GA-BOOST:

A GENETIC ALGORITHM FOR ROBUST BOOSTING

by

DONG-YOP OH

J. BRIAN GRAY, COMMITTEE CHAIR

MICHAEL CONERLY
BRUCE E. BARRETT
SAMUEL ADDY
JUNSOO LEE

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Applied Statistics
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2012

ABSTRACT

Many simple and complex methods have been developed to solve the classification problem. Boosting is one of the best known techniques for improving the prediction accuracy of classification methods, but boosting is sometimes prone to overfit and the final model is difficult to interpret. Some boosting methods, including Adaboost, are very sensitive to outliers. Many researchers have contributed to resolving boosting problems, but those problems are still remaining as hot issues.

We introduce a new boosting algorithm “GA-Boost” which directly optimizes weak learners and their associated weights using a genetic algorithm, and three extended versions of GA-Boost. The genetic algorithm utilizes a new penalized fitness function that consists of three parameters (*a*, *b*, and *p*) which limit the number of weak classifiers (by *b*) and control the effects of outliers (by *a*) to maximize an appropriately chosen *p*-th percentile of margins. We evaluate GA-Boost performance with an experimental design and compare it to AdaBoost using several artificial and real-world data sets from the UC-Irvine Machine Learning Repository.

In experiments, GA-Boost was more resistant to outliers and resulted in simpler predictive models than AdaBoost. GA-Boost can be applied to data sets with three different weak classifier options. We introduce three extended versions of GA-Boost, which performed very well on two simulation data sets and three real world data sets.

DEDICATION

This dissertation is dedicated to those who wholeheartedly supported and guided me, especially to my dear mother, Doja and my wife, Hyejin whom I love.

LIST OF ABBREVIATIONS AND SYMBOLS

a, b	Tuning parameter for the penalty terms
$D_t(i)$	Weights over the training data i on round t
h_t	Weak learner on round t
$h_t(\underline{x})$	Prediction of the t-th weak classifier h_t
$I(\bullet)$	Indicator function
K	Number of simulations
M_p	pth percentile of margins distribution
m_i	ith margin
N	Number of population solutions
n	Number of observations
p	Number of predictor variables
p	Percentile of margin
s	Solution in GA
T	Number of boosting rounds

T_s	Number of weak classifiers in the solution *s*
X	Predictor variable matrix (*n* x p) with elements x_{ij}
$\underline{x}$	Vector of predictor variable values
Y	Response (dependent) variable vector (*n* x 1) with elements y_i
$\{\}$	Set
α_t	Weight associated with the *t*-th weak classifier h_t
ρ	Maximum margin
ε_t	Train error on round *t*
$\in$	Element of a set
$\sum$	Summation
θ	Any number greater than zero
$<$	Less than
$>$	Greater than
$\leq$	Less than or equal to
$\geq$	Greater than or equal to
$=$	Equal to
$\vert$	Given, or conditioned upon

ACKNOWLEDGMENTS

I would like to express my gratitude to my Father in heaven. Without leaning on Him I would not have finished this dissertation. I would like to acknowledge all of my committee members for wonderful advice and encouragement. I sincerely acknowledge my committee chairman, Dr. J. Brian Gray, for his kindness, and guidance throughout this entire research process. I cannot express how much I appreciate him. I wish to acknowledge Dr. Samuel N. Addy in CBER for financial support during my academic years and for his endless sense of humor that motivated me. I am also grateful to Dr. Junsoo Lee, Dr. Michael Conerly, and Dr. Bruce Barrett for their enthusiastic teaching and contribution to the completion of this research.

It is also my pleasure to acknowledge the KPCT Young Adults Group and Rev. Barry McGlothin as my spiritual mentor, whose fellowship and support enriched my life during the academic journey.

Finally, I wish to thank my family for their love and prayers. First and foremost, my deepest appreciation goes to my wife, Hyejin Lee for her tremendous support, and encouragement on every step of this project, and my wonderful daughter, Hannah.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Many simple and complex methods have been developed to solve classification problems in a variety of application areas. Classification methods attempt to build models that can be used to optimally assign observations to classes or categories based on the values of one or more predictor variables, which can be quantitative or categorical.

Logistic regression and discriminant analysis are well known traditional methods for classification problems based on statistical models. They make many assumptions, such as multivariate normality, while newer methods, such as decision trees (Breiman, Friedman, Olshen, and Stone, 1984), have less restrictive assumptions. Decision trees are one of the most popular techniques for classification problems in data mining and machine learning because tree models are useful in assessing which predictor variables are important, are easily interpreted, and can model complex interaction relationships in data. But the predictive performance of tree models is not as good as that of ensemble models. Ensemble models are constructed as combinations of decision trees or weak classifiers. A variety of ensemble techniques has developed including random forests (Breiman, 2001), bagging or bootstrap aggregation (Breiman, 1996), and boosting (Schapire, 1990). Boosting is an important topic in predictive modeling and machine learning (Zhu 2008, Friedman 2003).

1.1 Boosting

Boosting (Schapire, 1990) came from the PAC (Probably Approximately Correct) (Leslie and Valiant, 1984) learning model. This is a widely used and powerful prediction technique that sequentially constructs an ensemble of weak classifiers. A weak classifier is a very simple model that has just slightly better accuracy than a random classifier, which has 50% accuracy on the training data set. The set of weak classifiers is built iteratively from the training data over hundreds or thousands of iterations. At each iteration or round, the examples in the training data are reweighted according to how well they are classified with larger weights given to misclassified examples. Weights are also computed for the weak classifiers based on their classification accuracy. The weighted predictions from the weak classifiers are combined using voting (for classification) or averaging (for regression) to compute a final prediction of the outcome Y (in $\{-1, 1\}$) of the form

$$\widehat{Y}(\underline{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\underline{x})\right), \tag{1.1}$$

where $\underline{x}$ is a vector of predictor variable values, $h_t(\underline{x})$ is the prediction of the t-th weak classifier h_t for $\underline{x}$, α_t is the weight associated with the t-th weak classifier h_t, and the sum is taken over all T weak classifiers. Boosting has excellent predictive performance, however, the final model is difficult to interpret.

Since the initial boosting method was proposed, various alternative boosting methods have been developed. AdaBoost or Adaptive Boosting (Freund and Schapire, 1995) is the most common boosting algorithm for binary classification. Freund and Schapire (1999) concluded that the AdaBoost algorithm is less vulnerable to overfitting compared to most learning algorithms because early studies showed that AdaBoost rarely overfits the training data in practice. An early mathematical result also suggested that the test error is not affected by the number of weak classifiers in a final model (Schapire, Freund, Bartlett, and Lee, 1998). However, Grove and Schuurmans (1998) found that AdaBoost sometimes does overfit. Dietterich (2000) also showed that if there are outliers, boosting does not work as well as other ensemble methods.

To improve the performance of boosting, properties of the cumulative margin distribution (CMD) have been studied. Schapire et al. (1998) define the margin of an observation as "the difference between the weight assigned to the correct label and the maximal weight assigned to any single incorrect label". The margin of an observation has a range in [–1, +1], which is positive if and only if the final model correctly classifies the observation. The magnitude of the margin can be interpreted as a measure of confidence in the prediction. Thus, we want to have margins that are large and positive. The graph of the cumulative margin distribution can provide useful information for assessing the model fit. AdaBoost generates a margin that is at least $\frac{1}{2}\rho$, where ρ is the maximum margin (Shapire et. al, 1998). Rätsch and Warmuth (2002) introduced a new version of AdaBoost denoted by AdaBoost$_{\rho}$, which can maximize the minimum margin of the examples to a value $M(\rho) \geq \frac{1}{2}\rho$. Grove and Schuurmans (1998) proposed a new method called the Linear Programming Boosting (LPBoost) algorithm that has as its criterion to maximize the minimum margin. The Direct Optimization Of Margins (DOOM) algorithm proposed by Mason, Bartlett, and Baxter (1998) also works to directly optimize a function of the cumulative margin distribution. Their experiments also show that the minimum margin from training data does not have much impact on generalization performance.

As another way to improve the performance of boosting in the presence of outliers, Friedman, Hastie, and Tibshirani (2000) proposed GentleBoost as an alternative to AdaBoost. GentleBoost gives less emphasis to misclassified examples, that is, it gives less emphasis to outliers by using a quadratic objective function of the margins, which increases more slowly than the exponential function of margins used by AdaBoost. Freund (2001) proposed a new boosting algorithm called BrownBoost, which is an adaptive version of the boost by majority (Freund, 1995) algorithm. The main assumption is that noisy examples will be repeatedly misclassified. Therefore to attain good results, we should eliminate examples that have large negative margins from the training set, and concentrate on examples with small negative margins. Servedio (2003) described a smooth boosting algorithm using smooth distributions of examples that don't over-weight any single example.

1.2 The Application of this Research

In our early experiments with AdaBoost, we made two interesting observations. First, in the sequence of hundreds or thousands of weak classifiers constructed by AdaBoost, there is typically a small set of weak classifiers that AdaBoost cycles through in a highly repetitive way. Secondly, we found that AdaBoost devotes too much attention to outliers in the data in trying to attain a perfect fit to the training data. This results in smaller positive margins for the vast majority of the data and the potential for overfitting. Based on these interesting findings, we propose a new boosting algorithm called GA-Boost. GA-Boost solves directly for the weak classifiers and the weights of those weak classifiers using a genetic algorithm. The genetic algorithm utilizes a new penalized fitness function that limits the number of weak classifiers. Thus, the final model has an interpretability advantage over other boosting algorithms due to the reduced model complexity. GA-Boost can control the effects of outliers by the choice of an appropriate fitness function. The genetic algorithm for GA-Boost is composed of three parts: (1) initialization and randomization, (2) genetic evolution, and (3) fitness function.

We report the effect of tuning parameters (*a*, *b*, and *p*) of a new penalized fitness function that limits the number of weak classifiers and controls the effects of outliers by the number of negative margins and maximizing an appropriately chosen *p*-th percentile of margins. We compare GA-Boost to AdaBoost on several artificial data sets and three real world data sets from UC-Irvine Machine Learning Repository. Results from these examples suggest that GA-Boost is more resistant to the effects of outliers and provides solutions that are simpler than those found by AdaBoost.

Finally, we introduce three different GA-Boost variants. Because AdaBoost does not maximize the margin, we would like to improve the solution by taking the best weak learners and their weights. GA-Boost utilizes the initial population of solutions randomly generated but we can consider drawing parameters from the final solution of AdaBoost, and then applies a genetic algorithm to optimize the parameters to produce the best prediction. From the results based on two simulation data sets and three real world data sets, the variants of GA-Boost performed very well as predictive methods. GA-Boost and GAdaBoost I performed better with a simpler solution and more resistant to outliers than AdaBoost, the other variants for a large proportion of outliers in a simple data set. Most extended versions have better performance than AdaBoost for testing the complex data set and the real data sets.

Chapter 2

Literature Review

2.1 The Classification Problem

One hot topic in statistics, data mining, and machine learning is classification. Classification techniques have been developed and applied in a variety of areas including filtering spam mail in computer science, credit scoring in finance, and drug discovery in medicine. Typically statistical classification tries to build a well-defined model that can be used to optimally assign observations into a response categorical variable *Y* with *K* levels, Y_k, *k*=1,2,…, *K*. In the case of two labeled classes, *k*=2, we call *Y* a binary response variable. $X = \{X_1, ..., X_p\}$ are called the "input" or "predictor" variables which are quantitative or categorical, and *n* is the sample size. The goal of classification is to build a predictive model for *Y* using the predictor *X* variables.

2.2 Classification Methods

There are many different types of methods for classification problems. We have traditional methods based on statistical models, such as logistic regression and discriminant analysis. There are also newer, more computationally intensive techniques that have been developed in the data mining and machine learning communities.

2.2.1 Discriminant Analysis and Logistic Regression

Discriminant analysis is a statistical technique that separates measurements of two or more classes of objects with a linear or quadratic surface. Linear and quadratic discriminant analyses are determined by whether the population covariance matrices are the same or not. The features of this method can be graphically and algebraically described well. Logistic regression, in contrast with linear regression, has a discrete target variable following a binomial or multinomial distribution. Because the predictions are probabilities ($\pi_i = P(Y_i = 1)$), the probabilities must be transformed to logits, or natural log of odds, in order to create a suitable model,

$$\text{logit}(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} \ . \tag{2.1}$$

The logit function is a linear combination of the predictor variables. These traditional methods are excellent under some strict assumptions. (One assumption of logistic regression is that the logit transformation of the probabilities of the target variable is linearly related to the input variables.)

2.2.2 Data Mining and Machine Learning

More recently, the areas of data mining and machine learning have developed very useful tools for solving the classification problem. These recent methods don’t have as many assumptions and are able to handle large data sets easily. In machine learning with respect to classification, the naive Bayes classifier and support vector machine have been employed. Support vector machines (Vapnik 1995) are the most vigorously studied and have been applied in many areas recently. Popular classification techniques in data mining include decision trees and neural networks. The performance of these techniques strongly depends on the data characteristics so that there is no best method for all problems. Statistics, machine learning, and data mining are very closely related to each other and each one borrows useful ideas from the others to improve performance. Boosted decision trees, random forests, and bagging are typical examples.

2.2.3 Decision Trees

Decision trees is one of the more popular techniques for classification problem in data mining as well as machine learning (Brieman, Friedman, Olshen, and Stone, 1984). The original data, known as the root node, are divided into two or more sub-samples as nodes according to a splitting criterion in order to maximize a homogeneity or purity measure of the target variable. This splitting process is repeated for each node until a final partition is reached. The final partition is determined by some stopping criterion. Thus to build decision trees we solve two problems. One is choosing the splitting attribute (split criterion) and the other is deciding on a stopping criterion. There are three major algorithms for constructing a decision tree. The most well-known algorithm in statistics is the CART algorithm (Breiman et al., 1984), which stands for Classification and Regression Trees, based on the Gini index for its splitting criterion. The CHAID algorithm (Kass, 1980) is used for multiway splits based on chi-squared test and F-test for discrete data and continuous data, respectively. The other algorithms are C4.5 and C5.0 (Quinlan, 1993), which are broadly used in computer science and are based on an entropy index.

Murthy, Kasif, and Salzberg (1994) used linear combinations of attributes for the split rule at each node:

$$\sum_{i=1}^{d} a_i x_i + a_{d+1} > 0 \ , \qquad (2.2)$$

where x_i's are real-valued attributes and $a_i, \cdots, a_{d+1}$ are real-valued coefficients. They called decision trees of this form oblique decision trees.

Decision trees offer many advantages over classical methods. First of all, a decision tree is easily interpreted and is useful for assessing which predictor variables are important. Indeed, it can reveal complex interactions among variables. Another advantage is that the decision tree is a useful method when the distribution of the data is unknown, because it doesn’t make strict assumptions about the distribution of the underlying data. In addition, it is computationally fast and easy to construct. Thus, decision trees have few assumptions and are more useful in identifying complex interactions among predictors compared to classical methods.

However, decision trees are not perfect and have some weaknesses. One is instability. Small changes in the training data can sometimes have large effects on the results of the tree. Some tree models generated may be very large so that they are difficult to interpret and understand. Also, the performance of tree models is not very good compared to more recent methods. These disadvantages cause many predictive modelers not to prefer trees as their final model; instead they desire to use trees as auxiliary tools in creating stronger models. In order to overcome the drawbacks of decision trees, especially instability and inaccuracy, many have been led to consider ensemble methods.

2.2.4 Ensemble Methods

The purpose of ensemble methods is to improve predictive model performance. A variety of ensemble techniques have been developed, including bagging (bootstrap aggregation) (Breiman 1996), random forests (Breiman 2001), and boosting (Schapire 1990). Because ensemble models are constructed by a combination of multiple models instead of using a single model, increased computation problems and more complex interpretation of the model result. However ensemble methods have great advantages. They can improve the model accuracy and overcome some of the problems of decision trees like model instability. Next, we discuss in more detail ensemble learning techniques: bagging, random forests, and boosting.

2.2.4.1 Bagging

Bagging (bootstrap aggregation), introduced by Breiman (1996) and well-known as a variance reduction technique, is an ensemble method for improving unstable estimation. We first draw the random samples using bootstrapping with replacement from training data set and build a tree on each bootstrap sample. We then take the majority vote of the trees for classification or take the average of fitted values for regression trees to create a single prediction.

2.2.4.2 Random Forests

Random forests (Breiman, 2001) are an improvement on bagging. A random forest is constructed by the following steps: First, T and m are chosen as the number of trees to grow and the number of variables used to split each node, respectively. m should be much less than M, which is the number of input variables. To grow each tree, a bootstrap sample of size n is sampled from the original data with replacement and trees are grown from these bootstrap samples. When growing a tree, at each node m variables are selected at random from the M predictor variables and used to find the best split. Each tree is fully grown with no pruning. Finally, we take votes from every tree in the forest and then use majority voting to decide on the class label.

2.2.4.3 Boosting

The idea for boosting (Schapire, 1990) came from the PAC (Probably Approximately Correct) (Leslie and Valiant 1984) learning model. Boosting has been the most widely used and one of the most powerful learning models for predictive modeling. First we produce a “weak classifier,” that is, a model that has slightly better accuracy than random guessing, which has 50% accuracy on the training data set. A succession of models is built iteratively. At this point the examples are being trained and reweighted. Finally, each model or a weak classifier is weighted according to its performance and combined with other weak classifiers using voting (for classification) or averaging (for regression) to create a final model.

These ensemble techniques can reduce model instability and have high predictive performance. But they bring about computation and interpretation problems because the final model is a combination of multiple models. Nevertheless, from these three ensemble methods, a variety of methods have been developed and extended to many areas.

2.3 AdaBoost

As we mentioned briefly in the previous section, boosting is a common method for improving the accuracy of any given learning algorithm. In this section we describe AdaBoost (Adaptive Boosting) (Freund & Schapire 1997), which is the most typical boosting algorithm for binary classification problem and the motivation for this research.

2.3.1 AdaBoost Algorithm

The AdaBoost algorithm, proposed by Freund & Schapire (1997), is probably the best known boosting algorithm for binary classification problems. The training data set is given by $(\underline{\mathrm{x}}_1, \mathrm{y}_1), (\underline{\mathrm{x}}_2, \mathrm{y}_2), \ldots, (\underline{\mathrm{x}}_n, \mathrm{y}_n)$, where we assume $y = \{-1, +1\}$ for the binary case. AdaBoost outputs a linear combination of weak learners. The "weak learner" was first proposed by Kearns and Valiant (1988). A weak learner is defined to be a classifier having just a little better training error than 50%. In papers on boosting, the terms "hypothesis," "learner," and "classifier" are equivalent.

The final AdaBoost model is given by

$$H(\underline{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\underline{x})\right), \qquad (2.3)$$

where

$\underline{x}$: A vector of predictor variable values
α_t : Weight attached to weak classifier(t)
$h_t(\underline{x})$: Weak classifier is defined to be a classifier having just a little better training error than 50% error over any distribution, $\{-1, +1\}$
$H(\underline{x})$: Strong (final) classifier is a weighted majority vote of the T weak classifiers where α_t is the weight assigned to h_t, outcome in $\{-1, +1\}$

Weak learners are sequentially created in a series of rounds, t = 1, 2… T. In each round, observations are reweighted by the previous weak learner. The set of weights over the training data i on round t is denoted $D_t(i)$, which is called the distribution of weight (Freund & Schapire 1999). All weights initially have the same value. After each round, the weight of each incorrectly classified example is increased, while the weight of each correctly classified example is decreased. Thus the new classifier focuses more on incorrectly classified examples.

```
Given: $(\underline{x}_1, y_1), \ldots., (\underline{x}_n, y_n)$ where $\underline{x}_i \in X$, $y_i \in Y = \{-1,+1\}$
Initialize $D_1(i) = 1/n$.
For $t = 1, \ldots., T$:
    Train weak classifier using distribution $D_t$.
    Get weak classifier $h_t : X \to \{-1,+1\}$ with error
        $\varepsilon_t = \Pr_{i \sim Dt}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$

    Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$.

    Update: $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
        where $Z_t$ is a normalization factor.
Output the final classifier: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$
```

Figure 1. AdaBoost Algorithm

On each round *t*, the training error is calculated by sum of the weight distribution. The formula for training error on each round *t* is

$$\varepsilon_t = \Pr_{i \sim Dt}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i) \neq y_i} D_t(i) \tag{2.4}$$

Therefore training error is the probability of misclassification or the sum of the weights of misclassified training examples. There are two weight parameters in AdaBoost. One is the weight distribution of example *i*, $D_t(i)$, which is recalculated on every round and another is the weight of the *t*-th weak hypothesis, α_t, measuring the importance that is assigned to the weak hypothesis. If the weak hypothesis can classify well, then we can give more weight to it; alternatively, if we have a poor hypothesis then we grant less weight. So the parameter α_t depends on training error. The formula for α_t in AdaBoost was originally given by Freund and Schapire (1997) as

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right). \tag{2.5}$$

Finally all hypotheses are combined, and a majority vote is taken for the final model.

2.3.2 Theoretical Property of AdaBoost

One of the properties of AdaBoost is that as we have more successive rounds, the training error decreases. If we continue to add more weak learners to a final model, then the training error eventually reaches zero. Theoretically, Freund and Shapire (1997) showed the training error exponentially decays as the number of rounds *T* increases. For the generalization error, we can calculate a probabilistic upper bound on the test error of a classification mode using the VC-dimension (Vapnik and Chervonenkis 1971), a typical measure of the "complexity" of the hypothesis space (Blumer, Ehrenfeucht, Haussler, and Warmuth 1987), and the number of boosting rounds *T*. Baum and Haussler (1989) showed that an upper bound on the generalization error of a classification model is given by

$$\Pr[H(x) \neq y] \leq \hat{\Pr}[H(x) \neq y] + O\left(\sqrt{\frac{Td}{m}}\right), \tag{2.6}$$

where $\hat{\Pr}[H(x) \neq y]$ is the training error, *d* is the VC-dimension of the weak learner, *m* is the sample size, and *T* is the number of boosting rounds. The bound suggests that boosting is prone to overfitting when we run boosting for too many rounds *T*. But in some experiments (e.g., Quinlan, 1996), empirical results suggest that boosting often does not overfit even with thousands of rounds.

To show that the bound is completely independent of *T*, the number of rounds, Schapire, Freund, Bartlett, and Lee (1998) used an alternative analysis in terms of the margins of the training data. Schapire et al. (1998) defined the margin for an observation as the difference between the weight assigned to the correct label and the maximal weight assigned to any single incorrect label. Margin has a range of [-1, +1] which is positive if and only if the final hypothesis correctly classifies the training set. We can interpret the magnitude of the margin as a measure of confidence in the prediction. Thus if we have the margin close to +1 (-1) then we have high confidence of a correct (incorrect) classification, and if we have the margin close to zero then we have a low confidence of classification.

Margin is defined as

$$\text{margin}(\underline{x}, y) = \frac{y\sum_t \alpha_t h_t(\underline{x})}{\sum_t \alpha_t}. \tag{2.7}$$

Schapire et al. (1998) showed that a superior upper bound on generalization error is given by

$$\Pr[H(x) \neq y] \leq \hat{\Pr}[\text{margin}(x, y) \leq \theta] + O\left(\sqrt{\frac{d}{m\theta^2}}\right). \tag{2.8}$$

Note that this upper bound is related to margin, sample size (*m*), and VC-dimension (*d*) and is obviously free from the number of rounds *T*. Thus to have more confidence and a better bound on generalization error, we require large margins for training examples.

2.3.3 Properties of Margin Distribution

One of the main issues in boosting is to study properties of the margin distribution in order to get better generalization performance. Schapire et al. (1998) discussed methods for improving the performance related to distribution of margins of the training data. Their experiments show that there is a good correlation between a reduction of the portion of training examples having small margin and generalization performance. Also they show that increasing the margins is very effective theoretically and experimentally.

AdaBoost generates a margin that is at least $\frac{1}{2}\rho$, where ρ is the maximum margin (Shapire et. al, 1998) and it is not the method to optimize alpha weight and weak learners because it doesn't maximize the margin (Rudin, Schapire and Daubechies, 2007). Rätsch and Warmuth (2002) introduce a new version of AdaBoost denoted by AdaBoost$_\rho$. This algorithm can maximize the minimum margin of the examples to a value $M(\rho) \geq \frac{1}{2}\rho$. Rudin, Schapire, and Daubechies (2004) showed that AdaBoost may converge to a margin which is significantly below the maximum. Grove and Schuurmans (1998) proposed the LPBoost algorithm (Linear Programming Boosting) which focuses on making the minimum margin as large as possible. They showed the average minimum margin values obtained by LPBoost are better than AdaBoost's which should mean more confident results. However, it is difficult to know whether this result gives better generalization performance than AdaBoost. In the same year, the DOOM (Direct Optimization of Margins) algorithm for optimizing margin cost functions using gradient descent was proposed by Mason, Bartlett, and Baxter (1998). Their experiments also show that the minimum margin from training data does not have a direct impact on generalization performance. These algorithms obviously have better minimum margins than AdaBoost but do not always yield better generalization performance. There is worse performance when there are outliers in data sets. Outliers usually have large negative margins and it is difficult to maximize the margins of these examples without affecting the margins of other examples. Therefore maximizing the minimum margin is not the best solution in terms of generalization performance. For that reason it is necessary to study other properties of the margin distribution for improving generalization performance.

2.3.4 Overfitting Problem and Noisy Data

The actual performance of boosting is dependent on the data set and the weak learner (Dietterich 2000). So there is still an overfitting problem because boosting is sensitive to noisy data and outliers. Thus mislabeled cases or outliers may cause the overfitting problem, for the new classifier focuses more on those observations that have a large negative margin value (classified incorrectly). Nevertheless, Schapire (1999) said AdaBoost is less vulnerable to the overfitting problem compared to most learning algorithms because early studies showed that AdaBoost rarely overfits the training data in practice. But Grove and Schuurmans (1998) show that AdaBoost sometimes does overfit if there are too many weak learners in the final model, which causes complexity problems. Dietterich (2000) compared three methods, bagging, boosting, and randomizing, based on predictive performance with and without classification noise. The results are that boosting is the best method as long as there are few outliers, but with added noise, bagging is best. Because the actual performance of boosting depends on the data, AdaBoost is not very successful in the presence of noisy data.

To improve the performance given noisy data, Friedman, Hastie, and Tibshirani (2000) suggested GentleBoost as an alternative to AdaBoost for the noisy data case. The reason is that it gives less emphasis to misclassified noisy data using a quadratic function of margin, which increases more slowly than exponential. Freund (2001) proposed a new boosting algorithm called BrownBoost, which is an adaptive version of the boost by majority (Freund 1995) algorithm. The main assumption is that noisy examples will be repeatedly misclassified. Therefore to attain good results, remove examples that have large negative margins from the training set, and concentrate on examples whose margins are small negative values. Servedio (2003) described a smooth boosting algorithm using smooth distributions of examples that don't overweight any single example. This algorithm is very similar to AdaBoost in maintaining the distribution of example at each round except in its treatment of noisy data. Usually the examples with high noise have very large negative margins, but smooth boosting can deal with this situation using limitations on the weight given to these examples.

2.4 Multiclass Classification

There are several boosting algorithms for multiclass problems in which *Y* has more than two possible case labels, $y \in \{1,2....,k\}$. Freund and Schapire (1997) extend AdaBoost for binary classification to the multiclass case with a method called AdaBoost.M1. Other more complicated methods have also been proposed. Schapire and Singer (1999) introduced a version of algorithm called AdaBoost.MH and Freund and Schapire (1997)'s algorithm AdaBoost.M2 which is a special case of AdaBoost.MR algorithm (Schapire and Singer 1999). AdaBoost.MR finds a weak learner that ranks the labels in *Y* with the hope that the correct labels will receive the highest ranks. In these methods, a set of more than two classes can be reduced to a binary set.

2.5 Genetic Algorithms

Holland (1975) developed a new class of optimization methods based on biological evolution principles for solving problems called Genetic Algorithms (GA). The origin of genetic algorithms began with biologists and computer scientists in the middle 1950's. Since then a variety of forms of genetic algorithms have been developed and applied in many different fields (for example, in economics and finance, science, data mining and data analysis, and medicine).

There are four main steps to perform in a genetic algorithm: (1) randomly generating an initial population which has *n* solutions (called chromosomes in nature) and evaluating the fitness of each solution in the population, (2) selecting the solutions according to fitness, (3) creating a new generation through genetic operations such as crossover, mutation, and elitism and evaluating the fitness of new solutions again, and (4) repeating this cycle (2) and (3) until some condition is satisfied. Figure 2 shows the cycle of the basic genetic algorithm. Genetic algorithms can be various and complex techniques to solve problems, but Goldberg (1989) said a simple genetic algorithm yields impressive and good results in many practical problems.

These are the most typical genetic operators:

1. Elitism
2. Mutation
3. Crossover
4. Grow / Prune

The elitism operator copies some of the best solutions from the previous population. The best solutions are copied according to their fitness values and put into a new population. Therefore we can preserve the best solutions for the next generation.

A mutation operator can be achieved by modifying the bits in randomly selected positions of solutions. In binary coded solutions, for example, randomly selected elements of solutions are changed 1 to 0 and vice versa. There is an alternative mutation method called permutation operation which exchanges the values of two bits.

To perform a crossover, two solutions are randomly chosen and the crossover points of any two solutions are randomly determined where the chosen two solutions are split. The new two solutions are created by swapping parts of two solutions. Thus after crossover, there is a partial exchange of information between solutions.

Finally, with the grow (prune) operation, some new weak classifiers are randomly generated (chosen) and added (removed) to the next generation to grow (prune) a solution.

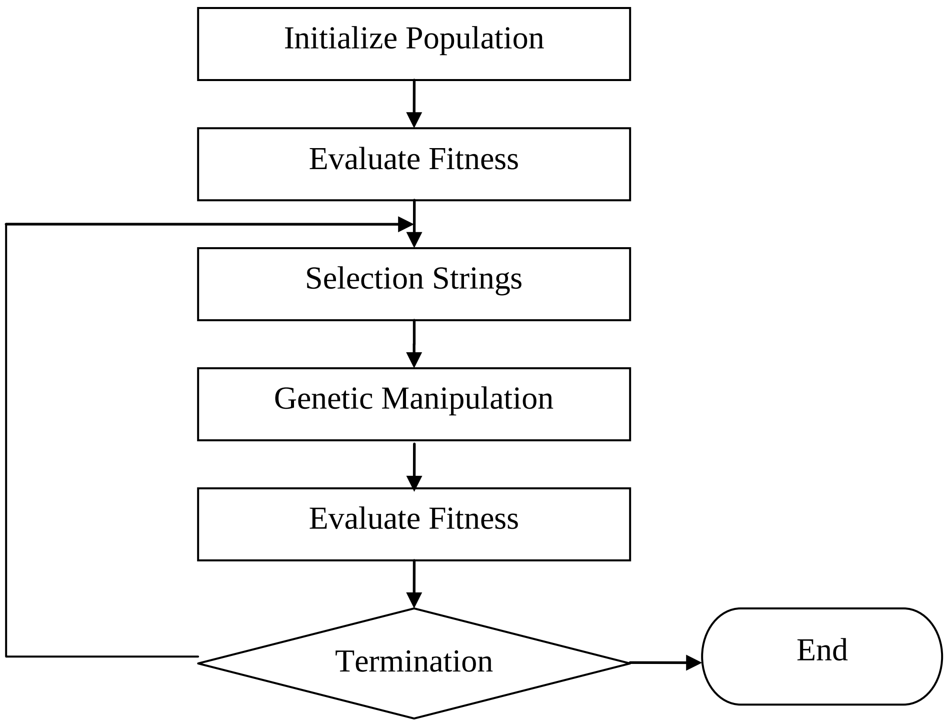


Figure 2. Genetic Algorithm Flow Chart

2.6 Summary

In this chapter, we introduced the classification problem and reviewed classical and recent methods for its solution. Traditional methods make many assumptions, while recent methods based on advanced computational techniques do not. Decision trees are popular for classification problems, but the predictive performance of tree models is not as good as that of ensemble models.

We reviewed previous studies with respect to boosting, which is one of the most successful ensemble methods. AdaBoost has remarkable performance on training data, but it has the potential of overfitting, especially with noisy data. To improve the performance given noisy data, GentleBoost (Friedman, 2000), BrownBoost (Freund, 2001), and SmoothBoosting (Servedio, 2003) are suggested as an alternative to AdaBoost for the noisy data case. Also, properties of margin distribution have been studied and other new boosting methods have been suggested to improve generalization performance. AdaBoost$_\rho$ (Rätsch et al., 2002), LPBoost (Grove et al., 1998), and DOOM (Mason et al., 1998) are focused on making the margins as large as possible to improve generalization performance. For multiclass classification, AdaBoost.MH and AdaBoost.MR were introduced by Schapire and Singer (1999).

Based on the literature review, because boosting is based on a large combination of decision trees or weak classifiers, it has a lack of interpretability. To attain a perfect fit in the training data, AdaBoost devotes too much attention to outliers in the data. From these observations, we started our research discussed in the following sections.

Chapter 3

Proposed Research

In this chapter, we describe the problems with boosting using a simple example and illustrate our new methodology for addressing these problems. We will also describe our planned research and expected results.

Since boosting was introduced, many researchers have been studying and developing new boosting algorithms. Nevertheless, some problems with boosting have not been addressed successfully or at all. Typically, boosting works in a fixed number of rounds, but it is difficult to decide when to stop the rounds. Test error often goes down after training error reaches zero, and a tremendous number of rounds may be required to get better general performance.

Usually boosting has hundreds or thousands of weak learners in the final model, which brings about an interpretation problem. AdaBoost can be unsuccessful given deficient data or complex weak learners. There can be an overfitting problem in the presence of noisy data.

In relation to generalization performance, it is necessary to study other properties of the margin distribution, not only maximizing the minimum margin but also other particular margins, such as the p-th percentile of margins for a small value of p.

3.1 Simple Example

We show the effect of outliers on AdaBoost using the following simple example. The data are generated with one predictive variable which is 100 consecutive integer values from 1 to 100; *X* = {1, 2, 3… 100}. The response variable *Y* is binary coded as {-1, 1}. *Y* is allocated to three regions where *Y* is +1 if *x* is less than 20 and greater than 60, otherwise *Y* = -1,

$$Y = \begin{cases} +1 & for \quad 1 \le X \le 20 \quad or \quad 61 \le X \le 100 \\ -1 & for \quad 21 \le x \le 60 \end{cases}$$

Also we assign three outliers randomly to the data set by flipping the signs of *Y* for observations 6, 38, and 81. Therefore the training data set is $\{(1,+1),(2,+1),\ldots,(45,-1),(46,-1),\ldots,(100,+1)\}$.

The AdaBoost algorithm suggested by Freund et al. (1999) is employed with 200 iterations. Stumps (one split, two terminal nodes) are used as weak learners for this algorithm. Intuitively, only 9 weak learners (stump trees) are required to have a perfect fit with *x*=5, 6, 20, 37, 38, 60, 80, 81, and 100 as split points. Table 3.1 shows the sequence of 200 weak classifiers constructed by AdaBoost, which correctly classifies all of the data points, including the three outliers. This suggests that AdaBoost is sensitive to the outliers and overfits the data. Although 200 weak classifiers are used, beginning with iteration 32, a sequence of nine weak classifiers (shown in bold in Table 1) is repeated over the remaining iterations. The nine weak classifiers correctly identify the true split points at 20, 60, and 100, but also include split points at 5, 6, 37, 38, 80 and 81 to fit the three outliers at 6, 38, and 81.

Zhu (2008) noted that AdaBoost tends to have low correlation between successive weak classifiers, but we can observe a repeating and cyclic pattern in the sequence of 200 weak classifiers constructed by AdaBoost through the simulated simple example. So we may ask a question: Why are the weak learners repeating? We suggest that AdaBoost constructs weak learners with their weights, but the weights are probably not optimized so that AdaBoost tends to revisit and reweight some weak learners later. Therefore we can adjust the weights of weak learners and reduce the number of weak learners. In order to solve this problem, we will optimize the weights and weak learners simultaneously instead of sequentially. GA-Boost, which is proposed in the next subsection, is one way to handle this problem using an objective fitness function. Another possibility is to find some best weak learners, perhaps from AdaBoost, then optimize the weights to produce the best prediction.

Table 3.1. AdaBoost Results for Simulated Data Set. Split points and predictions of each weak classifier on 200 iterations (1st column) from first simulated data set with outliers (at 6, 38, and 81).

t	split point	prediction	*t*	split point	prediction
1	60	-1	41	100	1
2	20	1	42	81	-1
3	100	1	43	80	1
4	60	-1	44	60	-1
5	20	1	45	38	1
6	100	1	46	37	-1
7	60	-1	47	20	1
8	20	1	48	6	-1
9	6	-1	49	5	1
•••			•••		
32	**100**	**1**	192	20	1
33	**81**	**-1**	193	6	-1
34	**80**	**1**	194	5	1
35	**60**	**-1**	195	100	1
36	**38**	**1**	196	81	-1
37	**37**	**-1**	197	80	1
38	**20**	**1**	198	60	-1
39	**6**	**-1**	199	38	1
40	**5**	**1**	200	37	-1

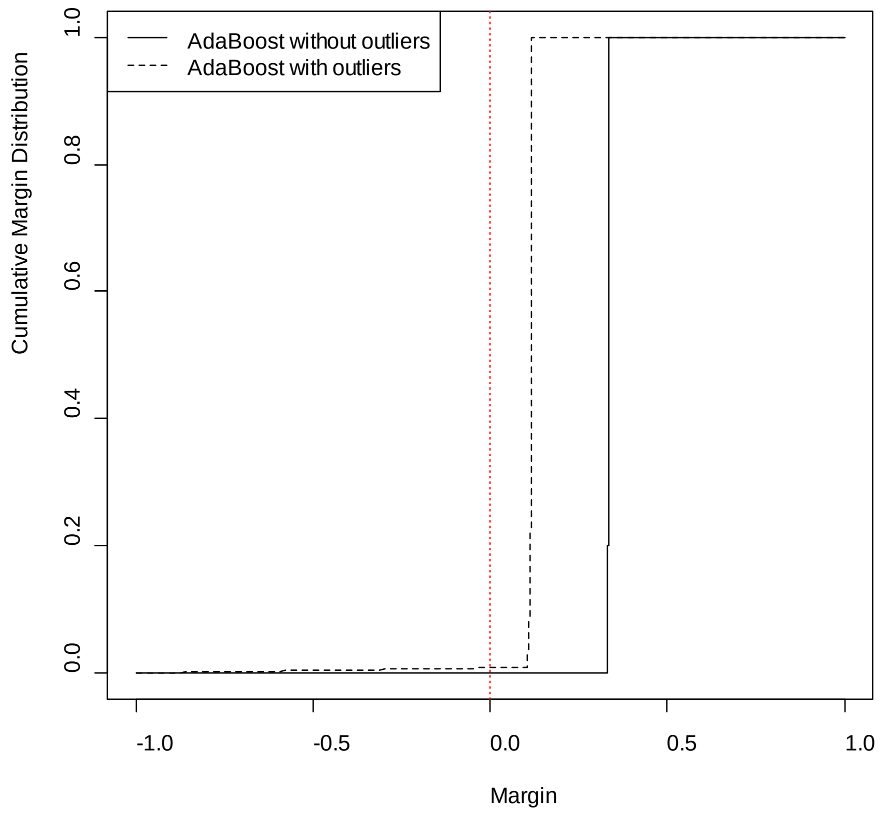


Figure 3.1. Cumulative Margin Distributions for AdaBoost (200 iterations): The CMD curves for the simulated data set without outliers (solid line) and with outliers (dashed line).

Figure 3.1 shows the cumulative margin distribution functions of AdaBoost for the simulated data set with the three outliers and without the outliers. The dashed line from AdaBoost with outliers is close to zero so that we have lower confidence predictions than the confidence from the solid line which is from AdaBoost without outliers.

There is another interesting observation. AdaBoost devotes too much attention to the outliers to get a perfect fit. The problem has also been suggested in earlier studies (Grove and Schuurmans 1998; Dietterich 2000). This results in smaller positive margins for the vast majority of the data and the potential for overfitting. So AdaBoost is sensitive to the outliers and overfits the data. In other words, to get a perfect fit including the three outliers, we have smaller positive margins for the majority of the data so that we have less confidence in most predictions and have the potential for overfitting.

3.2 New Boosting Algorithm (GA-Boost)

We propose a new boosting method based on a genetic algorithm that is called GA-Boost (Genetic Algorithm Boosting). GA-Boost is designed to solve directly for the weak learners and the weights of those weak learners using a genetic algorithm. In this new algorithm, the number of weak learners can be limited by utilizing a new penalized fitness function and controlling initial number of weak learners. We expect GA-Boost can dramatically reduce the number of weak learners but retain the same or better performance than AdaBoost. Thus, the final model has an interpretability advantage over AdaBoost due to the reduced model complexity as measured by the number of weak learners.

As seen above, AdaBoost is very sensitive to outliers. Other boosting algorithms based on maximizing the minimum margin give worse output because of the effects of outliers. Outliers may have large negative margins and maximizing the margin of an atypical example is not attractive. We anticipate GA-Boost can control the effects of outliers by the choice of an appropriate fitness function based on the cumulative margin distribution.

The genetic algorithm for GA-Boost is composed of three parts: (1) initialization and randomization, (2) fitness function, and (3) genetic evolution.

3.2.1 Initialization and Randomization

The initial population is randomly generated with 20 solutions for a given value or range of *T*, the number of weak classifiers. Each solution consists of *T* weak classifiers. Each of the *T* weak classifiers has a weight, α_t, and all of the weights are greater than zero and sum to 1.0 for each solution (Default of number of random weak classifiers in the initial random solutions: 5). Stumps, which are single-split trees with only two terminal nodes, are used as the weak classifiers. The split variables and split values of the stumps are randomly chosen for the initial population of solutions.

3.2.2 New Resistant Fitness Function

In this genetic algorithm, a solution is a set of weak learners and their weights. The fitness function of a genetic algorithm measures the goodness of a solution and is used to evaluate each solution in order to decide to what degree it will contribute to the next generation of solutions. Thus higher fitness solutions will influence the next generation more than solutions with lower fitness. The following fitness formula is used in the new boosting algorithm to measure the fitness of solutions.

$$f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0) - bT_s \tag{3.1}$$

where

M_p : p^{th} percentile of margins distribution

$I(\bullet)$: indicator function

m_i : i^{th} margin

T_s : number of weak classifiers in the solution s

a, b : tuning constant

This new fitness function for GA-Boost is based on maximizing the *p*-th percentile of the margin distribution for a few big negative margin outliers, $I(\bullet)$ is an indicator function, m_i is the *i*-th margin, *n* is the number of observations, T_s is the number of weak learners in the solution *s*, and *a* and *b* are weights (tuning constants). Effectively, $f(s)$ is the *p*-th percentile of the margin distribution with penalties for the number of negative margins and the number of weak learners.

Maximize p-th percentile of Margin

Boosting and AdaBoost try to increase margins in order to reduce generalization error. Some researchers have developed algorithms to maximize the minimum margin (e.g., LP-Boost, DOOM). These algorithms provide better minimum margins than AdaBoost, but their experiments do not show much impact on generalization performance. If there are outliers in the data, this would lead to putting too much emphasis on the outliers to the detriment of the majority of the data. Therefore they may have worse performance because of the effect of outliers.

Instead of maximizing the minimum margin, we consider maximizing a low percentile of the margins. We don’t concentrate on outliers. We trim or ignore *p*% of the smallest margins and consider maximizing the *p*-th percentile, leading to a more outlier-resistant solution. Maximizing the minimum margin is a special case of maximizing the *p*-th percentile of the margins. By maximizing the *p*-th percentile of the margins, we effectively trim or ignore *p*% of the smallest margins, leading to a more outlier-resistant solution. The M_p term in the fitness function helps in maintaining outlier resistance.

With respect to generalization error, the *p*-th percentile of the margins is utilized to get better generalization error bounds. Schapire et al. (1998) provided an upper bound on the generalization error given by

$$\Pr[\mathrm{H(x)} \neq \mathrm{y}] \leq \hat{\Pr}[\mathrm{margin}(x, y) \leq \theta] + O\left(\sqrt{\frac{d}{m\theta^2}}\right), \tag{3.2}$$

for any $\theta > 0$. Note that this upper bound is based on margin percentiles. To have more confidence and better training error, we need to decrease θ, but the complexity term, which scales as $O(\theta^{-1})$, is increased and vice versa. Thus the *p*-th percentile of margins (θ) plays an important role as a tuning constant, and we need to determine how to choose *p* based on different data sets.

Penalty for the number of negative margins

We allow for a small percentage of margins to be large and negative, but we also need to provide incentive in the fitness function to keep the number of negative margins to a minimum; otherwise, GA-Boost may try to allow exactly *p*% of the margins to have large negative values. We also believe this term helps in maintaining outlier resistance when the number of outliers in the data exceeds *p*%. In the case when the value of *p* is less than the noise level, the fitness function in GA-Boost allows for trimming less than the noise level. For example, suppose we maximize the 5th percentile of the margins for a data set with 10% noise level. GA-Boost ignores 5% of the examples with the smallest margins, but we still have 5% difficult to classify points. And they may have large negative margins. On the contrary, if the trim size (*p*) is larger than the noise level, GA-Boost permits extra trimming or ignores as much as proportion $p - r > 0$ of "good" data, where *r* is the noise level. Those examples may have large negative margins.

Because the penalty term for the number of negative margins can help keep the number of negative margins to a minimum, it should improve the fitness value when the trim size is not matched to the noise level.

Penalty for solution complexity (as measured by T_s)

Boosting and AdaBoost usually have hundreds or thousands of weak classifiers. There can be too many weak learners in the final model, which then causes complexity problems. In contrast with Zhu (2008), we observed a repeating and cyclic pattern in the sequence of weak learners constructed by AdaBoost, as illustrated in the simple example above. Thus we need to construct a simpler model combined with distinct weak learners so that we can keep out of the repeating and cyclic pattern in the sequence of weak learners. The fitness function is further penalized with respect to complexity as measured by the number of weak learners. With reduced complexity, GA-Boost solutions can be more easily interpreted than solutions from existing boosting methods. In many cases, we hope to achieve the predictive accuracy of AdaBoost with a much smaller number of weak learners.

Tuning parameters (a and b)

As tuning parameters, parameter *a* helps in maintaining outlier resistance by weighting the penalty for the number of negative margins and parameter *b* helps to control the complexity. If we choose a large value for *a*, we may have a more outlier resistant solution when *p*% is less than the proportion of outliers in the data set, and vice versa. If tuning parameter, *b*, is very small it makes for a lot of weak learners. Alternatively, if we use a large value of *b*, then we may have an underfitting problem. Therefore we need to study those tuning constants as well.

3.2.3 Genetic evolution

After 20 solutions are randomly generated, the fitness of each solution is evaluated. The next generation (as well as later generations) is created through genetic operations on the current generation of solutions. The evaluation of the fitness of new solutions is performed for each generation, and the evolution process is continued until some condition is satisfied. The solution in the final generation with the largest fitness value is chosen as the final solution. Five genetic operations are used in GA-Boost: elitism, mutation, crossover, grow, and prune.

The elitism operation copies some of the best solutions from the previous generation. Best solutions are copied according to their fitness values in the previous generation and placed into the new generation. In this way, good solutions are not lost (Default of number of best solutions copied to the new generation: 2).

To perform a crossover operation, two solutions are randomly chosen with probability proportional to fitness, and the single crossover points are randomly determined for splitting the two solutions. The chosen two solutions (parents) are split at their respective crossover points, and the two new solutions (children) are created by swapping parts of the two original solutions (parents). The new solutions are normalized so that their weak classifier weights sum to 1 (Default of number of crossovers in each generation: 8).

There are four different mutation operations that randomly change the weight, split rule, split value, or prediction of a randomly selected weak classifier on randomly selected solutions. In the first mutation operation, the weight of a randomly selected weak classifier is changed to a random value between 0 and 1, and then the weights of the mutated solution are normalized to sum to 1. In the split rule (split value) mutation, the split rule (split value) of a randomly selected weak classifier is replaced with a randomly generated split rule (split value). In the fourth type of

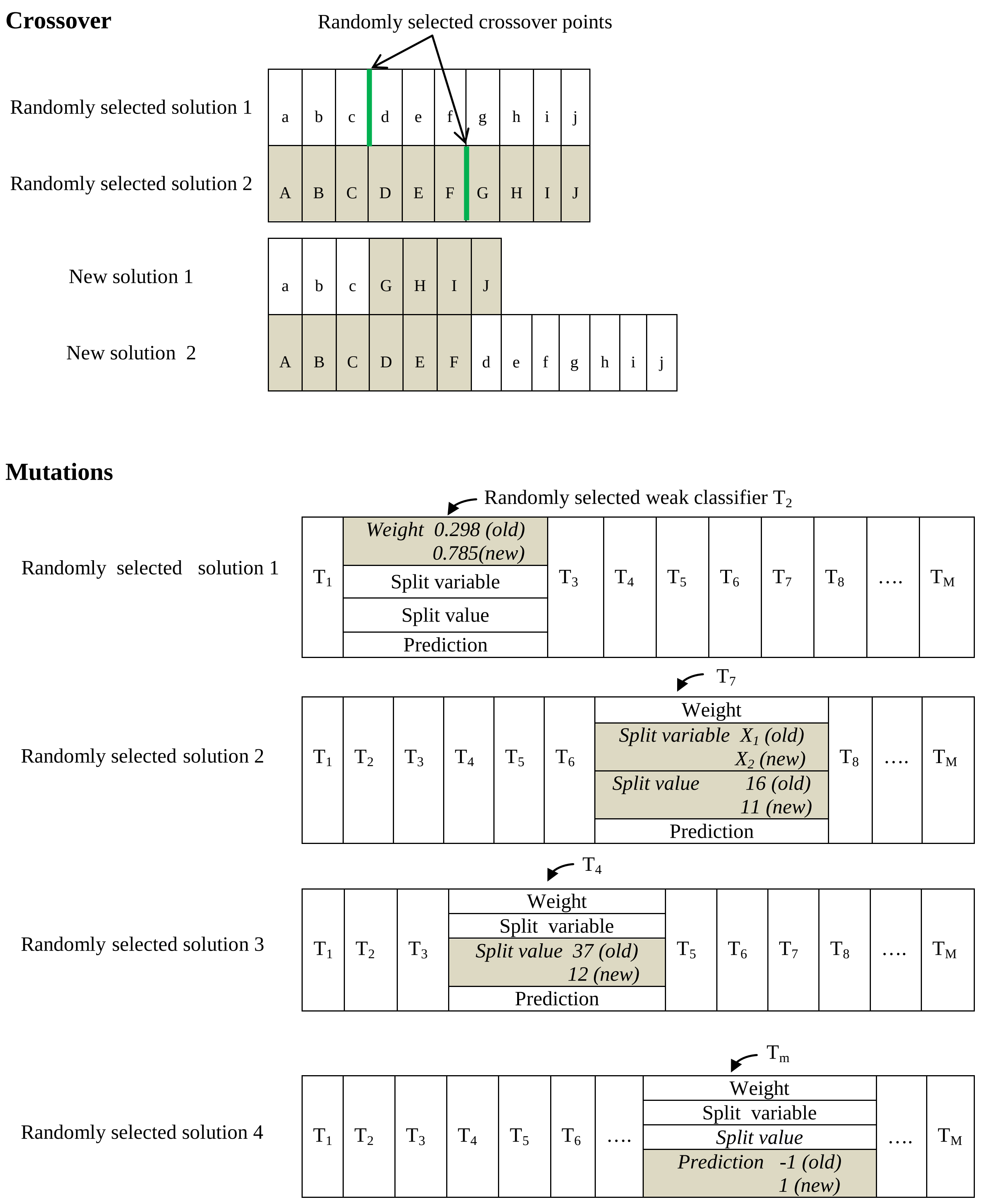
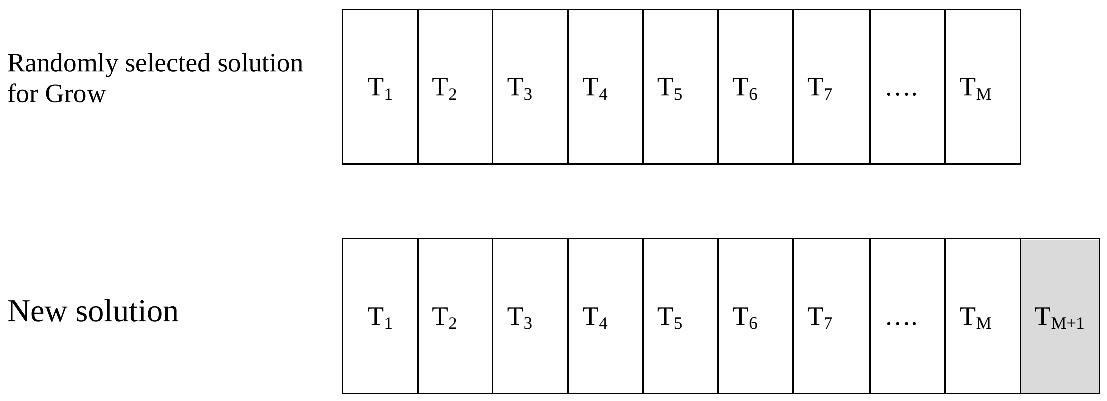

Crossover
Randomly selected crossover points
Randomly selected solution 1
a b c d e f g h i j
Randomly selected solution 2
A B C D E F G H I J
New solution 1
a b c G H I J
New solution 2
A B C D E F d e f g h i j
Mutations
Randomly selected weak classifier T2
Randomly selected solution 1
T1 | Weight 0.298 (old) 0.785(new) | Split variable | Split value | Prediction | T3 T4 T5 T6 T7 T8 …. TM
T7
Randomly selected solution 2
T1 T2 T3 T4 T5 T6 | Weight | Split variable X1 (old) X2 (new) | Split value 16 (old) 11 (new) | Prediction | T8 …. TM
T4
Randomly selected solution 3
T1 T2 T3 | Weight | Split variable | Split value 37 (old) 12 (new) | Prediction | T5 T6 T7 T8 …. TM
Tm
Randomly selected solution 4
T1 T2 T3 T4 T5 T6 …. | Weight | Split variable | Split value | Prediction -1 (old) 1 (new) | …. TM

Figure 3.2 Diagrams of crossover (above) and mutation (below). Weak classifiers T_2, T_7, T_4, and T_m in each randomly selected solution are randomly selected, and then *old* elements of the weak learners are randomly changed to the *new* values in mutation.

Grow


Randomly selected solution for Grow
T_1 T_2 T_3 T_4 T_5 T_6 T_7 …. T_M
New solution
T_1 T_2 T_3 T_4 T_5 T_6 T_7 …. T_M T_{M+1}

Prune

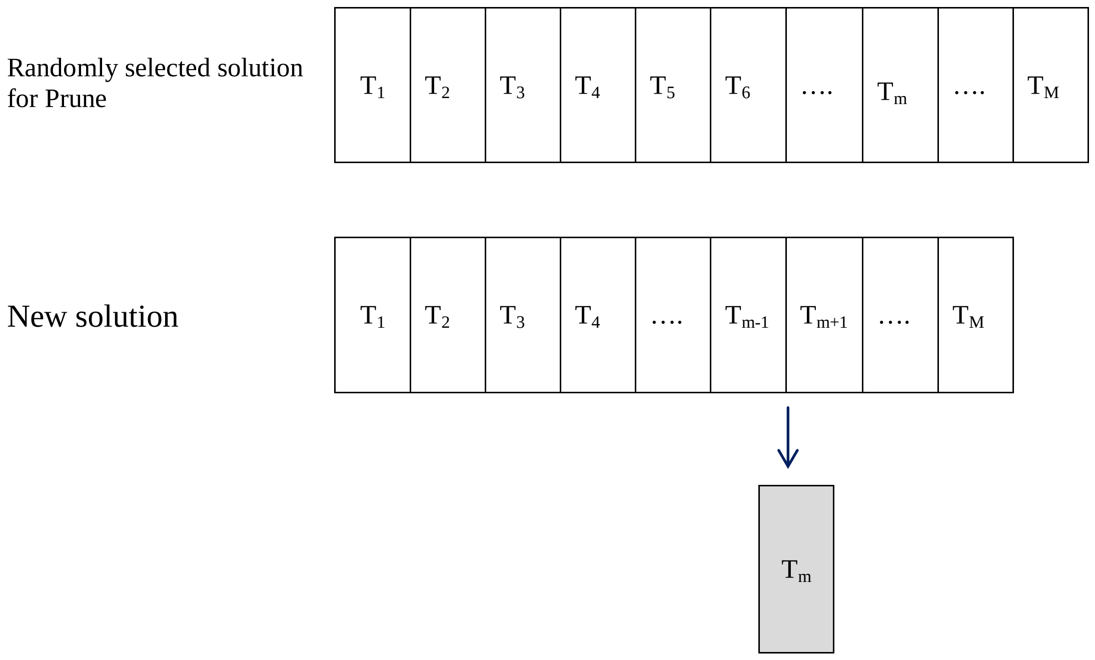

Randomly selected solution for Prune
T_1 T_2 T_3 T_4 T_5 T_6 …. T_m …. T_M
New solution
T_1 T_2 T_3 T_4 …. T_{m-1} T_{m+1} …. T_M
T_m

Figure 3.3 Diagrams of grow (above) and prune (below). A new weak classifier (T_{M+1}) is randomly generated and added to the next generation to grow a solution. And a weak classifier (T_m) is randomly chosen and removed from the next generation to prune a solution.

mutation, the prediction of a randomly selected weak classifier is switched from –1 to 1 (or 1 to –1) (Default of number of mutations in each generation: 8).

Finally, there are a grow operation and a prune operation. Some new weak classifiers are randomly generated and added to the next generation to grow a solution. And some weak classifiers are randomly chosen and removed from the next generation to prune a solution (Default of number of grown and prune in each generation: 1).

3.2.4 Algorithm

The pseudocode for GA-Boost algorithm is given in Figure 3.4. The input training set is given with *n* examples which are pairs: $(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)$, where $\underline{x}_i$ belongs to some instance space *X* and the label of y_i is noted by {-1, +1}. We choose some percentile, *p*, to maximize and choose the number of generations for the genetic algorithm and the number of weak learners. For the initializing step, the population consists of *N* randomly generated solutions (Default of number of solution: 20). A solution is a set of weak learners and their weights which are randomly chosen for the initial population of solutions. We evolve the solutions through *K* generations, and each generation is evaluated by the new fitness function (3.1). Genetic operations (elitism, mutation, crossover, grow and prune) are used to create a new generation of solutions and the fitness function of a genetic algorithm measures the goodness of each solution to determine the next generation of solutions. Finally after *K* simulations we choose the solution in the final generation with the largest fitness value as the final solution, which consists of *T* weak learners and their weights.

```
Input  $n$ examples $(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)$  where $\underline{x}_i \in X$, $y_i \in Y = \{-1,+1\}$
        $p$ (percentile), $K$ (number of generations), $T$ (initial number of weak learners)
Initialize  a randomly generated population of $N$ solutions
        (A solution $s$ consists of a set of $T_s$ weights ($\alpha_t$) and $T_s$ weak learners.)
        Weak learners are randomly chosen for the initial population of solutions.
Evolve  for $k = 1, 2, \ldots, K$ generations
          Evaluate fitness of solutions:  $f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0) - bT_s$
          Use genetic operations to create new generation of solutions:
              Elitism, Mutation, Crossover, Grow and Prune
Output  Final hypothesis with weights $\alpha_t$
```
Figure 3.4. The GA-Boost algorithm.

3.3 Comparison

In this section we compare GA-Boost to AdaBoost on the same simple data set (number of variables=1 with 100 consecutive integer values) described in section 3.1 above and two-class data sets from the UC-Irvine Machine Learning Repository. We use the AdaBoost algorithm suggested by Freund et al. (1999) with 200 iterations and the GA-Boost algorithm maximizing the 5th percentile of the margins as described in the previous section using 1,000 generations with *a*=0.1 and *b*=0.1. Stumps (one split, two terminal nodes) are used as weak classifiers for both algorithms.

3.3.1 Simulated simple data set

In Table 1 we saw the results from AdaBoost for the simple example in section 3.1. Table 3.2 shows the GA-Boost solution for the simple data set, which requires three weak classifiers to correctly identify the true split points. The outliers are ignored by the GA-Boost solution and are misclassified, as they should be in an outlier-resistant solution. The remaining 97 observations are correctly classified.

Table 3.2. GA-Boost Solution. Three weak classifiers for simulated data set with outliers (at 6, 38, and 81).

	Stump1	Stump2	Stump3
Weight	0.3358	0.3283	0.3359
Split Rule	$X \leq 60$	$X \leq 20$	$X \leq 100$
Prediction	-1	1	1

In Figure 3.5 the cumulative margin distributions for AdaBoost and the GA-Boost solution (maximizing the fifth percentile of the margins) are shown for the simulated simple data set with three outliers introduced by flipping the signs of the binary outcome variable for three observations. All examples are correctly classified by AdaBoost, but with small positive margins: most of the observations have margin values less than 0.2. AdaBoost provides a perfect fit, but at the expense of less predictive confidence for the good data when there are outliers.

On the other hand, GA-Boost utilized three stumps in its fit and misclassified only the three outliers (the only observations with negative margins). The GA-Boost fit has larger positive margins than AdaBoost, except for those of the three outliers. Of course, the GA-Boost criterion does not allow for a perfect fit in this situation. But it does allow up to 5% of the lowest margins to be effectively ignored so that the vast majority of the data are better classified with higher margins and confidence. Except for the three outliers, the GA-Boost fit has larger positive cumulative margins than AdaBoost. GA-Boost provides a simpler solution and is more resistant to outliers than AdaBoost.

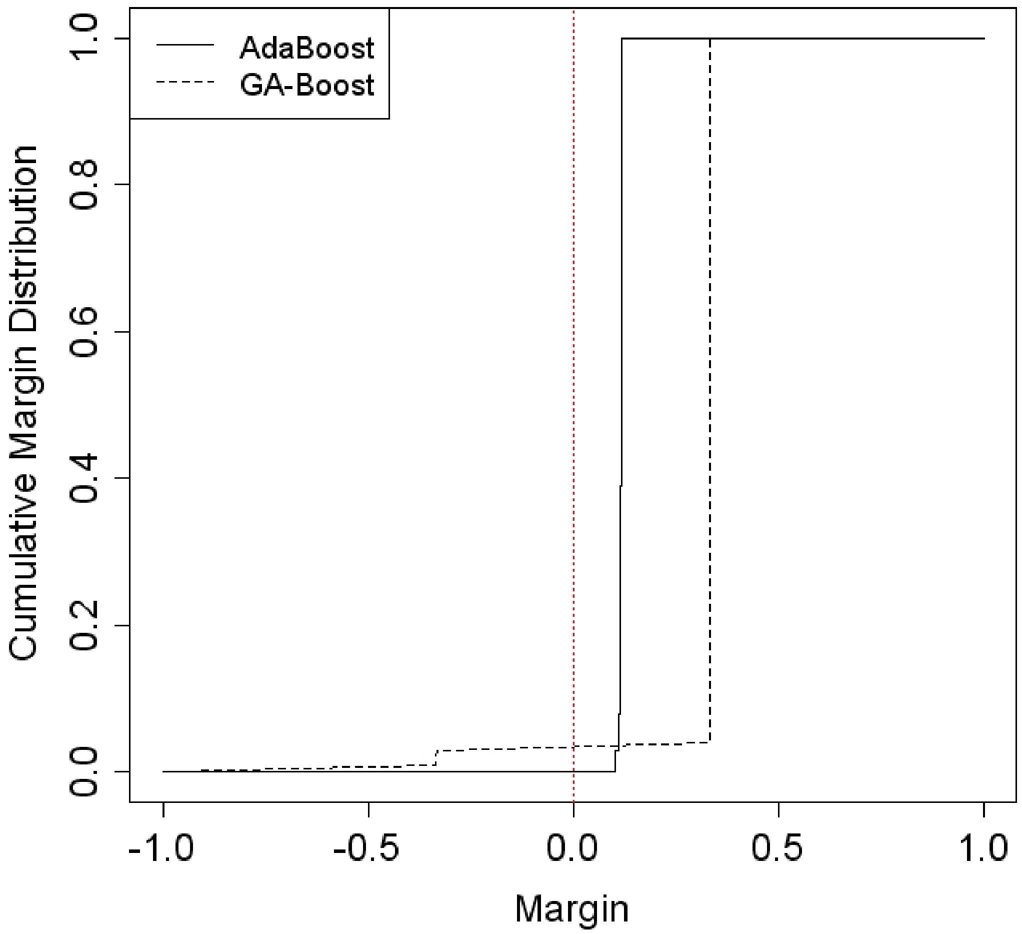


Figure 3.5 Cumulative Margin Distributions for AdaBoost and GA-Boost. The two methods compared are GA-Boost (maximizing the fifth percentile of the margin distribution) and AdaBoost (200 iterations) based on the simulated simple data set with three outliers.

3.3.2 Real Data Sets

For the real world examples, we consider two data sets from the UCI Machine Learning Repository: the Glaucoma data (n = 196, p = 62) and the Wisconsin Diagnostic Breast Cancer (WDBC) data (n = 569, p = 30). Table 3 shows the training error and number of weak learners utilized in AdaBoost and GA-Boost. The cumulative margin distributions of AdaBoost and GA-Boost (maximizing the 5th percentile of the margins) for these two data sets are shown in Figures 3.6 and 3.7.

Table 3.3 Training Errors for AdaBoost and GA-Boost

	Glaucoma	WDBC
AdaBoost	9.2% T=200 (162)	1.2% T=200 (152)
GA-Boost	9.7% T=3	3.2% T=7

We chose 200 rounds in AdaBoost in order to have a fair comparison with GA-Boost because of equalizing training errors from both algorithms. Thus we tried to find similar training error of both algorithms for those data sets. For the Glaucoma data in Table 3.3, AdaBoost has slightly better training errors than GA-Boost. AdaBoost used 200 stumps but 162 distinct stumps (the number in parentheses) are employed, and GA-Boost used 3 distinct stumps with 9.2% and 9.7% training error, respectively. AdaBoost has 152 distinct stumps for WDBC with 1.2% training error. On the other hand GA-Boost utilized only 7 stumps with 3.2% training error.

Figures 3.6 and 3.7 show that the GA-Boost fit has larger positive margins than AdaBoost, except for a few examples. Because GA-Boost focuses on maximizing the 5th percentile of the margins, it ignores the examples which are among the 5% lowest margins.

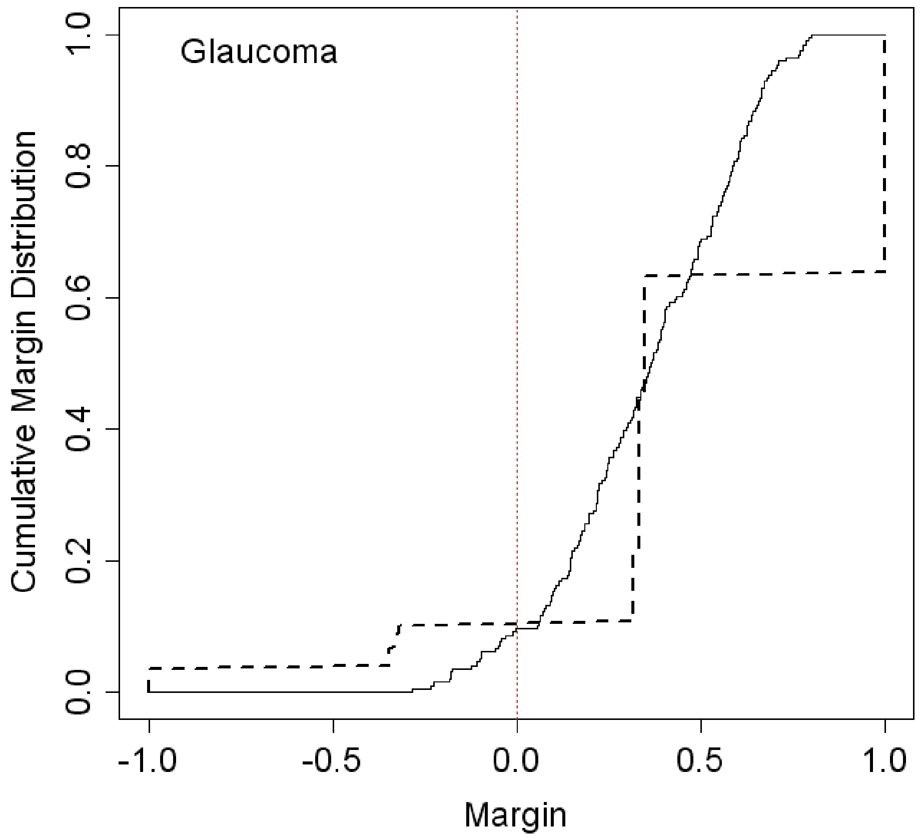


Figure 3.6. Cumulative Margin Distributions of AdaBoost (solid line) and GA-Boost (dashed line) for the Glaucoma Data.

For the Glaucoma data, GA-Boost ignores 5% of lowest margins, but 80% of its margins are bigger than AdaBoost. In Figure 3.7, the two margin distribution curves appear very close. For about 70% of the observations, GA-Boost has larger margins than AdaBoost.

The training errors for these two examples are similar, from the cumulative margin graphs, we see that the majority of margin values from GA-Boost is larger than for AdaBoost. Thus we have more confident predictions using GA-Boost. In addition, the GA-Boost solution used very few weak classifiers. While AdaBoost utilized 162 and 152 weak learners, GA-Boost required only 3 and 7 weak classifiers for the Glaucoma and WDBC data, respectively. Due to this complexity reduction, we anticipate that GA-Boost will provide better generalization performance. In chapter 4, we report the result in simulations to compare the test set performance of AdaBoost and GA-Boost.

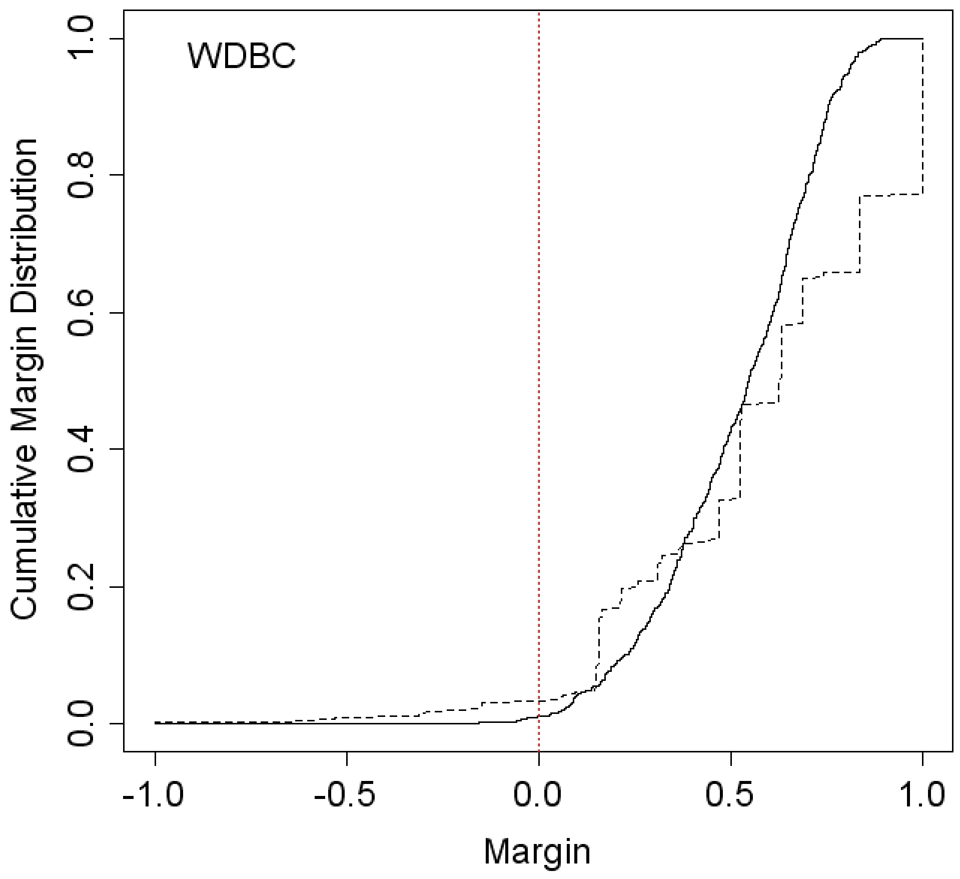


Figure 3.7. Cumulative Margin Distributions of AdaBoost (solid line) and GA-Boost (dashed line) for WDBC Data.

3.4 Summary and Consideration of GA-Boost

In this chapter, we have described GA-Boost, which is a new boosting method for binary classification using a genetic algorithm. GA-Boost is more resistant to the effects of outliers and provides solutions that are simpler than those found by AdaBoost, as illustrated by a simulated simple example and two real world data sets.

Simulation experiments to compare test set performance for AdaBoost and GA-Boost are showed in the next chapter including the following described possible improvements to GA-Boost to be considered.

GA-Boost utilizes stump trees (one split with two terminal nodes) as weak learners in this chapter. In the next chapter we see that the more complex weak learners with 3 split points and oblique stump trees work well rather than just a random split rule with one split, but it ought to be more expensive computationally and can worsen interpretability.

The new fitness function for GA is based on the *p*-th percentile of the margin distribution to allow outliers to have large negative margins. Thus there is an important question about how to choose *p* to trim or ignore *p*% of the smallest margins. In order to choose *p* we can draw cumulative margin graphs using various values of *p* to decide a best value of *p*. A plot of *p*-th percentile versus margins might also help in selecting *p*. Because outliers usually have big negative margins, to observe the gap among the margins of outliers and normal examples may also help to decide which *p* to use. Additionally an upper bound of test error based on margin percentiles suggested by Schapire et al. (1998) in equation (3.2) may be used for choosing *p* by comparing test errors from varied values of *p*. As *a* and *b* are tuning parameters in a new fitness function, parameter *a* helps in maintaining outlier resistance due to weight the penalty for the number of negative margin and parameter *b* helps to control the complexity (*T*). Thus it would be interesting to find best tuning parameters (*a* and *b*) for different data sets. We report the effect of those parameters for GA-Boost in the next chapter.

Chapter 4

Simulation Studies

In this chapter, we present simulation studies of the performance of the new boosting algorithm. There are two artificial data sets to demonstrate how GA-Boost works with different weak learners and how it compares to AdaBoost based on different noise levels. To look at the effect of noise, we add random noise at different levels to the data set and display the effect on GA-Boost with various *p*-th percentiles to be maximized. In addition we examine the effect of the tuning constants, *a* and *b*, in the fitness function of the genetic algorithm. Simulations were designed for 16 treatments corresponding to different combinations of *p*, *a*, and *b*.

In the first simulation, we generated data sets with four different noise levels (0%, 1%, 5%, and 10%) and a simple data structure with two predictors. Intuitively we expect that few weak learners are needed to classify correctly so that we assign a larger penalty *b* for the number of weak learners in the fitness function to produce few weak learners. Stump trees with two terminal nodes were used as weak learners for GA-Boost and AdaBoost. The second simulation uses a concentric circle data structure, which is a more complex data structure than the first one. Stump trees were used as weak learners for GA-Boost and AdaBoost. To extend weak learners beyond stump trees, we also considered decision trees with three split points and oblique stump trees for GA-Boost. Those were applied to a training set and their performance evaluated using a test set.

Through these simulation tests, we will see how to best maximize the *p*-th percentile of the margins based on a simple data set with different noise levels and for an intricate data set. There are two motivations for maximizing the *p*-th percentile of margins. One is that by maximizing a low percentile of the margins, we trim *p*% of the smallest margins which leads to a more outlier resistant solution. The other motivation is provided by a generalization error bound. As mentioned in chapter 3, Schapire et al. (1998) suggested the following upper bound on generalization error:

$$\Pr[H(x) \neq y] \leq \hat{\Pr}[\text{margin}(x, y) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right). \tag{4.1}$$

This upper bound is based on margin percentiles, where θ is the *p*-th percentile of the margins. If we ignore more examples with the smallest margins by increasing θ, then the training error, which is the first term in the generalization error bound, is increased, but the second term is decreased, and vice versa. A good choice for the value of the *p*-th percentile of margin will minimize the generalization error bound and hopefully lead to a small generalization error.

4.1 Simulation data I

The purpose of this simulation is to show how maximizing the *p*-th percentile in a new fitness function for GA-Boost performs, with respect to noise levels using two-terminal-node decision trees (stumps) as weak learners. For the first simulation, we consider a d = 2-dimension (*2* predictor variables) problem with n=100 observations. We take the predictor values to be randomly generated from a uniform distribution on the interval [0, 1]. We have a response variable, $Y \in \{-1, +1\}$ given by

$$Y = \begin{cases} -1 & for \quad x_1 \le 0.50 \quad and \quad x_2 \ge 0.50 \\ +1 & \qquad otherwise. \end{cases} \tag{4.2}$$

Y = -1 are marked by a circle and *Y* = +1 are marked by a cross in Figure 4.1. Noise points are randomly chosen without replacement in each quadrant of the plot (Figure 4.1) and their class labels are changed. Noise levels are 0%, 1%, 5%, and 10%. A set of 10,000 observations was generated for a test set.

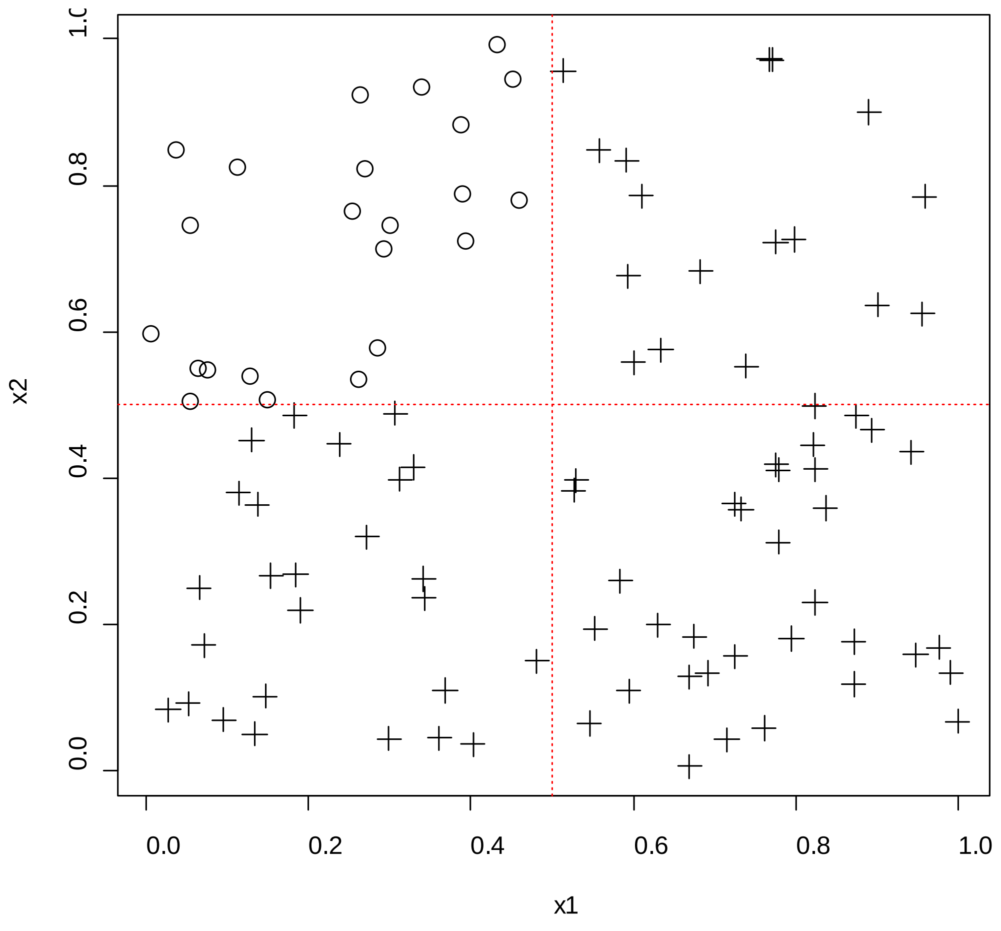


Figure 4.1. Simulation data I without noise

Before we construct the model based on these data sets, we need to set up the designed experiment on three factors, which are *p, a*, and *b*, where we set 5 as the initial number of weak learners. *p* is the percentile of margins to be maximized, and *a* and *b* are the tuning parameters for the penalties on the number of negative margins and the number of weak learners in the new fitness function of the genetic algorithm. Through experiments, we have a consistent convergence value of fitness with 1000 generations for this simple data set, which has 100 observations with 2 predictors. Because the best combination of *a*, *b*, and *p* depends on the data structure and it is impossible to test all values of *p*, *a*, and *b,* we designed a simulation experiment to study the effects of those parameters. First we set the maximizing percentile of the margins, *p,* to 0.0 (0%), 0.05, 0.10, and 0.15 to match the noise levels considered. The value of *p*=0.0 corresponds to maximizing the minimum margin.

We set the values of both *a* and *b* to 0.1 or 0.7 so that there are four different combinations (*a*, *b*): (0.1, 0.1), (0.1, 0.7), (0.7, 0.1) and (0.7, 0.7) with 4 different values of *p*, which are 0.00, 0.05, 0.10, and 0.15. With those combinations we can test the main effects as

Table 4.1. Treatment levels of designed experiments for data set I.

Treatment	1	2	3	4	5	6	7	8
p	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.05
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7

	9	10	11	12	13	14	15	16
p	0.10	0.10	0.10	0.10	0.15	0.15	0.15	0.15
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7

well as the interactions of *a*, *b,* and *p*. Therefore this designed experiment on three factors leads to 16 treatments for GA-Boost with 1000 generations each. Table 4.1 displays the 16 treatments of the tuning parameters, *a*, and *b* with the value of *p* indicating which percentile to maximize. For each data set with its noise level, we have 10 replicated runs per treatment.

Table 4.2. Test errors and standard errors of each treatment for data set I without noise and 1% noise level.

Treatment		1	2	3	4	5	6	7	8
	p	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.05
	a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
	b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
0%	*Test Error*	0.0213	0.0179	0.0200	0.0192	0.0219	0.0210	0.0179	0.0176
	St. Error	0.0029	0.0018	0.0031	0.0017	0.0025	0.0025	0.0021	0.0019
1%	*Test Error*	0.0187	0.0173	0.022	0.0169	0.0187	0.0202	0.0182	0.0169
	St. Error	0.0019	0.0231	0.0018	0.0015	0.0014	0.0023	0.0368	0.0026

		9	10	11	12	13	14	15	16
	p	0.10	0.10	0.10	0.10	0.15	0.15	0.15	0.15
	a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
	b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
0%	*Test Error*	0.0190	0.0185	0.0158	0.0176	0.0152	0.0175	0.0220	0.0198
	St. Error	0.0242	0.0021	0.0012	0.0021	0.0019	0.0012	0.0022	0.0023
1%	*Test Error*	0.0201	0.0187	0.0183	0.0185	0.0214	0.0201	0.0216	0.0216
	St. Error	0.0018	0.0023	0.0021	0.0024	0.0031	0.0020	0.0021	0.0020

4. 1. 1 Data Set I without noise and with 1% noise level

The first simulation data is based on pure and contaminated simple data sets. Four different combinations of (*a*, *b*) and four different *p* apply to this simulation data. The first four treatments test for maximizing the minimum percentile of margins with (*a*, *b*)’s. The next four treatments, which are 5th to 8th, test for *p*=0.05, 9th to 12th for *p*=0.10, and 13th to the last treatment for maximizing 15th percentile of the margins with four different combinations (*a*, *b*): (0.1, 0.1), (0.1, 0.7), (0.7, 0.1) and (0.7, 0.7). In order to see the differences among the treatments with respect to the three factors, the following results show test errors and standard deviations of each treatment in Table 4.2 and present the effect of each treatment through box plots in Figures 4.2 and ANOVA in Tables 4.3 and 4.4.

Table 4.3. ANOVA table from simulation data set I without noise.

```
One-way ANOVA: Error versus Treatment
Source      DF          SS          MS     F      P
Treatment   15  0.0006176  0.0000412  0.85  0.626
Error      144  0.0070094  0.0000487
Total      159  0.0076270

S = 0.006977   R-Sq = 8.10%   R-Sq(adj) = 0.00%

                               Individual 95% CIs For Mean Based on
                               Pooled StDev
Level  N      Mean     StDev  ---+---------+---------+---------+------
 1    10  0.021300  0.009322                 (----------*----------)
 2    10  0.017900  0.005763         (----------*----------)
 3    10  0.020000  0.009854              (----------*----------)
 4    10  0.019200  0.005371            (----------*----------)
 5    10  0.021900  0.008034                   (----------*----------)
 6    10  0.021000  0.007902                 (----------*---------)
 7    10  0.017900  0.006574         (----------*----------)
 8    10  0.017600  0.005967        (----------*----------)
 9    10  0.018600  0.008527           (---------*----------)
10    10  0.018500  0.006737          (----------*----------)
11    10  0.015800  0.003706    (---------*----------)
12    10  0.017600  0.006603        (----------*----------)
13    10  0.015200  0.005922  (----------*----------)
14    10  0.017500  0.003808        (----------*----------)
15    10  0.022000  0.007071                   (----------*----------)
16    10  0.019800  0.007208              (----------*---------)
                              ---+---------+---------+---------+------
                               0.0120    0.0160    0.0200    0.0240

Pooled StDev = 0.006977
```

Table 4.4. ANOVA table from simulation data set I with 1% noise level.

One-way ANOVA: Error versus Treatment

```
Source      DF          SS          MS     F      P
Treatment   15  0.0004247  0.0000283  0.62  0.851
Error      144  0.0065299  0.0000453
Total      159  0.0069546

S = 0.006734   R-Sq = 6.11%   R-Sq(adj) = 0.00%


                               Individual 95% CIs For Mean Based on
                               Pooled StDev
Level   N      Mean     StDev  ----+---------+---------+---------+-----
 1     10  0.018700  0.006019       (-----------*-----------)
 2     10  0.017300  0.005187   (-----------*-----------)
 3     10  0.022000  0.005715                (-----------*-----------)
 4     10  0.016900  0.004701  (-----------*-----------)
 5     10  0.018700  0.004373       (-----------*-----------)
 6     10  0.020200  0.007254            (-----------*-----------)
 7     10  0.019500  0.007934          (-----------*-----------)
 8     10  0.016900  0.008171  (-----------*-----------)
 9     10  0.020100  0.005567           (-----------*-----------)
10     10  0.018700  0.007273       (-----------*-----------)
11     10  0.018300  0.006499      (-----------*-----------)
12     10  0.018500  0.007619       (-----------*-----------)
13     10  0.021400  0.009823               (-----------*-----------)
14     10  0.020100  0.006297           (-----------*-----------)
15     10  0.021600  0.006670                (-----------*-----------)
16     10  0.021600  0.006433                (-----------*-----------)
                               ----+---------+---------+---------+-----
                                 0.0140    0.0175    0.0210    0.0245

Pooled StDev = 0.006734
```

As seen in Table 4.2 and ANOVA Tables 4.3-4.6, the values of *p* with four different combinations of (*a*, *b*) do not significantly affect the result based on 0% and 1% noise in the data set. The treatments having the small value (0.1) of *b* have a little better performance than treatments with 0.7 for *b*, but not significantly better. From Table 4.5 and 4.6 we would conclude that all main effects (*a*, *b*, and *p*) and interactions (*a***b*, *a***p*, *b***p*, and *a***b***p*) have an insignificant effect. This pure data set is very simple to classify so that we don't need many weak classifiers, just three stumps are sufficient to correctly classify all training data. The tuning constant *b* controls the number of weak classifiers. We didn't show any results with respect to number of trees, but 155 of the 160 tests from 10 replicated runs for 16 treatments have three stumps in the final classifier for the pure data set, and 153 tests have three stumps in the best solution based on the data set with 1% noise for GA-Boost with 1,000 generations. Therefore the simple data sets with 0% and 1% contamination are not significantly affected by the 16 treatments with regard to performance.

Table 4.5. Analysis of variance for simulation data set I with 0% noise level

```
Analysis of Variance for Error, using Adjusted SS for Tests

Source   DF     Seq SS     Adj SS     Adj MS     F      P
a         1  0.0000025  0.0000025  0.0000025  0.05  0.821
b         1  0.0000081  0.0000081  0.0000081  0.17  0.684
p         3  0.0001070  0.0001070  0.0000357  0.73  0.534
a*b       1  0.0000002  0.0000002  0.0000002  0.00  0.946
a*p       3  0.0003757  0.0003757  0.0001252  2.57  0.056
b*p       3  0.0000469  0.0000469  0.0000156  0.32  0.810
a*b*p     3  0.0000772  0.0000772  0.0000257  0.53  0.663
Error   144  0.0070094  0.0070094  0.0000487
Total   159  0.0076270

S = 0.00697685   R-Sq = 8.10%   R-Sq(adj) = 0.00%
```

Table 4.6. Analysis of variance for simulation data set I with 1% noise level

```
Analysis of Variance for Error, using Adjusted SS for Tests

Source   DF     Seq SS     Adj SS     Adj MS     F      P
a         1  0.0000000  0.0000000  0.0000000  0.00  0.991
b         1  0.0000638  0.0000638  0.0000638  1.41  0.238
p         3  0.0001675  0.0001675  0.0000558  1.23  0.301
a*b       1  0.0000150  0.0000150  0.0000150  0.33  0.566
a*p       3  0.0000539  0.0000539  0.0000180  0.40  0.756
b*p       3  0.0000527  0.0000527  0.0000176  0.39  0.762
a*b*p     3  0.0000719  0.0000719  0.0000240  0.53  0.664
Error   144  0.0065299  0.0065299  0.0000453
Total   159  0.0069546

S = 0.00673398   R-Sq = 6.11%   R-Sq(adj) = 0.00%
```

4. 1. 2 Data Set I with 5% noise

Table 4.7. Test errors, standard errors, and average *T* of each treatment for data set I with 5% noise.

Treatment	1	2	3	4	5	6	7	8
p	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.05
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
Test Error	0.0262	0.2479	0.0215	0.0458	0.0181	0.0426	0.0219	0.0226
St. Error	0.0031	0.0007	0.0033	0.0238	0.0025	0.0234	0.0028	0.0040
T	4.2	2.0	4.4	3.4	3.0	2.9	3.1	3.0

Treatment	9	10	11	12	13	14	15	16
p	0.10	0.10	0.10	0.10	0.15	0.15	0.15	0.15
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
Test Error	0.0163	0.1434	0.0223	0.0189	0.0174	0.2123	0.0267	0.0166
St. Error	0.0018	0.0217	0.0038	0.0015	0.0029	0.0021	0.0023	0.0015
T	3.0	3.0	3.2	3.0	3.0	3.0	3.6	3.0

We used the rectangular data set and added 5% noise to the data set by flipping the *Y* values of five randomly selected observations. Hence we can see how GA-Boost works based on this contaminated data with 16 different treatments. This data with 5% noise is treated as more complex to correctly classify all training data than previous data sets (0% and 1% contamination). From Tables 4.7 and 4.8, we see that the second treatment has the worst performance and others are not significantly different from each other. Treatment 2 is a combination of maximizing the minimum margin, *a*=0.1 tuning for the number of negative margins, and *b*= 0.7 tuning for the number of weak learners in fitness function. Therefore GA-Boost with treatment 2 didn't ignore any difficult to classify points and tried to classify all examples with a restricted number of weak learners. As a result, treatment 2 utilized only 2 weak learners (average *T* out of 10 runs) in the best solution, which is not sufficient to correctly classify all training data. The small value of *a*=0.1, when *b* is relatively large could not help in maintaining outlier resistance so that the performance from treatment 2 is the worst. Other treatments except treatment 2 have 3, 4, or 5 weak learners in their final solution in Table 4.7.

Table 4.8. ANOVA table for test error versus 16 treatments from simulation data set I with 5% noise level.

One-way ANOVA: Error versus Treatment

```
Source      DF       SS       MS      F      P
Treatment   15  0.47865  0.03191  30.46  0.000
Error      144  0.15087  0.00105
Total      159  0.62951

S = 0.03237   R-Sq = 76.03%   R-Sq(adj) = 73.54%


                             Individual 95% CIs For Mean Based on
                             Pooled StDev
Level   N     Mean    StDev   -+---------+---------+---------+--------
 1     10  0.02620  0.00993     (--*--)
 2     10  0.24790  0.00228                                    (-*--)
 3     10  0.02150  0.01043    (--*--)
 4     10  0.04580  0.07532        (--*-)
 5     10  0.01810  0.00784    (--*-)
 6     10  0.04260  0.07396       (--*--)
 7     10  0.02190  0.00884    (--*--)
 8     10  0.02260  0.01272    (--*--)
 9     10  0.01630  0.00581   (--*--)
10     10  0.04900  0.06859        (--*--)
11     10  0.02230  0.01204    (--*--)
12     10  0.01890  0.00477    (--*--)
13     10  0.01740  0.00924    (-*--)
14     10  0.02200  0.00639    (--*--)
15     10  0.02670  0.00718     (--*--)
16     10  0.01660  0.00472   (--*--)
                              -+---------+---------+---------+--------
                             0.000     0.070     0.140     0.210

Pooled StDev = 0.03237
```

To see the effect of maximizing the *p*-th percentile of the margins, we can compare treatment 2 with three other treatments that have the same values of *a* and *b* (0.1, 0.7) but not *p*. Treatments 2, 6, 10, and 14 have different *p*-th percentiles to maximize: 0th, 5th, 10th, and 15th of margins, respectively. Figure 4.2 shows the boxplots for test error vs. four different *p*-th percentiles of margins for treatments 2, 6, 10, and 14 described in Table 4.7. Maximizing the minimum margin results in the worst performance and the others have similar results. Therefore to classify this data set with a 5% noise level, it is a good idea to utilize $p > 5\%$. In the cases of treatments 10 and 14, the trim size is larger than the noise level, so that we permit additional trimming or ignoring of good examples which may be assigned large negative margins. But the penalty term for the number of negative margins can cover this problem. As a result, with respect to test error, the choice of *p* should be utilized to get better performance when data sets are contaminated. The main effects of *a*, *b*, *p* and their interactions are significantly different as shown in Table 4.9.

Table 4.9. Analysis of Variance for simulation data set I with 5% noise level

```
Analysis of Variance for Error, using Adjusted SS for Tests

Source   DF    Seq SS    Adj SS    Adj MS      F      P
a         1  0.036966  0.036966  0.036966  35.28  0.000
b         1  0.054391  0.054391  0.054391  51.91  0.000
p         3  0.111855  0.111855  0.037285  35.59  0.000
a*b       1  0.046240  0.046240  0.046240  44.14  0.000
a*p       3  0.072095  0.072095  0.024032  22.94  0.000
b*p       3  0.100709  0.100709  0.033570  32.04  0.000
a*b*p     3  0.056391  0.056391  0.018797  17.94  0.000
Error   144  0.150867  0.150867  0.001048
Total   159  0.629515

S = 0.0323680   R-Sq = 76.03%   R-Sq(adj) = 73.54%
```

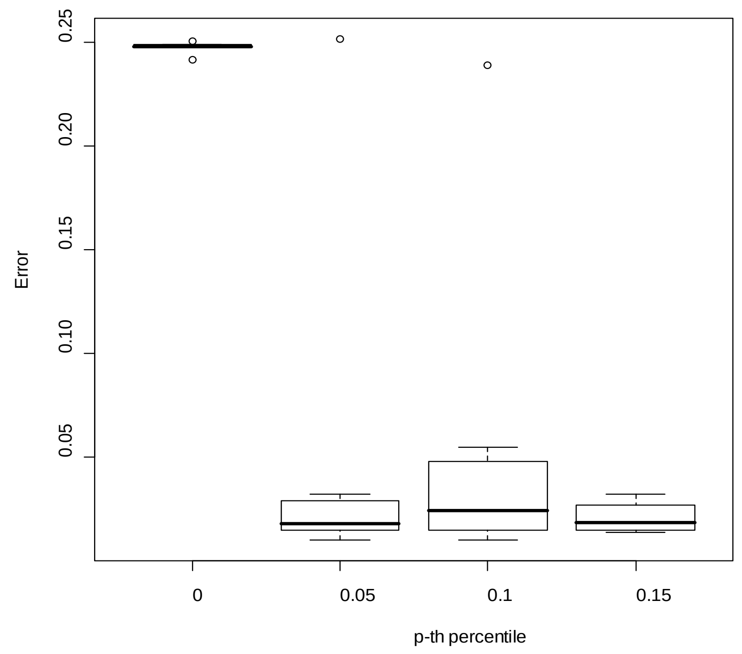


Figure 4.2. Boxplots of test error vs *p*-th percentile of margins for treatments 2, 6, 10, and 14 described in Table 4.7.

4.1.3 Data Set I with 10% noise

We generated data set I with 10% noise, which is more contaminated than previous data sets. We again see how GA-Boost performs on this contaminated data set with 16 different treatments. Treatment 2, having 0.1 for *a*, and 0.7 for *b* based on maximizing the minimum margin (*p*=0.0), has the worst performance based on Tables 4.10 and 4.11. GA-Boost with treatment 2 tries to maximize the minimum margin, which is not robust to noisy examples. Like the previous example, to see the effect of maximizing the *p*-th percentile of the margins, we can compare with other three different treatments that have same values of *a* and *b*, but not *p*, i.e., treatments 6, 10, and 14. Figure 4.3 shows the box plots for test error vs. four different values of *p* with same values of *a* and *b*.

Table 4.10. Test errors, standard errors, and average *T* of each treatment for data set I with 10% noise.

Treatment	1	2	3	4	5	6	7	8
p	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.05
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
Test Error	0.0845	0.2478	0.0513	0.0295	0.0270	0.0959	0.0371	0.0305
St. Error	0.0283	0.0003	0.0096	0.0046	0.0034	0.0331	0.0037	0.0044
T	3.8	2.0	5.1	3.1	3.1	2.7	3.6	3.2

Treatment	9	10	11	12	13	14	15	16
p	0.10	0.10	0.10	0.10	0.15	0.15	0.15	0.15
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
Test Error	0.0257	0.0747	0.0344	0.0238	0.0226	0.0934	0.0358	0.0257
St. Error	0.0091	0.0288	0.0967	0.0025	0.0037	0.0336	0.0053	0.0023
T	3.0	2.8	3.8	3.1	3.0	2.7	3.7	3.0

Therefore to classify this data set with a 10% noise level, it is optimal to maximize the 10th percentile of margins. In the case of treatment 14, the trim size is larger than the noise level so that we permit additional trimming or ignoring the 5% examples having the next smallest margins, which may have large negative margins. The case of maximizing the 5th percentile of margin is less than the 10% noise level. Thus the 5% of examples that are uncovered by the 5-th percentile may have large negative values of margins. But the penalty term for the number of negative margins in the fitness function may cover these problems. One might ask why the penalty term for the number of negative margins does not cover all the outliers' proportion in the case of maximizing the minimum margin. We explain that the benefit from maximizing the minimum margin term, $M_{p=0}$, in the fitness function is not better than the cost of the penalty term on the number of negative margins. This problem will be explained in more detail the next section. From Table 4.10 and the boxplots in Figure 4.3, maximizing the minimum margin has the worst performance. Others have a similar median but the deviation from maximizing the 10th percentile of the margins is the smallest among those in Figure 4.3. Similar to previous examples, we conclude that the main effects and interactions of *a*, *b*, and *p* are significant from the analysis of variance, except for an interaction between *b* and *p* in Table 4.12.

Table 4.11. ANOVA table for test error versus 16 treatments from simulation data set I with 10% noise level.

One-way ANOVA: Error versus Treatment

```
Source      DF        SS        MS      F      P
Treatment   15  0.48257  0.03217  12.59  0.000
Error      144  0.36787  0.00255
Total      159  0.85043

S = 0.05054   R-Sq = 56.74%   R-Sq(adj) = 52.24%


                             Individual 95% CIs For Mean Based on
                             Pooled StDev
Level   N     Mean    StDev   -+---------+---------+---------+--------
 1     10  0.08450  0.08939          (---*---)
 2     10  0.24780  0.00103                                (---*---)
 3     10  0.05130  0.03050     (---*---)
 4     10  0.02950  0.01446   (---*---)
 5     10  0.02700  0.01088  (---*---)
 6     10  0.09590  0.10458           (---*---)
 7     10  0.03710  0.01183   (---*---)
 8     10  0.03050  0.01391   (---*---)
 9     10  0.02570  0.00926  (---*---)
10     10  0.07470  0.09103        (---*---)
11     10  0.03440  0.01336   (---*---)
12     10  0.02380  0.00781  (---*---)
13     10  0.02260  0.01177  (---*---)
14     10  0.09340  0.10616           (---*---)
15     10  0.03580  0.01678    (--*---)
16     10  0.02570  0.00720  (---*---)
                              -+---------+---------+---------+--------
                             0.000     0.080     0.160     0.240

Pooled StDev = 0.05054
```

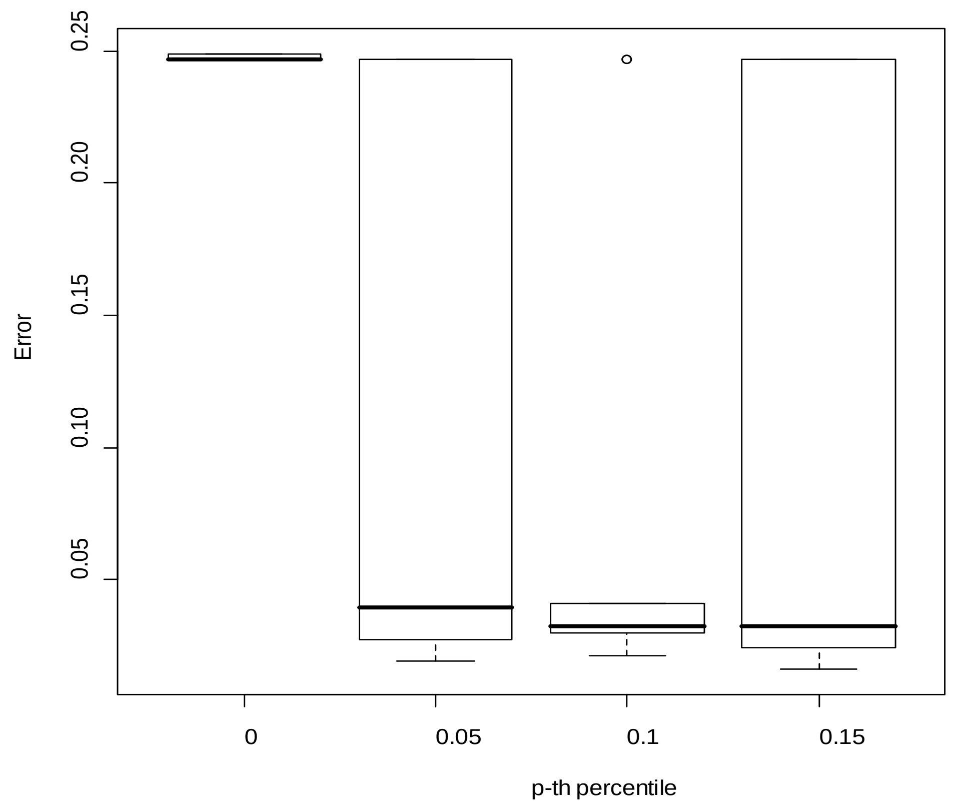


Figure 4.3. Boxplots of test error vs *p*-th percentile of margins for treatments 2, 6, 10, and 14 described in Table 4.10.

Table 4.12. Analysis of variance for simulation data set I with 10% noise level

```
Analysis of Variance for Error, using Adjusted SS for Tests

Source   DF    Seq SS    Adj SS    Adj MS      F      P
a         1  0.101758  0.101758  0.101758  39.83  0.000
b         1  0.057343  0.057343  0.057343  22.45  0.000
p         3  0.107108  0.107108  0.035703  13.98  0.000
a*b       1  0.100551  0.100551  0.100551  39.36  0.000
a*p       3  0.075896  0.075896  0.025299   9.90  0.000
b*p       3  0.015314  0.015314  0.005105   2.00  0.117
a*b*p     3  0.024597  0.024597  0.008199   3.21  0.025
Error   144  0.367866  0.367866  0.002555
Total   159  0.850431


S = 0.0505433   R-Sq = 56.74%   R-Sq(adj) = 52.24%
```

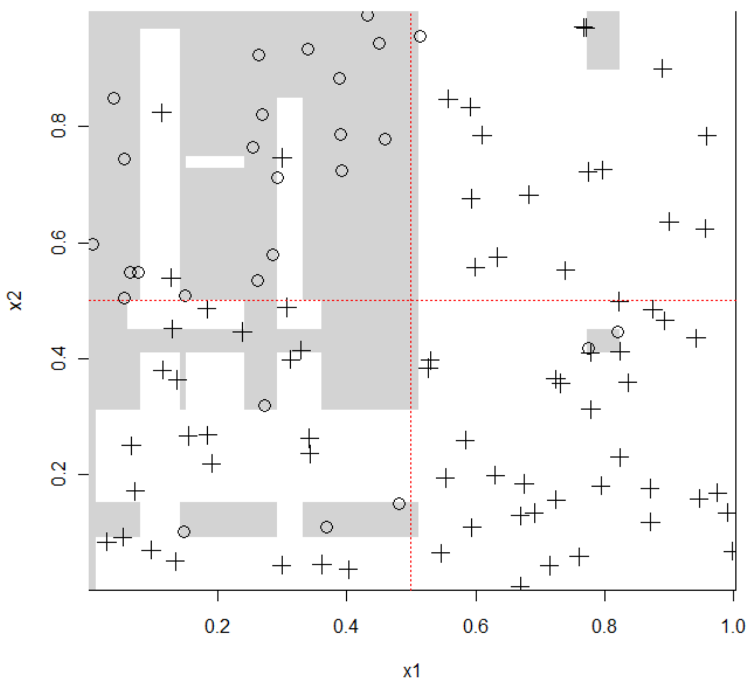
4. 1. 3. 1. Different options for Data Set I with 10% noise

To further examine the effect of *b*, we set up different values of *b* with a fixed value of *a* (0.1) and two values of *p* (0.00 and 0.10). For the data set including 10% noise with seven different values of *b* (0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, and 2.0), the value of *T* is related to the values of *b* in maximizing the minimum margin in Table 4.13. Small (large) *b* results in a model with large (small) *T*, number of stumps in a final model. We expect contaminated examples to have large negative margin values and to be difficult to correctly classify. If there is no trimming or ignoring of some difficult to classify examples, GA-Boost tries to classify all examples so that the small value of *b* encourages larger *T* and possibly leads to overfitting. The smaller *b*, the worse performance we see in Table 4.13 from maximizing the minimum margin. In the cases of maximizing the 10th percentile of margins (*p*=0.10), mostly three stumps are used for the final model except when *b* is greater than 1.0, which is very stable compared to the cases with *p*=0.0. The tuning parameter *b* doesn't affect the number of weak learners for a final best solution when *b* is less than 0.1. Because maximizing the 10th percentile of margins allows ignoring examples that have the 10% smallest margin values, we are classifying only normal examples so that only a few stumps are needed for this simple data set. From Table 4.13, final solutions maximizing the 10th percentile of margins include three or four stumps but have better general performance than final solutions with 4 to 14 stumps in the first treatment, which maximizes the minimum margin. Meanwhile, just one stump tree is the final model when there is a particularly large penalty for the number of weak learners, (*b* = 1.0 and 2.0), given a fixed *a* penalty for the number of negative margins (*a* = 0.1). Thus the value of *p* would affect the number of stumps if there are hard to classify examples in the data set.

Figure 4.4 shows the image plots for AdaBoost with *T* = 200 (above) and for GA-Boost maximizing the 10th percentile of margins (below) with *a*=0.1, *b* = 0.1 based on a data set with 10% noise. Both algorithms used two-terminal-node decision trees as weak learners. In these figures, the dark colored areas indicate a prediction of *Y*=-1 and the white area indicates a prediction of *Y* = +1. AdaBoost correctly classified all examples using 200 stumps and GA-Boost maximizing the 10th percentile of margins misclassified 10 examples which are the outliers, but correctly classified the remaining observations, by using three stumps. The outliers are ignored by the GA-Boost solution and are misclassified, as they should be in an outlier-resistant solution. GA-Boost provides a simpler solution that looks more like the original population of rectangular data I and is more resistant to outliers than AdaBoost.

Table 4.13 Test errors (standard errors) and number of weak learners in a final model of each treatment for data set I with 10% noise and *a*=0.1.

10% noise		***p***			
		0.00		0.10	
		Test Error (*St. Error*)	*T*	*Test Error* (*St. Error*)	*T*
b	0.0001	0.0697 (0.0245)	9, 10, 18	0.0257 (0.0026)	3, 4, 4
	0.001	0.0443 (0.0133)	6, 7, 14	0.0163 (0.0070)	3, 3, 3
	0.01	0.0357 (0.0080)	5, 8, 10	0.0283 (0.0039)	3, 3, 3
	0.1	0.0337 (0.0062)	4, 4, 5	0.0233 (0.0088)	3, 3, 3
	0.5	0.2110 (0.0350)	2, 2, 3	0.0270 (0.0035)	3, 3, 3
	1	0.2430 (0.0000)	1, 1, 1	0.2430 (0.0000)	1, 1, 1
	2	0.2430 (0.0000)	1, 1, 1	0.2430 (0.0000)	1, 1, 1



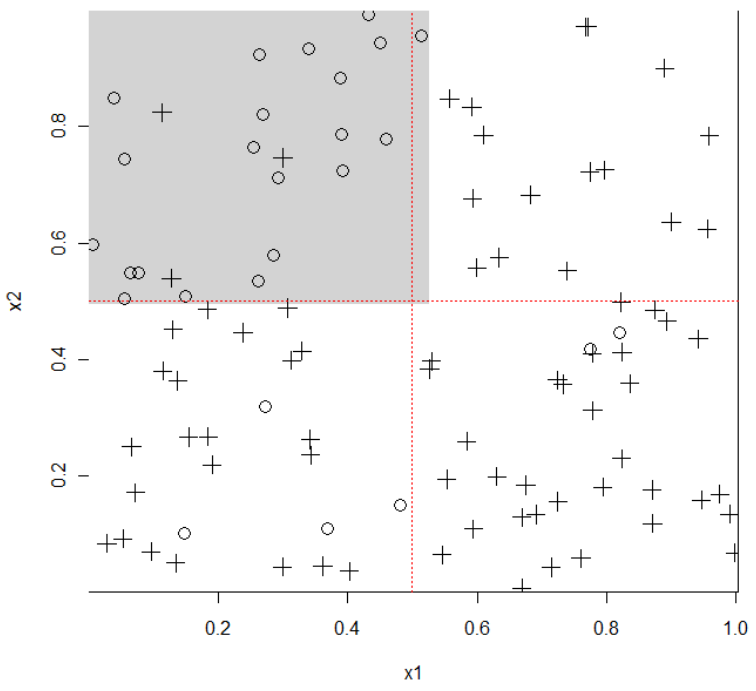


Figure 4.4. Image plots for AdaBoost with *T* = 200 (above) and for GA-Boost with maximizing the 10th percentile of margins (below) with *a*=0.1, *b* = 0.1 based on a data set with 10% noise.

Table 4.14. Test errors for AdaBoost with different iterations (*T*=50, 100, 200, 500, and 1,000*)* and GA-Boost with 16 treatments based on first simulation data with four different noise levels (0%, 1%, 5%, and 10%). Bold indicates win between AdaBoost and GA-Boost along with four different noise levels. The numbers in parenthese are the average number of weak learners (*T*).

Noise levels		0%	1%	5%	10%
AdaBoost	T=50	**0.011**	**0.014**	0.042	0.149
	T=100	**0.011**	**0.014**	0.042	0.097
	T=200	**0.012**	**0.014**	0.049	0.139
	T=500	**0.012**	**0.014**	0.057	0.146
	T=1000	**0.012**	**0.014**	0.057	0.179
GA-Boost	Treatment 1	0.021 (3.2)	0.019 (3.0)	**0.026 (4.2)**	**0.085 (3.8)**
	2	0.018 (3.0)	0.017 (3.0)	0.248 (2.0)	0.248 (2.0)
	3	0.020 (3.2)	0.022 (3.5)	**0.022 (4.4)**	**0.051 (5.1)**
	4	0.019 (3.0)	0.017 (3.0)	0.046 (3.4)	**0.030 (3.1)**
	5	0.022 (3.0)	0.019 (3.0)	**0.018 (3.0)**	**0.027 (3.1)**
	6	0.021 (3.0)	0.020 (3.0)	0.043 (2.9)	**0.096 (2.7)**
	7	0.018 (3.0)	0.018 (3.0)	**0.022 (3.1)**	**0.037 (3.6)**
	8	0.018 (3.0)	0.017 (3.0)	**0.023 (3.0)**	**0.031 (3.2)**
	9	0.019 (3.0)	0.020 (3.0)	**0.016 (3.0)**	**0.026 (3.0)**
	10	0.019 (3.0)	0.019 (3.0)	0.048 (3.0)	**0.075 (2.8)**
	11	0.016 (3.1)	0.018 (3.2)	**0.022 (3.2)**	**0.034 (3.8)**
	12	0.018 (3.0)	0.019 (3.0)	**0.019 (3.0)**	**0.024 (3.1)**
	13	0.015 (3.0)	0.021 (3.1)	**0.017 (3.0)**	**0.023 (3.0)**
	14	0.018 (3.0)	0.020 (3.0)	**0.021 (3.0)**	**0.093 (2.7)**
	15	0.022 (3.5)	0.022 (3.1)	**0.027 (3.6)**	**0.036 (3.7)**
	16	0.020 (3.0)	0.022 (3.0)	**0.017 (3.0)**	**0.026 (3.0)**

4. 1. 4. Comparisons I

Now we want to compare GA-Boost for the 16 treatments shown in Table 4.1 to AdaBoost with different numbers of iterations (*T*= 50, 100, 200, 500, and 1,000) based on the simple data set I with different noise levels, 0%, 1%, 5%, and 10%, and to evaluate the performance using a test set (n=10,000).

On the 0% noise data, AdaBoost has excellent performance as seen in Table 4.14, that is, 1.1% to 1.2% general performance error rate. For the 1% noise data set, AdaBoost did better than any results from GA-Boost. As the noise level increases, test errors from both algorithms increase, however in most cases GA-Boost is better than AdaBoost. Bold numbers in Table 4.14 indicate better performance.

Test error of AdaBoost is 0.042 on the first two iterations and after that it increases to 0.057 for the data set with 5% noise. For 10% noise data, the test error of AdaBoost goes down to 0.097 and then up as the number of iterations increases. The smallest test error among those is 0.097, which is not better than any test errors from GABoost except treatment 2, which maximizes the minimum margin with the values of a and b at 0.1 and 0.7, respectively. We verify that AdaBoost is overfitting and GA-Boost has better performance when there is a large proportion of outliers in a data set.

4. 2 Simulation data II

The differences between AdaBoost and GA-Boost are apparent in more complex simulation studies and real experiments as well. For the second simulation, we used a concentric circles structure with d = 2-dimensional artificial data. Two predictor values are randomly generated from a uniform (-1, 1) distribution with sample size n=300. The response value is generated to be an additive function of the two predictor variables from concentric circles. We have a response variable, Y = {-1, +1}

$$Y = sign\left(k \cdot \pi \cdot \sqrt{\sin(r^2)}\right) \tag{4.3}$$

where $r^2 = \sum_{j=1}^{p} x_j^2$.

Response values are equal to -1 if y_i is less than the median of r_i^2 and +1 otherwise. *Y* = -1 are marked by a circle and *Y* = +1 marked by a cross in Figure 4.5. Therefore our data set has the same number of observations in each class, 150. 300 observations are drawn for constructing the model and 5,000 observations are generated for test. The average test errors over 10 runs are reported as general performance. To build a final model, decision trees with one and three split points and oblique stump trees are employed as weak learners.

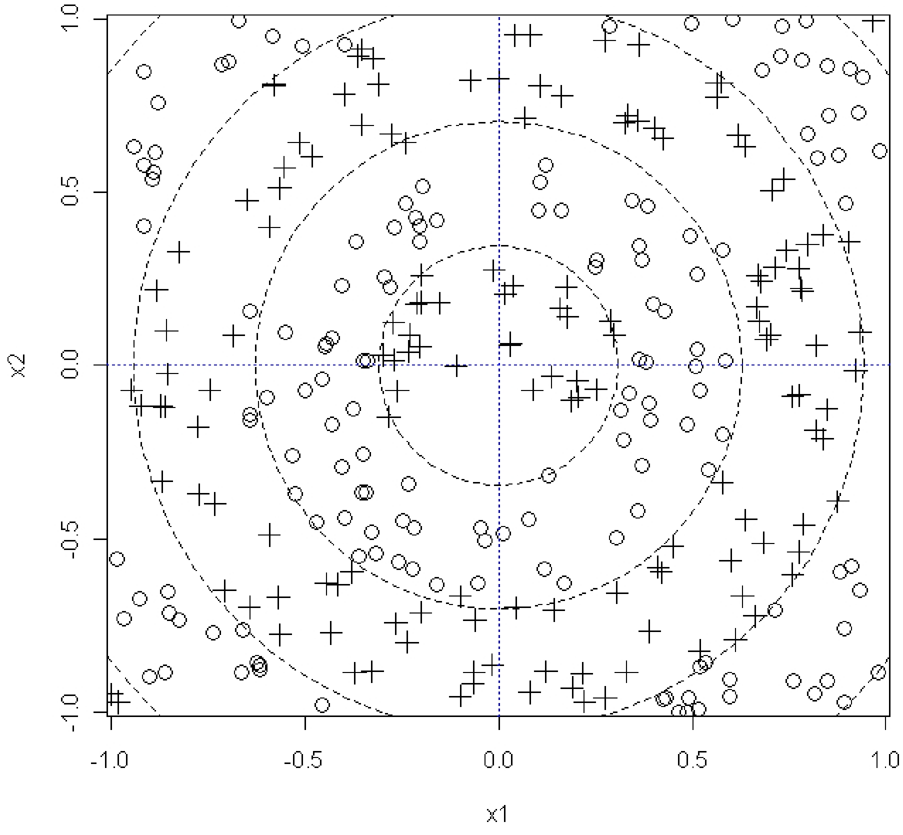


Figure 4.5. Concentric circle data structure

To examine the performance of GA-Boost, we set the values of *a* and *b* to 0.1 or 0.7 with 4 different values of *p*, which are 0.00, 0.05, 0.10, and 0.15. Therefore this designed experiment on three factors leads to 16 treatments for GA-Boost with 2,000 generations like the first simulation study except for the number of generations. The number of generations, 2,000, is not enough to get a full convergence value of fitness in GA based on this circle data set, but this simulation number was used to save computer time. By these 16 treatments we can test the main effects and the interactions of *a*, *b*, and *p* as well as which values of tuning parameters and *p* are the best for this simulation data.

Table 4.15. Test errors (standard errors) with averaged *T* of each treatment for data set II.

Treatment	1	2	3	4	5	6	7	8
p	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.05
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
Test Error	0.3561	0.4587	0.3547	0.3960	0.4009	0.4594	0.3674	0.3858
St. Error	0.0062	0.0028	0.0089	0.0102	0.0106	0.0028	0.0145	0.0133
average T	7.1	2.3	9.5	6.3	5.5	2.5	9.4	6.5

	9	10	11	12	13	14	15	16
p	0.10	0.10	0.10	0.10	0.15	0.15	0.15	0.15
a	0.1	0.1	0.7	0.7	0.1	0.1	0.7	0.7
b	0.1	0.7	0.1	0.7	0.1	0.7	0.1	0.7
Test Error	0.3960	0.4557	0.3629	0.3624	0.3889	0.4575	0.3553	0.3993
St. Error	0.0129	0.0034	0.0102	0.0113	0.0072	0.0045	0.0074	0.0086
average T	6.6	2.7	8.4	5.8	6.5	2.4	8.8	5.6

Table 4.16. ANOVA table for test error versus 16 treatments from simulation data set II with a circle data set.

One-way ANOVA: Error versus Treatment

```
Source      DF        SS        MS      F      P
Treatment   15  0.235323  0.015688  18.60  0.000
Error      144  0.121453  0.000843
Total      159  0.356777

S = 0.02904   R-Sq = 65.96%   R-Sq(adj) = 62.41%


                             Individual 95% CIs For Mean Based on
                             Pooled StDev
Level   N     Mean    StDev  ------+---------+---------+---------+---
 1     10  0.35558  0.01910  (----*---)
 2     10  0.45896  0.00877                            (----*---)
 3     10  0.35468  0.02821  (----*---)
 4     10  0.39608  0.03247            (----*----)
 5     10  0.40092  0.03360              (---*----)
 6     10  0.45942  0.00873                            (----*---)
 7     10  0.36738  0.04583     (----*---)
 8     10  0.38584  0.04206          (---*----)
 9     10  0.39602  0.04085            (----*----)
10     10  0.45566  0.01092                           (----*---)
11     10  0.36288  0.03226    (----*---)
12     10  0.36236  0.03586    (----*---)
13     10  0.38890  0.02287           (---*----)
14     10  0.45748  0.01424                            (---*----)
15     10  0.35530  0.02344  (----*---)
16     10  0.39932  0.02717             (----*---)
                             ------+---------+---------+---------+---
                                 0.360     0.400     0.440     0.480

Pooled StDev = 0.02904
```

The circle data is more complex for correctly classifying all training data than the first simulation data sets. From the table 4.15, 4.16 and Figure 4.6, we see that treatments 2, 6, 10 and 14 which have 0.1 and 0.7 for *a* and *b* have worse performance with over 0.45 misclassification rate. The final solutions from those treatments have two or three classifiers, which are not enough to classify this complicated data set well. We observe that the best treatments are 3, 7, 11, and 15, which averaged 8, 9, or 10 weak learners in a final solution. They performed significantly better than other solutions with 2 or 3 classifiers but not significantly different from all other treatments with solutions having 5, 6, and 7 classifiers. From Figure 4.7 we can compare the number of classifiers in the final solutions and the number of weak learners chosen for this data set. Also, it can be inferred which combinations of *a*, *b*, and *p* have the best performance. For example, if we need 10 weak learners for a final model, we can verify treatment 3 is best to produce 10 weak learners to classify examples. Therefore in order to classify a complex data set we need the final solution to include more weak learners. To do that, we put a small value of the tuning constant *b* for the number of weak learners in the fitness function. But four different levels of *p* do not significantly affect the results in ANOVA Table 4.17.

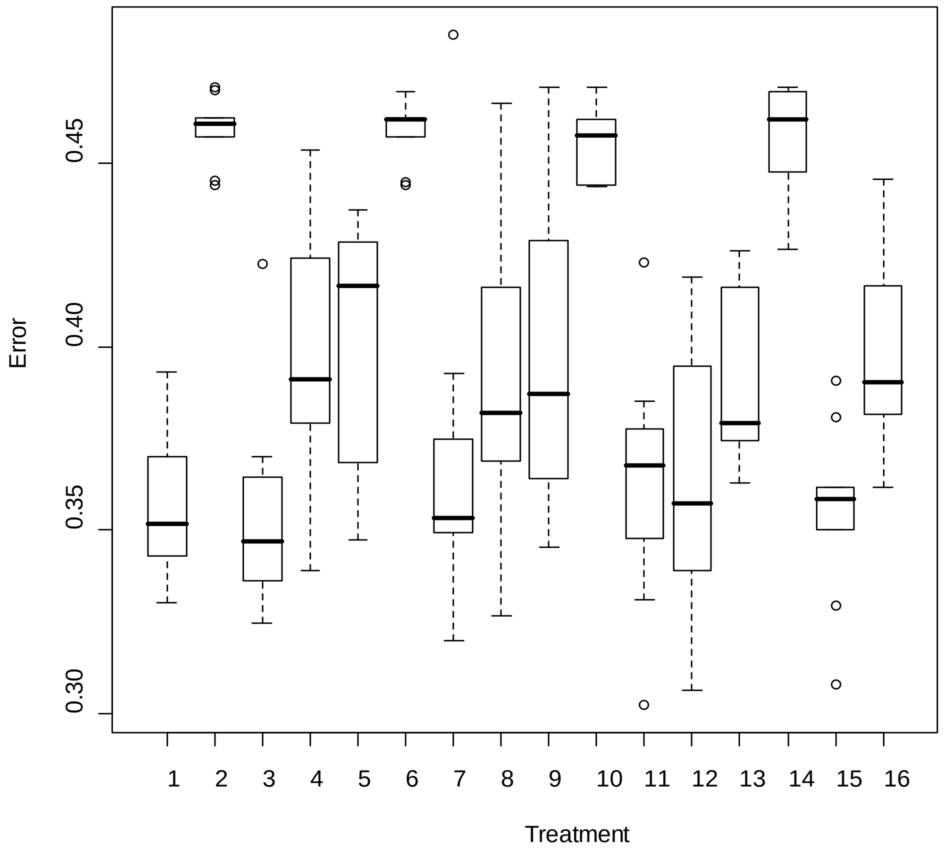


Figure 4.6 Boxplots of test error for 16 treatments described in table 4.15.

We observe that the main effects of the tuning parameters, *a* and *b*, have a strong influence on the results but the effects of *p* are insignificant from Table 4.17. The interactions of (*a*, *b*) and (*b*, *p*) are significant from the analysis of variance test. Through the boxplots of (*a*, *b*) in Figure 4.8 we see that the combinations of *a* and *b* are significantly different and the third combination of (*a*, *b*) = (0.7, 0.1) should be best.

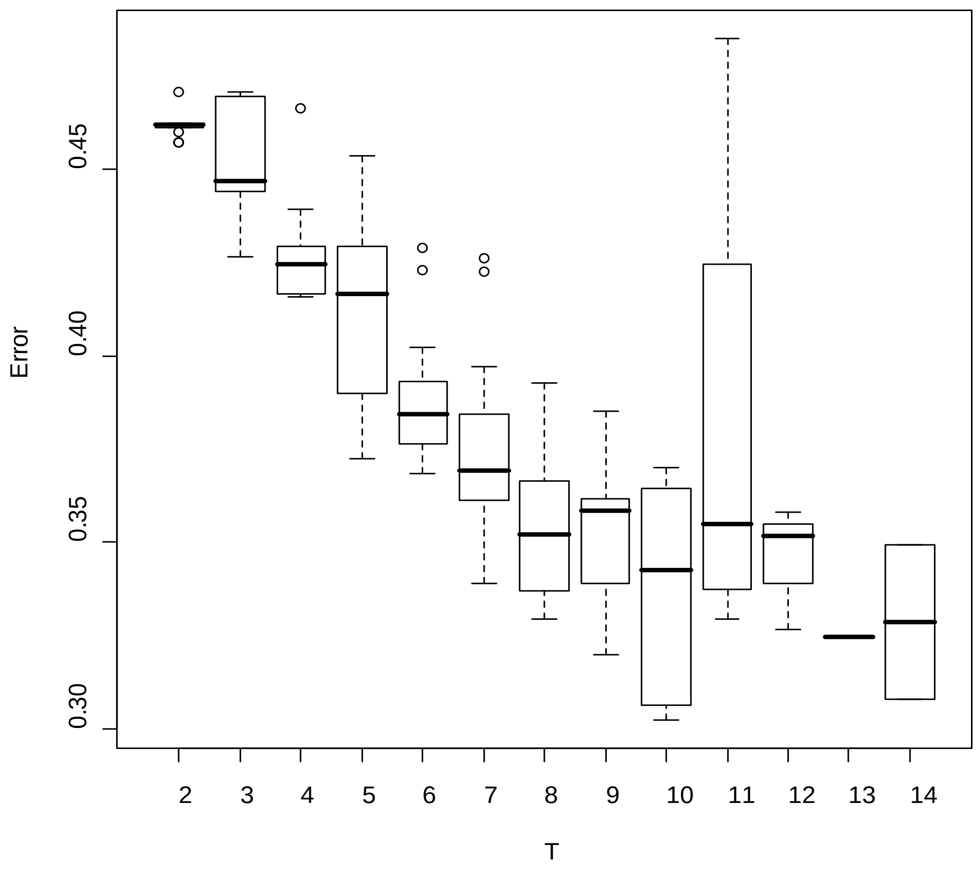


Figure 4.7. Boxplots of test error for the number of weak learners in a final solution.

Table 4.17. ANOVA tables for test error versus *p*, *a*, and *b* from the concentric circle data set.

```
Analysis of Variance for Error, using Adjusted SS for Tests

Source   DF    Seq SS    Adj SS    Adj MS       F      P
a         1  0.094624  0.094624  0.094624  112.19  0.000
b         1  0.096757  0.096757  0.096757  114.72  0.000
p         3  0.003637  0.003637  0.001212    1.44  0.234
a*b       1  0.021795  0.021795  0.021795   25.84  0.000
a*p       3  0.005250  0.005250  0.001750    2.07  0.106
b*p       3  0.010888  0.010888  0.003629    4.30  0.006
a*b*p     3  0.002373  0.002373  0.000791    0.94  0.424
Error   144  0.121453  0.121453  0.000843
Total   159  0.356777


S = 0.0290418   R-Sq = 65.96%   R-Sq(adj) = 62.41%
```

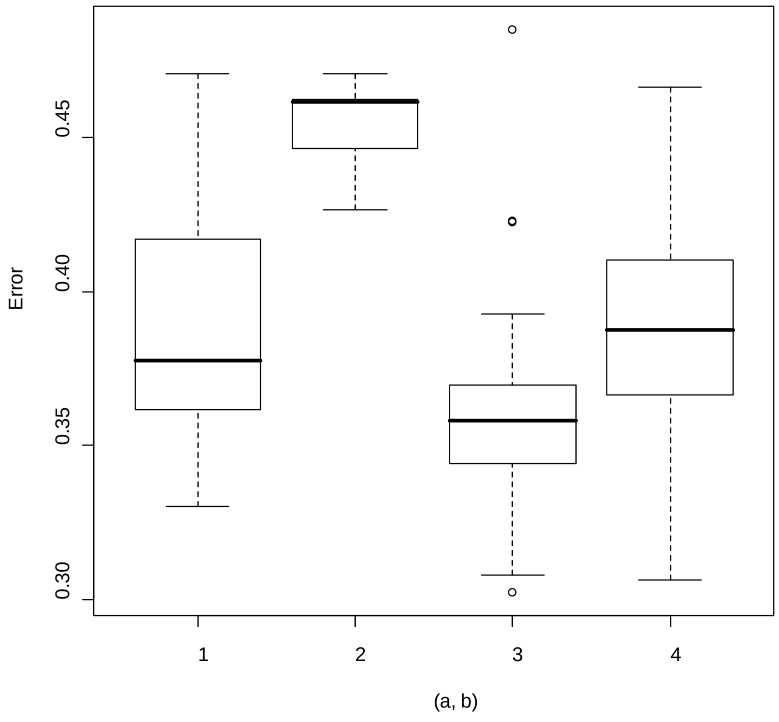


Figure 4.8. Boxplots of test error for interaction of *a* and *b* in fitness function based on the concentric circle data set. 1: (*a*, *b*) = (0.1, 0.1), 2: (*a*, *b*) = (0.1, 0.7), 3: (*a*, *b*) = (0.7, 0.1), 4: (*a*, *b*) = (0.7, 0.7).

Table 4.18. Test errors (standard errors) and number of weak learners in a final model of each treatment for circle data set.

Circle Data		*p*			
		0.00		0.10	
		Test Error (*St. Error*)	*T*	*Test Error* (*St. Error*)	*T*
b	0.0001	0.3289 (0.0264)	23, 23, 19	0.3218 (0.0155)	17, 28, 19
	0.001	0.3203 (0.0056)	17, 22, 18	0.3861 (0.0294)	15, 23, 11
	0.01	0.3313 (0.0136)	11, 16, 11	0.3809 (0.0189)	14, 9, 4
	0.1	0.3459 (0.0132)	7, 10, 10	0.4049 (0.0069)	6, 6, 6
	0.5	0.4240 (0.0152)	2, 5, 5	0.4404 (0.0012)	3, 3, 3
	1	0.4617 (0.0048)	1, 1, 1	0.4473 (0.0100)	1, 3, 3
	2	0.4612 (0.0046)	1, 1, 1	0.4571 (0.0051)	1, 1, 1

For the circle data set with different values of *b* and *p* from the Table 4.18 above, we can see how *T* is related to the values of *b*. Small *b* results in a model with large *T*, the number of stumps in the final model. The circle data set has a complex data structure, so we may need larger *T* to correctly classify them. To know the effect of *b*, we set seven different values of *b* with a fixed value of *a* (= 0.1) and two values of *p* (= 0.00 and 0.10) same as in Table 4.13. Even though the parameter *a* has a significant effect for this data set, we would like to consider this issue in the future. From Table 4.13 based on first simulation data set with 10% noise level, the smaller *b*, the worse performance we have for maximizing the minimum margin because more weak learners can lead to overfitting for the original population, which can be perfectly classified by three weak learners. Therefore if there is no trimming or ignoring of difficult to classify examples, GA-Boost tries to correctly classify all examples so that the small value of *b* can produce the large number of *T* and possibly bring out overfitting. Since the circle data is not contaminated, the values of *p* should not affect outcomes much. A small value of *b* produces large *T* with better performance than for a large *b*. If *b* is relatively large, like greater than 0.5 with a fixed *a* (0.1), it doesn't affect the number of weak learners for the final best solution. Therefore for this data set, regardless of maximizing the *p*-th percentile of margins, we need to use a small value of *b* to generate more weak learners.

4.3. Comparisons II

Table 4.19 compares GA-Boost with 16 treatments to AdaBoost with different numbers of iterations, (*T* = 50, 100, 200, 500, 1000, 2000, 5000, and 10000) based on the circle data set

Table 4.19. Test errors and number of stump trees as weak learners from AdaBoost and GA-Boost for 16 treatments based on circle data set.

AdaBoost		GA-Boost					
T	*Test Error*	Treatment	*T*	*Test Error*	Treatment	*T*	*Test Error*
50	0.4298	1	7.1	0.3561	9	6.6	0.3960
100	0.4136	2	2.3	0.4587	10	2.7	0.4557
200	0.4118	3	9.5	0.3547	11	8.4	0.3629
500	0.4244	4	6.3	0.3960	12	5.8	0.3624
1000	0.4064	5	5.5	0.4009	13	6.5	0.3889
2000	0.3834	6	2.5	0.4594	14	2.4	0.4575
5000	0.3812	7	9.4	0.3674	15	8.8	0.3553
10000	0.3752	8	6.5	0.3858	16	5.6	0.3993

and evaluates the performance using a test set, (n=5,000). 300 observations were used for constructing the model and stump trees were utilized as the weak learner for both algorithms. For AdaBoost, the test error roughly decreases from 0.4298 to 0.3752 as the number of iterations increases. To get the best performance of AdaBoost with stumps, it seems we should use as many stumps as we can. GA-Boost on the 16 treatments, on the other hand, achieves better interpretability and accuracy with less than 10 weak learners. GA-Boost evolved solutions for 2,000 generations on the training set to get the best solution over 10 runs for each treatment. The average value of T for GA-Boost over 10 runs is shown in the table. The values of T for each treatment from three different weak learners are very similar. Treatments 2, 6, 10, and 14 have less than 3 stumps in the final model due to the large penalty on the number of weak learners in the fitness function and have worse performance than other treatments. Treatments 1, 3, 7, 11, and 15, which have more than 7 stumps, outperformed AdaBoost on 8 different iterations. These models from GA-Boost have a significant interpretability advantage over AdaBoost.

Table 4.20 provides a summary of GA-Boost for decision trees with one and three split points, and oblique stump trees based on the circle data set. Decision trees with three split points are randomly produced and evolved into one of three different types of trees, which have either two terminal nodes with one split point (stump), or three terminal nodes with two split points, or four terminal nodes with three split points. A stump tree can split data set vertically or horizontally but an oblique stump tree can split obliquely as well as vertically or horizontally in the predictor space. So a two terminal node, oblique stump tree is similar to a stump tree but a more complicated and general model than a stump. Obviously GA-Boost for 3 split points tends to produce a better result than for stumps and oblique stump trees, and GA-Boost for oblique trees achieves better performance than stumps. Table 4.21 presents the win-tie-loss for all pairwise combinations of three weak learners. When we compare decision trees with one split point and oblique stump trees to decision trees with 3 split points, we see that decision trees with 3 split points are superior to the others (16-0-0, 13-0-3) but in treatment 1, 2, and 7, decision trees with 3 split points are inferior to oblique stump trees. In comparing decision trees with one split point to oblique stump trees, we can see oblique stump trees beat decision trees with one split point in all treatments (16-0-0). From this result, we conclude that the best weak learner with the circle data is a decision tree with 3 split points. Decision trees with 1 split point are not good choices for weak learners. We can also verify these results in Figure 4.9, which shows the plot of test error for 16 treatments along with three different weak learners

Treatments 2, 6, 10, and 14, which have a large value of b (= 0.7), have worse outcomes with 2 or 3 weak learners. The best performances are achieved with a large T in treatments 3, 7, 11, and 15 for all three different weak learners. This clearly indicates that more weak learners and more complicated weak learners are needed for a difficult to classify data set such as the circle data. The first four treatments are based on maximizing the minimum margin and the next four are maximizing the 5th percentile of the margins, and then the 10th and 15th percentiles of margins for the remaining treatments. As for stumps, four different levels of p seem not to be significantly different from the results for decision trees with 3 split points and oblique trees because we didn't find any differences among the different levels of p in Table 4.20. However we observed that the tuning parameter b has a strong influence on the results. The value of b = 0.1 produces better results than b = 0.7 and the combinations of (a, b) = (0.7, 0.1) would be selected for the circle data set among 16 treatments regardless of the four different values of p (0.00, 0.05, 0.10, and 0.15).

Table 4.20. Test errors (standard errors) of each treatment for decision tree with one and three split points, and oblique stump trees based on the circle data set

Treatment	DT with 1 split point		DT with 3 split points		Oblique Stump Tree	
	Test Error (*S. E.*)	*T*	*Test Error* (*S. E.*)	*T*	*Test Error* (*S. E.*)	*T*
1	0.3561 (0.0062)	7.1	0.3458 (0.0115)	7.3	0.3267 (0.0091)	8.3
2	0.4587 (0.0028)	2.3	0.4233 (0.0092)	2.7	0.4198 (0.0066)	3.1
3	0.3547 (0.0089)	9.5	0.2964 (0.0083)	10.1	0.3346 (0.0089)	10.3
4	0.3960 (0.0102)	6.3	0.3218 (0.0065)	7.5	0.3688 (0.0158)	7.9
5	0.4009 (0.0106)	5.5	0.3322 (0.0100)	7.6	0.3530 (0.0137)	7.6
6	0.4594 (0.0028)	2.5	0.3991 (0.0106)	3.3	0.4268 (0.0054)	3.0
7	0.3674 (0.0145)	9.4	0.3409 (0.0078)	8.7	0.3309 (0.0115)	9.4
8	0.3858 (0.0133)	6.5	0.3354 (0.0091)	6.8	0.3412 (0.0082)	8.6
9	0.3960 (0.0129)	6.6	0.3368 (0.0112)	7.4	0.3748 (0.0104)	7.0
10	0.4557 (0.0034)	2.7	0.4022 (0.0092)	3.3	0.4228 (0.0122)	3.2
11	0.3629 (0.0102)	8.4	0.3162 (0.0061)	8.7	0.3218 (0.0105)	10.4
12	0.3624 (0.0114)	5.8	0.3319 (0.0116)	7.2	0.3425 (0.0106)	7.6
13	0.3889 (0.0072)	6.5	0.3231 (0.0100)	7.3	0.3596 (0.0074)	6.9
14	0.4575 (0.0045)	2.4	0.4052 (0.0075)	3.0	0.4234 (0.0106)	3.4
15	0.3553 (0.0074)	8.8	0.3162 (0.0098)	9.5	0.3440 (0.0203)	8.7
16	0.3993 (0.0086)	5.6	0.3319 (0.0130)	8.1	0.3572 (0.0108)	7.4

Simulation data II

	DT with 1 split point	Oblique Stump Tree
Oblique Stump Tree	16-0-0	
DT with 3 split points	16-0-0	13-0-3

Table 4.21. Each cell contains the number of wins, ties, and losses between the weak learner in that row and the weak learner in that column

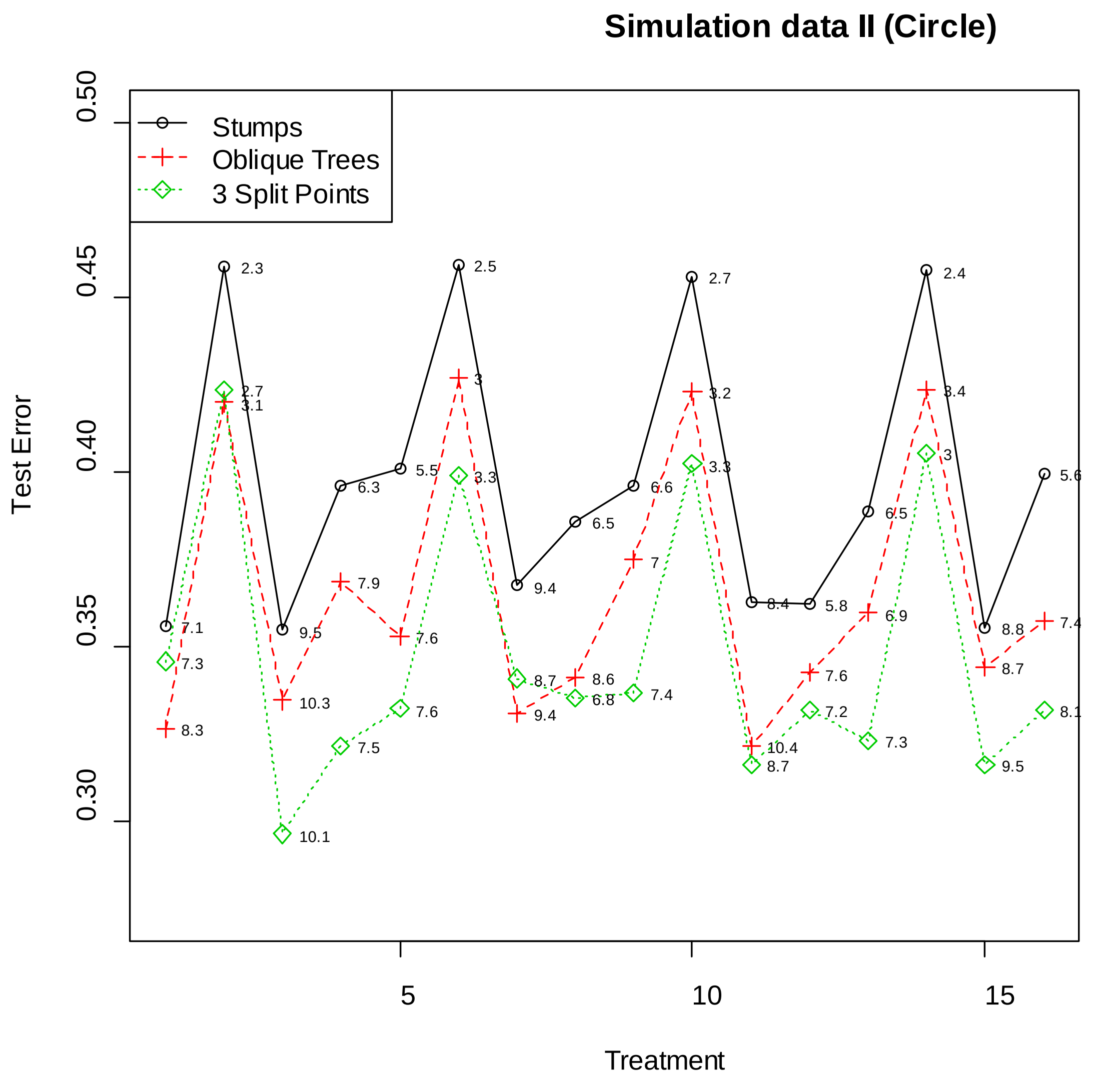


Figure 4.9. The plot of test errors for 16 treatments with three different weak learners. Labeling points are the averaged number of weak learners out of 10 runs.

4.4. Application of GA-Boost to Real World Data

In this section, comparisons based on real data are presented. We consider three real data sets from the UCI Machine Learning Repository: the kyphosis data, the glaucoma data, and the Wisconsin Diagnostic Breast Cancer (WDBC) data. The kyphosis data consists of three predictor variables with n=81 observations, of which 64 have -1 as their response value and 17 are +1. The glaucoma data having 62 predictors with n=196 observations is a more complex and larger data set than the kyphosis data. Class distribution is exactly half (-1) and half (+1) of data. Many studies of methods for a classification problem have used Wisconsin Diagnostic Breast Cancer (WDBC) data. This data set has 30 predictor variables with n=569 observations. 357 of n=569 have -1 response values and the remaining 212 have +1 response values.

For comparison, 10-fold cross-validation was used to estimate the test error. First we take 10 subsamples randomly partitioned from the original data. One subsample, a tenth of the original data, was used for testing the model. The remaining nine subsamples were used as the training set. This procedure was repeated 10 times, rotating the role of the test set, and the results were averaged to produce the test error. 16 treatments of the tuning parameters, a and b with the value of p to maximize, were tested for GA-Boost, which evolved solutions for 2,000 generations on the training set to get the best solution over the 10 runs for each treatment. The more generations in GA, the better fitness value we have. The number of generations, 2,000, is not sufficient to get a convergence of the fitness value for large data sets, especially for the Wisconsin Diagnostic Breast Cancer. Because it is too time consuming using a current computer program, we used 2,000 generations to construct the best solution for all real data sets. In the initial random solution, we set the number of random weak classifiers and their weights to five. The results of running the three different base classifiers (decision trees with two terminal nodes and three split points, and an oblique stump tree) are shown for GA-Boost in Tables 4.23, 4.25, and 4.27. The results from AdaBoost based on stump trees as a base classifier are shown in Table 4.25.

4.4.1. Experimental results

Table 4.22 presents the results for AdaBoost with different numbers of iterations (*T* = 50, 100, 200, 500, 1000, 2000, and 10,000) for the kyphosis, glaucoma, and WDBC data sets. Decision trees with one split point (stump) were utilized as a weak learner. In Tables 4.23, 4.25, and 4.27 we compare 16 treatments for GA-Boost with three different weak learners based on the three real data sets. A comparison among the average generalization errors with standard error is shown along with the average number of weak learners out of the 10 runs. For a fair comparison, a comparison between the test errors generated by AdaBoost and GA-Boost is for one split decision trees. Test errors were estimated by 10-fold cross-validation for both algorithms.

For the kyphosis data, the test errors from AdaBoost are 0.1778 to 0.2333 according to the number of iterations in Table 4.21. We can see that the performance is best at *T*=50 and the worst at *T*=2,000, and 10,000. AdaBoost with *T*=50 is also the best performance and with *T*=10,000 is worst for the glaucoma data. AdaBoost has worst performance at *T*=10,000 for both data sets. It is not supported that using a large number of weak learners produces better results. Based on the WDBC data, AdaBoost has the best performance at *T*=500 and the worst at *T*=50.

Table 4.22. 10-fold cross validation (standard errors) from AdaBoost for decision trees with one split point (Stump) based on the kyphosis, glaucoma, and WDBC data sets.

T		50	100	200	500	1,000	2,000	10,000
Kyphosis	*Test Error*	0.1778	0.2111	0.2000	0.1889	0.1889	0.2333	0.2333
	St. Error	0.0412	0.0482	0.0399	0.0470	0.0237	0.0452	0.0509
Glaucoma	*Test Error*	0.2125	0.2250	0.2250	0.2437	0.2188	0.2062	0.2500
	St. Error	0.0397	0.0408	0.0191	0.0421	0.0419	0.0229	0.0348
WDBC	*Test Error*	0.0411	0.0268	0.0286	0.0179	0.0232	0.0250	0.0232
	St. Error	0.0100	0.0089	0.0089	0.0071	0.0080	0.0040	0.0053

Meanwhile, Tables 4.23, 4.25, and 4.27 provide a summary of GA-Boost on the 16 treatments with three different weak learners. Here we consider the one split point decision tree to compare with AdaBoost performance. 16 treatments resulted in better interpretability with fewer than 8 weak learners on average in a final solution than the final model from AdaBoost for three data sets. For the kyphosis data, 8 of 16 treatments have less than 3 stumps. Treatment 3 has the largest average number of weak learners (*T*=6.5) but not a very good result in terms of test set error. GA-Boost on treatment 8 with 3.1 average weak learners did the best compared to AdaBoost. Treatments 4, 11, 12, and 13 have better performance than AdaBoost with *T*=2000 and 10,000. The remaining treatments are inferior to AdaBoost. When we compare for the glaucoma data, GA-Boost on treatments 5, 8, 10, 12, 14, 15, and 16 is superior to AdaBoost. So for the glaucoma data, generalization performances by GA-Boost except treatments 4, 7, and 9 are better than that of the combined classifiers by AdaBoost with 7 different iterations. Finally for the WDBC data, we compare GA-Boost to AdaBoost based on decision trees with one split point. Results show that AdaBoost had better performance than GA-Boost on all 16 treatments for WDBC data. Because this data is a relatively large set with *n*=569 and *p*=30, 2000 generations may not be enough to construct a good solution. But GA-Boost has a strong advantage of interpretation of its final models over AdaBoost. Also, the WDBC data has low data complexity and contamination. From the results of GA-Boost and AdaBoost based on simulation data I sets with 0% and 1% noise levels, in a case of low contaminated data, AdaBoost performs very well and better than GA-Boost.

From these comparisons, we can conclude that GA-Boost is superior and inferior to AdaBoost for the glaucoma and the WDBC data, respectively. For the kyphosis data, they are somewhat equally matched. However, the final models with a few weak learners by GA-Boost have a significant interpretability advantage over those by AdaBoost.

Now we consider more complex weak learners than stump trees. Tables 4.23, 4.25, and 4.27 present the performance of GA-Boost on 16 treatments with different weak learners (stump trees and decision trees with three split points, and oblique stump trees) based on the three different real world data sets (kyphosis, glaucoma, and WDBC). From these tables, we can see how GA-Boost works with different weak learners. We expect that the more complex weak learners with more than two splits and the linear combination splitting rule will work better than stump trees, but computation is more expensive.

Tables 4.24, 4.26, and 4.28 present the win-tie-loss for all pairwise combinations of three weak learners for each real data set. For the kyphosis data in Table 4.24, we see that decision tree with 3 split points is a little better than the stump tree and oblique tree (9-1-6, 10-0-6), and that the stump tree wins in 7 treatments, loses in 8 and ties in only 1 to the oblique tree. In Table 4.26, based on the glaucoma data, we see that the decision tree with 3 split points and oblique tree are equally matched (8-0-8). Finally we compare for WDBC in Table 4.28. A stump tree has a slight advantage over a decision tree with 3 split points (10-1-5) and other comparisons are close contests. In the kyphosis case, somehow more complex classifiers gain advantage over less complex classifiers. In contrast, less complex classifiers have better performance.

We can also verify the results through Figures 4.10, 4.11, and 4.12, which show the plots of test errors for 16 treatments along with the three different weak learners. The solid line is from the decision tree with one split point, the dashed line is from the oblique tree, and the dotted line is from the decision tree with 3 split points based on three different real data sets. Labeling points are the average numbers of weak learners (*T*). Some points are outliers in the plots of test errors for 16 treatments. One of 10 runs tends to have a very bad performance. For example, one test error rate of the 10-fold cross validations is 1.0 which indicates misclassification of all test data on treatment 1 for the kyphosis data.

Like the result of simulation data II (Circle data), we expect that more complex classifiers will work better than less complex classifiers. But the results for the real data sets are different from what we expect. We believe there are some reasons why the results for the real data sets are different from our expectation. For future study we plan to increase the number of generations in the genetic algorithm to get better convergence of the fitness value, because we utilized a decision tree with three split points and an oblique stump tree which are more complex than a stump tree as a weak learner. We also intend to increase the number of cross validations. The big difference from the test of simulation data II (Circle data) is the test set for measuring generalization performance. The simulation data II used 300 observations for constructing the model and 5,000 observations for test.

Table 4.23. 10-fold cross validations (standard errors) with average *T* of each treatment for decision trees with one and three split points and oblique stump trees based on the kyphosis data. Test errors are estimated by 10-fold cross validation.

Treat	*p*	*a*	*b*	DT with one split point		DT with 3 split points		Oblique Stump Tree	
				Test Error (S.E.)	*T*	Test Error (S.E.)	*T*	Test Error (S.E.)	*T*
1	0.00	0.1	0.1	0.3708 (0.0769)	2.4	0.3083 (0.0622)	2.7	0.2208 (0.0393)	3.3
2	0.00	0.1	0.7	0.2472 (0.0457)	2.0	0.2208 (0.0473)	2.0	0.2667 (0.0474)	2.0
3	0.00	0.7	0.1	0.2819 (0.0539)	6.5	0.2847 (0.0532)	5.2	0.2222 (0.0407)	6.4
4	0.00	0.7	0.7	0.2111 (0.0461)	3.3	0.1986 (0.0467)	3.2	0.2583 (0.0464)	3.4
5	0.05	0.1	0.1	0.2347 (0.0473)	2.5	0.2097 (0.0373)	2.9	0.3083 (0.0593)	2.9
6	0.05	0.1	0.7	0.2847 (0.0422)	2.0	0.2069 (0.0531)	2.0	0.2194 (0.0453)	2.0
7	0.05	0.7	0.1	0.2472 (0.0418)	4.9	0.2472 (0.0324)	5.1	0.2333 (0.0272)	5.2
8	0.05	0.7	0.7	0.1750 (0.0382)	3.1	0.1861 (0.0429)	3.3	0.2208 (0.0631)	3.3
9	0.10	0.1	0.1	0.2458 (0.0483)	3.1	0.2361 (0.0398)	3.3	0.2458 (0.0405)	3.0
10	0.10	0.1	0.7	0.2597 (0.0437)	2.0	0.1944 (0.0426)	2.0	0.2333 (0.0272)	2.0
11	0.10	0.7	0.1	0.2208 (0.0292)	5.0	0.2472 (0.0457)	4.2	0.2222 (0.0249)	5.4
12	0.10	0.7	0.7	0.2125 (0.0561)	3.3	0.2875 (0.0855)	3.1	0.2333 (0.0700)	3.4
13	0.15	0.1	0.1	0.2083 (0.0516)	2.8	0.2472 (0.0324)	1.8	0.3833 (0.0802)	2.2
14	0.15	0.1	0.7	0.2847 (0.0422)	2.1	0.2208 (0.0435)	1.3	0.2444 (0.0333)	1.6
15	0.15	0.7	0.1	0.2583 (0.0464)	5.5	0.2972 (0.0429)	5.0	0.2222 (0.0583)	5.5
16	0.15	0.7	0.7	0.3208 (0.0856)	2.6	0.2125 (0.0325)	2.4	0.2722 (0.0367)	2.6

Table 4.24. Each cell contains the number of wins, ties, and losses between the weak learner in that row and the weak learner in that column for the kyphosis data.

	DT with 3 split points	Oblique Stump Tree
DT with one split point	6-1-9	7-1-8
DT with 3 split points		10-0-6

Table 4.25. Test errors (standard errors) with average *T* of each treatment for decision trees with one and three split points and oblique stump trees based on the glaucoma data. Test errors are estimated by 10-fold cross validation.

Treat	*p*	*a*	*b*	DT with one split point		DT with 3 split points		Oblique Stump Tree	
				Test Error (S.E.)	*T*	Test Error (S.E.)	*T*	Test Error (S.E.)	*T*
1	0.00	0.1	0.1	0.2213 (0.0392)	4.0	0.2437 (0.0207)	3.5	0.1787 (0.0225)	4.7
2	0.00	0.1	0.7	0.2125 (0.0256)	1.4	0.2213 (0.0231)	1.5	0.2137 (0.0299)	1.2
3	0.00	0.7	0.1	0.2400 (0.0393)	7.2	0.1938 (0.0252)	7.1	0.2100 (0.0452)	8.2
4	0.00	0.7	0.7	0.2625 (0.0344)	3.7	0.1688 (0.0269)	4.4	0.2725 (0.0829)	4.3
5	0.05	0.1	0.1	0.2000 (0.0325)	3.5	0.2013 (0.0227)	3.5	0.1963 (0.0387)	5.0
6	0.05	0.1	0.7	0.2125 (0.0208)	2.2	0.2562 (0.0178)	2.2	0.2313 (0.0328)	2.1
7	0.05	0.7	0.1	0.2787 (0.0858)	7.0	0.2137 (0.0260)	7.6	0.2162 (0.0327)	10.0
8	0.05	0.7	0.7	0.1938 (0.0293)	4.2	0.1675 (0.0264)	4.6	0.2112 (0.0291)	4.0
9	0.10	0.1	0.1	0.3337 (0.1090)	3.9	0.1875 (0.0251)	4.7	0.2225 (0.0294)	5.1
10	0.10	0.1	0.7	0.2075 (0.0253)	2.0	0.1888 (0.0277)	2.0	0.1988 (0.0236)	2.0
11	0.10	0.7	0.1	0.1925 (0.0358)	6.8	0.2462 (0.0377)	7.5	0.2000 (0.0279)	9.2
12	0.10	0.7	0.7	0.1950 (0.0353)	4.1	0.2125 (0.0315)	4.4	0.1850 (0.0279)	5.0
13	0.15	0.1	0.1	0.2200 (0.0403)	3.1	0.2562 (0.0144)	3.6	0.1762 (0.0349)	3.9
14	0.15	0.1	0.7	0.1625 (0.0215)	2.5	0.2075 (0.0274)	2.4	0.2662 (0.0749)	2.2
15	0.15	0.7	0.1	0.1975 (0.0313)	6.6	0.1862 (0.0360)	7.4	0.2013 (0.0319)	7.0
16	0.15	0.7	0.7	0.2012 (0.0282)	3.8	0.2050 (0.0283)	3.6	0.1938 (0.0274)	4.3

Table 4.26. Each cell contains the number of wins, ties, and losses between the weak learner in that row and the weak learner in that column for glaucoma data.

	DT with 3 split points	Oblique Stump Tree
DT with one split point	9-0-7	7-0-9
DT with 3 split points		8-0-8

Table 4.27. Test errors (standard errors) with average *T* of each treatment for decision trees with one and three split points and oblique stump trees based on the WDBC data. Test errors are estimated by 10-fold cross validation.

Treat	*p*	*a*	*b*	DT with one split point		DT with 3 split points		Oblique Stump Tree	
				Test Error (S.E.)	*T*	Test Error (S.E.)	*T*	Test Error (S.E.)	*T*
1	0.00	0.1	0.1	0.0509 (0.0084)	5.3	0.0510 (0.0085)	5.4	0.0598 (0.0091)	5.9
2	0.00	0.1	0.7	0.0650 (0.0117)	2.5	0.0703 (0.0083)	2.2	0.0685 (0.0092)	2.7
3	0.00	0.7	0.1	0.0527 (0.0104)	6.5	0.0721 (0.0097)	8.4	0.0632 (0.0102)	10.4
4	0.00	0.7	0.7	0.1018 (0.0633)	4.5	0.0527 (0.0058)	4.8	0.0527 (0.0078)	5.9
5	0.05	0.1	0.1	0.1107 (0.0515)	4.2	0.0563 (0.0117)	4.4	0.0545 (0.0092)	4.9
6	0.05	0.1	0.7	0.0492 (0.0094)	3.0	0.0544 (0.0124)	3.0	0.0615 (0.0099)	2.8
7	0.05	0.7	0.1	0.0544 (0.0112)	6.2	0.0651 (0.0084)	6.7	0.0615 (0.0105)	8.0
8	0.05	0.7	0.7	0.0474 (0.0094)	5.0	0.0667 (0.0093)	5.0	0.0457 (0.0092)	4.9
9	0.10	0.1	0.1	0.0633 (0.0126)	3.5	0.0581 (0.0106)	3.3	0.0650 (0.0094)	3.3
10	0.10	0.1	0.7	0.0878 (0.0175)	1.6	0.1002 (0.0117)	1.1	0.0615 (0.0095)	1.4
11	0.10	0.7	0.1	0.0632 (0.0176)	6.8	0.0669 (0.0134)	8.0	0.0598 (0.0096)	6.8
12	0.10	0.7	0.7	0.0615 (0.0054)	4.0	0.0738 (0.0173)	4.3	0.0544 (0.0115)	4.2
13	0.15	0.1	0.1	0.0528 (0.0091)	3.3	0.0528 (0.0083)	3.1	0.1388 (0.0806)	3.1
14	0.15	0.1	0.7	0.0914 (0.0127)	1.5	0.0755 (0.0131)	1.9	0.1071 (0.0351)	1.4
15	0.15	0.7	0.1	0.0509 (0.0129)	7.2	0.0422 (0.0070)	8.9	0.0491 (0.0125)	9.0
16	0.15	0.7	0.7	0.0528 (0.0087)	3.2	0.0581 (0.0112)	4.3	0.0633 (0.0105)	5.1

Table 4.28. Each cell contains the number of wins, ties, and losses between the weak learner in that row and the weak learner in that column for WDBC data.

	DT with 3 split points	Oblique Stump Tree
DT with one split point	10-1-5	9-0-7
DT with 3 split points		7-1-8

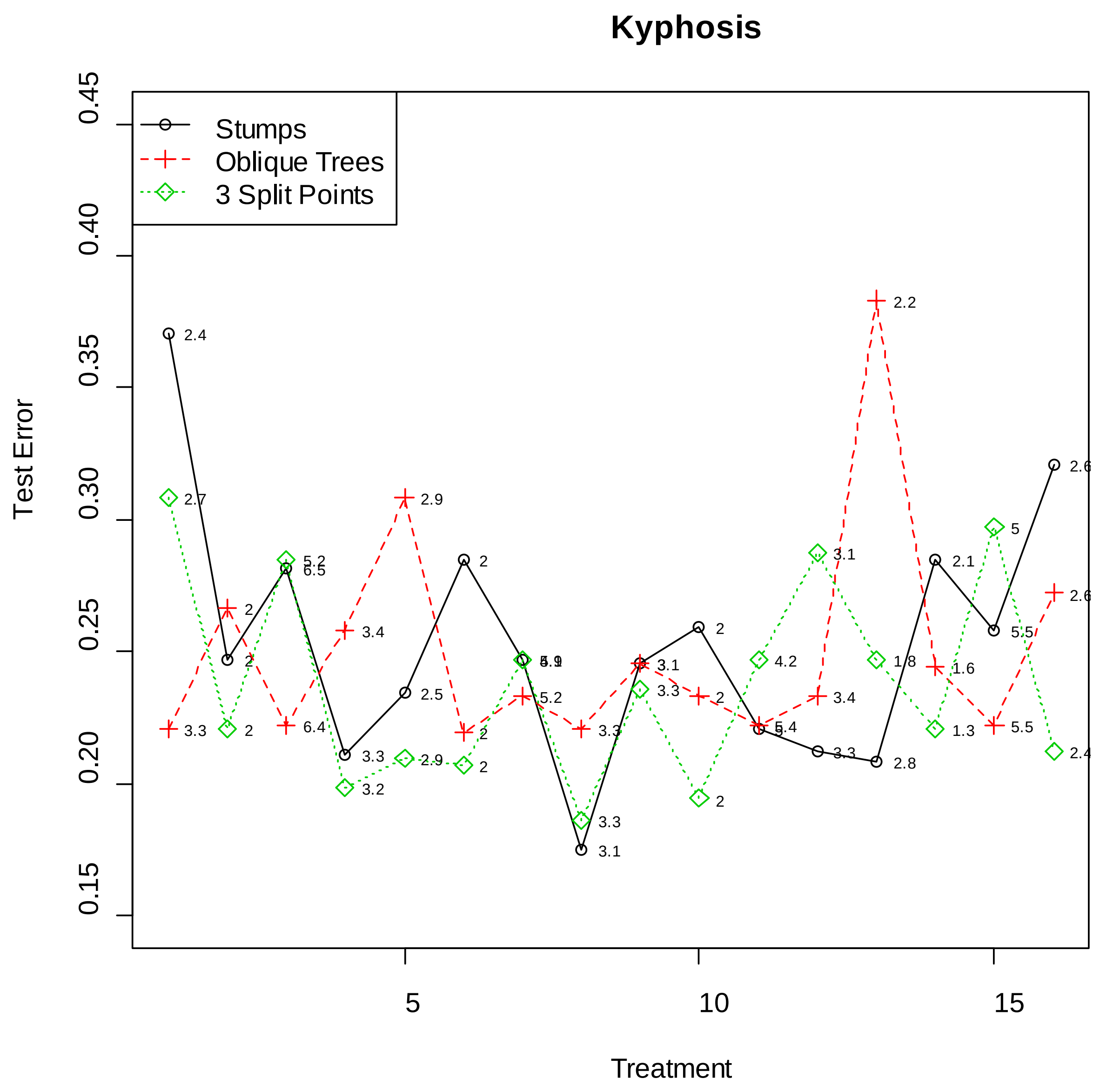


Figure 4.10. Plot of test errors versus 16 treatments. The solid line is from decision tree with one split point, the dashed line with red is from oblique tree, and the dotted line with green is from decision tree with 3 split points based on the kyphosis data. Labeling points are the average numbers of weak learners (*T*).

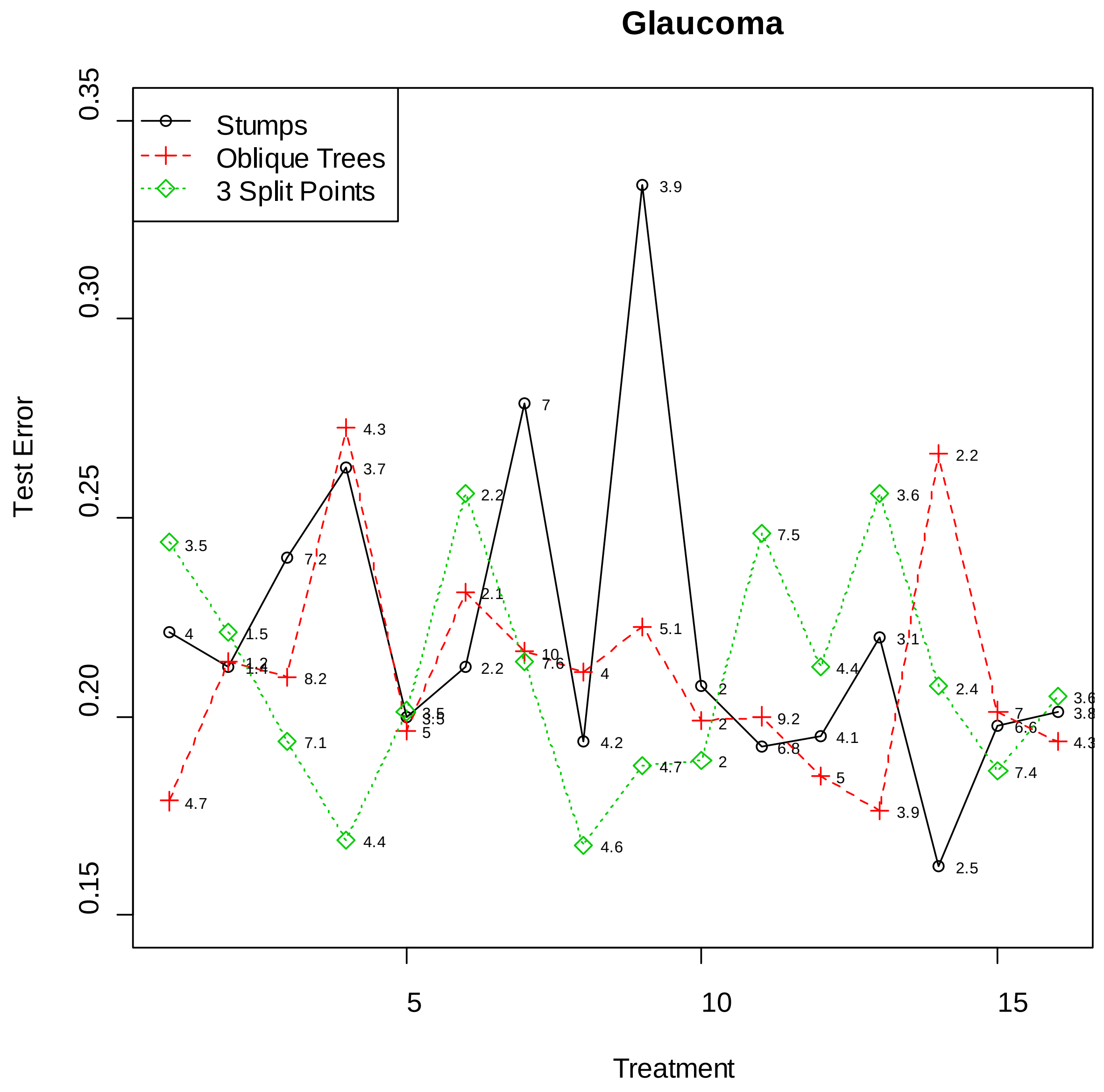


Figure 4.11. Plot of test errors versus 16 treatments. The solid line is from decision tree with one split point, the dashed line with red is from oblique tree, and the dotted line with green is from decision tree with 3 split points based on the glaucoma data. Labeling points are the average numbers of weak learners (*T*).

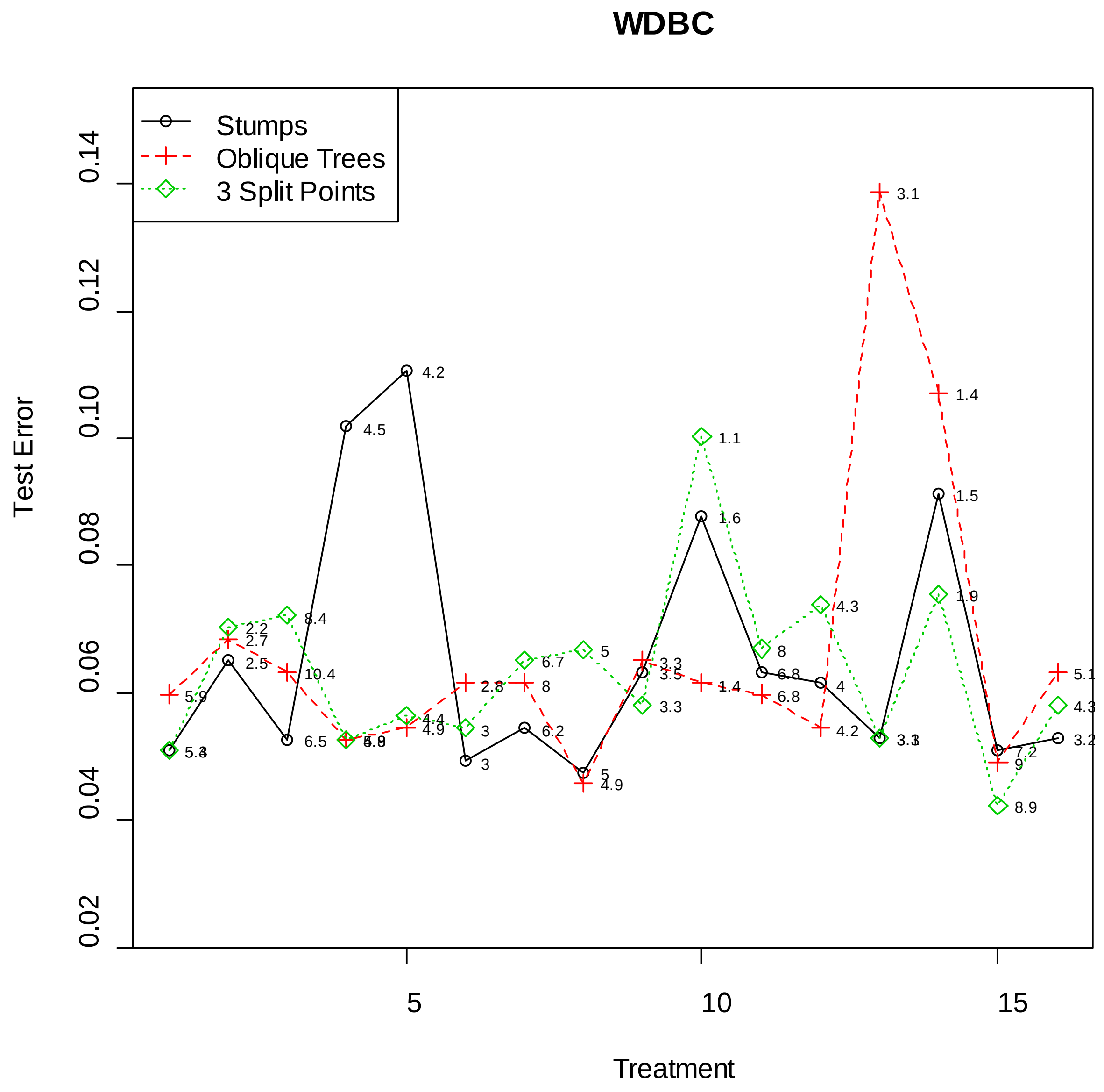


Figure 4.12. Plot of test errors versus 16 treatments. The solid line is from decision tree with one split point, the dashed line with red is from oblique tree, and the dotted line with green is from decision tree with 3 split points based on the WDBC data. Labeling points are the average numbers of weak learners (*T*).

4.5. Choosing the *p*-th percentile of margins

In this section we examine how to choose the *p*-th percentile of margins to maximize based on a simulation data set and two real data sets (kyphosis and glaucoma). The new fitness function for a genetic algorithm is based on optimizing the *p*-th percentile of the margin distribution so as to allow contaminated points to have large negative margins. By maximizing the *p*-th percentile of the margins, we can trim or ignore *p*% of the smallest margins. One of the strong advantages of GA-Boost is resistance to outliers through choosing *p*. Schapire et al. (1998) suggested that an upper bound of test error based on margin percentiles might be used for choosing *p* by comparing test errors from varied values of *p*.

There are at least two ways for choosing *p*. Cumulative margin distributions using various values of *p* are one way to determine a best value of *p*. But it would be difficult to see many cumulative margin lines based on various *p* on a cumulative margin graph. We could draw a cumulative margin graph for each *p* but that would make comparisons difficult. Alternatively, we would draw a plot of margins versus maximized percentile to select *p*. Because outliers usually have big negative margins, to observe the gap among the margins of outliers and normal examples may also help to decide which *p* to use. We might choose *p* where there is a large gap among margins. We use the second method to decide a best value of *p*. Also we can utilize the plot of margins versus maximized percentile as a diagnostic method for detecting outliers through the gap among the margins of outliers and normal examples.

4.5.1 Simulation Example

We considered the rectangular data set (2 predictor variables and size n=100) and added 4% noise levels to the data set by flipping the *Y* values for observations 1, 9, 27, and 67. To select a best *p*-th percentile in a new fitness function, GA-Boost used two-terminal-node decision trees (stumps) as weak learners with 500 generations for this simple data set. The tuning parameters *a*, and *b* are set to 0.1 for each. We used a sequence of 21 values of *p* in the new fitness function to observe how the maximized margins are changed according to different *p* values. The *p* values are determined by number of observations, i.e., if there are 100 observations, each observation represents 1% of the data and the *p*-th percentile of margins is maximized from the minimum margin to the twenty-first percentile of margins.

Figure 4.13 (above) plots all margins versus the *p* for the percentile maximized. The maximized *p*-th percentile margin is marked by black dots with a solid line and the dashed line is for the median of the maximized *p*-th percentile of margin. At the maximized minimum margin, the value of the smallest margin is close to zero. At the maximized 1% of ordered margins we can see one point has a -1 margin value which is misclassified with high confidence because GA-Boost ignores or trims the smallest margin and focuses on the 2nd smallest margin to maximize. Until maximizing the fourth percentile margin, the maximized margins are close to zero and median margins are also close to zero, which indicates low confidence of predictive classification. We observe that as the smallest margins are improved, the other margins worsen (Grove & Schuurmans, 1998). After the maximizing the fourth percentile margin, the maximized margins and median margins jump up to around 0.33. We find four observations that are below the maximized margins, and those observations are verified as outliers through Table 4.29.

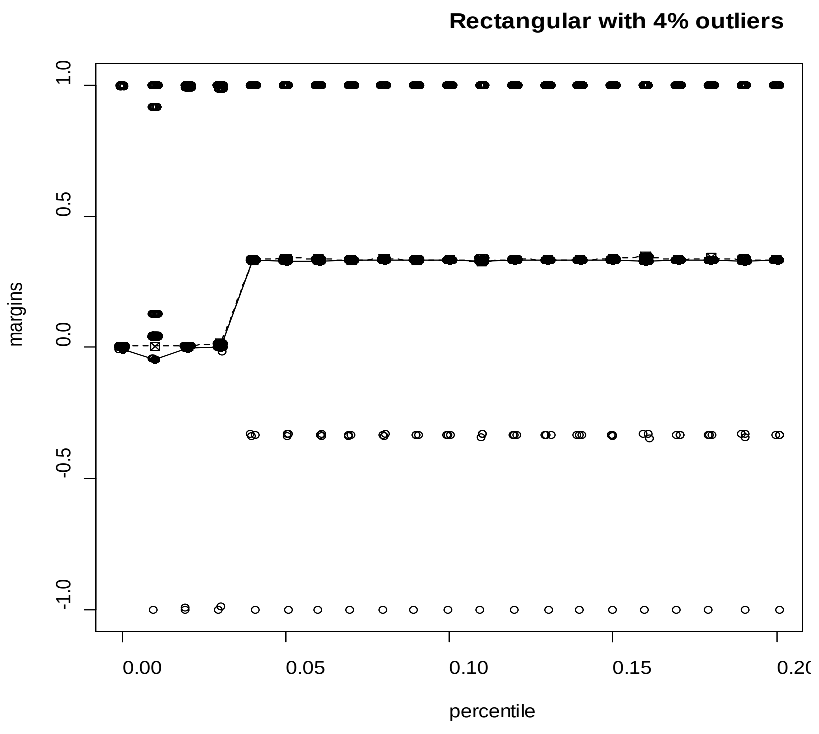
The first two tests of maximizing the *p*-th percentile (0% and 1%) of margins result in four weak learners and the other tests have three weak learners in the best solution in Table 4.29. It seems that a final solution needs more stumps to classify hard points. We can see the difference between the maximized *p*-th percentile of margin and the next smaller margin in Table 4.29 and Figure 4.13. There is a difference between those two numbers until the maximized fourth percentile margin, but there is no gap after the maximized fifth percentile margin. That is one option for deciding which *p* to use and detecting outliers. The bigger the gap between the two values, the more confidence we have that there are outliers.

With respect to a diagnostic method for detecting outliers, Table 4.29 presents important information on the observation numbers, which are below the maximized observation and are ordered by ascending margin. We can see that maximizing the minimum margin has no observation which is ignored, and maximizing the fourth percentile margin has four observations ignored and incorrectly classified. For all tests after maximizing the fifth percentile of margins, the observations 1, 9, 27, and 67 are always negative and there is a large gap with the fifth smallest margin in Figure 4.13.

Table 4.29. Results from GA-Boost for maximizing *p*-th percentile of margins based on the rectangular data with 4 noise levels. First row is the *p*-th percentile from 0.00 to 0.20. The second row is the number of weak learners in final solution, the third is the maximized margin value, the fourth row is the margin which is right before maximized margin, the fifth is the median of all margins on each test, and last row is observation numbers that are incorrectly classified.

p	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
T	4	4	3	3	3	3	3	3	3	3	3
MaxMargin	-0.006	-0.103	-0.005	0	0.331	0.33	0.33	0.332	0.331	0.332	0.333
MaxMargin - 1	-	-1	-0.991	-0.015	-0.331	0.33	0.33	0.332	0.331	0.332	0.333
Median Margin	0.004	0.044	0.005	0.015	0.331	0.339	0.337	0.332	0.337	0.333	0.333
Misclassified observation numbers	1	**1**	1	1	1	1	1	1	1	1	1
	9	67	**9**	67	24	67	67	24	67	24	67
	24	24	67	**24**	67	24	9	67	9	67	24
	67	9	24	9	**9**	9	27	9	27	9	9

p	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.2
T	3	3	3	3	3	3	3	3	3	3
MaxMargin	0.328	0.332	0.333	0.333	0.332	0.327	0.333	0.333	0.328	0.333
MaxMargin - 1	0.328	0.332	0.333	0.333	0.332	0.327	0.333	0.333	0.328	0.333
Median Margin	0.328	0.335	0.333	0.333	0.337	0.345	0.333	0.343	0.334	0.334
Misclassified observation numbers	1	1	1	1	1	1	1	1	1	1
	24	67	67	24	9	67	24	24	9	67
	9	9	24	67	24	24	9	9	67	24
	67	27	9	9	67	9	67	67	24	9



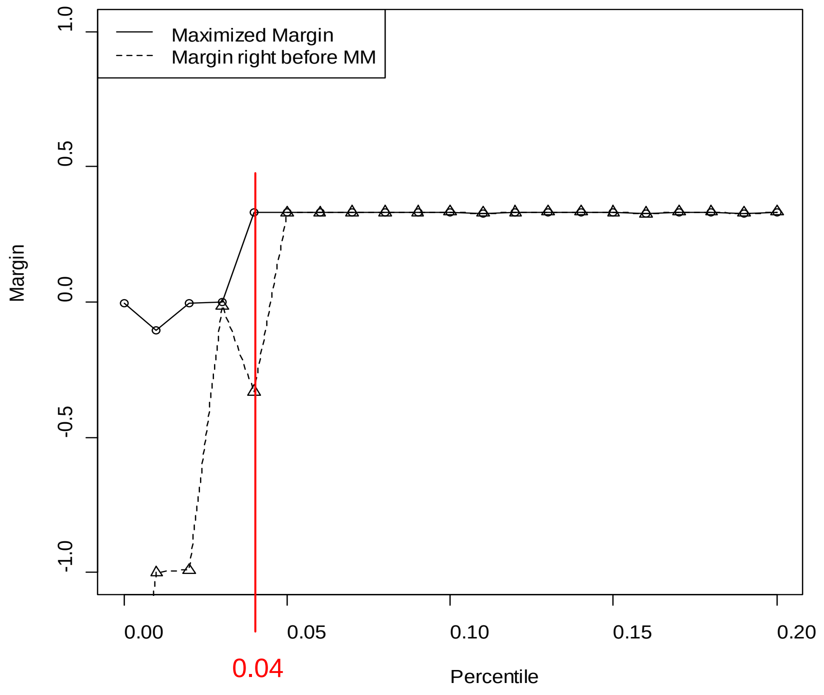


Figure 4. 13. Plots of margins versus percentile. The solid line is the maximized margin value and the dashed line is for the median margin (above) and the dashed line is for is the margins right before the maximized margin based on simulation data I with 4 outliers (below).

4.5.2 Real World Examples

This sub-section examines choosing the *p*-th percentile of margins based on two real data sets (kyphosis and glaucoma) from the UCI Machine Learning Repository. The kyphosis data consists of three predictor variables with *n*=81 observations. 64 of 81observations have -1 as the response value and 17 observations are +1. The glaucoma data having 62 predictors with n=196 is more complex and larger data set than the kyphosis data. Class distribution is exactly half (-1) and half (+1) of data.

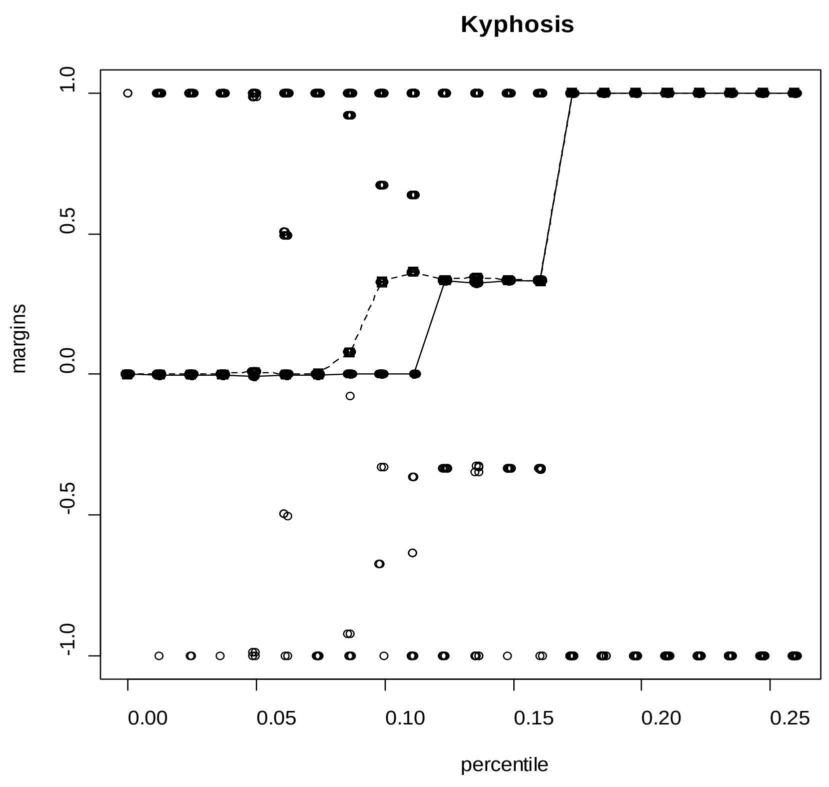
Two-terminal-node decision trees (stumps) were utilized as weak learners with 2,000 generations for both real data sets. We set the values of both *a* and *b* to 0.1. The *p* values are determined by the number of observations. 21 sequenced values of *p* were used for the kyphosis data and 33 sequenced values of *p* for the glaucoma data. Because the kyphosis data has 81 observations, each observation is 1.23% of the data, thus we can maximize the *p*-th percentile of ordered margins from the minimum margin to the 25.9th percentile of margins. For the glaucoma data (n=196), each observation accounts for 0.51%. Thus we can maximize *p*-th percentile of margin to 16.8% with 33 sequenced values of *p*.

The problem with our plotting method is that it is difficult to show for large data sets. For example, the Wisconsin Diagnostic Breast Cancer (WDBC) data set has 569 observations and each observation proportion is 0.176%. Thus if we want to test to up to the 20th percentile of margins for choosing *p*, we need 125 sequenced values of *p*. Larger data sets will require even more labor.

Based on the kyphosis data, Figure 4.14 (above) shows all margins versus the *p*%. The plot can be divided into three distinct sections. Until *p*=0.111, the maximized margins are almost zero values with negative signs and then jump up to around 0.33 for the next four *p* values. After *p*=0.173, the maximized margins are +1 and others are all -1 due to a weak learner in a single final solution for each test. Before the first fourteen *p* values, the final solutions for each test have three weak learners except for *p*=0.00 and 0.037 (with two and four weak learners, respectively). The median margins of each test are similar to the value of the maximized margin except at *p*=0.086, 0.099, and 0.111.

In Table 4.30 we see the observation numbers that are misclassified. Bold indicates the observation numbers which are maximized. We can see some observations are below the maximized margins and the order of observation numbers are presented as the worst to the best margin, but margin order is meaningless after maximizing *p*=17.3% because the margins are equal. The observation numbers 11, 23, 40, 43, 46, and 77 always have negative margins. So those observations are difficult to be classified and we would treat them as outliers. Observations 10 and 24 are correctly classified only at the 2.5th percentile. Therefore those points are not easily classified and can be regarded as atypical data.

The differences between the maximized margin and the next smallest margin are large until maximizing the 6.2th percentile of margins in Figure 4.14. After that, there is no difference between them except at *p*=0.1235, and 0.1728, where the maximized margins jump up. This is a possible clue for deciding how to choose *p* because hard points usually have big negative margins and normal examples should not be difficult to classify. To observe the gaps among the margins of outliers and normal examples may also help to decide which *p* to use. Therefore we might select the 6.2th percentile of margins to maximize for the kyphosis data.



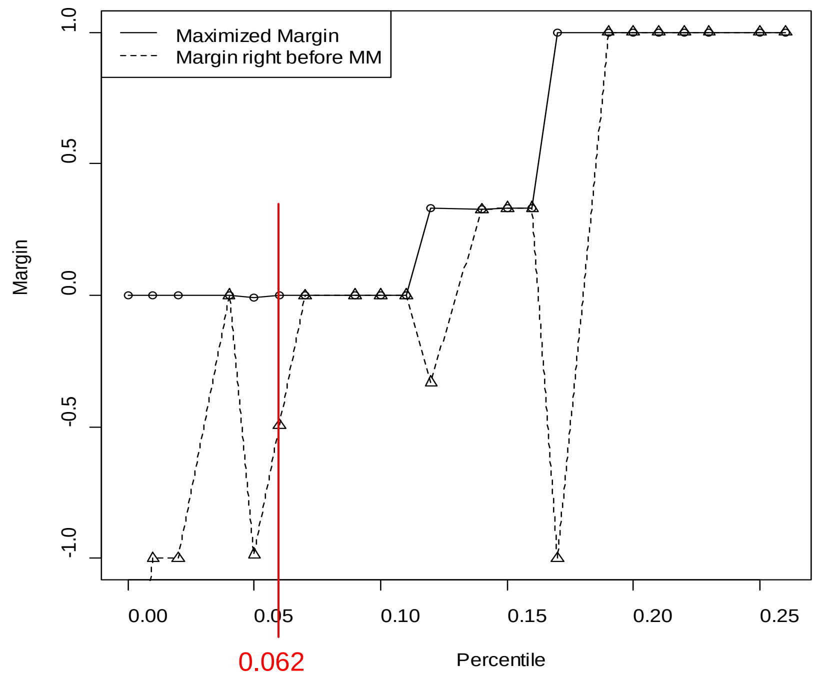


Figure 4. 14. Plots of margins versus percentile. The solid line is the maximized margin value and the dashed line is for median margin (above) and the dashed line is for is the margins right before the maximized margins based on simulation data I with 4 outliers (below) based on the kyphosis data.

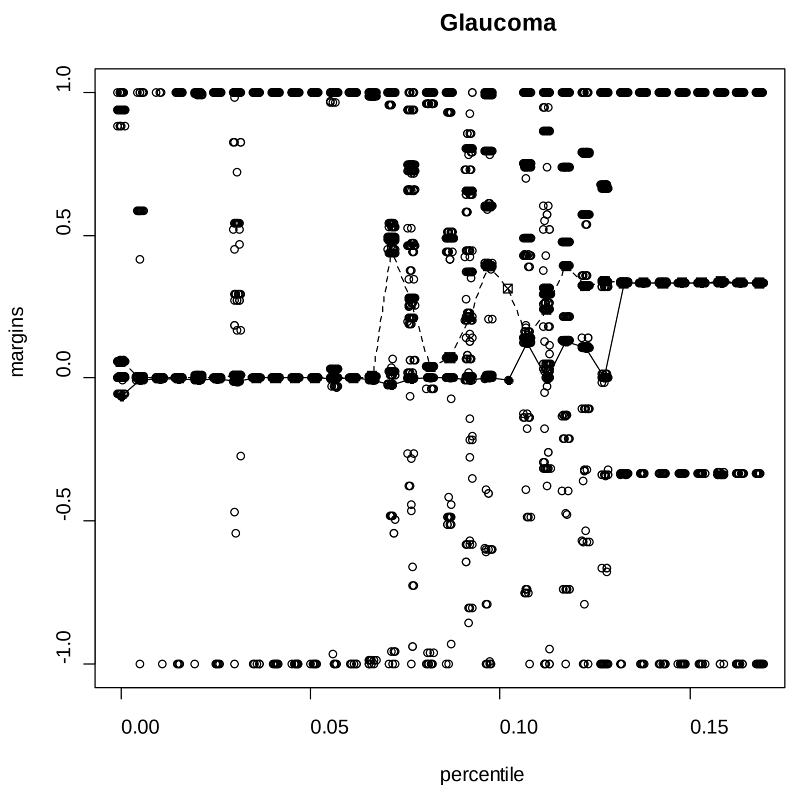
Table 4.30. Results from GA-Boost for maximizing the *p*-th percentile of margins based on the kyphosis data. First row is the *p*-th percentile from 0.00 to 0.2593. The second row is the number of weak learners in final solution, the third is the maximized margin value, the fourth row is the margin which is right before maximized margin, the fifth is the median of all margins on each test, and the last row is observation numbers that are incorrectly classified.

p	0.0000	0.0123	0.0247	0.0370	0.0494	0.0617	0.0741	0.0864	0.0988	0.1111	0.1235
T	2	3	3	4	3	3	3	3	3	3	3
MaxMargin	-0.0004	-0.0009	-0.0005	-0.0005	-0.0090	-0.0007	-0.0017	0.0000	-0.0002	-0.0001	0.3318
MaxMargin - 1	-	-1.0000	-1.0000	-0.0005	-0.9848	-0.4941	-0.0017	0.0000	-0.0002	-0.0001	-0.3318
Median Margin	0.0004	0.0009	0.0000	0.0005	0.0090	0.0007	0.0017	0.0788	0.3280	0.3649	0.3353
Misclassified observation numbers	**1**	**43**	11	25	10	10	11	10	43	11	11
	4	**1**	**77**	10	43	43	40	11	10	23	23
	10	13	**4**	**11**	24	38	43	4	24	40	40
	11	24	23	**23**	**63**	24	10	38	38	43	43
	13	44	38	40	**11**	**63**	24	41	41	10	46
	23	63	40	46	23	**23**	**23**	24	11	24	77
	24	10	41	77	40	46	**46**	**23**	23	46	38
	27	11	46	1	46	77	77	**40**	**40**	77	41
	40	23	49	24	77	40	38	46	**46**	**38**	10
	43	40	58	43	38	11	41	77	77	**41**	**24**
	44	46	61	44	39				41		
	46	77	39	63	43				80		
	63		43		51						
	77		59		59						

p	0.1358	0.1481	0.1605	0.1728	0.1852	0.1975	0.2099	0.2222	0.2346	0.2469	0.2593
T	3	3	3	1	1	1	1	1	1	1	1
MaxMargin	0.3251	0.3322	0.3319	1	1	1	1	1	1	1	1
MaxMargin - 1	0.3251	0.3322	0.3319	-1	1	1	1	1	1	1	1
Median Margin	0.3448	0.3348	0.3319	1	1	1	1	1	1	1	1
Misclassified observation numbers	10	43	11	1	1	1	1	1	1	1	1
	11	11	43	4	4	4	4	4	4	4	4
	23	23	10	10	10	10	10	10	10	10	10
	40	40	24	11	11	11	11	11	11	11	11
	43	46	23	13	13	13	13	13	13	13	13
	46	77	40	23	23	23	23	23	23	23	23
	77	38	46	24	24	24	24	24	24	24	24
	24	41	77	27	27	27	27	27	27	27	27
	38	10	38	40	40	40	40	40	40	40	40
	41	24	41	43	43	43	43	43	43	43	43
				44	44	44	44	44	44	44	44
				46	46	46	46	46	46	46	46
				63	63	63	63	63	63	63	63
				77	77	77	77	77	77	77	77

Figure 4.15 (above) shows all examples of margins according to maximizing the *p*-th percentile of margins based on the glaucoma data. Like the kyphosis data, the plot can be divided into three distinct sections. Until *p*=0.0663 the maximized margins and the median margins are almost zero values but the maximized margins are negative. At the section from *p*=0.0714 to *p*=0.1276, the maximized margins are almost zero except for the last five values of *p* and the median margins fluctuate with from three to six weak learners. And after the maximizing *p*=0.1327, the maximized margins and median margins jump up to around 0.33 and all margins remain consistent with three weak learners in the final solutions.

Misclassified observations from GA-Boost with varied values of *p* are presented in Table 4.31. The maximized observation numbers are bold. Observations 81 and 164 are always misclassified and we would treat them as outliers. Observations 43, 105, 107, and 187 are incorrectly classified except two times so that we might classify them as hard examples. There is the gap between the maximized margin and next smallest margin until maximizing the 13.27th percentile of margins except at *p*=0.0102, 0.0204, 0.0306, 0.0357, 0.0408, 0.0510, 0.0612, 0.0663, 0.0765, 0.0816, 0.0969, 0.1071, 0.1122, 0.1173, and 0.1224 from Table 4.31 and Figure 4.15. After *p*=0.1327, there is no difference between the maximized margin and next smallest margin. We can find there are no gaps from some *p* values before maximizing 13.27th percentile of margin. One of the reasons for this is that the glaucoma data is a complex and large data set. Using 2,000 generations for evolving a solution may not be sufficient. In this case, we might choose *p* just before a point where there is no gap between the maximized margin and next smallest margin. Therefore 0.1327 might be the best *p* for maximizing based on the glaucoma data.



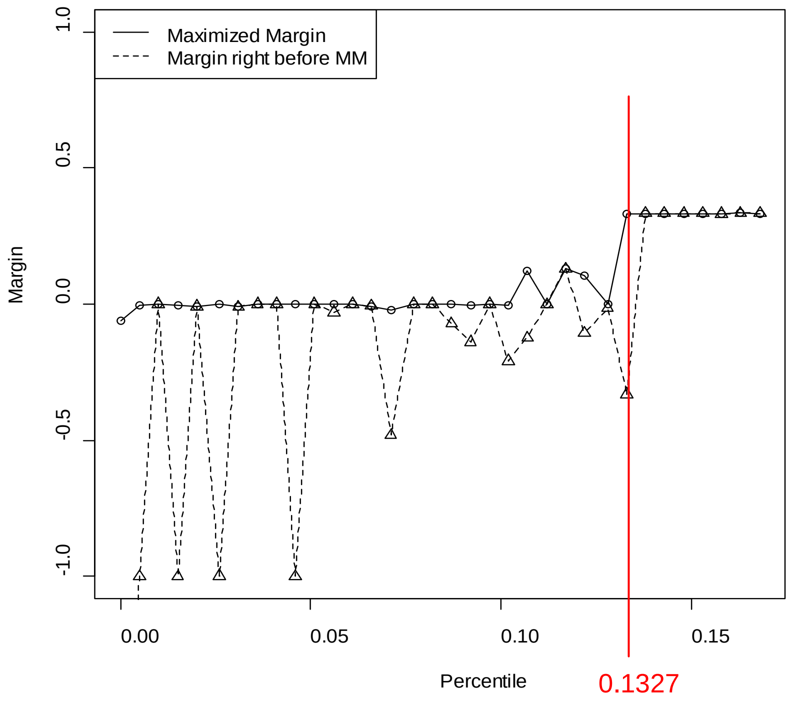


Figure 4. 15. Plots of margins versus percentile. The solid line is the maximized margin value and the dashed line is for median margin (above) and the dashed line is for is the margins right before the maximized margins based on simulation data I with 4 outliers (below) based on the glaucoma data.

Table 4.31. Results from GA-Boost for maximizing the *p*-th percentile of margins based on the glaucoma data. First row is the *p*-th percentile from 0.00 to 0.1684. The second row is the number of weak learners in final solution, the third is the maximized margin value, the fourth row is the margin which is right before maximized margin, the fifth is the median of all margins on each test, and last row is observation numbers that are incorrectly classified.

p	0.0000	0.0051	0.0102	0.0153	0.0204	0.0255	0.0306	0.0357	0.0408	0.0459	0.0510	0.0561
T	4	5	4	4	4	5	5	3	3	3	3	4
MaxMargin	-0.063	-0.005	-0.001	-0.004	-0.009	-0.002	-0.010	0.000	0.000	0.000	0.000	-0.002
MaxMargin - 1	-	-1.000	-0.001	-1.000	-0.009	-0.998	-0.010	0.000	0.000	-1.000	0.000	-0.030
Median Margin	0.057	0.001	0.000	0.000	0.000	0.001	0.009	0.000	0.000	0.000	0.000	0.002
Misclassified observation numbers	**27**	81	130	105	81	72	105	27	81	81	147	81
	104	**105**	29	72	105	81	43	39	103	105	164	105
	131	107	**53**	104	129	159	81	43	105	118	39	163
	147	164	72	**173**	147	5	88	72	107	129	43	173
	157	177	81	27	**177**	43	103	105	177	147	70	86
	163	183	94	39	187	**105**	104	88	190	163	129	72
	164	187	103	81	190	107	**106**	5	196	177	180	27
	167	190	105	51	106	177	127	**29**	104	187	72	29
	173	192	107	2	103	190	131	53	**118**	190	103	40
	179	148	177	20	104	19	133	81	131	**12**	105	43
	183	194	190	68	107	29	147	103	147	35	**107**	53
	187	196	6	79	127	39	150	104	152	43	118	**88**
	192	103	13	88	131	53	152	106	157	70	152	103
	48	104	63	5	152	95	159	107	163	88	159	104
	35	119	89	29	157	106	164	110	164	103	163	107
	39	131	196	40	164	133	173	127	167	104	177	118
	43	147	133	43	173	145	177	129	173	107	179	131
	57	152	159	53	179	153	187	133	179	119	190	133
	65	157	164	54	183	163	190	157	183	131	192	147
	70	173	166	107	192	164	192	159	187	152	193	152
	72	179	173	118	35	166	27	163	192	157	2	164
	81	39	187	131	40	173	29	164	35	164	6	177
	86	43	5	133	43	179	53	173	43	167	13	187
	88	55	39	147	65	187	72	177	65	173	21	190
		65	43	164		27		179	72	183	53	192
				187		103		187	86	192	64	
								190	88		73	
								193			76	

p	0.0612	0.0663	0.0714	0.0765	0.0816	0.0867	0.0918	0.0969	0.1020	0.1071	0.1122	0.1173
T	3	3	5	6	3	4	6	6	5	5	6	5
MaxMargin	0.000	-0.007	-0.024	-0.001	0.000	0.000	-0.007	0.000	-0.006	0.124	-0.002	0.130
MaxMargin - 1	0.000	-0.007	-0.482	-0.001	0.000	-0.072	-0.140	0.000	-0.209	-0.124	-0.002	0.130
Median Margin	0.000	0.007	0.449	0.262	0.039	0.072	0.214	0.393	0.312	0.140	0.240	0.393
Misclassified observation numbers	43	27	81	81	27	81	107	43	107	81	103	105
	81	39	107	107	39	177	43	81	159	72	105	103
	105	72	192	159	72	5	53	105	164	105	107	107
	118	81	72	125	81	107	81	107	187	107	177	164
	177	105	105	187	105	164	110	39	39	190	190	177
	187	104	118	164	107	187	159	104	81	5	81	190
	190	107	183	86	164	190	22	177	72	39	72	72
	104	133	187	183	173	27	29	164	103	43	104	81

	107	164	119	192	20	39	72	187	105	53	127	53
	127	173	164	196	58	43	105	133	177	29	159	187
	129	187	103	43	196	53	163	27	43	103	164	163
	131	5	177	144	104	57	193	53	94	177	173	173
	133	29	179	145	133	72	173	72	127	129	179	159
	147	**43**	190	39	145	105	187	5	173	152	196	5
	152	53	**39**	165	187	153	147	173	190	104	22	29
	163	57	88	**166**	5	29	164	190	183	133	43	94
	164	177	127	173	**29**	65	104	29	110	164	110	192
	173	2	157	182	43	**40**	129	163	145	173	187	196
	179	58	163	190	53	103	**27**	63	63	187	166	58
	192	71	6		177	104	23	**65**	196	40	126	43
	29	88	28		190	163	28		**28**	58	53	133
	40	196	76			173	63		89		13	145
	53		94				76				**55**	
	106						94					

p	0.1224	0.1276	0.1327	0.1378	0.1429	0.1480	0.1531	0.1582	0.1633	0.1684
T	5	4	3	3	3	3	3	3	3	3
MaxMargin	0.106	0.000	0.332	0.333	0.332	0.333	0.333	0.331	0.333	0.333
MaxMargin - 1	-0.106	-0.016	-0.332	0.333	0.332	0.333	0.333	0.331	0.333	0.333
Median Margin	0.322	0.338	0.332	0.333	0.333	0.333	0.334	0.338	0.333	0.333
Misclassified observation numbers	105	104	43	27	81	27	147	81	43	72
	177	105	81	39	107	43	150	104	72	81
	190	107	177	72	119	53	164	164	105	103
	72	118	27	81	147	72	187	29	107	104
	39	119	29	104	164	81	192	43	164	105
	43	163	40	105	177	105	29	53	177	107
	81	177	53	107	187	107	53	72	183	118
	107	179	72	133	190	133	72	103	187	164
	192	190	106	164	192	159	81	105	190	173
	69	192	104	173	27	164	103	107	192	177
	88	103	105	187	40	177	105	118	103	187
	5	43	107	5	43	187	107	150	104	27
	53	164	118	29	53	192	159	152	118	43
	103	173	127	43	72	106	177	159	119	163
	110	35	129	53	103	110	190	163	129	110
	179	81	131	57	104	166	104	177	131	120
	19	88	133	2	105	13	119	179	147	125
	29	131	147	20	118	39	129	190	152	127
	94	147	152	58	131	41	173	193	157	159
	196	152	163	71	152	58	20	106	163	166
	127	167	164	87	163	63	23	119	167	179
	164	187	173	88	173	29	94	129	173	182
	173	196	179		65	103	148	131	179	190
	187	65	187		183	118	183	147	65	55
		72	190			152	159	19	81	65
		129	**192**			163	27			86
		183				190				
						193				

Chapter 5

GA-Boost Variants

GA-Boost is not a sequential method for constructing the final model, but optimizes the weak learners and their weights by a genetic algorithm simultaneously. It indirectly controls the number of weak learners by the tuning parameter *b* and crossover operations in its genetic algorithm, but the actual number of weak learners in the final solution cannot be specified. It may be interesting to build the model with the specific number of weak learners we desire. We are also interested in how to generate the initial parameters in the population of solutions. Original GA-Boost utilizes the initial population of solutions randomly generated, but we can draw parameters from the final solution of AdaBoost or other ensemble methods. With these interests, we started to develop other versions of GA-Boost, and we describe three extensions of GA-Boost in this chapter.

The AdaBoost algorithm (Freund & Schapire, 1997) does not optimize the choice of weights and weak learners. Also, AdaBoost doesn't maximize the margin (Rudin et al., 2007). There is previous research dealing with the problem "Does AdaBoost maximize the margin?" Rätsch et al. (2002) suggested that AdaBoost perhaps maximizes the margins for the optimal case in that it chooses the best weak learner at each iteration, but the final solution doesn't necessarily maximize the margin. Rudin, Schapire, and Daubechies (2004) showed that AdaBoost may converge to a margin which is significantly below the maximum. In our experience through the simulated simple example in chapter 3, a repeating and cyclic pattern in the sequence of weak learners created by AdaBoost is observed. As we also mentioned, the weights based on weak learners from AdaBoost are probably not optimized so that AdaBoost tends to revisit and reweight some weak learners later.

We introduce three different GA-Boost variants. Because AdaBoost does not maximize the margin, we would like to improve the solution by taking the best weak learners and their weights. The core idea is that some weak learners with their weights could be drawn from the AdaBoost solution, and then we could apply a genetic algorithm to optimize the parameters to produce the best prediction. Figure 1 shows the flow chart of GA-Boost variants. Roughly speaking, the variants of GA-Boost are combining GA-Boost with AdaBoost.

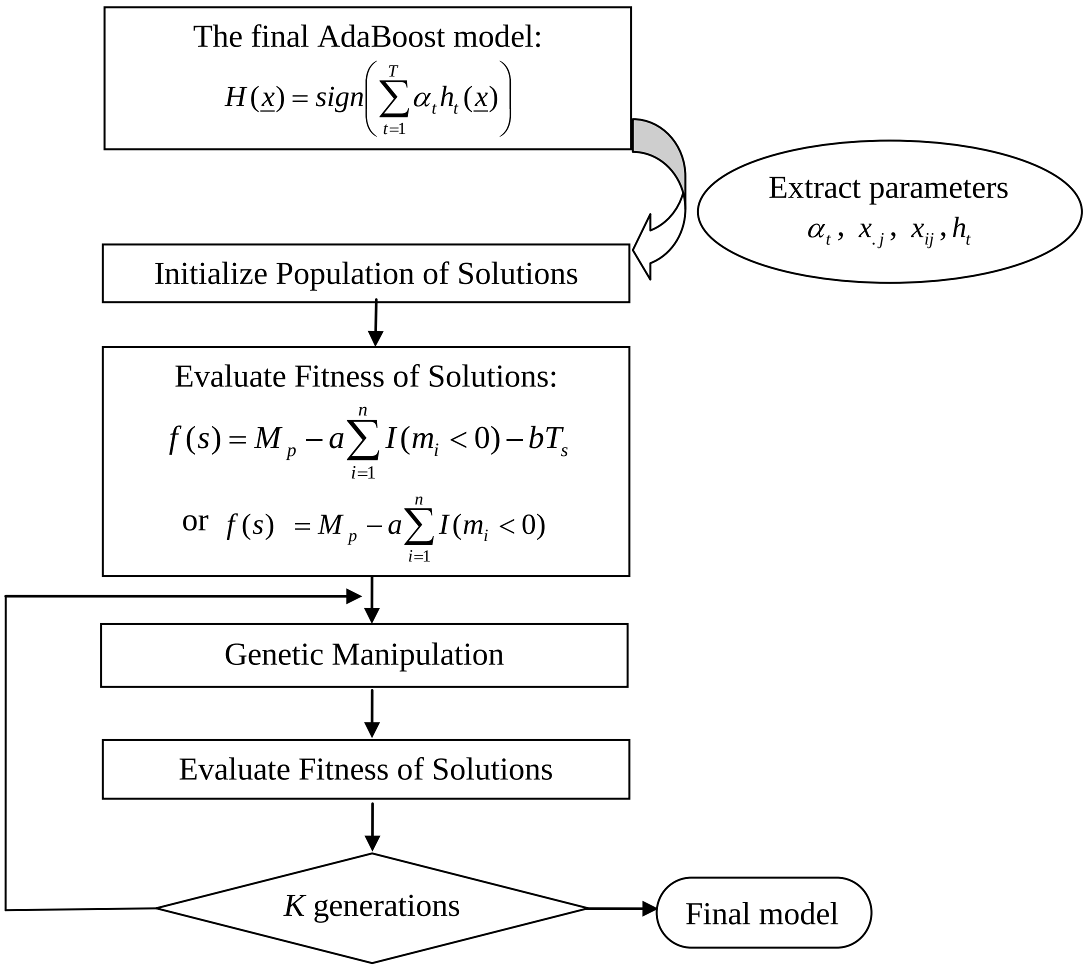

The final AdaBoost model:
$H(\underline{x}) = sign\left(\sum_{t=1}^{T}\alpha_t h_t(\underline{x})\right)$
Extract parameters
$\alpha_t,\ x_{.j},\ x_{ij}, h_t$
Initialize Population of Solutions
Evaluate Fitness of Solutions:
$f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0) - bT_s$
or $f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0)$
Genetic Manipulation
Evaluate Fitness of Solutions
K generations
Final model

Figure 5.1. GA-Boost Variants Flow Chart

5.1 GA-Boost Variants

Three variants have a very similar algorithm to the original GA-Boost. The main differences are how to generate the initial weak learners and their weights, and to preserve the size of solutions (except for GAdaBoost I). GA-Boost has an initial population of solutions which consists of weak learners and their weights randomly generated. Initial weak learners consist of the split variables, split values, and predictions of the stumps, which are randomly chosen. However, GA-Boost variants utilize the parameter values (weights, split variables, split values, and predictions) of a final AdaBoost model for the initial population of solutions instead of a randomly chosen one. After drawing parameters from AdaBoost, those can be optimized by a genetic algorithm with better starting members than those randomly chosen in GA-Boost. We may expect that three variants have better performance than AdaBoost and original GA-Boost, and could reduce the time required for convergence using a genetic algorithm because of the initial parameters which are closer to the best than the random initial parameters of GA-Boost.

5.1.1 GAdaBoost I

GAdaBoost I is the first variant of GA-Boost. It has the same algorithm as GA-Boost except for the initialization step. First, the parameters are extracted from the final solution of AdaBoost, then GAdaBoost I can optimize weights, variables, split points, and predictions.

The initial population is generated with 20 solutions and the length of a solution is T, the number of weak classifiers. Each solution consists of T weak classifiers and their weights. The number of initial weak learners in GAdaBoost I is determined by the number of iterations of AdaBoost. For example, if the iteration number (T) of AdaBoost is 50, the initial number of weak learners for GAdaBoost I is also 50. The sizes of solutions would be changed along with the values of a tuning parameter, *b* for a penalty in a fitness function, and genetic operations (crossover, grow and prune).

The fitness function of the genetic algorithm for the first variant is the *p*-th percentile of the margin distribution with penalties for the number of negative margins and the number of weak learners, similar to GA-Boost. Therefore GAdaBoost I has the same advantages of GA-Boost, which are more resistance to outliers and simpler predictive models than other boosting methods. Five genetic operations are used in GAdaBoost I to evolve solutions: elitism, mutation, crossover, grow, and prune.

We expect that GAdaBoost I achieves better predictive accuracy than GA-Boost with the same number of generations because we start optimizing the parameters with a better fitness value than that of initial solutions randomly generated. In other words, the fitness value from GA-Boost with 2,000 generations is same as that from GAdaBoost I with less than or equal 2,000 generations. Therefore with same number of generations in a genetic algorithm for GA-Boost and GAdaBoost I, the latter might have better performance.

5.1.2 GAdaBoost II

The second variant, named GAdaBoost II, retains the initial number of weak learners. Like GAdaBoost I, the weak learners and their weights are optimized by a genetic algorithm after drawing the starting population from an AdaBoost solution. We can have the final solution with the same initial number of weak learners as in the final model from AdaBoost.

During the progress of GA-Boost and GAdaBoost I, the number of weak learners in each generation could be changed (usually reduced) due to the genetic operations (crossover, grow and prune) and the tuning parameter *b* for complexity in a fitness function. To retain the initial number of weak learners, the penalty term with respect to complexity is removed from the new fitness function (5.1) in the genetic algorithm. The modified fitness function (5.2) for the second variant of GA-Boost consists of the *p*-th percentile of the margin distribution with only a penalty for the number of negative margins.

$$f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0) - bT_s \tag{5.1}$$

$$f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0) \tag{5.2}$$

In order to preserve the number of trees we must further control the genetic operations. In the existing crossover operation, crossover points of the two solutions are randomly determined, and the next two solutions are created by swapping parts of the two solutions (Figure 5.2). It results in a change the lengths of the new solutions because each solution has a separate choice of crossover point. Instead, a single crossover point on two solutions is selected so that the length of the new solutions is the same as the parents' lengths (Figure 5.3). Also, the genetic operations (grow and prune) which can change the size of a solution are not used to preserve the number of weak learners.

5.1.3 GAdaBoost III

The experience gained from the simulated simple example in chapter 3 shows that a repeating and cyclic pattern in the sequence of weak learners created by AdaBoost is observed. In conclusion, the weights based on weak learners from AdaBoost are probably not optimized so that AdaBoost tends to revisit and reweight some weak learners later. For this reason, we consider only optimizing the weights of the weak learners by a genetic algorithm after extracting the weak learners from an AdaBoost solution.

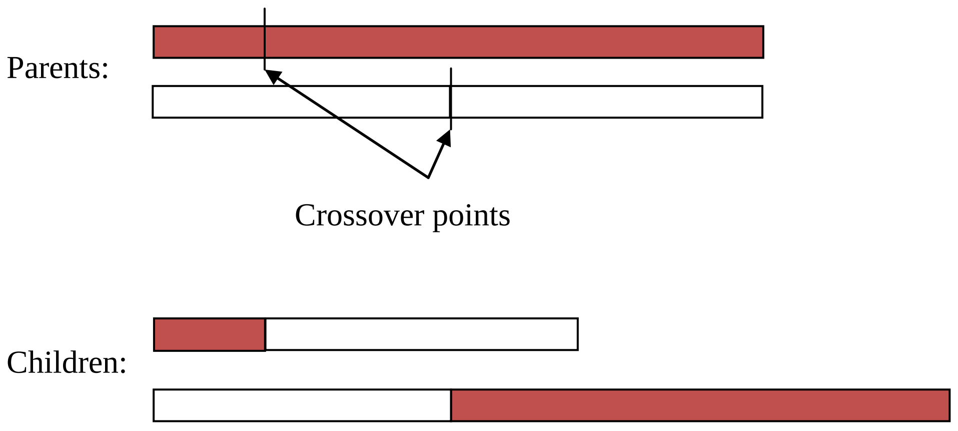


Figure 5.2. Crossover points of any two solutions are randomly determined. The chosen two solutions (parents) are split at crossover points and the next two solutions (children) are created by swapping parts of two solutions (parents).

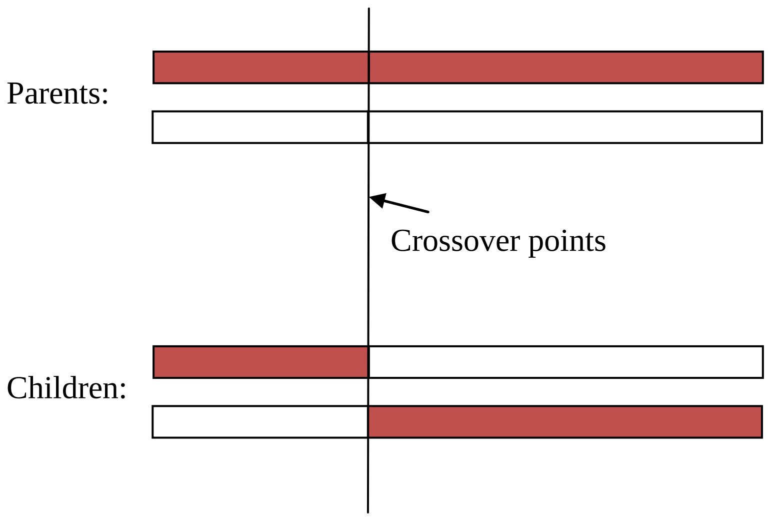


Figure 5.3. A single crossover point on two solutions (parents) is selected so that the length of the new solutions (children) is the same as the parents' length.

The last variant is named GAdaBoost III and it can retain the initial number of weak learners like the second variant. To do that, we ought to give restraint to control the number of weak learners.Therefore we control the genetic operations and the fitness function the same way as the second variant. The complexity term in penalization and two genetic operations (grow and prune) are excluded from algorithm. Crossover operation is limited to not changing the length of solutions; the length of the new solutions is the same as a parent's length. After modifying the algorithm, we optimize the solution only consisting of the weights of the weak learners, while we do not optimize other parameter values (split variables, split values, and predictions) of the final AdaBoost model but preserve these initial values to use for the final model.

For the second and the third variants, we believe those variants will achieve better predictions when compared to AdaBoost with the same number of trees. At least those methods can perform as well as AdaBoost does, because they can optimize the all parameters (weights, variables, split points, and predictions) for the second variant and only optimize weights for the third from a final model of AdaBoost. And by maximizing the *p*-th percentile of the margins, we don't pay too much attention to outliers in the data. But due to optimizing the parameters without reducing the number of weak learners, we encounter a speed problem in computation and it also makes interpretability more difficult.

5.1.4 Algorithm

The pseudocode for the three variants of GA-Boost is given in Figures 5.4 - 5.6. The input training set is given with *n* examples which are pairs: $(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)$, where $\underline{x}_i$ belongs to some instance space *X* and the label of y_i is noted by {-1, +1}. We choose some percentile, *p*, to maximize and choose the number of generations for the genetic algorithm and the number of weak learners. For the initialization step, the population consists of *N* generated solutions. We evolve the solutions through *K* generations, and each generation is evaluated by a new fitness function. Genetic operations are used to create a new generation of solutions and the fitness function of a genetic algorithm measures the goodness of each solution to determine the next generation of solutions. Finally after *K* simulations we choose the solution in the final

```
Input  n examples $(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)$   where $\underline{x}_i \in X$, $y_i \in Y = \{-1, +1\}$
       p (percentile), K (number of generations), T (initial number of weak learners)
Initialize   A generated population of N solutions
       (A solution s consists of a set of $T_s$ weights ($\alpha_t$) and $T_s$ weak learners.)
       The parameters are chosen from AdaBoost for the initial population of solutions.
Evolve   for k = 1, 2, …, K generations
         Evaluate fitness of solutions:   $f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0) - bT_s$
         Use genetic operations to create new generation of solutions:
             Elitism, Mutation, Crossover, Grow and Prune
Output   Final hypothesis with weights $\alpha_t$
```

Figure 5.4. The GAdaBoost I algorithm.

generation with a largest fitness value as the final solution, which consists of *T* weak learners and their weights.

The GAdaBoost I algorithm is in Figure 5.4. A solution is a set of weak learners and their weights, which are chosen from the final AdaBoost model for the initial population of solutions. Each generation is evaluated by the same fitness function as GA-Boost. Elitism, mutation, crossover, grow and prune are used as genetic operations.

In Figure 5.5, GAdaBoost II starts out under the same initial conditions of GAdaBoost I. But to retain the number of weak learners, the reduced fitness function, which sets the tuning constant *b* to zero, is utilized to evaluate each generation, and elitism, mutation, and crossover are chosen as genetic operations. A crossover operation with a single crossover point on two solutions is selected.

```
Input  n examples $(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)$   where $\underline{x}_i \in X$, $y_i \in Y = \{-1,+1\}$
       $p$ (percentile), $K$ (number of generations), $T$ (initial number of weak learners)
Initialize   A generated population of $N$ solutions
       (A solution $s$ consists of a set of $T_s$ weights ($\alpha_t$) and $T_s$ weak learners.)
       The parameters are chosen from AdaBoost for the initial population of solutions.
Evolve   for $k = 1, 2, \ldots, K$ generations
       Evaluate fitness of solutions:   $f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0)$
       Use genetic operations to create new generation of solutions:
           Elitism, Mutation, and Crossover
Output   Final hypothesis with weights $\alpha_t$
```

Figure 5.5. The GAdaBoost II algorithm.

```
Input  n examples $(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)$   where $\underline{x}_i \in X$, $y_i \in Y = \{-1,+1\}$
       p (percentile), K (number of generations), T (initial number of weak learners)
Initialize   A generated population of N solutions
       (A solution s consists of a set of $T_s$ weights ($\alpha_t$). )
       The parameters are chosen from AdaBoost for the initial population of solutions.
Evolve   for k = 1, 2, …, K  generations
       Evaluate fitness of solutions:   $f(s) = M_p - a\sum_{i=1}^{n} I(m_i < 0)$
       Use genetic operations to create new generation of solutions:
           Elitism, Mutation, and Crossover
Output   Final hypothesis with weights $\alpha_t$
```

Figure 5.6. The GAdaBoost III algorithm.

Figure 5.6 is the pseudocode for GAdaBoost III. The initial population of solutions *s* consists of a set of T_s weights (α_t), which are extracted from the final AdaBoost model. We optimize them by using GA. The reduced fitness function (5.2) and genetic operations, which are the same as GAdaBoost II in Figure 5.5, are employed with this algorithm.

5.2 Experiments

In this section, we present the performance of the three GA-Boost variants and compare them to GA-Boost and AdaBoost on two artificial data sets (rectangular data set with 10% noise and circle data sets) and data sets from the UC-Irvine Machine Learning Repository (kyphosis, glaucoma, and WDBC) described in chapter 4. So we demonstrate how variants of GA-Boost work and how they compare to one another and AdaBoost based on different data sets.

We use the AdaBoost algorithm with 10 and 50 iterations and the GA-Boost algorithm maximizing the 10th (or 5th) percentile of the margins as described in the previous section using 2,000 generations with a=0.7 and b=0.1 (or 0.7) for artificial data set and real world data sets. In the experiments we did not examine the effect of the tuning constants a, b, and p in the fitness function of the genetic algorithm. Stumps (one split, two terminal nodes) were used as weak classifiers for all algorithms.

5.2.1 Simulation data sets

The rectangular data set with 10% noise and the concentric circle data set are borrowed from the simulation study in chapter 4. We see how GA-Boost variants perform on those data sets and compare GA-Boost variants to AdaBoost as well as compare among variants. For the initial number of weak learners for GA-Boost and three variants, we considered two different initial numbers, which are 10 and 50 extracted from the final AdaBoost model combined with 10 or 50 weak learners. Because GA-Boost used less than or around 10 stump trees for simulation and real data sets in chapter 4, we want to use 10 weak learners for initial number of trees for variants. Therefore we can compare GA-Boost variants with AdaBoost with a simple model. Though AdaBoost usually uses hundreds or thousands weak learners in a final solution, we just tested AdaBoost with 50 iterations, which is the maximum iteration number since the current programming speed is not good to implement GA-Boost variants with a long length of solutions. The results are displayed in Table 5.1 and 5.2. Bold indicates win among all algorithms along with two different iteration numbers.

Rectangular Data

In the rectangular data set, 100 observations were considered as training set and a set of 10,000 observations was generated for a test set. We set the values of both *a* and *b* to 0.7 based on the best result in a previous study in chapter 4. From the result in chapter 4, few weak learners were employed to classify the rectangular data for GA-Boost and we apply this to the first GA-Boost variant, GAdaBoost I. Therefore we assign a larger penalty *b* (0.7) for the number of weak learners in the fitness function to produce fewer weak learners. We set *b* to 0.00 for the second and the third variants to disregard the effect of the penalty for the number of trees in a solution. To classify this data set with a 10% noise level, it is optimal to maximize the 10th percentile of margins which is robust to noisy examples.

For the table 5.1, the test errors for AdaBoost with different iterations (T=10 and 50), GA-Boost, and three different GA-Boost variants with different initial solutions (initial T = 10 and 50) based on the rectangular data set with 10% noise. The numbers in parentheses are the number of weak learners (T) in a final model of each algorithm. First, for the initial 10 stump trees, the best outcomes come from GA-Boost and GAdaBoost I, which is a 0.021 error rate with three stumps in a final model. AdaBoost has 0.049 error rate and GAdaBoost II and III have 0.067 and 0.072 error rates, respectively, with 10 combined weak learners in final models. Therefore the methods having three trees in a final model are proper for classifying the rectangular data set with a large proportion of outliers.

AdaBoost, GAdaBoost II, and III have very poor outcomes for 50 initial weak learners except GA-Boost and GAdaBoost I which have slightly worse performance than for the initial T=10. The test errors from all methods are increased as the number of iterations increases. The smallest test error among those is 0.027, which is from GAdaBoost I and the worst is 0.277 from

Table 5.1. Test errors for AdaBoost with different iterations (T=10, and 50) and GA-Boost with three different variants based on rectangular data set with 10% noise. The numbers in parentheses are the number of weak learners (T) in a final model of each algorithm.

Test Error (*Final T*)	AdaBoost	GA-Boost	GAdaBoost I	GAdaBoost II	GAdaBoost III
Initial T = 10	0.049 (10)	**0.021** (3)	**0.021** (3)	0.067 (10)	0.072 (10)
Initial T = 50	0.134 (50)	0.029 (3)	**0.027** (3)	0.202 (50)	0.277 (50)

GAdaBoost III. The test error of AdaBoost is 0.134 which is not better than test errors from GA-Boost and GAdaBoost I which are 0.029 and 0.027, respectively, but better performance than GAdaBoost II and III which are 0.202 and 0.277 for the rectangular data.

We verify GA-Boost and GAdaBoost I have better performance than AdaBoost and the second and the third variants, which are overfitting when there is a large proportion of outliers in a simple data set. Using T=10 for the initial number of weak learners has better results than T=50 for rectangular data set.

Circle Data

The second simulation used a concentric circle data structure, which is a more complex data structure for correctly classifying all training data than the rectangular set. 300 observations are drawn for constructing the model and 5,000 observations are generated for test. To build a final model for all ensemble methods, stump trees are employed as weak learners. The number of stumps in the final model, T, is related to the values of b. The circle data set has a complex data structure, so we may need larger T to correctly classify them. Therefore we put a small value of the tuning constant b for the number of weak learners in the fitness function since small b results in a model with large T. Based on the result in chapter 4, to examine the performance of GA-

Boost and GAdaBoost I, we set the combination of (*a*, *b*) to (0.7, 0.1) and to (0.7, 0.0) for other variants. Because we observed that the effects of *p* are insignificant for this data set from Table 4.17 in chapter 4 and also we misclassified many observations for complex structure from Table 4.15, we set *p*=0.10. To save run-time, the simulation number was 2,000, which is not enough to get a full convergence of the fitness in GA with this circle data set.

Table 5.2 is the results for AdaBoost, GA-Boost and three GA-Boost variants with different iterations or initial number of trees (*T*=10 and 50) based on circle data set. For *T* = 10, the best outcome is 0.2760 from GAdaBoost I with 10 stumps in a final model. Compared to AdaBoost with 0.4242 error rate, GA-Boost, GAdaBoost I and II perform well with 0.3344, 0.2760, and 0.3240 error rates, respectively. The number of combined trees in a final solution is 10 except GA-Boost has 11 stumps. At least we expect GAdaBoost III with optimized weights may better perform than AdaBoost but the performance of GAdaBoost III did not lead to the expected result in terms of test error. However, for *T*=50, the result for GAdaBoost III is as expected.

For an increased initial number of trees, *T*=50, GA-Boost and all variants work well and have better test errors than AdaBoost. The best result is from GAdaBoost II with a 0.2916 error

Table 5.2. Test errors for AdaBoost with different iterations (*T*=10, and 50) and GA-Boost with three different variants based on circle data set. The numbers in parentheses are the number of weak learners (*T*) in a final model of each algorithm.

Test Error (*Final T*)	AdaBoost	GA-Boost	GAdaBoost I	GAdaBoost II	GAdaBoost III
Initial *T* = 10	0.4242 (10)	0.3344 (11)	**0.2760** (10)	0.3240 (10)	0.4508 (10)
Initial *T* = 50	0.4298 (50)	0.3382 (11)	0.2976 (11)	**0.2916** (50)	0.3504 (50)

rate. The test error of AdaBoost, GA-Boost, GAdaBoost I and II are 0.4298, 0.3382 and 0.2976, respectively, which are worse performances than for *T*=10 but GAdaBoost III is improved. The final solutions have 50 stump trees for AdaBoost , GAdaBoost II, and III while GA-Boost and GAdaBoost I have 11 stumps in their final model. Through Table 5.2 we see that GAdaBoost I and II perform very well for the circle data set.

5.2.2 Real data sets

Our comparisons based on real data sets are presented in this section. We selected three real data sets from the UCI Machine Learning Repository: the kyphosis data, the glaucoma data, and the Wisconsin Diagnostic Breast Cancer (WDBC) data. For comparison, 10-fold cross-validation was used to estimate the test error. We set the values of both *a* and *b* to 0.7 but set *b* to 0.00 for the second and the third variants. The simulation number in GA is 2,000. Also we tested two different iterations or initial solutions (*T*=10 and 50).

Table 5.3 displays 10-fold cross validation with standard errors from AdaBoost, GA-Boost, and three variants based on three real data sets. There are two initial solutions (or iterations) which are *T*=10, and 50. Final *T* is the average number of weak learners in a final model over the 10 runs of each algorithm. GA-Boost and GAdaBoost I resulted in better interpretability with fewer than 9 weak learners on average in a final solution than the final model from AdaBoost, GAdaBoost II and III for three data sets. Bold numbers indicate that generalization performances by GA-Boost and the three variants are better than that of the combined classifiers by AdaBoost along with two different iterations. The performance of AdaBoost is not much different between *T*=10 and 50 for kyphosis and glaucoma but we can see that the performance is very much improved at *T*=50 compared to at *T*=10 for the WDBC.

For the kyphosis data, the test errors from AdaBoost are 0.2333 at *T*=10 and 0.2347 at *T*=50 in Table 5.3. At *T*=10, GA-Boost has excellent performance with a 0.1361 error rate and GAdaBoost II also performs well (0.2097). GAdaBoost I and III are inferior to AdaBoost. While GAdaBoost II has the best performance at *T*=50. GA-Boost, GAdaBoost I and II also have better results than AdaBoost at *T*=50. The number of weak learners in a final solution is around three trees for GA-Boost and GAdaBoost I for both initial *T*'s.

When we compare for the glaucoma data, GA-Boost and its variants have better performance than AdaBoost with *T*=10 and 50 except GAdaBoost I at *T*=50. GA-Boost has around 4 weak learners in a final model at both initial solutions. GAdaBoost I has a large average

Table 5.3. 10-fold cross validation (standard errors) from AdaBoost with iterations (*T*=10 and 50) and GA-Boost with three different variants based on real data sets: kyphosis, glaucoma, and WDBC data sets. Final *T* is the average number of weak learners in a final model of each algorithm. Test errors are estimated by 10-fold cross validation.

nGen=2,000 *a*=*b*=0.7, *p*=0.05		Kyphosis		Glaucoma		WDBC	
		Initial *T* = 10	*T* = 50	*T* = 10	*T* = 50	*T* = 10	*T* = 50
AdaBoost	*Test Error*	0.2333	0.2347	0.2137	0.2037	0.0667	0.0334
	(*Std. Err*)	(0.0272)	(0.0435)	(0.0309)	(0.0204)	(0.0093)	(0.0071)
	Final *T*	10	50	10	50	10	50
GA-Boost	*Test Error*	**0.1361**	**0.2083**	**0.2038**	**0.1475**	**0.0597**	0.0563
	(*Std. Err*)	(0.0293)	(0.0357)	(0.0346)	(0.0225)	(0.0075)	(0.0083)
	Final *T*	3	3.0	3.9	4.1	3.9	4.3
GAdaBoost I	*Test Error*	0.2861	**0.2083**	**0.1888**	0.2162	**0.0632**	0.0474
	(*Std. Err*)	(0.0837)	0.0357	(0.0163)	(0.0253)	(0.0098)	(0.0064)
	Final *T*	2.9	3.1	7.4	8.8	8.4	6.8
GAdaBoost II	*Test Error*	**0.2097**	**0.1722**	**0.1950**	**0.1888**	**0.0509**	**0.0334**
	(*Std. Err*)	(0.0559)	0.0377	(0.0320)	(0.0163)	(0.0115)	(0.0055)
	Final *T*	10	50	10	50	10	50
GAdaBoost III	*Test Error*	0.2472	0.2819	**0.1938**	**0.1925**	**0.0632**	0.0352
	(*Std. Err*)	(0.0457)	0.0506	(0.0189)	(0.0258)	(0.0083)	(0.0070)
	Final *T*	10	50	10	50	10	50

number of weak learners (7.4 at *T*=10 and 8.8 at *T*=50) but not a very good result at *T*=50. Except for GAdaBoost I, the performances at *T*=50 are superior to *T*=10 in terms of test set error.

Finally for the WDBC data, all algorithms at initial *T*=50 have better generalization errors than at initial *T*=10. At 10 weak learners in initial solutions, generalization performances by GA-Boost and the three variants are better than that of the combined classifiers by AdaBoost. At *T*=50, AdaBoost performs better than other algorithms, except that GAdaBoost II and AdaBoost have the same generalization error as 0.334 but GAdaBoost II has a better error rate. GAdaBoost I generates more weak learners in a final model than GA-Boost, but they have an advantage of interpretation over AdaBoost and other two variants.

5.3 Summary

In this chapter we have discussed the extended three variants of GA-Boost based on AdaBoost solutions. The variants of GA-Boost combine a sequential method for constructing the AdaBoost model with the GA-Boost method for optimizing the weak learners and their weights by a genetic algorithm. Regular GA-Boost has the initial solutions consisting of weak learners and their weights which are randomly generated, while GA-Boost variants utilize the AdaBoost solution (weights, split variables, split values, and predictions) extracted from the final AdaBoost model for the initial population of solutions.

The first variant of GA-Boost, GAdaBoost I, has the same algorithm as GA-Boost except for the initialization step. The number of weak learners in initial solutions is determined by the number of iterations of AdaBoost. We call the second and the third variants GAdaBoost II and III, respectively. They retain the initial number of weak learners. Like GA-Boost, the ensemble is optimized by a genetic algorithm but after starting from AdaBoost with a fixed number of iterations. The difference between the second and the third variants is that the former optimizes all parameters (weights, split variables, split values, and predictions) but the latter only optimizes the weights of the weak learners.

From the results based on two simulation data sets and three real world data sets, the variants of GA-Boost performed very well as predictive methods. When there is a large proportion of outliers in a simple data set, GA-Boost and GAdaBoost I performed better with a simpler solution that looks more like the original population of rectangular data and is more resistant to outliers than AdaBoost, GAdaBoost II, and III. For testing the second simulated data set (circle), all extended versions have better performance than AdaBoost, except GAdaBoost III at *T*=10. And for the real data sets, most results from GA-Boost and its variants performed better than AdaBoost except for WDBC at *T*=50. GAdaBoost II is the best method among the variants based on the real data sets examined.

We suggest some considerations and weak points with respect to the extended versions of GA-Boost. First when extracting the ensemble from AdaBoost, we might have duplicated weak learners. Thus we could reduce the number of weak learners for the initial population of solutions. We need to improve the fitness function in the GA, especially, for GAdaBoost II and III, which use two terms for the maximizing the *p*-th percentile of margin and the penalty for the number of negative margins to develop better performance. Due to the programming speed to implement the new variants, we considered 50 weak learners in the initial solutions. Thus we need to test more than 50 iterations for the initial solutions so that we can see how the variants work for a large number of iterations. Computational considerations kept us from running a larger simulation study. Finally, the current research is very limited with respect to the effect of the tuning constants, *a*, *b*, and *p* in the fitness function of the genetic algorithm.

Chapter 6

Summary, Conclusion and Future Research

6. 1 Summary

Boosting methods have been theoretically and experimentally developed to solve the classification problem. AdaBoost (Freund et al., 1995) is the most common boosting algorithm for binary classification. However, Adaboost is sensitive to noisy data and outliers so there is a possibility for overfitting. Grove et al. (1998) showed that AdaBoost sometimes does overfit. SmoothBoost (Servedio, 2003), BrownBoost (Freund, 2001), and GentleBoost (Friedman et al., 2000) are alternatives that are more robust to the presence of outliers. To improve the performance of boosting, properties of the cumulative margin distribution (CMD) have been studied. AdaBoost$_\rho$ (Rätsch et al., 2002), LPBoost (Grove et al., 1998), and DOOM (Mason et al., 1998) are focused on making the margins as large as possible to improve generalization performance.

This research has introduced a new boosting method called GA-Boost, which uses a genetic algorithm to solve the binary classification problem. We compared it with AdaBoost using three different weak classifiers: decision trees with two terminal nodes (stump), decision trees with three split points, and oblique stump trees.

Through the simulated simple example with outliers in chapter 3, we observe a repeating and cyclic pattern in the sequence of 200 weak classifiers constructed by AdaBoost and noted that AdaBoost devotes too much attention to outliers in the data in trying to attain a perfect fit to the training data. AdaBoost also has an interpretation problem with its hundreds or thousands of weak learners in the final model. Based on these interesting findings and disadvantages of AdaBoost, we proposed the GA-Boost (Genetic Algorithm Boosting) algorithm, which solves directly for the weak learners and the weights of those weak learners using a genetic algorithm. We developed a new penalized fitness function in a genetic algorithm. Using this new penalized fitness function, we have an interpretability advantage over other boosting algorithms due to the reduced model complexity and control the effects of outliers by the choice of the maximized *p*-th percentile of margins. The genetic algorithm for GA-Boost is composed of initialization and randomization steps, a fitness function, and a genetic evolution step.

We compared GA-Boost to AdaBoost on several data sets including some simple simulated data sets and three real world data sets from UC-Irvine Machine Learning Repository. Results from these examples suggest that GA-Boost is more resistant to the effects of outliers and provides solutions that are simpler than those found by AdaBoost. In addition, we tested the effect of the parameters (*a*, *b*, and *p*) of a new fitness function. To examine the effect, simulations were designed for 16 treatments corresponding to different combinations of *p*, *a*, and *b* shown in Table 4.1.

The purpose of the first simple simulation data with two predictors is to show how maximizing the *p*-th percentile in a new fitness function for GA-Boost works with respect to four different noise levels (0%, 1%, 5%, and 10%). Two-terminal-node decision trees (stumps) were used as weak learners. The values of *p* with four different combinations of (*a, b*) do not significantly affect the result based on 0% and 1% noise in the data sets. To classify the data sets with a 5% and 10% noise levels, it is a good idea to utilize a value of *p* that is greater than the noise level. As a result, with respect to test error, the choice of *p* should be made to get better performance. The main effects of *a* and *b*, and their interactions are also significant when data sets are highly contaminated. For the comparison to AdaBoost, on the 0% and 1% noise data sets, AdaBoost did better than any results from GA-Boost. As the noise level increases, test errors from both algorithms increase, however in most cases GA-Boost is better than AdaBoost. Therefore we verify that AdaBoost is overfitting and GA-Boost has better performance if a large proportion of outliers are in the data set.

The second simulation used a concentric circle data structure. We considered more complex weak learners beyond stump trees to get a better result because this data set has a more complex data structure than the first one. The main effects of *a* and *b* have a strong influence on the results. A small value of the tuning constant *b* (0.1), which controls the number of weak learners, is best for this data set but the four different levels of *p* do not significantly affect the results. We compared GA-Boost with 16 treatments to AdaBoost with different numbers of iterations. GA-Boost with less than 10 weak learners achieves better accuracy and interpretability than AdaBoost with hundreds or thousands of weak learners. To extend weak learners beyond stump trees, we also considered decision trees with three split points and oblique stump trees for GA-Boost with the circle data set. GA-Boost for 3 split points tends to produce a better result than stumps and oblique stump trees. GA-Boost with oblique trees achieved better performance than stumps.

To explore the effect of *b*, we set up different values of *b* with two values of *p* (= 0.00 and 0.10) and the fixed value of *a* (= 0.1) for the first simulation data set with 10% noise and the circle data set. For the first simulated data set, the smaller the value of *b*, the worse the performance with large weak learners we have in maximizing the minimum margin for the simple simulated data. Because GA-Boost with *p*=0.1 allows ignoring examples which have the 10% smallest margin values, we only need a few stumps to correctly classifying the normal examples. But a large *b* brings out a weak performance for both *p* values. To test the second simulated data set (circle), regardless of maximizing the *p*-th percentile of margins, we need to use a small value of *b* to generate more weak learners for better performance on the complex simulated data.

In comparisons based on three real data sets (kyphosis, glaucoma, WDBC), a 10-fold cross-validation was used to estimate the test error. From comparisons using stump trees as a weak learner, we can conclude that GA-Boost is better and worse than AdaBoost for the glaucoma and the WDBC data, respectively. For the kyphosis data, both algorithms are somewhat equally matched. The number of weak classifiers in the final models from GA-Boost has a significant interpretability advantage over the final model from AdaBoost for the three real world data sets. Through comparisons with more complex weak learners than stump trees, we would like to see how GA-Boost performs with different weak learners, with the expectation that a complex classifier as a weak learner works better than a simple one, but computational cost will be higher. There were few differences among three different weak learners we considered. In the future research we will consider why the results for the real data sets differed from our expectation.

The new fitness function use in the genetic algorithm is based on the *p*-th percentile of the margin distribution to allow contaminated points to have large negative margins. By maximizing the *p*-th percentile of the margins, we can trim or ignore *p*% of the smallest margins. In the literature review, margin percentiles affect the upper bound of test error (Schapire et al., 1998). The question is which value of *p* to use. To choose *p*, a plot of margins versus percentiles to maximize could be useful in selecting *p*. Because outliers usually have big negative margins, to observe the gap among the margins of outliers and normal examples may help to decide which *p* to use. We might choose *p* where there is a large gap among margins. Also we can utilize the plot of margins versus percentiles as a diagnostic method for detecting outliers through the gap between the margins of outliers and normal examples. From the simulated data with n=100, p=2, and 4 atypical observations, we can easily find four observations below the maximized margin (*p*th percentile) and those observations are verified as outliers. Based on the kyphosis data, we might select the 0.062 percentile of margins to maximize because the differences between the maximized margin and the next smallest margin are large until maximizing the 0.062 percentile of margins. After that, there is no difference between them. In the same way, 0.1327 might be the best *p* for the maximizing percentiles of margin based on the glaucoma data.

The three extensions of GA-Boost based on AdaBoost solutions have been discussed in chapter 5. The original GA-Boost has the initial population of solutions randomly generated, while the three variants of GA-Boost utilize the AdaBoost solution (weights, split variables, split values, and predictions) extracted from the final AdaBoost model for the initial solutions. The length of initial solutions is determined by the number of iterations of AdaBoost. The ensemble is optimized by a genetic algorithm after starting from AdaBoost with a fixed number of iterations. The first variant has the same algorithm as GA-Boost except for the initialization step. The second and third variants retain the initial number of weak learners. The difference between the second and third variants is that the former optimizes all parameters (weights, split variables, split values, and predictions) but the latter only optimizes the weights of the weak learners.

The three extension versions of GA-Boost performed very well as predictive methods based on the results of two simulation data sets (rectangular and circle) and three real world data sets (kyphosis, glaucoma, and WDBC).

For the rectangular data set with a large proportion of outliers, GA-Boost and GAdaBoost I performed better with simpler solutions and were more resistant to outliers than AdaBoost and the other two variants. All extended variants, including regular GA-Boost, have better performance than AdaBoost for the circle data set except the third variant at T=10. For the real data sets, GA-Boost variants performed better than AdaBoost except for WDBC at T=50. The second variant which optimizes all parameters (weights, split variables, split values, and predictions) is the best method among the variants based on the real data sets examined.

6. 2. Conclusions, limitations and future research

We used several simulation data sets and real data sets in the UCI Repository for a new boosting algorithm (GA-Boost) and three variants to find an optimal model combination of weak learners with their weights and compared to AdaBoost with different numbers of iterations. Based on this research, we realize there are additional things to do in future. In this section, we suggest limitations of our current study and additional experiments and ideas for further study of improve the new boosting method, GA-Boost.

The first concern is to find optimal combinations of the parameters (a, b, and p) in the fitness function. In our tests of the effects of combinations of them, we didn’t test all possible combinations. Just 16 combinations of these were selected. In the future, we need to consider more combinations of them or develop a method to find the best combination. The effect of the parameters will be different depending on the data structure. To choose the *p*-th percentile, we used a plot of margins vs. *p* maximized. We can choose *p* where there are large gaps among margins. But it is difficult to observe all observations in detail in our current plot. We are currently experimenting with dynamic graphics to help in choosing the *p*-th percentile.

Second, we plan further research on why the results for the real data sets differed from our expectation when we compared with the three different weak classifiers. We believe that the number of simulations is too small. Because two classifiers are more complex than a stump tree, to get a better solution, we need to increase the number of generations used in the genetic algorithm. Also we need to increase the number of cross-validations. We just used one 10-fold cross-validation to measure generalization performance. To reduce the variation in test errors, we will increase the repetitions of cross-validations.

Third, we need to improve the fitness function in the genetic algorithm for GA-Boost and its variants, especially, the second and the third variants, which use two terms for the maximizing the *p*-th percentile of margin and the penalty for the number of negative margins to develop better performance. If the programming speed to implement the new variants is solved, we need to test more than 50 iterations for the initial solutions so that we can see how the variants work for a large number of iterations and compare them with other ensemble methods. We are interested in extending to more than the two-class problem.

Finally, we used the R statistical program to implement GA-Boost. We need to improve programming speed and efficiency of our R implementation of GA-Boost or consider using Java or C++. Finally, we would like to run simulation experiments to compare GA-Boost to other robust boosting methods such as SmoothBoost (Servedio, 2003), BrownBoost (Freund, 2001), and GentleBoost (Friedman et al. 2000), and extend it to use various weak learners including decision trees that are fully grown with no pruning.

REFERENCES

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1987), “Occam’s Razor,” *Information Processing Letters*, 24, 377–380.

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

Breiman, L. (1996), “Bagging Predictors,” *Machine Learning*, 26, 123–140.

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

Dietterich, T. G. (2000), “An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization,” *Machine Learning*, 40, 139-157.

Draper, N. H. and Smith, H. (1998), *Applied Regression Analysis*, 3rd ed., John Wiley, New York.

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. (2004), “Least Angle Regression,” *Annals of Statistics*, 32, 407-499.

Freund, Y. (2001), “An Adaptive Version of the Boost by Majority Algorithm,” *Machine Learning*, 43, 293-318.

Freund, Y. (1995), “Boosting a Weak Learning Algorithm by Majority,” *Information and Computation*, 121, 256-285.

Freund, Y., and Schapire, R. (1999), “A Short Introduction to Boosting,” *Journal of the Japanese Society for Artificial Intelligence*, 14, 771-780.

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(1), 119–139.

Freund, Y., and Schapire, R. (1996), “Experiments with a New Boosting Algorithm,” *Machine Learning: Proceedings of the Thirteenth International Conference*, 148-156.

Friedman, J. H. (2006), “Recent Advances in Predictive (Machine) Learning,” *Journal of Classification*, 23, 175-197.

Friedman, J. H., Hastie, T. J., and Tibshirani, R. J. (2000), “Additive Logistic Regression: A Statistical View of Boosting,” *Annals of Statistics*, 28, 337-374.

Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning,* Reading, MA: Addison-Wesley.

Grove, A. and Schuurmans, D. (1998), “Boosting in the Limit: Maximizing the Margin of Learned Ensembles,” *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.

Hastie, T., Tibshirani, R. and Friedman, J.H. (2001), *The Elements of Statistical Learning: Data mining, inference and prediction*, Springer, New York, N.Y.

Holland, J.M. (1975), *Adaptation in Natural and Artificial Systems,* Ann Arbor, MI: The University of Michigan Press.

Kearns, M. and Valiant, L.G. (1988), Learning Boolean Formulae or Finite Automata is as Hard as Factoring (Technical Report TR-14–88). Cambridge, MA: Harvard University Aiken Computation Laboratory.

Mason, L., Bartlett, P., and Baxter, J. (1999), “Direct Optimization of Margins Improves Generalization in Combined Classifiers,” *In Advances in Neural Information Processing System 11*, MIT Press, Cambridge, MA.

Meir, R., and Rätsch, G. (2003), “An Introduction to Boosting and Leveraging”, *Advanced Lectures on Machine Learning*, LNAI 2600, 118-183.

Meir, R., El-Yaniv, R., and Ben-David, S. (2000), “Localized Boosting,” *In Proc. COLT*, pages 190–199, San Francisco, Morgan Kaufmann.

Miller, A. J. (2002), *Subset Selection in Regression*, 2nd ed., 57-59, Chapman & Hall/CRC.

Murthy, S. K., Kasif, S., and Salzberg, S. (1994), “A System for Induction of Oblique Decision Trees,” *Journal of Artificial Intelligence Research*, 2, 1-32.

Quinlan, J.R. (1996), “Boosting First-Order Learning,” *Lecture Notes in Computer Science*, 1160, 143.

Quinlan J. R. (1993), C4.5: *Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann Publishers, Inc.

Rätsch, G., and Warmuth, M. K. (2005), "Efficient Margin Maximization with Boosting," *Journal of Machine Learning Research*, 6, 2131-2152.

Rätsch, G., and Warmuth, M. K. (2002), "Maximizing the Margin with Boosting," *In Proc. COLT*, 2375 of LNAI, pages 319-333, Sydney, Springer.

Rudin, C., Schapire, R., and Daubechies, I. (2007), "Analysis of Boosting Algorithms Using the Smooth Margin Function," *Annals of Statistics*, 35, 2723-2768.

Rudin, C., Schapire, R., and Daubechies, I. (2004), "The Dynamics of AdaBoost: Cyclic Behavior and Convergence of Margins," *Journal of Machine Learning Research*, 5, 1557-1595.

Schapire, R. (1990), "The Strength of Weak Learnability," *Machine Learning*, 5, 197-227.

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.

Schapire, R. and Singer, Y. (1999), "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, 37, 297-336.

Servedio, R. A. (2001), "Smooth Boosting and Learning with Malicious Noise," *In Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*, 473–489.

Shawe-Taylor, J. and Cristianini, N. (1999), "Further Results on the Margin Distribution," *In Proceedings of the 12th Annual Conference on Computational Learning Theory ACM Press*, New York.

Shawe-Taylor, J. and Cristianini, N. (1998), "Margin Distribution Bounds on Generalization," *Proceedings of EuroCOLT'99*, Lecture Notes in Computer Science, 1572, 263-273.

Tibshirani, R. (1996), "Regression Shrinkage and Selection Via the Lasso," *Journal of Royal Statistical Society, Series B*, 58(1), 267-288.

Tsao, C. A. and Chang, Y. I. (2007), "A Stochastic Approximation View of Boosting", *Computational Statistics & Data Analysis*, 52, 325-334

Valiant, L. G. (1984), "A Theory of the Learnable," *Communications of the ACM*, 27, 1134-1142.

Vapnik, V. N. (1995), *The Nature of Statistical Learning Theory*, Springer Verlag, New York, 1995.

Vapnik, V. N. and Chervonenkis, A. Y. (1971), "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theory of Probability and its Applications*, 16(2), 264–280.

Zhu, M. (2008), “Kernels and Ensembles: Perspectives on Statistical Learning,” *The American Statistician*, 62, 97-109.