpert, 1992; Freund and Schapire, 1995) essentially target a reduction of bias (actually, Boosting also reduces variance; Bauer and Kohavi, 1999). Breiman came up in 1994 with the "Bagging" idea (Breiman, 1996b) in order to reduce the variance of a learning algorithm without increasing its bias too much. One can view Bagging as a particular instance of a broader family of ensemble methods that we will call *randomization methods*. These methods explicitly introduce randomization into the learning algorithm and/or exploit at each run a different randomized version of the original learning sample, so as to produce an ensemble of more or less strongly diversified models. Then, the predictions of these models are aggregated by a simple average or a majority vote in the case of classification. For example, Bagging introduces randomization by building the models from bootstrap samples drawn with replacement from the original dataset. Several other generic randomization methods have been proposed which, like Bagging, are applicable to any machine learning algorithm (e.g., Ho, 1998; Bauer and Kohavi, 1999; Webb, 2000; Breiman, 2000a). These generic methods often improve considerably the accuracy of decision or regression trees which, otherwise, are often not competitive with other supervised learning algorithms like neural networks or support vector machines. Furthermore, although ensemble methods require growing several models, their combination with decision/regression trees remains also very attractive in terms of computational efficiency because of the low computational cost of the standard tree growing algorithm. Hence, given the success of generic ensemble methods with trees, several researchers have looked at specific randomization techniques for trees based on a direct randomization of the tree growing method (e.g., Ali and Pazzani, 1996; Ho, 1998; Dietterich, 2000; Breiman, 2001; Cutler and Guohua, 2001; Geurts, 2002; Kamath et al., 2002). All these randomization methods actually cause perturbations in the induced models by modifying the algorithm responsible for the search of the optimal split during tree growing. In the context of an ensemble method, it is thus productive to somewhat deteriorate the "optimality" of the classical deterministic induction algorithm that looks for the best split locally at each tree node.

Although existing randomization methods for trees significantly randomize the standard tree growing algorithm, they are still far from building totally random trees. Yet, the very high variance of decision and regression tree splits suggests investigating whether higher randomization levels could improve accuracy with respect to existing ensemble methods. To this end, the present paper proposes and studies a new algorithm that, for a given numerical attribute, selects its cut-point fully at random, i.e., independently of the target variable. At each tree node, this is combined with a random choice of a certain number of attributes among which the best one is determined. In the extreme case, the method randomly picks a *single* attribute and cut-point at each node, and hence builds *totally randomized trees* whose structures are independent of the target variable values of the learning sample. While we also propose a way to select random splits for categorical attributes, we focus in this paper on the study of this randomization idea in the context of numerical attributes only.

The rest of the paper is organized as follows: Section 2 introduces the Extra-Trees (for extremely randomized trees) algorithm with its default parameter settings, and carries out a systematic empirical evaluation in terms of both accuracy and computational efficiency; Section 3 provides a detailed analysis of the effect of parameters in different conditions. Section 4 presents an empirical study of bias and variance of the Extra-Trees algorithm and Section 5 analyses the main geometrical properties of the Extra-Trees models. The paper ends with a discussion of related work, conclusions and suggestions for future work directions. The appendices collect relevant details and complementary simulations results.

2. Extra-Trees algorithm

We consider the standard *batch-mode* supervised learning problem, and focus on learning problems characterized by (possibly a large number of) numerical input variables and one single (categorical or numerical) target variable. We start with the description of the Extra-Trees (ET) algorithm and a brief discussion of its rationale. We continue with a systematic empirical evaluation based on a diverse set of classification and regression problems, where we compare this new method with standard tree-based methods, in terms of both accuracy and computational efficiency.

In the rest of the paper, the term *attribute* denotes a particular input variable used in a supervised learning problem. The *candidate attributes* denote all input variables that are available for a given problem. We use the term *output* to refer to the target variable that defines the supervised learning problem. When the output is categorical, we talk about a *classification problem* and when it is numerical, we talk about a *regression problem*. The term *learning sample* denotes the observations used to build a model, and the term *test sample* the observations used to compute its accuracy (error-rate, or mean square-error). N refers to the size of the learning sample, i.e., its number of observations, and n refers to the number of candidate attributes, i.e., the dimensionality of the input space.

2.1. Algorithm description and rationale

The Extra-Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other tree-based ensemble methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees.

Table 1 Extra-Trees splitting algorithm (for numerical attributes)

Split_a_node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
– If **Stop_split**(S) is TRUE then return nothing.
– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;
– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;
– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Pick_a_random_split(S,a)
Inputs: a subset S and an attribute a
Output: a split
– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;
– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
– Return the split $[a < a_c]$.

Stop_split(S)
Input: a subset S
Output: a boolean
– If $|S| < n_{\min}$, then return TRUE;
– If all attributes are constant in S, then return TRUE;
– If the output is constant in S, then return TRUE;
– Otherwise, return FALSE.

The Extra-Trees splitting procedure for numerical attributes is given in Table 1.[1] It has two parameters: K, the number of attributes randomly selected at each node and $n_{\min}$, the minimum sample size for splitting a node. It is used several times with the (full) original learning sample to generate an ensemble model (we denote by M the number of trees of this ensemble). The predictions of the trees are aggregated to yield the final prediction, by majority vote in classification problems and arithmetic average in regression problems.

From the bias-variance point of view, the rationale behind the Extra-Trees method is that the explicit randomization of the cut-point and attribute combined with ensemble averaging should be able to reduce variance more strongly than the weaker randomization schemes used by other methods. The usage of the full original learning sample rather than bootstrap replicas is motivated in order to minimize bias. From the computational point of view, the complexity of the tree growing procedure is, assuming balanced trees, on the order of N log N with respect to learning sample size, like most other tree growing procedures. However, given the simplicity of the node splitting procedure we expect the constant factor to be much smaller than in other ensemble based methods which locally optimize cut-points.

The parameters K, $n_{\min}$ and M have different effects: K determines the strength of the attribute selection process, $n_{\min}$ the strength of averaging output noise, and M the strength of the variance reduction of the ensemble model aggregation. These parameters could be adapted to the problem specifics in a manual or an automatic way (e.g. by cross-validation). However, we prefer to use default settings for them in order to maximize the computational advantages and autonomy of the method. Section 3 studies these default settings in terms of robustness and suboptimality in various contexts.

[1] The complete Extra-Trees algorithm, for numerical and categorical attributes, is given in Appendix A.

To specify the value of the main parameter K, we will use the notation ET^K, where K is replaced by 'd' to say that default settings are used, by '$*$' to denote the best results obtained over the range of possible values of K, and by 'cv' if K is adjusted by cross-validation.

2.2. Empirical evaluation

We now present an empirical evaluation of the performance of the Extra-Trees method with default settings. We first describe the datasets used for this purpose and discuss the range of conditions that are intended to be covered by them. Then we specify the algorithms (all are tree-based) with which we compare our method and the simulation protocol used to evaluate them. Finally, we provide and discuss the results in terms of accuracy and computational requirements.

2.2.1. Datasets

We have used 24 different datasets. Half of them concern classification problems, with a number of classes ranging from 2 to 26; the other half are regression problems. Overall, these datasets cover a wide range of conditions in terms of number of candidate attributes (between 2 and 617), learning sample size (between 300 and 10,000), observation redundancy (number N/n of observations per attribute between 10 and 1000). In terms of the relative importance of candidate attributes, we have datasets with a number of totally irrelevant variables, with equally important variables and with attributes of variable importance. In terms of problem complexity, we cover linear problems as well as strongly non-linear ones. Some problems present high noise, others are noise free. All datasets are well known and publicly available. Out of the 24 problems, 13 are synthetic ones, and for three of them the explicit form of the Bayes optimal model and its residual error are known. Notice that we have restricted our choice to datasets which originally had no missing values and only numerical attributes.[2] The choice of datasets was made a priori and independently of the results obtained with our methods, and no dataset was later excluded.

Appendix B provides further details concerning these datasets, in particular their size and numbers of candidate attributes and classes, and a brief discussion of their origins and results obtained by standard methods such as k-nearest neighbors (kNN) and linear models.

2.2.2. Compared algorithms

Below, the compared methods are given in the order of their publication. Except for the first one, they all build ensembles of trees.

2.2.2.1. Single CART Tree (ST/PST). We use the CART algorithm to build single trees (Breiman et al., 1984). Trees are grown in a deterministic way from the learning sample and pruned according to the cost-complexity pruning algorithm with error estimates by ten-fold cross-validation. We will use the acronym ST (respectively, PST) to denote unpruned single trees (respectively, pruned single trees).

2.2.2.2. Tree Bagging (TB). When talking about Tree Bagging we refer to the *standard* algorithm published by Breiman (1996b). In this algorithm, unpruned CART trees are grown

[2] See also (Geurts et al., 2005a) for results with categorical attributes.

from bootstrap replicas (obtained by N times random sampling with replacement in the original learning sample). In Tree Bagging, attribute and cut-point randomization is thus carried out implicitly (and in a parameter free way) via the bootstrap re-sampling.

2.2.2.3. (Local) Random Subspace (RS). We consider the variant where this method randomizes locally the set of attributes used to determine an optimal split at each tree node (Ho, 1998). To this end, it looks for the best split over a subset of attributes, selected locally at each test node by drawing without replacement a number K of attributes from the candidate attributes. The trees are built from the original learning sample and the cut-point for a given attribute is optimized node-wise. The strength of the randomization is inversely proportional to the parameter K. We use the notation RS^K to denote this method, with $K = *$ to denote the best result found over the range of values of $K = 1, \ldots, n$.

2.2.2.4. Random Forests (RF). This algorithm has been proposed by Breiman (2001) as an enhancement of Tree Bagging. To build a tree it uses a bootstrap replica of the learning sample, and the CART algorithm (without pruning) together with the modification used in the Random Subspace method. At each test node the optimal split is derived by searching a random subset of size K of candidate attributes (selected without replacement from the candidate attributes). Empirical studies have shown that Random Forests significantly outperform Tree Bagging and other random tree ensemble methods in terms of accuracy. In terms of degree of randomization, this algorithm is stronger than Tree Bagging, specially if K is small compared to the number of attributes, n. It is also stronger than Random Subspace since it combines this method with bootstrap sampling. Randomization is both implicit (attribute and cut-point) and explicit (attribute). We use the notation RF^K, with $K = d$ for the default setting, and $K = *$ for the best result over the range $K = 1, \ldots, n$.

2.2.2.5. Parameter choices. Except for CART, all algorithms have the common parameter M (number of trees of the ensemble). We use for this parameter the common value of $M = 100$, which is large enough to ensure convergence of the ensemble effect with all our datasets. Another common parameter of the ensemble methods is $n_{\min}$, which we set to 2 for classification (fully grown trees) and to 5 in regression (slight smoothing). Finally, Random Subspace, Random Forests and Extra-Trees have the common parameter K. For our method and unless otherwise specified, we use the default setting in all trials, which is $K = \sqrt{n}$ in classification, rounded to the closest integer, and $K = n$ in regression. This choice is denoted by ET^d and further discussed in Section 3.1. For the purpose of accuracy assessments, we did run Random Subspace and Random Forests systematically with K ranging from 1 to n, and we report the best results obtained (i.e., RS^* and RF^*). For the computational assessment, we did however run Random Forests with the same default K (denoted by RF^d) as our method, so as to put these methods in comparable conditions from the computational point of view.

2.2.2.6. Score measures. We use the same score measure for all methods. In regression this is the amount of output variance reduction and in classification it is a normalized version of the Shannon information gain. Detailed score formulas are given in Appendix A.

2.2.2.7. Aggregation scheme. For all ensemble methods we used majority vote for classification problems and arithmetic average for regression problems.

Table 2 Win/Draw/Loss records (corrected t-tests) comparing the algorithm in the column versus the algorithm in the row

	Classification problems					Regression problems				
	PST	TB	RS*	RF*	ET^d	PST	TB	RS*	RF*	ET^d
PST	–	8/4/0	11/1/0	11/1/0	10/2/0	–	10/2/0	8/4/0	10/2/0	10/2/0
TB	0/4/8	–	7/5/0	7/5/0	7/5/0	0/2/10	–	0/9/3	1/11/0	4/8/0
RS*	0/1/11	0/5/7	–	0/8/4	2/10/0	0/4/8	3/9/0	–	4/8/0	4/7/1
RF*	0/1/11	0/5/7	4/8/0	–	5/7/0	0/2/10	0/11/1	0/8/4	–	3/7/2
ET^d	0/2/10	0/5/7	0/10/2	0/7/5	–	0/2/10	0/8/4	1/7/4	2/7/3	–

2.2.3. Protocols

The algorithms are run ten times on each dataset, except for smaller datasets where they are run 50 times (these are marked by a star in Table 7, Appendix B). At each run, each dataset is first randomly divided into a learning (LS) and test (TS) sample (their respective sizes are given in Table 7). Then all algorithms are run on the learning sample and their errors are estimated on the test sample. We report and analyze the average and standard deviations of the errors of each method obtained in this way on each dataset. In classification problems these numbers refer to error rates in percent, whereas in regression problems they refer to mean square-errors, multiplied by the factor given in the last column of Table 7 (these factors are inversely proportional to the order of magnitude of the errors). These results are reported graphically on Fig. 1. Each individual graph shows, for a particular problem, in left to right order the accuracies of the five methods, with the pruned CART trees (PST) on the far left and the Extra-Trees with default settings (ET^d) on the far right. We recall that the Random Subspace and Random Forests results correspond to the best value of K and are therefore denoted by RS* and RF*.

All the numerical values are collected in Table 8 of Appendix D, with results obtained by other variants, the kNN method, and least squares linear regression.

We have also performed statistical tests to compare the different algorithms. For this purpose, we used a corrected paired two-sided t-test with a confidence level of 95% (see Appendix C). Table 2 reports on the "Win-Draw-Loss" statuses of all pairs of methods.

2.2.4. Discussion of results

Although this is difficult to assess from Fig. 1, let us first notice that for each problem the standard deviations of the errors of the last three methods (RS*, RF*, ET^d) are very close to each other and are significantly smaller than those of PST (50% on average). For Tree Bagging, on the other hand, these standard deviations are on classification problems close to those of PST, while on regression problems they are close to the other three methods.

Concerning average accuracies, Fig. 1 highlights that on a very large majority of problems Extra-Trees are as accurate or more accurate than the other ensemble methods. Among these other methods, Tree Bagging is less convincing on classification problems, while on regression problems it appears to be equivalent with the Random Forests. On regression problems, the Random Subspace method is occasionally significantly worse than the other ensemble methods (on the two Hwang problems, and a little less on Pumadyn-32nm). Only on Vehicle and Housing, Extra-Trees are visibly (slightly) less accurate than some other ensemble methods and, overall, they work very reliably. Finally, we note that all ensemble

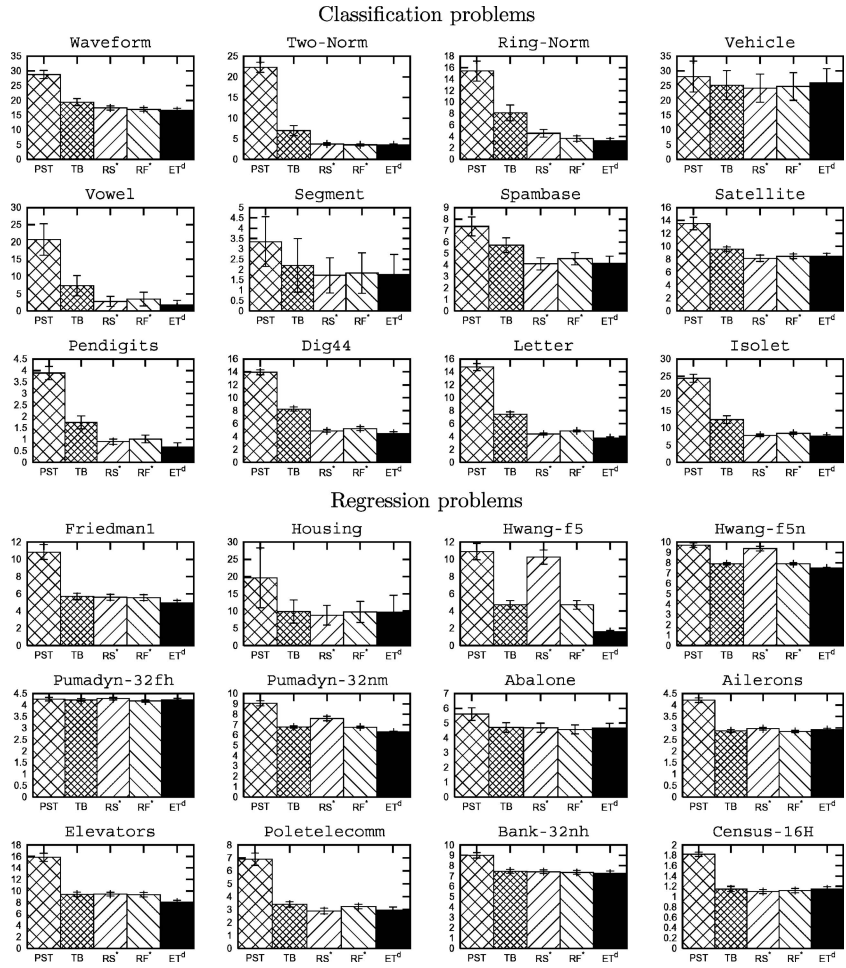

Fig. 1 Comparison (average error and standard deviation) on 12 classification and 12 regression problems.

methods are better (generally very much) than the single pruned CART trees. Indeed, this latter method is competitive only on Pumadyn-32fh. Notice that the results are sometimes slightly worse and sometimes slightly better for unpruned single CART trees (see Table 8, Appendix D).

Considering the significance tests of Table 2, we note that Extra-Trees never lose on classification problems, whereas Tree Bagging loses 7 times with respect to the other ensemble methods, Random Subspace loses 2 times with respect to Extra-Trees, and Random Forests loses 5 times with respect to Extra-Trees and 4 times with respect to Random Subspace. The advantage of Extra-Trees is also valid on regression problems, although here it occasionally loses (1 time with respect to Random Subspace and 2 times with respect to Random Forests).

To assess the effect of the default choice of the parameter K on these conclusions, we did run a side experiment where the value of K was adjusted for each run of the Extra-Trees method by using a 10-fold cross-validation technique internal to the learning sample. The detailed numerical results are given in Table 8 (Appendix D) in the column denoted ET^{cv};

significance tests show that 22 times out of 24 the ET^{cv} variant performs the same as ET^{d}, and two times it wins (on the Segment and Isolet datasets, where slightly better results are obtained for K values higher than the default setting). In terms of Win/Draw/Loss reports with respect to the other methods, the ET^{cv} variant also appears slightly better than ET^{d}. Finally, the comparison of the ET^{cv} version with an ET^{*} variant (see Table 8) shows that there is no significant difference (24 draws on 24 datasets) between these two variants.

These results confirm that the conclusions in terms of accuracy would have been affected only marginally in favor of the Extra-Trees algorithm if we had used the version which adjusts K by cross-validation instead of the default settings. Given the very small gain in accuracy with respect to the default setting and the very high cost in terms of computational burden, we do not advocate the use of the cross-validation procedure together with the Extra-Trees algorithm, except in very special circumstances where one can *a priori* foresee that the default value would be suboptimal. These issues are further discussed in Section 3, together with the analysis of the other two parameters of the method.

2.3. Computational requirements

We compare Extra-Trees with CART, Tree Bagging and Random Forests. In this comparison, we have used unpruned CART trees.[3] We also use the same default settings to run Random Forests as for Extra-Trees ($K = \sqrt{n}$ or $K = n$), so as to put them in similar conditions from the computational point of view. Notice that we dropped the Random Subspace method for this comparison, since its computational requirements are systematically larger than those of Random Forests under identical conditions.[4]

Tables 3 and 4 provide respectively tree complexities (number of leaves of a tree or of an ensemble of trees) and CPU times (in msec)[5] required by the tree growing phase, averaged over the 10 or 50 runs for each dataset. We report in the left part of these tables results for classification problems, and in the right part those for regression problems. Notice that, because in regression the default value of K is equal to n, the Random Forests method degenerates into Tree Bagging; so their results are merged in one column.

Regarding complexity, Table 3 shows that Extra-Trees are between 1.5 and 3 times larger in terms of number of leaves than Random Forests. The average ratio is 2.69 over the classification problems and 1.67 in regression problems. However, in terms of average tree depth, this increase in complexity is much smaller due to the exponential growth of the number of leaves with tree depth. Thus, Extra-Trees are on the average no more than two levels deeper than those produced by Tree Bagging or Random Forests. For example in our trials, the most complex trees were obtained on the Census-16H dataset, with an average depth of 11 for Tree Bagging and 12 for Extra-Trees. Thus, from a practical point of view, the increase in complexity is detrimental only in terms of memory requirements.

Regarding computing times, Table 4 shows that Extra-Trees training is systematically faster than that of Random Forests and Tree Bagging. In classification problems, the average

[3] Pruning by ten-fold cross-validation only slightly affects accuracy (see Table 8), but leads to learning times about ten-times higher than for unpruned trees.

[4] This is due to the fact that the only difference between these methods is that Random forests use bootstrap replicas, which leads to smaller trees (about 30%) and faster tree growing and testing times. Note also that bootstrap sampling could be combined with our Extra-Trees and lead to similar improvements. However, we have found that it deteriorates accuracy, often significantly (see Appendix D).

[5] The system is implemented in C under Linux and runs on a Pentium 4 2.4GHz with 1GB of main memory. In our experiments, all data and models are stored in main memory.

Table 3 Model complexities (total number of leaves, ensembles of $M = 100$ trees)

Classification problems					Regression problems			
Dataset	ST	TB	RF^d	ET^d	Dataset	ST	TB/RF^d	ET^d
Waveform	38	2661	3554	10782	Friedman1	118	7412	11756
Two-Norm	26	1860	2519	7885	Housing	180	11590	18831
Ring-Norm	23	1666	2237	7823	Hwang-f5	773	48720	78438
Vehicle	123	8880	11699	29812	Hwang-f5n	829	51683	79524
Vowel	127	10505	13832	33769	Pumadyn-32fh	755	48043	78503
Segment	62	5083	8071	24051	Pumadyn-32nm	754	47794	78022
Spambase	202	14843	21174	51155	Abalone	1189	76533	129016
Satellite	371	26571	34443	83659	Ailerons	1786	116242	205939
Pendigits	248	19783	29244	76698	Elevators	1946	124346	208356
Dig44	823	59999	81110	239124	Poletelecomm	434	29488	57342
Letter	1385	104652	144593	278984	Bank-32nh	1313	83385	139866
Isolet	171	12593	21072	48959	Census-16H	2944	187543	320074

ratio over the twelve datasets is 0.36 in favor of Extra-Trees with respect to Random Forests, and they are on the average 10 times faster than Tree Bagging. In regression problems, the average ratio is 0.81 (with respect to both methods).

Notice that our implementations of Tree Bagging and Random Forests pre-sort the learning sample before growing all trees to avoid having to re-sort it each time a node is split. On our problems, this pre-sorting reduced the average computing times of these methods by a factor two. Our implementation of Extra-Trees, on the other hand, does not use pre-sorting, which is a further advantage in the case of very large problems, where it may not be possible to keep in memory a sorted copy of the learning sample for each candidate attribute. Actually, strictly speaking, since pre-sorting requires on the order of $nN\log N$ operations, it makes the computational complexity of Random Forests depend linearly on the number of attributes, even when $K = \sqrt{n}$. Hence, for very large numbers of attributes the computational advantage of Extra-Trees is even higher. This is observed on the largest dataset (Isolet), where they are more than ten times faster than Random Forests.

Table 4 Computing times (msec) of training (ensembles of $M = 100$ trees)

Classification problems					Regression problems			
Dataset	ST	TB	RF^d	ET^d	Dataset	ST	TB/RF^d	ET^d
Waveform	68	4022	1106	277	Friedman1	7	372	284
Two-Norm	66	3680	830	196	Housing	12	685	601
Ring-Norm	101	4977	1219	251	Hwang-f5	16	917	742
Vehicle	80	5126	1500	685	Hwang-f5n	15	948	748
Vowel	236	14445	4904	694	Pumadyn-32fh	251	13734	9046
Segment	291	18793	5099	1053	Pumadyn-32nm	221	11850	9318
Spambase	822	55604	9887	8484	Abalone	73	4237	3961
Satellite	687	45096	11035	5021	Ailerons	495	29677	26572
Pendigits	516	34449	12080	5183	Elevators	289	15958	13289
Dig44	4111	259776	67286	9494	Poletelecomm	497	28342	26576
Letter	665	44222	17041	14923	Bank-32nh	613	34402	20178
Isolet	37706	2201246	126480	11469	Census-16H	597	35207	27900

3. On the effect of parameters

This section analyzes and discusses the effect of the three parameters K, $n_{\min}$ and M on the Extra-Trees.

3.1. Attribute selection strength K

The parameter K denotes the number of random splits screened at each node to develop Extra-Trees. It may be chosen in the interval $[1, \ldots, n]$, where n is the number of attributes. For a given problem, the smaller K is, the stronger the randomization of the trees and the weaker the dependence of their structure on the output values of the learning sample. In the extreme, for $K = 1$, the splits (attributes and cut-points) are chosen in a totally independent way of the output variable (we therefore use the term *totally randomized trees* to denote this variant). On the other extreme, when $K = n$, the attribute choice is not explicitly randomized anymore, and the randomization effect acts only through the choice of cut-points.

In order to see how this parameter influences accuracy, and to support our default settings, we have made a systematic experience for all our datasets, by varying the parameter over its range. Figure 2 shows the evolution of the error of Extra-Trees with respect to the value of K, top on classification problems, bottom on regression problems. For classification problems, the default value of $K = \sqrt{n}$ is shown as a vertical line on the graphs; for regression the default corresponds to the highest possible value of K (i.e. $K = n$).

Let us first discuss the results for classification problems, given in the upper part of Fig. 2. We see that there are three types of trends: monotonically decreasing (Vehicle, Satellite, Pendigits), monotonically increasing (only Two-Norm), decreasing followed by increasing (the other 8 problems). For the first two categories, the default setting $k = \sqrt{n}$, is obviously suboptimal.

Actually, the analysis of the Two-Norm problem leads to better understanding of the method. This problem is characterized by strong symmetry over the attributes, and its Bayes optimal decision surface is linear and invariant with respect to permutations of the attributes. This invariance is also obtained in the Extra-Trees (approximately for small values of M, and exactly for very large values of M) provided that we use $K = 1$. As soon as $K > 1$, the method starts fitting the attribute choices to the learning sample, which increases variance (without affecting bias) and hence error rates. We can generalize this finding by saying that problem symmetries should be reflected by the splitting procedure because this allows to reduce variance without increasing bias.

We conjecture that a high percentage of irrelevant variables yields the opposite behavior, namely a monotonic decrease of the error rate with increasing K. Clearly, in such situations using a higher value of K leads to a better chance of filtering out the irrelevant variables, which then leads to a significant reduction of bias over-compensating the increase of variance. In the more generic intermediate situations, where attributes are of variable importance, we obtain the non-monotonic behavior. In this case the default setting of K does generally a good job. To check this hypothesis, we carried out an experiment introducing irrelevant variables on the Two-Norm and Ring-Norm problems. Figure 3 compares on these problems the original trajectory of the error with K with the same curve obtained when we add as many irrelevant attributes as original attributes.[6] We see that on the Two-Norm problem the loss of symmetry

[6] The irrelevant attribute values were drawn from $N(0,1)$ distributions.

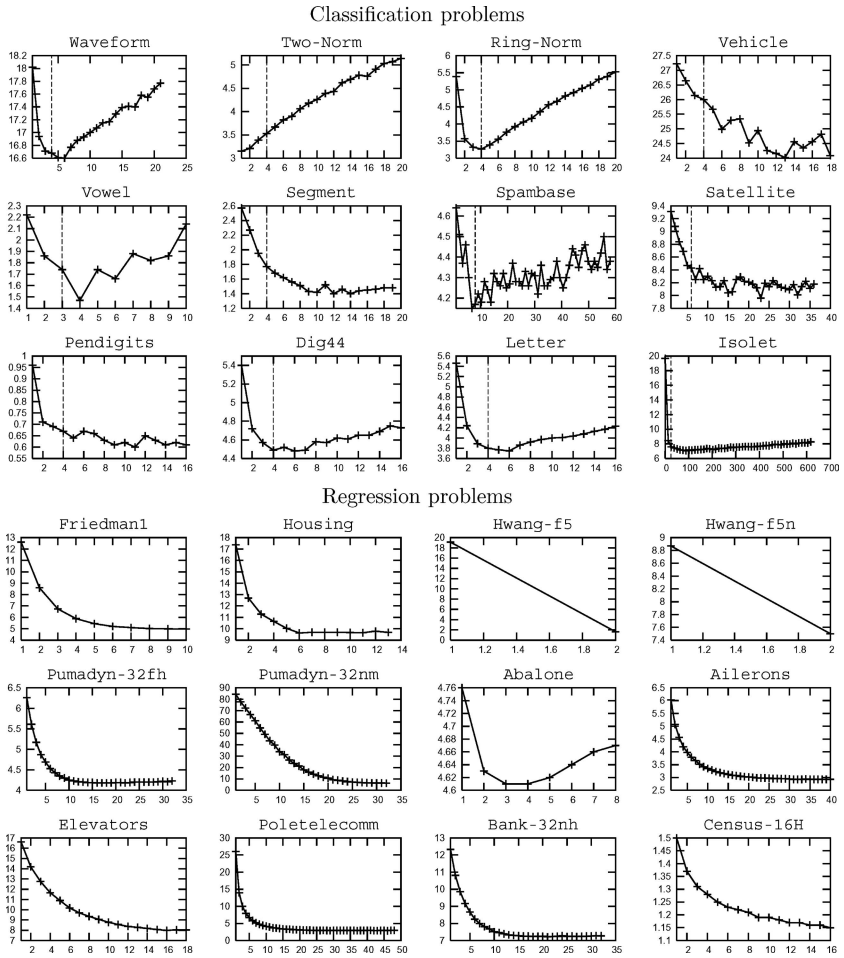

Fig. 2 Evolution of the error of Extra-Trees with K, on 12 classification and 12 regression problems.

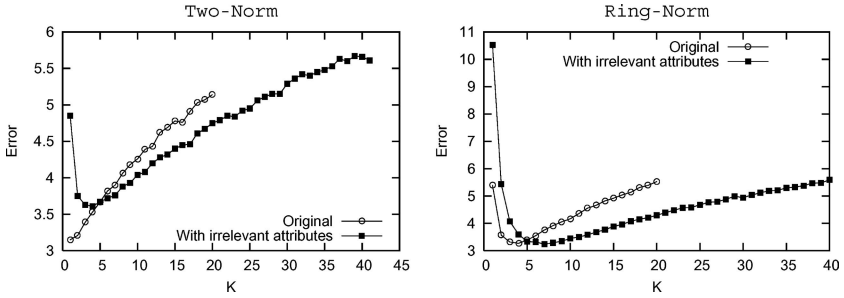

Fig. 3 Effect of irrelevant attributes on the evolution of the error of Extra-Trees with K.

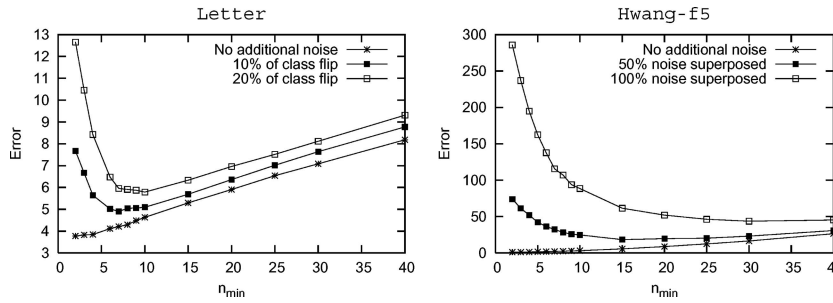

Fig. 4 Evolution of the error of ET^d with n_{min} for different levels of output noise.

indeed leads to the loss of monotonicity. On the other hand, the optimal value of K increases from 4 to 7 on the Ring-Norm problem, while the default value ($\sqrt{n}$, rounded) increases from 4 to 6.

Considering the results shown for the regression problems in Fig. 2, we find that with the exception of Abalone, the behavior is monotonically decreasing, which justifies the default setting of $K = n$ on these problems. Abalone is a rather un-typical regression problem, since its output is discrete (integer valued). Note that if we had treated this problem as a classification problem, we would have used the default value of $K = \lceil\sqrt{8}\rceil = 3$, which is also the optimal value found when it is treated as a regression problem (see Fig. 2).

3.2. Smoothing strength $n_{\min}$

The second parameter of the Extra-Trees method is the number $n_{\min}$ of samples required for splitting a node. Larger values of $n_{\min}$ lead to smaller trees, higher bias and smaller variance, and its optimal value hence depends in principle on the level of output noise in the dataset. To assess the robustness of its default values, we have tried out different ones on all 24 datasets. This did not yield a significant improvement on any classification problem[7], while on two regression problems, $n_{\min} = 2$ was better than the default of 5: on Hwang-f5, which is deterministic by construction, the error decreases from 1.62 to 1.13, and on Housing it decreases from 9.68 to 9.09. Conversely, on two other regression problems stronger smoothing was better: using $n_{\min} = 10$ decreased the error on Hwang-f5n (the noisy version of this problem) from 7.50 to 7.15 and on Abalone from 4.69 to 4.56. Therefore, although possibly slightly suboptimal, the default values of $n_{\min} = 2$ (classification) and $n_{\min} = 5$ (regression) appear to be robust choices in a broad range of typical conditions.

Clearly, in very noisy problems, ensembles of fully grown trees will over-fit the data. In order to highlight this in the context of Extra-Trees, we report an experiment, where we have added noise on the output values in the training set, and used ET^d with increasing values of $n_{\min}$. The results obtained for Hwang-f5 (an originally noise-free regression problem) and for Letter (classification) are shown in Fig. 4, where in addition to the curve corresponding to the original dataset, we show for each problem two curves where additional

[7] According to the corrected t-test with a significance level of 95%.

noise was superposed on the output variable: for the classification problem, we have randomly flipped the class of learning samples (respectively in 10% and 20% of the cases) and, for the regression problem, we have superposed Gaussian noise (with a standard deviation respectively of 50% and 100% of that of the output variable), while errors are computed using the original test samples (i.e. without additional noise). These results clearly illustrate that the noisier the output, the higher the optimal value of $n_{\min}$, and they also show the robustness of the method to high noise conditions, provided that the value of $n_{\min}$ is increased.

Similar experiments with the other datasets confirm this analysis, and show that the default values of $n_{\min}$ are often robust with respect to a moderate increase in the output noise. On the other hand, we note that from a purely theoretical point of view, one could ensure consistency (i.e., convergence to the Bayes optimal model with increasing values of sample size N) of the Extra-Trees by letting $n_{\min}$ grow slowly with N (e.g., $n_{\min} \propto \sqrt{N}$). In this respect, Extra-Trees are not different from other tree-based methods and the proofs of consistency given in (Breiman et al., 1984) still hold.

3.3. Averaging strength M

The parameter M denotes the number of trees in the ensemble. It is well known that for randomization methods the behavior of prediction error is a monotonically decreasing function of M (see, e.g., Breiman, 2001). So in principle, the higher the value of M, the better from the accuracy point of view. The choice of an appropriate value will thus essentially depend on the resulting compromise between computational requirements and accuracy. Different randomization methods may have different convergence profiles on different problems, depending also on the sample size and other parameter settings (e.g., K and $n_{\min}$). So not much more can be said in general about this value.

For the sake of illustration, the top of Fig. 5 shows on a classification and on a regression problem the evolution of the error when increasing the number of trees. To better illustrate the speed of convergence, the bottom of the same figure shows on the same problems the evolution with M of the ratio between the error with M trees and the error with 100 trees. In classification, we compare Bagging, Extra-Trees (with default setting and with $K = 1$), and Random Forests (also with $K = \sqrt{n}$). In regression, we compare Bagging, and Extra-Trees with $K = n$ and $K = 1$. The straight lines in the top of Fig. 5 represent the error of one single pruned tree.

These curves are typical of what we observed on most problems. In classification, the convergence of Extra-Trees with the default setting is slower than the convergence of Tree Bagging and of Random Forests (to a lesser extent). However, Extra-Trees quickly outperforms these two methods. In regression, RF^d degenerates into Tree Bagging and the speed of convergence of this latter method is now indistinguishable from that of Extra-Trees with default setting.

As concerns totally randomized trees (ET^1), we note that their speed of convergence in regression is comparable to that of the other ensemble methods. In classification, however, they converge more slowly. On the considered problem, they outperform Tree Bagging only after $M = 40$ models. This suggests that K has indeed an influence on the number of trees that need to be aggregated to ensure convergence, but we found that this influence is notable only for very small values of K and only for classification problems.

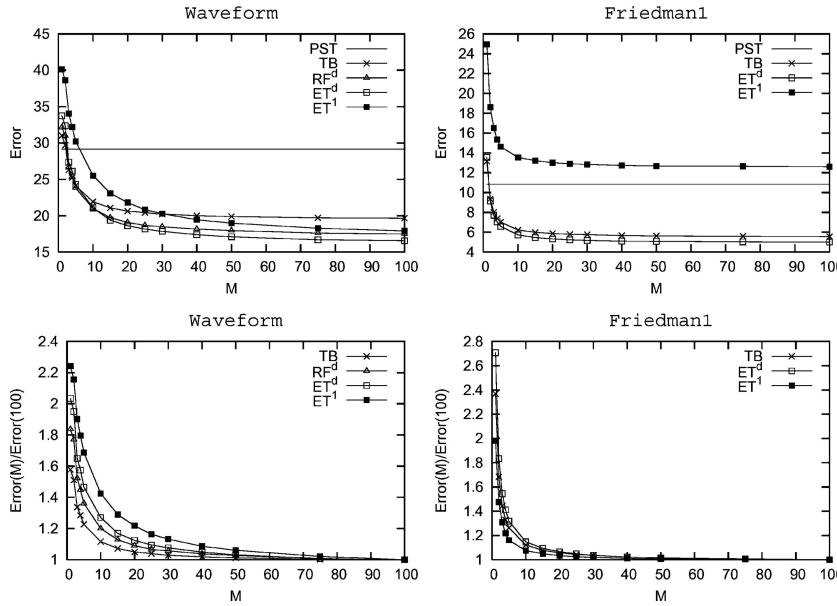

Fig. 5 Top, evolution of the error with the number M of trees in the ensemble, bottom, evolution of the ratio between the error for a given M and the error for 100 trees for the same method.

4. Bias/variance analysis

In this section, we analyse the bias/variance characteristics of the Extra-Trees algorithm and compare them with those of the other tree-based methods. In order to make the paper self-contained, Appendix E provides a theoretical analysis of the bias/variance decomposition of randomization methods. Before turning to the simulation results, we summarize the main findings of this analysis as follows:

- randomization increases bias and variance of individual trees, but may decrease their variance with respect to the learning sample;
- the part of the variance due to randomization can be canceled out by averaging over a sufficiently large ensemble of trees;
- overall, the bias/variance tradeoff is different in regression than in classification problems; in particular, classification problems can tolerate high levels of bias of class probability estimates without yielding high classification error rates.

4.1. Experiments and protocols

We have chosen three classification (Waveform, Two-Norm, Ring-Norm) and three regression problems (Friedman1, Pumadyn-32nm, Census-16H) corresponding to the larger datasets. To estimate bias and variance (see, e.g., Bauer and Kohavi, 1999), each dataset is split into two samples: a pool sample (PS) and a test sample (TS). Then, 100 models are learned from 100 learning samples (LS) randomly drawn (with replacement) from the pool. Finally, bias, variance, and average errors are estimated on the test sample by means of these 100 models. The sample sizes used in our experiments are given in Table 5.

Table 5 Sample sizes for bias/variance analysis

Dataset	PS size	TS size	LS size
Waveform	4000	1000	300
Two-Norm	8000	2000	300
Ring-Norm	8000	2000	300
Friedman1	8000	2000	300
Pumadyn-32nm	6192	2000	300
Census-16H	15000	7784	2000

In regression, since the Bayes model is unknown on some problems in our experiments, we call bias the sum of the true bias and the residual error. This sum represents the true error of the average model. In the case of classification, we will provide bias/variance decompositions of the average square-error of conditional class probability estimates, together with average error-rates.

In order to ease the comparison with the other methods, we discuss in this section results for the ET* and ET^1 variants of Extra-Trees, rather than ET^d. Notice, however, that on regression problems ET^d provides identical results with ET*, while on classification problems it has a slightly higher variance and lower bias.

4.2. Comparison of the different randomization methods

Figure 6 shows (on regression problems in the upper part and on classification problems in the lower part) the bias/variance decomposition for different learning algorithms. Error, bias, and variance are expressed in percent on classification problems and scaled according to the factor of Table 7 on regression problems. We observe the following:

- The variance of single trees (ST) is very high with respect to their bias, the latter including the residual error. All ensemble methods increase the bias and decrease the variance with respect to single trees, as predicted by the analysis of Appendix E.
- Among ensemble methods, Extra-Trees (ET* and ET^1) reduce most strongly variance, but they do also increase more strongly the bias than the other ensemble methods. Overall, ET*

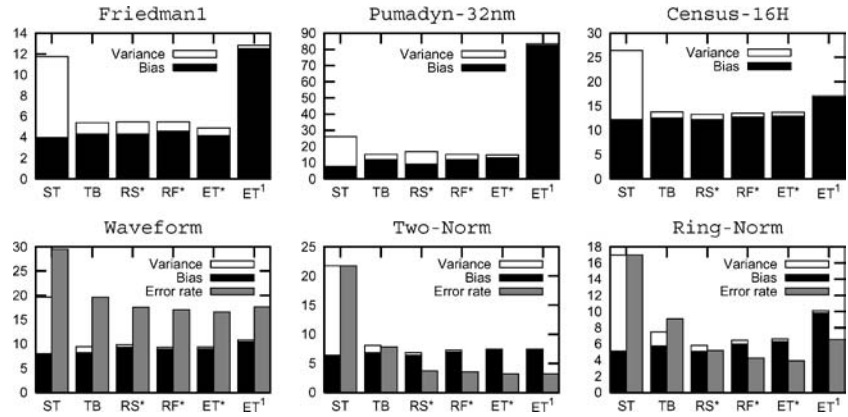

Fig. 6 Bias/variance decomposition for different algorithms.

provides the best tradeoff between bias and variance. Its variance reduction with respect to ST is very impressive in all cases (95% on the average), while its bias increase is moderate (21% on the average). We also observe that RS* has a lower bias than RF*, which is due to the fact that it uses the full learning sample rather than a bootstrap replica to build trees.

- Comparing the two ET variants, we observe on most problems that ET^1 has smaller variance and a higher bias than ET*, and that the increase of bias of ET^1 is more strongly marked in regression. A notable exception is the Two-Norm problem where ET^1 is identical with ET*, which confirms the analysis of Section 3.1 concerning the impact of the symmetric nature of this problem on bias and variance.
- The strongest increase of the bias of ET^1 is observed on Friedman1 and Pumadyn-32nm. Actually, these two problems have a large proportion of irrelevant attributes (5 out of 10 on Friedman, 30 out of 32 on Pumadyn-32nm). The effect of removing them is analyzed in Section 4.4.
- On classification problems, ET* provides the best results in terms of error rate, although in terms of "bias+variance" of probability estimates it is sometimes inferior to RS* and/or RF*. Note that this remains true for the ET^d variant.
- On all classification problems, ET^1 provides smaller error rates than Tree Bagging, even though its "bias+variance" in terms of class probability estimates is sometimes significantly higher (on Ring-Norm, and to a lesser extent, on Waveform). This is due to the fact that in classification problems higher bias of probability estimates does not necessarily imply higher error rates, as discussed in Appendix E.

4.3. Bias/variance tradeoff with K

Figure 7 shows the evolution with K of bias and variance of the mean square-error of Extra-Trees, left on the Friedman1 and right on the Waveform dataset. For the Waveform dataset, we actually refer to the mean square-error of probability estimates. We observe that bias monotonically decreases and variance increases when K increases, and that from this point of view there is no qualitative difference between classification and regression problems.

In classification (Waveform), we see from the curve labeled "error rate" that the optimum error rate corresponds to a much higher degree of randomization (i.e., a much smaller value of K) than the optimum of the square-error of probability estimates. Notice that this effect is also observed on other datasets, because the misclassification error is (intrinsically) more tolerant to an increase of bias than the regression error. One can therefore take better advantage in classification from the decrease of variance resulting from a stronger tree randomization.

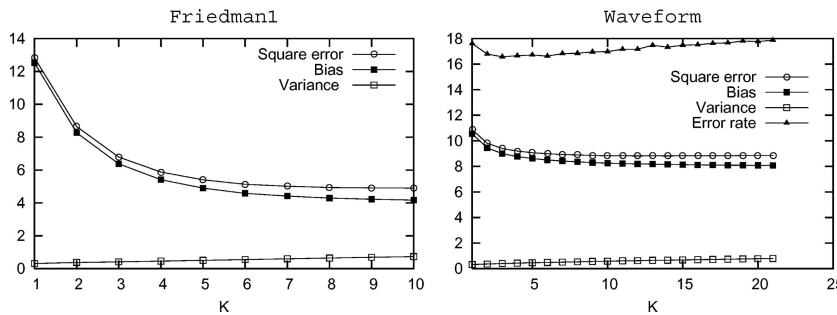

Fig. 7 Evolution of bias and variance with K, left on Friedman1, right on Waveform.

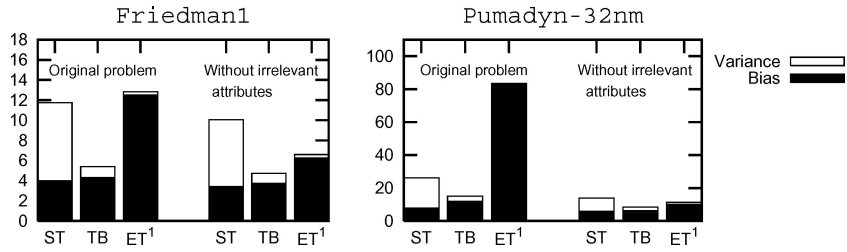

Fig. 8 Bias/variance on regression problems with irrelevant attributes removed.

This explains why Extra-Trees and totally randomized trees (and also Random Forests) are usually significantly better than Tree Bagging on classification problems while they are not on regression problems (see detailed results in Table 8, Appendix D).

4.4. Totally randomized trees and irrelevant variables

While the decrease of variance brought by totally randomized trees is not surprising, we would like to explain why bias increases so much on some problems. Since, with this method, attributes are selected randomly without looking at their relationship with the output, if there is a (locally or globally) irrelevant attribute, it will nevertheless be selected with the same probability as a relevant one. By averaging, the effect of irrelevant attributes on the ensemble prediction will be canceled, but this will have the side effect of increasing bias. Actually, splitting on an irrelevant attribute is tantamount to randomly splitting the learning sample. Thus, irrelevant attributes act virtually as a reduction of the sample size, which results in increased bias and variance of individual trees. While the variance increase is compensated by tree averaging, the increase in bias is not.[8]

This analysis is supported by the fact that the two regression problems that present the highest increase in bias of ET^1 (Friedman1 and Pumadyn-32nm) contain both a high proportion of irrelevant attributes. Indeed, by construction (Friedman, 1991), Friedman1 contains five totally irrelevant attributes among the 10 and, in Pumadyn-32nm, 2 attributes (out of 32) contain over 95% of the information about the output.[9] Fig. 8 shows, for single unpruned trees, Tree Bagging, and totally randomized trees, the effect of removing the irrelevant attributes on bias and variance. We see that variance is mostly unaffected for ensemble methods, while it is reduced for single trees. For all three methods bias is reduced, but this occurs in a much stronger way for the totally randomized trees. For example, by removing the irrelevant attributes from the Pumadyn-32nm problem, the bias of the ensemble of 100 totally randomized trees has dropped from 82.44 to 9.83, while variance has increased only slightly from 0.94 to 1.40. On the Friedman1 problem, bias drops from 12.52 to 6.25 and variance increases from 0.31 to 0.34.

[8] For example, on the Pumadyn-32nm problem, the bias and variance of a single totally randomized tree are respectively of 82.67 and 40.11, while for an ensemble of 100 such trees variance drops to 0.94 and bias remains unchanged up to the first decimal. Similarly, on the Friedman1 dataset, the bias of a single totally randomized tree is of 13.02 and variance is of 12.38, while for an ensemble of 100 such trees variance drops to 0.31, and bias slightly decreases to 12.52.

[9] This was found by computing attribute importance from a set of trees obtained by Tree Bagging. Attribute importance was computed according to the algorithm described in (Hastie et al., 2001).

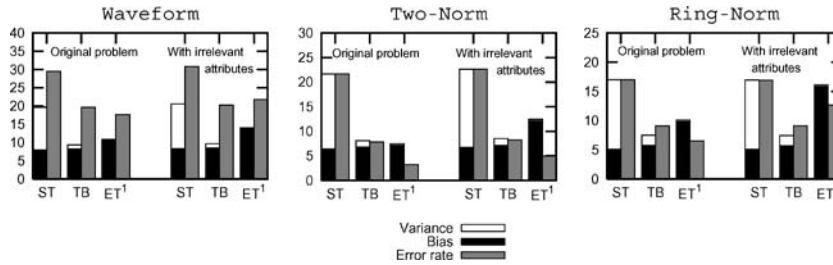

Fig. 9 Bias/variance on classification problems with irrelevant attributes added.

On the three classification problems of Fig. 6 all attributes are important for determining the output and, hence, the increase of bias of totally randomized trees is less important. To further illustrate this behavior, we have reproduced the experiment of Section 3.1 where we have added as many irrelevant attributes[10] as original attributes on the three classification problems. The results are shown on Fig. 9. We observe that with the irrelevant attributes, the increase of bias of ET^1 with respect to single trees is much more important. Note also that on Two-Norm the totally randomized trees nevertheless remain competitive with Tree Bagging in terms of error-rate, which is less strongly affected on this problem than the square-error of class probabilities.

5. Model characterizations

To provide further insight, we analyze in this section the Extra-Trees algorithm in terms of properties of the models it induces, from two different points of view. First, we provide a geometrical characterization of the models output by the Extra-Trees algorithm. Then, we show that these models can be considered as kernel-based models.

5.1. Geometrical point of view

To illustrate the geometrical properties of fully developed Extra-Trees ($n_{\min} = 2$), we show on Fig. 10 their models obtained for a simple one-dimensional noise-free regression problem, together with those of Tree Bagging under identical conditions. The models are obtained with a specific sample depicted on the figure, which was obtained by drawing 20 points uniformly in the unit interval. The figure shows also the true function behind this sample. In the left part, it gives models for ensembles of $M = 100$ trees and in the right part for $M = 1000$. These graphics illustrate the fact that Extra-Trees produce models which appear to be piecewise linear in the limit of $M \to \infty$, and are much smoother than those of Tree Bagging.

One can show that in general, for an n-dimensional input space and $n_{\min} \geq 2$, infinite ensembles of Extra-Trees produce a continuous piecewise multi-linear approximation of the sample. To make this explicit, let us consider a learning sample of size N

$$ls_N = \{(x^i, y^i) : i = 1, \ldots, N\},$$

[10] Irrelevant attributes values are drawn from $N(0, 1)$ distributions.

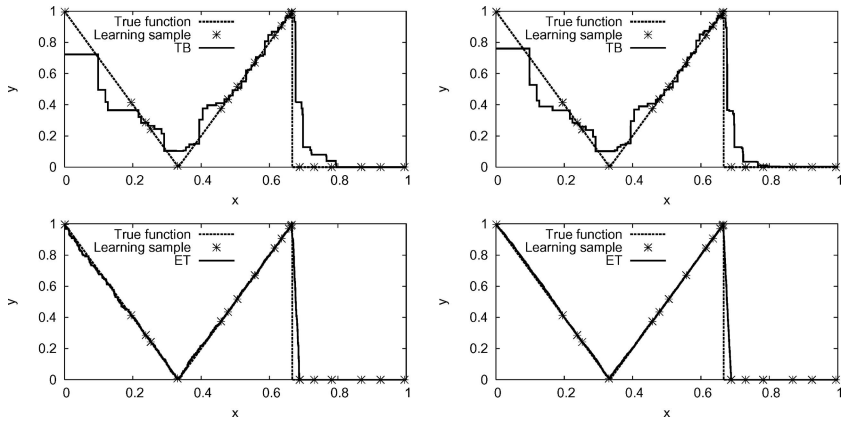

Fig. 10 Tree Bagging, and fully developed Extra-Trees ($n_{\min} = 2$) on a one-dimensional piecewise linear problem ($N = 20$). Left with $M = 100$ trees, right with $M = 1000$ trees.

where each $x^i = (x_1^i, \ldots, x_n^i)$ is an attribute vector of dimension n and y^i is the corresponding output value, and let us denote by

$$(x_j^{(1)}, \ldots, x_j^{(N)})$$

the sample values of the j^{th} attribute taken by increasing order, and for notational simplicity let us define

$$x_j^{(0)} = -\infty \quad \text{and} \quad x_j^{(N+1)} = +\infty, \forall j = 1, \ldots, n,$$

and denote, $\forall (i_1, \ldots, i_n) \in \{0, \ldots, N\}^n$, by $I_{(i_1,\ldots,i_n)}(x)$ the characteristic function of the hyper-interval

$$[x_1^{(i_1)}, x_1^{(i_1+1)}[\times \cdots \times [x_n^{(i_n)}, x_n^{(i_n+1)}[.$$

With these notations, one can show[11] that an infinite ensemble of Extra-Trees provides an approximation in the form of

$$\hat{y}(x) = \sum_{i_1=0}^{N} \cdots \sum_{i_n=0}^{N} I_{(i_1,\ldots,i_n)}(x) \sum_{X \subset \{x_1,\ldots,x_n\}} \lambda_{(i_1,\ldots,i_n)}^{X} \prod_{x_j \in X} x_j, \tag{1}$$

where the real-valued parameters $\lambda_{(i_1,\ldots,i_n)}^{X}$ depend on sample inputs x^i and outputs y^i as well as on the parameters $n_{\min}$ and K of the method.

In the particular case of fully developed trees ($n_{\min} = 2$) they are such that

$$\hat{y}(x^i) = y^i, \forall (x^i, y^i) \in ls, \tag{2}$$

[11] The proof is a straightforward adaptation of the proofs given in (Zhao, 2000). See Appendix F.

and if the input space is one-dimensional ($n = 1$, and $x = (x_1)$), the model degenerates into a piecewise linear model

$$\hat{y}(x) = \sum_{i_1=0}^{N} I_{(i_1)}(x) \sum_{X \subset \{x_1\}} \lambda_{(i_1)}^{X} \prod_{x_j \in X} x_j = \sum_{i=0}^{N} I_{(i)}(x_1)(\lambda_i^{\emptyset} + \lambda_i^{\{x_1\}} x_1), \tag{3}$$

where $I_{(i)}(x_1)$ denotes the characteristic function of the interval $[x_1^{(i)}, x_1^{(i+1)}[$. The values of $\lambda_i^{\emptyset}$ and $\lambda_i^{\{x_1\}}$ may be derived directly from the N eqs. (2), N continuity constraints, and constraints imposing a constant model over the intervals $[x_1^{(i)}, x_1^{(i+1)}]$, $\forall i \in \{0, N\}$.

Extremely and totally randomized tree ensembles hence provide an interpolation of any output variable which, for finite M is piecewise constant (and, hence non-smooth), and for $M \to \infty$ becomes piecewise multi-linear and continuous. This is in contrast with other tree-based ensemble methods whose models remain piecewise constant even for $M \to \infty$. From a bias/variance viewpoint, the continuous nature of the model translates into smaller variance and bias in the regions where the target function is smooth and hence leads to more accurate models in such regions.

5.2. Kernel point of view

For the sake of simplicity, we now particularize our discussion to regression trees.[12] Let us denote by $ls_N = \{(x^i, y^i) : i = 1, \ldots, N\}$ a learning sample of size N, by t a tree structure derived from it comprising l_t leaves, by $l_{t,i}(x)$ the characteristic function of the i^{th} leaf of t, by $n_{t,i}$ the number of learning samples such that $l_{t,i}(x) = 1$, and by

$$l_t(x) = \left(\frac{l_{t,1}(x)}{\sqrt{n_{t,1}}}, \ldots, \frac{l_{t,l_t}(x)}{\sqrt{n_{t,l_t}}} \right)^T \tag{4}$$

the vector of (normalized) characteristic functions of t. Then the model defined by the tree t can be computed by the equation

$$\hat{y}_t(x) = \sum_{i=1}^{N} y^i l_t^T(x^i) l_t(x), \tag{5}$$

which shows that tree-based models are kernel-based models, where the kernel

$$K_t(x, x') \stackrel{\triangle}{=} l_t^T(x) l_t(x') \tag{6}$$

is the scalar product over a feature space defined by the (normalized) characteristic functions of the leaf nodes of the tree. Furthermore, the kernel and model defined by an ensemble $\mathcal{T} = \{t_i : i = 1, \ldots, M\}$ of M trees are straightforwardly obtained by

$$K_{\mathcal{T}}(x, x') = M^{-1} \sum_{i=1}^{M} K_{t_i}(x, x'), \tag{7}$$

[12] The extension to the estimation of conditional class-probabilities is obtained by considering it as the regression of a vector of class-indicator variables.

and

$$\hat{y}_{\mathcal{T}}(x) = M^{-1} \sum_{i=1}^{M} \hat{y}_{t_i}(x) = \sum_{i=1}^{N} y^i K_{\mathcal{T}}(x^i, x). \tag{8}$$

Alternatively, we can construct a feature vector of $\sum_{i=1}^{M} l_{t_i}$ components by

$$l_{\mathcal{T}}(x) = \left(\frac{l_{t_1}^T(x)}{\sqrt{M}}, \ldots, \frac{l_{t_M}^T(x)}{\sqrt{M}} \right)^T \tag{9}$$

and compute the ensemble kernel by

$$K_{\mathcal{T}}(x, x') = l_{\mathcal{T}}^T(x) l_{\mathcal{T}}(x'). \tag{10}$$

The feature vector $l_t(x)$ induced by a single tree structure is sparse in the sense that, $\forall x \in X$ only one feature is different from zero. For an ensemble of M trees, the feature vector is also sparse, in the sense that $\forall x \in X$ only M features are non-zero.

In the particular case of ensembles of fully developed trees, we have $l_{t_i} = N, \forall i = 1, \ldots, M$ and $n_{t_i,j} = 1, \forall i = 1, \ldots, M, \forall j = 1 \ldots, N$, and the ensemble feature space is of dimensionality MN. In this case the tree ensemble *interpolates* the learning sample, namely

$$\hat{y}_{\mathcal{T}}(x^i) = y^i, \forall (x^i, y^i) \in ls. \tag{11}$$

In general, with $n_{\min} \geq 2$, it provides a bounded *approximation* of the sample, namely

$$\min_i y^i \leq \hat{y}_{\mathcal{T}}(x) \leq \max_i y^i, \forall x \in X. \tag{12}$$

With totally randomized trees, the kernel $K_{\mathcal{T}}(x, x')$ is independent of the output values y^i of the learning sample. Hence, we can view an ensemble of totally randomized tree structures as an ensemble of (randomized) metrics which allow one to interpolate or approximate output values from learning samples using their attribute values. It is clear that these metrics are invariant with respect to linear transformations of the coordinate axes (re-scaling).

For finite M, the ensemble kernel $K_{\mathcal{T}}(x, x')$ is obviously piecewise constant. However, the kernel corresponding to an infinite ensemble of Extra-Trees ($M \to \infty$) is continuous and piecewise multi-linear with respect to both x and x' (see Appendix F). Breiman (2000b) shows, under the assumptions of uniform prior distribution $P(x)$, infinite sample size ($N \to \infty$), and infinite ensembles of totally randomized trees of fixed number l of leaves, that the kernel is approximately given by[13]

$$K_{\mathcal{T}}(x, x') \approx \exp\{-\lambda |x - x'|_1\}, \tag{13}$$

[13] This exponential form, in apparent contradiction with the multi-linear piecewise form found earlier, is due to the infinite sample size hypothesis which causes the number of hyper-intervals of (multi-)linearity to grow to infinity and thus results in a non piecewise and nonlinear function.

where $|x - x'|_1$ denotes the city-block distance. In this expression λ denotes the "sharpness" of the kernel and is defined by

$$\lambda = \frac{\log l}{n}, \tag{14}$$

where n is the dimension of the input space. Notice that for balanced trees built from a finite sample of size N, the number of leaves l is on the order of

$$\frac{N}{n_{\min}},$$

which suggests that a high dimension of the input space has a very strong smoothing effect, actually much stronger than a high value of $n_{\min}$.

Along a slightly different line of reasoning, Lin and Jeon (2002) show that if the number of samples $n_{t_{i,j}}$ in each terminal node of all trees is kept constant and equal to k as $N \to \infty$, then the number of samples which could influence the prediction of an ensemble of trees at some point[14] is on the order of

$$k(\log N)^{(n-1)}. \tag{15}$$

This implies, in good agreement with the results of (Breiman, 2000b), that the averaging effect in ensembles of regression trees could grow exponentially faster with the number of dimensions of the input space than with the number of samples k kept in the leaves. These results may explain why for most of our high-dimensional (and sometimes very noisy) regression problems, we could not improve accuracies when increasing the value of $n_{\min}$.

The effect of increasing the value of K (attribute selection strength) in the Extra-Trees method is to make the kernel become sharper along those input directions along which the output variable varies more strongly and less sharp in the other directions, and hence to reduce (locally or globally) the dimension of the space over which the kernel actually operates. This may explain why in regression problems it is often better to use higher values of K in favor of an implicit reduction of the input-space dimensionality, which can reduce the over-smoothing effect of the curse of dimensionality in the presence of irrelevant variables. In classification problems this effect is less marked because classification problems are much more tolerant with respect to over-smoothed (biased) class probability estimates.

6. Related work on randomized trees

Besides the Random Subspace method (Ho, 1998) and Random Forests (Breiman, 2001), which have been used in this paper for comparison purposes, several other randomized tree growing algorithms have been proposed in the context of ensemble methods. Although some of them could be applied to regression problems as well, all these methods have only been evaluated on classification problems.

For example, Ali (1995) perturbs the standard tree induction algorithm by replacing the choice of the best test by the choice of a test at random among the best ones. If S^* is the score

[14] They call them the potential k nearest neighbors of this point.

of the optimal test, a test is randomly selected, with a probability proportional to its score, among the tests which have a score greater than $(1-\beta)S^*$, where β is some constant between 0 and 1. Dietterich (2000) proposes a similar approach that consists in randomly selecting a test among the k best splits. Choosing β equal to 1 in Ali's method or letting k increase to infinity in Dietterich's method, reduces these two algorithms to totally randomized trees.[15] However, these authors did not study these extreme parameter values, nor the impact of these parameters on accuracy. In (Ali, 1995) the value of β was set to 0.75 in all experiments and in (Dietterich, 2000) the sole value of $k = 20$ was considered. The results shown in the present paper suggest that these two methods could reach similar accuracy as Extra-Trees by tuning appropriately their parameters. However, even in the case of strong randomization ($\beta \to 1$ or $k \to \infty$), these methods would require the score computation of all possible splits at each test node, thus losing the computational advantages of Extra-Trees.

Zheng and Webb (1998)'s Stochastic Attribute Selection Committees (SASC) are close to the Random Subspace method. At each node, the best test is searched among only a subset of the candidate attributes where each attribute has a probability P of being selected in the subset. P plays a similar role as K in the Random Subspace method and its value has been fixed to 0.33. A study of bias and variance of this method and other boosting based algorithms in (Webb and Zheng, 2004) shows that SASC works mainly by reducing the variance of the standard decision tree method while bias remains mainly unaffected. Furthermore, the combination of SASC with Wagging (a variant of Bagging) shows improvement with respect to SASC alone, again because of a more important reduction of variance. This is consistent with our experiments showing that randomization should be quite high in classification.

Kamath et al. (2002) randomize the tree induction method by discretizing continuous attributes through histograms at each tree node, evaluating the score only of bin boundaries, and then selecting a split point in some interval around the best bin boundary. In their experiments, the number of bins was fixed at the square root of the local learning sample size. However, when the number of bins in the histograms is equal to the local learning sample size, this algorithm builds standard trees and when there is only one bin per attribute, this algorithm is equivalent to Extra-Trees with $K = n$. The reduction of computing times was also advanced as an argument in favor of this method with respect to other ensemble methods like Tree Bagging that require to evaluate every possible split.

Cutler and Guohua (2001) propose an algorithm for classification problems that builds almost totally randomized trees. To split each non-terminal node in their variant, two examples are first randomly selected from different classes in the local learning sample. Then, an attribute is selected at random and the cut-point is randomly and uniformly drawn between the values of this attribute for the two random examples. Like our Extra-Trees, the trees are fully grown from the original learning sample so as to perfectly classify the learning sample. For this reason, they are called perfect random trees, or PERT in short. This method was compared to Bagging and Random Forests on several classification problems. It often gives competitive results with these two methods and also comes with an important reduction in computing times. Although PERT splits are not totally independent of the output, we believe that this method is very close to our totally randomized trees on classification problems, both in terms of accuracy and computing times. Note however that this randomization scheme does not readily apply to regression problems.

In (Geurts, 2002) and (Geurts, 2003), we have proposed a randomized tree algorithm that, at each test node, generates random tests (without replacement of the attribute) until

[15] Not strictly for Ali's method since a test is randomly drawn with a probability proportional to its score.

finding one that realizes a score greater than some threshold. This score threshold plays a very similar role as the parameter K of the algorithm proposed here. When the score threshold is equal to 0, this method builds totally randomized trees and when it is equal to the maximal score value, it is equivalent to Extra-Trees with $K = n$. A comparison between these two methods does not show any significant differences in terms of accuracy. At first, we thought that filtering bad tests with a score threshold would improve computing times and also facilitate the choice of a default value for the parameter. However, we have found no evidence over many experiments that this was indeed the case. We therefore prefer the variant proposed here, for simplicity reasons, and because its computational complexity is more predictable.

Besides the tree world, our analysis suggests a possible way to apply the idea of extreme randomization to other algorithms. To design a good randomization method, we should be able to build randomized models that are good on the learning sample to keep the bias low and then aggregate several of these models to get a low variance. This idea is especially interesting with trees because building a perfect tree on the learning sample is trivial and very fast. However, there may be other kinds of models where the same idea could be applied. Herbrich et al. (2001) have proposed an algorithm that may be interpreted as the application of this idea to support vector machines; the algorithm consists in generating and aggregating several perfect linear models in the extended input space. These models are obtained by using a simple perceptron learning rule and by randomizing the order in which the learning cases are presented to the algorithm. Like our Extra-Trees, the main advantage of this algorithm is its computational efficiency with respect to the classical support vector machine approach. The parallel between these two approaches is certainly worth being explored.

7. Conclusion

In this paper, we have proposed an extremely randomized tree growing algorithm that combines the attribute randomization of Random Subspace with a totally random selection of the cut-point. In addition to the number M of trees generated (a common parameter of all ensemble methods), this method depends on one main parameter, called K, that controls the strength of the attribute randomization, and on a secondary parameter, called $n_{\min}$, that controls the degree of smoothing. The analysis of the algorithm and the determination of the optimal value of K on several test problem variants have shown that the value is in principle dependent on problem specifics, in particular the proportion of irrelevant attributes. Nevertheless, our empirical validations have shown that default values for K are near-optimal on 22 out of 24 diverse datasets, and only slightly suboptimal on two of them. They result also in competitive results with respect to state-of-the-art randomization methods, in terms of accuracy and computational efficiency.

This empirical validation was completed by a bias/variance analysis of the Extra-Trees algorithm and a geometrical and a kernel characterization of its models. The bias/variance analysis has shown that Extra-Trees work by decreasing variance while at the same time increasing bias. Once the randomization level is properly adjusted, the variance almost vanishes while bias only slightly increases with respect to standard trees. When the randomization is increased above the optimal level, variance decreases slightly while bias increases often significantly. We have also shown that this bias increase was due to the fact that over-randomization prevents the algorithm from detecting attributes of low relevance and reduces the effective sample size when there are many such attributes. Furthermore, we have highlighted the different nature of the bias/variance tradeoff in classification and

regression problems, explaining why classification problems can take advantage of stronger randomization.

The geometrical analysis has shown that Extra-Trees asymptotically produce continuous, piecewise multi-linear functions. The resulting models are thus smoother than the piecewise constant ones obtained with other ensemble methods which optimize the cut-points. This potentially leads to better accuracy in regions of the input space where the target function is indeed smooth. We have also shown that tree-based ensemble models can be written as kernel-based models. In the case of totally randomized trees, the kernel is independent of the output values and thus it defines a universal scale-invariant metric defined on the input space that can be used to approximate any target function. When K increases, the kernel is automatically adapted to the output values, its sharpness increasing in those directions along which the target function varies more strongly. Theoretical results from the literature furthermore suggest that the spreading of the kernel increases rapidly with the dimension of the input space resulting in a strong smoothing effect in the high dimensional case.

Actually, this paper has come up with two new learning algorithms that have complementary features: Extra-Trees with the default setting and totally randomized trees. Both methods are non parametric. The first one provides near optimal accuracy and good computational complexity, especially on classification problems. The second one, although not as accurate as the first one, is trivial to implement, even faster, and the models it induces are independent of the output variable, making of this algorithm a very interesting alternative to the kNN algorithm. Both methods have already proven useful in a number of applications. In particular, problems of very high dimensionality, like image classification problems (Marée et al., 2004), mass-spectrometry datasets (Geurts et al., 2005b), or time-series classification problems (Geurts and Wehenkel, 2005), make the Extra-Trees a first choice method due to its attractive computational performances. Also, the fact that totally randomized trees have a tree structure independent of the output variable has been exploited in the context of reinforcement learning where it ensures the convergence of the reinforcement learning algorithm and leads to a very efficient implementation (Ernst et al., 2005).

There remain several future work directions. First of all, while we have focused here on numerical attributes, it is also very desirable to handle other types of attributes. For categorical attributes, we propose to generate random (binary) splits by selecting a random subset of their possible values. This approach has already been successfully used in order to treat biological (genetic) sequence classification problems (Geurts et al., 2005a). However, more systematic empirical studies have to be carried out, and also the analytical characterization of the models obtained with such attributes still have to be explored and will certainly result in interesting, and probably very different properties, especially in the context of categorical attributes only.

Since bias is the dominant component of the error of Extra-Trees, future improvements of randomization methods should focus on this part of the error. There exist several techniques to reduce the bias. One simple technique in the context of trees could be to extend tree tests to take into account several attributes. This idea was already applied with some success in the context of Random Forests by Breiman (2001). On the other hand, since Boosting is a method known to reduce bias, it could possibly be combined with our Extra-Trees so as to reduce their bias (see Webb, 2000; Webb and Zheng, 2004, for a combination of Boosting and different randomization methods). Stochastic Discrimination (Kleinberg, 1990) is another theoretical framework to transform a weak classification algorithm (i.e. one with high bias) into a stronger one. A deeper analysis of this framework in our context of extreme randomization could also help in this direction.

Finally, along the line of the model characterization carried out in this paper, further work towards a theoretical analysis of randomization methods is still needed and could lead to a better understanding of these methods. For example, while we have shown that uniform sampling of cut-points leads to multi-linear models, it would be interesting to study the impact of different randomization schemes (e.g. cut-points drawn from a Gaussian distribution) on the analytical form of the approximation. Also, a theoretical analysis of the exact effect of values of K greater than 1 on the approximation and its corresponding kernel is still missing. Lastly, we have shown that it is possible to exploit problem symmetries to justify particular randomization schemes. Along this idea, a deeper theoretical analysis could also help to take advantage of a priori knowledge (e.g. invariances or symmetries) in designing ad hoc methods for specific classes of problems.

Appendix

A. Pseudo-code of the complete Extra-Trees algorithm and score measures

The complete Extra-Trees algorithm is described in Table 6, together with the node splitting procedures for both numerical and categorical attributes.

Our score measure in classification is a particular normalization of the information gain. For a sample S and a split s, this measure is given by:

$$\text{Score}_C(s, S) = \frac{2I_c^s(S)}{H_s(S) + H_c(S)},$$

where $H_c(S)$ is the (log) entropy of the classification in S, $H_s(S)$ is the split entropy (also called split information by Quinlan (1986)), and $I_c^s(S)$ is the mutual information of the split outcome and the classification. With respect to Quinlan's gain ratio, this normalization, proposed by Wehenkel and Pavella (1991), has the advantage of being symmetric in c and s and it also further mitigates the "end-cut" preference of this latter measure (Wehenkel, 1998). For a discussion of entropy based score measures and normalization, the interested reader can refer to (Wehenkel, 1996).

In regression, we use the relative variance reduction. If S_l and S_r denote the two subsets of cases from S corresponding to the two outcomes of a split s, then the score is defined as follows:

$$\text{Score}_R(s, S) = \frac{\text{var}\{y|S\} - \frac{|S_l|}{|S|}\text{var}\{y|S_l\} - \frac{|S_r|}{|S|}\text{var}\{y|S_r\}}{\text{var}\{y|S\}},$$

where $\text{var}\{y \mid S\}$ is the variance of the output y in the sample S.

B. Description of datasets

The experiments are conducted on 12 classification and 12 regression problems which are summarized in Table 7. Most datasets are available in the UCI Machine Learning Repository (Blake and Merz, 1998). Friedman1, Two-Norm, and Ring-Norm are three artificial problems introduced respectively in (Friedman, 1991) and (Breiman, 1996a). Pumadyn, Hwang,

Table 6 Pseudo-code of the Extra-Trees algorithm

Build_an_extra_tree_ensemble(S).
Input: a training set S.
Output: a tree ensemble $\mathcal{T} = \{t_1,\ldots,t_M\}$.
– For i=1 to M
  • Generate a tree: t_i= **Build_an_extra_tree**(S);
– Return $\mathcal{T}$.

Build_an_extra_tree(S).
Input: a training set S.
Output: a tree t.
– Return a leaf labeled by class frequencies (or average output, in regression) in S if
  (i) $|S| < n_{min}$, or
  (ii) all candidate attributes are constant in S, or
  (iii) the output variable is constant in S
– Otherwise:
  1. Select randomly K attributes, $\{a_1,...,a_K\}$, without replacement, among all (non constant in S) candidate attributes;
  2. Generate K splits $\{s_1,\ldots,s_K\}$, where s_i = **Pick_a_random_split**(S, a_i), $\forall i = 1,\ldots,K$;
  3. Select a split s_* such that Score(s_*, S) = $\max_{i=1,\ldots,K}$ Score(s_i, S);
  4. Split S into subsets S_l and S_r according to the test s_*;
  5. Build t_l = **Build_an_extra_tree**(S_l) and t_r = **Build_an_extra_tree**(S_r) from these subsets;
  6. Create a node with the split s_*, attach t_l and t_r as left and right subtrees of this node and return the resulting tree t.

Pick_a_random_split(S,a)
Input: a training set S and an attribute a.
Output: a split.
– If the attribute a is numerical:
  • Compute the maximal and minimal value of a in S, denoted respectively by $a^S_{\min}$ and $a^S_{\max}$;
  • Draw a cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
  • Return the split $[a < a_c]$.
– If the attribute a is categorical (denote by $\mathcal{A}$ its set of possible values):
  • Compute $\mathcal{A}_S$ the subset of $\mathcal{A}$ of values of a that appear in S;
  • Randomly draw a proper non empty subset $\mathcal{A}_1$ of $\mathcal{A}_S$ and a subset $\mathcal{A}_2$ of $\mathcal{A}\backslash\mathcal{A}_S$;
  • Return the split $[a \in \mathcal{A}_1 \cup \mathcal{A}_2]$.

Bank, and Census come from the DELVE repository of data[16] and Ailerons, Elevators, and Poletelecomm are taken from (Torgo, 1999).[17] Notice that the last column of Table 7 provides the normalizing factors that we have used to display mean square-errors for the regression problems. Notice also that the datasets marked with a star correspond to problems where either the learning sample or the test sample contain a small number of observations (≤ 300). For these, our protocol consists of running 50 experiments corresponding to 50 random LS/TS splits, instead of only 10.

To give some insight into the range of problems considered, we give a few indications about the performance of various learning methods and/or intrinsic properties of the datasets. To this end, we discuss separately regression and classification problems.

[16] http://www.cs.utoronto.ca/~delve.

[17] http://www.liacc.up.pt/~ltorgo.

Table 7 Datasets summaries

Classification problems					Regression problems				
Dataset	Atts	Class	LS size	TS size	Dataset	Atts	LS size	TS size	Err×
Waveform⋆	21	3	300	4700	Friedman1⋆	10	300	9700	1
Two-Norm⋆	20	2	300	9700	Housing⋆	13	455	51	1
Ring-Norm⋆	20	2	300	9700	Hwang-f5	2	2000	11600	10^3
Vehicle⋆	18	4	761	85	Hwang-f5n	2	2000	11600	10^2
Vowel⋆	10	11	891	99	Pumadyn-32fh	32	2000	6291	10^4
Segment⋆	19	7	2079	231	Pumadyn-32nm	32	2000	6291	10^5
Spambase	57	2	3221	1380	Abalone	8	3133	1044	1
Satellite	36	6	4435	2000	Ailerons	40	5000	8750	10^8
Pendigits	16	10	7494	3498	Elevators	18	5000	11559	10^6
Dig44	16	10	9000	9000	Poletelecomm	48	5000	10000	10^{-1}
Letter	16	26	10000	10000	Bank-32nh	32	3692	4500	10^3
Isolet	617	26	6238	1559	Census-16H	16	7784	15000	10^{-9}

Among the 12 classification problems, the first 3 are synthetic ones and the other 9 are real ones. On two problems, the *k*NN method yields very good results (Two-Norm and Vowel, see Table 8), and on two other problems its results are close to those of the tree-based ensemble methods (Pendigits, and Dig44). On four datasets (Ring-Norm, Spambase, Segment, and Isolet) the *k*NN gives very disappointing results. On Waveform, Vehicle, Satellite, and Letter, it is slightly suboptimal with respect to Extra-Trees. On Ring-Norm and Vehicle the best performance is obtained with quadratic discriminants. On Two-Norm the Bayes optimal classifier is linear. On the other problems the best performance published in the literature is obtained with non-linear multi-layer perceptron type of methods.

Among the 12 regression problems, 9 are synthetic ones and three are real datasets (Housing, Abalone, Census-16H). On one problem (Bank-32nh), linear regression slightly outperforms the best tree-based methods, while on three others (Pumadyn-32fh, Abalone, Ailerons), it performs equally well. On the other hand, it is largely suboptimal on most other problems. As for *k*NN, it works well on Hwang-f5n, and to a lesser extent on Census-16H. Hwang, Pumadyn, Bank, and Census are three families of problems specifically chosen in the DELVE project to evaluate regression methods. Hwang-f5 and Hwang-f5n are respectively a noise free and a noisy variant of the same two-dimensional non linear problem. By construction, Pumadyn-32fh is a fairly linear problem with high noise while Pumadyn-32nm is a highly non linear problem with medium noise. Bank-32nh is a highly noisy problem and Census-16H is defined in DELVE as a highly difficult problem.

C. Corrected *t*-test

In each run of random sub-sampling, the data set is divided into a learning sample of a given size n_L and a test sample of size n_T. The learning algorithm is run on the learning sample and its error is estimated on the test sample. The process is repeated N_s times and the resulting errors are averaged. Let e_A^i and e_B^i denote the errors of two methods A and B in the ith run ($1 \leq i \leq N_s$) of random sub-sampling and let d^i denote the difference $e_A^i - e_B^i$. The statistic corresponding to the t-test is:

$$t = \frac{\mu_d}{\sqrt{\frac{\sigma_d^2}{N_s}}}, \tag{16}$$

where

$$\mu_d = \frac{\sum_{i=1}^{N_s} d^i}{N_s} \quad \text{and} \quad \sigma_d^2 = \frac{\sum_{i=1}^{N_s} (d^i - \mu_d)^2}{N_s - 1} \tag{17}$$

Under the null hypothesis stating that A and B are equivalent and assuming that the differences d_i are independent, t follows a student distribution with $N_s - 1$ degrees of freedom. Under re-sampling, the hypothesis of independence is clearly violated as the different learning and test samples partially overlap. Nadeau and Bengio (2003) have proposed a correction to this t-test that takes into account this overlapping. With the same notations, the corrected statistic is the following:

$$t_{corr} = \frac{\mu_d}{\sqrt{(\frac{1}{N_s} + \frac{n_T}{n_L})\sigma_d^2}}, \tag{18}$$

This statistic is also assumed to follow a student distribution with $N_s - 1$ degrees of freedom. Experiments in (Nadeau and Bengio, 2003) show that this test improves the type I error with respect to the standard t-test. Note that Nadeau and Bengio (2003) suggest to use a value of n_L 5 to 10 times larger than n_T.

D. Detailed results of empirical study

Table 8 gives average errors of the different learning algorithms compared in this paper as they were obtained with the protocol described in Section 2.2.3. In classification problems these numbers refer to error rates in percent, whereas in regression problems they refer to (normalized) mean square-errors.

To make the comparison easier, we provide in the first column of Table 8 the standard deviations of the error rates over the different LS/TS splits, as obtained for each problem with the Extra-Trees algorithm with default settings (column named ET^d). Notice that these standard deviations are also indicative of those of all the other tree-based methods except ST and TB. The standard deviations of the errors of the single trees are at least twice as large on all problems, and this is also the case for Tree Bagging on classification problems. On regression problems, however, the standard deviation of the errors of Tree Bagging is close to that of the other ensemble methods. The five following columns give the results obtained by the reference methods (single unpruned and pruned CART trees, Tree Bagging, Random Subspace and Random Forests with optimal value of K). The subsequent columns provide error rates for the Extra-Trees with different ways of adjusting the parameter K: first $K = *$ (K is adjusted optimally on the average test set error rates), then $K = cv$ (it is adjusted for each LS/TS split by 10-fold cross-validation internal to the learning sample), then $K = d$ (K fixed a priori according to the default setting, i.e., $d = \sqrt{n}$ in classification problems, and $d = n$ in regression problems), then a version denoted ET_B^d, where $K = d$ is combined with bootstrap resampling of the training set, and finally the totally randomized version ($K = 1$). For the classification problems we also provide the results for the value of $K = n$. For regression problems this value is not explicitly given since it is equal to the default value of $K = d$. Instead, we provide the mean square error provided by a linear regression method for these latter problems (column LR). Finally, the last column provides the results obtained with the k-nearest neighbor method on these datasets; as suggested by the notation

Table 8 Error rates of all methods

Classification problems													
Dataset	σ_{ET^d}	ST	PST	TB	RS*	RF*	ET*	ET^{cv}	ET^d	ET^d_B	ET^1	ET^n	*k*NN*
Waveform	0.70	29.17	28.76	19.41	17.45	16.97	16.60	16.84	16.61	16.59	18.02	17.77	18.35
Two-Norm	0.27	21.53	22.31	6.97	3.76	3.54	3.15	3.52	3.53	3.47	3.15	5.14	2.78
Ring-Norm	0.38	16.31	15.40	8.12	4.57	3.64	3.27	3.55	3.27	3.54	5.39	5.53	36.22
Vehicle	4.71	26.80	28.05	25.15	24.16	24.73	24.02	24.47	26.00	25.72	27.22	24.09	28.09
Vowel	1.33	20.63	20.71	7.35	2.73	3.43	1.47	1.47	1.74	2.28	2.22	2.14	1.21
Segment	0.96	3.24	3.35	2.20	1.72	1.83	1.40	1.34	1.77	2.00	2.57	1.48	3.54
Spambase	0.60	8.33	7.36	5.73	4.10	4.54	4.15	4.41	4.17	4.36	4.64	4.38	9.48
Satellite	0.49	14.63	13.52	9.56	8.14	8.45	7.96	8.12	8.43	8.97	9.31	8.18	9.37
Pendigits	0.19	3.91	3.89	1.74	0.90	1.02	0.60	0.63	0.67	0.77	0.96	0.61	0.63
Dig44	0.24	15.03	13.96	8.24	4.87	5.23	4.48	4.47	4.49	4.82	5.40	4.73	4.42
Letter	0.15	14.84	14.75	7.44	4.39	4.87	3.75	3.77	3.80	4.24	5.46	4.23	6.33
Isolet	0.43	26.45	24.43	12.40	7.85	8.43	7.08	7.19	7.61	8.52	19.68	8.28	15.45

Regression problems													
Dataset	σ_{ET^d}	ST	PST	TB	RS*	RF*	ET*	ET^{cv}	ET^d	ET^d_B	ET^1	LR	*k*NN*
Friedman1	0.26	11.73	10.84	5.69	5.59	5.55	4.97	5.00	4.97	5.50	12.60	7.38	9.44
Housing	4.91	19.90	19.63	9.86	8.79	9.72	9.63	9.81	9.68	10.89	17.36	24.38	18.91
Hwang-f5	0.14	10.91	10.91	4.72	10.28	4.72	1.62	1.58	1.62	2.89	19.11	812.61	5.22
Hwang-f5n	0.07	12.32	9.71	7.91	9.37	7.91	7.50	7.51	7.50	7.25	8.87	87.88	7.56
Pumadyn-32fh	0.06	8.27	4.26	4.22	4.28	4.18	4.18	4.19	4.23	4.20	6.26	4.19	5.38
Pumadyn-32nm	0.08	13.41	9.06	6.77	7.59	6.75	6.28	6.29	6.28	6.72	84.33	72.75	76.89
Abalone	0.31	8.55	5.61	4.71	4.69	4.57	4.61	4.62	4.67	4.59	4.76	4.93	4.90
Ailerons	0.06	5.53	4.21	2.88	2.97	2.85	2.93	2.93	2.93	2.94	6.03	3.07	4.69
Elevators	0.36	19.49	15.86	9.42	9.45	9.36	7.99	8.01	8.03	8.86	16.62	8.43	16.53
Poletelecomm	0.25	6.90	6.90	3.41	2.90	3.24	2.87	2.91	2.95	3.81	26.06	92.98	11.84
Bank-32nh	0.22	14.44	9.00	7.46	7.41	7.35	7.21	7.24	7.27	7.29	12.33	6.96	10.22
Census-16H	0.04	2.20	1.82	1.15	1.10	1.12	1.15	1.16	1.15	1.20	1.50	2.09	1.46

kNN*, the column reports the average error corresponding to the best test set error value of k for each problem. The kNN method uses an euclidian metric over attributes rescaled by the inverse of their standard deviation.

The comparison of the accuracies of ET^d with those of ET^d_B confirm that bootstrap resampling tends to reduce the accuracy of the Extra-Trees method. Indeed, on two of the classification problems (Letter and Satellite) and 4 regression problems (Housing, Hwang-f5, Elevators, Poletelecomm) ET^d_B is significantly less accurate than ET^d, while it is significantly better only on two regression problems (Hwang-f5n and Abalone).

The reader may wish to compare the results of kNN* with those of the totally randomized trees (column ET^1) which is also a unsupervised kernel-based method. Also the relative performance of this method with respect to the results obtained with ET^* gives some indications of the diversity of the problems in terms of the diversity of relative performance of the kNN method. For example, it is interesting to observe that on Pumadyn-32fh both ET^1 and kNN work quite well, while on Pumadyn-32nm they provide very disappointing results. Actually, since in this problem 30 attributes among the 32 contain almost no information, this explains why kNN and ET^1, which treat all attributes equally, are so sub-optimal even with respect to single unpruned trees.

Finally, the comparison of the accuracies of ET^d with those obtained by a linear regression (LR) gives an indication of how competitive ET^d is in regression. We observe that the linear least squares regression method is much less accurate on most problems than ET^d. Only on Bank-32nh it outperforms the other methods, while it provides comparable results on Pumadyn-32fh, Abalone, and Ailerons.

E. Bias/variance formulation of ensembles of randomized trees

In this appendix we provide the derivations leading to the bias/variance decomposition of ensembles of randomized trees. We start with the case of regression and then consider classification problems.

E.1. Bias/variance decomposition of the square-error for regression problems

Before considering randomized methods, we first recall the standard derivation of the bias/variance decomposition of the square-error for deterministic learning algorithms.

E.1.1. Deterministic learning algorithms. A deterministic learning algorithm can be viewed as a function mapping a learning sample into a model. We denote the prediction of such a model at a point x of the input space by $f(x; LS)$.[18] Assuming a random sampling scheme generating learning samples of fixed size N, this prediction is a random variable, and so is its average error over the input space. We study the average value of this quantity defined by:

$$\begin{aligned} E_{LS}\{Err(f(\cdot; LS))\} &\stackrel{\triangle}{=} E_{LS}\{E_{X,Y}\{L(Y, f(X; LS))\}\} \\ &= E_X\{E_{LS}\{E_{Y|X}\{L(Y, f(X; LS))\}\}\} \\ &= E_X\{E_{LS}\{Err(f(X; LS))\}\}, \end{aligned} \tag{19}$$

[18] In this appendix we use upper case letters to denote random variables and lower case letters to refer to their realizations.

where $L(.\ ,\ .)$ is the loss function and where $Err(f(X;LS))$ denotes the expected loss at point X and $Err(f(\cdot;LS))$ its expectation over the input space. If $L(y,\hat{y}) = (y-\hat{y})^2$, the error locally decomposes into (see, e.g., Geman et. al., 1992; Hastie et al., 2001):

$$E_{LS}\{Err(f(x;LS))\} = \sigma_R^2(x) + \text{bias}_R^2(x) + \text{var}_R(x), \tag{20}$$

where

$$\sigma_R^2(x) \stackrel{\triangle}{=} E_{Y|x}\{(Y - f_B(x))^2\}, \tag{21}$$

$$\text{bias}_R^2(x) \stackrel{\triangle}{=} (f_B(x) - \overline{f}(x))^2, \tag{22}$$

$$\text{var}_R(x) \stackrel{\triangle}{=} E_{LS}\{(f(x;LS) - \bar{f}(x))^2\} \tag{23}$$

with

$$f_B(x) \stackrel{\triangle}{=} \arg\min_y E_{Y|x}\{(Y-y)^2\} = E_{Y|x}\{Y\} \tag{24}$$

$$\bar{f}(x) \stackrel{\triangle}{=} E_{LS}\{f(x;LS)\}. \tag{25}$$

The function $f_B(\cdot)$ is called the Bayes (optimal) model. By definition the Bayes model minimizes the average error defined by Eq. (19), and in the case of a squared loss function it is equal to the conditional expectation of Y given $X = x$. The first term of the decomposition (20) is the local error of this model. It is called the residual (squared) error, $\sigma_R^2(x)$. It provides a theoretical lower bound of the error which is independent of the learning algorithm. Thus the (local) sub-optimality of a particular learning algorithm is composed of two (non negative) terms:

- the (squared) bias, $\text{bias}_R^2(x)$, measuring the discrepancy between the Bayes model $f_B(x)$ and the average model, $\bar{f}(x)$.
- the variance, $\text{var}_R(x)$, measuring the variability of the predictions (around the average model) with respect to the learning sample randomness.

These local quantities can be "globalized" by averaging them over the input distribution.

E.1.2. Randomized learning algorithms. We use the following terminology:

- "Original algorithm" denotes any given automatic learning algorithm (we assume, without any limitation, that this algorithm is deterministic). A model returned by this algorithm for some learning sample *ls* will be denoted by $f(\cdot;ls)$.
- "Randomized algorithm" denotes the randomized version of the original algorithm, according to some perturbation technique. We formalize this by introducing an additional random variable ε which summarizes the random perturbation scheme, and denote by ϵ a value of this variable and accordingly by $f_r(\cdot;ls,\epsilon)$ a particular model generated by a call to the randomized algorithm.
- "Averaged algorithm" denotes the algorithm producing models built by aggregating M different models obtained with the randomized algorithm. An averaged model will be denoted by $f_{a^M}(\cdot;ls,\epsilon^M)$. Its predictions are computed as:

$$f_{a^M}(x;ls,\epsilon^M) = \frac{1}{M}\sum_{i=1}^{M} f_r(x;ls,\epsilon_i). \tag{26}$$

Let us analyze the bias and variance terms of the randomized and averaged algorithms and relate them to bias and variance of the original algorithm.

E.1.2.1. *Randomized algorithm.* Because the randomized algorithm depends on a second random variable ε, its average square-error and its bias/variance decomposition becomes:

$$E_{LS,\varepsilon}\{Err(f_r(x;LS,\varepsilon))\} = \sigma_R^2(x) + (f_B(x) - \overline{f}_r(x))^2 + var_{LS,\varepsilon}\{f_r(x;LS,\varepsilon)\}, \tag{27}$$

where:

$$\overline{f}_r(x) \stackrel{\triangle}{=} E_{LS,\varepsilon}\{f_r(x;LS,\varepsilon)\} \tag{28}$$

$$var_{LS,\varepsilon}\{f_r(x;LS,\varepsilon)\} \stackrel{\triangle}{=} E_{LS,\varepsilon}\{(f_r(x;LS,\varepsilon) - \overline{f}_r(x))^2\}. \tag{29}$$

The variance term (29) may be further decomposed into two (positive) terms:

$$var_{LS,\varepsilon}\{f_r(x;LS,\varepsilon)\} = var_{LS}\{E_{\varepsilon|LS}\{f_r(x;LS,\varepsilon)\}\} + E_{LS}\{var_{\varepsilon|LS}\{f_r(x;LS,\varepsilon)\}\}. \tag{30}$$

The first term is the variance with respect to the learning set randomness of the average prediction according to ε. It measures the dependence of the model on the learning sample, independently of ε. The second term is the expectation over all learning sets of the variance of the prediction with respect to ε. It measures the strength of the randomization ε.

E.1.2.2. *Averaged algorithm.* Assuming that the ϵ_i are independently drawn from the same distribution $P(\varepsilon|LS)$, the average model of the averaged algorithm is nothing but the average model of the randomized algorithm:

$$\overline{f}_{a^M}(x) = E_{LS,\varepsilon^M}\{f_{a^M}(x;LS,\varepsilon^M)\} = \frac{1}{M}\sum_{i=1}^{M} E_{LS,\varepsilon_i}\{f_r(x;LS,\varepsilon_i)\} = \overline{f}_r(x). \tag{31}$$

So, its bias is equal to the bias of the randomized algorithm. On the other hand, under the same assumptions it can be shown that its variance satisfies:

$$var_{LS,\varepsilon^M}\{f_{a^M}(x;LS,\varepsilon^M)\} = var_{LS}\{E_{\varepsilon|LS}\{f_r(x;LS,\varepsilon)\}\} + \frac{E_{LS}\{var_{\varepsilon|LS}\{f_r(x;LS,\varepsilon)\}\}}{M}. \tag{32}$$

So, averaging over M values of ε reduces the second part of the variance of the randomized algorithm by a factor M and leaves the remaining parts of the average square-error unchanged. This implies that the larger the number of ensemble terms, the smaller the average square-error of the averaged algorithm.

E.1.2.3. *The averaged vs the original algorithm.* Figure 11 shows the evolution of bias and variance from the original algorithm to the averaged algorithm. Asymptotically, with

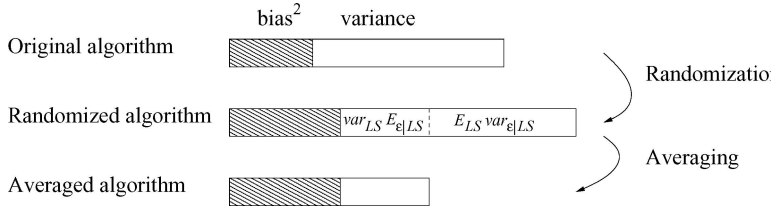

Fig. 11 Expected evolution of bias and variance by randomization and averaging.

respect to M, the whole process of randomization and averaging decreases the variance with respect to our original algorithm if:

$$var_{LS}\{E_{\varepsilon|LS}\{f_r(x; LS, \varepsilon)\}\} < var_{LS}\{f(x; LS)\}, \tag{33}$$

namely, if the randomization cancels part of the variance of the original algorithm.

This condition is easy to satisfy. However, to compare the averaged algorithm with the original algorithm, we need also to take into account their biases. Indeed, in general,

$$\overline{f}_{a^M}(x) = \overline{f}_r(x) \neq \overline{f}(x) \tag{34}$$

and hence bias could possibly be increased by the randomization. Actually, when the original learning algorithm explicitly tries to minimize empirical risk, it is likely that the randomization will disturb the algorithm in this goal. Consequently, the averaged algorithm is likely to have higher bias than the original one. It thus appears that in most randomization methods there is a bias/variance tradeoff controlled by the randomization strength:

- the stronger the randomization, the lesser the dependence of the randomized models on the learning sample and the smaller the variance of the averaged algorithm;
- the stronger the randomization, the lesser the dependence of output predictions on the input attributes and the higher bias of both the randomized and the averaged algorithms.

Note that the increase of variance resulting from the second term of (30) does not influence the tradeoff since this term is the one which is canceled by averaging. Nevertheless, larger values of this term will imply slower convergence with respect to the number M of ensemble terms, and thus lead to higher computational requirements in practice.

E.2. Bias/variance decompositions for classification problems

Several bias/variance decompositions have been proposed in the literature for the average error rate. They all try to mimic the main properties of the decomposition of the average square-error (see for example Geurts, 2002, or James, 2003, for a review of several decompositions). However, none of them is fully satisfactory. The problem in classification is that a decrease of the variability of the predictions at some point may actually increase the error (if the predictions are wrong in average). The consequence when studying randomization methods is that, unlike the square-error, the error rate of the averaged algorithm may be greater than the error rate of the randomized algorithm. So, it is very difficult to study randomization methods by trying to understand directly their effect on the misclassification error (see Bauer and Kohavi, 1999; Webb, 2000; Valentini and Dietterich, 2004, for empirical bias/variance analyses of ensemble methods).

However, many classification algorithms, like decision trees, work by first computing a numerical estimate $f_c(x; LS)$ of $P(Y = c|X = x)$ and then deriving a classification rule by predicting the class maximizing this estimate:

$$f(x; LS) = \arg\max_c f_c(x; LS). \tag{35}$$

We could thus consider classification models as multidimensional regression models and apply the previous analysis to study the effect of randomization methods. Thus, an approach to study classification algorithms is to relate the bias and variance terms of these estimates to the average misclassification error of the resulting classification rule (35). Friedman (1997) has done this connection in the particular case of a two-class problem and assuming that the distribution of $f_c(x; LS)$ with respect to LS is close to Gaussian. He shows that the average misclassification error at some point x may be written as:

$$E_{LS}\{Err(f(x; LS))\} = 1 - P(f_B(x)|x) + \Phi\left(\frac{E_{LS}\{f_{f_B(x)}(x; LS)\} - 0.5}{\text{var}_{LS}\{f_{f_B(x)}(x; LS)\}}\right)(2P(f_B(x)|x) - 1), \tag{36}$$

where $f_B(\cdot)$ is the Bayes optimal classifier and $\Phi(\cdot)$ is the upper tail of the standard normal distribution. According to the sign of the numerator in (36) we have:

- When a majority of models vote in the same way as the Bayes classifier, a decrease of variance will decrease the error.
- Conversely, when a majority of models vote wrongly, a decrease of variance will increase the error.

Another important conclusion that can be drawn from (36) is that, whatever the regression bias on the approximation of $f_c(x; LS)$, the classification error can be driven to its minimum value by reducing solely the variance, under the assumption that a majority of models are right. In other words, perfect classification rules can be induced from very bad (rather biased, but of small variance) probability estimators.

This discussion shows that where the average probability estimates of the randomized algorithm are consistent with the Bayes optimal decision rule, the reduction of variance obtained by the averaged algorithm will lead to a reduction of the average error rate. Conversely, when this is not the case, the averaged algorithm will be worse than the randomized one. Because of the non additive interaction between bias and variance of probability estimates and the average error rate, the best tradeoff in classification will not correspond to the best tradeoff in terms of the average square-error of probability estimates. Actually, since from Eq. (36) a high bias of probability estimates does not prevent a low misclassification error if the variance is low, we may expect that the best tradeoff in terms of misclassification error will correspond to a lower variance and hence a higher randomization level than the best tradeoff in terms of the square-error.

F. Geometrical characterization of the Extra-Trees kernel

In this appendix, we show that the Extra-Trees kernel is a continuous piecewise multi-linear function of its two arguments when the number of trees grows to infinity. The developments follow from an adaptation of the proofs given in (Zhao, 2000).

Let us consider a learning sample of size N

$$ls_N = \{(x^i, y^i) : i = 1, \ldots, N\},$$

where each $x^i = (x^i_1, \ldots, x^i_n)$ is an attribute vector of dimension n and y^i is the corresponding output value, and let us denote by

$$(x^{(1)}_j, \ldots, x^{(N)}_j)$$

the sample values of the j^{th} attribute taken by increasing order. Let us denote by I^l_k, $k \in \{1, \ldots, n\}, l \in \{1, \ldots, N-1\}$ the interval $]x^{(l)}_k, x^{(l+1)}_k]$ as well as the random event corresponding to a split on variable k in this interval at the top node of an Extra-Tree grown from ls_N. Let us further denote by $ls^l_{k,L}$ and $ls^l_{k,R}$ the left and right subsets of ls_N resulting from that split:

$$ls^l_{k,L} = \{(x^i, y^i) \in ls_N | x^i_k \leq x^{(l)}_k\} \text{ and } ls^l_{k,R} = \{(x^i, y^i) \in ls_N | x^i_k \geq x^{(l+1)}_k\}$$

For $M \to \infty$, the kernel defined by (7) (Section 5.2) becomes the average over the distribution of Extra-Trees of 0 when x and x' fall in different leaves in the tree and $1/n_l$ when x and x' fall in the same leaf l of size n_l.

Given this definition, the following recursive equations allow to compute this kernel:

– If $N < n_{min}$

$$K^\infty_{\mathcal{T}}(x, x'; ls_N) = \frac{1}{N}; \tag{37}$$

– Otherwise:

$$K^\infty_{\mathcal{T}}(x, x'; ls_N)$$

$$= \sum_{k=1}^{n} \sum_{l=1}^{N-1} P(I^l_k). \begin{cases} K^\infty_{\mathcal{T}}(x, x'; ls^l_{k,L}) & \text{if} \quad x_k \leq x^{(l)}_k \wedge x'_k \leq x^{(l)}_k \\ K^\infty_{\mathcal{T}}(x, x'; ls^l_{k,R}) & \text{if} \quad x_k > x^{(l+1)}_k \wedge x'_k > x^{(l+1)}_k \\ \frac{x^{(l+1)}_k - x'_k}{x^{(l+1)}_k - x^{(l)}_k} K^\infty_{\mathcal{T}}(x, x'; ls^l_{k,L}) + \frac{x_k - x^{(l)}_k}{x^{(l+1)}_k - x^{(l)}_k} K^\infty_{\mathcal{T}}(x, x'; ls^l_{k,R}) & \\ \qquad \text{if} \quad x^{(l)}_k < x_k < x'_k \leq x^{(l+1)}_k & \\ \frac{x^{(l+1)}_k - x_k}{x^{(l+1)}_k - x^{(l)}_k} K^\infty_{\mathcal{T}}(x, x'; ls^l_{k,L}) + \frac{x'_k - x^{(l)}_k}{x^{(l+1)}_k - x^{(l)}_k} K^\infty_{\mathcal{T}}(x, x'; ls^l_{k,R}) & \\ \qquad \text{if} \quad x^{(l)}_k < x'_k \leq x_k \leq x^{(l+1)}_k & \\ 0 \qquad \text{otherwise} & \end{cases} \tag{38}$$

where x_k (resp. x'_k) is the k^{th} component of the attribute vector x (resp. x').

The sum in (38) is over all possible splitting intervals. The first two choices correspond to the cases when both values x_k and x'_k are outside the interval I^l_k but on the same side with respect to this interval. In this case, the kernel is equal to the kernel computed from the left or the right subset of the learning sample corresponding to that split. The third and fourth choices correspond to the case when both values

fall in the interval I_k^l. For example, in the third situation, the kernel is equal to $K_{\mathcal{T}}^{\infty}(x, x'; ls_{k,L}^l)$ when the cut-point is greater than x'_k, which happens with probability $\frac{x_k^{(l+1)} - x'_k}{x_k^{(l+1)} - x_k^{(l)}}$ (since the cut-point is drawn from a uniform distribution) or equal to $K_{\mathcal{T}}^{\infty}(x, x'; ls_{k,R}^l)$ when the cut-point is lower than x_k, which happens with probability $\frac{x_k - x_k^{(l)}}{x_k^{(l+1)} - x_k^{(l)}}$.

In the case of totally randomized trees, the probabilities $P(I_k^l)$ are easily computed as:

$$P(I_k^l) = \frac{1}{n} \frac{x_k^{(l+1)} - x_k^{(l)}}{x_k^{(N)} - x_k^{(1)}}. \tag{39}$$

In the case of Extra-Trees, the probability of splitting in some interval I_k^l depends on the output values of the learning sample cases and also on the score measure used.

From Eq. (38), it is easy to show that the kernel is continuous with respect to both x and x'. For a given value of x' (resp. x), the function $K_{\mathcal{T}}^{\infty}(x, x'; ls_N)$ is also piecewise multi-linear with respect to x (resp. x'). From (38), it is clear that the function $K_{\mathcal{T}}^{\infty}(x, x'; ls_N)$ is a sum of products of x_k, $k \in \{1, 2, \ldots, n\}$. Furthermore, an attribute can appear at most once in each product. Indeed, the only terms that make appear explicitely x_k are the third and fourth choice in (38) and if x_k and x'_k are in I_k^l, they can only be outside splitting intervals on the k^{th} attribute appearing in $ls_{k,L}^l$ or $ls_{k,R}^l$. Hence, $K_{\mathcal{T}}^{\infty}(x, x'; ls_{k,L}^l)$ and $K_{\mathcal{T}}^{\infty}(x, x'; ls_{k,R}^l)$ can not make intervene anymore the k^{th} attribute through the third or fourth choice in subsequent applications of the recursion (38). Hence, the kernel is piecewise multi-linear with respect to both x and x'. Hyper-intervals where $K_{\mathcal{T}}^{\infty}(x, x'; ls_N)$ is multi-linear with respect to x (resp. x') are delimited by the values of the attributes observed in the learning sample as well as by the values x'_k (resp. x_k), $k = \{1, \ldots, n\}$.

Since the approximation given by an ensemble of Extra-Trees is written as a weighted sum of kernels (8), it is also continuous and piecewise multi-linear when $M \to \infty$. Furthermore, since kernels in (8) are all centered at learning sample points, hyper-intervals where the approximation is multi-linear are defined only by attribute values appearing in the learning sample. Hence, this proves also the form (1) given in Section 5.1 for the approximation provided by an infinite ensemble of Extra-Trees.

Acknowledgments Damien Ernst and Pierre Geurts gratefully acknowledge the financial support of the Belgian National Fund of Scientific Research (FNRS).

References

Ali, K., & Pazzani, M. (1996). Error reduction through learning multiple descriptions. *Machine Learning, 24:3*, 173–206.

Ali, K. (1995). On the link between error correlation and error reduction in decision tree ensembles. Technical report, Department of Information and Computer Science, University of California, Irvine.

Bauer, E., & Kohavi., R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning, 36*, 105–139.

Blake, C., & Merz, C.(1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Breiman, L., Friedman, J., Olsen, R., & Stone, C. (1984). Classification and regression trees. Wadsworth International.

Breiman, L. (1996a). Arcing classifiers. Technical report, University of California, Department of Statistics.

Breiman, L. (1996b). Bagging predictors. *Machine Learning, 24:2*, 123–140.

Breiman, L. (2000a). Randomizing outputs to increase prediction accuracy. *Machine Learning, 40:3*, 229–242.
Breiman, L. (2000b). Some infinity theory for predictor ensembles. Technical Report 579, University of California, Department of Statistics.
Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5–32.
Buntine, W., & Niblett, T. (1992), A further comparison of splitting rules for decision-tree induction. *Machine Learning, 8*, 75–85.
Buntine, W., & Weigend, A. (1991). Bayesian back-propagation. *Complex Systems, 5*, 603–643.
Buntine, W. (1992). Learning classification trees. *Statistics and Computing, 2*, 63–73.
Cutler, A.,& Guohua, Z. (2001), PERT — Perfect random tree ensembles. *Computing Science and Statistics* 33.
Dietterich, T., & Kong, E. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University.
Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning, 40:2*, 139–157.
Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research, 6*, 503–556.
Freund, Y., & Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In: *Proceedings of the 2nd European Conference on Computational Learning Theory*, 23–27.
Friedman, J. (1991). Multivariate adaptive regression splines. *Annals of Statistics, 19:1*, 1–141.
Friedman, J. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. Data Mining and Knowledge Discovery, *1*, 55–77.
Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemna. *Neural Computation, 4*, 1–58.
Geurts, P., Blanco Cuesta A., & Wehenkel, L. (2005a). Segment and combine approach for biological sequence classification. In: *Proceedings of IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, 194–201.
Geurts, P., Fillet,M., de Seny, D., Meuwis, M. -A., Merville, M. -P., & Wehenkel, L. (2005b). Proteomic mass spectra classification using decision tree based ensemble methods. *Bioinformatics, 21:14*, 3138–3145.
Geurts, P., & L. Wehenkel. (2000). Investigation and reduction of discretization variance in decision tree induction. In: *Proceedings of the 11th European Conference on Machine Learning*, 162–170.
Geurts, P., & Wehenkel, L. (2005). 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*. pp. 478–485.
Geurts, P. (2002). Contributions to decision tree induction: bias/variance tradeo. and time series classification. Ph.D. thesis, University of Liège.
Geurts, P. (2003). Extremely randomized trees. Technical report, University of Liège - Department of Electrical Engineering and Computer Science.
Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction*. Springer.
Herbrich, R., Graepel, T., & Campbell, C. (2001). Bayes point machines. *Journal of Machine Learning Research, 1*, 241–279.
Ho, T. (1998). The Random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20:8*, 832–844.
James, G. (2003). Variance and bias for generalized loss functions. *Machine Learning, 51*, 115–135.
Kamath, C., Cantu-Paz, E., & Littau, D. (2002). Approximate splitting for ensembles of trees using histograms. In: *Proceedings of the 2nd SIAM International Conference on Data mining*.
Kleinberg, E. (1990). Stochastic discrimination. *Annals of Mathematics and Artificial Intelligence 1*, 207–239.
Lin, Y., & Jeon, Y. (2002). Random forests and adaptive nearest neighbors. Technical Report 1055, University of Wisconsin, Department of Statistics.
Marée, R., Geurts, P., Piater, J., & Wehenkel, L. (2004). A generic approach for image classsification based on decision tree ensembles and local sub-windows. In: *Proceedings of the 6th Asian Conference on Computer Vision, 2*, 860–865.
Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning, 3*, 319–342.
Nadeau, C., & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning, 52:3*, 239–281.
Quinlan, J. (1986). C4.5: Programs for machine learning. Morgan Kaufmann (San Mateo).
Torgo, L. (1999). Inductive learning of tree-based regression models. Ph.D. thesis, University of Porto.
Valentini, G., & Dietterich, T. (2004). Bias-variance analysis of support vector machines for the development of SVM-based ensemble methods. *Journal of Machine Learning Research, 5*, 725–775.

Webb, G., & Zheng, Z. (2004). Multi-strategy ensemble learning: reducing error by combining ensemble learning techniques. *IEEE Transactions on Knowledge and Data Engineering*, *16:8*, 980–991.

Webb, G. (2000). Multiboosting: a technique for combining boosting and wagging. *Machine Learning, 40:2*, 159–196.

Wehenkel, L., & Pavella, M. (1991). Decision trees and transient stability of electric power systems. *Automatica, 27:1*, 115–134.

Wehenkel, L. (1996). On uncertainty measures used for decision tree induction. In: *Proceedings of Information Processing and Management of Uncertainty in Knowledge Based Systems*, 413–418.

Wehenkel, L. (1997). Discretization of continuous attributes for supervised learning: variance evaluation and variance reduction. In: *Proceedings of the International Fuzzy Systems Association World Congress*, 381–388.

Wehenkel, L. (1998). *Automatic Learning Techniques in Power Systems*. Boston: Kluwer Academic.

Wolpert, D. (1992). Stacked generalization. *Neural Networks, 5*, 241–259.

Zhao, G. (2000). A new perspective on classification. Ph.D. thesis, Utah State University, Department of Mathematics and Statistics.

Zheng, Z., & Webb, G. (1998). Stochastic attribute selection committees. In: *Proceedings of the 11h Australian Joint Conference on Artificial Intelligence*, 321–332.