THREE ESSAYS ON IMPROVING ENSEMBLE MODELS

by

JIE XU

J. BRIAN GRAY, COMMITTEE CHAIR
BRUCE E. BARRETT
B. MICHAEL ADAMS
JUNSOO LEE
JUDY GIESEN

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of
Information Systems, Statistics, and Management Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2013

ABSTRACT

Ensemble models, such as bagging (Breiman, 1996), random forests (Breiman, 2001a), and boosting (Freund and Schapire, 1997), have better predictive accuracy than single classifiers. These ensembles typically consist of hundreds of single classifiers, which makes future predictions and model interpretation much more difficult than for single classifiers. Breiman (2001b) gave random forests a grade of A+ in predictive performance, but a grade of F in interpretability. Breiman (2001a) also mentioned that the performance of an ensemble model depends on the strengths of the individual classifiers in the ensemble and the correlations among them. Reyzin and Schapire (2006) stated that "the margins explanation basically says that when all other factors are equal, higher margins result in lower error," which is referred to as the "large margin theory." Shen and Li (2010) showed that the performance of an ensemble model is related to the mean and the variance of the margins. In this research, we improve ensemble models from two perspectives, increasing the interpretability and/or decreasing the test error rate. We first propose a new method based on quadratic programming that uses information on the strengths of the individual classifiers in the ensemble and their correlations, to improve or maintain the predictive accuracy of an ensemble while significantly reducing its size. In the second essay, we improve the predictive accuracy of random forests by adding an AdaBoost-like improvement step to random forests. Finally, we propose a method to improve the strength of the individual classifiers by using fully-grown trees fitted on weighted resampling training data and then combining the trees by using the AdaBoost method.

DEDICATION

This dissertation is dedicated to my parents and my American family for their support these years.

LIST OF ABBREVIATIONS AND SYMBOLS

CART	Classification and Regression Tree
OOB	Out-of-Bag
OCPP	OOB correct prediction proportion
UCI	University of California, Irvine
QP	Quadratic Program
RF	Random Forests

ACKNOWLEDGMENTS

I would like to express my very great appreciation to Dr. Gray for his patience and generosity in guiding my research. Without his enthusiastic encouragement and useful critiques, I cannot complete this journey. He is the best advisor I have ever met. I have a very good time with him and will have an unforgettable memory in the future.

Also, I would like to thank my other committee members, Dr. Bruce Barrett, Dr. Michael Adams, Dr. Judy Giesen, and Dr. Junsoo Lee for their help in my research. I am particularly grateful for the financial assistance given by Dr. Judy Giesen and Dr. Charles Sox through my doctoral studies.

Finally, I would like to express my deep gratitude to my parents, Yanqiu Zhang and Jinkang Xu, for their unconditional support all these years. My gratitude and love also goes to my American family, Jan Gibson and Clyde Gibson. They give me a very good impression of this state and country.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

The objective of this research is to improve ensemble models, by either shrinking the size or improving the predictive performance. There are many types of ensemble models, but in this research we mainly focus on bagging, random forests and AdaBoost.

We are interested in two types of predictive models, individual decision trees and ensemble models. Individual decision trees, such as CART, have good performance and can be easily interpreted. An ensemble model, such as those produced by bagging, random forests, or AdaBoost, is an aggregation of ten, hundreds, or even thousands of individual trees. Each individual classifier in the ensemble predicts the binary class of a new observation, and these predictions are combined in a weighted or unweighted vote for the final ensemble prediction. Compared to single trees, ensemble models have better performance but are much more difficult to interpret.

Bagging and random forests use equal weights to combine fully-grown trees, which are built from bootstrapped data sets and the observations in each terminal node are homogeneous. Breiman (2001a) mentioned that the performance of an ensemble model depends on the strength of the individual classifiers in the ensemble and the correlations among them. At each node split, a bagging tree goes through all predictor variables and finds the best one as the split variable, for which the best split value is selected. A random forest tree selects the best split variable from a subset of predictor variables and then chooses the best split value from the variable. Comparing random forests to bagging, we can see that random forests inject randomness so as to decrease

the correlations among the trees while the strengths do not change a lot. Breiman (2001a) believed that is the reason why random forests outperform bagging.

Unlike the other two ensemble methods, boosting uses weak learners, like stumps or pruned trees, instead of fully-grown trees. In each round, boosting identifies the misclassified observations and assigns them higher weights. Then boosting fits a tree on the weighted training data and calculates the tree weight by using the misclassification error. Finally, those trees are combined with unequal weights. The margin of an observation, which is a measure of how well the observation is predicted by the ensemble, plays an important role in explaining the good predictive performance of boosting. Schapire *et al.* (1998) showed that AdaBoost is especially effective at increasing the margins of the training data. Reyzin and Schapire (2006) stated that "the margins explanation basically says that when all other factors are equal, higher margins result in lower error," which is referred to as the "large margin theory."

Led by this theory, many researchers intended to improve the predictive performance by enlarging the training margins. However, there is confusion about how to define large margin. Some use linear programming (LP) methods to maximize the minimum margin (Grove and Schuurmans, 1998). They called their method LP-boost, which reweights the tree weights of AdaBoost by using an LP optimization method. However, later they and other researchers found that this method does not necessarily reduce the generalization error of the ensemble, an error estimated with a test data set (Grove and Schuurmans, 1998; Reyzin and Schapire, 2006 and Breiman, 1999). Reyzin and Schapire (2006) suggested maximizing the average or the median margin. Others (see Wang *et al.*, 2011; Wang *et al.*, 2012) have provided alternative measures of margin.

Shen and Li (2010) showed that the performance of AdaBoost is related to the mean and variance of the margins. They provide a method to increase the mean of the margins while decreasing the variance of the margins, which we call "squeezing" the margins. In chapter 3, we compare the margins of bagging and random forests and find that random forests are squeezing the out-of-bag margins of bagging. After squeezing, random forests lose some predictive power in well predicted observations but gain it back in poorly predicted observations. The sacrifice of predictive performance of the well predicted observations does not impact the generalized error since most tree predictions of those observations are still correct; while the gain in poorly predicted observations could possibly improve the generalized predictive performance.

Inspired by Shen and Li's (2010) research and our findings, we use quadratic programming to squeeze the margins in chapter 2, minimizing the variance of the margins and maximizing the mean of the margins. The side effect of applying a quadratic program is that many of the tree weights zero out. Therefore we can see the size of the new ensemble is much smaller while the predictive performance is maintained.

In chapter 3, we provide another squeeze method, which is similar to boosting. We notice that some observations do not appear in the bootstrap data used in constructing a tree, and we call them out-of-bag data (OOB data). We use OOB data to identify the poorly predicted and well predicted observations. For those poorly predicted observations, we let them appear more times when they actually play the role of OOB data; as to those well predicted observations, we reduce their appearance times as inbag data. This method gives poorly predicted observations more weight and assigns smaller weights to well predicted observations. We applied this method to random forests, which we call boosting random forests. We show that the new model outperforms random forests in many cases.

The blending of boosting and random forests in chapter 3 gave us the inspiration to mix them in another way. In chapter 4 we apply random forests features to AdaBoost. We stated earlier that AdaBoost uses weak learners, like stumps or pruned trees, as single classifiers. AdaBoost cannot use fully-grown trees as single classifiers. AdaBoost uses training error to identify which observations are poorly predicted and which are not. A fully-grown tree perfectly predicts the training data and the training error is zero. Therefore, it is impossible to calculate the next weights of the observations and the boosting method can not continue. However, in chapter 3 we show the advantage of using fully-grown random forests trees. It is promising to use a fully-grown random forests tree to replace a stump or pruned CART tree in AdaBoost. To make this happen, we use a resampling method instead of a reweighting method so that we can use fully-grown trees as single classifiers. The results of the new model compares favorably to random forests and AdaBoost in many cases.

1.2 Single Trees

Classification and regression tree models are popular alternatives to the classical methods of discriminant analysis, logistic regression, and linear regression, and are widely used in data mining and predictive modeling. The earliest decision tree models were AID (Automatic Interaction Detection) proposed by Morgan and Sonquest (1963) and CHAID (Chi-square Automatic Interaction Detection) proposed by Kass (1980). Other methods followed, but the CART (Classification and Regression Trees) methodology proposed by Breiman, et al. (1984) is perhaps best known and most widely used.

Recursive partitioning algorithms grow a binary decision tree in a sequential fashion by using splitting rules based on the predictor variables to partition the data in such a way as to

reduce variation in the conditional response variable distribution. These methods perform a greedy, locally optimal search for the best split at each stage in growing the tree. Such a search is not guaranteed to lead to the best overall tree. The recursive partitioning approach for constructing decision trees is implemented in CART 5.0, C5.0, S-Plus, R, JMP, and SAS Enterprise Miner.

A decision tree with T leaves (terminal nodes) is constructed on the training data to partition the predictor variable space into T regions, S_t , for $t = 1,2,\ldots,T$. Due to the use of single variable split rules, these regions are hyper-rectangular in shape. Each of these regions corresponds to a terminal node of the decision tree. The prediction for a new observation in region S_t is typically given by the majority or average Y value of the training observations within the corresponding terminal node.

In CART specifically, trees are grown as large as possible to avoid early stopping, then pruned backward using a cost-complexity criterion to avoid overfitting of the training data and to obtain "right-sized" trees. CART uses cross-validation or a large independent test sample of data to select the best tree from the sequence of trees considered in the pruning process.

Breiman (2001b) suggested that CART and similar techniques perform well with regard to interpretation, but poorly with regard to prediction. He gave CART a grade of A^+ on interpretability, but only a grade of B with regard to its predictive performance on future data.

1.3 Ensemble Models

Before discussing specific ensemble models, it will be helpful to have some mathematical notation. We assume $\{(\mathbf{x}_i, y_i), i = 1,2,\ldots,n\}$ is a random sample of n training observations from a population following some distribution $P(\mathbf{x}, y)$. The $p \times 1$ vector $\mathbf{x}$ is the vector of predictor

variable values and y is the binary outcome. There is an ensemble of T classifiers $\{t_1, t_2, \ldots, t_T\}$, which we will assume to be decision trees constructed from the training sample. The trees are assigned corresponding weights $\alpha = \left[\alpha_1, \alpha_2, \ldots, \alpha_T\right]^t$. The prediction of a tree t is denoted by $h_t(\mathbf{x}) \in \{-1, 1\}$. The output of an ensemble for a vector $\mathbf{x}$ is given by $H_0(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$, and the final prediction of the ensemble for a vector $\mathbf{x}$ is $H(\mathbf{x}) = sign\left(H_0(\mathbf{x})\right)$, which is +1 if $H_0(\mathbf{x}) \geq 0$, and –1 if $H_0(\mathbf{x}) < 0$. Finally, the matrix $\mathbf{H} = \left\{h_t(\mathbf{x}_i)\right\}_{n \times T}$ consists of all of the individual tree *t* predictions for each training observation *i*.

Ensemble methods differ in how their classifiers are generated and in how they weight the predictions of the individual classifiers.

Breiman proposed the bagging (bootstrap aggregation) method for generating an ensemble of individual classifiers. Bagging relies on the instability of recursive partitioning methods, such as CART, to slight changes in the data in order to generate multiple versions of a decision tree classifier. A bootstrap sample of data is collected from the original training data by sampling with replacement, and a fully-grown tree is created from the bootstrap sample. The process is repeated to create an ensemble of classifiers, and equal-weight voting is employed to combine the predictions of the classifiers.

AdaBoost creates a sequence of weak learners to form an ensemble, but the weights of the classifiers are not equal. At each iteration, the training data are reweighted to provide more weight to observations misclassified in the previously constructed weak learner and to provide less weight to observations correctly classified. A new classifier (tree) is constructed on the reweighted data, and the weight given to the new tree is based on how well it fits the reweighted

training data. The final classifier for a new observation is based on the weighted average of the individual tree predictions.

In his random forests proposal, Breiman took his bagging idea and added a twist. In the tree construction process, rather than searching for the best split variable among all predictor variables, a split variable is picked at random. This adds diversity to the individual classifiers created from the bootstrapped training data. Generally speaking, random forests tend to outperform bagged ensembles.

1.4 Margins and the Large Margin Theory

Ensemble models have better predictive performance than single classifiers. Researchers have provided empirical findings and developed theory to suggest why ensemble models work so well. Many of these results are based on the concept of the margin of an observation.

Schapire, Freund, Bartlett, and Lee (1998) defined the margin for a training observation as the difference between the weight assigned to the correct label and the maximal weight assigned to any single incorrect label. In the case of a binary response variable, the margin is the signed difference between the weighted votes for the correct label and the incorrect label; the larger the margin, the more confidence in the classification. In terms of our notation introduced earlier, the margin of the *i*-th observation is given by

$$m_i = \frac{y_i \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})}{\sum_{t=1}^{T} \alpha_t},$$

where $-1 \le m_i \le 1$. A margin of +1 indicates that all of the classifiers correctly predicted the observation, while a margin of –1 indicates that all of the classifiers incorrectly predicted the observation.

Schapire, et al. (1998) developed an upper bound on the generalization error of a combined classifier given by

$$P[H(\mathbf{x}) \neq y] \le \hat{P}[m(\mathbf{x}, y) \le \theta] + O\left(\sqrt{\frac{d}{n\theta^2}}\right),$$

where $P[H(\mathbf{x}) \neq y]$ is the generalization error of the combined classifier, $\hat{P}[m(\mathbf{x}, y) \le \theta]$ is the proportion of training set margins less than a value $\theta > 0$, and *d* is the VC-dimension of the space of all possible weak classifiers (a measure of complexity). They used this bound to provide an explanation for the good performance of AdaBoost, which they show is highly effective in increasing the margins of the training data.

Even though this upper bound is quite loose in most practical situations, researchers have used it to conclude that larger margins should lead to a lower generalization error rate, everything else being equal. Freund and Schapire (1996) also found that AdaBoost increases margins quickly compared to other models, and suggested that improving the margins should lead to better generalization error. Reyzin and Schapire (2006) state that "the margins explanation basically says that when all other factors are equal, higher margins result in lower error." This interpretation of the upper bound on the generalization error has been referred to as the "large margins theory" in the boosting literature.

Researchers have taken different approaches to improve the margin distribution by adjusting the weights of the individual classifiers. Some have tried to maximize the minimum margin using linear programming techniques, while others have considered maximizing the

mean or median of the margins or some other function of the margins. These approaches have met with mixed success. More recently, Shen and Li (2010) have developed a quadratic programming technique for simultaneously maximizing the mean and minimizing the variance of the margin distribution, which has shown better performance.

1.5 Strength and Diversity

Breiman (1996) found that if a small change in the training data can result in large changes in classifiers (more diversity), then bagging will lead to improved accuracy. Dietterich (2000) realized that ensemble accuracy is related to the performance (or strength) of the individual classifiers and the diversity among them. He found that AdaBoost gives the best results in low noise settings and has higher diversity than other methods. However, in high noise settings, the diversity is reduced and the performance is not as good as with bagging. Breiman (2001a) established an upper bound for the generalization error of random forests, indicating two ingredients related to the generalization error, the strengths of the individual classifiers and the correlations among them.

Breiman (2001a) concluded that random forests with higher strength trees and lower correlations among the trees (*i.e.*, more diversity) tend to have better predictive performance. By using randomly selected inputs or combinations of inputs at each node in growing trees in a random forest, he increased the diversity of the ensemble while maintaining the strengths of the individual trees. The overall performance of a random forest is better than a single tree and compares favorably with AdaBoost. However, like other ensembles, random forests are too complicated to be interpreted easily and require more computational effort.

The shortcomings of ensemble models are also obvious. Since an ensemble model is an aggregation of single classifiers and usually the number of classifiers is large, it is often very complicated and difficult to interpret. Ensemble models consume a lot of memory and the computational cost is high compared to the improved performance. The idea of thinning or pruning an ensemble is becoming more and more attractive, and various methods have been proposed for ensemble reduction.

1.6 Statement of the Problem

The aim of this research is to improve ensemble models, reducing the generalized error and/or the size of the model. In Chapter 2, we use a quadratic programming method to reduce the size of random forests while maintaining the performance. In Chapter 3, we add AdaBoost features to random forests. Finally in Chapter 4, we blend AdaBoost and random forests from a different perspective. By replacing stumps or pruned trees with fully-grown trees, we improve the strengths of the single classifiers while maintaining the diversity among them so as to improve the predictive performance.

CHAPTER 2

A QUADRATIC PROGRAMMING ALGORITHM FOR IMPROVING AN ENSEMBLE MODEL

2.1 Introduction

Two major types of predictive models are trees and ensembles. Trees, such as CART and RPART, have good performance and can be easily interpreted. Ensemble models, like random forests, boosting, and bagging, are aggregations of hundreds of trees. Compared to single trees, ensemble models have better performance but are more difficult to interpret. In addition, ensemble models require more computation time.

Breiman (2001) concluded that random forests with higher strength trees and lower correlations among the trees (*i.e.*, more diversity) tend to have better predictive performance. By using randomly selected inputs or combinations of inputs at each node in growing trees in a random forest, Breiman injected some diversity while maintaining the strengths of the individual trees. The overall performance of random forests is better than a single tree and compares favorably with AdaBoost. However, like other ensembles, random forests are too complicated to be interpreted easily and require more computational effort.

Recently, researchers have proposed methods for reducing the size of ensemble models. Some create a new tree or select an existing tree to approximate the ensemble (Domingos 1998). Some researchers have defined new measures of strength and diversity and used these criteria in a forward selection process to iteratively construct a new ensemble from the trees in an existing ensemble or in a backward elimination process to reduce the size of an existing ensemble (see, *e.g.*, Margineantu and Dietterich 1997 and Banfield, Hall, Bowyer, and Kegelmeyer 2005). Some

researchers have tried to improve ensemble performance and reduce the size of the ensemble by employing linear programming techniques to optimize a function of the margins (*e.g.*, maximizing the minimum margin) by reweighting the weak classifiers in an ensemble (see, *e.g.*, Breiman 1999 and Grove and Schuurmans 1998). Other researchers believe that the generalization error of an ensemble model is related to the mean and variance of the margins (Shen and Li 2010).

In this paper, we propose a new method for improving the predictive performance and reducing the size of an ensemble model by reweighting the individual classifiers. The new weights are determined by solving a quadratic programming problem based on the strengths and correlations among the individual classifiers. As with the linear programming approaches discussed above, a side benefit of the proposed solution is that many of the new weights are zero, thus effectively reducing the size of the ensemble, which leads to better efficiency in future prediction calculations and possibly better interpretability of the ensemble model.

In section 2, we review the ensemble literature and recently proposed methods for reducing the size of ensembles. In section 3, we introduce our proposed methods and the theory behind them. In section 4, we apply the proposed method to several real and simulated data sets. We find that the sizes of the ensemble models are reduced significantly and the ensemble performance is maintained or improved. Finally, section 5 is the conclusion section.

2.2 Ensembles and Methods for Their Improvement

Our objective in this paper is to improve random forests; however, the proposed methodology also works with other ensemble models, including AdaBoost and bagging. In this

section, we review single decision trees, ensemble methods, and recently proposed methods for reducing the size of an ensemble while maintaining or improving its performance.

2.2.1 Single Trees

Classification and regression tree models are popular alternatives to the classical methods of discriminant analysis, logistic regression, and linear regression, and are widely used in data mining and predictive modeling. The earliest decision tree models were AID (Automatic Interaction Detection) proposed by Morgan and Sonquest (1963) and CHAID (Chi-square Automatic Interaction Detection) proposed by Kass (1980). Other methods followed, but the CART (Classification and Regression Trees) methodology proposed by Breiman, et al. (1984) is perhaps best known and most widely used.

Recursive partitioning algorithms grow a binary decision tree in a sequential fashion by using splitting rules based on the predictor variables to partition the data in such a way as to reduce variation in the conditional response variable distribution. These methods perform a greedy, locally optimal search for the best split at each stage in growing the tree. Such a search is not guaranteed to lead to the best overall tree. The recursive partitioning approach for constructing decision trees is implemented in CART 5.0, C5.0, S-Plus, R, JMP, and SAS Enterprise Miner.

A decision tree with T leaves (terminal nodes) is constructed on the training data to partition the predictor variable space into T regions, S_t , for $t = 1,2,\ldots\ldots,T$. Due to the use of single variable split rules, these regions are hyper-rectangular in shape. Each of these regions corresponds to a terminal node of the decision tree. The prediction for a new observation in

region S_t is typically given by the majority or average Y value of the training observations within the corresponding terminal node.

In CART specifically, trees are grown as large as possible to avoid early stopping, then pruned backward using a cost-complexity criterion to avoid overfitting of the training data and to obtain "right-sized" trees. CART uses cross-validation or a large independent test sample of data to select the best tree from the sequence of trees considered in the pruning process.

Breiman (2001b) suggested that CART and similar techniques perform well with regard to interpretation, but poorly with regard to prediction. He gave CART a grade of A^+ on interpretability, but only a grade of B with regard to its predictive performance on future data.

2.2.2 Ensemble models

Breiman (1996) proposed the bagging method to generate multiple versions of a classifier based on bootstrapping and used these to obtain an aggregated classifier. Bagging uses equal weights to combine classifiers; therefore, the output of the ensemble is the average of the predictions of the classifiers. AdaBoost (Freund and Schapire 1996) creates a sequence of weak learners to form an ensemble, but the weights of the classifiers are not equal. In his random forests proposal, Breiman (2001a) bootstrapped training data and randomly selected split variables in building classifiers.

Ensemble models have better predictive performance than single classifiers. Researchers have proposed empirical evidence and theory to suggest why ensemble models work so well. Breiman (1996) found that if a small change in the training data can result in large changes in classifiers (more diversity), then bagging will lead to improved accuracy. Freund and Schapire (1996) found that AdaBoost increases margins quickly compared to other models, and suggested

that improving the margins should lead to better generalization error. Dietterich (2000) realized that ensemble accuracy is related to the individual classifier performance and the diversity among them. He found that AdaBoost gives the best results in low noise settings and has higher diversity than other methods. However, in high noise settings, the diversity is reduced and the performance is not as good as with bagging. Breiman (2001a) established an upper bound for the generalization error of random forests based on two components, the strengths of the individual classifiers and the correlations among them.

The shortcomings of ensemble models are also obvious. Since an ensemble model is an aggregation of a large number of single classifiers, it is often very complicated and difficult to interpret. Ensemble models consume a lot of memory and the computational cost is higher (Margineantu and Dietterich 1997). The idea of thinning or pruning an ensemble is becoming attractive. Various methods for reducing the size of an ensemble have been proposed, and they will be discussed next.

2.2.3 Ensemble Reduction Methods

At the extreme of ensemble thinning, some researchers have proposed selecting or constructing a single classifier to approximate an ensemble (see, *e.g.*, Domingos 1998). Clearly, the single classifier will not have the same predictive accuracy as the ensemble, but the interpretation and computational aspects of the model are improved greatly.

Some researchers have followed Breiman (2001a) by developing new measures of the strength and diversity of an ensemble, then using these measures to prune or thin an ensemble (see, *e.g.*, Margineantu and Dietterich 1997; Latinne, Debeir, and Decaestecker 2001; Banfield, Hall, Bowyer, and Kegelmeyer 2005; and Ko, Sabourin, and Britto 2009).

In some cases, these measures of the strength and diversity of an ensemble are used as criteria for a forward selection or backward elimination search over the trees in an existing ensemble (see, *e.g.*, Margineantu and Dietterich 1997 and Banfield, Hall, Bowyer, and Kegelmeyer 2005). A forward selection approach adds trees, one at a time, from the existing ensemble to a new ensemble, while monitoring improvements to the strength and diversity measures of the new ensemble to decide when to stop adding trees. In backward elimination, trees are eliminated one at a time from the existing ensemble until the strength and diversity measures indicate stopping the process. Of course, as in stepwise regression, these greedy approaches are not guaranteed to lead to an optimal ensemble.

Ko, Sabourin, and Britto (2009) proposed new functions of the strength and diversity of single classifiers and used a genetic algorithm with these compound diversity functions as the fitness function to select classifiers from the existing ensemble to create a reduced ensemble model. While the genetic algorithm approach is not as greedy as the methods above, there is still no guarantee of optimality.

Some researchers have tried to improve ensemble performance and reduce the size of the ensemble by employing linear programming techniques to optimize a function of the margins (*e.g.*, maximizing the minimum margin) by reweighting the weak classifiers in an ensemble (see, *e.g.*, Breiman 1996 and Grove and Schuurmans 1998). Breiman (1999) proposed a boosting algorithm for maximizing the minimum margin; however, the test error of the proposed method is not as good as that of AdaBoost.

Other researchers considered not only the mean of the margins but also the variance of the margins. Shen and Li (2010) applied quadratic programming to boosting with an objective function that is a combination of the mean and the variance of the margins, so as to maximize the

mean and minimize the variance. However, the support for their method is based on the assumption of independence of classifiers, which is often not true. They also used stumps as the classifiers in boosting, while stronger learners are actually more popular.

From the previous work, we can see that a good thinning method needs to consider accuracy and diversity, which is the foundation of our model. Previous work mainly concentrates on how to select classifiers, but seldom mentions how to combine them. Should we use equal weights or unequal ones? We will discuss the above questions and introduce our proposed method in the following sections.

2.3 Proposed Method

The inspiration for our proposed method comes from the field of finance. Modern portfolio theory (Markowitz, 1952) formulates the problem of constructing a stock portfolio as a quadratic programming optimization problem. The percentages or weights of the portfolio to be invested in a fixed set of individual stocks are determined in such a way as to minimize the variance (risk) of the portfolio return for a fixed level of expected portfolio return. Input data to the problem formulation includes a history of returns for each of the individual stocks and a covariance or correlation matrix measuring the relationships among the individual stocks. By combining several diverse stocks into a portfolio, it is possible to reduce risk (variance) while maintaining the level of expected return.

By analogy, our proposed method is designed to construct an ensemble (portfolio) of individual classifiers (stocks) in such as way as to reduce the variation in ensemble strength (risk) while maintaining a desired level of expected strength (return). In effect, this increases the probability of correct ensemble predictions, as we explain next.

2.3.1 Notation and Preliminaries

We assume $(\mathbf{x}, y)$ is an observation from a population following some distribution $P(\mathbf{x}, y)$. The $p \times 1$ vector $\mathbf{x}$ is the vector of predictor variable values and y is the binary outcome. We are given an ensemble of T trees $\{t_1, t_2, ..., t_T\}$. These trees could be from a random forest, from bagging, or from boosting. The trees are assigned corresponding weights $\alpha_1, \alpha_2, ..., \alpha_T$. For random forests and bagging, the weights are equal. The prediction of a tree t is denoted by $h_t(\mathbf{x}) \in \{-1, 1\}$. The prediction of the ensemble for a given vector $\mathbf{x}$ is

$$H(\mathbf{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right) = \begin{cases} +1, \text{ if } \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) > 0 \\ -1, \text{ if } \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) < 0 \end{cases}$$

In the above formula, the sign function converts the linear combination of the individual tree predictions to +1 or –1. If the weighted tree prediction is larger than zero, then the ensemble prediction is equal to +1; if the weighted tree prediction is less than zero, then the ensemble prediction is equal to –1.

Let $O_t \equiv O_t(x, y)$ be a random variable representing the outcome of tree t on a new observation $(x, y) \sim P(x, y), t = 1,2,...,T$. If tree t correctly predicts the new observation ($h_t(\mathbf{x}) = y$), then $O_t = +1$; otherwise, if tree t incorrectly predicts ($h_t(\mathbf{x}) \neq y$), then $O_t = -1$.

$\pi_t^+ = P(O_t = +1)$ is the probability of a correct prediction by tree *t*. $\pi_t^- = P(O_t = -1)$ is the probability of an incorrect prediction by tree *t*. Clearly, $\pi_t^+ = 1 - \pi_t^-$. Since $\frac{1}{2}(O_t + 1)$ follows a

Bernoulli distribution with parameter π_t^+, then $E\left(\frac{1}{2}(O_t+1)\right)=\frac{1}{2}E(O_t)+\frac{1}{2}=\pi_t^+$. We can measure the strength of tree t using $S_t=E(O_t)=\pi_t^+-\pi_t^-$.

2.3.2 Quadratic programming formulation

We can also define a measure of ensemble strength as $S=\sum_{t=1}^{T}\alpha_t O_t=\sum_{t:O_t=+1}\alpha_t-\sum_{t:O_t\neq-1}\alpha_t$.

The random variable S is a linear combination of tree outcomes, and is equal to the difference between the sum of the weights associated with the trees making correct predictions and the sum of the weights of the trees making incorrect predictions. We note that the event $S>0$ is equivalent to $H(\mathbf{x})=y$, so that $P(S>0)=P(H(\mathbf{x})=y)$, the probability of correct ensemble prediction, which we want to maximize by choosing different weights $\alpha_1,\alpha_2,\ldots,\alpha_T$. The distribution of S is unknown, so direct calculation and maximization of $P(S>0)$ is not possible. However, the expectation and the variance of S are given by

$$E(S)=E\left(\sum_{t=1}^{T}\alpha_t O_t\right)=\sum_{t=1}^{T}\alpha_t E(O_t)=\sum_{t=1}^{T}\alpha_t(\pi_t^+-\pi_t^-)$$

and

$$Var(S)=\alpha^t\Sigma_O\alpha=\sum_{i=1}^{T}\sum_{j=1}^{T}\alpha_i\alpha_j\,\mathrm{cov}(O_i,O_j)=\sum_{i=1}^{T}\sum_{j=1}^{T}\alpha_i\alpha_j\sigma_{ij}$$

where Σ_O is the $T\times T$ variance-covariance matrix of the outcome variables O_t. The maximization of the quantity $Z=\frac{E(S)-0}{\sqrt{Var(S)}}$ should lead to a large value of $P(S>0)$.

We can solve the following quadratic programming problem to find the tree weights $\alpha_1,\alpha_2,\ldots,\alpha_T$ that minimize $Var(S)$ for a fixed minimum value *b* of $E(S)$:

$$\text{minimize} \qquad Var(S) = \alpha^t \sum_O \alpha$$

$$s.t. \qquad E(S) = \sum_{t=1}^{T} \alpha_t (\pi_t^+ - \pi_t^-) \geq b$$

$$\sum_{t=1}^{T} \alpha_t = 1$$

$$\alpha_t \geq 0$$

Although $\sum_O$, π_t^+, and π_t^- are unknown, we can estimate their values for the random forest trees $\{t_1, t_2, ..., t_T\}$, which were created from a training data set, using the training data or a validation data set.

The quadratic program is solved over a range of *b* values. We start by solving the quadratic program for $b = 0$, then use the value of the left-hand side in the $E(S)$ constraint as the initial value of *b*. Using a small step size, we increase *b* and solve a new quadratic program, stopping when there is no feasible solution (*i.e.*, when no set of weights will yield a value of $E(S) \geq b$). By the above steps, we are seeking the minimum *Z* value. Either *Z* or $P(S > 0)$, estimated from the training data or a validation data set, can be used to select from among the quadratic program solutions to determine the optimal weights $\alpha_1, \alpha_2, ..., \alpha_T$.

A side benefit of the quadratic programming solution is that in most cases, the majority of the optimal weights $\alpha_1, \alpha_2, ..., \alpha_T$ are zero, thus simplifying the solution by reducing the size of the random forest.

2.3.3 Evaluation of the proposed method

To evaluate our proposed method, we designed the following simulation:

Step 1: Randomly split the data into training (75%) and test sets (25%).

Step 2: Fit a random forest to the training data. We use the randomForest package in R to build the random forest for ensemble sizes T = 50, 100, 200, 300, 400, and 500.

Step 3: Use the training data to estimate the covariance matrix $\sum_O$ and the probabilities π_t^+ and π_t^- used to define the tree strengths.

Step 4: Solve the quadratic program along a grid of b values to maximize Z over the training data (using the quadprog package in R).

Step 5: Compute test set error rates for the random forest and the thinned random forest.

Step 6: Repeat Steps (1-5) 100 times and average the results.

In using the training data to estimate the parameters, the estimated parameters are biased. However, for random forests, each tree is built on bootstrapped data, which means that about 37% of the training data are not used in constructing the tree, so that the level of bias is less than might otherwise be expected.

In Step 4, if the covariance matrix is singular, then the quadratic program cannot be solved. This happens when prediction vectors for trees are linearly dependent or duplicates. To avoid this problem, we remove duplicates and any prediction vectors that are linearly related to other prediction vectors.

In the next section, we use the above procedure to evaluate our proposed method.

2.4 Simulation Results

We applied the proposed method to five different data sets. For each data set, we calculated the average test error rates to compare the performance between the original random forest and the reduced one.

The first data set is the Circle data with $n = 300$ observations and $p = 2$ predictor variables.

Random Forest		Thinned Forest	
Trees (T)	Test Error Rate (SE)	Test Error Rate (SE)	Avg Trees (SE)
50	.0809 (.0023)	.0817 (.0022)	35.05 (.28)
100	.0773 (.0022)	.0778 (.0024)	49.54 (.36)
200	.0785 (.0020)	.0760* (.0022)	64.43 (.47)
300	.0783 (.0020)	.0757* (.0022)	72.83 (.52)
400	.0774 (.0022)	.0768 (.0022)	78.78 (.65)
500	.0792 (.0021)	.0791 (.0021)	82.16 (.54)

* statistically lower (at the 5% level)

Table 1. Average Test Error Rates for Circle Data

There are three concentric rings of uniform data constrained to the unit square. The innermost y values are equal to +1, the middle y values = –1, and the outermost y values = +1. The simulation results from the circle data are displayed in Table 1, including the average test error rates of the original random forest and the thinned random forest along with their estimated standard errors. The average test error rate of the thinned forest is very close to that of the original random forest. The paired t tests show that thinned forests of size 200 and 300 are better than the original random forest. The size of the thinned forest is 1/6 to 2/3 of the original random forest size.

The second data set is Circle1500, which is similar to Circle, but with a larger number of observations (n = 1500, p = 2).

Random Forest		Thinned Forest	
Trees (T)	Test Error Rate (SE)	Test Error Rate (SE)	Avg Trees (SE)
50	.0257 (.0008)	.0256 (.0008)	31.37 (.28)
100	.0245 (.0008)	.0245 (.0008)	41.79 (.37)
200	.0271 (.0008)	.0258** (.0007)	52.97 (.49)
300	.0244 (.0008)	.0247 (.0008)	59.04 (.56)
400	.0254 (.0008)	.0239** (.0007)	63.46 (.60)
500	.0258 (.0007)	.0255 (.0007)	67.82 (.57)

** statistically lower (at the 1% level)

Table 2. Average Test Error Rates for Circle1500 Data

The performance of both methods is better than for Circle data because there are more data in Circle1500. Once again, the size of the thinned forest is much smaller than for the original random forest.

The third data set is the Sonar data (result is in Table 3), which is a real data set with n = 208 observations and p = 60 predictor variables. The average test error rate of the thinned forest is close to that of the random forest, but the tree size of the thinned forest is only 1/8 to 2/3 of the original random forest size.

The fourth data set is the Breast Cancer data (result in Table 4), which is also a real data set with n = 683 observations and p = 9 predictor variables.

Random Forest		Thinned Forest	
Trees (T)	Test Error Rate (SE)	Test Error Rate (SE)	Avg Trees (SE)
50	.1860 (.0054)	.1781* (.0048)	37.28 (.25)
100	.1881 (.0052)	.1823 (.0047)	54.34 (.40)
200	.1838 (.0055)	.1815 (.0048)	64.18 (.51)
300	.1671 (.0052)	.1731 (.0050)	65.51 (.45)
400	.1698 (.0046)	.1723 (.0052)	65.77 (.36)
500	.1862 (.0049)	.1779* (.0050)	64.30 (.45)

* statistically lower (at the 5% level)

Table 3. Average Test Error Rates for Sonar Data

Random Forest		Thinned Forest	
Trees (*T*)	Test Error Rate (SE)	Test Error Rate (SE)	Avg Trees (SE)
50	.0299** (.0011)	.0347 (.0011)	23.07 (.30)
100	.0295** (.0013)	.0339 (.0013)	29.25 (.32)
200	.0314** (.0014)	.0348 (.0014)	35.44 (.36)
300	.0296** (.0011)	.0331 (.0012)	36.81 (.37)
400	.0282** (.0012)	.0314 (.0012)	37.90 (.37)
500	.0285** (.0012)	.0346 (.0014)	38.10 (.38)

** statistically lower (at the 1% level)

Table 4. Average Test Error Rates for Breast Cancer Data

For this data set, the difference in average test error rates between the thinned forest and the original random forest is statistically significantly in favor of the original random forests, but there is still a dramatic drop in the size of the thinned forest.

The last example is for the Spam data (result in Table 5) with n = 4,601 observations and p = 57 predictor variables.

Random Forest		Thinned Forest	
Trees (*T*)	Test Error Rate (SE)	Test Error Rate (SE)	Avg Trees (SE)
50	.0491 (.0006)	.0491 (.0006)	43.69 (.21)
100	.0487 (.0007)	.0482 (.0007)	65.64 (.36)
200	.0482 (.0006)	.0468** (.0006)	88.15 (.54)
300	.0471 (.0006)	.0458** (.0006)	100.32 (.67)
400	.0482 (.0006)	.0468** (.0005)	110.12 (.67)
500	.0481 (.0006)	.0463** (.0006)	116.60 (.62)

** statistically lower (at the 1% level)

Table 5. Average Test Error Rates for Spam Data

The paired *t* tests show that thinned forests of size 200, 300, 400, and 500 are better than the original random forests. The size of the thinned forest is only 1/5 to 4/5 of the random forest size.

2.5 Conclusion

This article has proposed a method for minimizing the variance of ensemble strength given a specified expectation of ensemble strength, which should result in a higher probability of correct prediction, using a quadratic programming approach. The algorithm returns a vector of tree weights, many of which are zero. The new model combines the classifiers by using the weights. Those classifier associated with zero weights are effectively removed from the ensemble. Therefore, the new ensemble model is much smaller than the original. We can also see from the results that the proposed method produces an ensemble with the same or better predictive accuracy than the original ensemble in most cases.

REFERENCES

Banfield, R.E., Hall, L. O., Bowyer, K.W., and Kegelmeyer, W.P. (2005). Ensemble diversity measures and their application to thinning. *Information Fusion*, 6(1), 49-62.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26(2), 123-140.

Breiman, L. (1999). Prediction Games and Arcing Algorithms. *Neural Computation*, 11, 1493-1517.

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

Breiman, L. (2001b). Statistical Modeling: The Two Cultures. *Statistical Science*, 16, 199-231.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984), Classification and Regression Trees, Belmont, CA: Wadsworth.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. *Machine Learning*, 40, 1-22.

Domingos P. (1998). Knowledge discovery via multiple models. *Intelligent Data Analysis*, 3, 187-202.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 148-156.

Grove, A. J., and Schuurmans, D. (1998). Boosting in the Limit: Maximizing the Margin of Learned Ensembles. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*.

Ko, A.H., Sabourin, R., and Britto, A.D.S. (2009). Compound Diversity Functions for Ensemble Selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4), 659-686.

Latinne, P., Debeir O., and Decaestecker C. (2001). Limiting the Number of Trees in Random Forests. *Lecture notes in computer science*, 2096/2001, 178-187.

Margineantu, D.D. and Dietterich, T.G. (1997). Pruning Adaptive Boosting. *Proceedings of the Fourteenth International Conference on Machine Learning*, 211-218.

Schapire, R., Freund, Y., Bartlett, P., and Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5), 1651-1686.

CHAPTER 3

BOOSTING THE RANDOM FOREST

3.1 Introduction

Ensemble models, such as bagging, random forests, and AdaBoost, are aggregations of hundreds of trees. Random forests have better performance than bagging and comparable performance to AdaBoost (Breiman 2001).

Breiman (2001) showed that the performance of an ensemble model depends on the strengths of the individual classifiers in the ensemble and the correlations among them. Compared to bagging, random forests reduce the correlations among the individual trees with minor loss in the strengths of the trees. As a result, the predictive performance of random forests tends to be better than bagging.

Freund and Schapire (1997) found that AdaBoost increases margins quickly compared to other models and suggested that improving the margins should lead to better generalization error. Reyzin and Schapire (2006) stated that "the margins explanation basically says that when all other factors are equal, higher margins result in lower error." This has been referred to as the "large margins theory" in the boosting literature. The performance of AdaBoost is comparable to random forests (Breiman 2001).

Recently, researchers have proposed methods for improving the predictive accuracy of random forests or AdaBoost based on these two theories. Some researchers have defined new measures of ensemble strength and diversity and used these criteria in a forward selection process to iteratively construct a new ensemble from the trees in an existing ensemble or in a backward elimination process so as to reduce the size of an existing ensemble in order to

improve the strength and diversity. Some researchers have tried to improve ensemble performance by employing linear programming techniques to optimize a function of the margins (*e.g.*, maximizing the minimum margin) by reweighting the classifiers in an ensemble (see, *e.g.*, Breiman 1999 and Grove and Schuurmans 1998). Other researchers believe that the generalization error of an ensemble model is related to the mean and variance of the margins (Shen and Li 2010).

In this paper, we propose a new method for improving the predictive accuracy of a random forest by focusing more attention on observations that are poorly predicted, focusing less attention on observations that are well predicted, and re-growing trees in the random forest to have lower generalization error.

In Section 2, we review the ensemble models and recent methods for improving the predictive accuracy of ensembles. In Section 3, we introduce our proposed method and the theory behind it. In Section 4, we apply the proposed method to several real and created data sets. The method is shown to improve or maintain the predictive accuracy for those data sets. Finally, in Section 6, we draw conclusions and provide additional discussion.

3.2 Improving Ensemble Models

Ensemble models include bagging, random forests, and AdaBoost. They have better performance than single classifiers, such as decision trees produced by CART or C4.5. Researchers have proposed some methods for improving the performance of the ensembles. In this section, we will review these methods.

3.2.1 Ensemble models

Breiman (1996) proposed the bagging (bootstrap aggregation) method for generating an ensemble of individual classifiers. Bagging relies on the instability of recursive partitioning methods, such as CART, to slight changes in the data in order to generate multiple versions of a decision tree classifier. A bootstrap sample of data is collected from the original training data by sampling with replacement, and a fully-grown tree is created from the bootstrap sample. The process is repeated to create an ensemble of classifiers, and equal-weight voting is employed to combine the predictions of the classifiers.

Breiman (2001) later proposed the random forest method. Unlike bagging, random forests do not rely on CART. At each split, a variable is randomly selected from a group of predictor variables and then the split value is found that maximizes the Gini index reduction (Breiman et al. 1984). This adds some diversity to the model without losing too much strength. Generally speaking, random forests outperform bagging.

AdaBoost (Freund and Schapire 1996) creates a sequence of weak learners to form an ensemble, but the weights of the classifiers are not equal. At each iteration, the training data are reweighted to provide more weight to observations misclassified in the previously constructed weak learner and to provide less weight to observations correctly classified. A new classifier (tree) is constructed on the reweighted data, and the weight given to the new tree is based on how well it fits the reweighted training data. The final classifier for a new observation is based on a weighted average of the individual tree predictions.

3.2.2 Ensemble improvement methods

Researchers have followed two paths to improve ensemble models. The first is to maximize the margin. Schapire, Freund, Bartlett, and Lee (1998) defined the margin of a training observation as the difference between the weighted vote assigned to the correct label and the maximal weighted vote assigned to any single incorrect label. In the case of a binary response variable, the margin is the signed difference between the weighted vote proportions for the correct label and the incorrect label; the larger the margin, the more confidence in the classification.

We assume $(\mathbf{x}, y)$ is an observation from a population following some distribution $P(\mathbf{x}, y)$. The $p \times 1$ vector $\mathbf{x}$ is the vector of predictor variable values and y is the binary outcome. We are given an ensemble of T trees $\{t_1, t_2, ..., t_T\}$. The trees are assigned corresponding weights $\alpha_1, \alpha_2, ..., \alpha_T$. The prediction of a tree t is denoted by $h_t(\mathbf{x}) \in \{-1, 1\}$. The margin of the *i*-th observation is given by

$$m_i = \frac{y_i \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)}{\sum_{t=1}^{T} \alpha_t},$$

where $-1 \le m_i \le 1$. A margin of +1 indicates that all of the classifiers correctly predicted the observation, while a margin of –1 indicates that all of the classifiers incorrectly predicted the observation. According to Reyzin and Schapire (2006), when other factors are equal, higher margins should result in lower error. Freund and Schapire (1997) also found that AdaBoost increases margins quickly compared to other models, and suggested that improving the margins should lead to better generalization error. This is called the "large margins theory."

Following the "large margins theory," other researchers have built different methods to enlarge the margins in order to improve the predictive performance of an ensemble. Grove and Schuurmans (1998) maximized the minimum margin using linear programming techniques. Reyzin and Schapire (2006) have considered maximizing the mean or median of the margins, while Mason, Bartlett, and Baxter (2000) maximized other functions of the margins. More recently, Shen and Li (2010) developed a quadratic programming technique for simultaneously maximizing the mean and minimizing the variance of the margin distribution, which has been shown to have better performance in many cases.

Another direction follows Breiman's (2001) strength and diversity theory. He concluded that random forests with higher strength trees and lower correlations among the trees have better performance. Some researchers developed new measures of strength and diversity of an ensemble, then used forward or backward selection (Margineantu and Dietterich 1997) and genetic algorithms (Ko, Sabourin, and Britto 2009) to choose the classifiers from the ensemble so as to maximize the strength and diversity. Most of the methods succeed in thinning an ensemble model, but have difficulty improving the performance.

3.3 Proposed Method

3.3.1 Notation for proposed method

We assume (x, y) is an observation from a population following some distribution $P(x, y)$. The $p \times 1$ vector x is the vector of predictor variable values and y is the binary outcome. Given a sample from the population, we denote observations as i and trees as t. These trees are either from bagging or a random forest.

When a bootstrap sample is taken from the original training data to construct a tree, some of the training data are part of the bootstrap sample and used in the tree construction (inbag data), while other training observations are not in the bootstrap sample and not used in constructing the tree (outbag data). We define the inbag-outbag matrix Φ that denotes whether an observation is used in the construction of a tree or not. The dimension of Φ is $n \times T$, where n is the number of observations and T is the number of trees. The values in the matrix Φ are 0's and 1's, where 1 denotes inbag data, *i.e.*, the corresponding observation is used in building the tree, and 0 denotes outbag data, *i.e.*, the corresponding observation is not used in building the tree.

We use $n_{i|t}$ to denote the number of observations i for a given tree t and $n_{t|i}$ to denote the number of trees t for a given observation i. $n^c_{t|i}$ is the number of correct predictions of observation i by its corresponding trees t. Therefore, the outbag correct prediction proportion (OCPP) is equal to $\frac{n^c_{t|i}}{n_{t|i}}$ and the outbag margin is the difference between the outbag correct prediction proportion and the outbag incorrect prediction proportion, which is $\frac{2n^c_{t|i} - n_{t|i}}{n_{t|i}}$.

3.3.2 Margin and correct prediction proportion (OCPP)

Given a sample of training data, after we fit an ensemble of trees, how do we calculate the margins? The usual method is to use the training data. First, fit an ensemble model to the training data and predict the training data by the model. Then compare the prediction to the response variable, calculate the difference between the correct predictions and the incorrect predictions, and average the difference over the number of trees for each observation. This average is the usual measure of the margins. However, this training margin calculation is biased

to the training data. The training margin can be decomposed into three parts, the inbag predictions (predicted by the trees built from the inbag data), the outbag correct predictions, and the outbag incorrect predictions. Trees of a random forest are fully grown until the Gini index is equal to 0, *i.e.*, all the data in each terminal node are homogeneous with respect to the *y* values. Therefore the inbag predictions are always correct and the training margin overestimates the population margin. Also after bootstrapping (sampling with replacement), approximately 63.2% of the training data are selected as inbag data, but the variation in this percentage is fairly large from bootstrap sample to bootstrap sample. The inbag predictions will affect our measurement of the size of the margins. A better measure of margin is outbag margin. The data not used in building the tree are the outbag data. An observation can be inbag for some trees while outbag for other trees. If a tree predicts an outbag observation, the prediction is referred to as an outbag prediction.

Another concept we will use in the article is outbag correct prediction proportion (OCPP). Given an observation i, $OCPP_i = \frac{n^c_{t|i}}{n_{t|i}}$ is the ratio of the correct outbag predictions over the total number of outbag predictions for observation i.

3.3.3 Random forests vs. bagging

Both bagging and random forests use bootstrapped data to build fully-grown trees, so the recursive partitioning will not stop until one observation is left in a node or the data in a node are homogeneous with respect to the *y* values. The prediction of a terminal node of a tree is equal to the majority vote of the data in the node. Because of the homogeneity of the data, the prediction of a terminal node is the same as the common response variable value of the data in the node.

Therefore, predictions of inbag data are always correct. Beyond that, if we fit both trees (the bagging tree and the random forest tree) to the same bootstrapped data, the inbag data are the same for both trees and the predictions in either tree are correct, therefore the inbag predictions are the same for both tree predictions. Bagging depends on CART, which maximizes the Gini index reduction by searching all the split variables and split values at each node split. The method of random forests randomly selects a very small group of split variables and then maximizes the Gini index reduction by searching the predictor variables and split values of the group. Since the inbag predictions of both methods are the same, but random forests compare favorably to bagging, there must be some difference in their outbag predictions.

	Trees																			
Obs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	1	0	1	0	1	1	1	0	0	1	0	1	1	0	1	0	0	0
2	1	0	1	0	0	1	1	1	0	**1**	0	0	1	1	1	0	0	1	1	0
3	0	0	1	1	0	0	1	0	1	0	1	1	0	0	1	1	1	0	0	1
4	1	1	0	1	1	1	0	1	0	1	1	1	1	0	0	1	0	1	1	1
5	1	1	0	1	1	1	0	0	1	1	1	0	0	1	0	1	1	1	0	0
6	0	1	1	0	1	0	1	1	1	0	**0**	1	1	1	0	0	1	1	1	1
7	1	0	0	1	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1
8	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1	0
9	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0	1	0	0	1	0
10	1	0	1	0	1	0	1	0	1	0	1	1	1	1	0	1	1	1	0	1

Table 1. The inbag-outbag matrix of bagging

For the sake of explanation, consider the inbag-outbag matrix, shown in Table 1. The matrix has 10 rows and 20 columns, denoting 10 observations and 20 trees. We use 1's to denote inbag data and 0's to denote outbag data. The bold 1 in the matrix indicates that the second observation is used in building the 10^{th} tree, while the bold 0 means the 6^{th} observation is not used in building the 11^{th} tree. As we have already indicated, the predictions of 1's are always correct, while the predictions of 0's could be either correct or incorrect.

We first use bagging to fit the data and make a prediction for each observation. To make a distinction, we now denote inbag data with 2's, correctly predicted outbag data as 1's, and incorrectly predicted outbag data as 0's. Then we get Table 2 shown below:

	Tree																			
Obs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	2	2	0	2	1	2	2	2	0	1	2	1	2	2	0	2	1	1	0
2	2	1	2	1	1	2	2	2	0	2	1	0	2	2	2	1	1	2	2	1
3	1	0	2	2	1	1	2	1	2	1	2	2	1	1	2	2	2	1	1	2
4	2	2	0	2	2	2	0	2	1	2	2	2	2	1	0	2	0	2	2	2
5	2	2	1	2	2	2	1	0	2	2	2	1	1	2	1	2	2	2	0	1
6	1	2	2	1	2	0	2	2	2	1	1	2	2	2	1	1	2	2	2	2
7	2	1	0	2	2	2	1	2	0	2	2	1	2	0	2	2	0	2	0	2
8	2	2	2	1	1	2	2	1	2	2	2	1	1	2	2	1	2	1	2	1
9	0	2	1	2	2	2	0	2	1	2	0	2	2	2	0	2	0	0	2	0
10	2	0	2	0	2	1	2	0	2	1	2	2	2	2	1	2	2	2	1	2

Table 2. Inbag-outbag matrix of bagging with indications of inbag data (2) and the correct (1) and incorrect (0) predictions of outbag data

We now sort each row of Table 2 in descending order, putting 2's to the left of the table, 1's in the middle, and 0's to the right of the table. Note that this means the columns no longer correspond to a specific tree. We then sort the rows (observations) in descending order of OCPP, which leads to Table 3.

	Column																				Bagging
Obs	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	OCPP
8	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1.0
3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	.90
6	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	.86
2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	0	.77
5	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0	.75
10	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	0	0	0	.57
1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	0	0	0	.50
7	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0	0	0	0	0	.38
4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	0	0	0	0	.33
9	2	2	2	2	2	2	2	2	2	2	2	1	1	0	0	0	0	0	0	0	.22

Table 3. Ordered inbag-outbag matrix of bagging

Table 3 can be viewed as a rotated stacked bar chart. Each observation now is a stacked bar and there are ten bars. In each bar, we use borders in different colors to separate the numbers, a black border between 2's and 1's, and a red border between 1's and 0's. The outbag correct prediction proportion (OCPP) for an observation is the ratio between the number of 1's and the number of outbag data (1's and 0's) in that observation's row. For example, in the 3rd row (obs 6), there are six 1's and seven outbag data, so the OCPP is $6/7 \approx .86$. We order the observations in decreasing order by the OCPP from top to bottom. The last column on the right indicates the OCPPs.

	Column																			Bagging
Obs	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	OCPP
8	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0	.80
3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	0	.90
6	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	.71
2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	0	.67
5	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	0	.63
10	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	.71
1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0	0	0	.60
7	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	0	.63
4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0	0	0	.50
9	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	0	0	0	0	0	.44

Table 4. Random forest inbag-outbag matrix ordered by decreasing bagging OCPP

Table 4 is the inbag-outbag matrix of the random forest after following the above steps used in the bagging example. There are two differences from bagging. The first is that 1's and 0's are separated by a blue border. The second difference is that the observations are sorted by the order of the OCPPs from the bagging model, so we can see that the OCPPs in Table 4 are not in decreasing order from top to bottom according to the random forest OCPPs.

If we put Tables 3 and 4 together and remove the numerical values, we have a plot that compares the OCPPs between random forest and bagging, ordered by the bagging OCPPs. Figure 1 is the plot for the Sonar data with a tree size of 100 (156 training observation rows and

100 columns). We treat the border (black, red, or blue) of each observation as a point and connect the points across the rows with three lines.

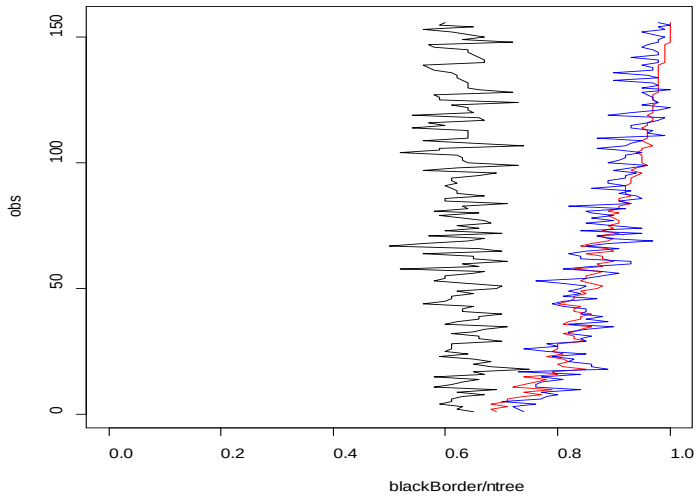

Figure 1. OCPP comparison between bagging (red) and random forest (blue) along with inbag-outbag boundary (black)

We can see the black border in Figure 1 has a large variance from one observation to another. We ordered the observations in decreasing order of OOB correct prediction proportion (OCPP) of bagging. If we order them by training predictions, the result could be misleading. It is possible that one observation has high training predictive accuracy because that observation appears in more trees than other observations, or equivalently we can say the observation is inbag for more trees than other observations. Since inbag predictions are always correct, this will inflate the predictive accuracy of the observation and the order of observations will be biased to the training data. By removing those inbag predictions, the order better reflects the predictive accuracy of the bagging model over the observations.

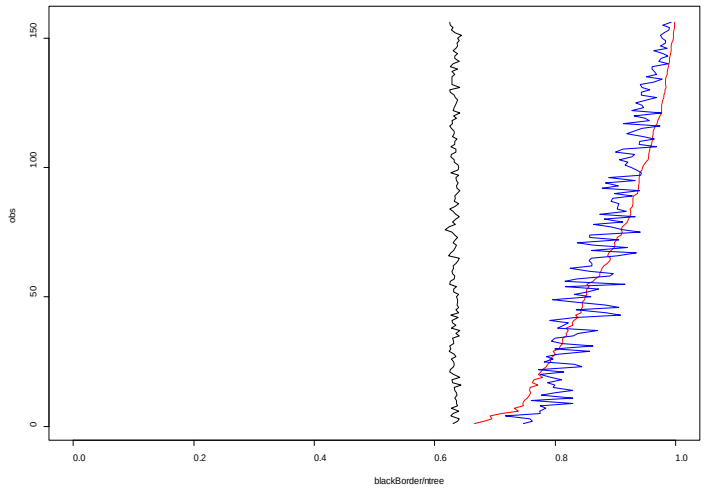

Figure 2. Average OCPP comparison between bagging (red) and random forest (blue) along with inbag-outbag boundary (black)

Figure 2 is a plot that summarizes the average results from a simulation of 100 bagging and random forest models, including the average number of inbag tree predictions (2's) for each observation, the average number of outbag correct predictions (1's) by bagging (indicated by red border), and the average number of outbag correct predictions (1's) of random forests (indicated by blue border).

The variance of the blue border (for random forests) is larger than that of the red border (for bagging) because the order of the observations (rows) is determined by the outbag correct prediction proportions (OCPPs) of bagging. For those observations with large OCPPs (near the top part of the figure), the red border is to the right of the blue border, which means that the OCPPs of bagging are larger than that of random forests. As to those observations with small OCPPs (near the bottom part of the figure), the red border is to the left of the blue border, which means that the OCPPs of bagging are smaller than that of random forests. It appears like the blue border is turned counterclockwise from the red border.

The result of the counterclockwise turn is that the blue border is more vertical than the red border, which means the variance of the OCPPs of the random forest is smaller than that of bagging. The point where the blue border crosses the red border is higher than .5. If we draw a horizontal line below this point, we will find the line first crosses the red border then the blue border. This means that for the same observation, random forests has a higher number of outbag correct predictions and outbag correct prediction proportion (OCPP) than bagging. Taking Figure 2 as an example, the obs-axis is the ordered observation numbers (for the Sonar data, the range of obs is from 1 to 156 and the step size is 1). The higher the obs value, the larger the outbag correct prediction proportion (OCPP). If obs = 156, the corresponding OCPP is the largest. Assuming when obs = 50, the blue border crosses the red border, which means that if obs is larger than 50, the blue border is to the left of the red border, and if smaller than 50, then the blue border is to the right of the red border. When obs = 37, the OCPP of bagging is .5. This means if obs > 37, the OCPP > .5 and the outbag ensemble prediction is correct, while if obs < 37, the OCPP < .5 and the outbag ensemble prediction is incorrect. When obs = 37, the blue border is to the right of the red border, indicating that the random forest has more outbag correct predictions (1's) than bagging. In other words, when the OCPP of random forests =.5, then obs < 37. Therefore, compared to bagging, the random forest has more correct predictions and the prediction error is lower. We believe that random forests have lower error than bagging since the variance of the outbag margins is reduced while the mean remains the same (Shen and Li, 2010).

We might be able to achieve further improvement if we can continue to push the blue border counterclockwise by boosting the low OCPP observations.

3.3.4 Boosting the random forest

To push the blue border further counterclockwise, we need to find a turning point, at which the outbag correct prediction proportion (OCPP) is larger than .5. The new ensemble model should pay more attention to those observations below the point than those above the point. We can change the weight of each observation as in AdaBoost or we can change the number of appearances of the observation in the inbag data from bootstrapping. In other words, we let poorly predicted observations appear more often while reducing the appearance of well predicted observations.

	Trees																			
Obs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	2	2	0	2	1	2	2	2	1	0	2	1	2	2	1	2	0	1	0
2	2	0	2	1	1	2	2	2	1	2	1	1	2	2	2	**0**	1	2	2	1
3	0	1	2	2	1	1	**2**	1	2	1	2	2	1	1	2	2	2	1	1	2
4	2	2	1	2	2	2	1	2	0	2	2	2	2	0	1	2	0	2	2	2
5	2	2	1	2	2	2	0	1	2	2	2	1	0	2	1	2	2	2	0	1
6	1	2	2	1	2	0	2	2	2	1	1	2	2	2	0	1	2	2	2	2
7	2	1	0	2	2	2	1	2	1	2	2	0	2	1	2	2	0	2	1	2
8	2	2	2	1	0	2	2	0	2	2	2	1	1	2	2	1	2	1	2	1
9	1	2	0	2	2	2	1	2	0	2	0	2	2	2	1	2	1	0	2	0
10	2	1	2	1	2	1	2	1	2	0	2	2	2	2	1	2	2	2	0	2

Table 5. Inbag-outbag matrix of random forest with indications of correct and incorrect predictions of outbag observations

	Column																				RF
Obs	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	OCPP
3	2	2	2	2	2	2	2	**2**	2	2	1	1	1	1	1	1	1	1	1	0	.9
8	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0	.8
6	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	.71
10	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	.71
2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	**0**	0	.67
5	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	0	.63
7	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	0	.63
1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	0	0	0	.6
4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0	0	0	.50
9	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	0	0	0	0	0	.44

Table 6. Ordered random forest inbag-outbag matrix by random forest OCPP

Table 5 is the inbag-outbag matrix for the random forest. Table 6 is the ordered inbag-outbag matrix of the random forest. In Table 6, suppose we choose .71 as the turning point. The purple horizontal line divides the table into two parts. The outbag correct prediction proportions (OCPPs) above the line are larger than or equal to .71 and the OCPPs below the line are smaller than .71. We want to improve the OCPPs of those observations below the line. The method is to convert some of the outbag false predictions (0's) below the line into inbag data (3's). For example, the 2nd observation in Table 5 has a rank of 5th for OCPP in Table 6. It is not used to build the 16th tree (the bold 0 in Table 5) or we can say the 5th observation is an outbag observation for the 16th tree. At the same time, the prediction of the 16th tree on the 5th observation is incorrect, as signaled by its value of 0. In Table 6, the OCPP of the 5th row is .67, with seven 1's and two 0's. If we randomly select a 0 and convert it into an inbag observation (marked as 3), we will have seven 1's and one 0 and the OCPP is 7/8 = .875 > .71. Although the improvement cannot directly reflect the increase of the predictive accuracy, it gives us a standard to decide how many 0's should be converted into inbag data. In this case, we want all the observations below the purple line to have higher OCPPs than .71. If the observation has very low OCPP, like row 10 in Table 6 (OCPP = .44), more 0's need to be converted into inbag data. By making these changes, the model will pay more attention to those 0's.

One problem in this method is that when we convert some 0's into inbag data, more data are added to the original bootstrap data set. Therefore, the size of the updated data set is larger than the original size (156 in this case). If we want to keep the original size, we need to move some inbag data to outbag.

The observations above the purple line in Table 6 have large OCPPs. If we remove some inbag observations in some trees, the performance may drop, but if the new OCPPs are larger

than .5, the predictions of those observations are probably still correct. For example, the OCPP of the first row in Table 6 is .9. In Table 5, it is the prediction of the 7th tree on the 3rd observation. Since it is an inbag observation, the prediction is correct. If we remove the observation from the 7th tree and replace it with some other observation (or observations, if the removed observation is repeated in the bootstrap sample for that tree), the correct prediction of the observation is unlikely to change because the OCPP is large, much higher than 0.5.

Therefore, we remove some inbag data in some trees by marking them with 4's and replacing them with those observations marked as 3's. We usually remove more 4's than the number of 3's because we want to give 3's more attention. So we bootstrap the 3's to the size of the 4's. Table 7 shows the result after these steps and Table 8 shows the change in the inbag-outbag matrix.

	Column																			RF
Obs	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	OCPP
3	2	2	2	2	2	4	4	**4**	4	4	1	1	1	1	1	1	1	1	1	0	.90
8	2	2	2	2	2	2	2	4	4	4	4	4	1	1	1	1	1	1	0	0	.80
6	2	2	2	2	2	2	2	2	4	4	4	4	4	1	1	1	1	1	0	0	.71
10	2	2	2	2	2	2	2	2	4	4	4	4	4	1	1	1	1	1	0	0	.71
2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	**3**	0	.67
5	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	3	0	0	.63
7	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	3	0	0	.63
1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	3	3	0	0	.60
4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	3	3	0	.50
9	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	3	3	3	0	0	.44

Table 7. Ordered random forest inbag-outbag matrix after data identification

	Trees																			
Obs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	2	2	0	2	1	2	2	2	1	3	2	1	2	2	1	2	3	1	0
2	2	0	2	1	1	2	2	2	1	2	1	1	2	2	2	3	1	2	2	1
3	0	1	2	2	1	1	2	1	4	1	4	2	1	1	4	4	2	1	1	4
4	2	2	1	2	2	2	1	2	3	2	2	2	2	0	1	2	3	2	2	2
5	2	2	1	2	2	2	3	1	2	2	2	1	0	2	1	2	2	2	0	1
6	1	4	4	1	2	0	4	2	2	1	1	2	2	2	0	1	4	4	2	2
7	2	1	0	2	2	2	1	2	1	2	2	3	2	1	2	2	0	2	1	2
8	4	2	4	1	0	2	2	0	2	4	2	1	1	4	2	1	2	1	4	1
9	1	2	3	2	2	2	1	2	3	2	0	2	2	2	1	2	1	0	2	3
10	2	1	2	1	4	1	4	1	4	0	2	4	2	2	1	2	2	2	0	2

Table 8. Random forest inbag-outbag matrix after data identification

We get the updated inbag- outbag matrix by changing 0's, 1's, and 4's into 0's and changing 2's and 3's into 1's, which is shown in Table 9.

	Trees																			
Obs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	1	1	0	1	0	1	1	1	0	1	1	0	1	1	0	1	1	0	0
2	1	0	1	0	0	1	1	1	0	1	0	0	1	1	1	1	0	1	1	0
3	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0
4	1	1	0	1	1	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1
5	1	1	0	1	1	1	1	0	1	1	1	0	0	1	0	1	1	1	0	0
6	0	0	0	0	1	0	0	1	1	0	0	1	1	1	0	0	0	0	1	1
7	1	0	0	1	1	1	0	1	0	1	1	1	1	0	1	1	0	1	0	1
8	0	1	0	0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	0
9	0	1	1	1	1	1	0	1	1	1	0	1	1	1	0	1	0	0	1	1
10	1	0	1	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	1

Table 9. Final updated inbag-outbag matrix

Then we build new trees from the updated inbag-outbag matrix. Each column in Table 9 denotes which observation will be used in constructing the tree. If the entry is a 1, then the observation will appear in the tree; otherwise, it will be an outbag observation for that tree.

However, an individual tree is actually built on bootstrap data, which is a sample with replacement from the training data. This means that some of the inbag data will appear more than once. From the inbag-outbag matrix, we know which observation will appear in the bootstrap

data, but we don't know how many times it will appear. For example, in Table 5, tree 9 is built on 6 inbag data (1, 3, 5, 6, 8, 10). Assume the corresponding bootstrap data is (10, 3,10, 8, 6, 5, 1, 3, 10, 6). We can see that (3, 6, 10) appears more than one time. After we manipulate the data, in Table 9, tree 3 includes data (1, 4, 5, 6, 8, 9). The observations (3, 10) disappear while the observations (4, 9) are added to the bootstrap data. Since observations 3 and 10 appear together 5 times, observations 4 and 9 need to appear 5 times. We first remove observation 3 and 10 from the bootstrap data set and then sample observations 4 and 9 five times with replacement to add to the bootstrap data set. The updated bootstrap data is (9, 9, 4, 8, 6, 5, 1, 4, 9, 6). The new trees will be built on the updated bootstrap data set.

The new trees we construct are different from the original trees in the random forest, but we will mimic the random forest's trees to the extent possible, which will be shown in the next section.

3.3.5 Mimicking trees

After we have the updated data set and inbag-outbag matrix, we need to build new trees to form a new random forest. If we use the traditional random forest method to build new trees from scratch, we need to randomly select a group of variables and use the one that maximizes the Gini index reduction at each node split. We may build a completely different tree even though we changed only a small part of the data.

The extra randomness injected by using the old tree building method could result in a very different tree. We want our new tree to be as similar as possible to the old tree, taking into account the slight modification of the data. We want to mimic the old tree but not replicate it.

Node	Left daughter	Right daughter	Split var	Split point	status	prediction
1	2	3	51	.01155	1	0
2	4	5	6	.08345	1	0
3	6	7	47	.04940	1	0
4	0	0	0	.00000	-1	2
5	8	9	23	.79425	1	0
6	10	11	29	.48045	1	0
7	12	13	14	.19035	1	0
8	14	15	1	.02125	1	0
9	0	0	0	.00000	-1	1
10	0	0	0	.00000	-1	1
11	0	0	0	.00000	-1	2
12	16	17	34	.76620	1	0
13	18	19	30	.27150	1	0
14	0	0	0	.00000	-1	1
15	0	0	0	.00000	-1	2
16	0	0	0	.00000	-1	1
17	0	0	0	.00000	-1	2
18	20	21	10	.21265	1	0
19	22	23	17	.69430	1	0
20	0	0	0	.00000	-1	1
21	0	0	0	.00000	-1	2
22	0	0	0	.00000	-1	1
23	24	25	18	.73045	1	0
24	0	0	0	.00000	-1	2
25	0	0	0	.00000	-1	1

Table 10. Matrix representation of a random forest tree

Table 10 is a matrix representation of a random forest tree. The first column contains the parent nodes. The first node is 1, which is the root node. The second and third columns are used to store child node indices. The fourth column is the split variable and the fifth column is the split point. The sixth column is the status, with 1 denoting an internal node and -1 denoting a terminal node. The last column is the prediction. It only gives predictions in terminal nodes, otherwise it is 0. The tree diagram of the matrix is shown in Figure 3.

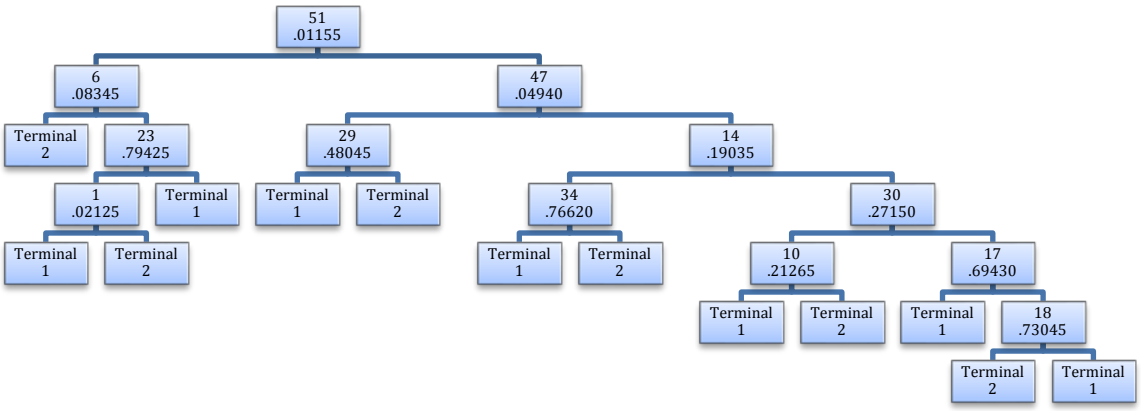

Figure 3. Original Tree Diagram

We want to maintain the tree structure and split variables to the extent possible while changing the split points to better fit the updated data. If a node is empty as a result of these changes, then we trim it and its descendant nodes from the tree. On the other hand, if a terminal node is not pure (*i.e.*, it contains observations with different *y* values), then we will let the tree continue to grow that node. Table 11 is the new tree matrix.

	Left daughter	Right daughter	Split var	Split point	status	prediction
1	2	3	51	.02105	1	0
2	4	5	6	.17740	1	0
3	6	7	47	.23215	1	0
4	8	9	12	.14535	1	0
5	0	0	0	.00000	-1	2
6	10	11	29	.59085	1	0
7	0	0	0	.00000	-1	1
8	0	0	0	.00000	-1	2
9	12	13	17	.69040	1	0
10	0	0	0	.00000	-1	1
11	0	0	0	.00000	-1	2

12	14	15	30	.77375	1	0
13	16	17	52	.01040	1	0
14	18	19	1	.01270	1	0
15	20	21	9	.22855	1	0
16	0	0	0	.00000	-1	2
17	0	0	0	.00000	-1	1
18	22	23	12	.23905	1	0
19	0	0	0	.00000	-1	1
20	0	0	0	.00000	-1	2
21	0	0	0	.00000	-1	1
22	0	0	0	.00000	-1	2
23	0	0	0	.00000	-1	1

Table 11. Matrix representation of new random forest tree

We can see the split variables of the first three rows are the same as in the old tree, while the split points are different. The fourth row in the old tree is a terminal node, but in the new tree, it continues growing. The fifth row in the new tree is a terminal node, but it is not in the old tree, so the new tree trims the nodes below the fifth node. The tree diagram of the new tree is shown in Figure 4. By comparing to Figure 3, we can see that Figure 4 grows the node on the left of Figure 3 while trimming the nodes on the right side of Figure 3.

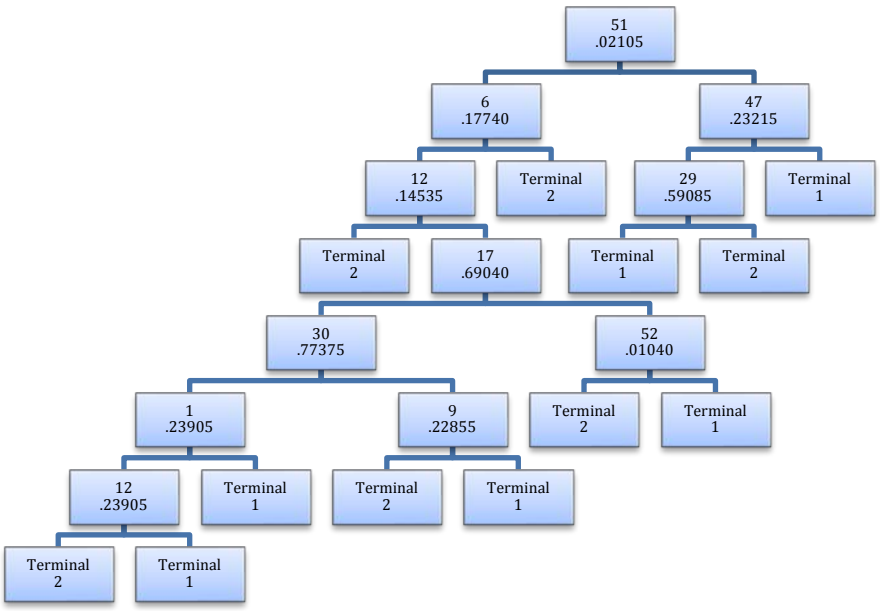

Figure 4. Mimic tree of the tree diagram in figure 3

3.3.6 Evaluation of proposed strategy

Steps 1-11 below describe the proposed algorithm and a simulation for evaluating the proposed strategy.

Step 1: Partition the data set into training and test sets with a ratio of .75:.25.

Step 2: Fit a random forest model to the training data set, predict the test data set, and compute the test set error rate.

Step 3: For each bootstrap sample used to create a tree in the random forest, find the inbag and data (denoted by 1 and 0), and create the inbag-outbag matrix (where each column corresponds to a bootstrap sample and its tree).

Step 4: For each tree, a column in the inbag-outbag matrix, change 1's into 2's. As for 0's, compare the corresponding predictions to the response value (y) of the bootstrap data set. If they match, which means a correct outbag prediction, then mark as a 1; otherwise, leave as a 0. Calculate the number of 1's for each observation (row) and divide by the sum of the number of the outbag predictions for the observation to obtain the outbag correct prediction proportion (OCPP). Sort the rows (observations) in the inbag-outbag matrix by the OCPP values and save in a vector.

Step 5: Set a cutoff (such as .9). Calculate the number of 0's for each observation with OCPP lower than .9. We convert some of these into inbag data so that the updated OCPP value of that observation reaches .9 (artificially). Randomly choose that number of 0's and convert into 3's. These observations are now "inbag" and will be used to update the trees for which they are "inbag".

Step 6: Within each column in the inbag-outbag matrix, calculate the number of 3's, and randomly choose the same number of 2's with OCPP larger than .9. Convert those 2's into 4's. Calculate the number of the observations indicated by the 2's we just choose in the bootstrapping data set and replace by the observations denoted as 3's. Convert the 2's in the inbag-outbag matrix into 4's.

Step 7: Set a push ratio (from .632 to .350). Calculate the average number of inbag predictions, for which OCPP is higher than .9, need to convert into outbag predictions so that the inbag-outbag ratio is changed from .632 to .350. Randomly choose the number of values in inbag-outbag matrix and convert the 2's into 5's. Remove those data in the bootstrapping data set and replace them with the observations denoted as 3's.

Step 8: Convert the 0's, 1's, 4's, and 5's in the inbag-outbag matrix into 0's, and 2's and 3's into 1's to form a new inbag-outbag matrix.

Step 9: Fit a mimic tree to the new bootstrap data set.

Step 10: Repeat Step 4 to Step 9 *ntree* times to form a new random forest. Calculate the test data error.

Step 11: Repeat Step 1 to Step 10 *nsim* times.

3.4 Simulation Results

We now apply the method to several different data sets. In random forests, *mtry* is a parameter of how many variables randomly selected in each split. The default *mtry* in the random forest R package (randomForest) is the floor of the square root of the number of predictors. For example, Sonar has 60 predictors, so *mtry* is 7. We use the default *mtry* when we rebuild our random forest.

The first example is the Sonar data. The Sonar data set has 60 predictor variables and 208 observations. We want to choose a turning point as large as possible, like .9, because our experiments show that the larger the turning point is, the more attention the poorly predicted observations get, and the better performance the ensemble model has; however, in the Sonar data, there are not enough observations with OCPP above .9. Therefore, the turning point we choose is .8, which means the observations with OCPP below .8 need to be repeated more times, while some observations with OCPP above .8 needs to be removed so that the whole inbag-outbag ratio goes from .632 to .450. We need not repeat those weak observations in all trees, but only in those trees with incorrect outbag predictions. Actually those observations are not used in building the trees, so we bring them back with other inbag data and repeat more times. After we randomly remove a number of strong points equal to the number of repeated weak points, we will get a new distribution of data with the same number of observations (208 in Sonar). Then we build a new tree from the new data set.

Ntree (ratio:.632 : .450)	Random Forest	Rebuilt RF	Nb model	Difference
50	.1863	.1877	.1708	.0169
100	.1629	.1704	.1444	.0260
200	.1688	.1710	.1450	.0260
300	.1777	.1787	.1546	.0241
400	.1667	.1748	.1483	.0265
500	.1815	.1808	.1473	.0335

Table 12. Simulation results for Sonar data

The first column of Table 12 is the number of trees, ranging from 50 to 500. The second column is the test set error rate of randomForest R package, and the third column is the error of our rebuilt random forest. We use the outcome of the R package as a reference for our rebuilt random forest. The fourth column is the new model and the last column is the improvement

compared with the rebuilt random forest. We can see the new model significantly improves the performance.

The second data set is the Ionosphere data set (Table 13). The turning point we choose here is .9 and we push the ratio to .350, so as to give poorly predicted observations more attention. Our principle is to choose the push ratio as small as possible. In other words, we tend to give poor predicted observations more attention but not to the point of making errors with the well-predicted observations. The push ratio .350 may not be the optimal one, but the results appear to be fairly robust to the choice.

Ntree (ratio:.632 : .350)	Random Forest	Rebuilt RF	Nb model	Difference
50	.0668	.0692	.0644	.0048
100	.0698	.0707	.0620	.0087
200	.0707	.0718	.0640	.0076
300	.0680	.0688	.0636	.0052
400	.0720	.0725	.0669	.0056
500	.0688	.0682	.0633	.0049

Table 13. Simulation results for the Ionosphere data

The new model still significantly improves on the rebuilt random forest.

The third data set (Table 14) is the breast cancer data set. We separate the turning point into two parts, the low part is .9, which means the observations with OCPP lower than .9 need to be repeated, and the high part is .99, which means we remove those observations predicted perfectly. Most predictive models do very well with this data set. Even if we push the ratio to .05, which means although we almost remove those perfectly predicted observations from the random forest, the performance remains the same and also the OCPP of these observations is still 1 or close to 1. Therefore, the new model cannot reduce the variance of the outbag proportions since the data set is so easy to predict.

We also notice that over 80% of the data have an OCPP over .9 and two-thirds of them are actually 1. Only a very small fraction of the data has an OCPP less than .5.

Ntree (ratio:.632 : .05)	Random Forest	Rebuilt RF	Nb model	Difference
50	.0311	.0309	.0309	0
100	.0291	.0292	.0288	.0004
200	.0292	.0294	.0295	-.0001
300	.0296	.0294	.0295	-.0001
400	.0281	.0278	.0292	-.0014
500	.0279	.0281	.0277	.0004

Table 14. Simulation results for the Breast Cancer data

The fourth data set is the Circle data, which is a simulated data set. The results for Circle data are similar to those for breast cancer data. We set the parameters similar to the Ionosphere data, with a .9 turning point and .350 push ratio. The outcome of the new model has some improvement but is not significant.

Ntree (ratio:.632 : .350)	Random Forest	Rebuilt RF	Nb model	Difference
50	.0828	.0821	.0809	.0012
100	.0852	.0840	.0803	.0037
200	.0851	.0855	.0815	.0040
300	.0837	.0829	.0799	.0030
400	.0793	.0791	.0776	.0015
500	.0805	.0802	.0779	.0023

Table 15. Simulation results for the Circle data

The last data set (Table 16) is the RingNorm1500 data, which is also a simulated data set. We set the parameter the same as for the Ionosphere data and Circle data. We first use default *mtry*=4, and the performance of the new model overwhelms that of random forest.

Ntree (ratio:.632 : .350)	Random Forest	Rebuilt RF	Nb model	Difference
50	.0550	.0548	.0380	.0168
100	.0516	.0531	.0357	.0174
200	.0515	.0510	.0315	.0195
300	.0501	.0501	.0318	.0183
400	.0493	.0492	.0305	.0187
500	.0507	.0504	.0325	.0179

Table 16. Simulation results for the RingNorm1500 data

However, if we set *mtry*=1, the performance of the random forest is still significantly worse than the new model, but the difference is not as great. The reason might be when we set *mtry*=1, we add more diversity to the model and thus reduce the variance of the OCPP. We set *mtry*=1 with other data sets, but only for RingNorm1500 does it improve the performance; other data sets have the same performance or even worse.

3.5 Conclusion and Discussion

When we compare the margin plot of random forests and bagging, we notice that random forests squeeze the outbag margin distribution of bagging. Inspired by this, we built a new model to squeeze random forests so as to improve the predictive performance. The results is positive, the predictive performance is improved in most examples.

There is one problem we need to pay attention to: how to choose the push ratio. We applied the new model to some data sets at different push ratios. The push ratio ranges from .35 to .60 by .005 and the tree size is 100. In each push ratio, the data is partitioned into training and test and the test error is calculated. This step is repeated 50 or 100 times and then average test error for each push ratio is calculated.

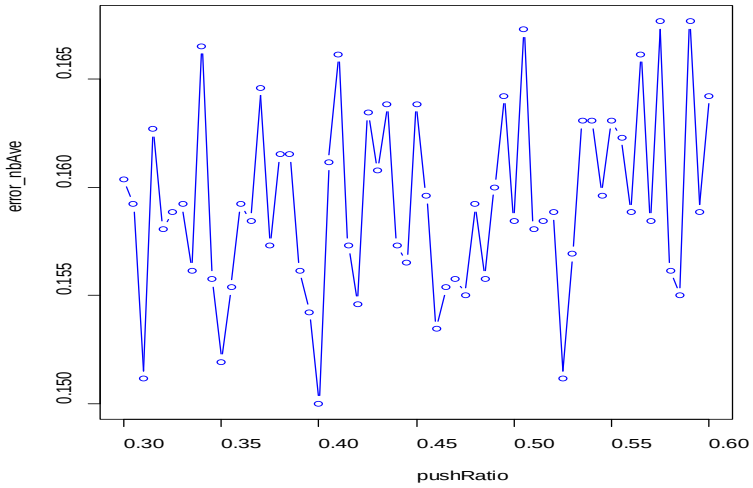

Figure 5. 50 simulations test error vs. push ratio for Sonar data

The above figure is the test error of Sonar vs. push ratio (100 trees and 50 simulations). The error of the random forests is much higher than all of the points in the graph and cannot be seen in the figure. From the plot we can see that the test error is robust to the choice of the push ratio in the Sonar data.

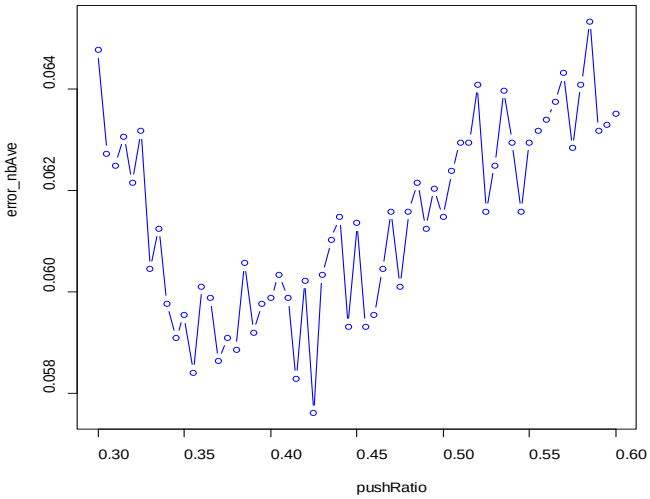

Figure 9. 100 simulations test error vs. push ratio for Ionosphere data

The second data set we test is Ionosphere (100 trees and 100 simulations). Still the error of the random forests is higher than the new model at each push ratio and does not show in the plot. There is a pattern in this plot; it seems that the test errors first decrease from .30 to .35 and then increase from .35 to .60.

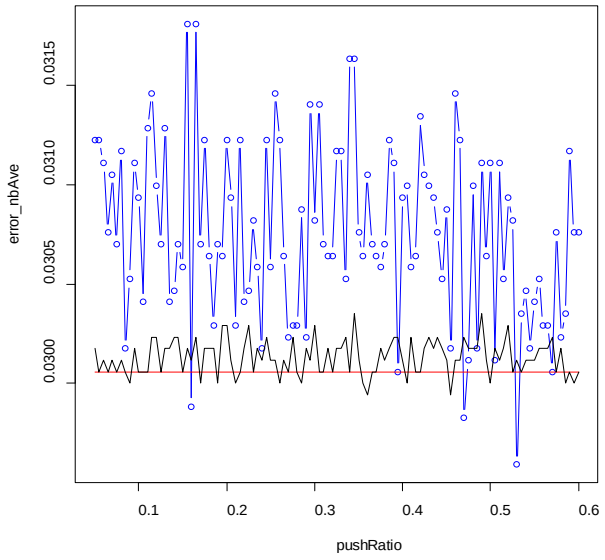

Figure 7. 100 simulations test error vs. push ratio for Breast Cancer data

The third data set is Breast Cancer (100 trees and 100 simulations). The range of the push ratio in this example goes from .05 to .60. We can see that the test errors of the new model are larger than random forests in most push ratios, which means that no matter how you change the push ratio, you cannot improve the predictive performance.

From the above three cases, the test error of the new model is robust to the push ratio. In practice, we suggest using a push ratio between .35 and .45.

REFERENCES

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26(2), 123-140.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, 11, 1493-1517.

Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5-32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and Regression Trees. Belmont, CA: Wadsworth.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Machine Learning*, 40, 1-22.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. *Machine Learning, Proceedings of the 13th International Conference*, 148-156.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 119-139.

Grove, A.J. and Schuurmans, D. (1998). Boosting in the limit: maximizing the margin of learned ensembles. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 692-699.

Ko, A.H., Sabourin, R., and Britto, A.D.S. (2009). Compound diversity functions for ensemble selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 23, 659-686.

Margineantu, D.D. and Dietterich, T.G. (1997). Pruning adaptive boosting. *Proceedings of the 14th International Conference on Machine Learning*, 211-218.

Mason, L., Bartlett, P., and Baxter, J. (2000). Improved generalization through explicit optimization of margins. *Machine Learning*, 38, 243-255.

Reyzin, L. and Schapire, R. (2006). How boosting the margin can also boost classifier complexity. *Proceedings of the 23rd International Conference on Machine Learning*, 753-760.

Schapire, R., Freund, Y., Bartlett, P., and Lee, W. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26, 1651-1686.

Shen, C. and Li, H. (2010). Boosting through optimization of margin distributions. *IEEE Transactions on Neural Networks*, 21(4), 659-666.

CHAPTER 4

BOOSTING WITH FULLY-GROWN TREES

4.1 Introduction

Ensemble models, like bagging, random forests, and AdaBoost, are aggregations of hundreds of single classifiers. They are more complicated but have better performance than single classifiers. Of the three ensemble models we mention above, the random forest model has better performance in general than bagging and compares favorably to AdaBoost (Breiman 2001).

Breiman (2001) mentioned that the performance of an ensemble model depends on the strengths of the individual classifiers in the ensemble and the correlations among them. Both bagging and random forests use fully-grown trees to fit bootstrapped data. Bagging trees are actually CART trees and they select the best split from all input variables at each node split, while random forest trees choose the best split from a subset of input variables at each node split. Compared to bagging, random forests increase the diversity among the trees while the strengths of the trees do not decrease by much.

AdaBoost uses weak learners, like stumps or pruned CART trees, to fit all of the training data. In each iteration, the misclassification error rate is calculated and used in calculating the new weights of observations. By assigning poorly predicted observations more weight and well predicted observations less weight, the trees grown on the weighted data set focus on the poorly predicted observations.

In this article, we use fully-grown trees instead of stumps or pruned CART trees to fit the weighted resampling data so as to improve the strengths of individual trees. We apply the new model to several data sets and compare the performance to random forests and AdaBoost. The

performance of the new model is better than random forests in most of the data sets and better than AdaBoost in some data sets.

In the next section, we introduce the three ensemble models, bagging, random forests, and AdaBoost. Section 3 shows the weighting method and resampling method in boosting models. We propose our fully-grown boosting method in section 4 and apply the method to several data sets. We show the results from of the different data sets in section 5 and compare the results of our model to random forests and AdaBoost. In the final section, we summarize and draw conclusions.

4.2 Ensemble Models

Ensemble models are aggregations of hundreds of single classifiers, usually trees. Each tree returns a prediction and the ensemble prediction is the average of all the tree predictions. There are three typical ensemble models, bagging, random forests, and AdaBoost, which use different methods to build and combine single classifiers. Bagging and random forests use fully-grown CART trees built on bootstrapped data sets, and the difference between them is that random forests select the split rule from a small, randomly chosen subset of predictor variables. Unlike the other two ensemble models, AdaBoost has more choices for a single classifier. It could be a stump, a tree with only one split, or a larger tree, like a tree with eight terminal nodes. The predictive performance of an ensemble model is generally better than for a single tree.

Breiman (1996) proposed a bagging method for generating multiple versions of a classifier and used these to create an aggregated classifier. Bagging first creates various bootstrapped data sets and then builds fully-grown CART trees from them. A fully-grown CART tree predicts the corresponding bootstrapped data set perfectly. The misclassifications of the

predictions of the bootstrapped training data are from the outbag predictions, the predictions for the data that do not appear in the bootstrapped data set. Since about 63.8% of the training data appear in the bootstrapped data, the strength of a fully-grown tree is usually high and difficult to classify as a weak learner, a classifier with the strength just slightly better than random selection. The performance of bagging is better than a single tree.

Random forests, like bagging, combine single trees with equal weights; however, the method in building trees in random forests is different from the CART method used in bagging. At each node split, a CART tree selects the best split rule from all predictor variables, while a random forest tree selects the best split rule from a small, randomly selected subset of predictor variables. Breiman (2001) mentioned that the performance of an ensemble model depends on the strengths of the individual classifiers and the diversity among them. The higher the strength and diversity are, the better the performance is. Compared to CART, random forests trees inject some randomness, so as to increase the diversity while maintaining the strength of the individual trees.

Unlike bagging and random forests, which are combined with equal weights, AdaBoost uses unequal weights to combine the single classifiers. The method to build a single classifier is also different from the other two ensemble models. In the initial run, AdaBoost assigns equal weights to each observation in the training data and then fits a weak learner, like stumps or pruned CART, to the training data. In the next run, the observations poorly predicted in the former run get more weight while the well predicted observations in the previous run have less weight. A tree weight is calculated and assigned to each tree. The ensemble prediction is the weighted average of all the single tree predictions. Freund and Schapire (1997) found that AdaBoost increases margins quickly compared to those for bagging and suggested that improving the margins should lead to better generalization error. Reyzin and Schapire (2006)

stated that “the margins explanation basically says that when all other factors are equal, higher margins result in lower error.” This has been referred to as the “large margins theory” in the boosting literature.

According to Breiman (2001), random forests have better performance than bagging and compares favorably to AdaBoost. Dietterich (2000) did an experimental comparison of bagging, boosting, and randomization. The randomization here is random split. At each node split, a split is randomly selected from 20 best splits. The single classifier he used is c4.5 instead of CART. He found that boosting gives the best results in most cases (with little or no noise). Random split is slightly better than bagging in low noise. However, with added classification noise, bagging is the best method.

4.3 Reweighting vs. Resampling

Bagging and random forests use bootstrapping methods to create diverse data sets, so as to build fully-grown trees with variation. Bootstrapping is a type of resampling method, sampling a training data set with replacement. The bootstrapped data set has the same number of observations as the training data; however, only about 63% of the training observations appear in the bootstrapped data. Which training observations appear is totally random. Therefore, randomness is injected to the bootstrapped data sets and the trees built on them have some diversity.

The AdaBoost model uses a reweighting method to create diverse data sets. By giving poorly predicted observations more weight and well predicted observations less weight, AdaBoost creates data sets with different distributions. However, the reweighting method can only be applied with base learners that are designed to handle example weights (Seiffert,

Khoshgoftaar, Hulse, Napolitano 2008). If this is not the situation, researchers apply an alternative approach, a resampling method, which bootstraps the training data set with probability proportional to the weight assigned to each observation.

Seiffert, Khoshgoftaar, Hulse, Napolitano (2008) compared these two methods by using 10 boosting algorithms, 4 learners and 15 data sets. They found that boosting by resampling performs as well as, or significantly better than, boosting by reweighting.

4.4 The Proposed Method

In this section, we propose a new ensemble method that blends the features of AdaBoost and random forests. In practice, we want to use fully-grown trees and the split variable selection technique of random forests in AdaBoost.

4.4.1 Problems and advantages of fitting fully-grown trees to AdaBoost

Friedman (2000) compared boosting with stumps to boosting with larger (eight-node) trees. He found that purely additive stump models seem to perform comparably to the larger trees for small data sets. On the larger data sets, eight-node tree models are significantly more accurate than the purely additive stump models. The strength of a stump is lower than a larger tree; however, when the tree size increases, the diversity among the trees decreases. At the other extreme are fully-grown trees. However, if we fit fully-grown CART trees to all training data, at each split, the split variable and point would be the same and there would be no difference among those full-grown trees. The ensemble model is just like hundreds of repeated trees stacked together. Therefore, the diversity in this situation is the lowest and the generalized performance is not good.

AdaBoost cannot continue with fully-grown trees as single classifiers. Both observation weights and tree weights are calculated from training prediction error. A fully-grown tree perfectly predicts the training data and the prediction error is zero. Therefore, after the first iteration, AdaBoost will stop.

4.4.2 Notation for the proposed method

We assume $(\mathbf{x}, y)$ is an observation from a population following some distribution $P(\mathbf{x}, y)$. The $p \times 1$ vector $\mathbf{x}$ is the vector of predictor variable values and y is the binary outcome. To implement fully-grown trees in AdaBoost, we apply some random forest features to AdaBoost. We first assign an equal weight $D_n^{(1)} = 1/n$ to each observation. Then a bootstrapped data set B_1 is created from the training data set with the probability proportional to the weight $D_n^{(1)}$. After that, we build a random forest tree t_1 on B_1. At each node split, the tree finds the optimal split rule from a small randomly selected subset of predictor variables. This procedure continues until all the observations in each terminal node are homogeneous. After the fully-grown tree is created, we use the tree t_1 to predict the training data set. The prediction is denoted by $h_1(x) \in \{-1, 1\}$. Then we calculate the prediction error ε_1. The tree weight is given by $\alpha_1 = \frac{1}{2}\log\left(\frac{1-\varepsilon_1}{\varepsilon_1}\right)$.

In round $m + 1$ $(m \geq 1)$, a new observation distribution is calculated with the misclassified cases in the last run assigned a new weight $D_n^{(m+1)} = D_n^{(m)} exp(\alpha_m)$ and the correctly predicted case weights are unchanged. The new bootstrapped data set B_{m+1} is sampled with replacement from the original training data following the weighted observation distribution. Then a new fully-grown random forest tree t_{m+1} is fit to the bootstrapped data set. The prediction is denoted

by $h_1(x) \in \{-1, 1\}$. After that, the prediction error ε_{m+1} and the tree weight $\alpha_{m+1} = \frac{1}{2}\log\left(\frac{1-\varepsilon_{m+1}}{\varepsilon_{m+1}}\right)$ were calculated.

This procedure is repeated *T* times and the final ensemble prediction is the weighted average of the predictions of single classifiers. The prediction of the ensemble for a given vector $\mathbf{x}$ is

$$H(\mathbf{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right) = \begin{cases} +1, \text{ if } \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) > 0 \\ -1, \text{ if } \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) < 0 \end{cases}$$

In the above formula, the sign function converts the linear combination of the individual tree predictions to +1 or –1. If the weighted tree prediction is larger than zero, then the ensemble prediction is equal to +1; if the weighted tree prediction is less than zero, then the ensemble prediction is equal to –1.

4.4.3 Evaluation of the proposed strategy

Step 1: Start with equally-weighted data $\left(d_i = \frac{1}{n}\right)$ as in AdaBoost.

Step 2: Bootstrap the data and fit a fully-grown tree to the bootstrapped data as in random forests.

Step 3: Use the tree to predict all of the data (perfect on inbag data, mixed on outbag data).

Step 4: Use the AdaBoost formulae to compute new data weights and the tree weight α_t.

Step 5: Resample the data following the distribution of the data weights and fit a fully-grown tree to the bootstrapped data.

Step 6: Repeat Steps 3-5 *T-1* times.

Step 7: Form the weighted ensemble prediction as $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.

Step 8: Repeat Steps (1-7) 100 times and average the results.

4.5 Results

We applied the model to different data sets, including real data sets and created ones. We split the data into two parts, training and test. We fit the model to the training data and then predict the test data to calculate the test error. The original AdaBoost (oriAdaBoost) fits stumps or pruned trees on all training data. However, the AdaBoost R package is different. It only uses part of the training data set (default 50%). We calculate the corresponding test error rates of random forests, AdaBoost, and oriAdaBoost, and repeat the process 100 times. After that we compare the average test error rates among the four models.

The first data set is the Glaucoma data, a real data set with 62 predictor variables and 192 observations. Table 1 shows the average test error rates of the three models. The first column is the number of trees, ranging from 50 to 500, and the remaining columns show the average test error rates of random forests, AdaBoost, oriAdaBoost, and our proposed fullTreeBoost model. The average test error rate along with its standard error is given in each cell. We use a paired *t* test to compare fullTreeBoost model with other methods. We use * to denote that the average test error rate of fullTreeBoost is lower than the corresponding method at the 5% level and use ** to denote that the average test error rate of fullTreeBoost is lower than the corresponding method at the 1% level.

Ntree	Random Forest	AdaBoost	oriAdaBoost	FullTreeBoost Model
50	.1565(.00453)	.1592(.00469)	.1747**(.00535)	.1559(.00418)
100	.1582*(.00449)	.1629**(.00452)	.1767**(.00462)	.1508(.00420)
200	.1427(.00504)	.1459**(.00487)	.1692**(.00515)	.1380(.00489)
300	.1547**(.00476)	.1571**(.00496)	.1686**(.00518)	.1471(.00469)
400	.1502(.00471)	.1516(.00497)	.1680**(.00520)	.1445(.00473)
500	.1551**(.00447)	.1543**(.00486)	.1700**(.00478)	.1453(.00486)

Table 1. Average Test Error Rates of Random Forests, AdaBoost, oriAdaBoost, and FullTreeBoost for Glaucoma Data

From the results, we can see that the new model has significantly lower test error rates than other methods in many cases.

The second data set is the Sonar data, which has 60 predictor variables and 208 observations.

Ntree	Random Forest	AdaBoost	oriAdaBoost	fullTreeBoost model
50	.1890**(.00538)	.2006**(.00625)	.1900**(.00478)	.1733(.00539)
100	.1775**(.00551)	.1710**(.00518)	.1756**(.00548)	.1540(.00523)
200	.1825**(.00524)	.1573(.00473)	.1625**(.00526)	.1500(.00483)
300	.1700**(.00502)	.1467(.00511)	.1615**(.00511)	.1444(.00467)
400	.1748**(.00531)	.1473(.00499)	.1579**(.00503)	.1448(.00522)
500	.1856**(.00646)	.1496(.00541)	.1681**(.00570)	.1548(.00540)

Table 2. Average Test Error Rates of Random Forests, AdaBoost, oriAdaBoost, and FullTreeBoost for Sonar Data

From the results, we can see that when the ensemble size is small, below 100 trees, the performance of the fullTreeBoost model is significantly better than the other three models. When the ensemble size grows to 200 to 300 trees, the performance of the fullTreeBoost is significantly better than random forests and oriAdaBoost and slightly better than AdaBoost. When the ensemble size continues to increase, the performance of fullTreeBoost is the same as AdaBoost but still significantly better than the other two methods.

The third data set is the liver data, with 6 predictor variables and 345 observations. We use # / ## to denote that the average test error rates of the fullTreeBoost is higher than random forests at the 5% / 1% level and use ^^ to denote the average test error rates of fullTreeBoost is higher than AdaBoost at the 1% level. Table 3 compares the results of the three models on different ensemble sizes.

Ntree	Random Forest	AdaBoost	oriAdaBoost	FullTreeBoost
50	.2797(.00420)	.2724(.00420)	.2841(.00394)	.2887#^^(.00426)
100	.2802(.00458)	.2866(.00444)	.2971(.00414)	.2908#(.00472)
200	.2751(.00441)	.2753(.00384)	.3012**(.00434)	.2872##^^(.00436)
300	.2742(.00511)	.2791(.00492)	.3067**(.00432)	.2827##(.00499)
400	.2784(.00499)	.2866(.00443)	.3108**(.00446)	.2856##(.00463)
500	.2779(.00446)	.2947(.00472)	.3151**(.00474)	.2929##(.00464)

Table 3. Average Test Error Rates of Random Forests, AdaBoost, oriAdaBoost, and FullTreeBoost for Liver Data

The test error rates of the fullTreeBoost is significantly higher than random forests but significantly lower than oriAdaBoost and close to AdaBoost.

The fourth data set is the Ionosphere data, with 34 predictor variables and 351 observations. Table 4 contains the results of the three models for different ensemble sizes.

Ntree	Random Forest	AdaBoost	oriAdaBoost	fullTreeBoost model
50	.07057**(.00252)	.06818*(.00268)	.07330**(.00293)	.06420(.00249)
100	.0692**(.00273)	.06818**(.00252)	.07239**(.00270)	.06205(.00259)
200	.06761**(.00213)	.06545**(.00240)	.06750**(.00220)	.05852(.00211)
300	.06773**(.00230)	.07273**(.00253)	.07239**(.00254)	.06080(.00233)
400	.06920**(.00273)	.07068**(.00249)	.07182**(.00267)	.06159(.00235)
500	.05977**(.00205)	.06432**(.00242)	.06386**(.00232)	.05307(.00211)

Table 4. Average Test Error Rates of Random Forests, AdaBoost, oriAdaBoost, and FullTreeBoost for Ionosphere Data

From the results, we can see that the fullTreeBoost model has the best overall performance compared to all other models.

The third data set is the Circle data, a simulated data set with 2 predictor variables and 600 observations. There are three concentric rings of uniform data constrained to the unit square. The innermost *y* values are equal to +1, the middle *y* values = –1, and the outermost *y* values = +1. Table 5 shows the results for the Circle data.

Ntree	Random Forest	AdaBoost	oriAdaBoost	fullTreeBoost model
50	.07560(.00225)	.08753**(.00259)	.08100**(.00238)	.07287(.00208)
100	.08200**(.00234)	.0791(.00239)	.0799(.00225)	.07627(.00212)
200	.08093**(.00206)	.07353(.00230)	.07780**(.00217)	.07327(.00211)
300	.08293*(.00245)	.07627(.00238)	.08647**(.00246)	.07907(.00229)
400	.08053**(.00218)	.07567(.00218)	.08213**(.00206)	.07493(.00213)
500	.08000**(.00204)	.07133(.00191)	.08460**(.00212)	.07420(.00194)

Table 5. Average Test Error Rates of Random Forests, AdaBoost, oriAdaBoost, and FullTreeBoost for Circle Data

From the results, we can see that the fullTreeBoost model has a better performance than random forests and oriAdaBoost. When the ensemble size is small, like 50 to 100 trees, the fullTreeBoost model has a better performance than AdaBoost. However, when the ensemble size increases, their performances are close.

The next data set is the Breast Cancer data, a real data set with 9 predictor variables and 683 observations. There is a problem when fitting the fullTreeBoost model to this data set. Observation 217 appears to be a serious outlier. No tree can predict it correctly. If it is in the data set, the model will focus on it and its weight is overly exaggerated to the point that the bootstrapped sample is composed mainly of this observation. To avoid this problem, we removed this observation from the original data set. Table 6 shows the results.

Ntree	Random Forest	AdaBoost	oriAdaBoost	fullTreeBoost model
50	.03171(.00110)	.03529**(.00121)	.03835**(.00120)	.03106(.00113)
100	.02806*(.00106)	.02912**(.00119)	.03276**(.00142)	.02624(.00115)
200	.02724(.00115)	.02735(.00123)	.03112**(.00124)	.02647(.00118)
300	.02871*(.00117)	.02888*(.00102)	.03024**(.00114)	.02706(.00122)
400	.02906**(.00115)	.02876*(.00123)	.03047**(.00131)	.02718(.00121)
500	.02912(.00114)	.02894(.00118)	.03071(.00117)	.02976(.00142)

Table 6. Average Test Error Rates of Random Forests, AdaBoost, and FullTreeBoost for Breast Cancer Data

From the results, we can see that the fullTreeBoost model has better performance than the other methods.

The last data set is RingNorm1500, with 20 variables and 1500 observations. RingNorm1500 is a simulated data set. In the random forest R package, mtry is used to denote the size of the subset of random select of all predictors at each split. The default setting is mtry = floor(sqrt(n predictors)). For this data set, the default mtry=4. However, we found that when we changed mtry from 4 to 1, the performance of random forests improved significantly. Since our model includes features of random forests, we also fit a fullTreeBoost with mtry equal to 1 and compared its test error rate to other models. We use * / ** to denote that the test error rate of fullTreeBoost with mtry equal to 4 is lower than the corresponding model at 5% / 1% level and use ++ to denote that the test error rate of fullTreeBoost with mtry equal to 1 is lower than the corresponding model at 1% level. Table 7 shows the results.

Ntree	RF mtry=4	RF mtry=1	AdaBoost	oriAdaBoost	FTB mtry=4	FTB mtry=1
50	.05696**++(.00117)	.04117**++(.00106)	.04128**++(.00089)	.04123**++(.00094)	.03571(.00075)	.03467(.00086)
100	.05395**++(.00114)	.03557**++(.00094)	.0336++(.00086)	.03797**++(.00092)	.03211++(.00085)	.02976(.00080)
200	.05131**++(.00013)	.03491**++(.00096)	.02941++(.00071)	.03549**++(.00078)	.02872++(.00083)	.02592(.00077)
300	.04883**++(.0013)	.03136**++(.00091)	.02907*++(.00087)	.03544**++(.00093)	.02744++(.00086)	.02440(.00072)
400	.04995**++(.00112)	.03341**++(.00095)	.02955++(.00084)	.03669**++(.00096)	.02979++(.00094)	.02541(.00083)
500	.04893**++(.00116)	.0304**++(.00077)	.02789++(.00078)	.03349**++(.00079)	.02691++(.00076)	.02325(.00066)

Table 7. Average Test Error Rates of Random Forests, AdaBoost, oriAdaBoost, and FullTreeBoost for RingNorm1500 Data

From the results, we can see when mtry = 4, the performance of fullTreeBoost is significantly better than random forests and oriAdaBoost and slightly better than AdaBoost. When mtry =1, fullTreeBoost outperforms all other methods.

4.6 Conclusion

In this article, we create a new ensemble model by adding random forest features to AdaBoost. The new model outperforms random forests and is better than AdaBoost for some data sets. In the future, we will apply this method to other data sets and explore other methods to blend random forests and boosting models.

REFERENCES

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26, 123-140.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, 11, 1493-1517.

Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5-32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and Regression Trees. Belmont, CA: Wadsworth.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Machine Learning*, 40, 1-22.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. *Machine Learning, Proceedings of the 13th International Conference*, 148-156.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 119-139.

Grove, AJ. and Schuurmans, D. (1998). Boosting in the limit: maximizing the margin of learned ensembles. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 692-699.

Schapire, R., Freund, Y., Bartlett, P., and Lee, W. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26, 1651-1686.

CHAPTER 5

CONCLUSIONS

5.1 Summary

Shen and Li (2010) suggested that increasing the mean and decreasing the variance of the margins might improve the generalization performance of ensembles, which is referred as "squeezing". Random forests use equal weights to combine single trees. We applied quadratic programming to minimize the variance and maximize the mean of the margins by changing the tree weights. Most of the tree weights are zeroed out and the size of the new ensemble model is much smaller. However, this method does not appear to decrease the generalization error. In most cases, the predictive performance is maintained.

To improve the predictive performance, we suggest squeezing the margins further by reweighting the observations. We first identify the poorly predicted and well predicted observations. For poorly predicted observations, we allow the falsely predicted OOB observations to appear more times in growing the trees. For the well predicted observations, we reduce their number of appearances in the inbag data of the trees. Although we do not change the weights of the observations, we change the distribution of the data to give more attention to poorly predicted observations. The thinking behind this method is similar to boosting. Therefore, we believe that boosting is also a squeezing method. We also notice that random forests squeeze the margins more than bagging, which means that random forests have some similarity with boosting.

The second method inspired us to blend random forests and boosting from a different perspective. We remodel AdaBoost by applying random forest features. Specifically we use fully-grown random forests trees to replace stumps or pruned trees, and we use bootstrap

samples instead of the full training data. The predictive performance of the new model compares favorably to random forests and AdaBoost.

The objective of the research is to increase the interpretability or predictive performance. From the results of the three proposed methods, we either reduce the size or the generalization error of ensemble models.

5.2 Future Research

In this research, we used quadratic programming to reweight an ensemble model. We will try other methods, like MCMC or cluster analysis. Although we reduce the size of an ensemble model, it is still difficult to interpret. Our second plan is to merge all the single trees of an ensemble model to build a new tree, which we refer to as an ensemble tree. We hope the ensemble tree maintains the same predictive performance as the original ensemble model, but has greater interpretability.

REFERENCES

Banfield, R.E., Hall, L. O., Bowyer, K.W., and Kegelmeyer, W.P. (2005). Ensemble diversity measures and their application to thinning. *Information Fusion*, 6(1), 49-62.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26(2), 123-140.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, 11, 1493-1517.

Breiman, L. (2001a). Random forests. *Machine Learning*, 45, 5-32.

Breiman, L. (2001b). Statistical Modeling: The Two Cultures. *Statistical Science*, 16, 199-231.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and Regression Trees. Belmont, CA: Wadsworth.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Machine Learning*, 40, 1-22.

Domingos P. (1998). Knowledge discovery via multiple models. *Intelligent Data Analysis*, 3, 187-202.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. *Machine Learning, Proceedings of the 13th International Conference*, 148-156.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 119-139.

Grove, A.J. and Schuurmans, D. (1998). Boosting in the limit: maximizing the margin of learned ensembles. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 692-699.

Ko, A.H., Sabourin, R., and Britto, A.D.S. (2009). Compound diversity functions for ensemble selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 23, 659-686.

Latinne, P., Debeir O., and Decaestecker C. (2001). Limiting the Number of Trees in Random Forests. *Lecture notes in computer science,* 2096/2001, 178-187.

Margineantu, D.D. and Dietterich, T.G. (1997). Pruning adaptive boosting. *Proceedings of the 14th International Conference on Machine Learning*, 211-218.

Mason, L., Bartlett, P., and Baxter, J. (2000). Improved generalization through explicit optimization of margins. *Machine Learning*, 38, 243-255.

Reyzin, L. and Schapire, R. (2006). How boosting the margin can also boost classifier complexity. *Proceedings of the 23rd International Conference on Machine Learning*, 753-760.

Schapire, R., Freund, Y., Bartlett, P., and Lee, W. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26, 1651-1686.

Shen, C. and Li, H. (2010). Boosting through optimization of margin distributions. *IEEE Transactions on Neural Networks*, 21(4), 659-666.