The Random Subspace Method for Constructing Decision Forests[1]

Tin Kam Ho

Bell Laboratories, Lucent Technologies

Abstract

Much of previous attention on decision trees focuses on the splitting criteria and optimization of tree sizes. The dilemma between overfitting and achieving maximum accuracy is seldom resolved. We propose a method to construct a decision tree based classifier that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity. The classifier consists of multiple trees constructed systematically by pseudorandomly selecting subsets of components of the feature vector, that is, trees constructed in randomly chosen subspaces. The subspace method is compared to single-tree classifiers and other forest construction methods by experiments on publicly available datasets, where the method's superiority is demonstrated. We also discuss independence between trees in a forest and relate that to the combined classification accuracy.

keywords: pattern recognition, decision tree, decision forest, stochastic discrimination, decision combination, classifier combination, multiple classifier system, bootstrapping

1 Introduction

A persistent problem in using decision trees for classification is how to avoid overfitting a set of training data while achieving maximum accuracy. Some previous studies attempt to prune back a fully-split tree even at the expense of the accuracy on the training data. Other probabilistic methods allow descent through multiple branches with different confidence measures. However, their effect on optimizing generalization accuracy is far from clear and consistent.

In [11] we first proposed that combining multiple trees constructed in randomly selected subspaces can achieve nearly monotonic increase in generalization accuracy while preserving perfect accuracy on training data, provided that the features are sufficient to distinguish all samples belonging to different classes, or that there is no intrinsic ambiguity in the datasets. This offers a better way to overcome the apparent dilemma of accuracy optimization and over-adaptation. In [11] we showed with empirical results that the method works well with binary trees that at each internal node split the data to two sides of an oblique hyperplane.

The validity of the method does not depend on the particulars of tree construction algorithms. Other types of splitting functions, including single-feature splits, such as the C4.5

[1] Parts of this paper have appeared in [11] and [12].

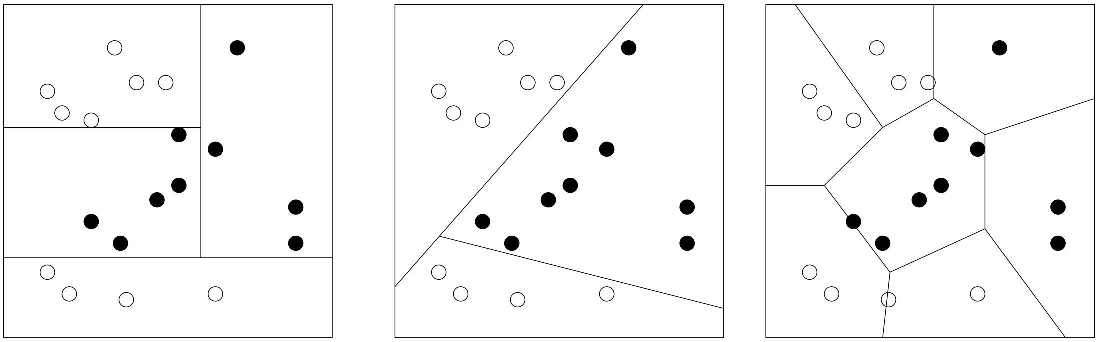
algorithm [26], supervised or unsupervised clustering, distribution map matching [13], and support vector machines [33] can be readily applied. In this paper we will investigate a number of alternative splitting functions.

For simplicity, in this paper we assume that the classification problems are in real-valued feature spaces, though the samples may have only binary or integer feature values. Categorical variables are assumed to be numerically encoded.

2 Methods for Constructing a Decision Tree

Many methods have been proposed for constructing a decision tree using a collection of training samples. The majority of tree construction methods use linear splits at each internal node. We will focus on linear splits throughout our discussions.

A typical method selects a hyperplane or multiple hyperplanes at each internal node, and samples are assigned to branches representing different regions of the feature space bounded by such hyperplanes. The methods differ by the number and orientations of the hyperplanes and how the hyperplanes are chosen. We can categorize such methods by the types of splits they produce that are determined by the number and orientation of the hyperplanes (Figure 1):



(a) axis-parallel linear splits (b) oblique linear splits (c) piecewise linear splits

Figure 1: Types of linear splits.

1. *axis-parallel linear splits*: A threshold is chosen on the values at a particular feature dimension, and samples are assigned to the branches according to whether the corresponding feature values exceed the threshold. Multiple thresholds can be chosen for assignment to multiple branches. Another generalization is to use Boolean combinations of the comparisons against thresholds on multiple features. These trees can be very deep but their execution is extremely fast.

2. *oblique linear splits*: Samples are assigned to the branches according to which side of a hyperplane or which region bounded by multiple hyperplanes they fall in, but the hyperplanes are not necessarily parallel to any axis of the feature space. A generalization is to use hyperplanes in a transformed space, where each feature dimension can be an arbitrary function of selected input features. The decision regions of these trees can be finely tailored to the class distributions, and the trees can be small. The speed of execution depends on the complexity of the hyperplanes or the transformation functions.

3. *piecewise linear splits*: Branches represent a Voronoi tessellation of the feature space. Samples are assigned based on nearest-neighbor matching to chosen anchor points. The anchor points can be selected among training samples, class centroids, or derived cluster centers. These trees can have large number of branches and can be very shallow.

Within each category the splitting functions can be obtained in many ways. For instances, single-feature splits can be chosen by Sethi and Sarvarayudu's average mutual information [28], the Gini index proposed by Breiman et al. [3], Quinlan's information gain ratio [25], or Mingers' G statistic [20] [21] etc. Oblique hyperplanes can be obtained by Tomek links [23], simulated annealing [7][8], or perceptron training [11]. Hyperplanes in transformed spaces can be chosen using the support vector machine method [33]. Piecewise linear or nearest-neighbor splits can be obtained by numerous ways of supervised or unsupervised clustering. There are also many variations of each popular method and we do not intend to provide a complete taxonomy. Interested readers are referred to a survey by Dattatreya and Kanal [5].

An important issue often addressed in previous literature is the stopping criterion for tree construction. Certain types of splits can be produced until the feature space is partitioned into regions containing only samples of a single class (given that there are no intrinsic ambiguities among classes), other types of splits have inherent stopping rules that would not allow such a complete partitioning. A stopping criterion can also be introduced artificially on any type of splits, for instance, by limiting the number of levels of the tree. Another approach is to build a fully-split tree and then prune back certain leaves that are considered overly specific. An alternative to these top-down construction methods is a recently proposed approach that determines the structure of the tree first, and then determines all the splits simultaneously by optimizing a global criterion [29].

Each splitting function defines a model for projecting classification from the training samples to unclassified points in the space. From this point of view, it is not surprising to see that, no method could be universally best for an arbitrary, finite training set. The advantage of each splitting function depends on the distribution of available training samples and the difference of that from the true distributions. On the other hand, if the training samples are sufficiently dense, any one of the functions, if used to generate fully-split trees, yields similar partitions and classification accuracy. Their only differences will be in the sizes of the trees and training and execution speed. Therefore we do not emphasize the advantages and disadvantages of each splitting function and we will leave these to empirical judgement.

3 Systematic Construction of a Forest

Our method is another example of improving accuracy through combining the power of multiple classifiers [15]. Here each classifier is a decision tree, and we call the combined classifier a *decision forest.*

Some previous attempts to construct multiple trees rely on certain heuristic procedures or manual intervention [19][30][31]. The number of different trees they can obtain is often severely limited. We emphasize that arbitrarily introduced differences between trees, such as using randomized initial conditions or perturbations in the search procedures [9], do not guarantee good combination performance. Here good combination performance is defined as 100% correct rate on training data and a monotonic decrease in generalization error as the forest grows in the number of trees, provided that the data do not have any intrinsic ambiguity. Also, it is important to have a systematic procedure that can produce a large number of sufficiently different trees.

Our method relies on an autonomous, pseudo-random procedure to select a small number of dimensions from a given feature space. In each pass, such a selection is made and a subspace is fixed where all points have a constant value (say, 0) in the unselected dimensions. All samples are projected to this subspace, and a decision tree is constructed using the projected training samples. In classification a sample of an unknown class is projected to the same subspace and classified using the corresponding tree.

For a given feature space of n dimensions, there are 2^n such selections that can be made, and with each selection a decision tree can be constructed. More different trees can be constructed if the subspace changes within the trees, that is, if different feature dimensions are selected at each split. The use of randomization in selecting the dimensions is merely a convenient way to explore the possibilities.

The trees constructed in each selected subspace are fully-split using all training data. They are hence perfectly correct on the training set by construction assuming no intrinsic ambiguities in the samples. When two samples cannot be distinguished by the selected features, there will be ambiguities, but if no decision is forced in such cases, this does not introduce errors.

For each tree, the classification is invariant for points that are different from the training points only in the unselected dimensions. Thus each tree generalizes its classification in a different way. The vast number of subspaces in high dimensional feature spaces provides more choices than needed in practice. Hence, while most other classification methods suffer from the *curse of dimensionality*, this method can take advantage of high dimensionality. And contrary to the Occam's Razor, our classifier improves on generalization accuracy as it grows in complexity.

This method of forest building leads to many interesting theoretical questions. For instance, how many of the subspaces must be used before we can achieve a certain accuracy with the combined classification? What will happen if we use all the possible subspaces? How do the results differ if we restrict ourselves to subspaces with certain properties?

Some of these questions are addressed in the theory of *stochastic discrimination* (SD) where the combination of various ways to partition the feature spaces is studied [17] [18]. In the SD theory, classifiers are constructed by combining many components that have weak discriminative power but can generalize very well. Classification accuracies are related to the statistical properties of the combination function, and it is shown that very high accuracies can be achieved far before all the possible weak classifiers are used. The ability to build classifiers of arbitrary complexity while increasing generalization accuracy is shared by all methods derived from the SD theory. Decision forest is one of such methods. While other SD methods start with highly projectable classifiers with minimum enrichment and seek optimization on uniformity [16], with decision forests, one starts with guaranteed enrichment and uniformity, and seeks optimization on projectability.

Another previously explored idea to build multiple classifiers originates from the method of cross-validation. There, random subsets are selected from the training set, and a classifier is trained using each subset. Such methods are also useful since overfitting can be avoided to some extent by withholding part of the training data. Two training set subsampling methods have been proposed: *bootstrapping*[4] and *boosting*[6]. In bootstrapping, subsets of the raw training samples are independently and randomly selected, with replacement, according to a uniform probability distribution. In boosting, the creation of each subset is dependent on previous classification results, and a probability distribution is introduced to prefer those samples on which previous classifiers are incorrect. In boosting weights are also used in final decision combination, and the weights are determined by accuracies of individual trees. Both these methods have been known to be superior to single C4.5 trees [27]. In our experiments we will compare accuracies of forests built by our *subspace* method to those built by such *training set subsampling* methods.

The random subspace method is a *parallel learning* algorithm, that is, the generation of each decision tree is independent. This makes it suitable for parallel implementation for fast learning that is desirable in some practical applications. Moreover, since there is no hill-climbing, there is no danger of being trapped in local optima.

4 The Combination Function

The decisions of n_t individual trees are combined by averaging the conditional probability of each class at the leaves. For a point x, let $v_j(x)$ be the terminal node that x falls into when it descends down tree T_j $(j = 1, 2, ..., n_t)$. Given this, let the probability that x belongs to class c $(c = 1, 2, ..., n_c)$ be denoted by $P(c|v_j(x))$.

$$P(c|v_j(x)) \;=\; \frac{P(c, v_j(x))}{\sum_{k=1}^{n_c} P(c_k, v_j(x))}$$

can be estimated by the fraction of class c points over all points that are assigned to $v_j(x)$ (in the training set). Notice that in this context, since the trees are fully split, most terminal nodes contain only a single class (except for abnormal stops that may occur in some tree construction algorithms) and thus the value of the estimate $\hat{P}(c|v_j(x))$ is almost always 1. The discriminant function is defined as

$$g_c(x) \;=\; \frac{1}{n_t}\sum_{j=1}^{n_t} \hat{P}(c|v_j(x))$$

and the decision rule is to assign x to class c for which $g_c(x)$ is the maximum. For fully-split trees, the decision obtained using this rule is equivalent to a plurality vote among the classes decided by each tree.

It is obvious that the discriminant preserves 100% accuracy on the training set, provided that there is no ambiguity in the chosen subspaces. However, it is possible that, two samples that are distinguishable in the original feature space become ambiguous in a chosen subspace, especially if there is a large reduction in dimensionality. So caution has to be taken in applying this method to very low-dimensional data, and the choice of number of features to select could have a strong impact on accuracy. These will be further illustrated in the experiments.

For an unseen point, $g(x)$ averages over the posterior probabilities that are conditioned on reaching a particular terminal node. Geometrically, each terminal node defines a neighborhood around the points assigned to that node in the chosen subspace. By averaging over the posterior probabilities in these neighborhoods (decision regions), the discriminant approximates the posterior probability for a given x in the original feature space. This is similar to other kernel-based techniques for estimating posterior probabilities, except that here the kernels are of irregular shapes and sizes, and do not necessarily nest.

Because of this, the discriminant is applicable to any algorithm that partitions the feature space into regions containing only or mostly points of one class, for instance, the method of learning vector quantization [10]. The analytical properties of the function and its several variants have been studied extensively by Berlind [2]. Essentially, accuracy of the function is shown to be asymptotically perfect (as number of component classifiers increases) provided that a number of conditions on enrichment, uniformity, symmetry, and projectability are satisfied.

5 Independence Between Trees

For a forest to achieve better accuracy than individual trees, it is critical that there should be sufficient independence or dissimilarity between the trees. There have been few known measures for correlation between decision trees. The difference in combined accuracy of the forest from those of individual trees gives strong but indirect evidence on their mutual independence.

A simple measure of similarity between two decision trees can be the amount that their decision regions overlap [32]. On fully split trees each leaf represents a region labeled with a particular class. On trees that are not fully split the regions can be labeled with the dominating class (ties broken arbitrarily). Given that, regardless of the structure of the trees, we can consider trees yielding the same decision regions equivalent (Figure 2). The similarity of two trees can then be measured by the total volume of the regions labeled with the same class by both trees.

Given two trees t_i and t_j, let their class decisions for a point x be $c_i(x)$ and $c_j(x)$ respectively, we define *tree agreement* $s_{i,j}$ to be

$$s_{i,j} = \int_R p(x)dx,$$

where $R = \{x|c_i(x) = c_j(x)\}$, and $p(x)$ is the probability density function of x.

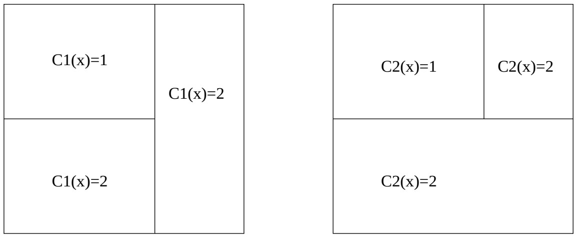


Figure 2: Two trees yielding the same decision regions in a 2-dim feature space ($Ci(x) = j$ if tree i decides that x is in class j).

Given two decision trees, the volume of the overlapping decision regions is determined and in theory can be calculated exactly. However, depending on the form of the splitting function and the dimensionality of the space, the calculation could be difficult to carry out.

Alternatively, one may get an approximation by Monte-Carlo integration, i.e., generating pseudo-random points to a sufficient density in the feature space and measure the agreement of their classification by the trees.

Assuming that the testing samples are representative for the given problems, we will use them to measure the tree agreement. This makes the estimation more computationally feasible and allows us to limit our concern within the neighborhood of the given samples.

It should be noted that severe bias could result if the samples for estimation are not chosen carefully. For instance, if the training samples are used in this context, since the trees are fully split and tailored to these samples, by construction the estimate of tree agreement will always be 1, and this will most likely be an overestimate. On the contrary, since the decision boundaries can be arbitrary in regions where there is no representative sample, if one uses pseudo-randomly generated points distributed over regions far outside those occupied by the given samples, one could risk severely underestimating the tree agreement that is relevant to the problem.

Using a set of n fixed samples and assuming equal weights, the estimate $\hat{s}_{i,j}$ can be written as

$$\hat{s}_{i,j} = \frac{1}{n} \sum_{k=1}^{n} f(x_k),$$

$$\text{where} \quad f(x_k) = \begin{cases} 1 & \text{if } c_i(x_k) = c_j(x_k) \\ 0 & \text{otherwise.} \end{cases}$$

6 Comparison of Results of Tree and Forest Classifiers

A series of experiments were carried out using publicly available data sets provided by the Project Statlog [24]. We used all the four datasets that have separate training and testing data ("dna", "letter", "satimage", and "shuttle"). In each of these datasets, both training and testing samples are represented by feature vectors with integer components. Though some of the feature components are not necessarily numerical by nature (say, the features in the "dna" set are binary codings of a 4-element alphabet), the samples are still treated as points in real-valued spaces. There are no missing values of any features in all four datasets. Table 1 lists the sizes of the training and testing sets provided in each collection.

We compared the accuracies of the following classifiers:

1. Decision forests constructed using the subspace method versus single decision trees.
2. Decision forests constructed using the subspace method versus those constructed using training set subsampling methods.

Table 1: Specifications of data sets used in the experiments.

name of data set	no. of classes	no. of feature dimensions	no. of training samples	no. of testing samples
dna	3	180	2000	1186
letter	26	16	15000	5000
satimage	6	36	4435	2000
shuttle	7	9	43500	14500

3. Decision forests constructed using the subspace method with different splitting functions.

4. Decision forests constructed using the subspace method with different numbers of randomly selected features.

6.1 Forest built on subspaces versus a single decision tree

We first compared the accuracies of the forests constructed using our method against the accuracy of using a single decision tree constructed using all features and all training samples. For easy repetition of results by others, we chose to use the C4.5 algorithm [26] (Release 8) to construct the trees in either case.

In the experiments the C4.5 algorithm was essentially untouched. The forest construction procedure was implemented external to the C4.5 algorithm. That means, the features were randomly selected before the data were input to the algorithm. Decision combination, in this case approximated by simple plurality voting, was done using the class decisions extracted from the output of the algorithm. Along with the use of publicly available data, this experiment can be easily duplicated.

In each run, we first used the original C4.5 algorithm to build an unpruned tree using all the features, and applied the tree to the testing set. Then we repeated the procedure with a pruned tree (using default confidence levels). We then constructed a decision forest, using the same C4.5 algorithm, but for each tree using only a half of the features that were randomly selected. Each tree was added to the forest and the accuracy of the forest on the testing set was measured. We continued until 100 trees were obtained in each forest. Figure 3 compares the testing set accuracies of the forests against those obtained using a single C4.5 tree with the full feature vectors. It is apparent that the forests yield superior testing set accuracies for these four datasets.

It should be noted that in some of these datasets the number of features is not very large (dna: 180; letter: 16; satimage: 36; shuttle: 9). But the method still works well with such relatively low-dimensional spaces. The only exception is that with the “shuttle” data, there are so few features to choose from that the forest does not display a great advantage over the single-tree classifier. To apply the method to very low dimensional data, we suggest first expanding the feature vectors by using certain functions of the original features. In many cases we found that the inclusion of pairwise sums, differences, products, or Boolean combinations (for binary and categorical features) of the raw features served as a good expansion. We will discuss this further in a later experiment with more data sets with very few feature dimensions.

6.2 Comparison to forests constructed by training set subsampling

In this experiment we compared our method of forest building to other methods where each tree was constructed using a randomly selected subset of the training data with all the features. We experimented with both bootstrapping and boosting methods that are of this category. In bootstrapping, subsets of the same size as the original training set were created by sampling with replacement in uniform probability. A tree was constructed with each subset. Boosting was done following the AdaBoost.M1 procedure described in [6].

Again, all trees were constructed by the C4.5 algorithm as in the previous experiment, and the sample selection procedures were implemented externally. Figure 3 shows the accuracies of all the three forests building methods. It can be seen that although the individual trees (say, the first tree in each forests) built by training set subsampling are sometimes more accurate, with more trees, the combined accuracies of the random subspace forests are superior. The results suggest that there is much redundancy in the feature components in these datasets, but the samples are relatively sparse. Interestingly the results also indicate that the bootstrapping and boosting methods are very similar in performance.

Table 2: Average tree agreement of forests constructed by each method.

construction	data set			
method	dna	letter	satimage	shuttle
random subspaces	0.7540	0.6595	0.8228	0.9928
bootstrapping	0.9081	0.8197	0.8294	0.9998
boosting	0.8969	0.7985	0.8205	0.9996

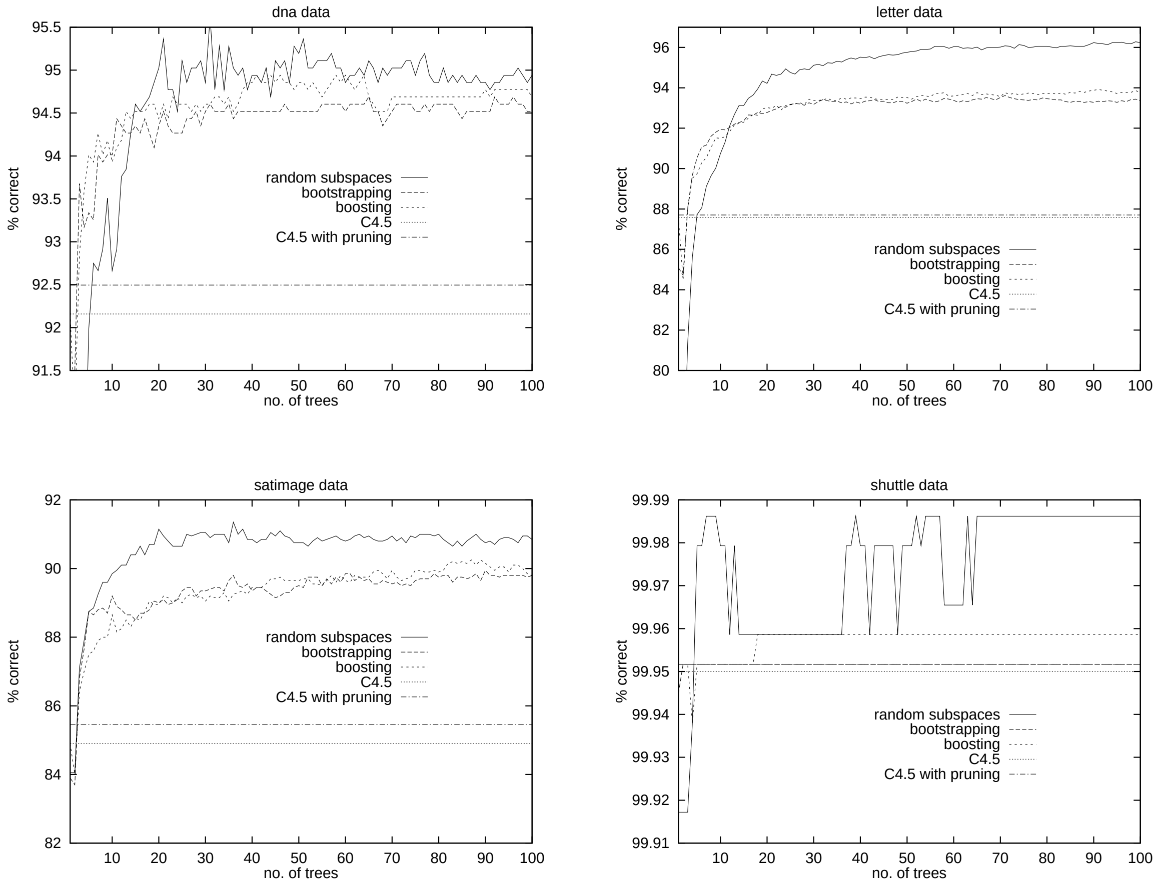


Figure 3: Comparison of test set accuracies: a single C4.5 tree, forests built by the random subspace method, and forests built by bootstrapping and boosting (for the "shuttle" data, accuracies of bootstrapping and the single-tree methods are essentially the same.)

Tree agreement

The differences in tree agreement in forests built using the subspace method and those built using the other two methods can be shown by our measure $\hat{s}_{i,j}$ estimated using the testing data. Table 2 lists the estimated tree agreement for each forest. Each estimate is averaged over all 4950 (100×99/2) pairs among 100 trees. For the dna and letter data, the subspace method yielded very dissimilar trees compared to the other methods. For the other two datasets the differences are not as obvious. Again, differences between the bootstrapping and boosting methods are insignificant.

6.3 Comparison among different splitting functions

In this experiment, we compared the effects of different splitting functions. Eight splitting functions were implemented. Some functions can produce multi-branch splits, but for comparison across different functions, only binary splits were used. For an internal node to be split into n branches, the eight functions assign the points in the following ways.

1. *single feature split with best gain ratio*: Points are assigned to different branches by comparing the value of a single feature against a sequence of $n-1$ thresholds. The feature and thresholds are chosen to maximize Quinlan's information gain ratio [25].

2. *distribution mapping*: Points are projected onto a line drawn between the centroids of two largest classes. Their distances to one centroid are sorted and a set of thresholds are selected so that there are the same number of points between each pair of successive thresholds. Points between a pair of thresholds are assigned to a branch at the next level [14].

3. *class centroids*: Centroids of the $n-1$ largest classes and that of the remaining points are computed. A point is assigned to a branch if it is closest to the corresponding centroid by Euclidean distance.

4. *unsupervised clustering*: n clusters are obtained by complete-linkage clustering using Euclidean distance. Points are matched by Euclidean distance to the centroids of each cluster, and assigned to the branch corresponding to the closest. If there are more points than a preset limit so that clustering is prohibitively expensive, points are sorted by the sum of feature values and divided evenly into n groups.

5. *supervised clustering*: n anchors are initialized using one point from each of the first n classes. The remaining points are matched to the closest anchor by Euclidean distance. The anchors are then updated as the centroids of matched points. The process is repeated for a number of passes and the resulting centroids are used to represent the branches. Points are assigned to the branch corresponding to the nearest centroid.

6. *central axis projection*: First we find the two classes whose means are farthest apart by Euclidean distance. The other classes are matched to these two by proximity of the means. The centroids of the two groups thus obtained are then computed and a line (the central axis) is drawn passing through both centroids. All data points are then projected onto this line. Hyperplanes that are perpendicular to this line are evaluated, and the one that best divides the two groups (causing minimum error) is chosen. Points are then assigned to two sides of the chosen hyperplane as two branches. This method permits only binary splits.

7. *perceptron*: Again the data points are divided into two groups by proximity of the class means. A hyperplane is then derived using the fixed-increment perceptron training algorithm [22]. Points are then assigned to two sides of the derived hyperplane. This method also permits only binary splits.

8. *support vector machine*: $n-1$ largest classes are chosen and the rest are grouped as one class. Points are transformed to a higher-dimensional space by a polynomial kernel of a chosen degree. Hyperplanes that maximize margins in the transformed space are chosen. Each hyperplane divides one class from the others. Points are compared to all chosen hyperplanes, and assigned to the branch corresponding to the one it matches with the largest margin [33].

Figure 4 shows the accuracies of the forests that use these eight functions. There are 50 trees in each forest. In each forest the subspace changes at each split. As expected, the effects of splitting functions are data-dependent and there is no universal optimum. Nevertheless, the improvements in the forest accuracy with increases in number of trees follow a similar pattern across different functions and different datasets. This demonstrates the validity of the forest construction method and its independence of the splitting functions.

Tree agreement

Table 3 shows the average tree agreement (averaged over all 1225 (50×49/2) pairs of 50 trees) for the eight splitting functions and the four datasets, estimated using the testing samples. The absolute magnitude of the measure is data dependent. But the relative magnitudes for the same dataset reveal an interesting pattern. Namely, weaker agreement is observed for forests built with unsupervised clustering, and stronger agreement is observed for the forests built with maximum gain ratio (except for the "satimage" data). The forests built with support vector machines are found to have both the strongest and weakest agreement depending on the data set. There are significant data-dependent differences in the relative order among the estimates given by different splitting functions. Once again this suggests there are no universally optimal splitting functions.

6.4 Comparison among different numbers of random features

In using the subspace method one important parameter to be determined is how many features should be selected in each split. When the splitting function uses a single feature, the evaluation of possible splits is constrained within only the selected features. In other case the hyperplanes are functions of the selected features, so that the number of random features used could affect the results significantly.

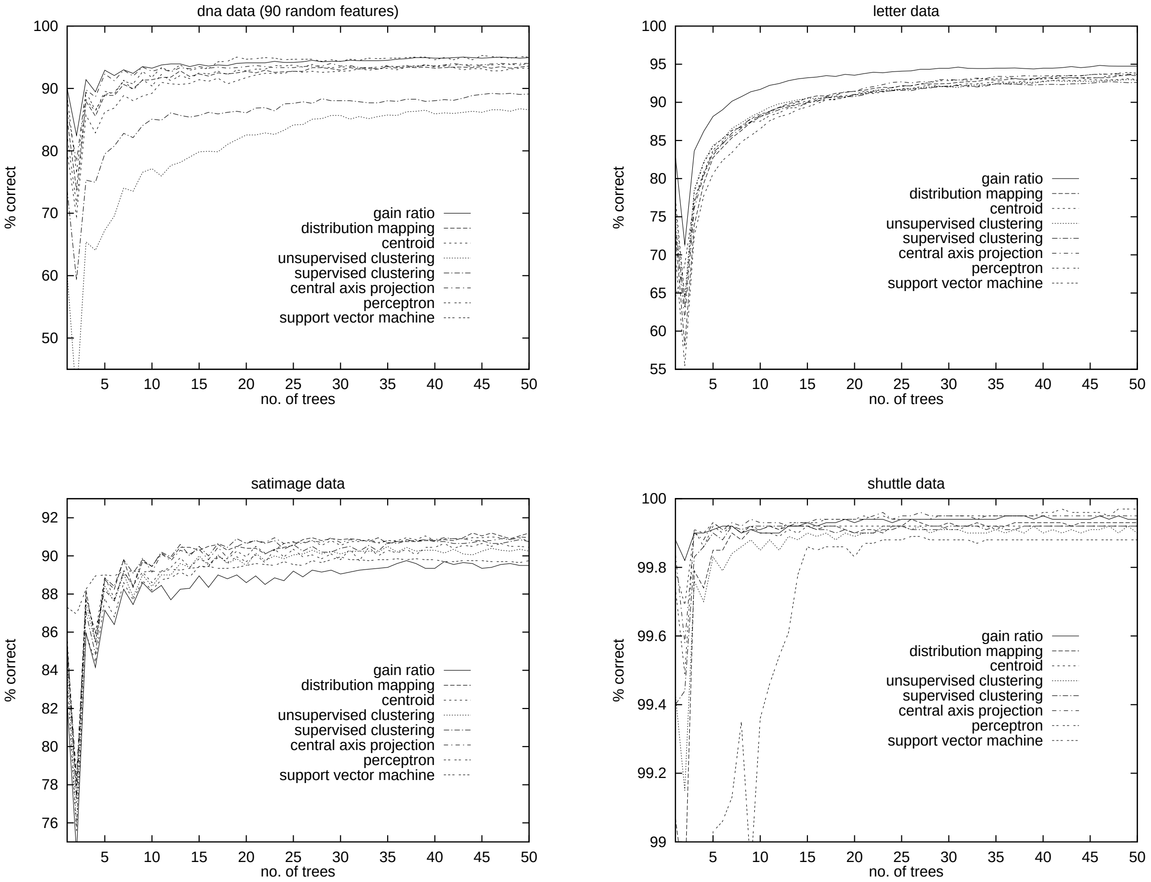


Figure 4: Comparison of test set accuracies with different splitting functions.

Table 3: Average tree agreement with different splitting functions.

splitting	data set			
function	dna	letter	satimage	shuttle
gain ratio	0.8804	0.7378	0.7980	0.9975
dist. mapping	0.7728	0.5861	0.8188	0.9864
centroid	0.7143	0.6593	0.8310	0.9975
unsup. clustering	0.5337	0.5701	0.8110	0.9885
sup. clustering	0.6137	0.6320	0.8378	0.9812
c. axis projection	0.8368	0.6481	0.8440	0.9911
perceptron	0.7913	0.5758	0.8200	0.9950
support vectors	0.7907	0.4827	0.9205	0.8720

We compared the effects of different numbers of features using the “dna” dataset which has 180 features. Each of the same eight splitting functions was used to construct a forest of 50 trees. Again, in each forest the subspace changes at each split. The results show that the effects are stronger on some splitting functions than others (Figure 5). In particular, the split by maximum gain ratio is less sensitive to the subspace dimensionality. Nevertheless, it appears that using half of the features resulted in the best or very close to the best combined accuracies with all splitting functions.

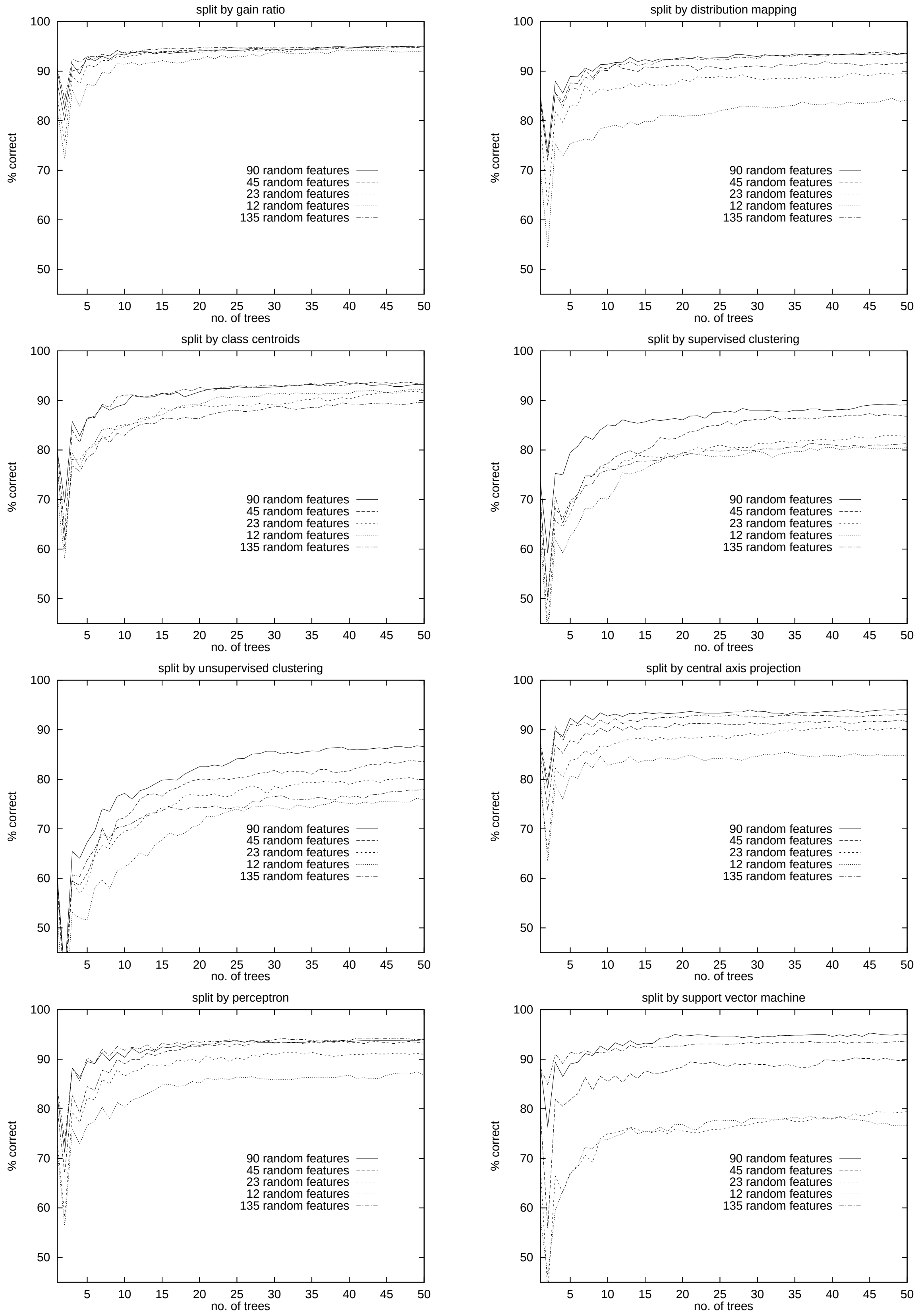
Tree agreement

Table 4 shows the average tree agreement for the eight splitting functions and different numbers of random features, estimated using the testing set (“dna” data). Again, each estimate is averaged over all 1225 (50×49/2) pairs of 50 trees. The agreement generally increases with the number of random features, and the relative ordering among different splitting functions is in good consistency. Trees built with maximum gain ratio are the most similar to each other, and those built with unsupervised clustering are least similar.

Table 4: Tree agreement for different numbers of random features (“dna” data).

splitting	number of random features				
function	12	23	45	90	135
gain ratio	0.7070	0.7773	0.8498	0.8804	0.8849
dist. mapping	0.5855	0.6555	0.7216	0.7728	0.7530
centroid	0.6168	0.6385	0.6717	0.7143	0.6567
unsup. clustering	0.4618	0.4723	0.4862	0.5337	0.5451
sup. clustering	0.5089	0.5253	0.5507	0.6137	0.5571
c. axis projection	0.6412	0.7082	0.7714	0.8368	0.8224
perceptron	0.6174	0.6811	0.7313	0.7913	0.7968
support vectors	0.4985	0.5198	0.6281	0.7907	0.8144

It is important to note that forest accuracy is affected by both the individual tree accuracy and the agreement between the trees. This means optimizing on either factor alone does not necessarily result in the best forest accuracy. For instance, for the “dna” data, although the clustering methods give trees with weakest agreement, their individual accuracies and thus the forest accuracies are not as good as those obtained by, say, splits by maximum gain ratio. But recall also that although the training set subsampling methods produce better individual trees, they are so similar to each other that the forest accuracies are not as good as those obtained by the subspace method. Ideally, one should look for the best individual trees with lowest similarity. But exactly how this dual optimization can be done with an algorithm remains unclear.



split by gain ratio

split by distribution mapping

split by class centroids

split by supervised clustering

split by unsupervised clustering

split by central axis projection

split by perceptron

split by support vector machine

Figure 5: Comparison of test set accuracies with different numbers of random features: 90 (1/2 of all), 45 (1/4), 23 (1/8), 12 (1/16), and 135 (3/4).

The tree agreement measure can be used to order the trees in the forest, so that the most dissimilar trees are evaluated first. This could be useful in practice to obtain better speed/accuracy tradeoff, that is, to use the least number of trees to achieve a certain accuracy. Figure 6 compares the accuracy gains obtained when the trees are sorted by increasing agreement and when they are unsorted. The forest was constructed for the "dna" data using random halves of the features and the maximum gain ratio as the splitting function.

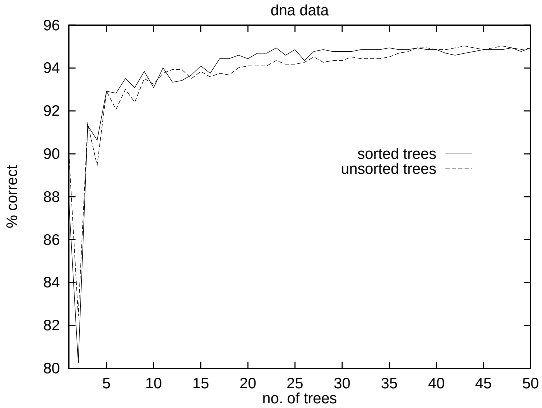


Figure 6: Comparison of test set accuracies between forests with sorted and unsorted trees.

7 Exploiting Redundancy in Data

The Statlog datasets we have used in previous experiments either have a large number of features or a large number of training samples. For some problems this may not be true. In this section we discuss the behavior of the subspace method on a collection of datasets that have a larger variety in size and feature dimensionality. We chose to use datasets from the UC-Irvine machine learning database that have at least 500 samples and have no missing values. We again used the C4.5 package to construct all the trees.

Since there is no separate training and testing sets for each problem, we employed a 2-fold cross-validation procedure. For each problem, we split the dataset randomly into two disjoint halves, constructed a forest with one half and tested it on the other half. Then the training and testing sets were swapped and the run repeated. We performed this procedure for ten times for each dataset. In reporting the test set accuracies, we deleted the outliers, i.e., the runs with the highest and lowest accuracies, and reported the average of the remaining eight runs. This is done to avoid misleading results due to occasional bad sampling on some very small datasets, such as missing an entire class in training or testing. Among these datasets, the "letter" and "splice" data also belong to the Statlog collection, but instead of being split into fixed and separated training and testing sets, they were used in the same way with the others in the cross-validation procedure. For those datasets that have categorical variables, have very few samples, or have very few feature dimensions, we included the cross-products of all pairs of features in random subspace selection. Table 5 shows the number of samples, classes, and features as well as the data types for each dataset.

Figure 7 shows the test set accuracies of decision forests constructed by the random subspace method, bootstrapping, and boosting, together with those of single C4.5 tree (both pruned and unpruned) classifiers. From these plots a few observations can be made:

1. For all datasets, the decision forests are more accurate than both pruned and unpruned single-tree classifiers, and in most cases there are large differences.
2. While all forests are similar in their behavior, namely, accuracies increase with number of trees, those built by bootstrapping or boosting tend to be in closer competition, while in some cases those built by the random subspace method follow a different trend.
3. The subspace method is better in some cases, about the same or worse in other cases when compared to the other two forest building methods.
4. The effect of data types, i.e., whether the features are numeric, categorical, or mixed, is not apparent.

To obtain some hints on when the subspace method is better, in Table 5 we have arranged the entries in the same order with the plots in Figure 7. For the datasets near the beginning of the table, the subspace method performs the best among the three forest building methods. For those towards the end of the table, the subspace method does not have an advantage over the other two methods. From this arrangement and the plots, it should be apparent that the subspace method is best when the dataset has a large number of features and samples, and that it is not good when the dataset has very few features coupled with a very small number of samples (like the datasets “pima”, “tic-tac-toe”, “yeast”) or a large number of classes (“abalone”). For most other datasets the three methods are in close neighborhood of one another. Therefore we expect that the subspace method is good when there is certain redundancy in the dataset, especially in the collection of features. This makes the method especially valuable for tasks involving low-level features, such as in image recognition (e.g. [1]) and in other domains of signal processing. For the method to work on other tasks, redundancy need to be introduced artificially using simple functions of the features.

8 Conclusions

We described a method for systematic construction of a decision forest. The method relies on a pseudo-random procedure to select components of a feature vector, and decision trees are

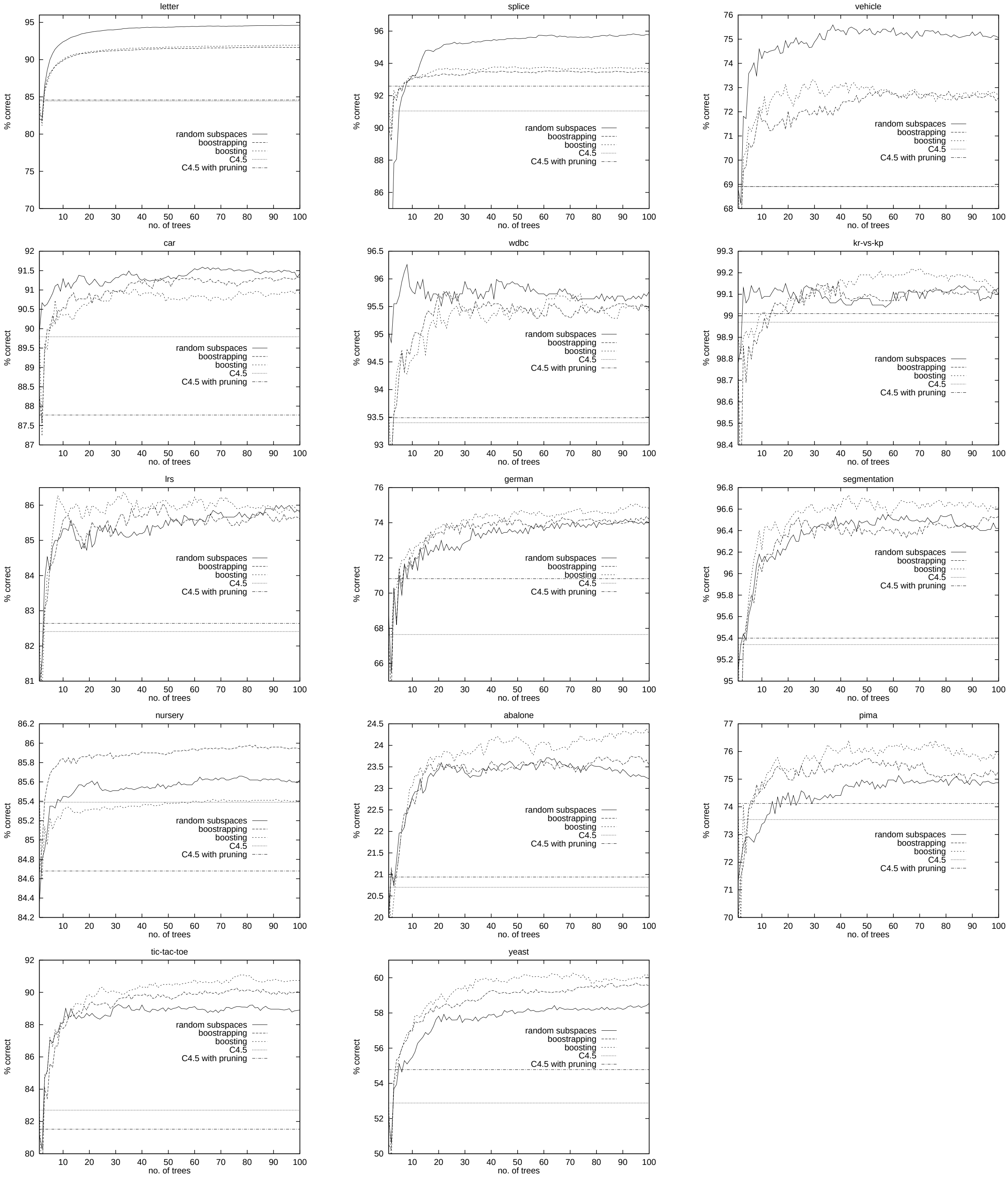


Figure 7: Comparison of test set accuracies of forests and single-trees.

Table 5: Characterstics of datasets used in comparison of methods.

data set	no. of samples	no. of features	no. of classes	feature type(s)
letter	20000	16	26	numeric
splice	3190	60	3	categorical
vehicle	846	18	4	numeric
car	1728	6	4	mixed
wdbc	569	30	2	numeric
kr-vs-kp	3196	36	2	categorical
lrs	531	93	10	numeric
german	1000	24	2	numeric
segmentation	2310	19	8	numeric
nursery	12961	8	5	mixed
abalone	4177	8	29	mixed
pima	768	8	2	numeric
tic-tac-toe	958	9	2	categorical
yeast	1484	8	10	numeric

generated using only the selected feature components. Each tree generalizes classification to unseen points in different ways by invariances in the unselected feature dimensions. Decisions of the trees are combined by averaging the estimates of posterior probabilities at the leaves.

Experiments were conducted using a collection of publicly available datasets. Accuracies were compared to those of single trees constructed using the same tree construction algorithm but with all the samples and full feature vectors. Significant improvements in accuracy were obtained using our method. Furthermore, it is clear that as the forests grow in complexity (measured in the number of trees), their generalization accuracy does not decrease, while maximum accuracies on the training sets are preserved. The forest construction method can be used with any splitting function. Eight splitting functions were implemented and tested. Though there are data dependent differences of accuracies, the improvements in accuracy with increases in the number of trees follow the same trend. The effects of the number of random features to be used were also investigated. It was shown that in the chosen example using half of the feature components yielded the best accuracy. Finally, when compared to two training set subsampling methods for forest building on fourteen datasets, the subspace method was shown to perform better when the dataset has a large number of features and not too few samples. The method is expected to be good for recognition tasks involving many redundant features.

Acknowledgements

The author would like to thank Linda Kaufman who provided the code for constructing support vector machines, and Don X. Sun for helpful discussions.

References

[1] Y. Amit, D. Geman, K. Wilder, Joint Induction of Shape Features and Tree Classifiers, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**, 11, November 1997, 1300-1305.

[2] R. Berlind, *An Alternative Method of Stochastic Discrimination with Applications to Pattern Recognition*, Doctoral Dissertation, Department of Mathematics, State University of New York at Buffalo, 1994.

[3] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, 1984.

[4] L. Breiman, Bagging Predictors, *Machine Learning*, **24**, 1996, 123-140.

[5] G.R. Dattatreya, L.N. Kanal, Decision Trees in Pattern Recognition, in L.N. Kanal, A. Rosenfeld (eds.), *Progress in Pattern Recognition 2*, Elsevier, North-Holland, Amsterdam, 1985.

[6] Y. Freund, R.E. Schapire, Experiments with a New Boosting Algorithm, *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy, July 3-6, 1996, 148-156.

[7] D. Heath, S. Kasif, S. Salzberg, Induction of Oblique Decision Trees, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, **2**, Chambery, France, Aug 28-Sep 3, 1993, 1002-1007.

[8] S. Murthy, S. Kasif, S. Salzberg, A System for Induction of Oblique Decision Trees, *Journal of Artificial Intelligence Research*, **2**, 1, 1994, 1-32.

[9] D. Heath, S. Kasif, S. Salzberg, Committees of Decision Trees, in B. Gorayska, J.L. Mey (eds.), *Cognitive Technology: In Search of a Humane Interface*, Elsevier Science B.V., 1996, 305-317.

[10] T.K. Ho, Recognition of Handwritten Digits by Combining Independent Learning Vector Quantizations, *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, October 20-22, 1993, 818-821.

[11] T.K. Ho, Random Decision Forests, *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, Canada, August 14-18, 1995, 278-282.

[12] T.K. Ho, C4.5 Decision Forests, *Proceedings of the 14th International Conference on Pattern Recognition*, Brisbane, Australia, August 17-20, 1998.

[13] T.K. Ho, H.S. Baird, Perfect Metrics, *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, October 20-22, 1993, 593-597.

[14] T.K. Ho, H.S. Baird, Pattern Classification with Compact Distribution Maps, *Computer Vision and Image Understanding*, **70**, 1, April 1998, 101-110.

[15] T.K. Ho, J.J. Hull, S.N. Srihari, Decision Combination in Multiple Classifier Systems, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**, 1, January 1994, 66-75.

[16] T.K. Ho, E.M. Kleinberg, Building Projectable Classifiers of Arbitrary Complexity, *Proceedings of the 13th International Conference on Pattern Recognition*, Vienna, Austria, August 25-30, 1996, 880-885.

[17] E.M. Kleinberg, Stochastic Discrimination, *Annals of Mathematics and Artificial Intelligence*, **1**, 1990, 207-239.

[18] E.M. Kleinberg, An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition, *Annals of Statistics*, **4**, 6, December 1996, 2319-2349.

[19] S.W. Kwok, C. Carter, Multiple Decision Trees, in R.D. Shachter, T.S. Levitt, L.N. Kanal, J.F. Lemmer (eds.), *Uncertainty in Artificial Intelligence*, **4**, Elsevier Science Publishers B.V. (North-Holland), 1990, 327-335.

[20] J. Mingers, Expert Systems – Rule Induction with Statistical Data, *Journal of the Operational Research Society*, **38**, 1987, 39-47.

[21] J. Mingers, An Empirical Comparison of Selection Measures for Decision-Tree Induction, *Machine Learning*, **3**, 1989, 319-342.

[22] N.J. Nilsson, *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*, McGraw-Hill, New York, 1965.

[23] Y. Park, J. Sklansky, Automated Design of Multiple-class Piecewise Linear Classifiers, *Journal of Classification*, **6**, 1989, 195-222.

[24] Project StatLog, LIACC, University of Porto, *internet address:* ftp.ncc.up.pt (*directory:* pub/statlog/datasets).

[25] J.R. Quinlan, Induction of Decision Trees, *Machine Learning*, **1**, 1986, 81-106.

[26] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[27] J.R. Quinlan, Bagging, Boosting, and C4.5, *Proceedings of 13th National Conference on Artificial Intelligence*, Portland, Oregon, August 4-8, 1996, 725-730.

[28] I.K. Sethi, G.P.R. Sarvarayudu, Hierarchical Classifier Design Using Mutual Information, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **4**, 4, July 1982, 441-445.

[29] I.K. Sethi, J.H. Yoo, Structure-Driven Induction of Decision Tree Classifiers Through Neural Learning, *Pattern Recognition*, **30**, 11, 1997, 1893-1904.

[30] S. Shlien, Multiple Binary Decision Tree Classifiers, *Pattern Recognition*, **23**, 7, 1990, 757-763.

[31] S. Shlien, Nonparametric Classification Using Matched Binary Decision Trees, *Pattern Recognition Letters*, **13**, February, 1992, 83-87.

[32] P. Turney, Technical Note: Bias and the Quantification of Stability, *Machine Learning*, **20**, 1995, 23-33.

[33] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, 1995.