Chapter 7

Perturbing and combining decision trees

In this chapter, perturb and combine algorithms are discussed in the context of decision tree induction. We start our presentation with the description of Breiman's now classical bagging method and provide some experimental results to show its value in the context of the test problems used in the preceding chapters. We then briefly discuss some other approaches along the same idea which have been proposed in the literature, and end up with a new proposal which goes one step further than existing P&C techniques. This is an extreme random algorithm which builds trees randomized on purpose which, when averaged, yield significantly better results than bagging. Although this algorithm is still in its infancy, we will provide a certain number of indications about the possible extensions of this idea which would allow one to cope efficiently with very large datasets and we argue that this technique can be viewed as a bias reduction algorithm.

7.1 Introduction

Experiments of the previous chapter have shown that the accuracy of a single decision tree is regulated by a very constraining tradeoff between bias and variance. Reducing the variance of the decision tree induction algorithm inevitably results in an increase of the bias and hence yields only slight improvement of accuracy. On the other hand, we have seen in Chapter 4 that perturb and combine algorithms allow to go beyond this intrinsic tradeoff in the context of a learning algorithm by averaging the predictions given by several models (and hence extending the hypothesis space of the method). Actually, most of the general perturb and combine (P&C) techniques described in chapter 4 have been introduced to improve decision trees. The application of P&C techniques in combination with decision trees is appealing for two reasons. First, since decision tree variance is high, averaging often gives impressive improvements with respect to the classical induction algorithm. Second, tree growing is fast and thus building several models is not too costly. This chapter is devoted to a study of P&C algorithms in the context of decision tree induction.

To give a first idea of what can be gained with P&C algorithms applied to decision trees, we first experiment with the popular bagging algorithm on our seven classification problems. Note that while bagging is a general purpose wrapper technique which can in principle be combined with any automatic learning method, we focus here on its use in the context of decision tree induction. Then, we discuss several P&C techniques specifically dedicated to decision trees which have been proposed in the literature and explain their success in the light of the analysis of our previous chapters. From this discussion and our

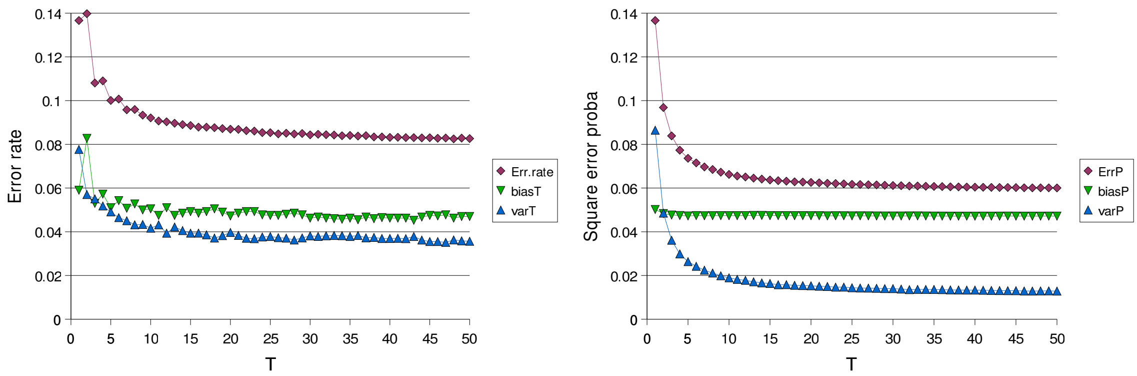

Figure 7.1: Evolution of bias/variance with the number of bootstrap samples with bagging on the omib problem. Left, with error rate, right with square error of probability estimates.

results on decision tree variance, we propose a new algorithm which goes one step further than existing algorithms by building trees almost fully randomly. We show that this technique gives better results than bagging and boosting. Eventually, we conclude with a discussion about interesting future work directions in the context of P&C algorithms, notably in the direction of a reduction of the bias.

7.2 Bagging

Bagging was one of the first ensemble methods proposed in the literature and its first validation was made on tree based algorithms [Bre96b]. In classification, bagging explicitly tries to find an approximation of the majority vote classifier by simulating sampling from the universe by sampling with replacement from the learning set. If the approximation is good, it should result in a decrease of variance while leaving the bias unchanged.

We kindly refer the reader to Section 4.3.1 for the description of this method. We merely note here that since bagging is a (pure) variance reduction technique, it is generally applied to unpruned decision trees (which have a smaller bias than pruned ones). Let us note however, that bagging, especially with a finite number of bootstrap samples, will not reduce the variance to zero. Hence, it is often possible to obtain slightly better results by post-pruning the ensemble of trees in an ad hoc way. We refer the interested reader to reference [GW00] which reports some of our work in this context.

7.2.1 Experiments

The left part of Figure 7.1 shows the evolution of the mean error rate with respect to the number of bootstrap samples on the omib problem (with fully grown trees and the reference learning sample size). Predictions are made according to average of conditional class probability estimates at leaf nodes. The right part of the same figure draws the evolution of the square error of these probability estimates. As expected, the bias of probability estimates is essentially constant whatever the number of trees which are averaged and the regression variance is monotonically decreasing. While Tibshirani's bias is slightly decreasing[1], the main effect on error rates is also a reduction of variance.

Mean results on all datasets are reported in Table 7.1. They are obtained by averaging probability estimates of 25 fully grown trees. On each problem, the reference learning

[1]The short increase in bias with two trees appears because many ties occur in this case and, in our implementation, they are always broken in favor of the same class.

Table 7.1: Effect of bagging (average over 7 problems)

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.2161	90	0.0966	0.1195	0.1326	0.1257	0.0486	0.0771
Bagging (25 trees)	0.1402	1631	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121

Table 7.2: Bagging on the omib problem

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
One bootstrap	0.1371	56	0.0492	0.0878	0.0887	0.1371	0.0483	0.0887
Bagging (25 trees)	0.0842	1386	0.0475	0.0367	0.0483	0.0611	0.0466	0.0144

sample size was used (see Table 5.2 p. 90). The reduction of variance is very impressive (almost a factor 3 for var_T and a factor 6 for regression variance). In the mean time, biases only very slightly increase. Average error rate thus goes from 21,6% to 14%, a relative reduction of 35%.

7.2.2 Bias/variance interpretation

In chapter 4, bias/variance analysis shows that good perturb and combine techniques should introduce some noise ε in the algorithm catching some of the variance coming from the learning set randomness but at the same time should not introduce too much bias. From Chapter 5, it also appears that both discretization variance and attribute selection variance was particularly high. So, bagging is a good idea to introduce such a noise in the context of decision tree induction. Indeed, inducing a tree from a bootstrap sample of the learning set is not going to increase very much the bias since it contains in average more than 60%[2] of the points of the learning set. In the mean time, as decision tree variance is high, even such a small perturbation of the learning set causes very variable models. For example, Table 7.2 compares, on the omib problem, single trees from the whole learning sample, single trees from bootstrap replicates of the learning sample, and, bagging of 25 trees. Building trees from random bootstrap samples slightly increases both regression and classification bias and variance with respect to the (non random) classical algorithm. When averaging such trees, the part of the regression variance which is captured by randomizing the learning set is here given by:

$$\begin{aligned} \text{var}_{LS}\{f_{LS}(\underline{a})\} - \text{var}_{LS}\{E_{\varepsilon|LS}\{f_{LS,\varepsilon}(\underline{a})\}\} &= 0.0789 - 0.0144 && (7.1) \\ &= 0.0645 && (7.2) \end{aligned}$$

i.e. 82% of the initial decision tree variance. The effect on the decomposition of the mean error rate is also a strong reduction of var_T (by 64% in the mean) and a slight increase of bias_T.

7.2.3 Geometrical interpretation

Besides the classical bias/variance interpretation of bagging, the good behavior of bagging can also be interpreted in a geometrical way.

An ensemble of several decision trees can always be represented by a much more complex tree, as it also cuts the input space into hyper-rectangular regions where the

[2] An instance of the learning sample has probability $1-(1-1/N)^N$ of being selected at least once in a bootstrap sample. For large N, this value converges to $1-1/e=0.632$ and hence a bootstrap sample contains about 63.2% of unique instances of the learning sample [BK99].

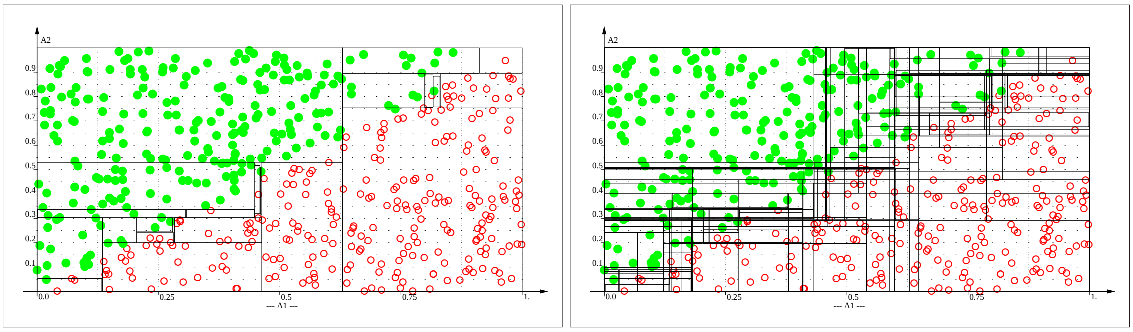

Figure 7.2: Partition of a bi-dimensional input space by a single tree (left) and ten bagged trees (right)

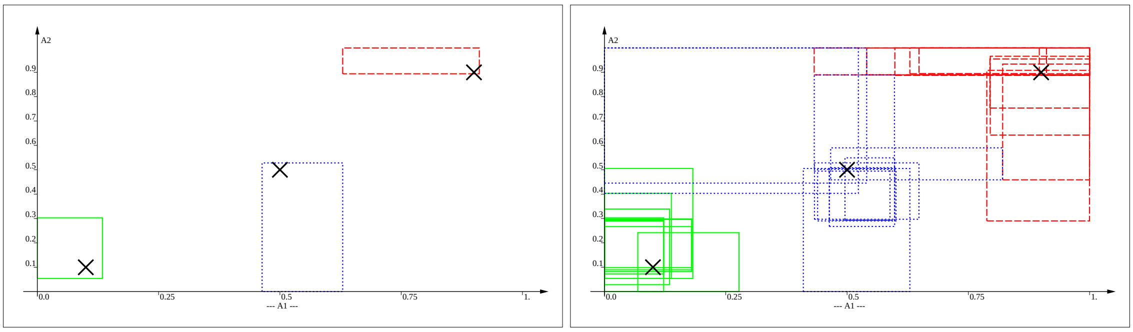

Figure 7.3: Classification regions at three points, single tree (left) and ten bagged trees (right)

prediction is constant. For example, the left part of Figure 7.2 shows the partitioning of a bi-dimensional input space done by a single tree trying to separate empty circles from full circles and the right part of the same figure shows the partitioning done by bagging of ten such trees. From this experiment, the finer cutting of bagging is obvious and this also shows clearly the high variance of decision trees (if all trees were identical, the two figures would also be identical). On Figure 7.3, we can also see the regions on which depend the classification of three points (marked by a cross) of the same input space: left with a single tree, right with bagging. In this latter case, each rectangular region corresponds to the terminal node of one of the ten trees averaged by bagging which is reached by the point. The classification regions with bagging are more extended than those obtained by a single tree and also overlap between each other even if the points are far from each other. This implies that the prediction at a particular point in the input space depends on a much larger number of learning objects on the one hand, and on the other hand, that the influence of a learning object on the prediction at a particular point tends to decrease progressively as the distance between the two attribute vectors increases. We are thus far from the recursive partitioning induced by a single tree.

Also, while the prediction of a single tree is very irregular, the prediction given by bagging is, for the same reason, much “smoother” (although still piecewise constant). Indeed, because of averaging, when we make a small move in the input space, only a small proportion of the terminal nodes reached by our test point changes and thus the perturbation of the aggregated prediction is reduced with respect to the perturbations of individual predictions. Decision tree bagging is thus closer in its behavior to the k-nearest-neighbor method than decision trees. The main advantages with respect to k-NN are the adaptation of the number of neighbors to the position in the input space (the size of the regions) and the implicit automatic induction of a distance measure which takes into account as much as possible only relevant attributes for classification.

Note that, although illustrated in the case of bagging, this analysis remains valid for other perturb and combine techniques in the context of decision trees.

7.3 Specific P&C methods for decision tree induction

Bagging is not the only P&C technique available in the context of decision tree induction. Besides the general P&C techniques presented in Chapter 4 (most of which have been validated exclusively with decision trees), there exist several algorithms specifically designed for tree-based models.

One of the first averaging algorithm is the option trees introduced by Buntine [Bun92] in a Bayesian framework (see section 4.3.5). The goal of option trees is to represent in the same structure several decision trees which posterior probability is high. For example instead of considering only one test at the first level, the most significant tests are all selected and used to build a tree. The process is repeated at all pairs of successor nodes resulting from the different splits. This yields a complex structure which grows exponentially with the depth of the tree. The prediction at a node is obtained by averaging the predictions of all trees starting from this node weighted by the posterior probability of each tree. Although this method was mainly designed to approximate a full Bayesian approach (by keeping only a finite set of highly probable trees), the final result is very similar to a bagged set of decision trees. The initial idea of Buntine was later studied and extended by others (e.g. [OD95] or [KK97]).

Several people have proposed to randomize the induction algorithm of decision trees to obtain a set of different models. For example, Kwok and Carter [KC90] propose to randomize the top node split of a decision tree and their algorithm shows improvement with respect to classical trees on two problems. Along the same idea, Ali and Pazzani [AP95, AP96] propose to select a test at random among the best tests. If S^* is the score of the optimal test, the set of best tests is defined as the tests which have a score greater than $S^* - \beta.S^*$, where β is some constant between 0 and 1. The probability of selecting a test among best ones is proportional to its score. A comparison of this algorithm with bagging and other averaging techniques shows that it gives similar improvement as bagging. Dietterich [Die00c] uses a very similar approach to build random trees: during node splitting, a test is simply selected at random among the 20 best splits. He compares his method with bagging and boosting on several datasets. While not as robust as bagging to noise, his random trees give occasionally better results than bagging.

A number of authors have proposed to randomize decision trees by randomizing the set of candidate attributes. For example, Ho [Ho95, Ho98] builds decision trees from randomly selected subsets of attributes. In a second stage, random trees are obtained by locally (instead of globally) perturbing the subset of attributes used to split a node. In a character recognition application [AG97], Amit and Geman define a very large set of candidate attributes corresponding to different geometrical features computed from images and handle such a large set by selecting random subsets of it at decision tree nodes. In a more recent publication [Bre01], Breiman builds random trees in two ways: first like Ho and Amit and Geman by locally selecting a random subset of attributes, second by constructing locally a set of random linear combinations of attributes and searching among them for the best split. These two variants are then shown to compare favorably with boosting[3] techniques on several classification and regression problems.

[3] Boosting is often better than bagging.

7.4 Extra-trees : extremely randomized trees

The good behavior in practice of the randomization techniques described in the previous section is not surprising in the light of our analysis of decision tree variance. All decisions taken during tree induction are significantly random, especially the choice of tested attributes and discretization thresholds. We have seen how small perturbations of the learning set translate into strong perturbations of the tests which are chosen at tree nodes. All the perturbation techniques discussed in the previous sections are essentially trying to simulate such perturbations. For example, if several tests have close scores, then it is likely that they will be selected in turn depending on the particular learning set choice. This variability can be simulated by selecting one of them at random during induction. However, even in this case, the choice of the best tests still depends on the learning set and hence some (possibly reducible) variance remains.

One interesting question we would like to answer in this section is: how much randomness can be introduced in the tree growing algorithm without decreasing accuracy of the aggregated model, and even more ambitiously, how much randomness should be introduced so as to maximize this accuracy ? From the development of section 4.3, the condition for preserving accuracy is simply not to increase too much the bias when perturbing the learning algorithm. One condition to ensure low bias is to keep the resubstitution error low. So, while previous authors have been quite timid, we propose to go one step further and randomize both discretization thresholds and attribute choices. The algorithm we propose is first described and then compared empirically to bagging on our seven test problems. Eventually, we discuss the potential of such a random algorithm to handle very large datasets.

7.4.1 The algorithm

Our modified tree growing algorithm is described in Table 7.3 (see Table 5.1 p. 86 for a reminder of the non random algorithm). The splitting attribute is first selected at random, then a threshold is drawn from a Gaussian distribution with mean and standard deviation given by the corresponding empirical values in the local subset. To avoid very bad tests, the attribute choice is rejected if the score of the test is lower than a given threshold (the value of s_{th} was fixed at 0.1 in the experiments below, which is 10% of the maximal possible value of the score) and another attribute is selected. The development of a branch is stopped only when the node is pure in terms of the classification output or when all attributes have constant values in the local subset. One important characteristic of this algorithm is that it still produces perfect trees on the learning set (at least to the same extent than the classical algorithm), although much more complex than trees induced classically, and thus it should preserve low bias.

Of course, whether this randomized method is "extreme" in the sense questioned above is an open question. It is however not far from a totally random algorithm, which would build a tree structure without even looking at the learning sample, and use this sample only to determine which nodes are terminal and what labels to attach to them.

With respect to existing random algorithms discussed above, the main originality of our extra-trees is the combination of the randomization of attribute and discretization threshold and the fact that the discretization threshold is really selected at random, i.e. not on the basis of any score (except for the filtering of very bad tests).

Table 7.3: Extra-tree growing

EXTRAtreegrow(ls):
(Build an extremely random tree from the set ls.)

- Create a node $\mathcal{N}$ and set $ls(\mathcal{N})$ to ls;
- Set $L = \{\mathcal{N}\}$, the list of open nodes.
- While L is not empty:
  - Select and remove a node $\mathcal{N}$ in L;
  - If stopsplitting($ls(\mathcal{N})$) is true then
    - Compute conditional class probability estimates $\hat{P}(C|\mathcal{N})$ from $ls(\mathcal{N})$ and attach it to the node $\mathcal{N}$;
  - Else
    - Select an attribute A_i at random from the set of candidate attributes
    - Compute the empirical mean, $\overline{A_i}(ls(\mathcal{N}))$, and standard deviation, $\sigma_{A_i}(ls(\mathcal{N}))$, of this attribute in ls.
    - Draw a threshold, a_{th} at random according to the distribution $N(\overline{A_i}(ls(\mathcal{N})), \sigma_{A_i}(ls(\mathcal{N})))$
    - If $\text{Score}_W(ls(\mathcal{N}), [A_i < a_{th}]) < s_{th}$ return at the first step and select another attribute
    - Compute the subsamples $ls_l = \{(\underline{a}, y) \in ls(\mathcal{N}) | a_i < a_{th}\}$ and $ls_r = \{(\underline{a}, y) \in ls(\mathcal{N}) | a_i \geq a_{th}\}$;
    - Create two nodes $\mathcal{N}_l$ and $\mathcal{N}_r$ and set $ls(\mathcal{N}_l) = ls_l$ and $ls(\mathcal{N}_r) = ls_r$;
    - Insert $\mathcal{N}_l$ and $\mathcal{N}_r$ in L.

Table 7.4: Extra-trees on the omib problem

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Single tree	0.1213	75	0.0420	0.0793	0.0789	0.1213	0.0424	0.0789
Single extra-tree	0.1299	221	0.0415	0.0884	0.0854	0.1299	0.0445	0.0853
Extra-tree averaging (25 trees)	0.0676	5450	0.0370	0.0306	0.0381	0.0525	0.0434	0.0090

Table 7.5: Decision stump averaging on the waveform problem

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
1 Decision stump	0.4492	3	0.3910	0.0582	0.1805	0.1882	0.1640	0.0242
bagging with stump (25)	0.3589	75	0.3250	0.0339	0.1328	0.1657	0.1592	0.0065
Random stump averaging (25)	0.2903	75	0.2510	0.0393	0.1219	0.1780	0.1766	0.0014

7.4.2 Empirical validation

Table 7.4 compares on the omib problem classical single trees with single extra-trees and the aggregation of 25 extra-trees (by averaging of probability estimates) built from the same learning sample. As expected, randomization increases the variance of probability estimates but at the same time keeps their bias low. However, the increase in variance with respect to classical trees is surprisingly very small on this problem. It announces that the variance artificially introduced by our algorithm is very close to the original variance of decision trees and further that aggregating such trees will give good results. Indeed, we observe from the last line of the table that averaging of these random trees yields a very impressive reduction of variance and no significant change in bias. The part of the regression variance which is captured by randomization is thus here $0.0789 - 0.0090 = 0.0699(88\%)$ which is greater than the reduction obtained by bagging on the same problem.

In terms of classification bias, we get the astonishing result that Tibshirani's bias actually decreases slightly, but this can be explained by a similar argument than the one we used in Section 3.3 to explain the decrease of bias with the small learning sets. Indeed, randomization allows to take into account more combinations of tests and hence increase the representation power of a set of decision trees. This phenomenon is more pronounced and easier to explain in the case of decision stumps. With a decision stump, only one attribute can be taken into account. If the choice of the top node attribute is relatively stable with respect to the learning set randomness, then the majority vote classifier will depend merely on this attribute. On the other hand, the random algorithm of Table 7.3 allows every attribute to appear during induction and hence the majority vote classifier can be more flexible. To illustrate this, Table 7.5 shows on the waveform problem the bias/variance profile of a single (non random) decision stump, of the average of 25 stumps obtained by bagging, and of the average of 25 random stumps. Both bagging and random stumps are reducing the bias but the reduction of bias is significantly more important with random test choices than with bagging. Although both methods reduce the variance, the reduction of error is in this experiment is mostly due to the reduction of bias. Since, the size of the learning set limits the size of the trees, similar arguments apply when dealing with fully grown trees: randomization may induce more flexible majority vote classifiers and hence may reduce bias with respect to bagging.

Figure 7.4 (p.138) further shows the evolution of the different bias and variance terms as the number of averaged extra-trees increases, again on the omib problem. The behavior of this method is very similar to the one of bagging in Figure 7.1, but with a more important reduction of variance. The bias of probability estimates is essentially constant whatever the number of extra-trees. After some fluctuations for small numbers of terms, Tibshirani's bias also remains constant when the number of extra-trees increases and the reduction of error rates is mainly due to a reduction of variance.

Classical decision trees, bagging, and, aggregated extra-trees are then compared on all datasets in Figure 7.5, top in terms of the square error of probability estimates, bottom in terms of the mean error rates. Average results are summarized in Table 7.6 for the same three methods as well as single extra-trees. They confirm the results obtained on the omib problem. Extra-tree averaging strongly reduces the regression variance (by 88% in the mean) and slightly increases the regression bias. On the other hand, it reduces both classification variance (by 70% in average) and bias (by 12%) on all datasets. The improvement with respect to bagging is significant and is attributed to both the bias and the variance reductions. Another interesting result is the surprisingly not so bad results of the extra-tree growing algorithm without averaging. In average on all datasets, it only increases the error rates by 5% with respect to the classical decision tree algorithm. On two-norm, it even yields a slight improvement; this further highlights the high variance of decision trees.

7.4.3 Discussion

Even if this algorithm produces better results than bagging (at least on our problems), it suffers one drawback. As thresholds and attributes are not selected on the basis of their discriminative power, induced trees are much more complex than classical trees and ensembles can be very big. In average, the complexity of ensembles of extra-trees is 5 times greater than the complexity of bagged ensembles (from 1630 nodes to 7776 nodes in average).

Hopefully, this increase of complexity still comes with a strong decrease of the computation times because of the strong simplification of the node splitting subroutine. Indeed, local learning sets do not have anymore to be sorted according to all attribute values and only one test has to be evaluated at each test node. To give an idea of what can be gained, we built and tested several models on the two-norm problem on a learning sample of size 5000 and a test sample of size 5000. Results are reported in Table 7.7 (p.139) in terms of complexity, computation times[4] for growing and testing (in second) and error rates. The construction of an extra-tree which is nevertheless 3 times more complex requires 10 times less seconds than the construction of a tree with the classical algorithm. Building an ensemble of 25 extra-trees is 6 times faster than bagging of 25 trees while the complexity is 5 times higher. Despite the increase in tree complexity, the testing time is also slightly smaller with extra-trees than with bagged ones. We explain this by the fact that the extra-trees tend to be more balanced than classical trees (because the threshold is always chosen around the mean in the local subset) and hence paths from the root node to terminal nodes are shorter in average.

Taking into account the fact that this algorithm may be easily parallelized (since models are induced independently of each others), this makes such a method a good candidate for tackling very large data sets. In this prospect, it would be interesting to consider further simplifications of the node splitting subroutine to realize good compromise between computation times and accuracy. For example, the computation of the mean (and hence one pass over the data) could be avoided by selecting the threshold value at random among attribute values appearing in the learning sample. Note that on our (non optimized) implementation of extra-trees, it is still less time consuming to build ten random trees than one single classical tree. Hence, while random tree averaging is certainly more efficient than bagging, it also constitutes an interesting alternative to single tree induction, more

[4]The system is implemented in Common Lisp and runs on a AMD athlon 1GHz microprocessor with 256Mo of main memory. Computation times does not include the time spent in the garbage collector.

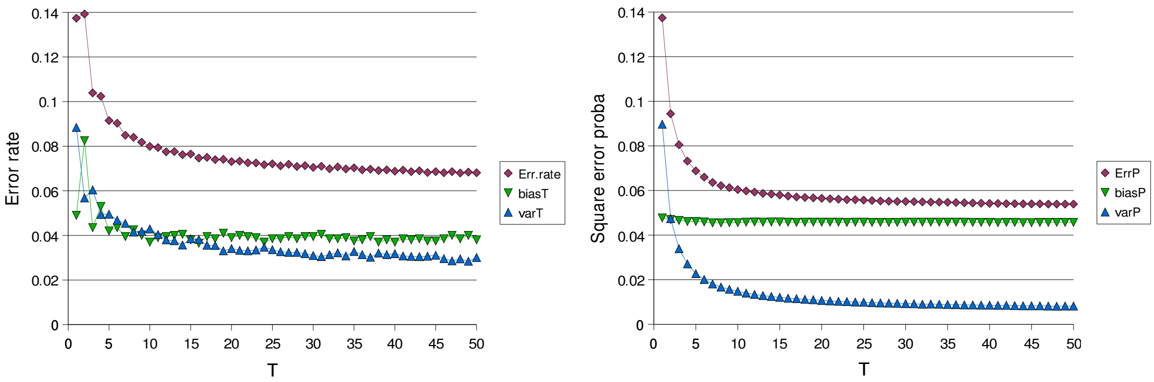

Figure 7.4: Evolution of bias/variance with the number of averaged extra-trees on the omib problem. Left, with error rate, right with square error of probability estimates

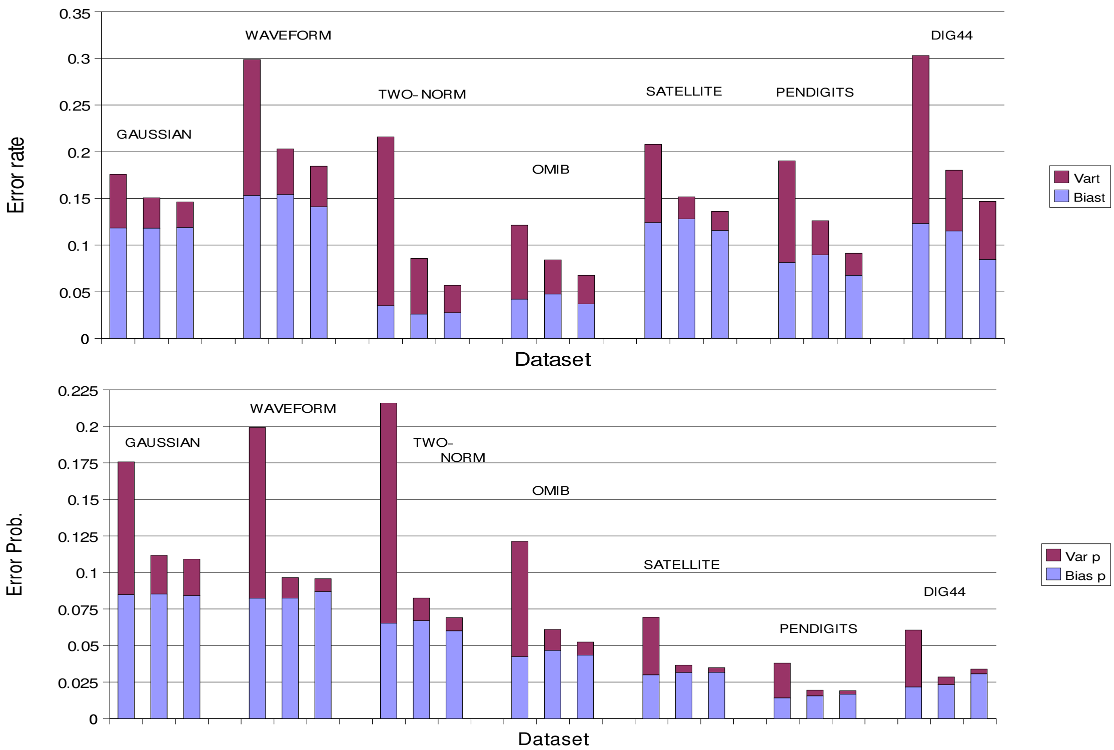

Figure 7.5: Comparison of single trees (left), bagging (center), and, extra-tree averaging (right) on all datasets. Top, in terms of error rates, bottom in terms of the square error of probability estimates

Table 7.6: Bagging and aggregated extra-trees on average on all datasets

	Mean error	compl	bias$_T$	var$_T$	var$_{KW}$	$Err_{\hat{P}}$	bias$^2_{\hat{P}}$	var$_{\hat{P}}$
Single tree	0.2161	89	0.0966	0.1195	0.1326	0.1257	0.0486	0.0771
Bagging (25 trees)	0.1402	1630	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121
Single extra-tree	0.2672	311	0.0925	0.1748	0.1712	0.1401	0.0520	0.0881
Extra-tree averaging (25 trees)	0.1184	7776	0.0845	0.0339	0.0555	0.0592	0.0505	0.0087

Table 7.7: Computation times of several variants on the two-norm problem, LS size 5000 and TS size 5000

	compl	Growing (s)	Testing (s)	Error(%)
Classical tree	671	19.4	0.4	15.84
Extra-tree	2217	1.7	0.3	16.18
Bagging (25 trees)	11809	290.4	5.5	4.3
Extra-tree averaging (25 trees)	54997	43.5	4.7	4.08

efficient and more accurate at the same time.

7.5 The question of bias reduction with P&C algorithms

Given the excellent results of tree averaging in terms of variance reduction, bias is now the dominant source of error of this algorithm. So, if we want to further improve accuracy of decision trees, it is necessary to develop approaches able to fight against bias as well as reducing the variance. Although the random tree algorithm does slightly reduce bias, this is only a side-effect of the increase of flexibility provided by randomization. One of the main sources of bias in decision trees is the limitation of the number of attributes which can be taken into account along a branch. Indeed, because of the recursive partitioning, the number of tests is dependent on the size of the learning set. If many[5] attributes contribute to the classification frontier, decision trees will suffer from high bias. There exist extensions of decision trees which improve the representation power of the candidate tests, for example by taking into account several attributes in one single test. The main drawbacks of these approaches are to deteriorate the computational efficiency of the tree growing algorithm and also to increase variance. From our experiments with random tree averaging algorithm, the solution is obvious: further increase the flexibility of decision trees by building a very large set of functional candidate attributes and search among only a small random subset of them for the best one to split each tree node. The questions of the increase of variance will (hopefully) be solved by aggregation and the question of computational efficiency will be (partially) solved by randomization. We have already cited some literature [AG97, Bre01] proposing methods along these lines. Now the success of these approaches is easy to understand from the preceding analysis. In the third part of this dissertation, we will suggest an application of this idea for time series classification.

Clearly, the very impressive results obtained on our test problems would encourage further research in the context of random tree building. For example, from the theoretical point of view, we would suggest to study the relationship with support vector machines (see [Bur98] for an introduction). Indeed, we see the following analogy between the two approaches: in support vector machines, the problem of bias is solved by applying a transformation from the initial input space into a new potentially very large (often infinite) functional input space where classes are linearly separable. The problem of computational efficiency is solved by considering only destination input space where similarity between objects are in fact easy to compute and expressing the separating hyperplane in the new input space as a function of this similarity measure. On the other hand, the problem of overfitting is solved by keeping the classification *margin* as large as possible. With respect to this approach, the advantage of random tree growing would be that it does not include the delicate a priori choice of a transformation.

Another well known approach to bias reduction is the boosting algorithm. For com-

[5] i.e. many more than average tree depth for the given learning sample size

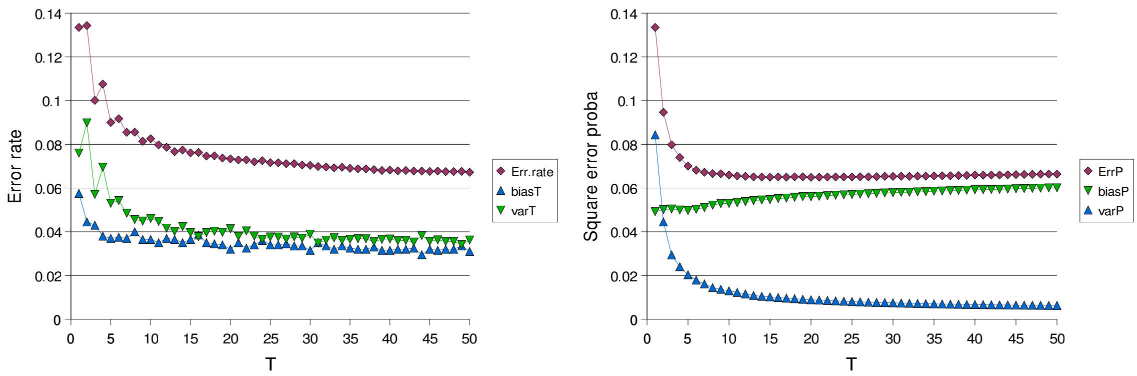

Figure 7.6: Evolution of bias/variance with the number of boosting iterations on the omib problem. Left, with error rate, right with square error of probability estimates

Table 7.8: Boosting vs bagging and extra-tree ensembles (average over all datasets)

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
Bagging (25 trees)	0.1402	1630	0.0969	0.0434	0.0689	0.0624	0.0502	0.0121
Extra-tree averaging (25 trees)	0.1184	7776	0.0845	0.0339	0.0555	0.0592	0.0505	0.0087
Boosting (25 trees)	0.1217	1065	0.0815	0.0403	0.0615	0.0639	0.0546	0.0093

parison, we have applied decision tree boosting[6] on the omib dataset. Figure 7.6 plots the evolution of bias, variance and error rate with respect to the number of iterations. These curves should be compared with the same curves obtained with bagging (on Figure 7.1 p.130) and with extra-trees (on Figure 7.4 p.138). Contrary to these latter methods, we see that boosting indeed reduces both classification bias and variance. On the other hand, it increases the bias of probability estimates. Average results on all datasets are reported in Table 7.8. We observe that the reduction of bias is more important than the reduction of bias by random tree averaging. However, the reduction of variance is also less important and in average, there is no significant difference between the two algorithms in terms of error rate. On the other hand, in terms of computational efficiency the random tree averaging method has the same advantages over boosting than it has over bagging.

Let us however mention that there exist several variants of boosting (see [Bre96a, HTF01]) which could possibly give an even stronger reduction of bias, and also that randomization could be combined with boosting so as to further reduce variance.

7.6 Conclusion

We have clearly seen that with perturb and combine algorithms, it is possible to reduce very strongly the prediction variance of decision trees. Bagging is not the optimal solution in this goal and it is possible to randomize much more the tree induction algorithm to further improve the accuracy by averaging. Our experiments show that, if accuracy is the only criterion, we can almost fully randomize the tree structure, provided that the tree is sufficiently developed so that its terminal nodes correspond to class pure learning sub-samples (which ensures that the empirical error of these trees is as low as possible). Nevertheless, if our experiments show that very much randomness may be introduced in

[6] Contrary to bagging which can cope with fully grown trees, the boosting algorithm works only with pruned trees. In our trials, we have used the stop-splitting criterion based on the G^2 statistic with $\alpha = 0.005$.

the decision tree induction algorithm and still improves accuracy, they do not answer to the question of how much randomness should be introduced to optimize accuracy. For example, we have not yet shown that our algorithm is better in this prospect than existing (less) random approaches discussed in Section 7.3. Also, many other perturbation techniques could be imagined that would maybe improve upon our algorithm. For example, we could experiment with different means to randomize the attribute choice or the discretization thresholds, and also study the impact of the threshold s_{th} which determines the strength of randomization. In the context of bagging, different variants can also be imagined for example by using smaller learning set sizes for bootstrap sampling and/or using sampling without replacement (like in [GW00] or [FH00b] for example). A further possibility would be to combine bootstrap sampling with randomization (like in [Bre01]).

Despite the very impressive improvement of accuracy, perturb and combine techniques still suffer two drawbacks: they do not preserve the efficiency and the interpretability of decision trees. The increase of computation times depends on the particular technique considered (e.g. in the case of extra-trees, the fact that a certain number of trees must be built is nicely counterbalanced by the fact that growing a single extra-tree is much faster than growing a classical tree), and it is also important to realize that the parallelization of bagging and random tree growing is almost trivial. To circumvent the problem of interpretability, some authors have tried to build a tree which does the same job as a set of trees ([Dom97a]). Others [Bre01, HTF01] have proposed some means to extract interpretable information (such as a ranking of the attributes according to their relevance for classification) from such a set. But, intrinsically, a decision tree ensemble is not interpretable. This is the case especially with our extra-tree algorithm where test choices are not really related to the classification.

At this stage of this dissertation, we have thus developed, on the one hand, a way to improve the interpretability of decision trees by stabilizing its parameters but which does not improve their accuracy, and, on the other hand, a way to improve their accuracy by averaging several perturbed models but which deteriorates interpretability and efficiency (to some extent). In the next chapter, we will show that the dual perturb and combine algorithm actually bridges the gap between these two extrema, i.e. improves the accuracy of decision trees but at same time saves interpretability and efficiency by doing a prediction with only one model. This approach will thus lead to a compromise between the two approaches already considered.