NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

RANDOMIZED ENSEMBLE METHODS FOR CLASSIFICATION TREES

by

Izumi Kobayashi

September, 2002

Dissertation Supervisor: Samuel E. Buttrey

Approved for public release; distribution is limited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2002	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: Randomized Ensemble Methods for Classification Trees			**5. FUNDING NUMBERS**
6. AUTHOR(S) Kobayashi, Izumi			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

We propose two methods of constructing ensembles of classifiers. One method directly injects randomness into classification tree algorithms by choosing a split randomly at each node with probabilities proportional to the measure of goodness for a split. We combine this method with a stopping rule which uses permutation of the outputs. The other method perturbs the output and constructs a classifier using the perturbed data. In both methods, the final classifier is given by an unweighted vote of the individual classifiers. These methods are compared with bagging, Adaboost, and random forests on thirteen commonly used data sets. The results show that our methods perform better than bagging, and comparably to Adaboost and random forests on average.

Additional computation shows that our perturbation method could improve its performance by perturbing both the inputs and with the outputs, and combining a sufficiently large number of trees. Plots of strength and correlation show an interesting relationship. We also explore combining sampling subsets of the training set with our proposed methods. The results of a few trials show that the performance of our proposed methods could be improved by combining sampling subsets of the training set.

14. SUBJECT TERMS Classification, Ensemble methods			**15. NUMBER OF PAGES** 141
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

RANDOMIZED ENSEMBLE METHODS FOR CLASSIFICATION TREES

Izumi Kobayashi
Technical Official Third Grade, Japan Defense Agency
B.S., Ochanomizu University, 1990
M.S., Ochanomizu University, 1992
M.S., Naval Postgraduate School, 1999

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author: _____
Izumi Kobayashi

Approved by:

_____	_____
Samuel E. Buttrey Associate Professor of Operations Research Dissertation Supervisor	Lyn R. Whitaker Associate Professor of Operations Research Dissertation Committee Chair
_____	_____
Robert F. Dell Associate Professor of Operations Research	Robert A. Koyak Assistant Professor of Operations Research
_____	
Craig W. Rasmussen Associate Professor of Mathematics	

Approved by: _____
James N. Eagle, Chair, Department of Operations Research

Approved by: _____
Carson K. Eoyang, Associate Provost for Instruction

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

We propose two methods of constructing ensembles of classifiers. One method directly injects randomness into classification tree algorithms by choosing a split randomly at each node with probabilities proportional to the measure of goodness for a split. We combine this method with a stopping rule which uses permutation of the outputs. The other method perturbs the output and constructs a classifier using the perturbed data. In both methods, the final classifier is given by an unweighted vote of the individual classifiers. These methods are compared with bagging, Adaboost, and random forests on thirteen commonly used data sets. The results show that our methods perform better than bagging, and comparably to Adaboost and random forests on average.

Additional computation shows that our perturbation method could improve its performance by perturbing both the inputs and with the outputs, and combining a sufficiently large number of trees. Plots of strength and correlation show an interesting relationship. We also explore combining sampling subsets of the training set with our proposed methods. The results of a few trials show that the performance of our proposed methods could be improved by combining sampling subsets of the training set.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

ACKNOWLEDGMENTS

First and most of all, I would like to thank Professor Samuel Buttrey for his guidance and encouragement throughout this exiting adventure at NPS. This work could not have been done without him and his patience. He also tried to teach me American idioms and jokes, even though I could never be a good student.

I am grateful to Professor Lyn Whitaker for reading my first (not very well organized) draft and giving me valuable advise. Professors Robert Dell, Robert Koyak, and Craig Rasmussen read my draft and made helpful suggestions. Professor Dell also helped me prepare for the qualifying exam.

Special thanks to my fellow students, especially Tom Cioppa. His sense of humor always made me feel better when we were struggling together to prepare for our qualifying exams and to work on our research. I also acknowledge the many ways I have been encouraged by Scott Frickenstein, Mehmet Ayik, and Martin Gustavo.

There are many people who have supported me outside the school, both in Monterey and Japan. The Handlers, the Cutlers, the Gonzalez, and the Torabayashis warmly welcomed me to their home and shared their time with me. Mr. Masaaki Kunigami helped me start school smoothly by giving me a lot of tips from his experience in Monterey. Season's greetings from Mr. Akira Maruyama and CDR Hirone Kawamura always encouraged me maybe much more than they are aware of.

Finally, I would like to thank the Maritime Staff Office of the Japan Defense Agency for giving me a wonderful opportunity to study at NPS.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

In classification problems, a population consists of observations from several groups (or classes). Each observation belongs to exactly one of the classes, and the task is to construct a "good" classification rule, called a *classifier*, for assigning new observations to groups. We are in general given a *training set*, which is a set of observations whose classes are known, together with measurements on each observation. Then, we construct a classifier using the given training set. The classifier uses the measurements to predict the class of a new observation. The quality of the classifier is usually measured by its accuracy, that is, how accurately it can predict the class of a new observation. Usually, accuracy of a classifier is estimated on the observations that are not used for constructing the classifier.

There are many algorithms for constructing classifiers. However, our study only considers the well-known classification tree. It has a tree structure and is constructed by recursively splitting a node into sub-nodes (called the *child nodes*). A node is split by a splitting rule, which is a condition on one of the input variables. The splitting rule for each node is found by an exhaustive search over the candidates so that the child nodes contain more homogeneous observations than their parent node. Classification tree models are easy to understand because of their simple structure, and they are relatively fast to construct and deploy to predict new observations.

Researchers have focused for years on producing accurate individual classifiers or improving the accuracy of known classifiers. Ensemble methods have given a new direction to research in classification. An ensemble of classifiers is a set of classifiers usually constructed from the same training set and classification algorithm. The prediction of an ensemble for a new observation is given by a weighted or unweighted vote of the predictions of the individual classifiers in the ensemble. Many empirical studies have shown that ensembles are more accurate than the individual classifiers forming the ensembles. Furthermore, some studies have shown that an ensemble could be as accurate as a single highly accurate classifier even if the individual classifiers of the ensemble are only moderately accurate.

Early proposed ensemble methods change the training set for constructing each individual classifier. For example, *bagging* draws a bootstrap sample from the training set and constructs a classifier by using the drawn set. A bootstrap sample is a sample drawn from the training set with replacement of the same size as the training set. Another bootstrap sample is drawn independently of the previous sample and a classifier is constructed using the same classification algorithm. This is repeated until we have sufficient classifiers to form an ensemble. The prediction by the ensemble is given by an unweighted vote of the predictions by the individual classifiers. Similarly, *Adaboost* draws a sample from the training set and constructs a classifier on the drawn set. The difference is that Adaboost maintains the weights on the observations in the training set and uses them in drawing a sample. The weights are the same for all observations at first, and then are changed depending on the accuracy of the previously constructed classifiers. The observations misclassified on the current iteration obtain more weight on the next iteration. The final classifier is given by a weighted vote of the individual classifiers. Adaboost is one of the best ensemble methods in many comparative studies. However, it is not robust when noise is added to the output.

Recently, some researchers have proposed that ensembles be generated by injecting randomness directly into classification tree algorithms. Empirical results have shown that these methods can generate ensembles as accurate as those generated by Adaboost and are sometimes more robust to noise than Adaboost.

In this dissertation, we consider two methods for constructing ensembles of classifiers. The first method injects randomness directly into the classification tree algorithm. Instead of choosing the best splitting rule at each node, we choose a split randomly from among all the possible splits with probabilities proportional to a measure of quality of the splits. This method is combined with a stopping rule (for classification trees) based on permutation of the output. The second method perturbs the output of the training set. That is, the class labels of the training observations are altered to one of the other labels with a given rate, and a classifier is constructed on the perturbed data. Both methods use an unweighted vote of the individual classifiers to form the final classifier.

We apply these methods to thirteen commonly used data sets and estimate their error rates. Other ensemble methods such as bagging and Adaboost are also applied to the same data sets for comparison. The results show that our methods perform better than bagging, and comparably to Adaboost.

Additional computation with the perturbation method suggests that its performance could be improved by perturbing both the inputs and the output, and combining a sufficiently large number of trees. We also consider combining our proposed methods with sampling subsets of the training set. The results from a few trials show that our proposed methods could improve their performance by combining with sampling subsets.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In classification, a population consists of observations from several classes. Each observation belongs to one of the classes, and the task is to construct a "good" rule (called a *classifier*) for classifying new observations. We are, in general, given *a training set* which is a set of observations whose classes are known, together with measurements on each observation. The classifier then uses the measurements to predict the class of a new observation. The quality of the classifier is usually measured by its accuracy, that is how accurately it can predict the class of a new observation. There are many algorithms for constructing classifiers. Researchers have for years focused on producing accurate individual classifiers or improving accuracy of known classifiers.

Ensemble methods have given a new direction to research in classification. Many empirical studies have shown that ensembles are more accurate than the individual classifiers forming the ensembles. Furthermore, some studies have shown that an ensemble could be as accurate as a single highly accurate classifier even if the individual classifiers are only moderately accurate.

As a military application, Monterio (2002) uses *MART* (multiple additive regression trees) an ensemble method developed by Fiedman (2001), in a fraud detection problem at DFAS (Defense Finance Accounting Service). MART is applied to the problem of detecting potentially fraudulent or suspect transactions. The results show that MART can detect fraud about as accurately as the methods currently employed by DFAS. The research also suggests that MART can deliver results in a few hours, comparable to, or better than, the current methodologies that require months of hands-on development by experts.

An ensemble of classifiers is a set of classifiers usually constructed from the same training set and classification algorithm (called a *base classifier*). The prediction made by an ensemble is given by a weighted or unweighted vote of the predictions of the individual classifiers in the ensemble. Methods such as bagging (Breiman, 1996) or Adaboost (Freund and Shapire, 1997) change the training sets used for constructing each individual classifier. Adaboost has been shown to be one of the best ensemble methods

in many comparative studies even though it is not robust to noise. Recently, some research, e.g., Dietterich (1999), Breiman (2001), Cutler and Zhao (2001), has proposed that ensembles be generated by injecting randomness directly into classification algorithms. Empirical results have shown that these methods can generate ensembles as accurate as those generated by Adaboost and are sometimes more robust to noise than Adaboost.

In this dissertation, we consider two methods of constructing ensembles of classifiers. One method injects randomness directly into the classification tree algorithm and the other perturbs the output of the training set. These methods are compared with the well-known ensemble methods such as bagging on several commonly used data sets. The results show that our methods perform better than bagging, and comparably to Adaboost.

A. CLASSIFICATION

In the general classification problem, we have observations consisting of $(y_i, \boldsymbol{x_i})$, $i = 1, \ldots, n$, where y_i is a class label and $\boldsymbol{x_i}$ is a vector of measurements of covariates. The task is to predict the class label y_0 for a new observation with measurement vector $\boldsymbol{x_0}$. The quality of the model is usually measured by accuracy or error rate on test sets.

We are given a training set, and construct a classification model with the training set. There are many types of classification algorithms. The linear discriminant analysis first introduced by Fisher (1936) is a classical technique of classification. The nearest-neighbor technique (Fix and Hodges, 1951) may be the oldest nonparametric classification technique. Classification trees (Breiman, Friedman, Olshen and Stone, 1984) and neural networks (Ripley, 1996) are examples of recent approaches. These classification techniques are introduced in the following sections. The classification tree is the only classification algorithm used in our research and is described in detail in Chapter II.

All of the examples in the following sections are obtained by applying each classification algorithm to Fisher's iris data (Fisher, 1936). The statistical software S-Plus (Insightful, 2000) is used to produce the examples. Fisher's iris data contain 150

observations from each of three species of iris flower: *Setosa*, *Versicolor* and *Virginica*, with 50 observations for each class (species). For each observation, there are four continuous covariates: *Sepal.L.* (sepal length), *Sepal.W.* (sepal width), *Petal.L.* (petal length), and *Petal.W.* (petal width). To graphically illustrate the differences between classification models, among these four covariates, only the sepal length and sepal width are used as predictor variables. The observations are plotted in Figure I-1. In the figure, sepal length is the horizontal axis and sepal width is the vertical axis. The symbols "s," "c," and "v" respectively stand for *Setosa*, *Versicolor* and *Virginica*. The class *Setosa* is easy to separate from the other species in this picture.

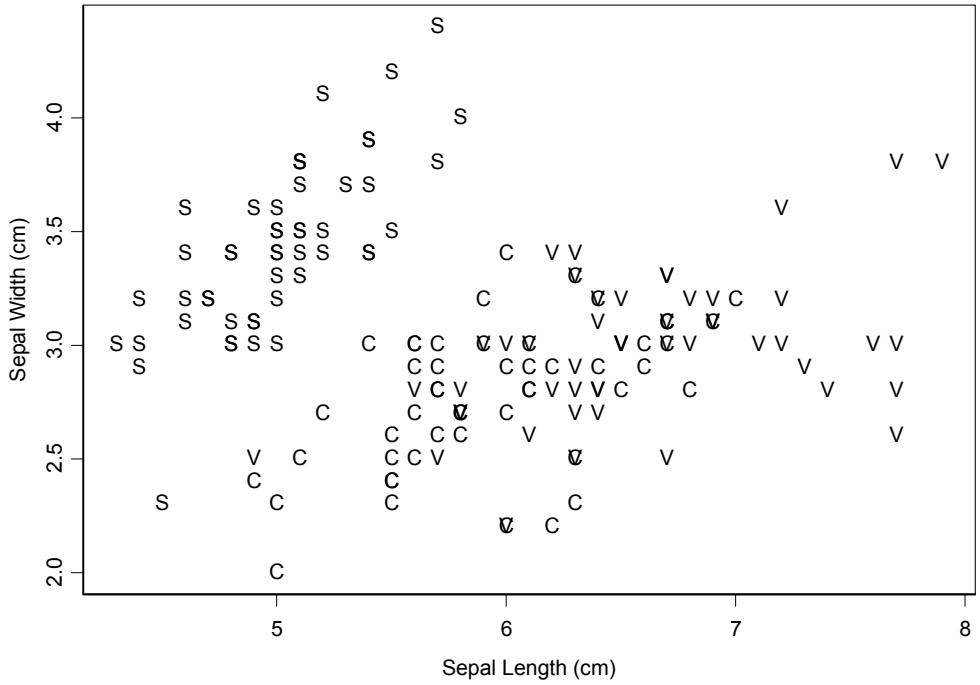

Figure I-1: Fisher's Iris Data.

This Fisher's iris data (Fisher, 1936) contain 150 observations from three species of iris flower: *Setosa*, *Versicolor* and *Virginica*. There are 50 observations for each species. Every observation consists of the class label and four continuous input variables: *Sepal.L.* (sepal length), *Sepal.W.* (sepal width), *Petal.L.* (petal length), and *Petal.W.* (petal width). The observations are plotted according to their sepal length (horizontal axis) and sepal width (vertical axis). The symbol "s" stands for *Setosa*, "c" for *Versicolor* and "v" for *Virginica*. The species *Setosa* is easy to separate from the other two species in this picture.

Before we start to describe each classification algorithm, we introduce the notation used in the dissertation. Throughout the paper, a vector is written in boldface (e.g., $\boldsymbol{a}$), and a matrix is written in boldface capital letters (e.g., $\boldsymbol{A}$). All vectors are assumed to be column vectors. The transposes of a vector and a matrix are written as $\boldsymbol{a}^T$ and A^T, respectively.

Let there be K classes that are named c_k, $k = 1, \ldots, K$. Each observation in the population belongs to exactly one of these classes. The proportion of the class c_k observations in the population is denoted by π_k, $k = 1, \ldots, K$. Naturally $\sum_{k=1}^{K} \pi_k = 1$. They are called the prior probabilities.

Let X denote the input (or predictor) variable and Y denote the output (or response) variable. In our study, X is a p-tuple of which each component may be continuous or categorical, and $Y \in \{c_1, \ldots, c_K\}$. When we refer to a specific component of the input variable, say component j, we denote it by X_j, capital with subscript. We denote an observation of X by $\boldsymbol{x}$ with boldface and italic, and an observation of Y by y. Assume that the observations in the training set T are randomly drawn from the population with the distribution of (Y, X).

Suppose there are n observations in the training set, of which n_k observations are from class c_k. So, $n = \sum_{k=1}^{K} n_k$. Each observation is indexed by i, $i = 1, \ldots, n$. The i^{th} observation is represented by $(y_i, \boldsymbol{x_i})$, where y_i takes one of the values $\{c_1, c_2, \ldots, c_K\}$ and $\boldsymbol{x_i}$ is a vector of p covariates. When we are not referring to a specific observation, we often omit the indices and just write y and $\boldsymbol{x}$. A set of n input vectors is given by the n by p matrix $\boldsymbol{X}$.

1. Discriminant Analysis

Discriminant rules partition the covariate space into non-overlapping regions. Each region is labeled by one of the class labels and a new observation is classified by the class label attached to the region into which it falls. In the case of linear discriminant analysis, the covariate space is partitioned by a set of hyperplanes. There are two quite different approaches for linear discriminant techniques. The first approach, developed by

Fisher (1936), is an empirical method with no assumption about the distributions of the inputs. It finds a linear combination of the inputs by which the classes are "best" discriminated. The second approach assumes that the inputs have the Normal distribution with different means for each class, but a common covariance matrix across classes, and uses the method of maximum likelihood. With two classes of the same number of observations, these two methods give the same prediction for a new observation. Figure I-2 and Figure I-3 give examples of linear discriminant by the two different methods.

Fisher's Linear Discriminant: Fisher's linear discriminant technique is introduced in Fisher (1936). This method does not assume any particular probability distributions for the populations. In his paper, Fisher treats only the case of two classes, finding a linear combination of the predictor variables which maximizes the ratio of the difference between the two means to the standard deviations within classes. Let $\bar{\boldsymbol{x}}$ denote the overall means of the inputs vectors defined by $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$ and let $\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_2$ denote the means for the two classes c_1 and c_2 defined by $\bar{\boldsymbol{x}}_k = \frac{1}{n_k} \sum_{i \in \{i | y_i = c_k\}} \boldsymbol{x}_i$. The objective of Fisher's method is to find a linear combination $\boldsymbol{\alpha}^T \boldsymbol{x}$ which maximizes

$$\frac{\left[\boldsymbol{\alpha}^T (\bar{\boldsymbol{x}}_1 - \bar{\boldsymbol{x}}_2)\right]^2}{\boldsymbol{\alpha}^T \boldsymbol{W} \boldsymbol{\alpha}},$$

where $\boldsymbol{W} = \sum_{k=1}^{2} \sum_{i \in \{i | y_i = c_k\}} (\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)^T$. The numerator of the ratio is the squared distance between the means for the two classes in the projected space and the denominator is proportional to the pooled sample variance of $\boldsymbol{\alpha}^T \boldsymbol{x}$. The vector $\boldsymbol{\alpha}$ which maximizes the ratio is obtained by differentiating the ratio with respect to $\boldsymbol{\alpha}$ and setting the results equal to zero. The solution is

$$\boldsymbol{\alpha} \propto \boldsymbol{W}^{-1} (\bar{\boldsymbol{x}}_1 - \bar{\boldsymbol{x}}_2).$$

When $\boldsymbol{\alpha}$ is determined, new observations are classified by measuring their distance to the mean vectors in the projected space. That is, an observation $\boldsymbol{x}$ is classified to class c_1 if

$$\left|\boldsymbol{\alpha}^T \boldsymbol{x} - \boldsymbol{\alpha}^T \bar{\boldsymbol{x}}_1\right| < \left|\boldsymbol{\alpha}^T \boldsymbol{x} - \boldsymbol{\alpha}^T \bar{\boldsymbol{x}}_2\right|,$$

and class c_2 otherwise.

When there are more than two classes, Fisher's criterion is extended to finding a linear combination $\boldsymbol{\alpha}^T\boldsymbol{x}$ which maximizes the ratio of the between-class sum of squares to the within-class sum of squares (Mardia, Kent and Bibby, 1979). Let $\boldsymbol{B}$ and $\boldsymbol{W}$ be the between-class and within-class sums of squares of $\boldsymbol{X}$. They are defined by

$$\boldsymbol{B} = \sum_{k=1}^{K} n_k \left(\bar{\boldsymbol{x}}_k - \bar{\boldsymbol{x}}\right)\left(\bar{\boldsymbol{x}}_k - \bar{\boldsymbol{x}}\right)^T,$$

and

$$\boldsymbol{W} = \sum_{k=1}^{K} \sum_{i \in \{i | y_i = c_k\}} \left(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k\right)\left(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k\right)^T,$$

where $\bar{\boldsymbol{x}}_k$ is the mean of the class c_k, $k = 1, \ldots, K$. Then the between-class and within-class sums of squares in the projected space ($\boldsymbol{X}\boldsymbol{\alpha}$, i.e., a linear combination of the columns of $\boldsymbol{X}$) are respectively $\boldsymbol{\alpha}^T\boldsymbol{B}\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^T\boldsymbol{W}\boldsymbol{\alpha}$. Therefore, we need to seek $\boldsymbol{\alpha}$ which maximizes the ratio

$$\boldsymbol{\alpha}^T\boldsymbol{B}\boldsymbol{\alpha} / \boldsymbol{\alpha}^T\boldsymbol{W}\boldsymbol{\alpha}.$$

The optimal $\boldsymbol{\alpha}$ is the eigenvector of $\boldsymbol{W}^{-1}\boldsymbol{B}$ corresponding to the largest eigenvalue.

New observations are assigned to the class whose mean is closest to the observation in the projected space; that is, a new observation $\boldsymbol{x}$ is classified as class c_k if

$$\left|\boldsymbol{\alpha}^T \boldsymbol{x} - \boldsymbol{\alpha}^T \bar{\boldsymbol{x}}_k\right| < \left|\boldsymbol{\alpha}^T \boldsymbol{x} - \boldsymbol{\alpha}^T \bar{\boldsymbol{x}}_j\right| \quad \text{for all } j \neq k.$$

Fisher's linear discriminants applied to the iris data with only two covariates (the sepal length and the sepal width) is shown in Figure I-2. The S-Plus (Insightful, 2000) function `discr()` is used to compute the coefficient vector $\boldsymbol{\alpha}$. The decision boundaries are parallel lines. The upper left region is labeled by *Setosa* and the lower right region is labeled by *Virginica*. The region between two lines is labeled by *Versicolor*.

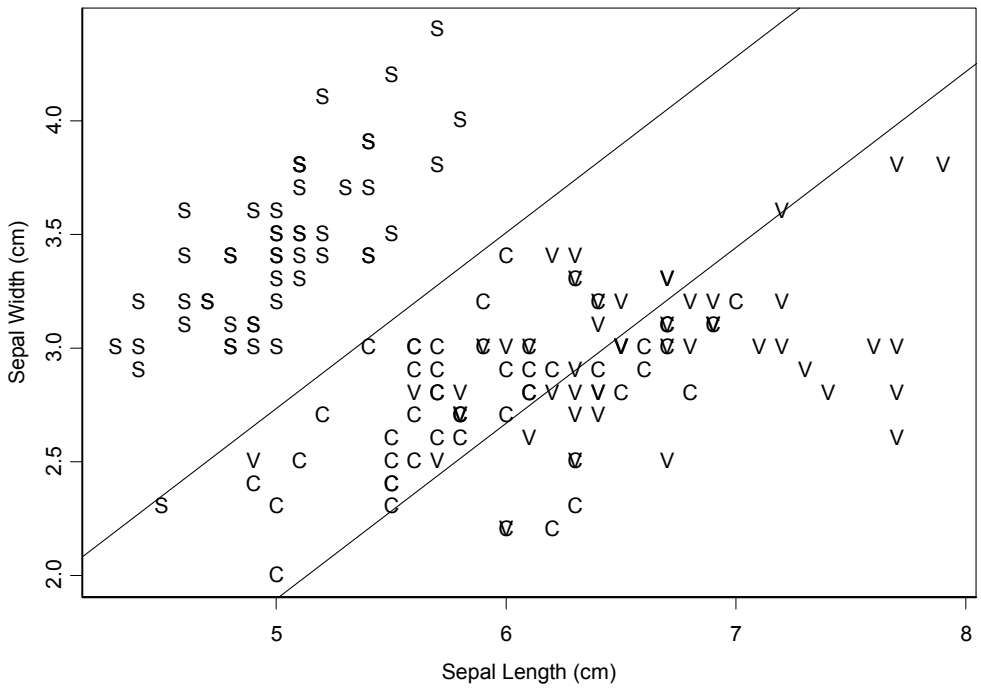

Figure I-2: Fisher's Linear Discriminants Applied to the Iris Data.

Among the four input variables, only the sepal length and the sepal width are used to estimate the parameters of the Fisher's linear discriminants. The upper left region is labeled by *Setosa* and the lower right region is labeled by *Virginica*. The region between two lines is labeled by *Versicolor*. The decision boundaries are parallel lines. The misclassification rate on the training set is 0.22.

Discriminants with Normality Assumptions: Another approach to the linear discriminant depends on the assumptions about the distributions of the observations. In this approach, we assume that the observations in the training set are random samples from populations having the Normal distribution with the different means for the different classes but the same covariance matrix for all classes. Then an observation $\boldsymbol{x}$ is assigned to the class with the largest posterior probability, that is, an observation $\boldsymbol{x}$ is assigned to class c_k, if

$$P(Y = c_k \mid X = \boldsymbol{x}) > P(Y = c_j \mid X = \boldsymbol{x}), \text{ for all } j \neq k, \tag{1.1}$$

where $P(Y = c_k \mid X = \boldsymbol{x})$ is the probability of class c_k given $X = \boldsymbol{x}$. By Bayes' theorem,

$$P(Y = c_k \mid X = \boldsymbol{x}) = \frac{\pi_k f_k(\boldsymbol{x})}{\sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x})},$$

where $f_k(\boldsymbol{x})$ is the joint density of X for class c_k. Because the denominator is common for all classes, (1.1) is equivalent to

$$\pi_k f_k(\boldsymbol{x}) > \pi_j f_j(\boldsymbol{x}).$$

Under the assumptions of normality and common covariance matrices, the density function for the class c_k is given by

$$f_k(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k) \right),$$

where $\boldsymbol{\mu}_k$ is the mean vector for class c_k, $\boldsymbol{\Sigma}$ is the covariance matrix common to each class, and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$.

Consider first the case of two classes. In this case, an observation $\boldsymbol{x}$ is assigned to class c_1 if

$$\pi_1 f_1(\boldsymbol{x}) > \pi_2 f_2(\boldsymbol{x}),$$

and class c_2 otherwise. Using the Normal density function, this condition is written as

$$\frac{\pi_1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_1) \right) > \frac{\pi_2}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_2) \right).$$

By canceling the common terms and taking logarithms of both sides, we obtain the following inequality:

$$\boldsymbol{x}^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) > \log\left(\frac{\pi_2}{\pi_1}\right). \tag{1.2}$$

The decision boundary is obtained where equality holds in (1.2). Since (1.2) is linear in $\boldsymbol{x}$, the decision boundary between two classes is a hyperplane in the covariate space. In real problems, the population parameters are usually not known, and thus the parameters are estimated from the training set and substituted into the formula; $\bar{\boldsymbol{x}}_k$ for $\boldsymbol{\mu}_k$, $\boldsymbol{S}$ for $\boldsymbol{\Sigma}$ and p_k for π_k ($k = 1, 2$). The estimates p_k and $\boldsymbol{S}$ are obtained by

$$p_k = \frac{n_k}{n}, k = 1, 2,$$

$$\boldsymbol{S} = \frac{1}{n-2}\sum_{k=1}^{2}\sum_{i\in\{i|y_i=c_k\}}(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)^T .$$

When the prior probabilities are equal for two classes (i.e., $\pi_1 = \pi_2$), this discriminant rule gives the same classification rule as the Fisher's method does. A proof can be found in Mardia *et al.* (1979).

When there are more than two classes, define the linear discriminant function $\delta_k(\boldsymbol{x})$ for class c_k by

$$\delta_k(\boldsymbol{x}) = \log(\pi_k) + \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - (1/2)\boldsymbol{\mu}_k{}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k.$$

This is obtained by taking the logarithm of $\pi_k f_k(\boldsymbol{x})$ and dropping the terms common to all classes. New observations are classified as the class in which the linear discriminant function $\delta_k(\boldsymbol{x})$ is the largest. As in the two-class case, when the population parameters are not known the corresponding sample estimators are substituted into the formula; $\bar{\boldsymbol{x}}_k$ for $\boldsymbol{\mu}_k$, $\boldsymbol{S}$ for $\boldsymbol{\Sigma}$ and p_k for π_k. These estimators are given by

$$\bar{\boldsymbol{x}}_k = \frac{1}{n_k}\sum_{i\in\{i|y_i=c_k\}}\boldsymbol{x}_i , k = 1, \ldots, K,$$

$$p_k = \frac{n_k}{n}, k = 1, \ldots, K,$$

$$\boldsymbol{S} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i\in\{i|y_i=c_k\}}(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)^T .$$

When the assumption of the equal covariance matrices is dropped, with the same procedure, we get the quadratic discriminant rules. In the quadratic discriminant, the boundary between two regions is a quadratic surface.

Figure I-3 shows linear discriminants applied to the same iris data as before. The S-plus (Insightful, 2000) function `discrim()` is used with the argument `family = Classical("homoscedastic")` to build the model. The decision boundaries are lines, but not the same as those by Fisher's linear discriminants in Figure I-2. The upper left region is labeled by *Setosa*, the lower left by *Versicolor* and the rest by the *Virginica*.

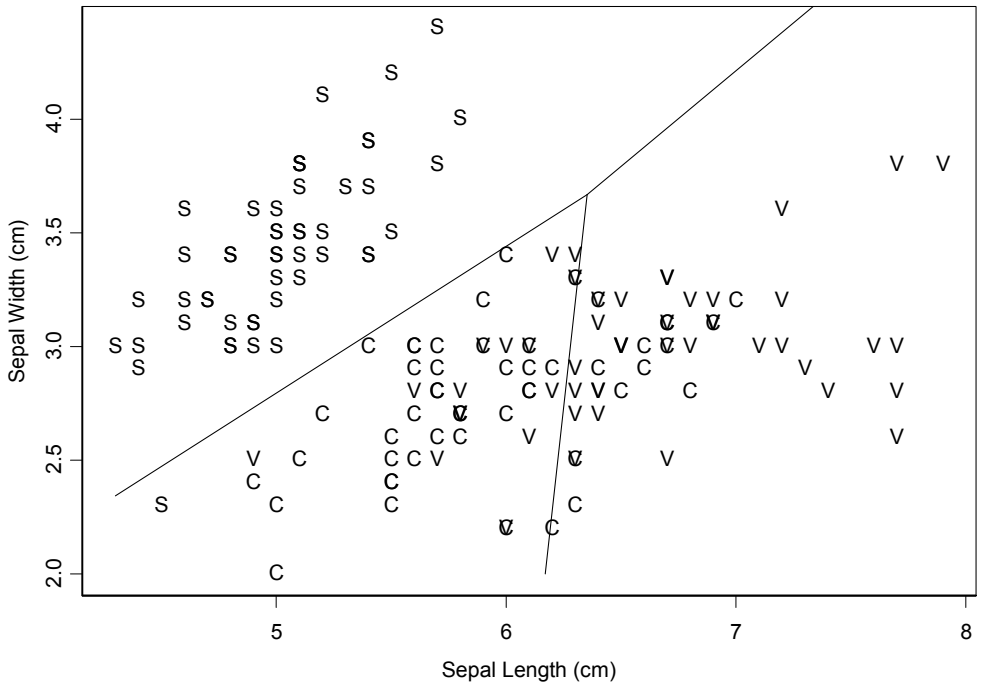

Figure I-3: Linear Discriminants Applied to the Iris Data.

The data of Figure I-2 is used to estimate the parameters of the linear discriminants. The decision boundaries are lines, but not the same as those by Fisher's discriminants in Figure I-2. The upper left region is labeled by *Setosa* and the lower left by *Versicolor* and the rest by *Virginica*. The misclassification rate on the training set is 0.20.

Other discriminant Rules: Logistic discriminant assumes that the log of odds is linear in the input covariates. That is, in the case of two classes, we assume that

$$\log\left(\frac{P(Y = c_2 \mid X = \boldsymbol{x})}{P(Y = c_1 \mid X = \boldsymbol{x})}\right) = \alpha_0 + \boldsymbol{\alpha}^T \boldsymbol{x},$$

where α_0 is a scalar and $\boldsymbol{\alpha}$ is a p-dimensional vector. For more than two classes, the logistic model assumes that

$$\log\left(\frac{P(Y = c_k \mid X = \boldsymbol{x})}{P(Y = c_K \mid X = \boldsymbol{x})}\right) = \alpha_{k,0} + \boldsymbol{\alpha_k}^T \boldsymbol{x}, \; k = 1, \ldots, K-1,$$

where $\alpha_{k,0}$, $k = 1, \ldots, K-1$, are scalars and $\boldsymbol{\alpha_k}$, $k = 1, \ldots, K-1$, are p-dimensional vectors. A simple calculation shows that this is equivalent to

$$P(Y = c_k \mid X = \boldsymbol{x}) = \frac{\exp\left(\alpha_{k,0} + \boldsymbol{\alpha_k}^T \boldsymbol{x}\right)}{1 + \sum_{k=1}^{K-1} \exp\left(\alpha_{k,0} + \boldsymbol{\alpha_k}^T \boldsymbol{x}\right)}, \; k = 1, \ldots, K-1,$$

$$P(Y = c_K \mid X = \boldsymbol{x}) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp\left(\alpha_{k,0} + \boldsymbol{\alpha_k}^T \boldsymbol{x}\right)}.$$

The logistic discriminant models are usually fit by maximum likelihood.

The support vector classifier (SVC) finds linear decision boundaries which separate classes using a different criterion than with Fisher's linear discriminant. The goal of SVC is to find a hyperplane which maximizes the distance from the nearest observations in the training set to the hyperplane. Finding such a hyperplane turns out to be an optimization problem with a quadratic objective function and linear constraints. Such optimization problems are routinely solved (e.g., Bazaraa, Sheali and Shetty, 1993).

The support vector machine (SVM) is a generalization of SVC. SVM first maps the input space into a higher dimension space, using a basic expansion such as polynomials, and finds SVC in the mapped space. In general, linear boundaries in the mapped space translate to non-linear boundaries in the original space.

The logistic discriminant, SVC, SVM and other discriminant models are described in classification and pattern recognition books, for example, Hastie, Tibsirani and Friedman (2001), Duda, Hart and Stork (2000), and Ripley (1996).

2. Nearest Neighbor Methods

The nearest-neighbor technique may be the oldest non-parametric classification technique. It was first introduced formally by Fix *et al.* (1951). A review of the extensive literature on this topic can be found in Dasarathy (1991).

With the nearest-neighbor method, no parametric assumptions about the densities are made. For the *k*-nearest-neighbor rule, given an observation $\boldsymbol{x}$, we look for the closest *k* observations in the training set and predict the class of $\boldsymbol{x}$ by majority vote among the classes of the *k* neighbors. Ties are broken at random. A widely used distance measure is Euclidean distance, after standardizing the inputs so that each of them has mean zero and variance one. The use of Mahalanobis distance may make sense if the within-class distributions are roughly normal with similar covariance matrices. The choice of distance metric is critical to classification especially when there are a small number of observations in the training set. The nearest neighbor rule can also depend on the choice of *k*, the number of neighbors. The cross-validation technique is usually used to determine *k*. Nearest neighbors have shown to be successful in many classification problems. Cover and Hart (1967) prove that, in a large sample, the error rate of the nearest-neighbor with $k = 1$ is at most twice of the Bayes error rate.

One drawback of the nearest-neighbor method is that it is very computationally intensive because we have to measure the distances from the observation being classified to all the observations in the training set and store the information. As an illustration, the five-nearest-neighbor rule is applied to Fisher's iris data as in the last section. The S-plus function `knn()` in the *class* library due to Venables and Ripley (Venables and Ripley, 1999) is used to build the model. The results are shown in Figure I-4. The decision boundaries are nonlinear.

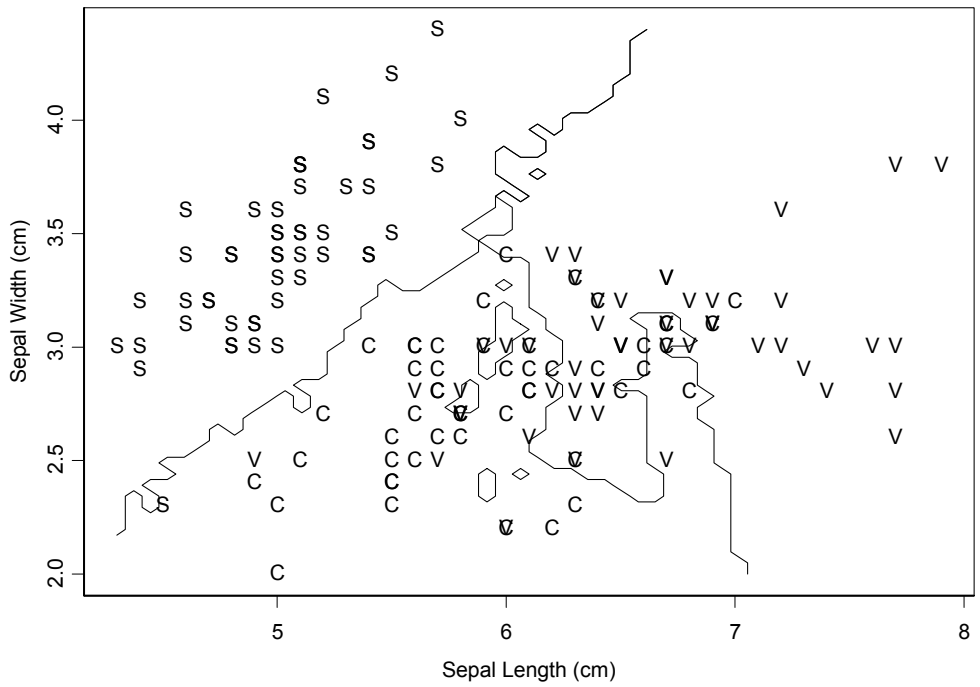

Figure I-4: 5-Nearest Neighbors Applied to the Iris Data.

The data of Figure I-2 is used. For a new observation x_0, the five closest observations in the training set are found. The class of x_0 is chosen by plurality vote of the classes of these five observations. The decision boundaries are nonlinear. The misclassification rate on the training set is 0.16.

3. Neural Networks

Neural networks seem mysterious and almost magical because they are complicated to interpret and yet very accurate. But they are simply nonlinear models represented as weighted directed graphs. An example of the feed-forward network, which is the most commonly used network, is shown in Figure I-5. A detailed description of a network is given in Ripley (1996); a taxonomy of neural networks can be found in Jain and Mao (1996).

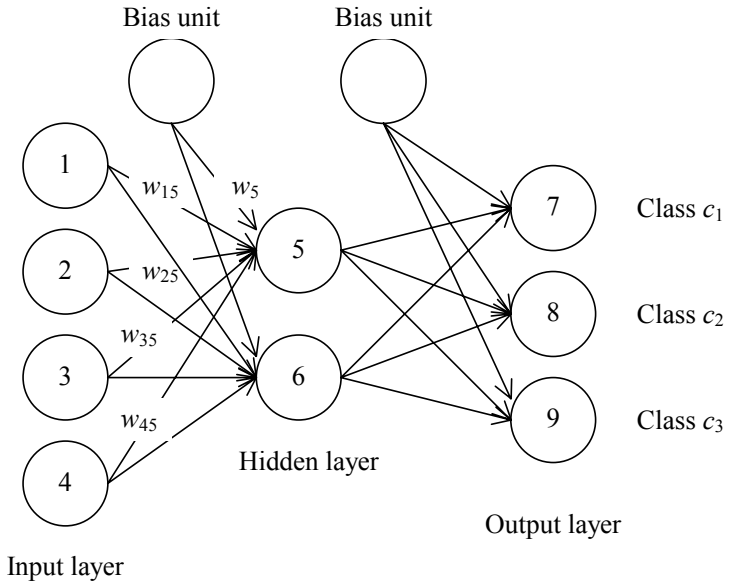

Figure I-5: One Hidden Layer Feed-Forward Network (with Bias Units).

Nodes in the graph are called units and arranged in the layers. There are four units in the input layer, two in the hidden layer and three in the output layer. The connections go from the input units to the hidden units and from the hidden units to the output units. Each connection has its own weight. The input for unit 5 is given by $u_5 = w_5 + \Sigma_{t=1,\,2,\,3,\,4}\ w_{t5}v_t$, where v_i is the output for unit t, w_{t5} is the weight from unit t to unit 5 and w_5 is the bias. The output of unit 5 is $v_5 = f_5(u_5)$, where f_5 is the activation function for unit 5. Output v_5 is used to create the inputs for units 7, 8 and 9.

A feed-forward network is constructed out of nodes called "units." These units can be arranged in layers called the input, hidden and output layers. Networks can have

multiple hidden layers. However, we consider only one hidden layer in this section. A good example using multiple hidden layers can be found in Hastie *et al.* (2001). Connections generally go from every unit in the input layer to every unit in the hidden layer, and from every unit in the hidden layer to every unit in the output layer. There are no connections between units in the same layer.

The network operates on one observation at a time. The input units (the units in the input layer) represent the input variables; for a particular observation, then, the values in the input units are just the values of the different variables for that observation. The number of the input units depends on the number and type of the input variables. We need one unit for each continuous variable and multiple units for each categorical variable (usually one unit for each level of the categorical variable). For the K-class problem, there are K units in the output layer and the k^{th} output unit models the probability of class c_k. The number of hidden units (units in the hidden layer) is determined by the user and the optimal number differs depending on the problem. Having too few units gives a poor fit, and too many units will overfit to the training set.

Unit t takes a scalar u_t as the input for the *activation function* f_t and emits the output $v_t = f_t(u_t)$. In the case of units in the input layer, u_t is the relevant variable's value, and f_t is typically taken to be the identity function. Therefore each input unit emits the observation's value for its variable. These outputs traverse the outgoing connections to the units in the next layer. Each connection carries a weight; these weights make up the set of parameters to be estimated. Each unit in a layer (except the input layer) computes the sum of the products of its inputs and its connections' weights, and adds a constant to create the input for its activation function. In the notation of Figure I-5, unit 5 in the hidden layer computes $u_5 = \sum_i w_{i5} v_i + w_5$. That unit then emits the value $v_5 = f_5(u_5)$ down its connections to nodes 7, 8 and 9. The constant w_5 is called the bias and can be considered the weight of the connection from a unit whose value is permanently one (the bias units are indicated in the figure). Activation functions can be different for different layers, and furthermore can be different for different units. As desired properties of activation functions, Duda *et al.* (2000) suggest that they be nonlinear, continuous, smooth and monotone. As noted above, the identity function $f(u) = u$ is usually used for

input units. For the hidden and output units, the logistic function $f(u) = \exp(u)/(1 + \exp(u))$ is usual.

In general, the output of unit t in the output layer can be represented by

$$v_t = f_t\left(w_t + \sum_h w_{ht} f_h\left(w_h + \sum_i w_{ih} u_i \right) \right), \tag{1.3}$$

where w_{ih} is the weight of the connection from input unit i to hidden unit h, w_{ht} is the weight of the connection from hidden unit h to output unit t, w_t is the bias for unit t, and f_t is the activation function for unit t. The first summation is over all hidden units, the second over all input units. Equation (1.3) shows that the output can be expressed as a nonlinear function of the input variables.

Even when the logistic function is used as the activation function for output units, there are no guarantees that the outputs from the output units sum to one. To make the output units sum to one so that they can be used as probability estimates, the softmax method is often employed; apply the identity function in the output units, i.e., $v_t = f_t(u_t) = u_t$ for unit t in the output layer and compute the output of unit t as

$$o_t = \frac{\exp(v_t)}{\sum_{t \in \{t \mid \text{unit } t \text{ is an output unit}\}} \exp(v_t)}.$$

The value o_t then gives the estimated probability of the class corresponding to unit t.

In the example in Figure I-5, there are three classes and four continuous covariates, and we model the problem by using a network with two hidden units. This network needs four input units and three output units. The units except the bias units are numbered from one through nine: units 1 through 4 are in the input layer, 5 and 6 in the hidden layer, and 7 through 9 in the output layer. Suppose that we use the identity function for the input units, the logistic function for the hidden units and the softmax method for the output units. For the input units, the input value u_t is the value of the input variable and the outputs are given by $v_t = f_t(u_t) = u_t$, $t = 1, 2, 3, 4$. So, the input units exist just to take the input variables and provide them to the hidden units. The outputs v_t, $t = 1, 2, 3, 4$, are used to create the inputs for the hidden units. For instance, the input for unit 5 is computed as

$$u_5 = w_5 + \sum_{i=1}^{4} w_{i5} u_i ,$$

and the output is

$$v_5 = f_5(u_5) = \exp(u_5)/(1 + \exp(u_5)),$$

where w_{i5} is the weight on the connection from unit i, i = 1, 2, 3, 4, to unit 5 and w_5 is the bias. The output for unit 6, v_6, is computed similarly. The output units take weighted sums of v_5 and v_6 to create the inputs u_t, t = 7, 8, 9, and apply the softmax method to compute their outputs. The output of unit 9, for example, is given by

$$o_9 = \exp(v_9) / \exp(v_7 + v_8 + v_9) = \exp(u_9) / \exp(u_7 + u_8 + u_9),$$

which is the estimate of the probability for class c_3 (the class corresponding to unit 9).

Once the model is specified, the weights can be found by minimizing error functions that measure how well the model fits to the training set. For the error function, we often choose sum-of-squares

$$\sum_{i=1}^{n} \left\| y_i - \hat{y}_i \right\|^2 = \sum_{i=1}^{n} \sum_{k=1}^{K} \left(y_{ik} - \hat{y}_{ik} \right)^2 ,$$

or cross-entropy

$$- \sum_{i=1}^{n} y_i \log \hat{y}_i = \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik} ,$$

where y_i is the output of the i^{th} observation in the training set and $\hat{y}_i$ is its estimate by the network. The output y_i is represented by a vector of K components $(y_{i1}, y_{i2}, \ldots, y_{iK})^T$ in which the k^{th} component y_{ik} is one and the others are zero if the class of the i^{th} observation is c_k. The k^{th} component of $\hat{y}_i$ is the output of unit t in the output layer corresponding to class c_k. In general, the weights are obtained by numerical methods such as gradient descent. The different initial values for the weights can lead to different solutions, which makes neural networks unstable. Hastie *et al.* (2001) recommend choosing the initial weights to be random values near zero. Some other issues such as avoiding local minima and overfitting in training neural networks can be found also in Duda *et al.* (2000), Hastie *et al.* (2001) or Ripley (1996).

The resulting model can be used for predicting the classes of new observations. The class estimation for an observation is usually given by the label of the class corresponding to the output unit with the largest output value.

The neural network algorithm is applied to the same iris data as the previous sections (Figure I-6). The data have four continuous covariates, but only two of them are

used as predictor variables. The S-Plus function `nnet()` from the *nnet* library due to Venables and Ripley (Venables *et al.*, 1999)) is used. The feed-forward network has one hidden layer with two units. Because we use two continuous predictor variables and there are three classes, the network needs to have two input units and three output units. The region is separated by nonlinear curves.

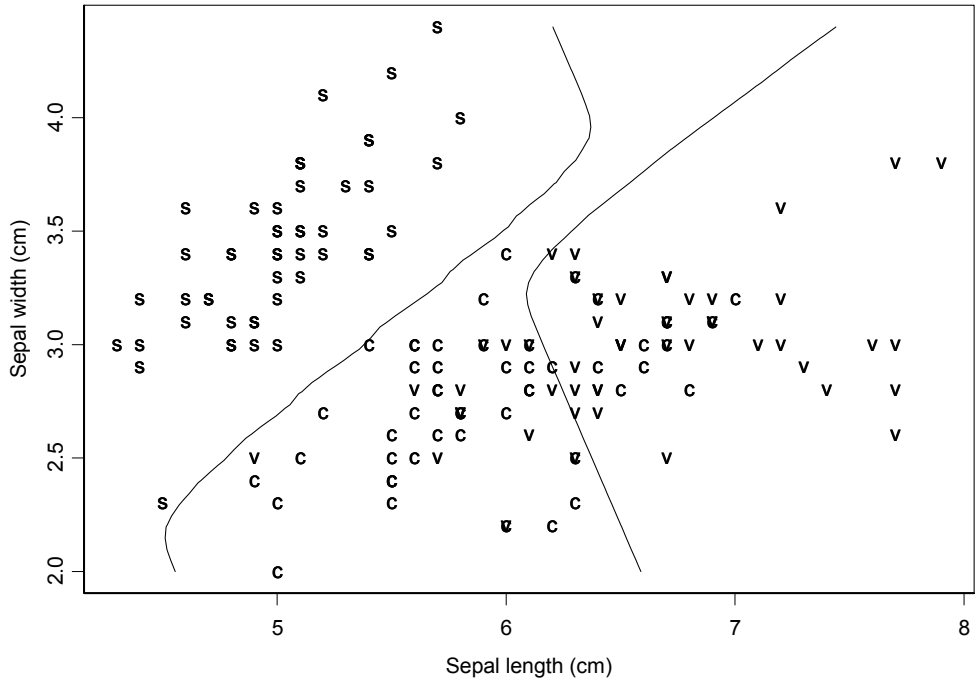

Figure I-6: One Hidden Layer Neural Network Applied to Iris Data.

The data of Figure I-2 is used. This network has two input units and three output units, because we use two continuous predictor variables and there are three classes. The hidden layer consists of two hidden units and three output units. The region is separated by smooth nonlinear curves. The misclassification rate on the training set is 0.17.

4. Classification Trees

The final technique introduced in this chapter is the only classification algorithm used in our study: the classification tree [Breiman *et al.* (1984), Ripley (1996, Chapter 7)]. The classification tree is an nonparametric method which partitions the covariate space into non-overlapping rectangular spaces. Because of their simple structure, classification tree models are easy to understand. The classification tree technique is relatively fast to fit and predict. It can easily handle both continuous and categorical data with no extra effort. It is invariant to monotone transformations of the individual continuous variables. It deals with interactions automatically. It is well known that classification trees are unstable, that is, that small changes in the training set can drastically change the model structure. This sounds like a disadvantage of classification trees, however, unstableness takes an important role in bagging which is an ensemble method introduced briefly in the next section. A detailed description of classification trees is contained in Chapter II.

Figure I-7 gives an example of classification tree which is built using the same iris data as in the previous sections. In the figure, nodes represented by ellipses are non-terminal nodes and by rectangular boxes are terminal nodes. The numbers underneath the nodes indicate the node numbers. There are five terminal nodes, which implies that this tree partitions the covariate space into five non-overlapping regions. The resulting partition of the covariate space by this tree is shown in Figure I-8. We can see that the tree partitions the region into five non-overlapping rectangular regions. Each partition in Figure I-8 corresponds to one of the terminal nodes of the tree of Figure I-7. For instance, the left upper region labeled by *Setosa* corresponds to node 5.

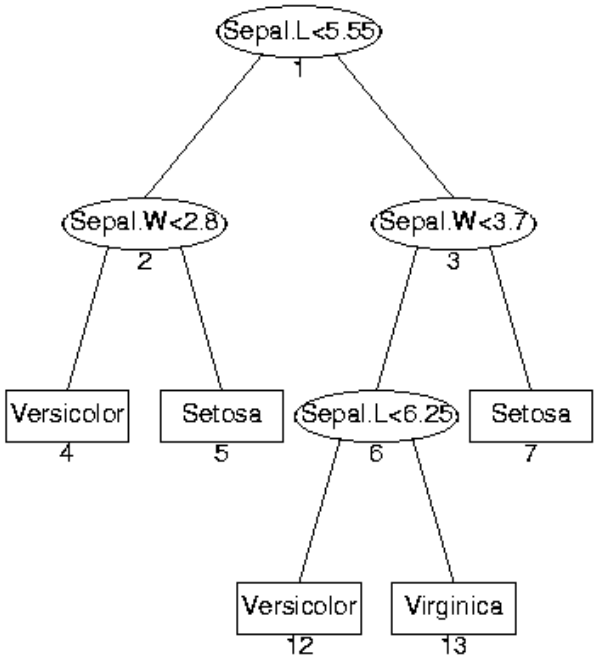

Figure I-7: Classification Tree Applied to Iris Data.

Classification tree algorithm is applied to the same iris data as Figure I-2. Nodes represented by ellipses are non-terminal nodes and by rectangular boxes are terminal nodes. The numbers underneath the nodes indicate the node numbers. There are five terminal nodes, which implies that this tree partitions the covariate space into five non-overlapping regions.

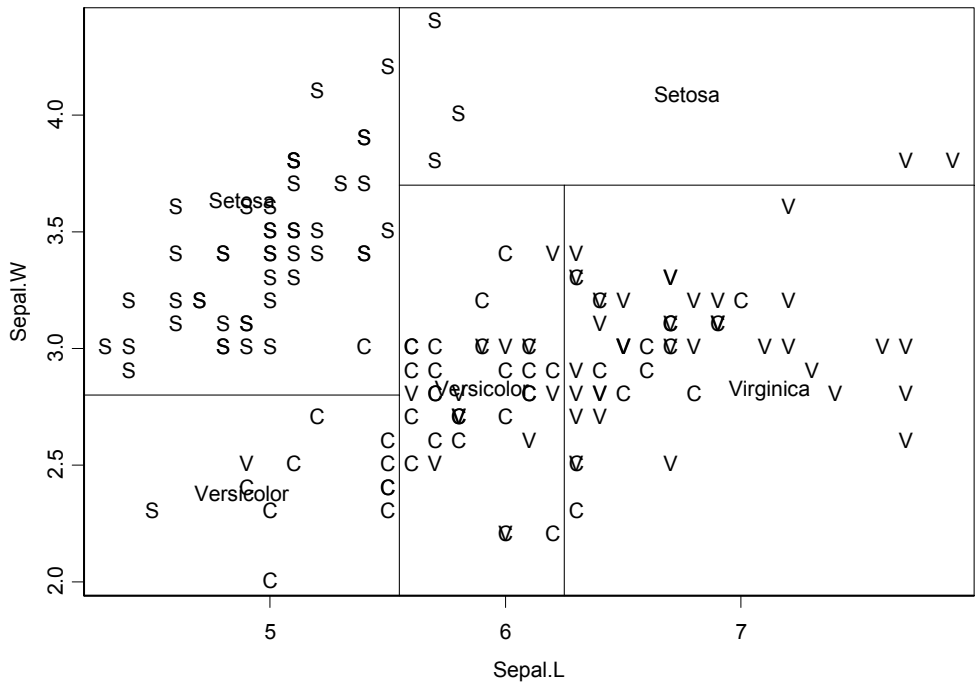

Figure I-8: Decision Boundaries of Classification Tree.

This figure shows how the tree in Figure I-7 divides the covariate space. The region is partitioned into the five non-overlapping rectangular regions labeled by the class labels. The label for each region is given by the plurality vote of the observations in the training set falling into that region. These regions correspond to the terminal nodes in Figure I-7. For example, the upper left region labeled *Setosa* corresponds to node 5. The misclassification rate on the training set is 0.21.

B. ENSEMBLE METHODS

Ensemble methods have received much attention recently, and finding methods for constructing good ensembles is an active area of research in classification. An ensemble of classifiers is a set of classifiers, usually constructed from the same training set. New observations are predicted by combining predictions of individual classifiers using a weighted or unweighted vote.

Many methods for constructing ensembles have been shown empirically to be very accurate. Breiman (1996) constructs ensembles that consist of classifiers built on bootstrap samples (samples of size n drawn with replacement) of the training set T and called this method *bagging* (bootstrap aggregating), where n is the size of T. Freund *et al.* (1997) develop a method called *Adaboost* which from many empirical studies appears to be one of the best ensemble methods. As in bagging, Adaboost draws, on the m^{th} iteration, a sample $T^{(m)}$ of size n from the training set T and builds a classifier $C^{(m)}$ on $T^{(m)}$. However, Adaboost differs from bagging in two important respects. First, the sample drawn on each iteration depends on the previous iterations because observations classified incorrectly by $C^{(m)}$ receive more weights on the $(m + 1)^{\text{th}}$ iteration. In contrast, in bagging, all observations in the training set have equal weights at all iterations and hence the drawn samples are independent one from another. Second, Adaboost uses a weighted vote for predicting a new observation while bagging uses an unweighted vote. Breiman (1998) calls this type of method *arcing* (adaptive reweighting and combining). He suggests that arcing works well not because of the specific form of Adaboost (which he calls *arc-fs* in his paper), but because of the adaptive resampling. This is demonstrated by the construction of an *ad hoc* arcing method called *arc-x4* that can perform as well as Adaboost.

The following figures give examples of ensemble methods. In Figure I-9, bagging is applied to the iris data introduced in Figure I-1. This ensemble consists of 100 classification trees grown on the bootstrap samples. That is, for each tree, we draw a bootstrap sample and grow it on the drawn sample. The final (ensemble) classifier is given by an unweighted vote of these 100 trees. In Figure I-10, Adaboost is applied to

the same data. This ensemble also consists of 100 classification trees. We use an algorithm by Breiman (1998) described in Section IV.D.4 and prune trees in the way introduced in the same section.

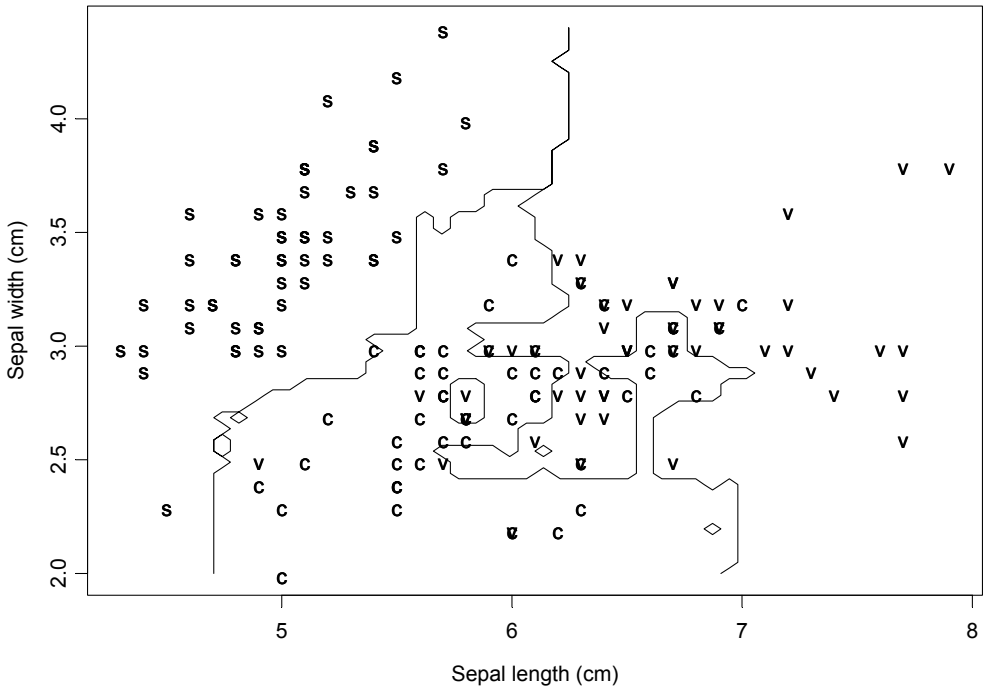

Figure I-9: Bagging Applied to the Iris Data

Bagging is applied to the Fisher's iris data in Figure I-1. This ensemble consists of 100 classification trees. Each tree is constructed using a bootstrap sample of the training set and not pruned. The final classifier is given by an unweighted vote of 100 trees. The misclassification rate on the training set is 0.12.

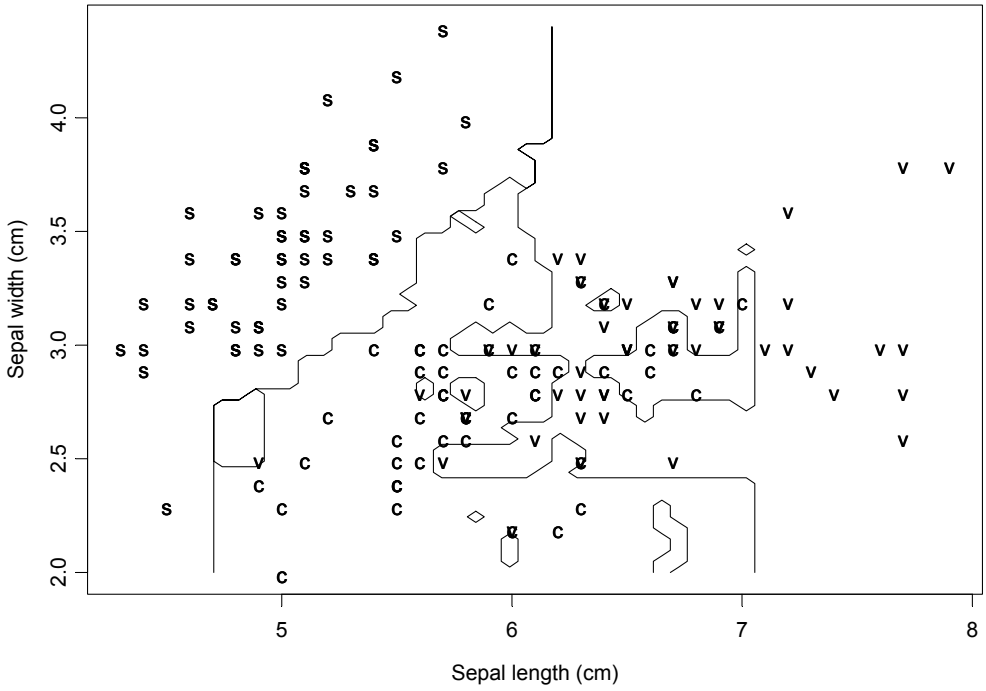

Figure I-10: Adaboost Applied to the Iris Data

Adaboost is applied to the iris data in Figure I-1. This ensemble consists of 100 (pruned) classification trees. The final classifier is given by a weighted vote of these trees. An algorithm by Breiman (1998) is used with the pruning method introduced in the same paper. The algorithm and pruning method are described in Section IV.D.4. The misclassification rate on the training set is 0.073.

At each iteration, bagging and arcing produce new data sets whose values all appear in the original training set. Breiman (2000) changes the data, perturbing the output variable by adding random noise. He proposes two ensemble methods (that he calls *output smearing* and *output flipping*) of this type and shows that these methods consistently perform better than bagging. In his remarks, he mentions that Adaboost averages a 5% lower misclassification rate than the output smearing.

Instead of changing the training sets, we can generate ensembles by injecting randomness into classification algorithms. Classification tree algorithms have been used in most research of this type partly because it is easy to inject randomness into tree algorithms. A tree is built by recursively splitting nodes starting from a node called the root node. The key decision in tree algorithms is choosing good splits for internal nodes. In many tree algorithms, a split at a node is chosen from all possible splits at that node so as to optimize a certain criterion. In recent research [Dietterich (1999), Breiman (2001), Cutler *et al.* (2001)], randomness is added to the split selection at each node.

Dietterich (1999) selects a split randomly from among the twenty best splits at each node. Even though the goal of his study is not to propose the method but to explore how well his randomized method works, his method performs at least as well as bagging for most data sets used in his study. He suggests some ways to refine his "crude" randomization method, which inspired our study at the beginning. Breiman (2001) randomly chooses a small set of input variables to split on at each node and uses the best split from among them. Cutler *et al.* (2001) propose a method called *PERT* (perfect random tree ensembles) which selects a split randomly at each node. Their results show that PERT is comparable to Breiman's method (Breiman, 2001) and Adaboost in its performance.

Many empirical studies show that ensemble methods can provide very accurate classifiers. Why ensemble methods work so well, however, has not been fully explained. Breiman (1998) introduces a bias and variance decomposition of error rates and shows that bagging reduces variance. He also applies the same analysis to arcing and suggests

that the bias-variance setting is not enough to explain why arcing works well, because arcing seems to reduce bias as well as variance.

Schapire, Freund, Bartlett, and Lee (1998) propose a new explanation using margins and provide an upper bound for error rate in terms of the margin distributions. A margin measures how confident the prediction is for an observation and takes the values between –1 and 1. A large margin indicates a confident prediction and a positive margin indicates a correct prediction. Schapire *et al.* (1998) suggest that the higher the margins, the lower the generalization error. Breiman (1997) shows that this is not true, by introducing a new arcing method which produces margins higher than those produced by Adaboost but which has larger error rates.

Breiman (2001) formulates a general framework for randomization methods with tree algorithms, methods which he terms *random forests*. He defines the "strength" and "correlation" of a random forest and provides an upper bound for the generalization error in terms of those two quantities. The strength of a random forest measures how well the individual classifiers classify new observations on average and the correlation captures how their predictions are related. Definitions of strength and correlation are given in Section II.C. This upper bound shows that one key for a good ensemble is to build less-correlated trees while maintaining moderate strength. The results of Cutler *et al.* (2001) indicate that the relatively weak individual trees can form ensembles that perform well if they have low correlation.

C. PROPOSED METHODS AND OUTLINE OF THIS DISSERTATION

Dietterich (1999) explores a randomized ensemble method for classification tree techniques which grows a tree by choosing a split uniformly at random from among the 20 best splits at each node. As one of the refinements of his "crude" randomization technique, he suggests choosing a split with probability proportional to a measure of goodness of a node, which is one of the ideas we investigate in our study. We also consider a stopping rule for the classification tree technique and combine it with our randomized splitting method. We empirically show that our method can produce

ensembles as accurate as Adaboost does. Adaboost is one of the best methods in many empirical studies on ensemble methods, for example, Dietterich (1999).

Breiman (2000) constructs ensembles by perturbing the outputs: on each iteration, add a random noise to the outputs of the training set and grow a tree using the perturbed data. We explore a perturbation method which is similar to Breiman's. The results show that on average our perturbation method performs as well as Adaboost on the data sets we use. Furthermore, we consider perturbing some of the input variables together with the outputs. The experiments on a few data sets suggest that perturbing both the inputs and the output, and combining a sufficiently large number of trees, can lower the error rate of this method.

Combining different ensemble methods sometimes improves accuracy. Breiman (2001) combines bagging with his random forest algorithms; he draws a bootstrap sample and grows a tree on the bootstrap sample using a random forest algorithm. We have explored combining our methods with sampling subsets of the training set instead of bootstrap sampling. This method, we call *sub-sampling*, draws a fraction of the training set without replacement on each iteration, and builds a tree by using an ensemble method on the drawn observations. The results of a few trials show that the error rate is smaller with sub-sampling than without sub-sampling for some combinations of the data sets and the ensemble methods.

One advantage of our proposed methods over Adaboost is that the individual trees forming an ensemble are grown independently of one another. These methods are obvious candidates for parallel processing, because the different trees can be developed simultaneously on different computers.

The outline of this dissertation is as follows: Chapter II gives a review of the classification tree technique. The classification tree is the only classification algorithm used in our study. The main terminology of classification trees is given first. Then we describe how to build a tree and how to decide the "right" size of the tree. Chapter II also gives more detail about the well-known ensemble methods: bagging, Adaboost, perturbation of outputs and randomization with tree algorithms.

In Chapter III, we propose two ensemble methods: one uses randomized splitting and the other uses perturbation. We apply both methods to several data sets together with other well-known ensemble methods such as bagging and Adaboost, and estimate error rates. Our methods give lower error rates than bagging for most data sets and Adaboost for some data sets. The method for estimation and the results are given in Chapter IV. We also find that the perturbation method can construct ensembles comparable in quality to Adaboost if we perturb some of the input variables together with the output and combine a sufficiently large number of trees. Strength and correlation show an interesting relationship which gives us a conjecture. These are discussed in Chapter V with other issues. Finally, Chapter VI gives the conclusions.

II. REVIEW OF CLASSIFICATION TREES AND ENSEMBLE METHODS

The classification tree technique was originally developed in the 1960s by social scientists (Morgan and Sonquist, 1963). The work by Breiman *et al.* (1984) brought the technique to the attention of statisticians by proposing new algorithms for constructing trees. A comprehensive survey of classification trees can be found in Ripley (1996, Chapter 7). The first section of this chapter gives a review of the classification tree technique. The terminology is given first, then how to grow a tree and determine its size.

Some well-known ensemble methods are briefly introduced in Section I.B. This chapter gives more detailed descriptions of these methods. Bagging (Breiman, 1996), Adaboost (Freund *et al.*, 1997), the random split selection methods with the classification tree technique by Diettrich (1999) and Cutler *et al.* (2001), random forests (Breiman, 2001) are described in Section B.

Although much empirical research has shown that ensemble methods can produce good classifiers, there have been few theoretical explanations as to why ensemble methods work well. This topic is described in Section C. The definitions of strength and correlation for random forests are also given in the same section.

A. CLASSIFICATION TREES

A classification tree is a simple model with a tree structure. Figure I-7 and Figure II-1 give examples. These trees are constructed with the Fisher's iris data (Fisher, 1936). Although the same data is used, these trees are not the same. This is because we use different numbers of covariates for constructing them. Two covariates (sepal length and sepal width) are used for constructing the tree in Figure I-7, and all four covariates are used in Figure II-1.

By convention, a node called the *root node* is displayed at the top, connected to the other nodes that are called the *child nodes* of the root node. The child nodes are similarly connected to their own child nodes and the process is repeated until we reach the *terminal nodes* (or the *leaves*), which have no child nodes. Nodes that are not

terminal nodes are called *non-terminal nodes*. A tree in which every non-terminal node has only two child nodes is called a binary tree. In the rest of this dissertation, we consider binary trees only.

In a binary tree, the root node is called node 1 by convention. The left and right child nodes of node t are called node $2t$ and $(2t + 1)$ respectively. Each terminal node has attached to it the class label of the node. Associated with every non-terminal node is *a split*, a rule by which we determine to which child node we should send the observations in that node. A split is a condition on an input variable, say variable j, and takes the forms $X_j < a_0$ when the variable j is numeric and $X_j \in A$ when j is categorical, where A is a set of values of variable j. A new observation, starting from the root node, traverses the tree by evaluating the splits, going left (by convention) if the answer is "yes" and right if the answer is "no," down to a terminal node. Then, the observation is assigned to the class attached to the terminal node.

Figure II-1 gives an example of a classification tree. This tree is constructed with the Iris data (Fisher, 1936) used also in the examples in Chapter I. The non-terminal nodes are indicated by ellipses, while the terminal nodes are indicated by rectangular boxes. The number under each node shows the number of that node. The labels inside nodes are the splits for the non-terminal nodes and the class labels for the terminal nodes. For instance, node 6 is a non-terminal node with the split "*Petal.L. < 4.95*" and node 2 is a terminal node with the class label *Setosa*. Suppose we have a new observation with *Petal.L.* = 2.6 and *Petal.W.* = 1.2. It would go right at the root node answering "no" to the split "*Petal. L. < 2.45*," left at node 3 answering "yes" to "*Petal.W. < 1.75*," left at node 6 answering "yes" to "*Petal.L. < 4.95*" and fall into node 12. Because the label of node 12 is *Versicolor*, this new observation would be classified as *Versicolor*.

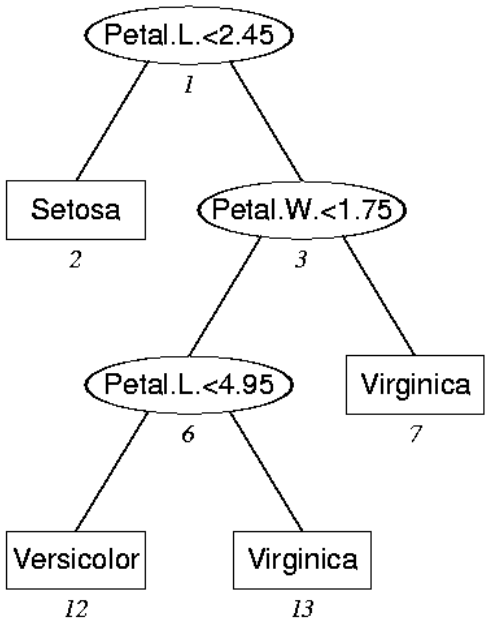

Figure II-1: Example of Classification Tree.

This classification tree is constructed using the Fisher's Iris data (Fisher, 1936). Because all of the four input variables are considered to train this tree, the tree is different from that of Figure I-7. The terminal nodes are represented by rectangular boxes while non-terminal nodes are represented by ellipses. The number under each node indicates the node number. Each terminal node has attached to it the class label of the node. An observation is assigned to the class with the class label of the terminal node it falls into. Associated with every non-terminal node is a split, a rule by which we determine to which child node we should send the observations in that node. Those observations in node *t* answering "yes" to the splitting rule go to the left child node (node 2*t*) and those answering "no" go to the right child node (node 2*t*+1). For example, if a new observation had *Petal.L.* = 2.6 and *Petal.W.* = 1.2, it would go right at the root node, go left at node 3, go left at node 6 and fall into node 12. Therefore, this observation would be classified as *Versicolor*.

When constructing a tree, there are two main questions: how to choose a split at each node and how large the tree should be (or, when to stop splitting). They are described in the following sections.

1. Growing a Tree

Most tree algorithms define a measure of impurity for a node and use it to determine a split at each node. In general, a node impurity measure is non-negative and takes value zero when a node is pure, that is, when all observations in the node have the same class label. There are several kinds of impurity measures. The commonly used measures are the entropy and the Gini index that were also used in Breiman *et al.* (1984). In a node t, suppose there are n_t observations of which n_{tk} are from class c_k. So, $n_t = n_{t1} + n_{t2} + \ldots + n_{tK}$. Let

$$p_{tk} = n_{tk} / n_t \text{ , } k = 1, \ldots, K.$$

Then the entropy at node t is defined by

$$i_e(t) = -\sum_{k=1}^{K} p_{tk} \log p_{tk} \text{ ,}$$

and the Gini index by

$$i_g(t) = \sum_{k \neq j} p_{tk} p_{tj} = \sum_{k=1}^{K} p_{tk}(1 - p_{tk}).$$

The tree algorithm implemented in S-Plus (Insightful, 2000) as the function `tree()` uses deviance as the impurity measure (Chambers and Hastie, 1992, Chapter 9); this is defined by

$$i_d(t) = -2\sum_{k=1}^{K} n_{tk} \log p_{tk} \text{ .}$$

With the entropy or the Gini index, the split which most reduces the average impurity is chosen at each node. Let $\Omega(t)$ be the set of all the possible splits at node t, and let l and r be the left and right child nodes of node t. For a particular split $\omega \in \Omega(t)$,

suppose among n_t observations in node t, n_l of them go to node l and n_r go to node r. Then, if we use the entropy, the split which maximizes

$$i_e(t) - (n_l/n_t)\, i_e(l) - (n_r/n_t)\, i_e(r)$$

is chosen as the split of node t. We choose a split at each node similarly when using the Gini index. When deviance is used, the split which most reduces the impurity, that is, the split which maximizes

$$i_d(t) - i_d(l) - i_d(r)$$

is chosen as the split of node t.

The set $\Omega(t)$ of all the possible splits at node t is created as the union of all the possible splits for each input variable. For a continuous variable, there are at most n_t different values among observations at node t, and hence, there are at most $(n_t - 1)$ ways to partition the observations into two disjoint groups by a split of the form $X_j < a_0$. If a variable is categorical with q categories, there are $2^{q-1} - 1$ possible partitions of the q values into two groups.

The tree building process starts at the root node. Given a training set of size n, all observations in the training set are in the root node at first. The best split is found by exhaustive search and the observations in the root node are partitioned into two sets by the chosen split; those cases that answer "yes" to the split go to the left child node and those cases that answer "no" go to the right child node. The effect is to divide the data space into two non-overlapping rectangular regions. The same process is repeated recursively in its child nodes.

2. Determining the Tree Size

When should we stop splitting? Or, how large should we grow the tree? There have been suggested several ways to stop splitting, such as to use a threshold or to use a criterion based on the statistical significance. These methods have not worked satisfactorily and hence pruning is widely used; a very large tree is initially built and then it is pruned back to the "right" size.

In many tree algorithms, if a node is pure enough, that is, if the impurity measure of the node is small enough, or if a node is small enough, that is, if the node does not contain enough observations to split, we stop splitting the node and declare it terminal. Once a node becomes terminal, it is labeled by the class label of the plurality class among the observations in that node. The building process ends when there are no remaining nodes to split. Now, we need to prune this large tree back to the "right" size.

One of the best-known procedures for tree pruning is the minimal cost-complexity pruning proposed by Breiman *et al.* (1984). Let $R_\lambda(\tau)$ be the cost-complexity measure of a tree τ:

$$R_\lambda(\tau) = R(\tau) + \lambda \text{ size}(\tau),$$

where $R(\tau)$ is the cost of τ, size(τ) is the number of terminal nodes of τ, and $\lambda \geq 0$ is the cost-complexity parameter. The procedure, given an initially built large tree τ_0, tries to minimize $R_\lambda(\tau)$ among trees τ that are subtrees of τ_0 with the same root node as τ_0. If λ is small, the penalty for having a large number of terminal nodes is small and optimal subtree will be large. As λ increases, the size of optimal subtree will decrease. There are some candidates for the cost $R(\tau)$. In the original development, Breiman *et al.* (1984) used the misclassification rate on the training set as $R(\tau)$. The impurity measure can also be used as the cost $R(\tau)$.

The cost-complexity procedure is based on the proposition proven in Breiman *et al.* (1984, p.68): For any value of λ, there exists a unique subtree $\tau(\lambda)$ such that

(1) $\tau(\lambda)$ minimizes the cost-complexity measure, i.e., $R_\lambda(\tau(\lambda)) \leq R_\lambda(\tau)$, and

(2) If there is a subtree τ with the same cost-complexity value as $\tau(\lambda)$, i.e., $R_\lambda(\tau(\lambda)) = R_\lambda(\tau)$, then $\tau(\lambda)$ is a subtree of τ with the same root node as τ,

where τ is a subtree of τ_0 with the same root node as τ_0 and thus $\tau(\lambda)$.

Using this fact, we can find an increasing sequence $\{\lambda_j\}$, that is, $\lambda_j < \lambda_{j+1}$, $j = 0, 1, 2, \ldots$, where $\lambda_0 = 0$, with the following property: for each λ_j, τ_j is the subtree (of τ_0) which minimizes the cost-complexity measure and τ_j is a subtree of τ_{j-1}. The "right-

sized" tree will then be one of the members of the subtrees τ_j. To choose the optimal tree from among these, we can use a validation set if we have one. Otherwise, Breiman *et al.* (1984) propose using cross-validation.

B. ENSEMBLE METHODS

Most of the ensemble methods can be applied to any classification algorithm but some methods are algorithm-specific. This section gives detailed algorithms of bagging, Adaboost, perturbation of outputs and random split selection with classification trees. Among these ensemble methods, bagging, Adaboost and perturbation of outputs are applicable to any classification algorithm. Randomized split selection method introduced in this section are specific to classification trees.

1. Bagging

Bagging is a simple method for constructing ensembles, developed by Breiman (1996). In bagging, at each iteration, a sample of size n is randomly drawn from the original training set of n observations with replacement. Each of these bootstrap samples is used to build a different classifier and the final classifier is obtained by an unweighted vote of predictions by the individual classifiers. This method works well for unstable classifiers, where unstable means that a small perturbation in the training set significantly changes the resulting classifier. Classification trees and neural networks are examples of unstable classifiers. The linear discriminant and nearest neighbors are very stable. It is generally believed that bagging is a variance-reduction technique [Breiman (1998), Bauer and Kohavi (1999), Freund and Shapire (1998)].

2. Adaboost

Adaboost was originally developed by Freund *et al.* (1997) and has been the best technique for many data sets in comparative studies of ensemble methods [Breiman (1998), Dietterich (1999), Bauer *et al.* (1999)]. Some studies point out that Adaboost is not robust when random noise is added to the outputs compared to bagging [Breiman (1998), Dietterich(1999)]. That is, Adaboost could perform worse than the original classifier when the data is noisy, while bagging usually performs at least as well as the original classifier.

Adaboost maintains a set of weights over the training set T consisting of n observations. Initially, these weights are equal. On the m^{th} iteration, $m = 1, 2, \ldots, M$, a sample of size n is drawn with replacement from T with probabilities $p^{(m)}(i)$, $i = 1, \ldots, n$, proportional to the current weights, and a classifier is built with the drawn sample. The weights are updated at the end of each iteration and used as the weights on the next iteration. This differs from bagging, where equal weights over the training set are used in all iterations. The weights in Adaboost are updated so that if the classifier built on the current iteration classifies an observation incorrectly, then the weight of that observation for the next iteration is increased by an amount corresponding to the weighted error rate of the classifier over T. The final classifier is given by a weighted vote of the individual classifiers. The detailed description of the algorithm is as follows (Freund *et al.*, 1997):

1. Let $p^{(1)}(i) = 1/n$ for $i = 1, \ldots, n$.
2. For $m = 1, \ldots, M$
   a. Using the probabilities $\{p^{(m)}(i)\}$, draw a sample $T^{(m)}$ of size n from T with replacement. Build a classifier $C^{(m)}$ on $T^{(m)}$.
   b. Classify each observation of T by $C^{(m)}$. Let $d(i) = 1$ if $C^{(m)}$ incorrectly classifies the ith observation of T, and zero otherwise.
   c. Define

$$\varepsilon_m = \sum_{i=1}^{n} p^{(m)}(i) d(i),$$

   the weighted error rate of $C^{(m)}$ on T. If $\varepsilon_m > 1/2$, then set $M = m - 1$ and stop the procedure.
   d. Set

$$\beta_m = \varepsilon_m / (1 - \varepsilon_m)$$

   and update the probabilities by

$$p^{(m+1)}(i) = p^{(m)}(i) \beta_m^{\,1-d(i)} \Big/ \sum_{i=1}^{n} p^{(k)}(i) \beta_m^{\,1-d(i)}, \; i = 1, \ldots, n.$$

The final prediction is given by a weighted vote of predictions by each classifier where the m^{th} classifier ($C^{(m)}$) has the weight $\log(1/\beta_m)$. In our study, we use the version

described in Breiman (1998), which differs only slightly from the original Adaboost. Breiman's modification is described in Section IV.D.4.

In Adaboost, if a classification algorithm can manage weights on the training set, the weights over the training set can be used in the algorithm directly instead of in the drawing of the samples. Breiman (1998) empirically shows that there are no significant differences in the results between reweighting and resampling while Friedman, Hastie and Tibsirani (2000) suggest that reweighting is slightly better than resampling.

3. Perturbation of Outputs

Breiman (2000) examines two randomization methods that construct ensembles by perturbing only the outputs: output smearing and output flipping. Output smearing formulates a K-class problem in terms of K multiple outputs, where the k^{th} output is 1 if the class label is c_k, and other outputs are zero. Then, on each iteration, Gaussian noise is added to each output independently of the others and a tree is built with the smeared data. The total sum of squares is used as the impurity measure for a node. A split is chosen based on minimizing the total sum of squares at each node. The prediction by a tree for a new observation is given as the class with the largest output. An unweighted vote of the individual trees is taken for prediction by an ensemble. In output flipping, with a given flipping rate, the class labels are altered to other class labels. An ensemble uses an unweighted vote of the individual trees for predicting new observations. Breiman (2000) suggests that it is important to keep the proportion of each class in the training set relatively invariant after the perturbation. Both methods perform better than bagging but not as well as Adaboost on the data sets used.

4. Randomization Methods with Tree Algorithms

To construct an accurate ensemble, it is necessary that individual classifiers be more accurate than a random guess and that different classifiers make mistakes on different observations. Recently, some studies have proposed new methods to construct ensembles consisting of less-correlated classifiers by injecting randomness (Dietterich, 2000). As before, some randomizing methods can be applied to any classification

algorithm and others are algorithm-specific. The random split selection methods introduced in this section are specific to classification tree algorithms.

Each tree algorithm defines its own measure of impurity (or purity) for a node. In most tree algorithms, a tree is constructed by choosing a split so as to optimize the impurity measure by exhaustive search at each node. The random split selection methods described in this section select a split from a distribution of splits instead of finding a locally optimal split at each node. These methods differ in how to choose a split at each node. Trees constructed with random split selection are typically weaker than those constructed with the usual algorithms, but the randomness yields trees less correlated with one another.

Random Split Selection: Dietterich (1999) builds trees by randomly selecting a split from the twenty best splits at each node. The final classifier is constructed by using an unweighted vote of the individual trees. The performance of this method is compared with bagging and Adaboost. For most of the data sets he uses, the randomization method is more accurate than bagging but less accurate than Adaboost. The study also shows that, for noisy data, the randomization method is more robust than Adaboost but less so than bagging.

Cutler *et al.* (2001) introduce another randomized method they call PERT (perfect random tree ensembles). In PERT, a split is selected randomly at each node in the following way: Two observations belonging to different classes are selected from among all observations in the node. Then, a variable on which to split is selected randomly. The splitting value for the selected variable is produced by random interpolation of the values of the two observations on the variable. Only continuous input variables are considered in the study. For the final classifier, an unweighted vote is used. The results show that PERT can perform comparably to Adaboost and the random forests that are introduced next.

Random Forests: Breiman (2001) gives a general framework to randomization methods with trees, a framework he calls random forests. He defines a random forest as

"a classifier consisting of tree-structured classifiers $\{h(x, \Theta_k), k = 1,\ldots\}$ where the $\{\Theta_k\}$ are independently identically distributed random vectors and each tree casts a unit vote for the most popular class at input x." The random split selection methods introduced above are examples of random forests. For instance, in Dietterich's method, Θ_k consists of a number of independent random integers between 1 and 20. Bagging is also a random forest in which Θ_k is a random sampling of size n from $\{1, 2, \ldots, n\}$ with replacement, where n is the size of the training set.

Two algorithms for random forests denoted by *Forest-RI* and *Forest-RC* are proposed by Breiman (2001). In Forest-RI, at each node, F of the input variables are selected at random on which to split and the best split from among the selected variables becomes the split at that node. The value of F is fixed at the beginning of the procedure. Breiman tried two values of F; $F = 1$ and $[\log_2(p) + 1]$, where p is the number of input variables and $[x]$ = the largest integer less than x. The results suggest that random forests are not very sensitive to the value of F. The average absolute difference between the error rates using these two values of F is less than 1%. In Forest-RC, L of the input variables are selected randomly at each node and combined with coefficients generated from the uniform distribution on [-1, 1]. A number F of linear combinations is generated, and the best split from among these linear combinations is chosen. The values of L and F are fixed at the beginning of the procedure. In Breiman's study, $L = 3$ and $F = 2, 8$ are used. He reports that except for the large data sets, $F = 2$ performs better. Overall, both Forest-RI and Forest-RC show the comparable performance to Adaboost.

C. WHY ENSEMBLES WORK

Although research continues into the question of why ensemble methods work so well, the explanation is as yet incomplete. Breiman (1998) defines the bias and variance decomposition of classification error. His results show both bagging and arcing reduce the variance, which he considers to be their major contribution to increased accuracy. In his experiments, arcing reduces not only the variance but also the bias. Breiman suggests that the bias-variance setting seems to be suitable to explain the effect of bagging but not enough for arcing. Freund *et al.* (1998) reach the same conclusion by showing that

boosting (called arcing by Breiman) can reduce both bias and variance in an example using stumps. A stump is a tree with only one split, and thus tends to be highly biased but has low variance. Freund *et al.* (1998) also give an example in which boosting increases the variance but decreases the bias to reduce the final error rate. By these examples, they conclude that variance-reduction alone cannot completely explain the performance of arcing.

Freund *et al.* (1998) introduce an alternative explanation in terms of margins. The margin of an observation is defined as the number of votes for the correct class minus the largest number of votes assigned to any incorrect class. They find that boosting is good at increasing the margins of the training observations and suggest that this causes a decrease in the generalization error (that is, the expected error rate for a new observation) even after boosting achieves zero training error. Breiman (1997) gives a counter-example by developing a new arcing method, *arc-gv*. He shows that arc-gv consistently produces larger margins than Adaboost but results in higher test set error rate. This example suggests that the margin explanation does not fully answer the question of why arcing works so well.

Breiman (2001) built a new framework for randomization methods in terms of strength and correlation, and provides an upper bound for the generalization error of a random forest in terms of them. The bound shows that generalization error is lower as correlation is smaller and strength is higher. The empirical results by Cutler *et al.* (2001) suggest that PERT works well because it can construct trees that are almost uncorrelated. Definitions of strength and correlation are described below.

Strength and Correlation: Breiman (2001) defines strength and correlation for a random forest, and provides an upper bound in terms of them. We introduce his definitions and the upper bound here. The definitions of strength and correlation do not depend on the use of classification tree algorithms, and so we use more general term "classifiers" instead of "trees" in this section. We follow Breiman's notation except that we represent a random vector in a capital-case letter like Θ and its representation in a

lower-case letter like θ. Assume that a training set T is drawn randomly from the joint distribution of the random vector (X, Y) and $\theta_1, \theta_2, \ldots$ are drawn randomly from the distribution of the random vector Θ. For example, in bagging, Θ consists of all the possible random samples of size n from $\{1, \ldots, n\}$ with replacement, assuming that the size of the training set is n. There are n^n elements in Θ, and θ_m, $m = 1, 2, \ldots$ are randomly drawn from Θ independently of one another. When $n = 10$ (although the training set of size 10 is not realistic), θ_m is a vector of 10 components such as $(7, 2, 8, 10, 2, 1, 6, 10, 8, 1)^T$. A classifier constructed using T and θ_m is denoted by $h(\cdot, \theta_m)$, and its prediction at $\boldsymbol{x}$ by $h(\boldsymbol{x}, \theta_m)$.

In real applications, we construct only a finite number of classifiers, $h(\cdot, \theta_1)$, $h(\cdot, \theta_2)$, …, $h(\cdot, \theta_M)$. Suppose there are K classes labeled by $c_1, c_2, \ldots, c_K$. Prediction at input $X = \boldsymbol{x}$ is given by

$$\arg\max_{c \in \{c_1, \cdots, c_K\}} \frac{1}{M} \sum_{m=1}^{M} I\left[h(\boldsymbol{x}, \theta_m) = c\right]$$

which is the class with the largest number of votes from the M trees. Here, $I(\cdot)$ is the indicator function. For an input $X = \boldsymbol{x}$ with true class $Y = y$, a forest classifies it incorrectly if another class gets more votes than the true class, i.e.,

$$\frac{1}{M} \sum_{m=1}^{M} I\left[h(\boldsymbol{x}, \theta_m) = y\right] - \max_{k \in \{k: c_k \neq y\}} \frac{1}{M} \sum_{m=1}^{M} I\left[h(\boldsymbol{x}, \theta_m) = c_k\right] < 0 .$$

The generalization error PE^* for the forest is the probability that the forest will misclassify a new observation. PE^* is obtained by taking the probability of this inequality over X, Y, that is,

$$PE^* = P_{X,Y}\left\{ \frac{1}{M} \sum_{m=1}^{M} I\left[h(X, \theta_m) = Y\right] - \max_{k \in \{k: c_k \neq y\}} \frac{1}{M} \sum_{m=1}^{M} I\left[h(X, \theta_m) = c_k\right] < 0 \right\}.$$

Breiman (Theorem 1.2, Breiman, 2001) shows that as $M \to \infty$, PE^* converges almost surely to

$$P_{X,Y}\left\{P_{\Theta}\left[h(X,\Theta)=Y\right]-\max_{k\in\{k:c_k\neq y\}}P_{\Theta}\left[h(X,\Theta)=c_k\right]<0\right\}. \tag{2.1}$$

He then gives an upper bound for the quantity (2.1) in terms of the strength and the correlation of an ensemble defined below. (Note that equation (4) in Breiman (2001) incorrectly refers to bound being on PE^*.)

We first introduce the definitions of strength and correlation and then give the upper bound. Let $c(X, Y)$ be the class label with the largest probability among the classes other than the true class over Θ. $c(X, Y)$ is given by

$$c(X,Y)=\arg\max_{c\in\{c_k|c_k\neq Y\}}P_{\Theta}\left[h(X,\Theta)=c\right].$$

The margin function is the difference in probability of classifying X correctly over Θ and probability of classifying X as $c(X, Y)$ over Θ, i.e., defined by

$$mr(X,Y)=P_{\Theta}\left[h(X,\Theta)=Y\right]-P_{\Theta}\left[h(X,\Theta)=c(X,Y)\right].$$

The strength s of the forest is defined as the mean of the margin over X, Y, i.e.,

$$s = E_{X,Y}[mr(X, Y)].$$

By its definition, $-1 \le s \le 1$.

Next, define the raw margin function as

$$\begin{aligned} rmg(\Theta,X,Y) &= I\left[h(X,\Theta)=Y\right]-I\left[h(X,\Theta)=c(X,Y)\right] \\ &= \begin{cases} 1 & \text{if classify correctly at } X \\ -1 & \text{if classify as } c(X,Y) \\ 0 & \text{otherwise} \end{cases}. \end{aligned}$$

Using the raw margin function, the margin function can also be expressed as

$$mr(X,Y)=E_{\Theta}\left[rmg(\Theta,X,Y)\right].$$

The correlation is defined by

$$\bar{\rho}=\frac{E_{\Theta,\Theta'}\left[\operatorname{cov}_{X,Y}\left(rmg(\Theta),rmg(\Theta')\right)\right]}{E_{\Theta,\Theta'}\left[sd(\Theta)sd(\Theta')\right]},$$

where Θ, Θ' are independent with the same distribution, $\mathrm{cov}_{X,Y}(rmg(\Theta), rmg(\Theta'))$ is the covariance between $rmg(\Theta, X, Y)$ and $rmg(\Theta', X, Y)$ holding Θ, Θ' fixed, and $sd(\Theta)$ is the standard deviation of $rmg(\Theta, X, Y)$ holding Θ fixed.

Assuming $s \geq 0$, Breiman (2001) gives an upper bound on the quantity (2.1) in terms of strength s and correlation $\bar{\rho}$:

$$P_{X,Y}\left\{P_\Theta[h(X,\Theta)=Y]-\max_{k\in\{k:c_k\neq y\}}P_\Theta[h(X,\Theta)=c_k]<0\right\}\leq \bar{\rho}(1-s^2)/s^2.$$

This upper bound implies that the higher the strength and the lower the correlation, the lower the bound.

Breiman points out that the measure of strength defined above depends on the forest since the forest determines $c(X, Y)$. He gives another definition of the strength which does not depend on the forest and provides an upper bound in terms of this alternative strength. He suggests this new bound would be interesting in a many-class problem.

Ensembles produced by using Adaboost are not random forests since the individual classifiers are not constructed with independently identically distributed random vectors. Adaboost draws the training sample on the $(m + 1)^{th}$ iteration depending on the accuracy of the classifier on the m^{th} iteration. It also forms the final classifier by a weighted vote of the individual classifiers instead of an unweighted vote. Hence, strictly speaking, it is not appropriate to apply the theory of strength and correlation to Adaboost. However, we estimate strength and correlation for Adaboost as well as the other methods for comparison. Breiman (2001) conjectures that Adaboost produces a random forests in its later stage with the support of empirical evidence.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROPOSED METHODS

We propose two randomized ensemble methods here. One uses random split selection and the other uses perturbation. The random split selection can be applied only to classification tree algorithms, while the perturbation method can be applied to any classification algorithm.

A. RANDOMIZED SPLITTING AND PERMUTATION STOPPING

In this section, we consider a new random split selection method similar to that of Dietterich (1999) but which incorporates a stopping rule based on permutation. We do not worry about pruning of trees very much, since (1) pruning does not significantly improve the performance for many data sets in Dietterich (1999), (2) the studies of randomization [Breiman (2001), Cutler *et al.* (2001)] use unpruned trees, and (3) unpruned trees should perform better for bagging because bagging reduces variance and unpruned trees tend to have higher variance but less bias than pruned trees.

Randomized Splitting: Dietterich (1999) chooses a split at each node randomly and equi-probably from the twenty best splits instead of choosing a single best split. His method performs better than bagging and is more robust to noise than Adaboost. We take a similar approach.

In our method, we choose a split randomly from among all possible splits at each node with the probabilities proportional to the decrease in impurity measure. We stop splitting a node when the node is pure or small enough. Here, a node is called "pure" when all the observations falling into the node have the same class label. When there are not enough observations to split the node, the node is said to be "small."

We also consider a variant of this method similar in spirit to Dietterich's approach of choosing from the 20 best splits. A split is chosen from among some "good" splits with the probabilities proportional to the decrease in impurity. The "good" splits are the splits that reduce the impurity of the node within $(100r)\%$ from the best split. We try

several values of r (= 0.05, 0.1, 0.2, 0.5) with a few data sets. This method does not perform as well as the original one with $r = 1.0$, and hence is not considered further.

Permutation Stopping: In many tree algorithms, a node becomes terminal if the node is pure enough or small enough. Instead of using this general rule, we stop splitting a node and declare it terminal if the node is pure or if no possible splits in the node are "significant," i.e., no splits reduce the impurity by more than a threshold which is obtained using permutation at each node. More precisely, at each node, we permute the outputs of the observations in the node and find the maximum decrease in impurity of all the possible splits with the permuted output. This is repeated several times and the largest value among the maximum decreases in impurity on each iteration is computed. Call it D_{max}. If no split obtained by the original output in the node decreases impurity by more than D_{max}, then the node becomes a terminal node.

Combination of Randomized Splitting and Permutation Stopping: Our ensemble method is obtained by combining randomized splitting and permutation stopping in the following way: Choose Q, the number of permutations. At each node, if a node is pure, then the node becomes terminal. Otherwise, find all the possible splits at that node and record the maximum decrease in impurity D among them. Then, permute the output variable (of the data in that node) Q times, each time recording the maximum decrease in impurity D_q of all the possible splits using the permuted data. Let $D_{max} = \max_{q = 1,\ldots,Q}\{ D_q \}$. If $D \leq D_{max}$, then the node becomes terminal. If not, i.e., if there is at least one split (from the non-permuted data) which decreases impurity by more than D_{max}, then choose a split randomly from among all the splits that reduce the impurity by more than D_{max} with the probability proportional to the decrease in impurity. The detailed description of the algorithm is as follows:

1. Set Q, the number of permutations at each node. All observations in the training set T are in the root node (node 1). Start the process from node 1.

2. At node t, if all the observations have the same class label, declare it terminal. Otherwise, find all possible splits $\Omega(t) = \{\omega_1, \omega_2, \ldots, \omega_{N(t)}\}$ and the corresponding decreases in impurity $\Delta i_1, \Delta i_2, \ldots, \Delta i_{N(t)}$. Set

$$D = \max_{d=1,\ldots,N(t)} \Delta i_d .$$

3. For $q = 1, \ldots, Q$,
   a. Permute the output variable of the observations in the current node.
   b. Find the maximum decrease in impurity D_q of all the possible splits with the permuted data.
4. Set

$$D_{\max} = \max_{p=1,\ldots,Q} D_q .$$

If $D \le D_{max}$, then declare the current node to be terminal. Otherwise, let

$$\Omega(t)_+ = \{\omega_d \mid \Delta i_d > D_{max}\},$$

and set

$$p_d = \frac{\Delta i_d}{\sum_{\omega_d \in \Omega(t)_+} \Delta i_d} \text{ for } d \in \{d \mid \omega_d \in \Omega(t)_+\}.$$

Then choose a split ω from $\Omega(t)_+$ using the probabilities p_d. With the chosen split ω, split the observations in the current node into its left child node l and the right child node r.

5. If node l is small enough, then declare it terminal and go to step 6. Otherwise, repeat 2 through 4 at node l.
6. If node r is small enough, then declare it terminal. Otherwise repeat 2 through 4 at node r.

The process ends when there are no nodes to split. The class label of each terminal node is determined by plurality vote among the observations in the training set falling into that node.

We use this method with five different values of Q; 0 (no permutation), 1, 5, 10 and 20, and compare the results with other ensemble methods such as bagging, random

forests and Adaboost. The results given in Section IV.E show that this method is comparable to other ensemble methods.

B. PERTURBATION

The perturbation method we use in our study is similar to the output flipping introduced in Breiman (2000). We perturb the outputs of the data as follows:

Fix the perturbation rate r, $0 \le r \le 1$. As before, Y takes class values $\{c_1,\ldots, c_K\}$. Let p_k be the fraction of classes $c_k = 1,\ldots, K$ in the training set. Suppose $Y = y$. Change its value with probability r to one of the other values using the probability proportional to the fraction of the values in the training set. That is,

$$\tilde{y} = \begin{cases} y & \text{with probability} \quad 1-r \\ c_k & \text{with probability} \quad r\dfrac{p_k}{\sum_{k\in\{k|c_k\neq y\}} p_k} \text{ for } k \in \{k \mid c_k \neq y\}, \end{cases}$$

where $\tilde{y}$ is the value of Y after perturbation.

We use perturbation rates ranging from 0.1 to 0.5 in increments of 0.1. This method performs better than bagging for most of the data sets we use. The perturbation rate which gives the lowest misclassification rate varies depending on the data. Similar results are also reported in Breiman (2000).

We also perturb the input variables for a few data sets and find that the results are as good as Adaboost. The input variables in the data sets we use are all continuous and perturbed as follows:

For a continuous variable V, compute the sample standard deviation sd first. Let $V = v$. The new value of V is given by

$$\tilde{v} = v + r \cdot sd \cdot z,$$

where z is generated from the standard normal distribution.

IV. SIMULATIONS AND RESULTS

We apply the methods proposed in Chapter III to 13 data sets. To compare the performance of our methods to other ensemble methods, bagging, Adaboost, and random forests are also run on the same data sets. Most of the data sets are chosen because they have been used in other research on ensemble methods. For our computational tool, we use S-Plus (Insightful, 2000), which makes our computation very slow. Details about the simulation and the results are given in the following sections: Section A gives brief information about the data sets. Computational tools are introduced in Section B. We use test set estimation for data sets with the test set and cross-validation for those with no test sets. Methods for estimating error rates and other estimates are given in Section C. Section D describes how we implement the ensemble methods in S-Plus precisely. The results are shown in Section E.

A. DATA SETS

The thirteen data sets used in our study are summarized in Table IV-1. For each data set, the size of the training set (and the test set if available), the number and type of the input variables, and the number of classes are shown in the table. These data sets are chosen mainly because they have been used in other studies on ensemble methods or classification. Among these data sets, the Image, Vowel and Waves data sets have the separate test sets. All but the Waves data sets were downloaded from the UC Irvine Data Repository (Blake and Merz (1998)). The Waves data were generated using the rule described in Breiman *et al.* (1984). The data are different from each other in their size, and number or type of the input variables. The input variables are continuous in most data sets. Binary inputs are treated as continuous in our computation. The Biopsy data originally contained 699 observations from which 16 observations with missing data were removed. The other data have no missing values. Appendix A gives more detailed descriptions of the data.

Data	Train size	Test size	Inputs	Type of inputs	Classes
Biopsy	683	-	9	Continuous	2
Diabetes	768	-	8	Continuous	2
Ecoli	336	-	7	Continuous	8
German credit	1000	-	24	Continuous	2
Glass	214	-	9	Continuous	6
Ionosphere	351	-	34	Continuous	2
Liver	345	-	6	Continuous	2
Sonar	208	-	60	Continuous	2
Vehicle	846	-	18	Continuous	4
Votes	435	-	16	Categorical	2
Image	210	2100	19	Continuous	7
Vowel	528	462	11	Continuous	11
Waves	300	1500	21	Continuous	3

Table IV-1: Summary of Data.

For each data, size of the training set and the test set (if available), number and type of input variables, and number of classes are shown. Image, Vowel and Waves have separate test sets. Binary variables were treated as continuous variables. The original set of Biopsy contained 699 observations, but 16 cases with missing values were removed.

B. PROGRAMMING LANGUAGE

For our computational tool, we use the S-Plus (Insightful, 2000) because we are familiar with the software and we can use its built-in functions to write our own functions. Several functions based on the S-Plus built-in function `tree()` were written for all ensemble methods except for random forests. The function `tree()` employs deviance that is, $-2 \times$ (the multinomial log-likelihood), as an impurity measure and, at each node, chooses the split with the largest deviance reduction as the split at that node. Other functions such as `burl.tree()`, `edit.tree()` and `snip.tree()` are also heavily used in our functions. The software for the random forests was downloaded from Breiman's web site [http://www.stat.berkeley.edu/users/breiman/] and modified so that it could be run from S-plus.

C. ESTIMATION OF ERROR

Error rates are estimated to compare the performance of our proposed method to other well-known ensemble methods. For data sets with test sets, the test sets are used to estimate the error rates. We construct classifiers with the training set and estimate the error rate on the test set. Among the 13 data sets, the Image, Vowel and Waves are analyzed this way. When a test set is not available, five-fold cross-validation is used. The summary of error rates estimation is shown in the following table:

Data	Size of ensembles	Error estimation
Biopsy Diabetes Ecoli German Glass Ionosphere Liver Sonar Vehicle Votes	100	5-fold cross-validation
Image Vowel Waves	100	Test set

Table IV-2: Summary of Estimation.

For each combination of data and ensemble method, an ensemble of 100 trees is built. When a test set is available, the test set is used to estimate the error rate. For data sets with no test sets, five-fold cross-validation is used.

In the case where the test set is available, for each combination of data and ensemble method, an ensemble of 100 trees is built with the training set. The constructed ensemble is used to predict the observation in the test set and compute the test set error. The test set error is given by the number of observations in the test set misclassified by the ensemble divided by n_V, where n_V is the number of observations in the test set. The detailed algorithm is as follows: Assume there are K classes named $c_1, c_2, \ldots, c_K$. For each ensemble method,

0. (Initialization) Given the training set T of size n_T and the test set V of size n_V, determine M, the size of the ensemble, i.e., the number of trees in the ensemble. (M is 100 in our study.)

1. For $m = 1, \ldots, M$,
   a. (Construction) Construct tree h_m with T by using the chosen ensemble method.
   b. (Prediction by individual trees) Get the prediction on V by tree h_m. Denote the prediction for the i^{th} observation in V by tree h_m as $\hat{y}_{im}$.

2. (Prediction by ensemble) Predict the class of the i^{th} observation by the ensemble by

$$\hat{y}_i = \arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} I\left(\hat{y}_{im} = c\right) \text{ (for unweighted vote),}$$

   or

$$\hat{y}_i = \arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} w_m I\left(\hat{y}_{im} = c\right) \text{ (for weighted vote),}$$

   where w_m is the weight for tree h_m.

3. (Estimation) Compute the test set error rate by

$$\frac{1}{n_V} \sum_{i=1}^{n_V} I\left(\hat{y}_i \neq y_i\right),$$

   where y_i is the true class for the i^{th} observation in V.

When five-fold cross-validation is used, the training set is randomly divided into five disjoint subsets, with each containing nearly the same number of observations. One of them is set aside as a validation set and an ensemble consisting of 100 trees is constructed on the remaining four sets. Then the error rate of the ensemble is computed on the validation set. Each set is used in turn as a validation set for an ensemble constructed on the other four sets. The final error rate is obtained by averaging the five individual error rates. More precisely:

0. (Initialization) Given the training set T. Determine the size of each ensemble, M.

1. Divide T into five disjoint subsets T_1, …, T_5, with each containing about 1/5 of the observations in T. Let n_j be the size of T_j, $j = 1, 2, \ldots, 5$.

2. For $j = 1, \ldots, 5$:

   Set aside T_j as the validation set. Let $U_j = T - T_j$.

   a. For $m = 1, \ldots, M$

      i. (Construction) Construct tree h_m^j with U_j by using the chosen ensemble method.

      ii. (Prediction by individual trees) Predict the observations in T_j by tree h_m^j. Denote the prediction for the i^{th} observation in T_j by $\hat{y}_{im}^j$.

   b. (Prediction by ensemble) Predict the class of the i^{th} observation in T_j by

$$\hat{y}_i^j = \arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} I\left(\hat{y}_{im}^j = c\right) \text{ (for unweighted vote),}$$

   or

$$\hat{y}_i^j = \arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} w_m^j I\left(\hat{y}_{im}^j = c\right) \text{ (for weighted vote),}$$

   where w_m^j is the weight for tree h_m^j.

   c. (Estimation for the j^{th} iteration) Compute the test set error rate of the j^{th} iteration by

$$e_j = \frac{1}{n_j} \sum_{i=1}^{n_j} I\left(\hat{y}_i^j \neq y_i^j\right),$$

   where y_i^j is the true class of the i^{th} observation in T_j and n_j is the size of T_j.

3. (Estimation) The cross-validated estimate of error rate is given by

$$e = \frac{1}{5} \sum_{j=1}^{5} e_j \ .$$

In addition to error rate, the strength, correlation, average size of the individual trees and average error rate of the individual trees are computed for each ensemble. Strength and correlation are defined in Section II.C. To compute strength and correlation, we follow Breiman (2001) except one equation (equation (A2), Appendix II in Breiman), which is discussed in Appendix B. Given an ensemble of M trees and a test set consisting of n_V observations, the estimates are obtained as follows: For the i^{th} observation in the test set, the margin is given by

s_i = (number of trees that classify observation i correctly) / M

− (number of trees that classify observation i as class $c(i)$) / M,

where $c(i)$ is the class label other than the true class with the maximum votes among M trees, i.e., $c(i) = \underset{c \in \{c_k | c_k \neq c_l\}}{\arg\max} \sum_{m=1}^{M} I(\hat{y}_{im} = c)$, assuming that the true class of the i^{th} observation is c_l. If s_i is positive, then the ensemble classifies the i^{th} observation correctly. The estimated strength of the ensemble is given by the average of s_i over the test set, that is,

$$\hat{s} = \frac{1}{n_V} \sum_{i=1}^{n_T} s_i .$$

To compute the correlation of the ensemble, for tree m, let

p_1 = (number of observations in the test set classified correctly by tree m) / n_V,

$$p_2 = \textstyle\sum_i p_2(i) / n_V,$$

where $p_2(i) = 1$ if observation i is classified as $c(i)$ by tree m, and 0 otherwise. Set

$$sd(m) = [p_1 + p_2 - (p_1 - p_2)^2]^{1/2}.$$

The estimated correlation is given by

$$\hat{\bar{\rho}} = \frac{\text{var}(s)}{\sum_{m=1}^{M} sd(m)},$$

where $\text{var}(s) = \frac{1}{n_T} \sum_{i=1}^{n_T} s_i^{\,2} - s^2$.

The average size of the ensemble is obtained by taking average number of terminal nodes of the individual trees in the ensemble. The average error rate is given by averaging the error rate for each individual tree in the ensemble. The error rate for each tree is computed as the number of observations in the test set misclassified by the tree divided by n_V. When five-fold cross-validation is used, the estimates on each iteration are computed on the hold-out set. The final estimates are obtained by taking the averages of five iterations.

D. IMPLEMENTATION OF ALGORITHMS IN S-PLUS

Some well-known ensemble methods such as bagging and Adaboost are introduced in Section II.B, and our proposed methods are given in Chapter III. We implement these algorithms in S-Plus and apply them to the data sets in Section IV.A. This section gives the detailed algorithms of the ensemble methods we use and the parameter values for each method. Throughout this section, let $T = \{(y_i, \boldsymbol{x_i})\}$ be the training set with n observations from K classes labeled by $c_1, c_2, \ldots, c_K$, where y_i takes one of the values from $\{c_1, c_2, \ldots, c_K\}$ and $\boldsymbol{x_i} = (x_{i1}, x_{i2}, \ldots, x_{ip})$ is a row vector of p covariates. Let M be the number of trees in an ensemble. When we refer to the output vector $(y_1, \ldots, y_n)^T$, we denote it by $\boldsymbol{y}$. Likewise, we denote the vector of covariate j by X_j, i.e., $X_j = (x_{1j}, x_{2j}, \ldots, x_{nj})^T, j = 1, \ldots, p$. We represent a set of n input vectors as $\boldsymbol{X}$, a $n \times p$ matrix. Sometimes we write $T = (\boldsymbol{y}, \boldsymbol{X})$.

1. Randomized Splitting and Permutation Splitting

In Chapter III, we propose a new randomized splitting method with permutation stopping rule. This method randomly chooses a split among from all the possible splits at each node with the probabilities proportional to the decrease in deviance. When permutation stopping is not used, a node becomes a terminal node when (1) its deviance is less than 1% of the deviance of its parent node, or (2) there are fewer than four observations in the node. When the permutation stopping rule is used, permutation of the output variable at each node gives a threshold value by which we decide (1) if we make that node terminal or (2), if we continue to split the node, which splits, from all the possible splits in the node, should be included as candidates from which a split is chosen.

A node also becomes terminal when the maximum decrease in deviance at that node is less than the threshold value given by permuting the output variable. The procedure for randomized splitting and permutation stopping is as follows:

0. (Initialization) Given T, set the number of permutation at each node.
1. (Construction) For $m = 1, \ldots, M$:

   Construct tree h_m with T using the algorithm in Section III.A.
2. (Final classifier) The final classifier is given by an unweighted vote of the M trees, that is, a new observation $\boldsymbol{x}$ is classified as

$$\arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} I[h_m(\boldsymbol{x}) = c].$$

We wrote S-Plus functions to train a tree at step (1). The function repeatedly uses the built-in functions `tree()`, `burl.tree()`, and `edit.tree()` to implement the algorithms, which makes the process very slow. For example, for the German data set, it takes about three minute on average to grow a tree using the one permutation stopping rule. With no permutations, the time required to grow a tree is more than twice the time required for with one permutation for most data sets.

2. Perturbation

In the perturbation approach, each tree in the ensemble is constructed by perturbing the output variables with the given rate. A tree is then produced using the perturbed data. The perturbation scheme is given in Section III.B. Values raging from 0.1 to 0.5 were used by increments of 0.1 for perturbing the output variables. Some input variables were also perturbed in the Vowel and Waves data. The detailed steps for perturbation methods are as follows:

0. (Initialization) Given T. Set the parameters for perturbation: r_Y = perturbation rate for the output, m_X = number of the input variables to be perturbed, and r_X = perturbation rate for the inputs.

1. (Construction) For $m = 1, \ldots, M$:
   a. Perturb $\boldsymbol{y}$ with the rate r_Y, call it $\boldsymbol{y_p}$. Randomly choose m_X input covariates to be perturbed and perturb each with rate r_X. Call the inputs after perturbation $\boldsymbol{X_p}$.
   b. Construct tree h_m using $T_p = (\boldsymbol{y_p}, \boldsymbol{X_p})$.
2. (Final classifier) The final classifier is given by an unweighted vote of the M trees, i.e., a new observation $\boldsymbol{x}$ is classified as

$$\arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} I[h_m(\boldsymbol{x}) = c].$$

In step (1-b), the S-Plus function `tree()` is used to construct trees using its default setting with which a node becomes terminal if (1) its deviance is less than 1% of the root node deviance or (2) it contains fewer than 10 observations. Trees are not pruned.

3. Bagging

In bagging, on each iteration, a sample of size n is randomly drawn from the training set T with replacement and a tree is constructed with the drawn sample. The final classifier is given by an unweighted vote of the individual trees in the ensemble. Bagging is done with the following procedure:

0. (Initialization) Start with the training set T.
1. (Construction) For $m = 1, \ldots, M$:
   a. Draw a bootstrap sample $B^{(m)}$, a sample of size n randomly drawn from T with replacement.
   b. Construct tree h_m with $B^{(m)}$.
2. (Final classifier) The final classifier is given by an unweighted vote of the M trees, i.e., the class of a new observation $\boldsymbol{x}$ is given by

$$\arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} I[h_m(\boldsymbol{x}) = c].$$

As in perturbation, in step (1-b), trees are constructed using the function `tree()` with the setting in which a node becomes terminal if (1) its deviance is less than 1% of the root node deviance or (2) it contains fewer than four observations. Trees are not pruned.

4. Adaboost

Instead of using Adaboost.M1 by Freund *et al.* (1997) introduced in Section II.B.2, an algorithm called *arc-fs* by Breiman (1998) is used. The algorithm arc-fs is essentially the same as Adaboost.M1 except for the following two minor points: (1) (At step (2-c) in the algorithm in Section II.B.2) if the weighted error rate ε_m becomes greater than 1/2, then set all $\{p(i)\}$ equal and restart; (2) If $\varepsilon_m = 0$, then again set all $\{p(i)\}$ equal and restart.

For Adaboost, ensembles of unpruned and pruned trees are constructed for comparison. To prune a tree, on each iteration, two samples each of the same size n are drawn independently of one another from the training set T using the same weights on T. One of them is used to construct an initial large tree. The other is used to find the right size for the tree using the cost-complexity pruning procedure. The first tree is then pruned back to the chosen size. The usage of another sample for pruning is introduced in Breiman (1998). We decide to use this method instead of the more commonly used cross-validation approach to save computation time. The details of the algorithm are as follows:

0. (Initialization) Given the training set T. Let $p^{(1)}(i) = 1/n$ for $i = 1,\ldots, n$.
1. For $m = 1,\ldots, M$:
   a. Using the probabilities $\{p^{(m)}(i)\}$, draw a sample $T^{(m)}$ of size n from T with replacement. If pruning is not being used, skip to (c).
   b. Draw another sample $V^{(m)}$ in the same manner as $T^{(m)}$. $V^{(m)}$ is used as the validation set in pruning process.

c. Construct a tree with $T^{(m)}$. If pruning is not being used, call the tree h_m and skip to (e).

d. Perform the cost-complexity pruning procedure on the tree built in step (c) with $V^{(m)}$ and find a right size. Prune the tree to the chosen size. Call it h_m.

e. Classify the observations of T by h_m. Let $d(i) = 1$ if the ith observation is classified incorrectly and $d(i) = 0$ otherwise.

f. Define

$$\varepsilon_m = \sum_i p^{(m)}(i)\, d(i),$$

$$\beta_m = (1 - \varepsilon_m) / \varepsilon_m,$$

and update the weights by

$$p^{(m+1)}(i) = p^{(m)}(i)\, \beta_m^{d(i)} / \sum_i p^{(m)}(i)\, \beta_m^{d(i)}.$$

2. (Final classifier) The final classifier is given by a weighted vote of the M trees, i.e., the class of observation $\boldsymbol{x}$ is given by

$$\arg\max_{c \in \{c_1, \cdots, c_K\}} \sum_{m=1}^{M} \log(\beta_m) I[h_m(\boldsymbol{x}) = c].$$

In step (1-c), the function `tree()` is used with the same parameter values as in bagging. As in bagging, a node becomes a terminal node when (1) its deviance is less than 1% of the root node deviance or (2) it contains fewer than four observations.

When we prune trees, to find the "right" size for an initially grown large tree in step (1-d) using the cost-complexity procedure, we run the S-Plus built-in function `prune.misclass()` with the argument "`newdata` = $V^{(m)}$." This gives us a sequence of trees with the corresponding sizes and numbers of misclassifications on $V^{(m)}$. The size with the smallest number of misclassification is chosen as the "right" size. To prune the tree to the chosen size, the function `prune.misclass()` is again used with the argument "`best` = (the chosen size)."

E. GENERAL RESULTS

Error rates of ensembles, strength and correlation, average size (number of terminal nodes) of the individual trees and average error rate of the individual trees are shown in the following sections. These results are compared with results from the other ensemble methods such as bagging, boosting and random forest. Again, test sets are used for the Image, Vowel and Waves, while five-fold cross-validation is used for the other data sets. Each ensemble consists of 100 trees. In the tables below, the results of each ensemble method are displayed in the following columns:

Column name	Method
Random (Q = 0, 1, 5, 10, 20)	Randomized splitting and permutation stopping. For $Q = 0$, permutation stopping is not used. For the other values of Q, the output variable is permuted Q times at each node.
Forest (default)	Random forest with the default setting. At each node, $F = [\log_2 p + 1]$ of input variables are randomly chosen on which to split, where p is the number of input variables and $[x]$ gives the largest integer which does not exceeds x.
Forest (mtry = 1)	Random forest in which one input variable is randomly chosen for splitting at each node.
Perturb Y (a %)	Perturbation method in which the output variable (Y) is perturbed with rate $r = a/100$.
Adaboost	Adaboost without pruning. A weighted vote is used for the final classifier.
Adaboost (pruned)	Adaboost with pruning. A weighted vote is used for the final classifier.
Bagging	Bagging without pruning.

Table IV-3: Column Names and Corresponding Ensemble Methods in Table IV-4 through Table IV-11.

1. Test Set Error of Ensembles

Table IV-4 shows the error rate estimates for each data set by each ensemble method. A test set error estimate is used for the Image, Vowel and Waves data and five-fold cross-validation is used for the others. In the table, the method with the lowest error

rate is shown in bold for each data. No ensemble methods perform uniformly better than the others. Randomized splitting is the best method, i.e., gives the lowest error rates, for the Biopsy, Ionosphere and Vowel data sets. Perturbation is the best for the Liver, Sonar and Votes data sets. Random forest is the best for the Ecoli, German credit and Image data sets. Adaboost is the best for the Glass, Vehicle and Waves data sets. Bagging is the best for the Diabetes data.

For the number of permutations, five different values (including 0) are used. The number of permutations providing the best error rate depends on the data. For 9 out of 13 data sets, the error rate is lowest when the number of permutations is 0 or 1. We discuss the sensitivity of these results to different numbers of permutation in more detail in Section V.A.

Among the five perturbation rates, the one with the smallest error rate differs from one data set to another. The best perturbation rate is 0.4 for the Glass and Image data, and 0.5 for the Vowel data. For the other data sets, the best rates are 0.1, 0.2 or 0.3. Similar results are reported in Breiman (2000). One conclusion we can easily reach from the results is that a perturbation rate of 0.5 is a very bad choice for binary class problems (Biopsy, Diabetes, German credit, Ionosphere, Liver, Sonar, and Votes). As we show in Section V.B, if we perturb some input variables together with the output variable and combine more trees, then the perturbation method could produce classifiers as good as the best in our study.

We estimate the standard deviation of ensemble error rate for some data sets to see how accurate the estimates are. We use the Image, Vowel, Waves data sets, because these data have the separate test sets. For each data, 10 sets of ensembles are produced by the randomized splitting with one permutation using the different random number seed, with estimating the error rate on the test set for each ensemble. Then, the mean and standard deviation of 10 error rates are calculated. The variability of error rates are different depending on the data sets. For the Waves data, the mean of the ensemble error rates is 0.1885 with the standard deviation of 0.0041. The mean of the error rates is 0.0502 and the standard deviation is 0.0036 for the Image data, and the mean is 0.3870

and the standard deviation is 0.0135 for the Vowel data. The complete tables for the results are given in Appendix C.

Data	Random (Q = 0)	Random (Q = 1)	Random (Q = 5)	Random (Q = 10)	Random (Q = 20)	Forest (default)	Forest (mtry = 1)
Biopsy	0.0282	**0.0238**	0.0282	0.0311	0.0253	0.0281	0.0282
Diabetes	0.2515	0.2498	0.2567	0.2462	0.2527	0.2406	0.2527
Ecoli	0.1562	0.1539	0.1500	0.1571	0.1588	0.1509	**0.1450**
German credit	0.2326	0.2576	0.2648	0.2638	0.2713	**0.2271**	0.2531
Glass	0.2522	0.2543	0.2856	0.2973	0.3138	0.2327	0.2335
Ionosphere	0.0664	0.0709	0.0654	**0.0632**	0.0659	0.0637	0.0716
Liver	0.3052	0.3095	0.3261	0.3152	0.3453	0.3047	0.2935
Sonar	0.1689	0.1624	0.1811	0.2159	0.2362	0.1674	0.2024
Vehicle	0.2690	0.2565	0.2695	0.2695	0.2646	0.2483	0.2687
Votes	0.0422	0.0468	0.0423	0.0471	0.0471	0.0404	0.0422
Image	0.0581	0.0500	0.0490	0.0619	0.0571	**0.0414**	0.0505
Vowel	**0.3918**	**0.3918**	0.3939	0.4026	0.4156	0.4156	0.4654
Waves	0.1813	0.1853	0.1887	0.2033	0.2027	0.1827	0.1907

Data	Perturb Y: 10%	Perturb Y: 20%	Perturb Y: 30%	Perturb Y: 40%	Perturb Y: 50%	Adaboost	Adaboost (Pruned)	Bagging
Biopsy	0.0325	0.0312	0.0311	0.0457	0.4810	0.0311	0.0310	0.0325
Diabetes	0.2473	0.2537	0.2600	0.3090	0.4850	0.2652	0.2640	**0.2344**
Ecoli	0.1737	0.1504	0.1612	0.1686	0.1692	0.1662	0.1798	0.1653
German credit	0.2405	0.2312	0.2398	0.2804	0.4784	0.2320	0.2424	0.2279
Glass	0.2897	0.2514	0.2654	0.2513	0.3205	**0.2210**	0.2425	0.2752
Ionosphere	0.0764	0.0644	0.0667	0.1028	0.5209	0.0659	0.0666	0.0749
Liver	0.3016	**0.2778**	0.2833	0.3351	0.5155	0.2981	0.3070	0.3029
Sonar	**0.1330**	0.1584	0.1862	0.2371	0.5365	0.1525	0.1627	0.1793
Vehicle	0.2561	0.2467	0.2394	0.2425	0.2452	0.2298	**0.2194**	0.2601
Votes	0.0468	**0.0384**	0.0428	0.0473	0.4739	0.0398	0.0485	0.0449
Image	0.0629	0.0476	0.0448	0.0424	0.0452	0.0419	0.0481	0.0562
Vowel	0.5649	0.5195	0.5087	0.4870	0.4567	0.5173	0.5087	0.5368
Waves	0.1960	0.1913	0.1873	0.1973	0.2073	0.1820	**0.1767**	0.1953

Table IV-4: Error Rates of Ensembles. Randomized Splitting and Forests (top) , Perturbation, Adaboost and Bagging (bottom).

Test set estimation is used for Image, Vowel and Waves, and five-fold cross-validation is used for the other data sets. For each data set, the method with the lowest error rate is shown in bold face. Each ensemble consists of 100 trees. Randomized splitting has the lowest error rate for Biopsy, Ionosphere and Vowel.

To compare the ensemble methods, we analyze the ranks of the methods within datasets. For each data, the method with the lowest error rate is assigned rank one, the second lowest rank two, and so on. We use average ranks in case of ties. The ranks for each data are shown in Table IV-5. The last row of the table is the sum of ranks for each method. In the table, the order of the columns is the same as Table IV-4. Random forest with the default parameter value has the smallest rank sum (column Fr1, sum 47.5), followed by Adaboost without pruning (B1, 70.5), perturbation with the rate 0.2 (P2, 74), randomized splitting with permutation = 0 (Rs1, 85) and 1 (Rs2, 87).

We conduct the Friedman test (Conover, 1999) for testing the statistical significance of the differences of rank sums. The null hypothesis is that, on average, the ensemble methods are equally good. We follow the notation in Conover (1999) here. Using Conover's equation (6) (p.370), we can calculate the test statistic T_2, which is 4.5178. The approximate distribution of T_2 is the F, with degrees of freedom given by $k_1 = 14$ and $k_2 = (13 - 1)(15 - 1) = 168$, when the null hypothesis is true. The p-value is 0.0000007 and thus the null hypothesis is rejected with the level 0.05. Further, if a difference in rank sums is greater than 39.89, then the difference is statistically significant at level 0.05. This implies that the methods with rank sums smaller than 47.5 + 39.89 = 87.39 are not statistically significantly different from random forest with the default setting, the best method in our study in terms of the rank sum. The methods with such rank sums are randomized splitting with permutation = 0 or 1, perturbation with a rate of 0.2, and Adaboost without pruning.

In addition, we also conduct the same test on the lowest error rate of each method to test if there are significant differences between ensemble methods when we assume that we could find the best parameters for each method. That is, we compare the lowest error rate among the randomized splitting with the five different numbers of permutation, the lower of two random forests, the lowest among perturbation, the lower of two Adaboost, and bagging. For example, for the Biopsy data, the smallest error rate by randomized splitting is 0.238 when the number of permutation is one, 0.281 by random forest with the default value, 0.311 by perturbation when perturbation rate is 0.2, 0.310 by Adaboost when pruning is done, and 0.325 by bagging. The lowest error rates for

each ensemble method and their ranks for each data are given in Table IV-6 and Table IV-7.

In Table IV-7, random forest has the smallest rank sum. The null hypothesis is the same as before, i.e., the ensemble methods perform equally well on average, assuming we could choose the best parameters for each method. The test statistic $T_2 = 4.095$, and the approximate distribution of this statistic is the F, with $k_1 = 4$ and $k_2 = (13 - 1)(5 - 1) = 48$. The p-value is 0.0062 and thus the null hypothesis is again rejected with the significance level 0.05. Therefore, we conclude that the ensemble methods do not perform equally well on average. We can also compute the largest difference of rank sums with which we conclude no significant difference. This is 14.57 with the level 0.05. Hence, on average, random forests are not statistically significantly better than randomized splitting, perturbation and Adaboost.

	Rs1	Rs2	Rs3	Rs4	Rs5	Fr1	Fr2	P1	P2	P3	P4	P5	B1	B2	Bag
Biopsy	5	1	5	9	2	3	5	12.5	11	9	14	15	9	7	12.5
Diabetes	6	5	10	3	7.5	2	7.5	4	9	11	14	15	13	12	1
Ecoli	6	5	2	7	8	4	1	14	3	9	12	13	11	15	10
German	5	10	12	11	13	1	9	7	3	6	14	15	4	8	2
Glass	7	8	11	13	14	2	3	12	6	9	5	15	1	4	10
Ionosphere	7	10	4	1	5.5	2	11	13	3	9	14	15	5.5	8	12
Liver	8	10	12	11	14	7	3	5	1	2	13	15	4	9	6
Sonar	7	4	9	12	13	6	11	1	3	10	14	15	2	5	8
Vehicle	13	9	14.5	14.5	11	7	12	8	6	3	4	5	2	1	10
Votes	4.5	9.5	6	11.5	11.5	3	4.5	9.5	1	7	13	15	2	14	8
Image	13	9	8	14	12	1	10	15	6	4	3	5	2	7	11
Vowel	1.5	1.5	3	4	5.5	5.5	8	15	13	10.5	9	7	12	10.5	14
Waves	2	5	7	14	13	4	8	11	9	6	12	15	3	1	10
Total	85	87	103.5	125	130	47.5	93	127	74	95.5	141	165	70.5	101.5	114.5

Table IV-5: Rank of Each Ensemble Method for Each Data.

For each data, the method with the lowest rank is assigned rank one, the second lowest rank two, and so on. Average ranks are used in case of ties. The last row gives the rank sum for each ensemble method. Random forest with the default parameter setting has the smallest rank sum of 47.5. The Friedman test shows that this method is not significantly better than the following four methods with the significance level 0.05: perturbation with the rate of 0.2 (rank sum of 74), randomized splitting with $p = 0, 1$ (respectively, 85 and 87), and the random forest with different parameter (93). "Rs" stands for randomized splitting, "Fr" random forests, "P" perturbation, "B" Adaboost and "Bag" bagging.

	Random splitting	Random forests	Perturbation	Adaboost	Bagging
Biopsy	0.0238	0.0281	0.0311	0.0310	0.0325
Diabetes	0.2462	0.2406	0.2473	0.2640	0.2344
Ecoli	0.1500	0.1450	0.1504	0.1662	0.1653
German credit	0.2326	0.2271	0.2312	0.2320	0.2279
Glass	0.2522	0.2327	0.2513	0.2210	0.2752
Ionosphere	0.0632	0.0637	0.0644	0.0659	0.0749
Liver	0.3052	0.2935	0.2778	0.2981	0.3029
Sonar	0.1624	0.1674	0.1330	0.1525	0.1793
Vehicle	0.2565	0.2483	0.2394	0.2194	0.2601
Votes	0.0422	0.0404	0.0384	0.0398	0.0449
Image	0.0490	0.0414	0.0424	0.0419	0.0562
Vowel	0.3918	0.4156	0.4567	0.5087	0.5368
Waves	0.1813	0.1827	0.1873	0.1767	0.1953

Table IV-6: Lowest Error Rate for Each Ensemble Method.

Each column gives the lowest error rate within the ensemble method. For example, for the Diabetes data with randomized splitting, the lowest error rate is 0.2462 when permutation = 10.

	Random splitting	Random forests	Perturbation	Adaboost	Bagging
Biopsy	1	2	4	3	5
Diabetes	3	2	4	5	1
Ecoli	2	1	3	5	4
German credit	5	1	3	4	2
Glass	4	2	3	1	5
Ionosphere	1	2	3	4	5
Liver	5	2	1	3	4
Sonar	3	4	1	2	5
Vehicle	4	3	2	1	5
Votes	4	3	1	2	5
Image	4	1	3	2	5
Vowel	1	2	3	4	5
Waves	2	3	4	1	5
Total	39	28	35	37	56

Table IV-7: Ranks of Ensemble Methods for Each Data.

Each row shows the ranks of the ensemble methods in Table IV-6. For each data set, the method with the lowest error rate is assigned rank one, the second lowest rank two, and so on. Random forests have the smallest rank sum of 28. The Friedman test rejects the null hypothesis that these ensemble methods perform equally well on these 13 data sets. The Friedman test also leads to the conclusion that random splitting, random forests, perturbation and Adaboost are not statistically significantly different from one another at level 0.05 for the data sets used in our study.

2. Strength

Table IV-8 depicts the strength of ensembles. For each data set, the method with the largest strength is shown in boldface. The method with the lowest ensemble error rate is shown with gray background. Randomized splitting has the largest strength for six data sets and bagging has the largest strength for six data sets. Ensembles of randomized splitting with the permutation stopping rule have higher strength than those without permutation stopping. In perturbation, the higher the perturbation rate, the lower the strength. Higher strength does not necessarily produce lower error rates.

3. Correlation

The correlation of trees in the ensembles is shown in Table IV-9. In the table, for each data, the method with the smallest correlation is shown in boldface. The method with the lowest ensemble error rate is shown with gray background. For most data sets, the correlation is the smallest for perturbation with the rate 0.5. However, these classifiers are worse than random guesses for the binary class problems. This suggests that low correlation does not guarantee low error rate if the individual trees do not have at least moderate strength. Ensembles of randomized splitting trees with permutation stopping rule have higher correlation than those without permutation stopping. In perturbation, as perturbation rate gets larger, the correlation gets lower. The correlations of Adaboost are relatively small.

4. Tree Size

The average size of the individual trees is given in Table IV-10. For random forests, tree size was not available. In random splitting with permutation stopping, as the number of permutation increases, the size of trees gets smaller. Trees built by randomized splitting with permutation stopping are much smaller than those without permutation stopping in their size. In perturbation, trees with larger perturbation rates are bigger than those with smaller rates, but not much. Adaboost and the bagging methods construct smaller trees than perturbation in most data set.

5. Average Error Rate of Individual Trees

Average error rates of the individual trees by each ensemble method are given in Table IV-11. For each data set, the method with the lowest average error rate is shown in boldface and with the lowest ensemble error rate with gray background. Randomized splitting has the lowest average error rates in six data sets and bagging does in seven data sets. Except in the Biopsy data, the method with the lowest average error rate does not coincide with the method with the lowest ensemble error rate. For most data sets, the method with the lowest average error rate is the same as the one with the largest strength, which is natural due to the definition of strength. As higher strength does not guarantee the lower error rate, more accurate individual classifiers do not produce a more accurate ensemble.

Data	Random (Q = 0)	Random (Q = 1)	Random (Q = 5)	Random (Q = 10)	Random (Q = 20)	Forest (default)	Forest (mtry = 1)
Biopsy	0.8950	**0.9059**	0.9044	0.9031	0.9038	0.8856	0.8852
Diabetes	0.3689	0.4512	0.4530	0.4570	**0.4646**	0.3510	0.3229
Ecoli	0.5610	0.6141	**0.6188**	0.6078	0.6052	0.5730	0.5308
German credit	0.3505	0.4200	**0.4266**	0.4253	0.4215	0.3343	0.2954
Glass	0.3468	**0.3604**	0.3488	0.3409	0.3183	0.3564	0.3163
Ionosphere	0.6547	0.7104	0.7252	0.7232	0.7235	0.7371	0.6838
Liver	0.2110	**0.2620**	0.2366	0.2241	0.2233	0.1972	0.1852
Sonar	0.3560	**0.4236**	0.4051	0.3934	0.3674	0.3660	0.2706
Vehicle	0.3570	0.4160	0.4175	0.4131	0.4109	0.4039	0.3316
Votes	0.7638	0.7941	0.7833	0.7796	0.7830	0.8437	0.7593
Image	0.6229	0.6804	0.6911	0.6802	0.6628	0.7578	0.6010
Vowel	0.0592	0.0894	0.0915	0.0909	0.0944	**0.0981**	0.0354
Waves	0.3375	0.3942	0.3862	0.3735	0.3723	0.3626	0.2929

Data	Perturb Y: 10%	Perturb Y: 20%	Perturb Y: 30%	Perturb Y: 40%	Perturb Y: 50%	Adaboost	Adaboost (Pruned)	Bagging
Biopsy	0.8403	0.7373	0.5681	0.3280	0.0185	0.7882	0.7906	0.8932
Diabetes	0.3279	0.2452	0.1618	0.0782	0.0063	0.2091	0.2050	0.3922
Ecoli	0.5837	0.5212	0.4310	0.3147	0.1984	0.3810	0.3703	0.6035
German credit	0.3256	0.2413	0.1629	0.0847	0.0101	0.1898	0.1911	0.3671
Glass	0.3368	0.2921	0.2409	0.1746	0.0783	0.2406	0.2296	0.3504
Ionosphere	0.6184	0.4577	0.3203	0.1658	-0.0006	0.5202	0.5136	**0.7461**
Liver	0.2194	0.1683	0.1216	0.0580	-0.0004	0.1307	0.1316	0.2259
Sonar	0.3911	0.2684	0.1784	0.0988	0.0037	0.2725	0.2708	0.4100
Vehicle	0.4193	0.3676	0.2995	0.2279	0.1570	0.2828	0.2856	**0.4353**
Votes	0.8344	0.6995	0.4997	0.2634	0.0109	0.6982	0.6943	**0.8789**
Image	0.7633	0.6944	0.5986	0.4883	0.3497	0.5737	0.5588	**0.7896**
Vowel	0.0290	0.0540	0.0622	0.0620	0.0519	0.0203	0.0125	0.0384
Waves	0.3686	0.3133	0.2408	0.1689	0.1094	0.3037	0.3034	**0.4036**

Table IV-8: Strength of Ensembles. Randomized Splitting and Forests (top) , Perturbation, Adaboost and Bagging (bottom).

Test set estimation is used for Image, Vowel and Waves, and 5-fold cross-validation is used for the other data sets. For each data set, the method with the largest strength is shown in boldface and the method with the lowest ensemble error rate is shown with gray background. Ensembles of 100 trees are constructed for each combination of data and methods. Randomized splitting has the largest strength for seven data and bagging has the largest strength for five data.

Data	Random (Q = 0)	Random (Q = 1)	Random (Q = 5)	Random (Q = 10)	Random (Q = 20)	Forest (default)	Forest (mtry = 1)
Biopsy	0.4340	0.4815	0.5006	0.4856	0.4827	0.4186	0.3956
Diabetes	0.3068	0.4825	0.5241	0.5381	0.5452	0.2683	0.2310
Ecoli	0.4184	0.5137	0.5231	0.5259	0.5122	0.4167	0.3629
German credit	0.2564	0.4552	0.5293	0.5379	0.5660	0.2215	0.1943
Glass	0.2639	0.3355	0.3741	0.3942	0.4011	0.3063	0.2434
Ionosphere	0.2422	0.3031	0.3124	0.3268	0.3377	0.3123	0.2418
Liver	0.2153	0.4092	0.4693	0.5104	0.5517	0.1711	0.1395
Sonar	0.1254	0.2019	0.2546	0.2632	0.2956	0.1272	0.0809
Vehicle	0.2505	0.3347	0.3688	0.3869	0.3818	0.2797	0.2137
Votes	0.2660	0.3218	0.3232	0.3186	0.3270	0.3622	0.2778
Image	0.2251	0.2514	0.2499	0.2612	0.2567	0.2716	0.2050
Vowel	0.0754	0.0975	0.1087	0.1199	0.1143	0.1306	**0.0719**
Waves	0.1360	0.1992	0.2166	0.2209	0.2185	0.1696	0.1158

Data	Perturb Y: 10%	Perturb Y: 20%	Perturb Y: 30%	Perturb Y: 40%	Perturb Y: 50%	Adaboost	Adaboost (Pruned)	Bagging
Biopsy	0.3174	0.1759	0.1035	0.0452	**0.0104**	0.2288	0.2380	0.4641
Diabetes	0.2455	0.1297	0.0624	0.0231	**0.0092**	0.1034	0.1027	0.3338
Ecoli	0.5256	0.3730	0.2451	0.1483	**0.0755**	0.1946	0.1959	0.5316
German credit	0.2239	0.1162	0.0548	0.0210	**0.0104**	0.0744	0.0781	0.2836
Glass	0.4002	0.2513	0.1791	0.1067	**0.0615**	0.1336	0.1275	0.3646
Ionosphere	0.2055	0.0796	0.0388	0.0165	**0.0089**	0.1070	0.1027	0.4279
Liver	0.2115	0.1159	0.0542	0.0198	**0.0112**	0.0833	0.0787	0.2069
Sonar	0.1517	0.0718	0.0361	0.0169	**0.0105**	0.0656	0.0743	0.2109
Vehicle	0.3362	0.2368	0.1598	0.0980	**0.0559**	0.1264	0.1251	0.3585
Votes	0.3604	0.1545	0.0655	0.0227	**0.0117**	0.1983	0.1952	0.5099
Image	0.3605	0.2384	0.1532	0.1018	**0.0603**	0.1041	0.0973	0.3739
Vowel	0.3413	0.2666	0.2030	0.1476	0.0986	0.1263	0.1170	0.3064
Waves	0.2114	0.1341	0.0792	0.0465	**0.0229**	0.1139	0.1130	0.2506

Table IV-9: Correlation of Trees. Randomized Splitting and Forests (top) , Perturbation, Adaboost and Bagging (bottom).

Test set estimation is used for Image, Vowel and Waves, and 5-fold cross-validation is used for the other data sets. For each data set, the method with the lowest correlation is shown in boldface and the method with the lowest ensemble error rate is shown with gray background. Ensembles of 100 trees are constructed for each combination of data sets and ensemble methods.

Data	Random (Q = 0)	Random (Q = 1)	Random (Q = 5)	Random (Q = 10)	Random (Q = 20)
Biopsy	44.55	21.25	14.97	13.41	11.92
Diabetes	174.01	24.72	12.81	10.63	9.00
Ecoli	69.34	25.35	17.21	15.43	13.91
German credit	290.13	46.87	17.49	12.41	9.64
Glass	59.37	22.53	14.31	12.04	10.26
Ionosphere	64.10	24.93	17.66	15.45	13.84
Liver	94.47	14.89	6.10	4.17	3.09
Sonar	48.10	10.77	6.81	5.72	4.74
Vehicle	227.06	73.31	48.52	41.54	37.13
Votes	30.49	23.40	18.62	17.26	15.71
Image	56.42	27.13	21.32	19.87	17.92
Vowel	179.26	90.70	67.35	60.05	54.16
Waves	89.60	25.85	17.63	15.17	13.63

Data	Perturb Y: 10%	Perturb Y: 20%	Perturb Y: 30%	Perturb Y: 40%	Perturb Y: 50%	Adaboost	Adaboost (Pruned)	Bagging
Biopsy	50.62	59.69	64.01	65.90	66.79	16.99	15.34	13.44
Diabetes	71.03	78.21	82.77	85.42	86.37	52.34	48.33	54.27
Ecoli	30.16	35.56	38.97	40.84	42.05	26.38	24.89	21.87
German credit	95.13	102.92	107.82	110.16	110.74	67.12	62.27	72.01
Glass	23.59	25.33	26.38	26.82	27.01	23.25	20.81	23.22
Ionosphere	23.54	30.28	34.39	36.75	37.39	12.23	10.67	12.82
Liver	37.93	39.68	40.73	41.55	41.72	34.82	30.19	39.19
Sonar	15.89	17.92	19.17	19.93	20.20	10.20	8.75	11.89
Vehicle	63.42	73.45	81.70	87.85	91.71	47.86	45.68	42.33
Votes	36.96	44.91	47.98	49.37	49.85	17.50	15.42	12.51
Image	20.33	24.73	28.24	30.91	32.31	15.39	14.31	13.30
Vowel	46.26	54.11	61.10	67.52	72.75	35.38	35.35	35.57
Waves	31.75	36.58	40.22	42.74	44.32	19.35	17.89	22.21

Table IV-10: Average Size of Individual Trees. Randomized Splitting (top), Perturbation, Adaboost and Bagging (bottom).

Test set estimation is used for Image, Vowel and Waves data sets, and 5-fold cross-validation is used for the other data sets. The size of a tree is defined by its number of terminal node. Tree size is not available for random forests.

Data	Random (Q = 0)	Random (Q = 1)	Random (Q = 5)	Random (Q = 10)	Random (Q = 20)	Forest (default)	Forest (mtry = 1)
Biopsy	0.0525	**0.0471**	0.0478	0.0484	0.0481	0.0572	0.0574
Diabetes	0.3156	0.2744	0.2735	0.2715	**0.2677**	0.3245	0.3385
Ecoli	0.2538	0.2163	**0.2141**	0.2186	0.2192	0.2435	0.2713
German credit	0.3247	0.2900	**0.2867**	0.2874	0.2893	0.3328	0.3523
Glass	0.3967	0.3742	0.3740	0.3745	0.3889	0.3845	0.4188
Ionosphere	0.1726	0.1448	0.1374	0.1384	0.1382	0.1314	0.1581
Liver	0.3945	**0.3690**	0.3817	0.3880	0.3883	0.4014	0.4074
Sonar	0.3220	**0.2882**	0.2974	0.3033	0.3163	0.3170	0.3647
Vehicle	0.3735	0.3289	0.3268	0.3283	0.3300	0.3366	0.3938
Votes	0.1181	0.1030	0.1083	0.1102	0.1085	0.0781	0.1203
Image	0.2327	0.1933	0.1888	0.1961	0.2047	0.1426	0.2495
Vowel	0.6882	0.6593	0.6570	0.6463	0.6482	0.6476	0.7256
Waves	0.3491	0.3146	0.3183	0.3252	0.3262	0.3363	0.3816

Data	Perturb Y: 10%	Perturb Y: 20%	Perturb Y: 30%	Perturb Y: 40%	Perturb Y: 50%	Adaboost	Adaboost (Pruned)	Bagging
Biopsy	0.0799	0.1313	0.2160	0.3360	0.4908	0.1059	0.1047	0.0534
Diabetes	0.3360	0.3774	0.4191	0.4609	0.4968	0.3954	0.3975	0.3039
Ecoli	0.2349	0.2883	0.3603	0.4437	0.5310	0.3593	0.3661	0.2248
German credit	0.3372	0.3793	0.4185	0.4577	0.4950	0.4051	0.4044	0.3164
Glass	0.3714	0.4202	0.4652	0.5241	0.5945	0.4493	0.4554	**0.3710**
Ionosphere	0.1908	0.2711	0.3399	0.4171	0.5003	0.2399	0.2432	**0.1270**
Liver	0.3903	0.4159	0.4392	0.4710	0.5002	0.4347	0.4342	0.3871
Sonar	0.3044	0.3658	0.4108	0.4506	0.4982	0.3638	0.3646	0.2950
Vehicle	0.3245	0.3750	0.4431	0.5088	0.5776	0.4116	0.4100	**0.3086**
Votes	0.0828	0.1502	0.2501	0.3683	0.4946	0.1509	0.1529	**0.0606**
Image	0.1391	0.2009	0.2855	0.3829	0.5038	0.2683	0.2767	**0.1204**
Vowel	0.6130	0.6237	0.6529	0.6871	0.7264	0.6681	0.6744	**0.6140**
Waves	0.3372	0.3887	0.4484	0.5103	0.5622	0.3696	0.3696	**0.3092**

Table IV-11: Average Error Rates of Individual Trees. Randomized Splitting and Forests (top) , Perturbation, Adaboost and Bagging (bottom).

Test set estimation is used for Image, Vowel and Waves, and 5-fold cross-validation is used for the other data sets. For each data, the method with the lowest average error rate is shown in boldface. Gray background indicates the method with the smallest ensemble error rate for each data set. Ensembles of 100 trees are produced for each combination of data and methods.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DISCUSSION

We see that randomized splitting with permutation stopping performs at least as well as other ensemble methods. This method has the lowest error rates for 3 out of 13 data sets. Before we started the simulation, we had expected that several permutations at each node would be needed to construct a good ensemble. To our surprise, no permutation or one permutation at each node give the lowest or close to the lowest error rate among the five values of permutation. This is discussed in Section A. The perturbation method also produces good ensembles, as is also shown by Breiman (2000). As we see in Section B, by perturbing inputs and combining more trees, the perturbation method could construct ensembles comparable to those by the best method known so far. The problem with this approach is that there is no way to know what perturbation rate to use nor how many trees we need to combine. This problem might be solved by combining ensemble methods with sub-sampling (which means sampling of subsets without replacement) of the training set. We have explored this idea which is similar to the use of bagging with random forests by Breiman (2001). The results of a few trials are encouraging and given in Section C. We have found that the plots of correlation versus strength for some data sets form very smooth curves. Some typical examples are shown in Section D. During our simulations, we found an interesting feature of Adaboost which is discussed in Section E.

A. PERMUTATION STOPPING RULE

In randomized splitting with permutation stopping, we employ five values for number of permutations: 0 (no permutation stopping), 1, 5, 10 and 20. The error rates and the average tree sizes are shown in Table IV-4 and Table IV-10 together with other ensemble methods, and given again in the following tables. Among the five values of permutation, zero or one permutation performed best in 9 out of 13 datasets. Table V-2 depicts the average sizes of individual trees. For most data sets, the average size of the trees with one permutation is less than half of that without permutation. This suggests that the number of exhaustive searches to find splits is smaller for randomized splitting

with one permutation than for randomized splitting with no permutation. Because the tree size is much smaller on average, trees built with one permutation are faster in predicting a new observation than trees built without permutation. At the cost of a little extra time when a tree is constructed (the one permutation at each node), we can produce a classifier for which prediction is computationally less expensive. Of course, compared to the cost (computational time) to train a tree, to run a new data set down a tree is much cheaper. Nevertheless, if we would like to predict massive data sets by using an ensemble consisting of thousands of trees, tree size might matter.

Permutation stopping is not computationally cheap, because it permutes the output and does exhaustive search to find the threshold value (that is, the maximum deviance reduction) with the permuted data. If we could come up with a less expensive way to find a good threshold, we can reduce the time needed to construct effective ensembles.

Data	permutation=0	permutation=1	permutation=5	permutation=10	permutation=20
Biopsy	0.0282	**0.0238**	0.0282	0.0311	0.0253
Diabetes	0.2515	0.2498	0.2567	**0.2462**	0.2527
Ecoli	0.1562	0.1539	**0.1500**	0.1571	0.1588
German credit	**0.2326**	0.2576	0.2648	0.2638	0.2713
Glass	**0.2522**	0.2543	0.2856	0.2973	0.3138
Ionosphere	0.0664	0.0709	0.0654	**0.0632**	0.0659
Liver	**0.3052**	0.3095	0.3261	0.3152	0.3453
Sonar	0.1689	**0.1624**	0.1811	0.2159	0.2362
Vehicle	0.2690	**0.2565**	0.2695	0.2695	0.2646
Votes	**0.0422**	0.0468	0.0423	0.0471	0.0471
Image	0.0581	0.0500	**0.0490**	0.0619	0.0571
Vowel	**0.3918**	**0.3918**	0.3939	0.4026	0.4156
Waves	**0.1813**	0.1853	0.1887	0.2033	0.2027

Table V-1: Error Rates of Randomized Splitting and Permutation Stopping.

For each data set, the methods with the smallest error rate is shown in boldface. Permutation = 0 or 1 has the lowest or close to the lowest error rate for most data sets.

Data	permutation=0	permutation=1	permutation=5	permutation=10	permutation=20
Biopsy	44.55	21.25	14.97	13.41	11.92
Diabetes	174.01	24.72	12.81	10.63	9.00
Ecoli	69.34	25.35	17.21	15.43	13.91
German credit	290.13	46.87	17.49	12.41	9.64
Glass	59.37	22.53	14.31	12.04	10.26
Ionosphere	64.10	24.93	17.66	15.45	13.84
Liver	94.47	14.89	6.10	4.17	3.09
Sonar	48.10	10.77	6.81	5.72	4.74
Vehicle	227.06	73.31	48.52	41.54	37.13
Votes	30.49	23.40	18.62	17.26	15.71
Image	56.42	27.13	21.32	19.87	17.92
Vowel	179.26	90.70	67.35	60.05	54.16
Waves	89.60	25.85	17.63	15.17	13.63

Table V-2: Average Size of Individual Trees by Randomized Splitting and Permutation Stopping.

As we permute more, the trees get smaller. In most data sets, the average size of the trees with one permutation is less than half of that of no permutation.

B. PERTURBATION

As shown in Table IV-4, perturbation produces classifiers that are more accurate than bagging for most data sets in our study but less accurate than randomized splitting or Adaboost for some data sets. Similar results are also reported in Breiman (2000). During our simulation, simple questions came up: Can we improve accuracy by perturbing one or some of the input variables together with the output variable, or by combining more trees? To answer the questions, additional computations are conducted using the Vowel and Waves data. These data sets are chosen only because they have separate test sets. The results are shown in Table V-3 for the Vowel and in Table V-4 for the Waves.

Before we look at the results, let us go back to Table IV-4 where we show the error rates of ensembles of 100 trees. The lowest error rate for the Vowel data is 0.3918, obtained by using randomized splitting with 0 or 1 permutations. The best error rate for the same data using perturbation is 0.4567 with perturbation rate of 0.5, which is about 16% higher than the lowest error rate of all ensemble methods. For the Waves data, the lowest error rate is 0.1767 by Adaboost with pruning. The lowest rate using perturbation

is 0.1873 with perturbation rate of 0.3. This itself is not a very bad result, only 6% higher than the best.

Table V-3 and Table V-4 give the results when we change the parameters for perturbation and number of trees combined. These tables show perturbation rates of the output and input variables (*y.rate*, *x.rate*), number of the input variables to be perturbed (*#x*), test set error rates for ensembles consisting of *n* trees (*Ensemble(n)*), average size (*Size*), average test set error rate (*Test err.*), strength (*Strength*), and correlation (*Corr.*). Ensembles of 1,000 trees are used to estimate the average size, average test set error rate, strength, and correlation. The size of a tree is given by the number of terminal nodes. In the tables, we can see that the error rate changes as the perturbation rate changes. For the Vowel data, the test set error is smallest when the perturbation rate of the output is 0.5 and of the input is 0.2 with all the input variables being perturbed. By perturbing the inputs, the test set error decreases by about 10% (from 0.4481, without perturbing the inputs, to 0.4004, when 1,000 trees are combined). For the Waves data, the test set error is the smallest (0.1700) when perturbation rate of the output is 0.5. Perturbing the inputs does not seem to contribute significantly to reducing the test set error rate. These error rates are as good as the best ones in Table IV-4. When combining a large enough number of trees, as perturbation rate of the output gets larger, error rates get lower. One explanation of this is that correlation decreases pretty quickly as perturbation rate of the output increases for these data.

As to the number of trees combined, when a small number, say 100, of trees are combined, the perturbation method gives higher test set error rates than the lowest error rate of all methods (0.3918 for the Vowel and 0.1767 for the Waves). Test set error rates get lower as more trees are combined. Eventually, for some perturbation rates, error rates are as low as the best rate achieved by any other methods. Test set error rates are minimized after building about 300 trees for the Vowel data and after about 500 trees for the Waves data.

The results in this section show that, by perturbing some input variables and by combining more trees, perturbation could provide accuracy as good as the randomized

splitting and Adaboost. A drawback of this method is, as Breiman (2000) points out, that there are no good ways to find a good perturbation rate for a given data set.

y.rate	x.rate	#x	Ensemble (100)	Ensemble (300)	Ensemble (500)	Ensemble (700)	Ensemble (1000)	Size	Test err.	Strength	Corr.
0.10	0.00	0	0.5649	0.5649	0.5671	0.5693	0.5628	46.46	0.6144	0.0273	0.3430
0.20	0.00	0	0.5195	0.5195	0.5238	0.5195	0.5195	54.34	0.6253	0.0622	0.2594
0.30	0.00	0	0.5087	0.5000	0.4957	0.5043	0.5065	61.57	0.6482	0.0719	0.2039
0.30	0.10	5	0.4935	0.4870	0.4870	0.4762	0.4892	61.97	0.6484	0.0757	0.1906
0.30	0.10	8	0.5000	0.4913	0.4870	0.4762	0.4805	62.36	0.6479	0.0807	0.1806
0.30	0.20	5	0.4740	0.4589	0.4524	0.4502	0.4567	62.98	0.6527	0.0793	0.1652
0.30	0.30	5	0.4848	0.4697	0.4654	0.4654	0.4610	63.82	0.6596	0.0806	0.1439
0.40	0.00	0	0.4870	0.4740	0.4654	0.4632	0.4762	67.63	0.6825	0.0732	0.1485
0.40	0.10	1	0.4913	0.4762	0.4762	0.4719	0.4632	67.63	0.6800	0.0766	0.1451
0.40	0.10	3	0.4654	0.4567	0.4632	0.4524	0.4567	67.80	0.6812	0.0751	0.1423
0.40	0.20	3	0.4502	0.4481	0.4589	0.4545	0.4545	68.16	0.6851	0.0763	0.1297
0.40	0.10	5	0.4502	0.4481	0.4437	0.4351	0.4416	67.71	0.6800	0.0787	0.1384
0.40	0.20	5	0.4654	0.4567	0.4416	0.4502	0.4416	68.44	0.6850	0.0768	0.1215
0.40	0.10	8	0.4610	0.4459	0.4524	0.4394	0.4437	68.04	0.6833	0.0766	0.1309
0.40	0.10	11	0.4481	0.4567	0.4545	0.4524	0.4437	68.19	0.6835	0.0796	0.1295
0.50	0.00	0	0.4567	0.4416	0.4416	0.4459	0.4481	72.58	0.7247	0.0652	0.0964
0.50	0.10	11	0.4567	0.4545	0.4372	0.4351	0.4351	72.78	0.7295	0.0651	0.0843
0.50	0.20	11	0.4351	0.4026	**0.3874**	0.4004	0.4004	73.78	0.7356	0.0647	0.0636
0.50	0.20	5	0.4221	0.4156	0.4091	0.4156	0.4221	72.87	0.7283	0.0644	0.0798

Table V-3: Results for Perturbation (Vowel).

Each column displays the following: *y.rate* = perturbation rate for the output, *x.rate* = perturbation rate for the input, #*x* = number of input variables to be perturbed, *Ensemble (n)* = test set error rate of the ensemble consisting of *n* trees, *Size* = average size of individual trees, *Test err.* = average test set error rate of individual trees, *Strength* = strength, *Corr* = correlation, where ensembles of 1,000 trees are used to estimate the latter four values.

When perturbing the output with rate 0.5 and all the input variables with rate 0.2, i.e., when *y.rate* = 0.5, *x.rate* = 0.2, #*x* = 11 (the second row from the bottom), the error rate is as small as the best ensemble methods (randomized splitting) in Table IV-4.

y.rate	x.rate	#x	Ensemble (100)	Ensemble (300)	Ensemble (500)	Ensemble (700)	Ensemble (1000)	Size	Test err.	Strength	Corr
0.10	0.00	0	0.1960	0.1940	0.1907	0.1907	0.1940	31.91	0.3375	0.3703	0.2065
0.10	0.10	11	0.1913	0.1927	0.1927	0.1907	0.1880	31.93	0.3339	0.3772	0.2011
0.20	0.00	0	0.1913	0.1867	0.1840	0.1820	0.1867	36.57	0.3935	0.3064	0.1260
0.30	0.00	0	0.1873	0.1793	0.1740	0.1740	0.1713	40.09	0.4524	0.2415	0.0739
0.40	0.00	0	0.1973	0.1813	**0.1693**	0.1713	0.1733	42.62	0.5112	0.1724	0.0385
0.50	0.00	0	0.2073	0.1733	0.1700	0.1700	0.1700	44.34	0.5693	0.1060	0.0153
0.50	0.10	1	0.2373	0.1953	0.1847	0.1807	0.1780	44.35	0.5727	0.1017	0.0152
0.50	0.10	3	0.2387	0.1853	0.1747	0.1780	0.1760	44.43	0.5699	0.1039	0.0151
0.50	0.10	5	0.2240	0.1940	0.1867	0.1853	0.1807	44.28	0.5688	0.1048	0.0156
0.50	0.10	8	0.2140	0.1913	0.1907	0.1753	0.1707	44.38	0.5681	0.1049	0.0150
0.50	0.10	11	0.2133	0.1753	0.1780	0.1713	0.1733	44.37	0.5691	0.1051	0.0158
0.50	0.10	15	0.2160	0.1960	0.1793	0.1847	0.1713	44.39	0.5703	0.1044	0.0153
0.50	0.05	11	0.2253	0.1967	0.1780	0.1760	0.1760	44.30	0.5701	0.1044	0.0154

Table V-4: Results for Perturbation (Waves).

Test set error tends to be lower for smaller perturbation rates when an ensemble consists of 100 trees. When the size of ensembles is large enough, test set errors tend to be lower for larger perturbation rate. In the cases where the perturbation rate for the output variable is larger than 0.3, test set error rates are not significantly different each other and are as good as the best ensemble method in Table IV-4. Perturbing the inputs does not change the results as much as for the Vowel data.

C. COMBINING PROPOSED METHODS WITH SUB-SAMPLING

In the study on random forests (Breiman, 2001), Breiman combines bagging with random forests as follows: Given a training set *T*, a new sample of the same size as *T* is randomly drawn from *T* with replacement, i.e., a new bootstrap sample is drawn from *T*. Then a tree is built on the drawn set using a random forest algorithm. For each observation in *T*, the "out-of-bag" classifier is formed and used to obtain the out-of-bag estimate of error rate, strength and correlation for the forest. The out-of-bag classifier for an observation in *T* is the ensemble formed by only those trees built with bootstrap samples that do not contain the observation. The out-of-bag estimate for the generalized error is defined as the error rate among the out-of-bag classifiers on *T*. Breiman uses bagging because (1) the use of bagging seems to improve accuracy, and (2) bagging can be used to obtain out-of-bag estimates of the generalization error, strength and correlation. The use of out-of-sample estimates seemed to be useful, and we tried the

same method as Breiman on our randomized splitting algorithm for two data sets. When we combined bagging with the randomized splitting, the accuracy of the ensemble was improved on one data set and not very different on the other data set. But we do not explore this idea further here because we have difficulty in combining bagging with the perturbation method. The perturbation method has shown to perform as well as other ensemble methods on average even though it is such a simple idea. One shortcoming of this method is that, as Breiman (2000) further points out, there are no good ways to find an appropriate perturbation rate to use for a given data set. Although the use of out-of-bag estimates seems to be helpful, we do not believe that combining bagging with perturbation is a good idea. For, if we perturb the outputs of a bootstrap sample, we would have a number of observations with the same input variables but different class label. Then, what if we use sampling without replacement instead of sampling with replacement? This is where we started to consider combining sub-sampling with ensemble methods.

Our idea succeeds in a way similar to Breiman's use of bagging in random forests. In the sub-sampling method, instead of drawing a bootstrap sample on each iteration, we randomly draw a fraction of the given training set (without replacement). The rest of the observations in the training set is used for estimation of the error rate, strength and correlation. We call this estimation "out-of-sample" estimation after "out-of-bag" estimation. At first, using a random fraction of the training set to build each classifier did not seem to be a good idea since the resulting individual classifiers in the ensemble would not be as good as those built using the whole training set. However, the use of a different fraction of the training set for each classifier might reduce correlation and thus the resulting ensembles might not be much worse than ensembles of classifiers built using the whole training set. Sub-sampling might also help to reduce training time for huge data sets.

We have tried the sub-sampling method on our proposed methods using a sampling rate of 0.5. For the randomized splitting with permutation stopping, one and no permutations are used. Perturbation rates of 0.1 and 0.2 are used for the perturbation methods. We do estimation in the same way as before. For each combination of the data

and ensemble methods, 100 trees are grown. For data sets with test sets, the test sets are used for estimation, while 5-fold cross-validation is used for the other data sets. The results are shown in the tables below. Each table depicts the error rate of ensemble, average size of the individual trees, average error rate of the individual trees, strength, and correlation on the top. The out-of-sample estimates are given in the bottom. The last two columns of the table on the top show the error rates of the ensembles constructed using the same method but without sub-sampling, and change in error rates in percent. A negative value of the change means that the ensemble with sub-sampling produces a lower error rate than the ensemble without sub-sampling. The ensemble error rates corresponding to these negative values are shown in boldface. An error rate shown with gray background indicates that the error rate is lower than the lowest error rate of all the ensemble methods used in our study.

Table V-5 and Table V-6 give the results of combining sub-sampling with randomized splitting using one and no permutations, respectively. In both tables, the error rates of ensembles are within 10% of those of ensembles constructed by using the same method without sub-sampling except for the Glass, Sonar, and Image data sets. The error rates are even smaller for some data sets. The results of the perturbation method with sub-sampling are shown in Table V-7 and Table V-8. With a perturbation rate of 0.1, the error rates of the ensembles are smaller than those of the ensembles constructed by using the same methods without sub-sampling for most data sets in our study. For the Image, Vowel, and Waves data sets, we have also tried different sampling rates and parameter values. When we use a sampling rate of 0.7 on the randomized splitting with one permutation for the Image data set, the error rate of the ensemble is reduces to 0.0486 from 0.0657 by a sampling rate of 0.5. Also, when we use a sampling rate of 0.5 on the perturbation method with perturbation rate 0.3 for the Image data set, the error rate of ensemble is 0.0376 which is smaller than the smallest error rate of all the methods we used. The out-of-sample estimates seem to be good estimates on average. This is encouraging. We might be able to use out-of-sample estimates to decide the good parameters for a given ensemble method. More exploration is needed on this sub-sampling method.

Data	Ensemble	Size	Size Stdev	Test err.	Test stdev	Strength	Corr.	Ensemble (all)	Change (%)
Biopsy	0.0253	13.22	3.15	0.0518	0.0163	0.8964	0.4359	0.0238	6.3
Diabetes	**0.2380**	14.22	4.01	0.2785	0.0342	0.4430	0.4322	0.2498	-4.7
Ecoli	**0.1407**	15.90	2.77	0.2493	0.0487	0.5683	0.4323	0.1539	-8.6
German	0.2639	24.53	10.90	0.2979	0.0276	0.4042	0.4257	0.2576	2.4
Glass	0.2982	12.12	2.70	0.4234	0.0754	0.2839	0.3022	0.2543	17.3
Ionosphere	**0.0676**	13.59	3.20	0.1709	0.0509	0.6583	0.2691	0.0709	-4.7
Liver	0.3135	7.33	3.95	0.3968	0.0652	0.2063	0.2817	0.3095	1.3
Sonar	0.2344	5.77	1.54	0.3267	0.0688	0.3467	0.1886	0.1624	44.3
Vehicle	0.2610	41.03	6.44	0.3622	0.0381	0.3735	0.2793	0.2565	1.8
Votes	**0.0445**	14.74	6.65	0.1440	0.1178	0.7120	0.2460	0.0468	-4.9
Image	0.0657	17.32	2.53	0.2649	0.0483	0.5726	0.1931	0.0500	31.4
Vowel	0.3983	49.26	3.77	0.6779	0.0544	0.0765	0.0917	0.3918	1.7
Waves	0.1880	14.53	2.11	0.3359	0.0284	0.3607	0.1753	0.1853	1.5

Data	Ensemble (oos)	Test err. (oos)	Test stdev (oos)	Strength (oos)	Corr. (oos)
Biopsy	0.0264	0.0529	0.0143	0.8942	0.4238
Diabetes	0.2399	0.2775	0.0268	0.4466	0.4420
Ecoli	0.1439	0.2463	0.0438	0.5701	0.4179
German	0.2592	0.2987	0.0268	0.4023	0.4364
Glass	0.2908	0.4213	0.0635	0.2802	0.2970
Ionosphere	0.0705	0.1692	0.0420	0.6632	0.2722
Liver	0.3183	0.3892	0.0487	0.2228	0.2827
Sonar	0.2381	0.3246	0.0559	0.3539	0.2120
Vehicle	0.2672	0.3635	0.0322	0.3674	0.2795
Votes	0.0486	0.1457	0.1145	0.7077	0.2651
Image	0.0857	0.2830	0.0655	0.5446	0.2215
Vowel	0.0568	0.4456	0.0511	0.3571	0.0950
Waves	0.1733	0.3273	0.0434	0.3814	0.1833

Table V-5: Randomized Splitting (Permutation = 1) with Sub-sampling (Sampling Rate = 0.5)

The column *ensemble (all)* gives the error rates of the ensembles constructed using the same ensemble method without sub-sampling. For each data set, boldface indicates that the error rate of ensemble is smaller than that without sub-sampling. Gray background indicates that the error rate is smaller than the smallest error rate of all the ensemble methods used in our study. The error rates of ensembles are within 10% of those of ensembles constructed by using the same method without sub-sampling except for the Glass, Sonar, and Image data sets. For the Ecoli data, the error rate is smaller than that of the best ensemble in our study.

Data	Ensemble	Size	Size Stdev	Test err.	Test stdev	Strength	Corr.	Ensemble (all)	Change (%)
Biopsy	**0.0282**	25.33	5.09	0.0575	0.0179	0.8850	0.3961	0.0282	0.0
Diabetes	**0.2367**	88.25	5.95	0.3200	0.0340	0.3599	0.2701	0.2515	-5.9
Ecoli	**0.1522**	37.92	3.99	0.2878	0.0526	0.5179	0.3598	0.1562	-2.6
German	0.2390	146.48	6.91	0.3346	0.0334	0.3308	0.2278	0.2326	2.8
Glass	0.2873	31.84	2.23	0.4654	0.0740	0.2539	0.2102	0.2522	13.9
Ionosphere	0.0722	35.20	4.64	0.1998	0.0496	0.6004	0.2216	0.0664	8.7
Liver	0.3111	48.05	3.36	0.4037	0.0596	0.1926	0.1585	0.3052	1.9
Sonar	0.2026	25.58	2.78	0.3547	0.0744	0.2905	0.1023	0.1689	20.0
Vehicle	0.2693	121.74	6.41	0.4100	0.0396	0.3107	0.2051	0.2690	0.1
Votes	0.0437	17.92	7.11	0.1644	0.1245	0.6712	0.2312	0.0422	3.6
Image	0.0781	33.54	3.63	0.3053	0.0472	0.5242	0.1862	0.0581	34.4
Vowel	**0.3658**	102.45	3.69	0.7103	0.0512	0.0597	0.0589	0.3918	-6.6
Waves	**0.1793**	46.39	4.11	0.3629	0.0257	0.3169	0.1215	0.1813	-1.1

Data	Ensemble (oos)	Test err. (oos)	Test stdev (oos)	Strength (oos)	Corr. (oos)
Biopsy	0.0267	0.0552	0.0145	0.8896	0.3968
Diabetes	0.2382	0.3174	0.0261	0.3662	0.2753
Ecoli	0.1451	0.2847	0.0436	0.5180	0.3475
German	0.2368	0.3349	0.0266	0.3299	0.2354
Glass	0.2859	0.4619	0.0622	0.2488	0.2093
Ionosphere	0.0835	0.2030	0.0406	0.5957	0.2346
Liver	0.2790	0.3979	0.0413	0.2052	0.1619
Sonar	0.2144	0.3538	0.0584	0.2933	0.1124
Vehicle	0.2745	0.4099	0.0340	0.3060	0.2083
Votes	0.0592	0.1614	0.1198	0.6765	0.2553
Image	0.1000	0.3292	0.0709	0.4926	0.1918
Vowel	0.0530	0.4613	0.0438	0.3593	0.0766
Waves	0.1667	0.3529	0.0464	0.3402	0.1265

Table V-6: Randomized Splitting with Sub-sampling (Sampling Rate = 0.5)

The error rates of ensembles are within 10% of those of ensembles constructed by using the same method without sub-sampling except for the Glass, Sonar, and Image data sets. For the Vowel data set, the error rate is smaller than that of the best ensemble in our study.

Data	Ensemble	Size	Size Stdev	Test err.	Test stdev	Strength	Corr.	Ensemble (all)	Change (%)
Biopsy	**0.0325**	51.09	6.15	0.1202	0.0336	0.7596	0.1883	0.0325	0.0
Diabetes	**0.2368**	63.56	4.95	0.3476	0.0380	0.3048	0.1867	0.2473	-4.2
Ecoli	**0.1566**	30.06	3.84	0.3082	0.0582	0.5049	0.3329	0.1737	-9.8
German	**0.2253**	131.74	7.72	0.3590	0.0325	0.2819	0.1590	0.2405	-6.3
Glass	**0.2635**	22.42	2.38	0.4393	0.0796	0.2827	0.2229	0.2897	-9.0
Ionosphere	**0.0699**	21.94	3.40	0.2101	0.0574	0.5797	0.1473	0.0764	-8.5
Liver	**0.2617**	34.55	3.35	0.4088	0.0635	0.1824	0.1250	0.3016	-13.2
Sonar	0.1891	11.25	1.47	0.3471	0.0764	0.3058	0.1183	0.1330	42.2
Vehicle	**0.2544**	86.50	6.55	0.3910	0.0379	0.3465	0.2172	0.2561	-0.7
Votes	**0.0414**	47.03	5.84	0.1401	0.0459	0.7198	0.1584	0.0468	-11.5
Image	**0.0467**	19.57	2.56	0.2323	0.0449	0.6547	0.1947	0.0629	-25.8
Vowel	**0.5238**	66.42	4.27	0.6485	0.0396	0.0536	0.2177	0.5649	-7.3
Waves	**0.1880**	24.22	2.41	0.3653	0.0300	0.3369	0.1509	0.1960	-4.1

Data	Ensemble (oos)	Test err. (oos)	Test stdev (oos)	Strength (oos)	Corr. (oos)
Biopsy	0.0308	0.1191	0.0281	0.7620	0.1874
Diabetes	0.2353	0.3469	0.0309	0.3070	0.1931
Ecoli	0.1533	0.3114	0.0511	0.4926	0.3208
German	0.2365	0.3574	0.0267	0.2856	0.1717
Glass	0.2451	0.4361	0.0656	0.2854	0.2199
Ionosphere	0.0734	0.2122	0.0474	0.5743	0.1572
Liver	0.2768	0.4000	0.0460	0.2015	0.1353
Sonar	0.1998	0.3436	0.0616	0.3119	0.1262
Vehicle	0.2473	0.3888	0.0335	0.3450	0.2210
Votes	0.0453	0.1386	0.0359	0.7235	0.1832
Image	0.0952	0.2588	0.0577	0.6108	0.2272
Vowel	0.0777	0.3736	0.0387	0.4571	0.1497
Waves	0.1767	0.3643	0.0456	0.3430	0.1536

Table V-7: Perturbation (*Y*'s Perturbation Rate = 0.2) with Sub-sampling (Sampling Rate = 0.5)

The error rates are smaller than those of the ensembles constructed by using the same methods without sub-sampling except for the sonar data set. For the German and Liver data sets, the error rates are smaller than those of the best ensembles in our study.

Data	Ensemble	Size	Size Stdev	Test err.	Test stdev	Strength	Corr.	Ensemble (all)	Change (%)
Biopsy	**0.0252**	65.43	5.70	0.1904	0.0449	0.6192	0.1063	0.0312	-19.2
Diabetes	**0.2300**	70.59	4.99	0.3862	0.0426	0.2277	0.1001	0.2537	-9.3
Ecoli	0.1639	37.33	4.11	0.3862	0.0665	0.4092	0.2351	0.1504	9.0
German	**0.2259**	138.79	6.43	0.3947	0.0376	0.2105	0.0876	0.2312	-2.3
Glass	0.2642	25.28	2.67	0.4838	0.0833	0.2431	0.1709	0.2514	5.1
Ionosphere	0.0651	26.77	3.42	0.2866	0.0695	0.4269	0.0699	0.0644	1.1
Liver	0.3047	37.04	3.51	0.4340	0.0598	0.1321	0.0752	0.2778	9.7
Sonar	0.1916	12.18	1.42	0.3923	0.0847	0.2154	0.0617	0.1584	21.0
Vehicle	0.2476	100.22	6.40	0.4541	0.0441	0.2828	0.1477	0.2467	0.4
Votes	0.0455	55.35	4.77	0.2257	0.0604	0.5485	0.0750	0.0384	18.5
Image	**0.0471**	24.84	3.12	0.3231	0.0504	0.5528	0.1389	0.0476	-1.1
Vowel	**0.5152**	77.03	4.14	0.6837	0.0429	0.0471	0.1713	0.5195	-0.8
Waves	0.1933	28.80	2.42	0.4278	0.0330	0.2665	0.1002	0.1913	1.0

Data	Ensemble (oos)	Test err. (oos)	Test stdev (oos)	Strength (oos)	Corr. (oos)
Biopsy	0.0289	0.1893	0.0387	0.6214	0.1171
Diabetes	0.2516	0.3901	0.0323	0.2205	0.1115
Ecoli	0.1482	0.3811	0.0594	0.4132	0.2300
German	0.2438	0.3935	0.0311	0.2138	0.0999
Glass	0.2564	0.4819	0.0689	0.2415	0.1760
Ionosphere	0.0700	0.2890	0.0552	0.4223	0.0834
Liver	0.2934	0.4258	0.0457	0.1475	0.0806
Sonar	0.2163	0.3882	0.0631	0.2239	0.0725
Vehicle	0.2482	0.4533	0.0357	0.2778	0.1541
Votes	0.0452	0.2239	0.0474	0.5523	0.0864
Image	0.0857	0.3457	0.0652	0.5128	0.1747
Vowel	0.0852	0.4420	0.0448	0.3904	0.1079
Waves	0.1667	0.4192	0.0499	0.2839	0.1007

Table V-8: Perturbation (*Y*'s Perturbation Rate = 0.2) with Sub-sampling (Sampling Rate = 0.5)

For the German data set, the error rate is smaller than that of the best ensemble in our study.

D. STRENGTH AND CORRELATION

During our computation, for each combination of the data sets and ensemble methods, we recorded the error rate, strength and correlation of the ensemble. To explore the relationship of error rates with strength and correlation, we drew a plot like Figure V-1 for each data set. In the figure, the error rates are plotted according to the corresponding strength and correlation with the ensemble methods used. The numbers in the plot indicate the error rate of the ensembles in percent and symbols under the error rates indicate the ensemble methods used. In the figure, "ad" stands for Adaboost, "bg" bagging, "fr" random forests, "pe" perturbation, "ps" perturbation with sub-sampling, "rp" randomized splitting with permutation stopping, and "rs" randomized splitting with permutation stopping combined with sub-sampling. The points that produce the smallest error rate for the data set are indicated by rectangular boxes.

Figure V-1 shows the correlation and strength for the Waves data set. The correlation and strength have a strong smooth non-linear relationship in this figure. Among 12 other data sets used in our study, the Biopsy, Diabetes, Ecoli, German, Ion, and Votes show the similar relationship. Figure V-2 depicts the same plot for the Image data sets. The points form a small cloud rather than a non-linear curve. The points in upper-left side of the cloud tend to have higher error rates than the points in the lower-right side of the cloud. The same plot for the Vowel data set is shown in Figure V-3. This figure looks different from the previous two figures. In the figure, toward the upper-left corner, the error rate seems to increase.

From the figures, we conjecture that there is a non-linear curve which works as a sort of boundary with the following properties: (1) The optimal point (the method with the smallest error rate) can be found among the points on the boundary, (2) the points above the boundary tend to have larger error rates, and (3) no ensemble methods produce ensembles having the strength and correlation which is plotted below the boundary.

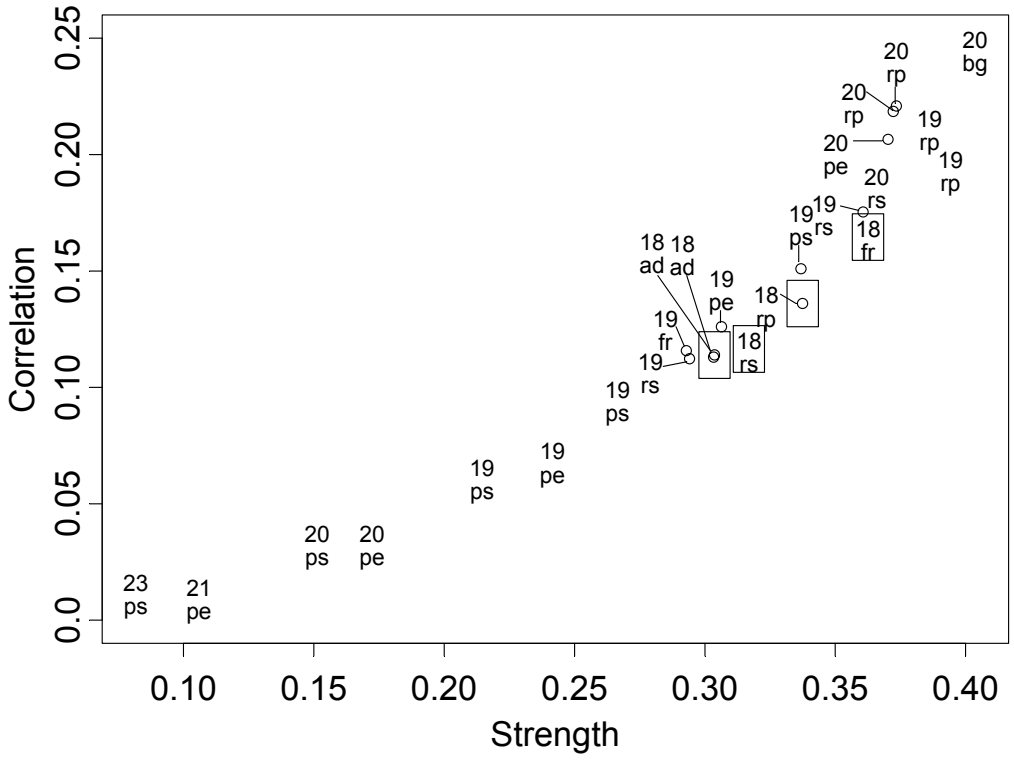

Figure V-1: Strength and Correlation (Waves)

The numbers in the figure are the error rate in percent. The corresponding ensemble method is shown under each error rate [ad = Adaboost, bg = bagging, fr = random forests, pe = perturbation, ps = perturbation with sub-sampling, rp = randomized splitting (and permutation stopping), rs = randomized splitting (and permutation stopping) with sub-sampling]. The points which produce the smallest error rate are indicated by rectangular boxes. The strength and correlation seem to have a non-linear relationship.

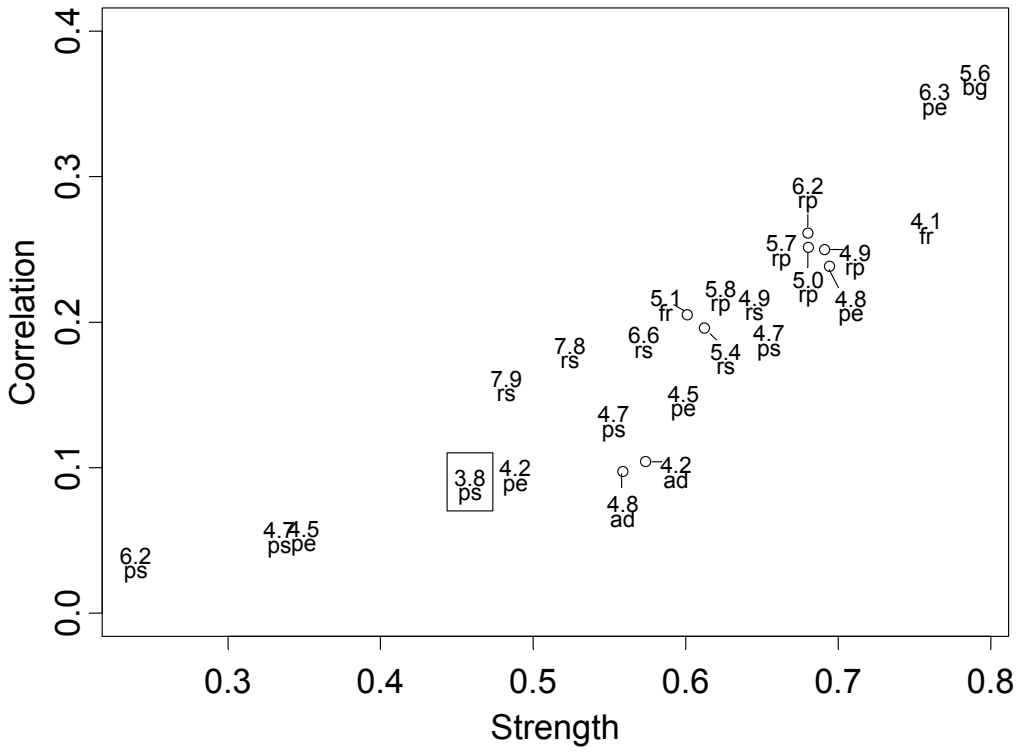

Figure V-2: Strength and Correlation (Image)

The numbers indicate the error rates in percent. The corresponding ensemble method is shown under each error rate. The point which produces the smallest error rate is indicated by a rectangular box. The relationship still seems to be non-linear, but the points form a small cloud rather than a non-linear curve. The points in upper-left side of the cloud tend to have higher error rates than the points in the lower-right side of the cloud.

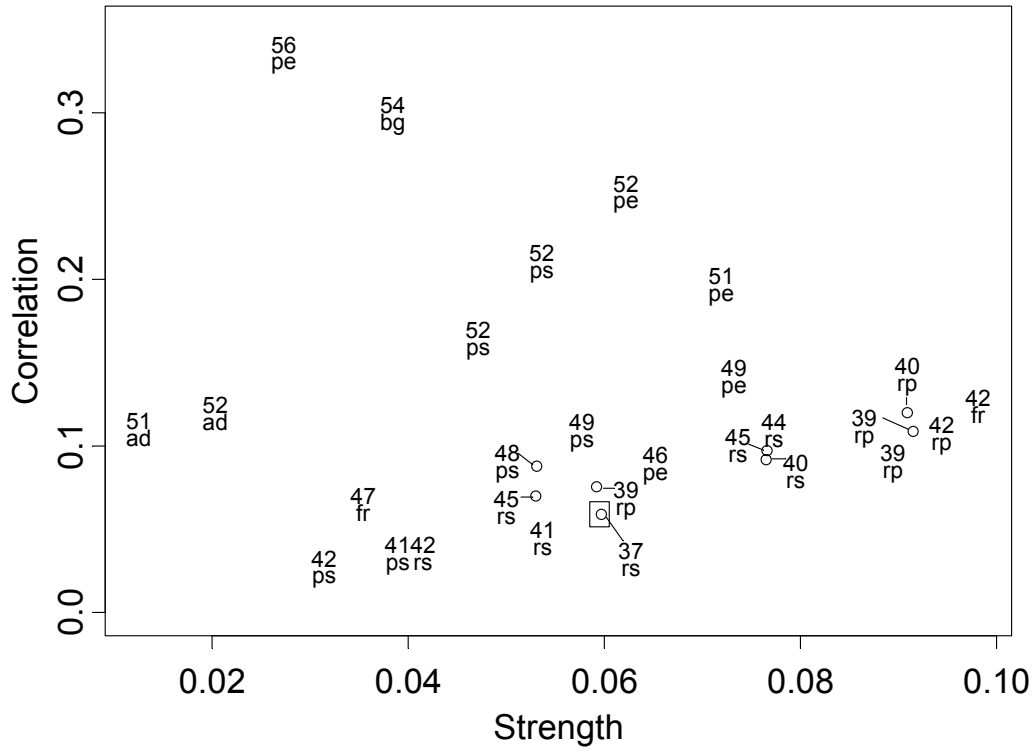

Figure V-3: Strength and Correlation (Vowel)

The numbers indicate the error rates in percent. The corresponding ensemble method is shown under each error rate. The point which produces the smallest error rate is indicated by a rectangular box. Toward the upper-left corner, the error rate tends to get larger.

E. WEIGHTED AND UNWEIGHTED VOTES IN ADABOOST

When using Adaboost, the final classifier is usually given by a weighted vote over classifiers in the ensemble. In our simulation, test set error estimates by an unweighted vote are also computed for comparison. They are displayed in the second and forth columns in Table V-9. Surprisingly, the error rates are not significantly different between weighted and unweighted votes. We have not seen this reported in other research and the reason is still not clear.

Data	AdaBoost.m1 (No pruning) Weighted	AdaBoost.m1 (No pruning) Unweighted	AdaBoost.m1 (Pruning) Weighted	AdaBoost.m1 (Pruning) Unweighted
Biopsy	0.0311	0.0326	0.0310	0.0339
Diabetes	0.2652	0.2625	0.2640	0.2601
Ecoli	0.1662	0.1635	0.1798	0.1837
German credit	0.2320	0.2351	0.2424	0.2415
Glass	0.2210	0.2210	0.2425	0.2425
Ionosphere	0.0659	0.0631	0.0666	0.0671
Liver	0.2981	0.3046	0.3070	0.3013
Sonar	0.1525	0.1562	0.1627	0.1731
Vehicle	0.2298	0.2287	0.2194	0.2184
Votes	0.0398	0.0398	0.0485	0.0458
Image	0.0419	0.0419	0.0481	0.0476
Vowel	0.5173	0.5173	0.5087	0.5065
Waves	0.1820	0.1813	0.1767	0.1760

Table V-9: Error Rate Estimates of Adaboost by Weighted and Unweighted Votes. Test set error by weighted and unweighted votes are not significantly different.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

We have proposed two ensemble methods: randomized splitting with permutation stopping and perturbation. We applied these ensemble methods to 13 frequently-used data sets in research on ensembles or classification, and estimated the error rates. Test set estimation was used for three data sets with the test sets and five-fold cross-validation was used for the other sets. Bagging, Adaboost and random forests were also run on the same data sets for comparison.

For randomized splitting with permutation stopping, five different values of permutation were tried. The results show that randomized splitting with permutation stopping performs at least as well as other ensemble methods such as Adaboost. For some data sets, our method gives the lowest error rates.

We tried a slightly different perturbation scheme from the one by Breiman (2000). Five different perturbation rates for the output were tried with all data sets. The results suggest that perturbation works as well as other methods on average. Additional computation with the Vowel data indicates that, the accuracy of this method could be improved by perturbing some input variables and combining more trees.

A. FUTURE RESEARCH

Our experiment shows that the randomized ensemble methods proposed in this dissertation can construct ensembles that are as good as well-known ensemble methods such as Adaboost. One of the shortcomings of these methods is that we do not know a good way to find good parameters. For example, we do not know how many times we need to permute the outputs or what a good perturbation rate for the variables is. More importantly, we do not even know how many trees we should build to construct a good classifier. As mentioned in Section V.C, we have explored combining sub-sampling with the proposed ensemble methods. The encouraging results suggest that combining sub-sampling with ensemble methods might be useful in finding good values of parameters like perturbation rate or number of permutations. It may furthermore help in finding a

good ensemble method to use for a given data set. More exploration of sub-sampling is needed.

We have applied perturbation only to classification trees. It might be interesting to see if perturbation works well with other classification algorithms such as neural networks.

As mentioned in Section V.A, the permutation stopping rule in randomized splitting is computationally expensive since it permutes the output variable at each node. Without the permutation stopping rule, we might be able to save computational time when constructing an ensemble; however, the resulting individual trees are much larger in size than those created with the stopping rule. The error rates of ensembles constructed without the permutations stopping rule and those constructed with one permutation are not significantly different for most of the data sets we have used. Even though each tree can predict a new observation quickly, it may take some time for an ensemble consisting of a large number of large-sized trees to predict a massive data set. Finding a good threshold value, to let us stop splitting a node in a less expensive way, would reduce computational time for constructing ensembles. A good threshold value would also prevent us from constructing unnecessarily large trees, which would in turn help reduce computational time for prediction.

The plots of strength and correlation show interesting relationships between them. We give a conjecture, but have not established any theoretical results. Using base classifiers other than classification trees might help to get more insight into strength and correlation.

For a given data set, which ensemble method should be applied? Researchers have attempted to categorize features of a data set by which we can determine what type of ensemble method should be used. For example, some researchers have observed that Adaboost does not perform well in situations where there is a large amount of classification noise, that is, there are many observations with incorrect class labels. Thus, Adaboost should not been used in such a situation. Breiman (2001) suggests random forests might be able to work for data sets with many weak inputs. This type of data is

becoming more common and difficult for the usual classifiers. We are not aware of any complete answer to the question of which ensemble method will be effective on a given data set. This is still an open question in research on ensemble methods as well as in classification generally.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DATA SETS

The data sets used in our study are summarized in Table IV-1. All but the Waves data were downloaded from the UC Irvine Data Repository (Blake *et al*., 1998). The Waves data were generated using the rule described in Breiman *et al.* (1984). This appendix gives more details of these data and shows a few examples of each data set. In the tables below, the class variables are shown in bold face.

1. **Biopsy**. This Wisconsin Breast Cancer Database was originally developed by Dr. William H. Wolberg at the University of Wisconsin Hospitals, Madison. Several studies such as Mangasarian and Wolberg (1990) have used this dataset. We downloaded the data from the UC Irvine Data Repository (Blake *et al*., 1998). The data consists of 699 observations from two classes: *benign* and *malignant*, with nine covariate variables. The task is to predict if the result of biopsy is *benign* or *malignant* given a set of medical test results. Each of the nine covariates has integer values raging from one to ten. The descriptions of the covariates are as follows:

V1: Clump Thickness
V2: Uniformity of Cell Size
V3: Uniformity of Cell Shape
V4: Marginal Adhesion
V5: Single Epithelial Cell Size
V6: Bare Nuclei
V7: Bland Chromatin
V8: Normal Nucleoli
V9: Mitoses

There are 16 observations with missing values and they were omitted in our study. The class distribution among the remaining 683 observation is 444 of *benign* and 239 of *malignant*. The first ten observations are shown in Table A-1.

2. **Diabetes**. This Pima Indians Diabetes Database was also downloaded from the UC Irvine Data Repository (Blake *et al*., 1998). There are 768 observations from two classes: 0 and 1, where class 1 indicates "tested positive for diabetes." All observations in the data are from female patients at least 21 years old of Pima Indian heritage, living near Phoenix, Arizona, USA. We want to predict whether a patient would show sign of

diabetes according to the World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose is at least 200 mg/dl at any survey examination or if found during routine medical care) given the set of measurements below:

Preg : Number of times pregnant
Plasma: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
Bp : Diastolic blood pressure (mm Hg)
Skin : Triceps skin fold thickness (mm)
Serum : 2-Hour serum insulin (mu U/ml)
Mass : Body mass index (weight in kg/(height in m)^2)
Pedigree: Diabetes pedigree function
Age : Age (years)

These eight covariates are all numerical. There are 500 observations from class 0 and 268 from class 1. The first 10 observations of the data are shown in Table A-2. As we can see in the table, this data set has many strange zeros where we would not expect zeros. For example, *Mass* for the 10th observation (the last row in the table) is 0.0, which is impossible in the real world since there does not exist such a person whose weight is zero kilogram. In our study, we used the data set without any treatment to these strange zeros.

3. **Ecoli**. The Protein Localization Sites Data was also obtained from the UC Irvine Data Repository (Blake *et al*., 1998). The first ten observations are shown in Table A-3. This data contains 336 observations from eight classes with seven covariates given below:

mcg : McGeoch's method for signal sequence recognition
gvh : von Heijne's method for signal sequence recognition
lip : von Heijne's Signal Peptidase II consensus sequence score (binary)
chg : Presence of charge on N-terminus of predicted lipoproteins (binary)
aac : score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins
alm1 : score of the ALOM membrane spanning region prediction program
alm2 : score of ALOM program after excluding putative cleavable signal regions from the sequence

Among these covariates, two binary variables are treated as continuous in our study. The task of this data is to predict the localization site of protein given these seven measurements. The detail of the classes and their distribution are as follows:

cp : cytoplasm 143
im : inner membrane without signal sequence 77

pp	: perisplasm	52
imU	: inner membrane, uncleavable signal sequence	35
om	: outer membrane	20
omL	: outer membrane lipoprotein	5
imL	: inner membrane lipoprotein	2
imS	: inner membrane, cleavable signal sequence	2

4. **German credit**. The German Credit data can also be found at the UC Irvine Data Repository (Blake *et al*., 1998) in two different forms. The original one provided by Professor Hans Hofmann, Universität Hamburg, contains 20 covariates (7 numerical, 13 categorical). For the algorithms that needed numerical attributes, several indicator variables were added and several ordered covariates were coded as integer. This new version with 24 numerical covariates is used in our study since this version has been used in some other research such as Breiman (2001) or Friedman *et al.* (2000). There are 1,000 observations from two classes: 1 (good) and 2 (bad). Among them, 700 observations are from class 1 and 300 from class 2. The covariates in the original data are (1) status of existing checking account, (2) duration of current account in month, (3) credit history, (4) purpose (e.g. car, furniture), (5) credit amount, (6) savings account/bonds, (7) length of present employment, (8) installment rate in percentage of disposable income, (9) personal status and sex (e.g. single, divorced), (10) other debtors/guarantors, (11) length at present residence, (12) property (e.g. real estate, car), (13) age in years, (14) other installment plans (e.g. bank, stores), (15) housing (e.g. rent, own), (16) number of existing credits at this bank, (17) job, (18) number of people being liable to provide maintenance for, (19) telephone and (20) foreign worker. This dataset is provided with a cost matrix, but we do not use the cost to construct classifiers. Table A-4 shows the first ten observations of the data.

5. **Glass**. The Glass Identification Database were also downloaded from the UC Irvine Data Repository (Blake *et al*., 1998). The data contain 214 observations from six classes with nine continuous covariates. The objective here is to identify the type of glass given a set of chemical measurements. The details of the input variables are given below:

RI : Refractive Index
Na : Sodium
Mg : Magnesium

Al : Aluminum
Si : Silicon
K : Potassium
Ca : Calcium
Ba : Barium
Fe : Iron

Except *RI*, the unit of measurement is weight, by percent, of the corresponding oxide. As an interesting background of this dataset, the study of classification of glass types was motivated by criminological investigation. If the glass left at the scene of the crime is classified correctly, it can be used as evidence. The distribution of each class is as follows:

1: building windows, float processed	70
2: building windows, non-float processed	76
3: vehicle windows, float processed	17
5: containers	13
6: tableware	9
7: headlamps	29

In the definition of the dataset, there exists class *4* which indicates "vehicle windows, non-float processed." However, there are no observations from class *4* in the given dataset and thus class *4* is not listed above. The first ten observations are given in Table A-5.

6. **Ionosphere**. The Johns Hopkins University Ionosphere database can be found at the UC Irvine Data Repository (Blake *et al*., 1998). The radar data were collected by a system in Goose Bay, Labrador, Canada. The targets were free electrons in the ionosphere. There are 351 observations from two classes: *g* (good) and *b* (bad), with 34 continuous covariates. Class *g* indicates "good" radar returns that show evidence of some type of structure in the ionosphere. "Bad" radar returns indicated as class *b* do not show any evidence, that is, they pass through the ionosphere. Two covariates are used per pulse number, corresponding to the complex values given by processing received signals using an autocorrelated function that arguments are the time of a pulse and the pulse number. There are 17 pulse numbers for the Goose Bay system and hence 34 covariate variables in the data. The data consists of 126 observations from class *b* and 225 from class *g*. The first five observations are shown in Table A-6.

7. **Liver**. The BUPA Liver Disorders Dataset was also obtained from the UC Irvine Data Repository (Blake *et al*., 1998). There are 345 observations from two classes: class 1 and 2, with six continuous covariates. Each observation is the record of a single male individual. The first five covariates are the results of blood tests that are thought to be sensitive to liver disorders maybe caused by excessive alcohol consumption. The details of covariates are as follows:

mcv : mean corpuscular volume
alkphos : alkaline phosphotase
sgpt : alamine aminotransferase
sgot : aspartate aminotransferase
gammagt : gamma-glutamyl transpeptidase
drinks : number of half-pint equivalents of alcoholic beverages drunk per day

The datasets contains 145 observations from class 1 and 200 from class 2. Table A-7 shows the first 10 observations of this data.

8. **Sonar**. This dataset can be found under Undocumented Databases at the UC Irvine Data Repository (Blake *et al*., 1998). The data were obtained by bouncing sonar signals off metal cylinders and roughly cylindrical rocks at various angles and under various conditions. There are 208 observations from two classes with 60 continuous covariates ranging from 0.0 to 1.0. The task here is to determine if an object is a mine (metal cylinder) or a rock given a set of signals. The class labels for these two classes are *M* for mines and *R* for rocks. The dataset contains 111 observations from class *M* and 97 from class *R*. Each covariate represents the energy within a particular frequency band, integrated over a certain period of time. Table A-8 gives the first 10 observations of the data.

9. **Vehicle**. This dataset is originally from the Turing Institute, Glasgow, Scotland. We downloaded the data from the UC Irvine Data Repository (Blake *et al*., 1998). The data contain 846 observations from four classes: *OPEL*, *SAAB*, *BUS*, *VAN*, with 18 covariates. We want to classify a given silhouette as one of four types of vehicle given a set of covariates extracted from the silhouette. The original dataset contains 946 observations of which 240 belong to class *OPEL*, 240 *SAAB*, 240 *BUS* and 226 *VAN*. Since 100 observations are being kept for validation, we can use 846 observations with the distributions of 212 *OPEL*, 217 *SAAB*, 218 *BUS*, and 199 *VAN*. This data was

originally gathered at the Turing Institute in 1986-87 by JP Siebert using four "Corgie" model vehicles: a double decker bus, Cheverolet van, Saab 9000 and an Opel Manta 400. The original purpose was to find a method of distinguishing 3D objects within a 2D image. For each vehicle, a set of image silhouettes were obtained from a number of angles of rotations at three different angles of elevation. These images were processed to produce the following 18 covariates:

V1: compactness = (average perim)**2/area
V2: circularity = (average radius)**2/area
V3: distance circularity = area/(av.distance from border)**2
V4: radius ratio = (max.rad-min.rad)/av.radius
V5: pr.axis aspect ratio = (minor axis)/(major axis)
V6: max.length aspect ratio = (length perp. max length)/(max length)
V7: scatter ratio = (inertia about minor axis)/(inertia about major axis)
V8: elongatedness = area/(shrink width)**2
V9: pr.axis rectangularity = area/(pr.axis length*pr.axis width)
V10: max.length rectangularity = area/(max.length*length perp. to this)
V11: scaled variance along major axis
= (2nd order moment about minor axis)/area
V12: scaled variance along minor axis
= (2nd order moment about major axis)/area
V13: scaled radius of gyration= (mavar+mivar)/area
V14: skewness about major axis
= (3rd order moment about major axis)/sigma_min**3
V15: skewness about minor axis
= (3rd order moment about minor axis)/sigma_maj**3
V16: kurtosis about minor axis
= (4th order moment about major axis)/sigma_min**4
V17: kurtosis about major axis
= (4th order moment about minor axis)/sigma_maj**4
V18: hollows ratio = (area of hollows)/(area of bounding polygon),

where sigma_maj**2 is the variance along the major axis and sigma_min**2 is the variance along the minor axis. The area of hollows is given by (area of hollows) = (area of bounding poly-area of object). More details about this dataset can be found at the UC Irvine Data Repository. The first 10 observations are shown in Table A-9.

10. **Votes**. This 1984 United States Congressional Voting Records Database were also obtained from the UC Irvine Data Repository (Blake *et al*., 1998). The data include votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac (CQA). For each key vote, there are

nine different types of votes simplified to three categories: (1) voted for, paired for, and announced for (these are simplified to "yea"), (2) voted against, paired against, and announced against (simplified to "nay"), (3) voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (simplified to an unknown deposition). We denote "yea" by "y," "nay" by "n," and "unknown" by "ne." There are 435 observations from two classes: *democrats* (267 observations) and *republicans* (168). The objective here is to predict the party (class) of a given representative given the results of 16 key votes given below:

V2 : handicapped infants
V3 : water project cost sharing
V4 : adoption of the budget resolution
V5 : physician fee freeze
V6 : el-salvador aid
V7 : religious groups in schools
V8 : anti-satellite test ban
V9 : aid to Nicaraguan contras
V10 : mx missile
V11 : immigration
V12 : synfuels corporation cutback
V13 : education spending
V14 : superfund right to sue
V15 : crime
V16 : duty free exports
V17 : export administration act South Africa

The first 10 observations of the data are shown in Table A-10.

11. **Image**. The Image Segmentation data were also obtained from the UC Irvine Data Repository (Blake *et al*., 1998). The data have the training set of 300 observations and the test set of 2,100 observations. There are seven classes: *brickface*, *sky*, *foliage*, *cement*, *window*, *path*, *grass*. The class distributions are 30 observations per class in the training set and 300 observations per class in the test set. These observations were drawn randomly from a database of 7 outdoor images that were hand segmented to create a classification for every pixel. Each observation is a 3x3 region. The details of the 19 input variables are as follows:

V2: region-centroid-col = the column of the center pixel of the region.
V3: region-centroid-row = the row of the center pixel of the region.
V4: region-pixel-count = the number of pixels in a region = 9.

V5: short-line-density-5 = the results of a line extractoin algorithm that counts how many lines of length 5 (any orientation) with low contrast, less than or equal to 5, go through the region.
V6: short-line-density-2 = same as short-line-density-5 but counts lines of high contrast, greater than 5.
V7: vedge-mean = measure the contrast of horizontally adjacent pixels in the region. The mean is given here and the standard deviation is given as *V8*. This attribute is used as a vertical edge detector.
V8: vegde-sd = (see *V7*)
V9: hedge-mean = measures the contrast of vertically adjacent pixels. The mean is given here and the standard deviation is given as *V10*. This attribute is used for horizontal line detection.
V10: hedge-sd = (see *V9*).
V11: intensity-mean = the average over the region of (R + G + B)/3
V12: rawred-mean = the average over the region of the R value.
V13: rawblue-mean = the average over the region of the B value.
V14: rawgreen-mean = the average over the region of the G value.
V15: exred-mean = measure the excess red = (2R - (G + B))
V16: exblue-mean = measure the excess blue = (2B - (G + R))
V17: exgreen-mean = measure the excess green = (2G - (R + B))
V18: value-mean = 3-d nonlinear transformation of RGB
V19: saturatoin-mean = (see *V18*)
V20: hue-mean = (see *V18*)

The first 5 observations are given in Table A-11.

12. **Vowel**. The Vowel data originally collected for speaker independent recognition of the eleven vowels of British English can be found at the UC Irvine Data Repository (Blake *et al*., 1998). The data were collected from fifteen individual speakers (eight male and seven female speakers), each saying each vowel six times. The 11 words used to record the vowels are as follows: heed (i), hid (I), head (E), had (A), hard (a:), hud (Y), hod (O), hoard (C:), hood (U), who'd (u:) and heard (3:). Here, the symbols in parentheses are (ASCII approximations to) the International Phonetic Association (I.P.A.) symbols. There are 990 observations in total. Among them, 528 observations recorded by four male and four female speakers were used as the training samples. The remaining 462 observations obtained from the other four male and three female speakers were used to create the test set. Each speech signal was processed to produce 10 input variables. In addition to these 10 variables, the gender of the speaker (column *female* in the table below) was also used as an input. Hence, there are 11 covariates for each observation. The variable *female* is binary but was treated as continuous in our study.

Friedman *et al.* (2000) used the training and test sets that we did. Breiman (2001) combined the training and test sets, and used the combined set as one training set. He set aside 10% of the set as the test set and built a random forest on the remaining data. This was repeated 100 times and the average test set error was reported. Thus, Breiman's results have much lower error rates than ours. Table A-12 shows the first 10 observations of the data.

13. **Waves**. This waveform data are introduced in Breiman *et al.* (1984). It is a three class problem based on the three waveforms in Figure A-1. In every graph, the horizontal axis indicates the variable numbers and the vertical axis indicates the values. For example, in the Waveform 1, variable 1 takes value 0 and variable 7 takes value 6.

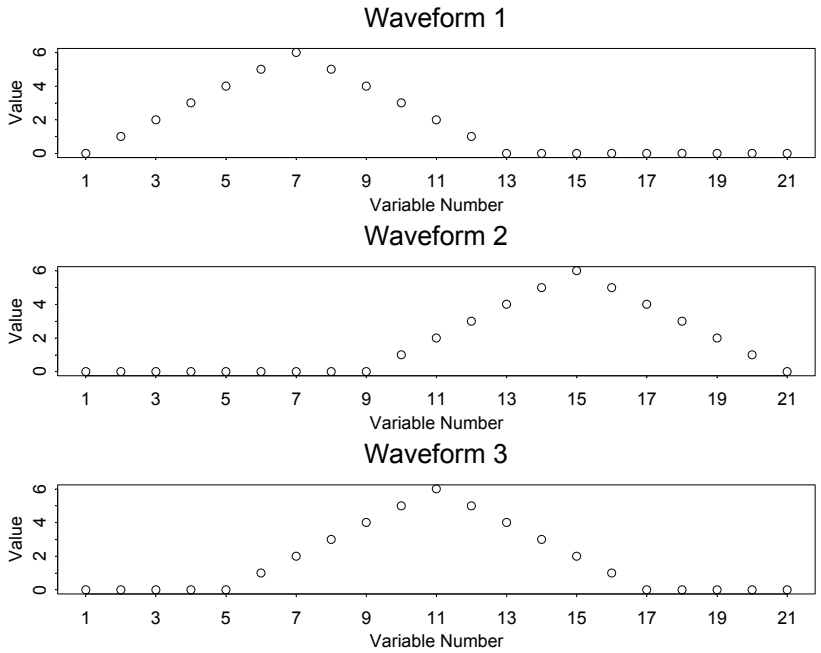

Figure A-1: Waveforms.

In each plot, the horizontal axis indicates the variable numbers and the vertical axis indicates the values of the variables. These three waveforms are used to create the input variables for each class. There are three classes named 0, 1, and 2. The input variables for class 0 are generated as random combinations of the inputs of waveforms 1 and 2 with independent noise added. Likewise, waveforms 1 and 3 are used for class 2, and 2 and 3 for class 2.

Each class is created as a random convex combination of two of these three waveforms with a random noise added. More precisely, for class 0, the values of the input variables 1, …, 21 are generated by

$$\begin{aligned}(\text{value of variable } j) &= u \times (\text{value of variable } j \text{ at waveform 1}) \\ &\quad + (1-u) \times (\text{value of variable } j \text{ at waveform 2}) + e_j,\end{aligned}$$

where u is drawn randomly from the uniform distribution on the range (0, 1) and e_j is drawn randomly from the normal distribution with mean 0 and variance 1. Likewise, waveforms 1 and 3 are used for class 1 and waveforms 2 and 3 for class 2. The datasets are generated with the prior probabilities of (1/3, 1/3, 1/3). Breiman *et al.* (1984) report that this data has the Bayes error rate of 0.14. A few examples of the waveforms data are given in Table A-13.

Again, the examples from each data are shown in the following tables:

Biopsy	: Table A-1
Diabetes	: Table A-2
Ecoli	: Table A-3
German credit	: Table A-4
Glass	: Table A-5
Ionosphere	: Table A-6
Liver	: Table A-7
Sonar	: Table A-8
Vehicle	: Table A-9
Votes	: Table A-10
Image	: Table A-11
Vowel	: Table A-12
Waves	: Table A-13

V1	V2	V3	V4	V5	V6	V7	V8	V9	**Class**
5	1	1	1	2	1	3	1	1	**Benign**
5	4	4	5	7	10	3	2	1	**Benign**
3	1	1	1	2	2	3	1	1	**Benign**
6	8	8	1	3	4	3	7	1	**Benign**
4	1	1	3	2	1	3	1	1	**Benign**
8	10	10	8	7	10	9	7	1	**malignant**
1	1	1	1	2	10	3	1	1	**Benign**
2	1	2	1	2	1	3	1	1	**Benign**
2	1	1	1	2	1	1	1	5	**Benign**
4	2	1	1	2	1	2	1	1	**Benign**

Table A-1: Example of Biopsy data.

There are two classes: *benign* and *malignant*. Nine input variables are all integer-valued with the range 1 – 10. Among 699 observations in the original dataset, 16 observations with missing values were removed in our study. There are 458 *benign* and 241 *malignant* in the original dataset, and 444 *benign* and 239 *malignant* in the data set we used.

Preg	Plasma	Bp	Skin	Serum	Mass	Pedigree	Age	**Diabetes**
6	148	72	35	0	33.6	0.627	50	**1**
1	85	66	29	0	26.6	0.351	31	**0**
8	183	64	0	0	23.3	0.672	32	**1**
1	89	66	23	94	28.1	0.167	21	**0**
0	137	40	35	168	43.1	2.288	33	**1**
5	116	74	0	0	25.6	0.201	30	**0**
3	78	50	32	88	31.0	0.248	26	**1**
10	115	0	0	0	35.3	0.134	29	**0**
2	197	70	45	543	30.5	0.158	53	**1**
8	125	96	0	0	0.0	0.232	54	**1**

Table A-2: Example of Diabetes Data.

There are two classes (0 and 1) where class 1 indicates "tested positive for diabetes." All input variables are numeric. The data contain 768 observations of which 500 observations are from class 0 and 268 from class 1.

Mcg	gvh	lip	Chg	aac	alm1	alm2	class
0.49	0.29	0.48	0.5	0.56	0.24	0.35	**cp**
0.07	0.40	0.48	0.5	0.54	0.35	0.44	**cp**
0.56	0.40	0.48	0.5	0.49	0.37	0.46	**cp**
0.59	0.49	0.48	0.5	0.52	0.45	0.36	**cp**
0.23	0.32	0.48	0.5	0.55	0.25	0.35	**cp**
0.67	0.39	0.48	0.5	0.36	0.38	0.46	**cp**
0.29	0.28	0.48	0.5	0.44	0.23	0.34	**cp**
0.21	0.34	0.48	0.5	0.51	0.28	0.39	**cp**
0.20	0.44	0.48	0.5	0.46	0.51	0.57	**cp**
0.42	0.40	0.48	0.5	0.56	0.18	0.30	**cp**

Table A-3: Example of Ecoli Data.

This data consists of 336 observations from eight classes. There are seven input variables including two binary variables. All input variables are treated as continuous.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	class
1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	**1**
2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	**2**
4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	**1**
1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	1	**1**
1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	1	**2**
4	36	2	91	5	3	3	4	4	35	3	1	2	2	1	0	0	1	0	0	0	0	1	0	**1**
4	24	2	28	3	5	3	4	2	53	3	1	1	1	1	0	0	1	0	0	1	0	0	1	**1**
2	36	2	69	1	3	3	2	3	35	3	1	1	2	1	0	1	1	0	1	0	0	0	0	**1**
4	12	2	31	4	4	1	4	1	61	3	1	1	1	1	0	0	1	0	0	1	0	1	0	**1**
2	30	4	52	1	1	4	2	3	28	3	2	1	1	1	1	0	1	0	0	1	0	0	0	**2**

Table A-4: Example of German Credit Data.

This data contain 1,000 observations of which 700 are from class 1 (indicates "good") and 300 from class 2 ("bad"). There are 24 numerical covariates.

Type	RI	Na	Mg	Al	Si	Potassium	Ca	Ba	Fe
1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0	0.00
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0	0.00
1	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0.00
1	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0	0.00
1	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0	0.00
1	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26
1	1.51743	13.30	3.60	1.14	73.09	0.58	8.17	0	0.00
1	1.51756	13.15	3.61	1.05	73.24	0.57	8.24	0	0.00
1	1.51918	14.04	3.58	1.37	72.08	0.56	8.30	0	0.00
1	1.51755	13.00	3.60	1.36	72.99	0.57	8.40	0	0.11

Table A-5: Example of Glass Data.

There are 214 observations from six classes with nine chemical measurements.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760
1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549
1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198
1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000
1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399

V11	V12	V13	V14	V15	V16	V17	V18	V19
0.85243	-0.17755	0.59755	-0.44945	0.60536	-0.38223	0.84356	-0.38542	0.58212
0.50874	-0.67743	0.34432	-0.69707	-0.51685	-0.97515	0.05499	-0.62237	0.33109
0.73082	0.05346	0.85443	0.00827	0.54591	0.00299	0.83775	-0.13644	0.75535
0.00000	0.00000	0.00000	0.00000	-1.00000	0.14516	0.54094	-0.39330	-1.00000
0.52798	-0.20275	0.56409	-0.00712	0.34395	-0.27457	0.52940	-0.21780	0.45107

V20	V21	V22	V23	V24	V25	V26	V27	V28
-0.32192	0.56971	-0.29674	0.36946	-0.47357	0.56811	-0.51171	0.41078	-0.46168
-1.00000	-0.13151	-0.45300	-0.18056	-0.35734	-0.20332	-0.26569	-0.20468	-0.18401
-0.08540	0.70887	-0.27502	0.43385	-0.12062	0.57528	-0.40220	0.58984	-0.22145
-0.54467	-0.69975	1.00000	0.00000	0.00000	1.00000	0.90695	0.51613	1.00000
-0.17813	0.05982	-0.35575	0.02309	-0.52879	0.03286	-0.65158	0.13290	-0.53206

V29	V30	V31	V32	V33	V34	**class**
0.21266	-0.34090	0.42267	-0.54487	0.18641	-0.45300	**g**
-0.19040	-0.11593	-0.16626	-0.06288	-0.13738	-0.02447	**b**
0.43100	-0.17365	0.60436	-0.24180	0.56045	-0.38238	**g**
1.00000	-0.20099	0.25682	1.00000	-0.32382	1.00000	**b**
0.02431	-0.62197	-0.05707	-0.59573	-0.04608	-0.65697	**g**

Table A-6: Example of Ionosphere Data.

There are two classes: *g* (good) and *b*(bad). The data of size 341 consist of 126 *bad* and 225 *good*. All of 34 attributes are continuous.

Mcv	alkphos	sgpt	sgot	gammagt	Drinks	**class**
85	92	45	27	31	0.0	**1**
85	64	59	32	23	0.0	**2**
86	54	33	16	54	0.0	**2**
91	78	34	24	36	0.0	**2**
87	70	12	28	10	0.0	**2**
98	55	13	17	17	0.0	**2**
88	62	20	17	9	0.5	**1**
88	67	21	11	11	0.5	**1**
92	54	22	20	7	0.5	**1**
90	60	25	19	5	0.5	**1**

Table A-7: Example of Liver Data.

There are 345 observations of which 145 observations belong to class 1 and 200 to class 2. The six covariates are all continuous.

X1	V1	V2	V3	V4	V5	···	V59	V60
R	0.0200	0.0371	0.0428	0.0207	0.0954	···	0.0090	0.0032
R	0.0453	0.0523	0.0843	0.0689	0.1183	···	0.0052	0.0044
R	0.0262	0.0582	0.1099	0.1083	0.0974	···	0.0095	0.0078
R	0.0100	0.0171	0.0623	0.0205	0.0205	···	0.0040	0.0117
R	0.0762	0.0666	0.0481	0.0394	0.0590	···	0.0107	0.0094
R	0.0286	0.0453	0.0277	0.0174	0.0384	···	0.0051	0.0062
R	0.0317	0.0956	0.1321	0.1408	0.1674	···	0.0036	0.0103
R	0.0519	0.0548	0.0842	0.0319	0.1158	···	0.0048	0.0053
R	0.0223	0.0375	0.0484	0.0475	0.0647	···	0.0059	0.0022
R	0.0164	0.0173	0.0347	0.0070	0.0187	···	0.0056	0.0040

Table A-8: Example of Sonar Data.

The data contain 208 observations (111 *M* and 97 *R*). There are 60 continuous covariates with the range of 0.0 – 1.0.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	**class**
95	48	83	178	72	10	162	42	20	159	176	379	184	70	6	16	187	197	**van**
91	41	84	141	57	9	149	45	19	143	170	330	158	72	9	14	189	199	**van**
104	50	106	209	66	10	207	32	23	158	223	635	220	73	14	9	188	196	**saab**
93	41	82	159	63	9	144	46	19	143	160	309	127	63	6	10	199	207	**van**
85	44	70	205	103	52	149	45	19	144	241	325	188	127	9	11	180	183	**bus**
107	57	106	172	50	6	255	26	28	169	280	957	264	85	5	9	181	183	**bus**
97	43	73	173	65	6	153	42	19	143	176	361	172	66	13	1	200	204	**bus**
90	43	66	157	65	9	137	48	18	146	162	281	164	67	3	3	193	202	**van**
86	34	62	140	61	7	122	54	17	127	141	223	112	64	2	14	200	208	**van**
93	44	98	197	62	11	183	36	22	146	202	505	152	64	4	14	195	204	**saab**

Table A-9: Example of Vehicle Data.

All covariates are continuous and produced by processing the picture images of four cars: *OPEL*, *SAAB*, *BUS*, *VAN*. The data contain 846 observations (212 *OPEL*, 217 *SAAB*, 218 *BUS* and 199 *VAN*).

Class	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17
Republican	n	y	n	y	Y	y	n	n	n	y	ne	y	y	y	n	y
Republican	n	y	n	y	Y	y	n	n	n	n	n	y	y	y	n	ne
Democrat	ne	y	y	ne	Y	y	n	n	n	n	y	n	y	y	n	n
Democrat	n	y	y	n	Ne	y	n	n	n	n	y	n	y	n	n	y
Democrat	y	y	y	n	Y	y	n	n	n	n	y	ne	y	y	y	y
Democrat	n	y	y	n	Y	y	n	n	n	n	n	n	y	y	y	y
Democrat	n	y	n	y	Y	y	n	n	n	n	n	n	ne	y	y	y
Republican	n	y	n	y	Y	y	n	n	n	n	n	n	y	y	ne	y
Republican	n	y	n	y	Y	y	n	N	n	n	n	y	y	y	n	y
Democrat	y	y	y	n	N	n	y	Y	y	n	n	n	n	n	ne	ne

Table A-10: Example of Votes Data.

This data shows the votes for each of the U.S. House of Representatives Congressmen on the 16 key votes. Each observation consists of the party (*republican* or *democrat*) of a representative and the votes on each of 16 key votes. There are 435 observations: 267 *democrat* and 168 *republican*. All covariates are categorical.

Class	V2	V3	V4	V5	V6	V7	V8	V9	V10
BRICKFACE	140	125	9	0.0000000	0	0.2777779	0.06296301	0.6666667	0.31111118
BRICKFACE	188	133	9	0.0000000	0	0.3333333	0.26666674	0.5000000	0.07777774
BRICKFACE	105	139	9	0.0000000	0	0.2777778	0.10740744	0.8333333	0.52222216
BRICKFACE	34	137	9	0.0000000	0	0.5000002	0.16666673	1.1111110	0.47407418
BRICKFACE	39	111	9	0.0000000	0	0.7222223	0.37407416	0.8888889	0.42962950

V11	V12	V13	V14	V15	V16	V17	V18
6.185185	7.333334	7.666666	3.555556	3.4444444	4.444445	-7.888889	7.777778
6.666666	8.333334	7.777778	3.888889	5.0000000	3.333333	-8.333333	8.444445
6.111111	7.555555	7.222222	3.555556	4.3333335	3.333333	-7.666666	7.555555
5.851852	7.777778	6.444445	3.333333	5.7777777	1.777778	-7.555555	7.777778
6.037037	7.000000	7.666666	3.444444	2.8888888	4.888889	-7.777778	7.888889

V19	V20
0.5456349	-1.1218182
0.5385802	-0.9248173
0.5326279	-0.9659458
0.5736331	-0.7442716
0.5629188	-1.1757725

Table A-11: Example of Image Data.

There are seven classes with 19 continuous covariates. The size of the training set is 300 and that of the test set is 3,000. Both sets contain equal number of observations per class.

class	female	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
0	0	-3.639	0.418	-0.670	1.779	-0.168	1.627	-0.388	0.529	-0.874	-0.814
1	0	-3.327	0.496	-0.694	1.365	-0.265	1.933	-0.363	0.510	-0.621	-0.488
2	0	-2.120	0.894	-1.576	0.147	-0.707	1.559	-0.579	0.676	-0.809	-0.049
3	0	-2.287	1.809	-1.498	1.012	-1.053	1.060	-0.567	0.235	-0.091	-0.795
4	0	-2.598	1.938	-0.846	1.062	-1.633	0.764	0.394	-0.150	0.277	-0.396
5	0	-2.852	1.914	-0.755	0.825	-1.588	0.855	0.217	-0.246	0.238	-0.365
6	0	-3.482	2.524	-0.433	1.048	-1.995	0.902	0.322	0.450	0.377	-0.366
7	0	-3.941	2.305	0.124	1.771	-1.815	0.593	-0.435	0.992	0.575	-0.301
8	0	-3.860	2.116	-0.939	0.688	-0.675	1.679	-0.512	0.928	-0.167	-0.434
9	0	-3.648	1.812	-1.378	1.578	0.065	1.577	-0.466	0.702	0.060	-0.836

Table A-12: Example of Vowel Data.

The data were collected from 15 individuals, with each individual saying each of 11 vowels 6 times. The data collected from 8 individuals were used to construct the classifiers and the data from the other 7 individuals were used to test the classifiers. In our simulation, one binary variable (*female*) was treated as continuous.

Classes	X1.2	X1.3	X1.4	X1.5	X1.6	X1.7	X1.8
2	0.59913600	-1.5988016	1.03781911	-0.9744512	-1.4788742	0.5076652	1.4807339
2	1.17882006	-1.6733797	-0.86311536	-1.0215469	0.3966210	-0.1506631	2.2128171
2	-0.17101014	0.7357474	-0.24514848	-1.0475136	0.9568550	2.2466765	2.5832400
2	0.44012650	0.2931246	0.08967792	1.0105115	0.2255792	0.8485877	1.6187344
0	-1.23070143	0.8685622	-1.06699036	-0.7406484	-0.6715534	0.4975041	2.2066389

X1.9	X1.10	X1.11	X1.12	X1.13	X1.14	X1.15
2.6650686	3.9770633	4.624737	5.490596	4.791417	2.655197	3.9561689
4.4392521	4.8371736	3.173975	6.613343	3.513606	4.080843	2.5008874
3.0780681	3.4131326	5.268698	4.466209	5.533505	4.196308	3.8052033
2.8062179	2.3718326	4.207450	5.535619	5.326545	2.337126	3.0735455
1.6797110	0.9782114	2.389071	2.701694	2.182613	3.688760	4.8192319

X1.16	X1.17	X1.18	X1.19	X1.20	X1.21	X1.22
3.931763	2.538280	0.5688606	0.888531226	-0.3976138	-0.29674828	-0.2064987
3.322939	0.502316	1.1480119	-0.678875092	2.4612141	0.80033821	-1.0166428
3.852213	2.546429	1.3335961	1.095506429	-0.6996193	1.31534282	-0.7889561
2.925911	1.571524	2.2573636	0.657779661	1.9059963	0.19702150	0.1667793
3.076204	3.589269	3.5573838	1.501688016	0.2210232	-0.51249594	1.7482087

Table A-13: Example of Waves Data.

This waveforms data are introduced in Breiman *et al.* (1984). We generated a training set of size 300 and a test set of size 1,500.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. COMPUTATION OF CORRELATION

Breiman (2001) shows how to compute the out-of-bag estimates of strength and correlation in Appendix II. In our computation, we follow his appendix to compute the strength and correlation of ensembles. However, we consider the equation (A2) given in the appendix may be a typographical error and thus use the equation shown below. As in Section II.C, $h(\cdot, \theta)$ is a classifier constructed with the training set and θ, and $h(\boldsymbol{x}, \theta_m)$ is its prediction at $X = \boldsymbol{x}$, where the training set is drawn randomly from the distribution of the random vector X, Y and θ is randomly drawn from the distribution of the random vector Θ.

In Breiman (2001), the following equation (A3) is derived to compute $sd(\Theta)$:

$$sd(\Theta) = [p_1 + p_2 + (p_1 - p_2)^2]^{1/2} ,$$

where $p_1 = E_{\boldsymbol{X}, Y} I[h(X, \Theta) = y]$ and $p_2 = E_{\boldsymbol{X}, Y} I[h(X, \Theta) = c(X, Y)]$.

However, equation (A.3) should be

$$sd(\Theta) = [p_1 + p_2 - (p_1 - p_2)^2]^{1/2} .$$

In these equations above, $sd(\Theta)$ is the standard deviation of the row margin function $rmg(\Theta, X, Y)$ holding Θ fixed. The row margin function is defined by

$$rmg(\Theta, X, Y) = I[h(X, \Theta) = Y] - I[h(X, \Theta) = c(X, Y)],$$

where $c(X, Y) = \underset{c \in \{c_k | c_k \neq y\}}{\arg\max} P_{\Theta}[h(X, \Theta) = c]$. Note that the notation is slightly different from that in Breiman (2001). The variance of the raw margin function $rmg(\Theta, X, Y)$ is

$$\begin{aligned}
&\mathrm{var}_{\boldsymbol{X},Y} [rmg(\Theta, X, Y)] \\
&\quad = \mathrm{var}_{\boldsymbol{X},Y} \{ I[h(X, \Theta) = Y] - I[h(X, \Theta) = c(X, Y)] \} \\
&\quad = \mathrm{E}_{\boldsymbol{X},Y} \{ I[h(X, \Theta) = Y] - I[h(X, \Theta) = c(X, Y)] \}^2 \\
&\quad - (\mathrm{E}_{\boldsymbol{X},Y} \{ I[h(X, \Theta) = Y] - I[h(X, \Theta) = c(X, Y)] \})^2
\end{aligned}$$

$$= \mathrm{E}_{\boldsymbol{X},Y} [\ \{\ I[h(X,\ \Theta) = Y]\ \}^2 - 2\ I[h(X,\ \Theta) = Y]\ I[h(X,\ \Theta) = c(X,\ Y)]$$
$$+\ \{\ I[h(X,\ \Theta) = c(X,\ Y)]\ \}^2\ ]$$
$$-\ \{\ \mathrm{E}_{\boldsymbol{X},Y}\ I[h(\boldsymbol{x},\ \Theta) = Y]\ - \mathrm{E}_{\boldsymbol{X},Y}\ I[h(X,\ \Theta) = c(X,\ Y)]\ \}^2.$$

Since $I(\cdot)$ is the indicator function,

$$[I(\cdot)]^2 = I(\cdot),$$

$$I[h(X,\ \Theta) = Y]\cdot I[h(X,\ \Theta) = c(X,\ Y)] = 0.$$

Thus

$$\mathrm{var}_{\boldsymbol{X},Y}\ (rmg(\Theta,\ X,\ Y))$$
$$= \mathrm{E}_{\boldsymbol{X},Y}\ \{\ I[h(X,\ \Theta) = Y\ ] + I[h(X,\ \Theta) = c(X,\ Y)]\ \}$$
$$-\ \{\ \mathrm{E}_{\boldsymbol{X},Y}\ I[h(X,\ \Theta) = Y\ ]\ - \mathrm{E}_{\boldsymbol{X},Y}\ I[h(X,\ \Theta) = c(X,\ Y)]\ \}^2$$
$$= p_1 + p_2 - (p_1 - p_2)^2.$$

Hence,

$$sd(\Theta) = [p_1 + p_2 - (p_1 - p_2)^2]^{1/2}\ .$$

We use this equation instead of the equation (A2) in Breiman (2001, p.31) to compute the standard deviation $sd(\Theta)$ in our simulation. Section IV.C describes how we actually estimate strength and correlation.

APPENDIX C. STANDARD DEVIATION OF ENSEMBLE ERROR RATES

To compute the standard errors of the ensemble error rate, we produce several ensembles with the same size as in our study, using different random number seeds. We use the data sets with the test sets to save the computational time. For each data set, we construct 10 sets of ensembles each of size 100 using the different random number seeds, each time estimating the ensemble error rate, average size and error rate of the individual trees, strength and correlation on the test set. We use randomized splitting with one permutation. The results are shown in the tables below.

Run no.	No. of misclass	Test set size	Error rate	Average size	Average error rate	Strength	Corr.
0	105.0	2100	0.0500	28.53	0.1954	0.6770	0.2414
1	119.0	2100	0.0567	29.49	0.1944	0.6820	0.2588
2	107.0	2100	0.0510	28.61	0.1957	0.6779	0.2429
3	94.0	2100	0.0448	29.20	0.1942	0.6819	0.2469
4	106.0	2100	0.0505	28.27	0.1892	0.6870	0.2559
5	103.0	2100	0.0490	28.90	0.1917	0.6852	0.2484
6	105.0	2100	0.0500	28.85	0.1951	0.6787	0.2403
7	107.0	2100	0.0510	27.89	0.1931	0.6822	0.2504
8	114.0	2100	0.0543	28.66	0.1983	0.6755	0.2450
9	95.0	2100	0.0452	29.42	0.1959	0.6778	0.2361
Mean	105.5	2100	0.0502	28.78	0.1943	0.6805	0.2466
Stdev	7.5	-	0.0036	-	-	0.0038	0.0070

Table C-1: Image Data.

Ten sets of ensembles each of size 100 are constructed using the different random number seeds. Each column shows the following: *No. of misclass* = number of observations in the test set misclassified by the ensemble. *Test set size* = the number of observations in the test set. *Error rate* = error rate of the ensemble on the test set = (No. of misclass)/(Test set size). *Average size* = average size of the individual trees. *Average error rate* = average error rate on the test set of the individual trees. *Strength* = strength. *Corr.* = correlation.

Run no.	No. of misclass	Test set size	Error rate	Average size	Average error rate	Strength	Corr.
0	180.0	462	0.3896	93.31	0.6609	0.0868	0.1009
1	188.0	462	0.4069	93.03	0.6613	0.0803	0.1087
2	173.0	462	0.3745	92.65	0.6594	0.0876	0.0939
3	180.0	462	0.3896	94.28	0.6683	0.0802	0.0915
4	181.0	462	0.3918	92.68	0.6549	0.0883	0.1051
5	177.0	462	0.3831	92.64	0.6615	0.0855	0.1029
6	179.0	462	0.3874	93.23	0.6613	0.0879	0.0973
7	183.0	462	0.3961	92.81	0.6615	0.0853	0.0986
8	182.0	462	0.3939	91.98	0.6565	0.0914	0.0990
9	165.0	462	0.3571	92.88	0.6509	0.0981	0.0973
Mean	178.8	462	0.3870	92.95	0.6596	0.0872	0.0995
Stdev	6.2	-	0.0135	-	-	0.0052	0.0051

Table C-2: Vowel Data.

Run no.	No. of misclass	Test set size	Error rate	Average size	Average error rate	Strength	Corr.
0	283.0	1500	0.1887	25.78	0.3185	0.3871	0.2065
1	286.0	1500	0.1907	25.73	0.3187	0.3861	0.2075
2	287.0	1500	0.1913	25.29	0.3167	0.3903	0.2132
3	280.0	1500	0.1867	25.06	0.3189	0.3859	0.2056
4	291.0	1500	0.1940	25.78	0.3192	0.3864	0.2108
5	272.0	1500	0.1813	25.59	0.3142	0.3950	0.2036
6	275.0	1500	0.1833	25.24	0.3165	0.3906	0.2142
7	287.0	1500	0.1913	24.98	0.3152	0.3927	0.2117
8	288.0	1500	0.1920	24.88	0.3194	0.3852	0.2132
9	278.0	1500	0.1853	25.55	0.3170	0.3896	0.2022
Mean	282.7	1500	0.1885	25.39	0.3174	0.3889	0.2089
Stdev	6.2	-	0.0041	-	-	0.0033	0.0043

Table C-3: Waves Data.

LIST OF REFERENCES

Bauer, E. and Kohavi, R. (1999), "An Empirical Comparison of Voting Classification Algorithm: Bagging, Boosting and Variants," *Machine Learning*, 36 (1-2), pp.105-139.

Bazaraa M. S., Sherali, H. D. and Shetty, C. M. (1993), *Nonlinear Programming, Second Edition*, John Wiley & Sons, Inc., New York, NY.

Blake, C. L. and Merz, C. J. (1998), UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984), *Classification and Regression Trees*, Wadsworth and Brooks, Monterey, CA.

Breiman, L. (1996), "Bagging Predictors," *Machine Learning*, 24 (2), pp.123-140.

Breiman, L. (1997), "Prediction Games and Arcing Algorithms," Technical Report 504, Statistics Department, U.C.Berkeley, December 1997, revised December 1998.

Breiman, L. (1998), "Arcing Classifiers," *The Annals of Statistics*, 26 (3), pp.801-824.

Breiman, L. (2000), "Randomizing Outputs to Increase Prediction Accuracy," *Machine Learning*, 40 (3), pp. 229-242.

Breiman, L. (2001), "Random Forests," *Machine Learning*, 45 (1), pp.5-32.

Chambers, J. M. and Hastie, T. J. (1992), *Statistical Models in S*, Wadsworth and Brooks Cole Advanced Books and Software, Pacific Grove, CA.

Conover, W. J. (1999), *Practical Nonparametric Statistics, Third Edition* , John Wiley & Sons, Inc., New York, NY.

Cover, T. M. and Hart, P. E. (1967), "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, 13 (1), pp.21-27. [Reprinted in Dasarathy (1991)]

Cutler, A. and Zhao, G. (2001), "PERT – Perfect Random Tree Ensembles," *Computing Science and Statistics*.

Dasarathy, B. V. (ed.) (1991), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, CA: IEEE Computer Society Press.

Dietterich, T. G. (1999), "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," *Machine Learning*, 40 (2), pp. 1-22.

Dietterich, T. G. (2000), "Ensemble Methods in Machine Learning," *Multiple Classifier Systems, Lecture Notes in Computer Science*, 1857, pp.1-15.

Duda, R. O., Hart, P. E. and Stork, D. G. (2000), *Pattern Classification*, John Wiley & Sons, Inc., New York, NY.

Fisher, R. A. (1936), "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, 7,pp. 179-188.

Fix, E. and Hodges, J. L. (1951), "Discriminatory Analysis – Nonparametric Discrimination: Consistency Properties," Report No.4, U.S. Air Force School of Aviation Medicine, Random Field, Texas. [Reprinted in Dasarathy (1991)]

Freund, Y. and Schapire, R. E. (1997), "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, 55 (1), pp.119-139.

Freund, Y. and Schapire, R. E. (1998), Discussion in "Arcing Classifiers," by Breiman, L. *The Annals of Statistics*, 26 (3) pp.824-832.

Friedman, J. (2001), "Greedy Function Approximation: the Gradient Boosting Machine, " *The Annals of Statistics*, 29 (5), pp. 1189-1232.

Friedman, J., Hastie, T., Tibsirani, R. (2000), "Additive Logistic Regression: A Statistical View of Boosting," *The Annals of Statistics*, 28 (2), pp. 337-407.

Hastie, T., Tibsirani, R. and Friedman, J. (2001), *The Element of Statistical Learning*, Springer, New York, NY.

Insightful (2000), *S-Plus 2000 Guide to Statistics*, *Volume 1*, Data Analysis Products Division, Seattle, WA: Insightful Corp.

Jain, A. K. and Mao, Jianchang (1996), "Artificial Neural Networks: A Tutorial," *Computer*, 29 (3) pp.31-44.

Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979), *Multivariate Analysis*, Academic Press, New York, NY.

Mangasarian, O. L. and Wolberg, W. H. (1990), "Cancer Diagnosis via Linear Programming," *SIAM News*, 23 (5), pp 1 and 18.

Monteiro, A. S. (2002), "Multiple Additive Regression Trees – A Methodology for Predictive Data Mining – for Fraud Detection," Master's Thesis, Operation Research Department, Naval Postgraduate School.

Morgan, J. N. and Sonquist, J. A. (1963), "Problems in the Analysis of Survey Data, and a Proposal," *Journal of the American Statistical Association*, 58 (302), pp.415-434.

Ripley, B. D. (1996), *Pattern Recognition and Neural Networks*, Cambridge, New York, NY.

Venables, W. N. and Ripley, B. D. (1999), *Modern Applied Statistics with S-PLUS, Third Edition*, Springer, New York, NY.

Schapire, R. E., Freund, Y., Bartlett, P. and Lee, W. S. (1998), "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods," *Annals of Statistics*, 26 (5), pp.1651-1686.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Samuel E. Buttrey
   Naval Postgraduate School
   Monterey, California

4. Professor Lyn R. Whittaker
   Naval Postgraduate School
   Monterey, California

5. Professor Robert F. Dell
   Naval Postgraduate School
   Monterey, California

6. Professor Robert A. Koyak
   Naval Postgraduate School
   Monterey, California

7. Professor Craig W. Rasmussen
   Naval Postgraduate School
   Monterey, California

8. Operations Evaluation Office, MSO
   Japan Defense Agency
   Tokyo, Japan

9. Izumi Kobayashi
   Operations Evaluation Office, MSO
   Japan Defense Agency
   Tokyo, Japan