Stony Brook University

OFFICIAL COPY

The official electronic file of this thesis or dissertation is maintained by the University Libraries on behalf of The Graduate School at Stony Brook University.

© All Rights Reserved by Author.

Multi-Class ROC Random Forest for Imbalanced Classification

A Dissertation Presented

by

Jiaju Yan

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

May 2017

Copyright by
Jiaju Yan
2017

Stony Brook University
The Graduate School

Jiaju Yan

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation.

Wei Zhu – Dissertation Advisor
Deputy Chair, Professor, Department of Applied Mathematics and Statistics

Song Wu - Chairperson of Defense
Associate Professor, Department of Applied Mathematics and Statistics

Pei Fen Kuan
Assistant Professor, Department of Applied Mathematics and Statistics

Keli Xiao
Assistant Professor, College of Business

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Multi-Class ROC Random Forest for Imbalanced Classification

by

Jiaju Yan

Doctor of Philosophy

In Applied Mathematics and Statistics

(Statistics)

Stony Brook University

May 2017

The imbalanced class problem in classification is highly relevant in many realistic scenarios such as the detection of a rare condition. One solution is to design specific algorithms incorporating the unbalanced classes in the training process of a classifier. In this dissertation, we propose a novel multi-class classification tree based on the area under the ROC curve (AUC) to resolve the imbalanced classification problem. This tree classifier aims to maximize the sum of AUC for all one versus all classifiers at the node attribute selection stage while balancing the performance of sensitivity and specificity of all one versus all classification at the node threshold selection stage. The ROC tree is extended to ROC random forest with suitable modifications. Furthermore, the volume under surface (VUS), the extension of AUC for multi-class classification, is discussed in this dissertation as well and used to measure the performance of classifiers. The simulation results show that this multi-class ROC tree/forest method is superior

to the classic CART/random forest on severely imbalanced multi-class classification problems, while the ROC random forest performs equally well as the SMOTE random forest on imbalanced binary classification problems. The application on Boston housing data shows that the ROC random forest can also be used for model ensemble and it performs better than all the base models and other ensemble methods in this application.

Keywords: Imbalanced Classification, ROC, AUC, Tree Based Method, Random Forest, VUS, Model Ensemble

Table of Contents

List of Figures

List of Tables

Chapter I: Introduction

1.1 Classification

Classification is a center piece in machine learning, playing an increasingly important role in modern life. It is used in banks to read checks and detect fraud, in vehicles to voice interact with people, in digital cameras to automatically detect human faces – just to name a few of its multitude of applications. Generally speaking, classification is a process to put items into several known categories. More specifically, the goal of classification is to build a model, or a classifier, to predict the class of incoming observations based on a set of training data with given target labels.

For example, the email spam filter problem is a typical classification problem [1]. In this problem, the training data are a set of emails with each labeled as “spam” or “non-spam”. The standard procedure is to generate features from these emails, analyze the relationships between the features and the labels, and establish a suitable classification model based on these features to predict whether an email is spam or not. In the end, we are able to use the classification model thus built to identify future emails.

Another example is the handwritten digits recognition. In this, we are given a picture of a handwritten digit, which belongs to one of the 10 categories. The same process is applied with features generated on the image, and a classification model trained, and ready for future prediction.

These two examples can also be done by human or rule based methods. The advantages of classification algorithms over human are mainly its speed and scalability for large data, and the advantages of classification algorithms over rule based methods are mainly their performance.

1.1.1 Classification Algorithms

There are many classification algorithms which can be divided to several families. The perceptron [2] is one of the earliest classification algorithms, which was developed to neural networks, or multi-layer perceptron. Furthermore, there could be more and more layers in neural networks with different purposes. Back propagation [3] was developed based on gradient descent to train neural networks. The advantages of neural networks are its ability to achieve excellent performance and its coverage for many fields. The disadvantages of neural networks includes their lack of interpretability, their being computational expensive and their requirement of huge amount of training data.

The Support Vector Machine (SVM) [4] is another classification algorithm which became very popular in the 1990s. SVM tries to find a boundary or a separating hyperplane in the original or transformed feature space to divide data points into different classes. Compared to the neural network, SVM is theoretically simpler and require less computational power. However, as computation power become cheaper and more abundant these years, SVM became less popular than the neural networks due to its lower performance.

Another family is the Decision Trees. Technically speaking, rule based methods are also tree based methods, since they have the similar model structure. However, almost all decision tree methods are able to automatically choose the criteria and threshold based on their design.

The CART proposed by Leo Breiman [5] is an important member of the decision tree family, which will be discussed in detail Chapter 2. The main advantages of decision trees are their simple design and wide applicability in different classification problems. The performance of a single decision tree is normally much worse than SVM or neural networks, but with the help of the ensemble methods, a bunch of trees, befittingly named as a forest, can compete with other classifiers.

There are also other classification methods such as the discriminant analysis, which is based on a strong assumption of the normal distribution. These methods are normally simpler in design and worse in performance, though they can be combined with other methods to achieve some specific goals and/or better performance.

1.1.2 Ensemble Methods

The idea of ensemble is to combine a lot of weak base learners to achieve a higher predictive power. The weak learners here indicate classifiers with lower predictive performance. For example, for a certain classification problem, an SVM classifier may achieve 80% accuracy, while a tree classifiers can only achieve 60% accuracy. In this case they are called weak learner. Simple classifiers are preferred for base learners, as many of them will be used in the ensemble structure. With the help of ensemble methods, the final model is still able to learn a complicated structure presented by the data. The tree classifier is a popular base learner because it is fast and simple.

The ensemble methods can be divided into 2 families. In the first family, the process of building base learners is separated from the process of combining all the base learners. In the first stage, we need to develop a population of base learners. In the second stage, we need to

combine these base learners to build a composite classifier. Furthermore, we can build several composite classifiers and combine them to build a mega model, and this process is called stacking [24]. The stacking methods will be further discussed in Chapter 6.

One popular method to combine many different classifiers is to ask them to vote for the results. The process of sampling with replacement from the original training data is called Bagging [6]. The random forest algorithm [7] uses CART as base learner, uses this bagging technique to build trees with different training data, and ask them to vote for the result. The class with majority votes will be the final result.

In the second family, the process of building base learners and final model are combined together in an iterative way. A widely used ensemble method in this family is Boosting, which can be viewed as a weighted sum of base learners in the functional space of classifiers [25]. In boosting, the later base learner treated the residual of previous models as target, so the classifiers have to be trained sequentially. A weight is assigned to the new base learner to be added into the final model.

The ensemble methods is very important to tree based methods in the sense of performance improvement. The random forest, gradient boosting trees and regularized greedy forest are simpler than deep neural networks and can be used on any data, which make them very popular in classification problems.

1.2 Imbalanced Class Problem

These aforementioned classification algorithms work well when the size of different classes are similar to each other, but not so if the size of different classes are severely skewed. For example, in fraud detection, the percentage of credit card fraud in all card transactions is less than 0.1% [8]. If we directly apply classification algorithms on this kind of imbalanced problems, the classifiers will tend to assign every observations to the majority class, especially when the two classes are hard to differentiate. In this case the accuracy is very high because most observations are correctly classified, however the classifiers are not differentiating the observations.

Take random forest for example. Suppose there is no relationship between the predictors and the response, and each tree is allowed to grow without restriction, in the end each leaf will have only one observation, and the probability for a new random observation falling into any leaf would roughly be the same. Therefore the probability of a single tree giving a minority prediction will be the same to the percentage of the minority class in the training data. After the majority voting in the random forest algorithm, it is very unlikely for a random forest model to give a prediction of the minority class. This situation happens in real world problems where the relationship between the predictors with the response is not strong enough.

1.3 Solutions to Imbalanced Class Problem

Much has been done to deal with the imbalanced class problem. They are trying to solve the problems from different directions, which can be divided to four families. The performance of each method depends on the specific problem given.

1.3.1 Pre-processing Methods

The idea of pre-processing methods is to modify the training data before the modeling step. The sampling methods belong to this category, which can be divided into under-sampling and over-sampling. The idea of under-sampling is to shrink the number of observations in the majority class to a smaller figure, hence it is a problem of dimension reduction. One under-sampling method is based on clustering [9]. Clustering methods are used to divide data to many small clusters, and the majority class is sampled with different frequency from each cluster.

The idea of over-sampling is to enlarge the number of observations in minority class, for example, bootstrap can be used to enlarge the minority class [26]. However, this approach does not change the point location of the minority class in the feature space. The SMOTE algorithm [10] is designed to generate synthetic data in the feature space for minority class. The key idea of the SMOTE algorithm is to use several closest minority observations in the feature space to synthetically generate a new minority observation, whose features are determined by a weighted average of the corresponding feature of the real observations. In this case the SMOTE algorithm is able to generate new minority observations in the feature space, which will render the minority class denser in certain zones.

1.3.2 Post-processing Methods

The focus of post-processing methods is to modify the model after training. For single model, the processing procedure differs by algorithm. A simple example is to choose different threshold for the score given by the classifier. Castro and colleagues [11] proposed a method to modify the weights of SVM after the model is built.

Some ensemble methods that designed to deal with imbalanced data problem belong to this category. Galar and team [12] provided a detailed overview on ensemble methods for class imbalance problem, which further divided this category to several subcategories and introduced many modified bagging and boosting based methods.

1.3.3 Cost Sensitivity Learning

The focus of cost sensitivity learning is to assign cost to false positive and false negative, and use algorithms to minimize the cost function related to false positive and false negative. Elkan [13] provided a general introduction to this field.

A popular method with similar idea is to assign weight to each observations, and minimize the weighted objective function. This method is comparatively simpler and is implemented in many packages.

1.3.4 Algorithm Specific Approach

Methods in this category modify the original model training procedure to handle the imbalanced problem. The ROC tree and ROC random forest [17] belongs to this category and will be introduced in the next several chapters.

Note that all these approaches can be combined together to achieve a better performance. For example, this is a sampling stage while building the trees in random forest, and we can assign a higher weight to the minority observations so they have a larger chance to be selected. In this case the whole model still benefit from the full training data, with each tree having a more balanced class distribution.

1.4 The ROC Curve and the Area Under ROC Curve (AUC)

As it's mentioned in Section 1.2, the total accuracy is no longer a good measure for the performance of classifiers when the class size is severely skewed, because it will be very high when we assign every observations to the majority class. Therefore we need to use a different measure for imbalanced data classification. The ROC curve [14] is widely used for evaluating the performance of binary classification result. It is a plot with y axis as true positive rate (Sensitivity) and x axis false positive rate (1- Specificity). Each point on the ROC curve represents a threshold, and by choosing that threshold we can compute the corresponding sensitivity and specificity for the data set. The point on the plot that indicates best performance is (0, 1), which means the true positive rate is 1 and false positive rate is 0. In this case the classifier perfectly identifies the two classes. Normally a classifiers could not achieve this point, and there are several method to determine the best threshold of a given classifier. For example, we can choose the point on ROC curve that is closest to (0, 1) to be the threshold.

To compare the ROC curve of two classifiers, we need another measure, which is the area under the ROC curve (AUC). The AUC is defined as the area bounded by the ROC curve, the positive x axis and the line passing through (1,0) and (1,1). AUC is a measure that considers both Sensitivity and Specificity of a classifier, and it is frequently used as a performance indicator for classifiers in imbalanced data classification.

One thing specific about AUC is that AUC doesn't depend on the threshold and it is generated by the score of prediction. The **Algorithm 1** below illustrates the process of calculating the AUC (code written in R). Note that the AUC given by **Algorithm 1** may be less than 0.5, which is information needed for future computation.

Algorithm 1 AUC_calculation

Input(s): $\boldsymbol{p}$, the prediction vector; $\boldsymbol{\omega}$, the corresponding label vector
Output(s): Area Under Curve (AUC)

let data frame X=cbind($\boldsymbol{\omega}$, $\boldsymbol{p}$)
Sort X by column $\boldsymbol{p}$**:** X=X[order(X[,2]),]
Get the sorted unique values of $\boldsymbol{p}$: uniq_values=sort(unique($\boldsymbol{p}$))
total_true=length(which($\boldsymbol{\omega} == \mathbf{1}$))
total_false=length($\boldsymbol{\omega}$) – total_true
tp_array=rep(0, length(uniq_values))
fp_array=rep(0, length(uniq_values))
auc_result=0
for i in 1:uniq_values:
    indice=which(p<=uniq_values[i])
    tp_array[i] = which(X[indice,1]==1)/total_true
    fp_array[i] = which(X[indice,1]!=1)/total_false
    if $i == 1$ **then**
        auc_result= $\frac{\text{tp\_array[i]} * \text{fp\_array[i]}}{2}$
    else
        auc_result=auc_result+(fp_array[i]-fp_array[i-1])*(tp_array[i-1]+tp_array[i])/2
    end if
end for
return auc_result
end

Chapter II: Binary ROC Tree

2.1 Brief Introduction of CART

2.1.1 The Structure of a CART

The Classification and Regression Tree (CART) [5] is a classifier with binary tree structure. Every tree is a combination of nodes. Each node has the following contents: (1) the feature used in this node (2) the threshold of that feature (3) left child node and right child node. (4) score of this node. Figure 1 below shows an example of a classification tree.

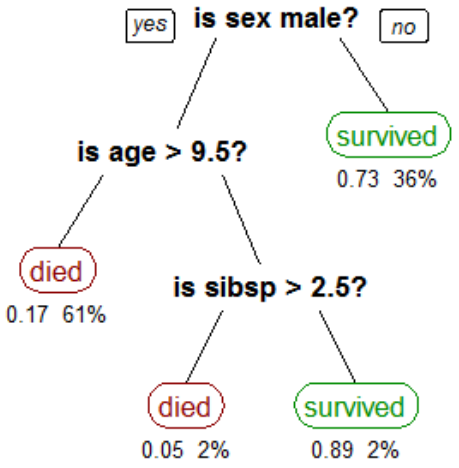

Figure 1: Classification Tree Example from Wikipedia
https://en.wikipedia.org/wiki/Decision_tree_learning

In Figure1, in the root node, the feature is “sex”, and the threshold is “male”. If the “sex” of an incoming observation is “male”, then it will proceed to the left child, otherwise it will go to the right child. The right child of the root node is an end node, whose score is 0.73. It can be

regarded as the probability of survival, and we can further choose a threshold for the score to assign the label prediction of this end node. In this example, we can set the label to be 0.5, so end nodes with score larger than 0.5 is labeled as survived. The left child of the root node is not an end node, so there is another feature and threshold to compare. Each observation will finally reach an end node, and returns the score of that end node as prediction.

2.1.2 Feature and Threshold Selection

The selection of feature and threshold in each node is based on the Gini Impurity of the left and right child given by that split. The Gini Impurity, I_G, for a data set with m classes is defined as

$$I_G = \sum_{k=0}^{m} \pi_k (1 - \pi_k),$$

where π_k is the percent of class k in this set.

In CART [5], the algorithm will go over every possible split of every feature that can divide the data into two parts, and select the feature and threshold that minimize the Gini Impurity sum of the left child and the right child. This method is computational expensive since it will find each unique value along each feature and check the Gini Impurity of this split. Moreover, it was shown that features with more splits would be more likely to be selected because it has more possible splits, which will increase the bias of the tree model [30]. Loh and Shih proposed a method based on clustering to solve this problem [29].

For categorical variable, the traditional method is to check every possible split of two sets. In this case, suppose there are N categories for one categorical variable, we need to check $\sum_{i=1}^{[\frac{N}{2}]} C_N^i$ different splits and select the best one from which. To simplify this process for

categorical variable in binary classification, CART sorts all the levels of that categorical variable by the percentage of class 0, and then treats them as ordinal variables. In this case the method for ordinal variables can be used on categorical variable, which speeds up the computation.

2.1.3 Advantages and Disadvantages of CART

CART has many advantages and disadvantages compared to other classification algorithms. The foremost advantage is its simple structure and design. Compared to SVM and neural network which are designed to minimizing a global cost function, CART is more like a greedy algorithm. For each node, CART selects the current best split instead of the global best split. Therefore, if we want a tree that has the minimum Gini Impurity sum, CART may not give the best one. On the other hand, the greedy design also makes CART very fast and efficient, which is desirable for base learners in ensemble methods.

The second advantage of CART is its wide applicability. CART requires very few data preprocessing steps, including outlier filtering, variable transformation, variable selection and interaction checking. The splitting method makes CART insensitive to outliers. Even if the predictor does not have any relationship between the response variable, CART is more likely to select a split in the dense area since there are more possible splits. CART is also immutable to variable transformation which maintains the relative order of each observation, because the Gini Impurity depends only on the proportion of classes in the two child nodes. As for variable selection, CART will automatically choose the feature and split. If a variable is not related to the response variable, it will not be selected in the node. CART also considers interaction between different variables, although it does not check for linear interactions like generalized linear regression models. Additionally, the CART models are easy to interpret.

There are also many disadvantages to CART. First of all, CART can only make one split at each node, which makes the decision boundary of the tree not as smooth as other classifiers. Therefore a single node in CART can only divide observations to two categories instead of learning any linear or non-linear patterns in the data. As CART grows deeper, the tree can learn more complicated pattern. For example, suppose variable Y is linearly related to variable X and we want to predict Y using X. CART will be able to learn a linear relationship by splitting multiple times on the X variable, and the classifier will be a polygonal line that approximates the linear relationship. This design makes CART easier to overfit and less robust, since the prediction is highly dependent upon the split threshold, and the split threshold depends on only a few points in the data set. To deal with this problem, one approach is to fit linear regression in the leaf nodes [31], so that the final classifier will be smoother in the feature space.

Secondly, the design of CART doesn't have explicit regularization, although it contains much implicit regularization. For example, the maximum depth, setting the deepest depth of the tree, regulates the maximum level of interactions and therefore controls how complicated the classifier would be. Another similar criterion is the maximum number of nodes. The minimum leaf size threshold sets the minimum number of observations in one leaf. A split has to make the size of both child nodes larger than this threshold to be qualified. The minimum split size threshold sets the minimum number for a node to continue splitting. Both minimum split size and minimum leaf size will ensure that the tree will not learn too complicated structure from a small size of data. Finally, the minimum deduction of Gini threshold sets the minimum Gini Impurity deduction after the split. If there doesn't exist any split that has Gini Impurity deduction larger than this threshold, the node will not split and become an end node.

All the criteria mentioned above are implicit regularization measures aiming to control the complexity of the tree classifier. However, there doesn't exist one uniform penalty term that can control the regularization strength in CART. The more the parameters are, the harder it is to tune the model and control model overfit.

Most importantly, the performance of a single CART is usually minor than logistic regression, SVM and neural network, so it's normally used with ensemble methods. This is also a result of its simple design.

2.2 ROC Tree Introduction

Like other classification algorithms, CART doesn't work well on severely biased data. The Gini Impurity will always be very small whichever split is chosen. Therefore, an attribute selection method based on AUC was first proposed in 2002 [15]. The resulting decision tree [15] has multiple child nodes and multiple split points, which is different from the CART framework. Another splitting method based on AUC for binary decision tree was proposed in 2008 [16]. At each node, the feature that gives the highest AUC was first chosen. Then the misclassification rate for the left child and the right child was computed for every possible split of that feature, and the split with the lowest misclassification rate was chosen for that node.

In his doctoral thesis published in 2015, Bowen Song proposed another node attribution selection method [17] for binary classification and also extended the ROC tree using the random forest (RF) framework. In his ROC RF [17], the feature with the highest AUC was selected, and the threshold that gives the largest harmonic mean of Sensitivity and Specificity was chosen. This approach considers both Sensitivity and Specificity and thus yields a better result.

2.2.1 Node Attribute Selection

In Bowen's ROC tree, the attribute that gives the largest AUC was selected. This step is described below by **Algorithm 2**. Note that this is Algorithm 9 in his thesis [17].

Algorithm 2 Node_Attribute_Selection

Input(s): $\boldsymbol{X}$, the matrix of training examples; $\boldsymbol{\omega}$, the corresponding label vector; P and N, the number of positive and negative examples; $\boldsymbol{\mathcal{A}}$, the attribute set

Output(s): $\mathcal{A}$, attribute with the highest AUC; α, (TPR, FPR) and $thres$, corresponding splitting direction, ROC points and threshold vector

$max_{\mathcal{A}} \leftarrow 0$
$max_{\alpha} \leftarrow 1$
$max_{\mathrm{ROC}} \leftarrow NULL$
$max_{\mathrm{thres}} \leftarrow -\mathrm{Inf}$
for each attribute $\mathcal{A}_i \in \boldsymbol{\mathcal{A}}$ **do**
    $temp_{ROC}, temp_{thres} \leftarrow$ ROC_generating$(\mathcal{A}_i, \boldsymbol{\omega}, P, N)$
    $temp_{AUC} \leftarrow$ AUC_calculation$(temp_{ROC})$
    $temp_{\alpha} \leftarrow 1$
    if $temp_{AUC} < 0.5$ **then**
        $temp_{AUC} = 1 - temp_{AUC}$
        $temp_{\alpha} = -1$
    end if
    if $temp_{AUC} >$ max_$\mathcal{A}$ **then**
        $max_{\mathcal{A}} = temp_{AUC}$
        $max_{\alpha} = temp_{\alpha}$
        $max_{\mathrm{ROC}} = temp_{ROC}$
        $max_{\mathrm{thres}} = temp_{thres}$
    end if
end for
return $max_{\mathcal{A}}, max_{\alpha}, max_{\mathrm{ROC}}, max_{\mathrm{thres}}$
end

2.2.2 Node Threshold Selection

After the attribute is selected for this node, the threshold that gives the largest F1 score (harmonic mean of Sensitivity and Specificity) is selected. This process is described below by **Algorithm 3**, which is Algorithm 10 in [17].

Algorithm 3 Node_Splitting_threshold

Input(s): $\mathcal{A}_\alpha$, $\mathcal{A}_{(TPR,FPR)}$ and $\mathcal{A}_{thres}$, the splitting direction, ROC points and corresponding threshold associating with attribute $\mathcal{A}$, which has the largest AUC
Output(s): $thres$, final splitting threshold for attribute $\mathcal{A}$

1. $cur_{hmean} \leftarrow -\text{Inf}$
2. $thres \leftarrow -\text{Inf}$
3. **for** each ROC points $(TPR_i, FPR_i) \in \mathcal{A}_{(TPR,FPR)}$ **do**
4. **if** $\mathcal{A}_\alpha == 1$ **then**
5. $temp_{hmean} = \text{Harmonic\_Mean}(TPR_i, FPR_i)$
6. **else then**
7. $temp_{hmean} = \text{Harmonic\_Mean}(1 - TPR_i, 1 - FPR_i)$
8. **end if**
9. **if** $temp_{hmean} > cur_{hmean}$ **then**
10. $cur_{hmean} = temp_{hmean}$
11. $thres = \mathcal{A}_{thres_i}$
12. **end if**
13. **end for**
14. **return** $thres$
15. **end**

2.2.3 ROC Tree Building

The node building process is completed by **Algorithm 2** and **Algorithm 3**. The tree building process is a recursive process of node building. It builds the current node and uses the data satisfying the criteria to build the left child node and uses the data that doesn't satisfy the criteria to build the right child. This process for ROC tree is described below by **Algorithm 4**, which is Algorithm 11 in [17].

Algorithm 4 ROC_tree

Input(s): $\boldsymbol{X}$, the matrix of training examples; $\boldsymbol{\omega}$, the corresponding label vector; P and N, the number of positive and negative examples; $\boldsymbol{\mathcal{A}}$, the attribute set
Output(s): $\mathcal{T}$, the final ROC tree

1. **if** $(\boldsymbol{X}, \boldsymbol{\omega}, P, N)$ contains no record **then**
2. **return** a single NULL node
3. **end if**
4. **if** $(\boldsymbol{X}, \boldsymbol{\omega}, P, N)$ consists of records all with the same label value **then**
5. **return** a single leaf node labeling this value
6. **end if**
7. $(\mathcal{A}_\alpha, \mathcal{A}_{(TPR,FPR)}, \mathcal{A}_{thres}) \leftarrow \text{Node\_Attribute\_Selection}(\boldsymbol{X}, \boldsymbol{\omega}, P, N, \boldsymbol{\mathcal{A}})$

8. $thres \leftarrow$ Node_Splitting_threshold($\mathcal{A}_\alpha$, $\mathcal{A}_{(TPR,FPR)}$, $\mathcal{A}_{thres}$)
9. **set** $(\boldsymbol{X}, \boldsymbol{\omega}, P, N)_{left}$ as the negative child node and $(\boldsymbol{X}, \boldsymbol{\omega}, P, N)_{right}$ as the positive child node based on $\mathcal{A}_\alpha$ and $thres$
10. Recursively applies ROC_tree to subsets $\left(\boldsymbol{X}, \boldsymbol{\omega}, P, N, \boldsymbol{\mathcal{A}_{left}}\right)_{left}$ and $\left(\boldsymbol{X}, \boldsymbol{\omega}, P, N, \boldsymbol{\mathcal{A}_{right}}\right)_{right}$ until they are empty or the stopping criteria are met. Here $\boldsymbol{\mathcal{A}_{left}}$ and $\boldsymbol{\mathcal{A}_{right}}$ are the randomly selected attributes subset for the left tree and right tree.
11. **return** ROC tree $\mathcal{T}$
12. **end**

2.3 Limitation of the Current ROC Tree

The ROC tree described above designed to solve the binary imbalanced data classification problem outperforms the original CART as a base learner on many imbalanced datasets [17]. It also removes one disadvantage of CART that features with more splits would be selected. In ROC tree, the attribute is selected based on AUC, and more split points on attribute will only make the ROC curve smoother. Therefore it will have less bias than CART in the attribute selection process.

However, there were still some limitations to this ROC tree design. For example, it only works for binary classification, so a method that can deal with multiple classes needs to be proposed. To solve this problem, a new split selection method based on both AUC and the idea of One vs. All technique is proposed in Chapter 3, which works for mutli-class classification problems, and is a natural expansion of the binary ROC tree in [17].

Chapter III: Multi-Class ROC Tree

3.1 Review of Concepts

3.1.1 Notations

Table 1: Definition of TP, FP, FN and TN

		Reality	
		Positive	Negative
Prediction	Positive	True Positive (TP)	False Positive(FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 1 above shows the definition of true positive, false positive, false negative and true negative.

- True Positive (TP): The number of observations predicted as positive is also positive in reality.
- False Positive (FP): The number of observations predicted as positive is negative in reality.

- False Negative (FN): The number of observations predicted as negative is positive in reality.
- True Negative (TN): The number of observations predicted as negative is also negative in reality.

Based on these definitions, we can define some measures for the performance of a classifier based on a set of test data.

- True Positive Rate (TPR): also called Sensitivity, Recall, defined as $\frac{TP}{TP+FN}$
- False Positive Rate (FPR): defined as $\frac{FP}{FP+TN}$
- Specificity: equals to 1-FPR, defined as $\frac{TN}{FP+TN}$
- Precision: defined as $\frac{TP}{TP+FP}$

3.1.2 Harmonic Mean

Harmonic mean is one of the three Pythagorean Means. It is normally used to calculate the mean of rates. The Harmonic Mean of $x_1, \dots, x_n$is defined as

$$HM(x_1, \dots, x_n) = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

When there are only two numbers Sensitivity and Specificity, the equation can be written as

$$HM(Sensitivity, Specificity) = \frac{Sensitivity * Specificity}{Sensitivity + Specificity}$$

The Harmonic mean of Precision and Recall is called F1 score [18], which is a measure for binary classification.

The following **Algorithm 5** shows the process of calculating the harmonic mean.

Algorithm 5 Harmonic_mean

Input(s): $\boldsymbol{V}$, The vector of contents to calculate harmonic mean
Output(s): H, the harmonic mean
1: H=0
2: **for** i in 1:length($\boldsymbol{V}$)
3: $H = H + \frac{1}{\boldsymbol{V}[i]}$
4: **end for**
5: $H = \frac{length(\boldsymbol{V})}{H}$
6: **return** H
7: **end**

3.1.3 Multi-Class Classification Technique

Some of the binary classification algorithms can be expanded to multi-class naturally. For example, in CART, the Gini Impurity for two classes can be expanded to multiple classes, so we can still choose the feature and split that minimizing the multi-class Gini Impurity.

For algorithms that can't be expanded naturally, there are two kinds of general strategies to reduce multi-class classification problem to binary classification problem [19].

The first strategy is the One vs. All strategy (also called One vs. Rest). For each class, we use all the data to train a classifier treating that class as class 1 and every other classes as class 0. Therefore we will have n classifiers for a classification problem of n classes. In the prediction process, all the classifiers will be applied to incoming observations, and the class with the largest score will be chosen as the prediction.

The second strategy is the One vs. One strategy. We need to train a classifier for each pair of classes, using only the data belongs to these two classes. Therefore we will have $\frac{n*(n-1)}{2}$ (C_n^2) binary classifiers for a classification problem of n classes. In the prediction process,

all the classifiers will be applied to incoming observations, which returns their predictions for this observation. Then a voting process is applied, and the class that gets the most positive votes will be chosen as the result.

These two techniques are mature and widely used in algorithms like SVM for multi-class classification.

3.2 Node Split Selection Method for Multi-Class ROC Tree

The One vs. All and One vs. One framework work well on balanced classification problems. However, they can't be directly applied to severely biased data, which may leads to several severely imbalanced binary classification problems, and the performance of voting may vary. On the other hand, this method will produce multiple trees only to perform as one multi-class classifier, which is not memory and computationally efficient. Therefore we incorporate the idea of One vs. All technique in the node attribute selection step.

There are two steps in the node selection process for multi-class ROC Tree.

3.2.1 Attribute Selection

Step 1, find the feature having the largest sum AUC for all classifiers. In binary ROC tree, the feature that yields the highest AUC was selected. In multi-class ROC tree, we calculate the AUC for the entire one vs. all classification problems, and select the feature with the largest sum of AUC. **Algorithm 6** below states this process. Note that the *AUC_sign* variable tells whether the AUC is smaller than 0.5 or not. If it is smaller than 0.5, the observation that smaller than the threshold should be classified to negative in the One vs. All classifier.

Algorithm 6 Node_Attribute_Selection

Input(s): $\boldsymbol{X}$, the matrix of training data; $\boldsymbol{\omega}$, the corresponding label vector; $\boldsymbol{\mathcal{A}}$, and the set of column indices in $\boldsymbol{X}$ as available attributes

Output(s): $\mathcal{A}$, column indice of attribute with the highest total AUC; *max_auc*, the AUC sum of this feature for all One vs. All classifiers; *AUC_sign*, the vector of signals to tell whether AUC for this class is smaller than 0.5

```
max_auc=0
uniq_class=unique(ω)
A=0
AUC_sign=rep(0, length(uniq_class))
for i in A
    tmp_auc=0
    for j in 1:length(uniq_class)
        tmp_label=rep(0,length(ω))
        tmp_label[which(ω == uniq_class[j])]=1
        auc_result= AUC_calculation(X[,i], tmp_label)
        if auc_result<0.5 then
            auc_result=1- auc_result
        tmp_auc=tmp_auc+ auc_result
    end for
    if tmp_auc>max_auc then
        max_auc=tmp_auc
        A = i
    end if
end for
for j in 1:length(uniq_class)
    tmp_label=rep(0,length(ω))
    tmp_label[which(ω == uniq_class[j])]=1
    auc_result= AUC_calculation(X[,A], tmp_label)
    if auc_result<0.5 then
        AUC_sign [j]=1
    end if
end for
return A, max_auc, AUC_sign
end
```

3.2.2 Threshold Selection

Step 2, select the split threshold with the largest harmonic mean of all Sensitivity and Specificity. After the feature is chosen, we calculate Sensitivity and Specificity for all One vs. All classifiers on each possible split, and select the split with the largest harmonic mean of all

those Sensitivity and Specificity. **Algorithm 7** below states this process. Note that the Sensitivity and Specificity are related to the *AUC_sign*. If *AUC_sign* is smaller than 0.5 for a One vs. All classifier, we need to classify observations smaller than that threshold to negative, and if not, we need to classify observations smaller than that threshold to positive. This would change the result of Sensitivity and Specificity.

Algorithm 7 Node_Threshold_Selection

Input(s): $\boldsymbol{X}$, the matrix of training data; $\boldsymbol{\omega}$, the corresponding label vector; $\mathcal{A}$, column indice of attribute with the highest total AUC; *AUC_sign*, the vector of signal to tell whether AUC for this class is smaller than 0.5

Output(s): *thre_result*, the threshold of for this split

```
thre_result=0
uniq_class=unique(ω)
tp_array=rep(0,length(uniq_class))
fp_array=rep(0,length(uniq_class))
uniq_splits=sort(unique(X[,A]))
total_true=rep(0,length(uniq_class))
for i in 1:length(uniq_class)
    total_true[i]=length(which(ω ==uniq_class[i]))
end for
total_false=rep(nrow(X),length(uniq_class)) – total_true
max_harmean=0
for i in 1:length(uniq_splits)
    indice=which(X[,A]<uniq_splits[i])
    for j in 1:length(uniq_class)
        if AUC_sign[j]==0 then
            tp_array[j] = length(which(ω[indice]==uniq_class[j]))/total_true[j]
            fp_array[j] = length(which(ω[indice]!=uniq_class[j]))/total_false[j]
        else
            tp_array[j] = length(which(ω[indice]!=uniq_class[j]))/total_true[j]
            fp_array[j] = length(which(ω[indice]==uniq_class[j]))/total_false[j]
        end if
    end for
    tmp_harmean=Harmonic_mean(c(tp_array,rep(1,length(uniq_class))-fp_array))
    if tmp_harmean > max_harmean then
        max_harmean = tmp_harmean
        thre_result=uniq_splits[i]
    end if
end for
return thre_result
end
```

3.3 Stopping Criteria for Multi-Class ROC Tree

The tree building process is a greedy algorithm based on the divide and conquer framework. At each node, we select the split based on the split selection method presented in Section 3.2, and build the left child with data that satisfy this criterion, and build the right child with data that doesn't satisfy this criterion. We need a stopping criterion so that this recursive process will stop at some suitable point, and prevent the overfitting of the data.

There are two major purposes to the stopping criteria -- efficiency and regularization. Firstly, if there is no stopping criteria, the tree will grow until there is only one training observation in the end node. In this case the tree will be very large, deep, and overfitted since the final result highly depends on the individual training observations. Secondly, the stopping criterion can be used as regularization for CART, although they are implicit regularization. These criteria help build shallower trees so that little or no tree pruning procedure will be needed.

The usual stopping criteria include but are not limited to

- Max depth of the tree. The tree will stop growing after reaching this depth.
- Max nodes of the tree. The tree will stop growing after having this number of nodes.
- Min split size. The node will stop to split if the training observations in this node are smaller than this amount.
- Min node size. Splits that make either child node have observations less than this threshold will be omitted.
- Min Gini deduction. Splits that reduce Gini less than this threshold will be omitted.

Some of the criteria are similar to each other. For example, the max depth of the tree is similar to the max nodes of the tree. Both of them try to control the size of the tree regardless of the amount of training data. On the other hand, the following three criteria are also similar to each other. All of them is related to the amount of the data and are based on node level instead of controlling the size of the tree.

Some of the stopping criteria for CART can also be applied to the Multi-Class ROC Tree, and the following conditions are the stopping criteria for Multi-Class ROC Tree.

- When the maximum tree depth is reached.
- When a node is pure. In other words, there is only 1 class in this node.
- When the number of observations in this node is smaller than a certain threshold.

Therefore a maximum depth and a minimum number of observations for each node need to be defined before building the tree.

3.4 Multi-Class ROC Tree Algorithm

Algorithm 8 states the recursive algorithm for multi-class ROC Tree. This code is designed for R, so a data frame is used to store the tree and linked by id. In C++, the tree can be stored using struct and linked by pointers. To prevent overfitting, only part of the data is used to train the node, and the rest of the data can be used as out of bag score to evaluate the performance. A stopping can be defined based on the difference of training data score and out of bag score. The node score is calculated based on the percentage of each class in training data in this node, and the

Algorithm 8 Multi_Class_ROC_Tree

Input(s): $\boldsymbol{X}$, the matrix of training data; $\boldsymbol{\omega}$, the corresponding label vector; $\boldsymbol{\mathcal{A}}$, the set of column indices in $\boldsymbol{X}$ as available attributes; cur_id, the current id of this node; max_depth, the maximum depth for this tree; min_leaf, the minimum number of observations on each node; train_ratio, the ratio of data to be used for training; $\boldsymbol{N_a}$, the number of attribute allowed in each node

Output(s): *roc_tree*, the data frame of the multi-class ROC tree

1: *roc_tree*=data.frame(id=cur_id, split_var="", thre="",lchild=0,rchild=0,nodelabel= -1, nodescore="",oob_score="")
2: Randomly generate the training indices and testing indices of the rows of $\boldsymbol{X}$ based on the train_ratio
3: Calculate the nodesocre and nodelabel based on the training data $\boldsymbol{X}$[training_indice,]
4: Calculate the oob_score based on the testing data $\boldsymbol{X}$[testing_indice,] for comparison
5: Check whether the stopping criterion is met or not. **If** it is met, **return** *roc_tree*
6: Sample $\boldsymbol{N_a}$ features from $\boldsymbol{\mathcal{A}}$ and set them to be $\boldsymbol{\mathcal{A}'}$
7: Find the attribute $\mathcal{A}$ using training data $\boldsymbol{X}$[training_indice,], the attributes set $\boldsymbol{\mathcal{A}'}$, the label vector $\boldsymbol{\omega}$[training_indice] and function Node_Attribute_Selection (**Algorithm 6**)
8: Find the threshold *thre_result* using the attribute $\mathcal{A}$, training data $\boldsymbol{X}$[training_indice,] and corresponding label vector $\boldsymbol{\omega}$[training_indice] and function Node_Threshold_Selection (**Algorithm 7**)
9: Build the left child by calling Multi_Class_ROC_Tree (**Algorithm 8**) recursively. Let the left child id to be cur_id*2, left child max depth = max_depth – 1. Append the return dataframe with the data frame *roc_tree*
10: Build the right child based using same function. Let the left child id to be cur_id*2+1, right child max depth = max_depth – 1. Append the return dataframe with the data frame *roc_tree*
11: **return** *roc_tree*
12: **end**

After the tree is built and the dataframe is returned, a recursive algorithm can be used to get the prediction of new observations. The process is stated in **Algorithm 9**.

Algorithm 9 ROC_Tree_Prediction

Input(s): ROC_Tree, the tree built by Algorithm 5; $\boldsymbol{x}$, a new observation; pred_type, the type of prediction needed; id, the id of tree node

Output(s): $\hat{\omega}$, the predicted label for $\boldsymbol{x}$

1: Find the row in the ROC_Tree whose id equals to the input id
2: **If** the split_var and thre is empty **then**
3: **If** pred_type=="score" **then**
4: **return** the nodescore of this row as $\hat{\omega}$
5: **else**
6: **return** the nodeclass of this row as $\hat{\omega}$
7: **end if**

8: **end if**
9: Check whether $\boldsymbol{x}$ satisfy the condition of this row.
10: **If** $\boldsymbol{x}$ satisfy the split_var and thre **then**
11: call the function again with new id as the left child id
12: **else**
13: call the function again with new id as the right child id
14: **end if**
15: **return** the result as $\hat{\omega}$
16: **end**

3.5 Summary of Multi-Class ROC Tree

The multi-class ROC tree can also be applied to the binary classification problems where it's the same as the binary ROC tree algorithm proposed in [17]. In binary classification problems, the multi-class ROC tree consider both the AUC of class 1 vs. class 0 and the AUC of class 0 vs. class 1, which are the same after converting AUC to be larger than 0.5. Hence the multi-class ROC tree is a natural expansion of the binary ROC tree in [17], which can be regarded as a degenerate case of multi-class ROC tree. The following Table 2 is modified from Table 1, which shows the concept.

Table 2: Definition of TP, FP, FN and TN in Different Settings

		Reality	
		Positive (Negative)	Negative (Positive)
Prediction	Positive (Negative)	True Positive (TN)	False Positive (FN)
	Negative (Positive)	False Negative (FP)	True Negative (TP)

As we can see from the table, if we switch the definition of positive and negative, the true positive in the old setting will be the true negative, and vice versa. Recall that the x axis of the ROC curve is the false positive rate (1 – true negative rate) and the y axis of ROC curve is the true positive rate (1 – false negative rate). Therefore after switching the label, the x axis is plotting the false negative rate, which is 1 – true positive rate, and the y axis is plotting the true negative rate, which is 1 – false positive rate. It actually transposes the ROC curve plot, and the AUC of the new plot equals to 1 – AUC of the old plot. Since AUC is defined to be larger than 0.5, the AUC after switching label will be the same to the old one.

In reality, the performance of a multi-class ROC tree is slightly better than the performance of the original binary ROC tree because it randomly select part of the input training data to train each node. This method prevents the overfitting problem to some extent, and provides the out of bag score to compare with the node score, so that we can detect the overfitting problem in the training process. The simulation result shows that including this randomness will increase the performance of classification.

Chapter IV: ROC Random Forest

4.1 Random Forest Introduction

The random forest [7] was first proposed in the 1990s and soon became very popular. Compared to other classification algorithms, the idea of the random forest is very simple, but it turned out to be useful on many real world problems. On the other hand, the random forest can be regarded as an ensemble framework instead of one single classification algorithm.

4.1.1 Brief Introduction on Methodology

Generally speaking, the random forest is an ensemble of many trees. There is a "random" in its name because each tree in this forest is different. There are two type of randomness in random forest.

Firstly, the training data used to build each tree is randomly sampled with replacement from the whole data set. Secondly, at node attribute selection step of each tree, only a part of the possible features are selected to be possible split. In this design, the correlations of these trees are reduced. After that, the majority voting process is applied on these trees to obtain the final forest classifier. By doing so, the bias of the classifiers still remain the same, while the variance of the classifiers will be greatly reduced. Therefore the random forest performs much better than a single tree especially for the out-of-bag testing data.

An advantage of this design is that random forest will not become more overfitted when we increase the number of trees in the forest. Overfitting means the performance of a classifier is inconsistent between the training data and the testing data, when the training data and testing

data are from the same distribution. It happens when a classifier tries to find every detailed difference between the classes in the training data, so it would also take noise into considerations. However in random forest, the classifiers are built with different data and using different feature pools, so the correlation between each single tree is reduced. Furthermore, the majority voting process ensembles all the trees and reduce the variance of each single classifier. Therefore the variance of classifiers will likely to be reduced when new trees are added to the forest. However, to control the overfitting degree of a random forest model, it is important to have regularization parameters including max depth or max nodes in the model, which makes the performance of random forest classifier similar on both training and testing data.

4.1.2 Advantages of Random Forest

Besides the important "no overfitting by extra trees" feature, there are several other advantages to the random forest.

1 Scalability. Another important advantage of random forest is its speed. The idea of random forest is very simple and it is also very fast. Moreover, the trees in random forest can be fit independently. Hence we can parallel this process and make it even faster.

2 Few parameters. Also random forest has very few parameters, which is very important in application. When we are fitting Support Vector Machine, we need to find the best penalty (or slack variable) by cross validation, and determine the best kernel for this problem. In neural networks, the design of the network structure is very important. Including or excluding a hidden unit may totally change its performance. However, in random forest, the most important parameter is the number of trees if we are focusing on the testing data performance only. We can also define the maximum depth, minimum leaf size, percentage of data used for training, number

of features to select from, but they have comparatively lower importance to the performance on the testing data. Therefore we don't need to worry about parameter tuning while using random forest to maximize the model performance on testing data.

3 Low demand on data format. Before we use SVM/neural network or other algorithms that rely on distance, we need to normalize the data to prevent saturation. However tree doesn't require the data to be pre-processed. The variable and threshold selection of a tree perfectly dealt with this problem. Therefore we don't need to worry about the data before we use the random forest.

4 Variable Importance. Random forest calculates the variable importance in the training process, which can greatly help the variable selection process. Random forest can detect the non-linear relationship between the response and predictors, while the step regression assumes the monotone relationship. Random forest also considered the interaction between variables by putting them on different levels of a tree.

4.1.3 Disadvantage of Random Forest

The random forest algorithm also has a few disadvantages, with two being major disadvantages.

Firstly, as a complicated ensemble algorithm, random forest is able to achieve much better performance than a single tree and most simple classifiers. However, the interpretability of a classifier is reduced during this process. Unlike a single tree, it is hard to tell the exact impact of each variable in the random forest classifier. There are two measures developed in random forest to help this process, the variable importance and partial dependency plot.

The variable importance is calculated by performing permutation of each selected features in the forest, which requires extra calculation time. Then the importance of each feature is measured by the drop of performance after changing all that features in the classifier, and it is possible that different features have different order of importance because of different measures. However the importance ranking in different measures should be very similar, and we are able to have more understanding about the classifier by the variable importance.

The partial dependency plot can be used as a tool to understand the relationship between the response variable and selected feature, or the interaction between two features regarding the response variable. This plot ignores all other features and plots the relationship between the response and given features. We can have more understanding about the impact of the features on the response variable via the partial dependency plot. Furthermore, this information can be used to change features into splines or dummy variables and use simpler models to achieve similar performance.

The second disadvantage of random forest is its performance compared to other complicated classifiers. The performance of random forest is normally worse than gradient boosting trees, and the gradient boosting trees requires shallower trees than random forest to achieve a good performance. Therefore random forest is seldom used as a single model in data science competitions.

However random forest is still able to provide information including variable importance and less correlated predictions with other algorithms, hence it is frequently used in data science problems for exploratory data analysis or as a base model.

4.2 Representative Tree of Random Forest

One way to interpret a random forest model is to find and plot the most representative tree in the forest, so that we have the interpretability advantage of a single tree. This idea is first proposed in [45] along with several measures to define the distance of trees, and it was further developed in [44].

The key point in this method is to define the distance measure of trees. There are several methods proposed in these two papers, which are summarized below.

4.2.1 Distance Measures

A very intuitive idea is that two trees should be close if they give similar predictions. Therefore the first measure is based on the similarity of the prediction. This idea is proposed in [45] and the distance is based on the difference of predicted value of each tree.

Let T be the space of trees, and $y_i, i = 1 \dots n$ be the true response of selected testing data. For any $T_m, T_l \in T$, let y_i^m and y_i^l be the prediction of T_m, T_l on observation $i, 0 < i \leq n$. Suppose the distance between y_i^m and y_i^l is defined to be the square error, that $dist(y_i^m, y_i^l) = (y_i^m - y_i^l)^2$. In this case the distance of trees, d_0, could be defined as

$$d_0(T_m, T_l) = \frac{1}{n}\sum_{i=1}^{n} dist(y_i^m, y_i^l)$$

which is the average distance between all predicted values.

The advantage of this method is its simplicity, and the disadvantage is that the distance of each tree depends on the testing data.

Another idea is that we can define the distance based on the similarity of tree nodes. This idea is proposed in [45] and the distance is based on the different features used in the trees as following.

$$d_1(T_m, T_l) = \frac{\#\ of\ covariate\ mismatches\ between\ T_m\ and\ T_l}{\#\ of\ covariates\ in\ the\ study}$$

However, in random forest, the feature pool in each tree is different, which will introduce bias in this distance measure. Therefore this measure is not adopted.

The distance between the trees can also be defined by the similarity of terminal nodes. The similarity of terminal nodes is defined based on whether two observations would end up in same terminal node. If two observations fall in the same terminal node in one tree but not the other one, then the distance of the two trees is increased. The formula of calculating distance is shown below.

$$d_2(T_m, T_l) = \frac{\sum_{i>j}\sum_j |I_{T_m}(i,j) - I_{T_l}(i,j)|}{\binom{n}{2}}$$

where $I_{T_m}(i,j) = 1$ when observation i and observation j ends up in the same terminal node in tree T_m, and $I_{T_m}(i,j) = 0$ if otherwise.

The major disadvantage of this distance measure is its expensive computation. For n observations, it requires $O(n^2)$ computation time for each pair of trees. Suppose there are N trees in the random forest, the total time to compute the distance of each tree would be $O(N^2n^2)$, which is much more expensive than the distance by prediction method.

Furthermore, it is possible that two terminal nodes has the same label (for classification) or similar prediction (for regression), especially for deep trees. Therefore it is possible that two

observations end up in one terminal node in one tree but not the other, but both tree have same prediction for them. In this case, we probably should not increase the distance of the two trees.

There are other distance measures including some combination of the above measures, and it is hard to compare the effectiveness of different measures. Therefore in this dissertation, we adopt the distance by prediction to measure the distance between the ROC trees in the ROC random forest for simplicity.

4.2.2 Features of Representative Tree

Based on the current distant measure, we can derive several features of representative tree. First of all, since the distance is based on the predicted values, we can also calculate the distance between trees and the whole random forest model. Given that the representative tree is defined as the tree that has the smallest average distance with other trees, the representative tree is also very close to the random forest model.

Secondly, the distance between representative tree with the single classification and regression tree model (CART) is smaller than the average distance between CART and other trees in the random forest, because the representative tree should be the most unbiased tree in the random forest.

Thirdly, the distance between representative tree and the random forest model is related to the regularization strength of the random forest model. For example, a larger number of maximum nodes in random forest will increase the variance of the trees, which will increase the distance between the representative tree and the random forest model.

4.2.3 Representative Tree in UCI Census Income Data

Here we use the UCI Census Income data to explain the features of the representative trees in a classification setting. This data set has 32561 observations and each observation represents a person. There are 15 variables in this data set, including whether this person has a yearly income larger than 50K each year, which is the target of the classification random forest models.

The data set is randomly split into training data (50%) and testing data (50%). The random forest models and CART models are built with the same maximum nodes regularization, and the random forest model contains 100 trees. The distance between the trees is defined as the percentage difference on the testing prediction. The distance mean and standard deviation after 10 runs between CART, random forest and the representative tree of random forest in different situations is summarized below.

Table 3: Distance between Representative tree, Random Forest and CART in Different Regularization

Max number of Nodes	Avg Tree vs. CART Distance	Rep Tree vs. CART Distance	Rep Tree vs. RF Distance	CART vs. RF Distance
2^4	0.0818 ± 3.79e-03	0.0186 ± 9.61e-03	0.0165 ± 3.31e-03	0.0213 ± 3.15e-03
2^5	0.0836 ± 3.48e-03	0.0201 ± 8.77e-03	0.0176 ± 4.92e-03	0.0186 ± 2.60e-03
2^6	0.1103 ± 1.97e-03	0.0716 ± 9.90e-03	0.0286 ± 6.30e-03	0.0721 ± 2.87e-03
2^7	0.1129 ± 3.73e-03	0.0820 ± 2.32e-02	0.0425 ± 1.01e-02	0.0727 ± 8.87e-03
2^8	0.1203 ± 1.33e-03	0.0963 ± 1.33e-02	0.0499 ± 9.84e-03	0.0822 ± 5.40e-03
2^9	0.1247 ± 1.54e-03	0.0941 ± 6.65e-03	0.0472 ± 7.36e-03	0.0706 ± 2.54e-03

We can see from this table that the result agrees with the features of representative tree discussed in Section 4.2.2. The distance between representative tree and CART is smaller than

the average distance between CART and all the trees in the random forest, which means the representative is closer to CART than the average trees in the random forest.

As the maximum number of nodes increases, the regularization strength decreases, therefore there will be more variance in single trees in the random forest model. This increases the distance between representative tree and CART as well as the distance between representative tree and the random forest model.

However, the distance between the representative tree and the random forest is always smaller than the distance between CART and the random forest, which means the representative tree is better than CART in interpreting the random forest model.

4.3 Multi-Class ROC Random Forest Algorithm

The random forest framework can also be applied to the ROC trees to obtain the ROC random forest [17]. As for Multi-Class ROC random forest, the process stays the same, which is shown in Algorithm 10 below.

Algorithm 10 Multi_Class_ROC_Random_Forest

Input(s): $\boldsymbol{X}$, the matrix of training examples; $\boldsymbol{\omega}$, the corresponding label vector; $\boldsymbol{N_t}$, the number of trees to be generated; $\boldsymbol{N_a}$, the number of attributed needed for each node

Output(s): $\boldsymbol{\mathcal{F}}$, the final forest

1: **set** $\boldsymbol{\mathcal{F}}$ to NULL
2: **for** $i = 1$ **to** $\boldsymbol{N_t}$
3: sample a set of row indices from the original data with replacement noted as bagging_indice
4: train $tree_i$ by this sampled data $\boldsymbol{X}$[bagging_indice,], $\boldsymbol{\omega}$[bagging_indice,] with Multi_Class_ROC_Tree (**Algorithm 8**)
5: append $tree_i$ to $\boldsymbol{\mathcal{F}}$
6: **end for**
7: **return** $\boldsymbol{\mathcal{F}}$
8: **end**

However, we can't directly apply the original prediction method to ROC random forest since there will be some problems when the class sizes are highly skewed. To reduce the correlation between each single trees, the ROC tree in the random forest is using less data and fewer features. This design is acceptable when the class sizes are roughly balanced, however when the class sizes are highly skewed, the ROC tree will tend to assign each observation to the majority class. Therefore in the multi-class ROC random forest algorithm, the prior probability of each class is used to balance the prediction score, which is shown below in Algorithm 11.

Algorithm 11 ROC_Forest_Prediction

Input(s): $\boldsymbol{\mathcal{F}}$, the ROC forest; $\boldsymbol{x}$, a new observation; pred_type, the type of prediction needed; prior_prob, the prior distribution of each class

Output(s): $\hat{\omega}$, the predicted label for $\boldsymbol{x}$ or $\hat{p}$, the predicted probability for $\boldsymbol{x}$

1: set *score* as an empty data frame
2: **for** each $tree_i$ in $\boldsymbol{\mathcal{F}}$:
3: *score*$[i,]$=ROC_Tree_Prediction($tree_i$, $\boldsymbol{x}$,pred_type,id=1)
4: **end for**
5: sum up *score* for each class and calculate the score sum percentage for each class as $\hat{p}$
6: $\hat{p}$=$\hat{p}$/prior_prob/sum($\hat{p}$/prior_prob)
7: **If** pred_type=="score" **then**
8: **return** $\hat{p}$
9: **else**
10: $\hat{\omega}$=which.max($\hat{p}$)
11: **return** $\hat{\omega}$
12: **end if**
13: **end**

To sum up, the ROC random forest inherits the major advantages of the random forest including the representative tree. The correlation between each single tree is reduced by the randomness and all the trees can be built in parallel. Compared to random forest, the ROC random forest is specialized in dealing with imbalanced classification problem because it uses the ROC tree as the base learner.

The major improvement of Multi-Class ROC forest over the ROC forest proposed in [17] is that it is capable of processing multiple class classification problems. The introduction of sampling in the node attribute selection step also improved the robustness of ROC tree as well as the ROC random forest.

Chapter V: Performance of Multi-Class ROC Tree

5.1 Performance Measure of Multi-Class Classification Problems

The performance of imbalanced binary classification can be evaluated by the ROC curve and AUC. However the original ROC curve only works in binary classification problems, so new measures needs to be defined to evaluate the performance of multi-class classification.

5.1.1 Generalized Form of Notations

The notations in Section 3.1.1 can be extended to multi-classes naturally. In [20], these measures for l classes classification problem are defined as following.

- Average Accuracy: $\frac{\sum_{i=1}^{l}\frac{tp_i+tn_i}{tp_i+tn_i+fp_i+fn_i}}{l}$ which measures the average accuracy of each class
- Precision_{μ}: $\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} tp_i+fp_i}$ which measures the micro-average accuracy for positive predictions
- Recall_{μ}: $\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} tp_i+fn_i}$ which measures the micro-average true positive rate
- Precision_{M}: $\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fp_i}}{l}$ which measures the macro-average accuracy for positive predictions
- Recall_{M}: $\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fn_i}}{l}$ which measures the macro-average true positive rate

Note that in multi-class classification, the Precision_{μ} and Recall_{μ} will be the same since $\sum_{i=1}^{l} tp_i + fp_i$ is the sum of predicted positive for each class, which is the number of

observations, and $\sum_{i=1}^{l} tp_i + fn_i$ is the sum of real positive for each class, which is also the number of observations.

5.1.2 Volume Under Surface (VUS)

The VUS is an extension for AUC. In binary classification, the ROC curve shows the performance of a classifier on a plot with y axis as TPR and x axis as FPR for class 1. The FPR for class 1 can also be regarded as 1 – TPR for class 0. Therefore the axis in the ROC curve plot can be regarded as TPR for different class.

Therefore in multi-class classification problems, a coordinate system similar to ROC curve can be built to measure the performance of classifiers [21]. In an l class classification problem, the dimension of this system will be l and the axis is the TPR for each class. And the classifier will be a hyperplane in this space. The ROC curve in 2D is a degenerate form of this surface. Hence similar to AUC, the Volume Under Surface can be defined to evaluate the performance of a classifier, which involves the calculation of volume of a convex hull.

5.1.3 Point Selection for ROC Surface

The original ROC curve is generated by a vector of prediction scores in Algorithm 1. A threshold is defined and all observations with predicted score lower than this threshold are assigned to one class, which gives us one pair of true positive rate and false positive rate. Hence each threshold represents one point on the curve. However, in high dimension, we can't simply define and move the threshold from the minimum to the maximum of the prediction score. If we still find all possible points in high dimension, the VUS will be augmented since we need to find a convex hull and compute the volume under this convex hull.

Figure 2 below shows the difference of ROC curve with predicted score and single predicted label.

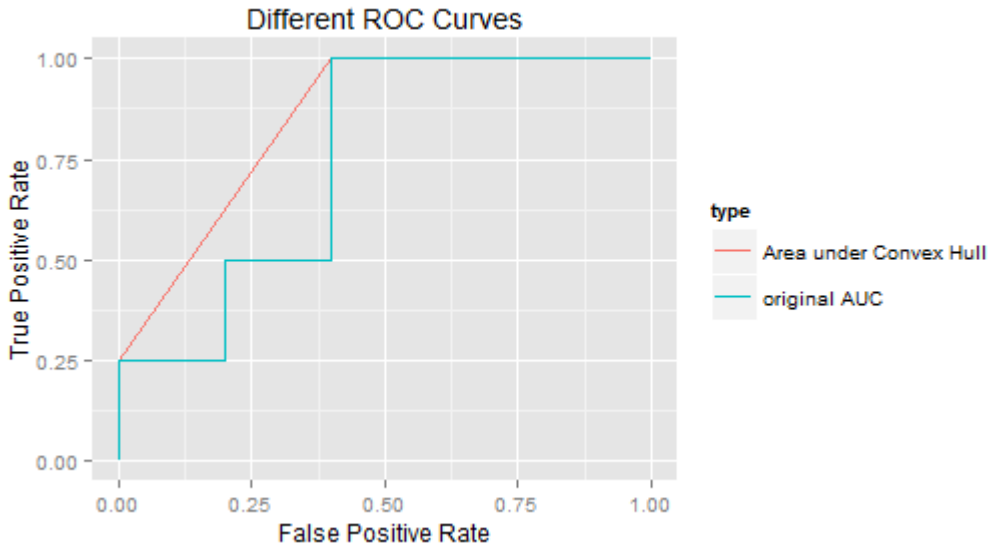

Figure 2: ROC Curve Plot. The area below the red line is considered to be the area under the convex hull of ROC curve, and the area below the blue line is the original area under ROC curve

In [21], the convex hull is found based on one vertex given by a vector predicted label and other trivial vertexes on the axis. For example, in a three-class classification problem, assigning all the data to class 1 will let the TPR for class 1 to be 1 and TPR for class 2 and 3 to be 0. Therefore the (1, 0, 0) vertex is naturally on the ROC Surface, and so is the vertex (0, 1, 0), (0, 0, 1). Another vertex is generated by the class prediction of the classifier, and the best situation is the point (1, 1, 1), which means the classifier has perfectly separated all the 3 classes.

5.1.4 Qhull Algorithm

After we found a bunch of points given by the prediction in ROC space, we need to find a convex hull of these points to get the ROC surface, then the Volume Under Surface also needs to be calculated.

This process can be done with the Qhull algorithm presented in [21]. The implementation of Qhull algorithm given by [22] is used for the evaluation in this dissertation.

5.2 Imbalanced Classification Simulation Settings and Results

There are many situations in multi-class classification, and here we compared the performance of Multi-Class ROC Tree, Multi-Class ROC Random Forest, CART, random forest and random guess on four situations. The number of classes is set to be 4, with 1 dominate class of 4750 observations (95% of the total data), and 2 minor classes of 100 observations (2% of the total data), and 1 rare class of 50 observations (1% of the total data). The random guess is generated by a multi-nominal distribution with probability as the class distribution. The CART is fit using the rpart package in R, and the random forest is fit using the randomForest package in R. All the performance is measured by out of bag testing data, which is generated using the same method as training data.

5.2.1 Setting 1: 2 dimensions, easy to differentiate

In this setting, we generate observations from Gaussian distribution with 2 dimensions so that it is easy to visualize the data. All the variables have a variance of 1. The Class 1 is centered

at (0, 0), the Class 2 is centered at (2, 2), the Class 3 is centered at (-2, -2), the Class 4 is centered at (-2, 2). Figure 3 shows the distribution of the data.

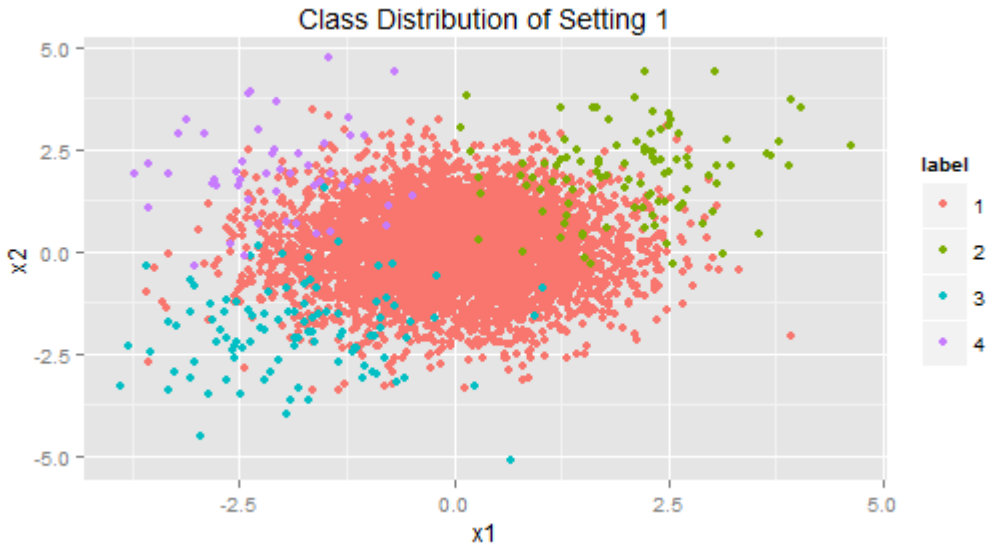

Figure 3: The distribution of simulated data in setting 1

The performance of Multi-Class ROC Tree, CART and random guess is showed in Table 4. The ROC tree has slight advantage over CART in Recall_M and VUS.

Table 4: Performance of Classifiers on Setting 1

	Average Accuracy	Precision_μ	Precision_M	Recall_M	VUS
ROC Tree	0.9809	0.9618	0.7345	0.5794	0.09657
CART	0.9817	0.9634	0.8110	0.5360	0.08933
Random Guess	0.9513	0.9026	0.2476	0.2473	0.04167

5.2.2 Setting 2: 2 dimensions, hard to differentiate

In this setting, we still generated observations that are normally distributed with variance 1, while the class centers are closer. The Class 1 is centered at (0, 0), the Class 2 is centered at (1,

1), the Class 3 is centered at (-1, -1), the Class 4 is centered at (-1, 1). Figure 4 shows the distribution of the data.

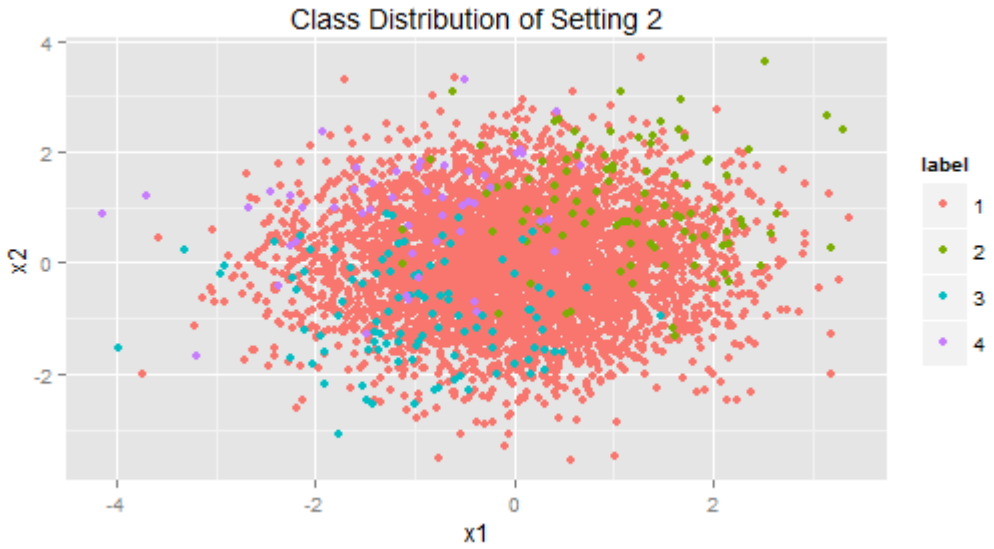

Figure 4: The distribution of simulated data for setting 2

The performance of Multi-Class ROC Tree, CART and random guess is showed in Table 5 below. The CART classified every observations to class 1 so the Precision_M is NA and the VUS is the same to random guess. In this case the Multi-Class ROC Tree outperformed CART.

Table 5: Performance of Classifiers on Setting 2

	Average Accuracy	Precision_μ	Precision_M	Recall_M	VUS
ROC Tree	0.9699	0.9398	0.3371	0.2767	0.04612
CART	0.975	0.95	NA	0.25	0.04167
Random Guess	0.9513	0.9026	0.2476	0.2473	0.04167

5.2.3 Setting 3: 10 dimensions

In this setting, high dimension data is used to check the performance of multi-class ROC Tree. All the data are normally generated with variance 1 and the centers for each class is listed below.

- Class 1: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- Class 2: (0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)
- Class 3: (-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5)
- Class 4: (-0.5, 0.5, -0.5, 0.5, -0.5, 0.5, -0.5, 0.5, -0.5, 0.5)

Table 6 shows the performance of each classifier on this setting. Similar to setting 2, CART predicted all test observations to class 1 so it is no better than random guess. Both ROC random forest and traditional random forest are built with max depth equals to 4 (max nodes equals to 2^4). The prior probability is used on Random Forest algorithm and the Naïve Bayes algorithm to get a comparable result with ROC Random Forest. After the transformation, the multi-class ROC random forest still has large advantage over traditional random forest, and we can see that the performance of multi-class ROC random forest is very close to Naïve Bayes. Note that Naïve Bayes is the optimal classifier in this problem because the underlying distribution of each classes is normal.

Table 6: Performance of Classifiers on Setting 3

	Average Accuracy	Precision_{μ}	Precision_{M}	Recall_{M}	VUS
ROC Tree	0.9676	0.9352	0.3215	0.2830	0.04716
CART	0.975	0.95	NA	0.25	0.04167
Random Guess	0.9513	0.9026	0.2476	0.2473	0.04167
ROC RF (100 trees)	0.6004	0.2008	0.2750	0.5610	0.09349

RF (100 trees)	0.9579	0.9158	0.3423	0.3393	0.05654
Naïve Bayes	0.7433	0.4866	0.2908	0.6217	0.1036

Table 7 shows the sensitivity and specificity of each class in detail for random forest, Naïve Bayes and ROC random forest.

Table 7: Sensitivity and Specificity of Classifiers on Setting 3

Algorithm	Random Forest		Naïve Bayes		ROC Random Forest	
Class	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
1	0.957	0.144	0.477	0.812	0.174	0.920
2	0.130	0.984	0.650	0.822	0.770	0.688
3	0.130	0.982	0.700	0.814	0.720	0.705
4	0.140	0.991	0.660	0.850	0.580	0.798

5.2.4 Setting 4, 10 dimension with noise

In this setting, each class has the same center and variance with setting 3, but 10 features with normal distribution centered at 0 with variance 1 is added into the problem as noise. There in this setting, each classifier is built with 20 features, but only 10 of them is related to the response.

The following table shows the performance of each classifier in this setting. Both ROC random forest and traditional random forest are built with max depth equals to 4 (max nodes equals to 2^4), and scores from both traditional random forest and Naïve Bayes model are divided by the prior distribution probability before calculating the label. We can see that the

VUS performance of ROC random forest is still better than random forest after the noise is added, and it's very close to Naïve Bayes, which is the optimal classifier in this problem.

Table 8: Performance of Classifiers in Setting 4

	Average Accuracy	Precision_{μ}	Precision_{M}	Recall_{M}	VUS
ROC Tree	0.9649	0.9298	0.3328	0.2815	0.04692
CART	0.975	0.95	NA	0.25	0.04167
Random Guess	0.9513	0.9026	0.2476	0.2473	0.04167
ROC RF (100 trees)	0.6015	0.2030	0.2730	0.5545	0.09241
RF (100 trees)	0.9591	0.9182	0.3026	0.2907	0.04845
Naïve Bayes	0.7621	0.5242	0.2900	0.5922	0.09871

The following table shows the sensitivity and specificity of each classifier.

Table 9: Sensitivity and Specificity of Classifiers on Setting 4

Algorithm	Random Forest		Naïve Bayes		ROC Random Forest	
Class	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
1	0.963	0.084	0.519	0.748	0.178	0.900
2	0.080	0.983	0.630	0.835	0.670	0.692
3	0.080	0.986	0.640	0.832	0.690	0.706
4	0.040	0.994	0.580	0.862	0.680	0.796

5.3 Balanced Classification Simulation

The results in Section 5.2 showed that ROC tree has advantage over CART, and ROC random forest has large advantage over traditional random forest in imbalanced classification problems. Therefore the ROC random forest is desirable when the class sizes are highly skewed. However, the performance of ROC random forest on balanced classification problems also needs to be compared to the classic random forest.

In this simulation, the size of each class is balanced. Each class has 1250 observations, and the centers are the same to setting 3.

- Class 1: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- Class 2: (0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)
- Class 3: (-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5)
- Class 4: (-0.5, 0.5, -0.5, 0.5, -0.5, 0.5, -0.5, 0.5, -0.5, 0.5)

Table 10 shows the performance of each classifier in this setting. We can see that the performance of ROC RF is very close to the traditional random forest and support vector machine, and the performance of a single ROC tree is slightly better than CART.

Table 10: Performance of Classifiers on balanced classification

	Average Accuracy	Precision_{μ}	Precision_{M}	Recall_{M}	VUS
ROC Tree	0.7511	0.5022	0.5097	0.5022	0.0837
CART	0.7306	0.4612	0.4402	0.4612	0.07687
Random Guess	0.6208	0.2416	0.2417	0.2416	0.04167
ROC RF (100 trees)	0.8121	0.6242	0.6052	0.6242	0.1040
RF (100 trees)	0.8172	0.6344	0.6265	0.6344	0.1057
SVM	0.8221	0.6442	0.6388	0.6442	0.1073

5.4 Performance on UCI Repository Data

The ROC random forest and ROC tree has shown its superiority in these simulated data sets which are hard to classify. Next we selected several data sets from UCI repository data to check the performance of ROC random forest on real world data sets. The algorithms we are comparing here are Ferri's ROC tree [15] and the classical SMOTE algorithm [10] combined with random forest classifier.

We choose four data sets listed below in Table 11. These data sets have different sizes, feature number and minority class percentage, representing different kinds of real world problems. All the four data sets are binary classification problems since the SMOTE algorithm can only deal with binary cases.

Table 11: Chosen binary classification UCI repository data set

Data Set Name	Observation Number	Feature Number	Minority Class Percentage
Letter Recognition: A	20000	16	3.95
Optical Recognition of handwritten digits: 0	5620	64	9.86
Pen-based Recognition of handwritten digits: 0	10992	16	9.4
Ionosphere	351	34	35.9

The SMOTE algorithm is performed by the unbalanced package in R and the random forest model is built by the randomForest package in R. The performance of Ferri's tree is given

in B. Song's work [17]. We ran 10 fold cross validation 10 times on the data and got the following performance. The ROC random forest and SMOTE random forest both contain 100 trees so that their performance is comparable. The performance of each method is shown in the following Table 12.

Table 12: Algorithm Performance Measured by AUC

Data Set	Letter A	Opt Digit 0	Pen Digit 0	Ionosphere
Ferri's Gain Ratio	98.9±1.4	94.2±1.4	99.6±0.5	90.4±7.0
Ferri's AUC Split	99.3±0.7	98.5±1.8	99.4±0.6	89.7±6.7
Proposed ROC Random Forest	99.9±0.02	99.9±0.03	99.8±0.1	96.7±3.1
SMOTE Random Forest	99.9±0.02	99.9±0.04	99.9±0.1	96.9±2.9

We can see from this table that the proposed ROC random forest performs better than Ferri's methods, and it performs equally well with the SMOTE random forest model when measured by AUC.

Compared to the SMOTE algorithm, the ROC random forest is much faster since it does not require any pre-processing steps on the data, and it is equally fast compared to random forest. Let n be the number of training observations at one node, and k be the number of unique classes, m be the number of features considered at this node, and s be the average number of possible splits along each feature. Given the time complexity of calculating AUC is $O(nlog(n))$ [42], the time complexity of building a ROC tree node is

$$O(m(nlog(n) + ks))$$

On the other hand, the time complexity of building a CART tree node is

$$O(m(nlog(n) + s))$$

The majority of the time is spent on sorting the observations based by each feature to calculate the AUC and Gini impurity. Therefore the speed of building a ROC tree node should be the same as the speed of building a CART node.

5.5 Summary of Performance

The simulation results show the superiority of ROC Tree to CART and ROC random forest to traditional random forest on simulated imbalanced data sets when the trees are shallow. The ROC based methods outperformed traditional tree methods in all settings when the class percentage is 95%, 2%, 2% and 1%. The advantage of Multi-Class ROC tree/random forest will be more significant when

- the classification problem is harder to differentiate. The splitting method based on the sum of all One vs. All AUC will tend to balance the AUC.
- the dimension of the problem is higher. The advantage of Multi-Class ROC Tree will be able to cumulate as the dimension grows.

Meanwhile, the multi-class ROC random forest outperformed the random forest in both setting 3 and setting 4, which indicates that the random forest framework works well on the ROC tree. The ROC random forest is able to achieve a better performance than traditional random forest when the depth restriction is small, which means the AUC based node selection method is more efficient in selecting the first several features and thresholds.

The result of balanced classification simulations showed that ROC tree and ROC random forest performed equally well as CART and traditional random forest in different measures,

which means the splitting criteria based on AUC would not reduce the classifier performance in balanced classification settings.

The performance of ROC forest on the UCI repository data shows that the ROC forest could achieve similar performance with the SMOTE algorithm without any data pre-processing steps. Note that in the process of SMOTE algorithm, it needs to find several nearest neighbors to generate synthetic minority examples, which requires $O(n^2)$ time complexity for each generated node to search for its neighbors. Therefore the SMOTE algorithm is comparatively time consuming. It may take more time than building a random forest model, and increases the number of parameters a predicting system needs to tune.

The time complexity of building a ROC tree node is similar to building a CART tree node. The majority of the time is spent on the sorting process to calculate the AUC and Gini impurity. Furthermore, in the ROC tree node selection process, the threshold of attributes that are not selected will not be used, which potentially reduce the computation load. The disadvantage of ROC forest is that it needs to turn categorical data into dummy variables, and in this case one node can only choose one level as the attribute.

Generally speaking, the ROC Tree and ROC random forest are performing well in different data sets. They can be used as a solution for highly skewed classification problems. They can also be used in regression problems as an ensemble method, which is discussed in detail in Chapter VI.

Chapter VI: Ensemble Application on Boston Housing Data

6.1 Ensemble and Model Stacking Introduction

Generally speaking, ensemble methods refer to methods that combine different classifiers to predict the target. Model stacking is part of ensemble methods, which means building meta models using but not limited to the predictions of other models. This stacking idea is first proposed by Wolpert in 1992 [24], and has since grown into a widely used technique in machine learning competitions. Almost all recent winning solutions of public competitions either used ensemble methods or deep learning methods.

6.1.1 Bias Variance Tradeoff

The explanation to the ensemble methods is based on the bias-variance tradeoff theory. Suppose y is the target we want to predict using features X, and f is the true relationship between X and y that E[f(X)] = y. The model we built to predict y based on X is $\hat{f}$, then the expected error of this model can be decomposed to two components shown below:

$$E\left[\left(y-\hat{f}(X)\right)^2\right] = Bias[\hat{f}(X)]^2 + Var\left(\hat{f}(X)\right) + \sigma^2$$

where $Bias\left[\hat{f}(X)\right] = E[\hat{f}(X) - f(X)]$, and $Var\left(\hat{f}(X)\right) = E\left[\hat{f}(X)^2\right] - E[\hat{f}(X)]^2$. Here σ^2 represents the error contained in y [36].

Based on this formula, we can see that the error of a model actually comes from two parts, bias of the model and variance of the model. Figure 5 below explains this formula and the theory behind [37].

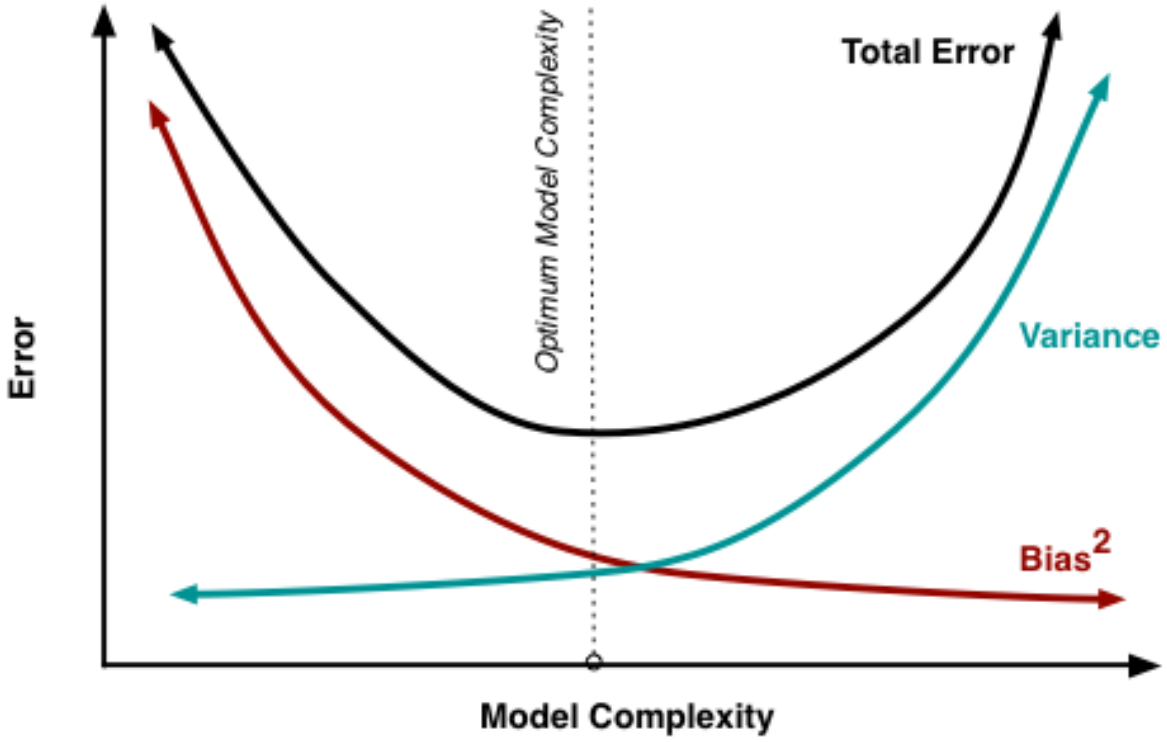

Figure 5: Bias Variance Tradeoff (S. Fortmann [37])

The total error equals to the sum of Bias square and Variance, as it is shown in the formula. As the model becomes more complex, the bias will decrease and the variance will increase. We can see that the training error always decreases when we have more complex models, because a complex model will be able to learn more details than a simple model. However, these details might not be true relationship but some random errors, and in this case the variance of the complex model will increase, which will be an important part of the test error. For example, as we increase the number of trees in gradient boosting machine, the training error

will always go down, but the test error will start to go up at some point. This situation is also known as overfitting.

6.1.2 Variance Deduction by Ensemble

The major purpose of combining different classifiers is to reduce the variance of the models. As stated in Section 6.1.1, when the complexity of the model increases, the model will be able to learn more details of the training data. A polynomial regression will always has a larger R^2 than linear regression on the training data, and a tree with depth n+1 will always has a smaller error rate than a tree with depth n on the training data. However, when the model is focusing on so many details in the training data, it actually learns this specific training data set instead of the possible relationship between the features and the target, which increase the variance of the model, because the model will change a lot if we have another training data from the same distribution.

However, the decrease of bias and increase of variance happen at the same time a lot. Stacking methods can be used to reduce the variance of complex models while having the benefit of small bias. Take average ensemble for regression problems as an example. Suppose F is the functional space which includes all functions mapping from the feature X to the target y, and $\hat{f}_i \in F, i = 1, \dots, k$ are the k models we built to predict y based on X. Suppose we have

- $E\left(\widehat{f}_i(X)\right) = y + \mu_i, i = 1, \dots, k$ where μ_i are the bias of model $\widehat{f}_i$, and
- $Var\left(\widehat{f}_i(X)\right) = \sigma_i^2, i = 1, \dots, k$ where σ_i^2 are the variances of model $\widehat{f}_i$.

Then define the average ensemble $\widehat{f}_k' = \frac{1}{k}\sum_{i=1}^{k}\widehat{f}_i$, and we can have

$$E\left(\widehat{f}_k'(X)\right) = y + \frac{1}{k}\sum_{i=1}^{k}\mu_i$$

and

$$Var\left(\widehat{f}_k'(X)\right) = \frac{1}{k^2}\sum_{i=1}^{k}\sigma_i^2 + \frac{1}{k^2}\sum_{i\neq j} Cov(\widehat{f}_i(X), \widehat{f}_j(X))$$

In reality, the correlation between $\widehat{f}_i(X)$ will be very large. Suppose the correlation is 0.9 between all $\widehat{f}_i \; and \; \widehat{f}_j$, and suppose $\sigma_i^2 = \sigma_j^2 = \sigma^2, \forall\, i,j,s.t. 1 \leq i,j \leq k$, which means the variance of all the models are the same. In this setting, we can have

$$Var\left(\widehat{f}_k'(X)\right) = \frac{1}{k}\sigma^2 + 0.9 * \frac{k(k-1)}{k^2}\sigma^2 = 0.9\sigma^2 + \frac{1}{10k}\sigma^2 < \sigma^2$$

Therefore, the average ensemble will reduce the variance of the models when the models have non-perfect correlation and similar variance. However, in reality, it is hard to tell the variance of a model, but the average ensemble will work when the models are less correlated and have similar performance.

6.1.3 Ensemble Methods

The stacking methods can be roughly divided into two categories, depends on whether it requires extra training process or not. The first category includes the average ensemble mentioned above, the majority voting process used in random forest, and weighted average ensemble where the weights are based on the performance on the out of bag training data. This category doesn't require extra training data, which is a large advantage since in most cases the feature space is very sparse and more data means better model performance. However these

methods are not able to ensemble the models differently for different observations, so they can't learn complicated structures.

The second category requires extra training data, which includes meta modeling (model stacking) and gradient boosting trees. The training process of gradient boosting trees is different from meta modeling, since it doesn't require extra training data. The summation coefficient of each tree and each leaf in gradient boosting trees is calculated based on the same training data that built the tree.

Meta modeling, or model stacking, refers to those methods that build a higher level model which integrates the prediction of base models. For example, we can build a logistic regression or linear regression model which takes the original features as well as the output of the base models as input to predict the label or the response. In this case we can have different ensemble for observations with different features. On the other hand, this also introduces more parameters into the model architecture, which increases the complexity of the final model and the risk of overfitting.

In reality, the ensemble methods can be combined together to build several layers of models. In the Otto Group Product Classification Challenge on kaggle, the winning solution is based on a 3-layer learning architecture [38]. This solution uses 33 base models in the first layer, and 3 meta models in the second layer which takes the result of these 33 models and 7 other features as input. The third layer is a combination of weighted geometric mean and arithmetic mean of the 3 meta models. This complex architecture finally won this competition.

6.2 Data Introduction and Pre-processing

In this application, we are going to use Multi-Class ROC random forest as a meta model to predict the house price. We use different methods to build three base models to predict the house prices, and use ROC random forest to assign weight to the three base models. This method is compared to all the base models as well as other popular ensemble methods to show that ensemble is improving the performance and Multi-Class ROC random forest works well as an ensemble method in this data set.

6.2.1 Boston Housing Data Introduction

The data set is called Boston Housing data, which is described in [32]. The data set has 2930 rows and 82 columns. Each row represents a residential house sale record occurred within Ames from 2006 to 2010. Each column is a feature of the house, including Neighborhoods, Overall Condition, Year Built, Foundation, Ground living area, etc. The documentation of all the columns can be found in [39]. Observations with living area square feet larger than 4000 were removed because they are regarded as outliers in the documentation. There are 2925 observations after filtering.

This data set is already used in several textbook for regression analysis. In this application, we are going to predict the Sale Prices of the home based on all other features excluding ID.

6.2.2 Imputation and Evaluation

There are many missing values in the data so imputation is necessary before we start the modeling process. 42 out of the 82 features are categorical features, and there were only a few numerical features with missing data, which are Lot Frontage, Masonry Veneer Area, Basement

Square Feet, and Garage Year Built. These numerical features are imputed to be zero. For all the categorical features, missing data is treated as another level.

All the categorical features are transformed into dummy variables, which increase the number of features to 285. This is necessary because the gradient boosting trees and ROC random forest used here can only take numerical features. We used 80% of the data to be training data (2340 rows) and 20% of the data to be test data (585 rows) by random split. We further split the training data to sub model training data (1404 rows, 60%) and meta model training data (936 rows, 40%) by random split. The reason that we further split the training data is to avoid overfitting of the meta model. The performance of each base models on the training data will be better than its performance on the testing data, therefore if we build the meta model using the same training data with the base models, there is a risk of overfitting, since the decrease of performance of each model on the testing data may not be the same.

The evaluation measure of the model performance is the logarithm root of mean square (log rmse), which is

$$(\frac{1}{n}\sum_{i=1}^{n}(\log(pred_i) - \log(truevalue_i))^2)^{0.5}$$

6.3 Base Models

Three based models, gradient boosting trees (XGBoost package in R), random forest (randomForest package in R) and regularized regression (glmnet package in R), are used in this application.

6.3.1 XGBoost Model

The XGBoost package is chosen to build the gradient boosting trees because it is very efficient and has pretty good performance on a lot of data sets [41]. The response of the leaf in each tree is calculated by the first and second derivative of the cost function, which is deduced by Taylor expansion of the cost function. Therefore it is easy to change the cost function in XGBoost, and it takes comparatively less time to build the tree compared with the gbm package. Furthermore, the performance of an XGBoost model is pretty good, which makes it very popular in data science competitions.

The gradient boosting tree model here used all 283 features and built 600 trees, which was selected by cross validation. The parameters are tuned to minimize the logarithm rmse on the test data. This model achieves a log rmse of 0.1194 on the meta training data and a log rmse of 0.1119 on the testing data.

6.3.2 Random Forest Model

One advantage of random forest over gradient boosting trees is that it requires less effort to train the model. The random forest model used 283 features and 1000 trees, and this model achieves a log rmse of 0.1340 on the meta training data and a log rmse of 0.1164 on the testing data. The variable importance given by random forest showed that a lot of features contributed to the prediction, and the several most important features are general living area, overall quality, etc.

6.3.3 Regularized Regression

The regularized regression model is chosen because it could achieve a much more robust result than tree based models. Although it could not learn any interactions between the features,

or non-linear relationship between the features and the target, it could provide a prediction less correlated to tree based methods, which is important to the ensemble part.

For this regularized regression model, some numerical features including last sold year, built year are not used since it doesn't make sense to assign a coefficient strictly related with year, and some of the numerical features are imputed to 0. Since there are so many features in this problem, the feature space is very sparse, so we need to use large penalty to avoid overfitting. The final regularized regression model used both lasso and ridge penalization and there were 58 non-zero coefficients. The target of this model is the sale price after log transformation. This model achieves a log rmse of 0.1183 on the meta training data and a log rmse of 0.1073 on the testing data.

6.3.4 Correlation of Base Models

The following Table 13 shows the correlation of the testing data prediction of the three base models.

Table 13: Correlation on Testing Data of Each Base Model

	XGBoost	Random Forest	Regularized Reg
XGBoost	1	0.9906	0.9816
Random Forest	0.9906	1	0.9819
Regularized Reg	0.9816	0.9819	1

The correlation between all the base models are very high, since they are predicting the same response variable using the same features, and they all perform pretty well. However, there is still room for ensemble methods to improve based on this level of correlation. Furthermore,

the ensemble will be able to learn from a different set of training data, so they will perform better than the base models.

6.4 Ensemble Methods for Comparison

Next we are going to compare several ensemble methods including average ensemble, Multi-Class ROC forest ensemble, classification random forest ensemble and regression random forest ensemble.

6.4.1 Average Ensemble

The idea of average ensemble is simple, that for each testing observation we use the average of the three base model predictions to be the final prediction. This method doesn't need extra training and achieves a log rmse of 0.1012 on the testing data with a standard deviation of 0.00127 after 10 runs, which is better than any single base model. The average ensemble works well when the performance of each base model is close and the prediction of each base model are less correlated.

6.4.2 Regularized Regression Ensemble

The regularized regression ensemble model is similar to the base regularized regression model. The regularization is a combination of L1 norm and L2 norm, and the model is trained with the 283 features as well as the predictions of the 3 base models on the meta training data to predict the sale price directly. This model can be regarded as an advanced version of weighted average ensemble because it is actually assigning weights to all the base models and calculates the weighted average of the model predictions. It is different from weighted average because it

also considers other features and has regularization to prevent overfitting. This method is able to find a set of weights to ensemble the base models and assign weights differently based on different value in the features.

The regularized regression ensemble achieves a log rmse of 0.1030 on the testing data with a standard deviation of 0.00342 after 10 runs.

6.4.3 ROC Forest Ensemble

The multi-class roc random forest is a classification algorithm, therefore in order to use this algorithm to perform ensemble, we need to transform the regression ensemble problem to a classification one.

For each prediction on the meta training data, the model that gives the closest prediction is regarded as the target. Therefore the ensemble problem is transformed into a problem to predict the best model. The predictions are very close for some observations, so these observations are dropped. The criteria for acceptable labels are that it is the closest to the true label and the absolute deviance is at most two thirds of the average absolute deviance of the other two base models. After filtering out observations with obscure best model, we have 960 rows in the meta training data. The ROC random forest is built with 30 trees using 283 features together with the prediction of the three base models to predict the best model.

After the prediction of this ROC random forest ensemble is given on each testing observation, we use the probability of each class to get a weighted average of the 3 base models as the ensemble prediction, because the weighted average can help reduce the variance of the final ensemble. The major advantage of applying ROC random forest here is that we are able to ensemble the predictions differently for observations with different features.

The final ROC random forest ensemble achieves an average log rmse of 0.1006 with a standard deviation of 0.00207 after 10 runs.

The most representative tree in this ROC random forest ensemble is plotted below as Figure 6.

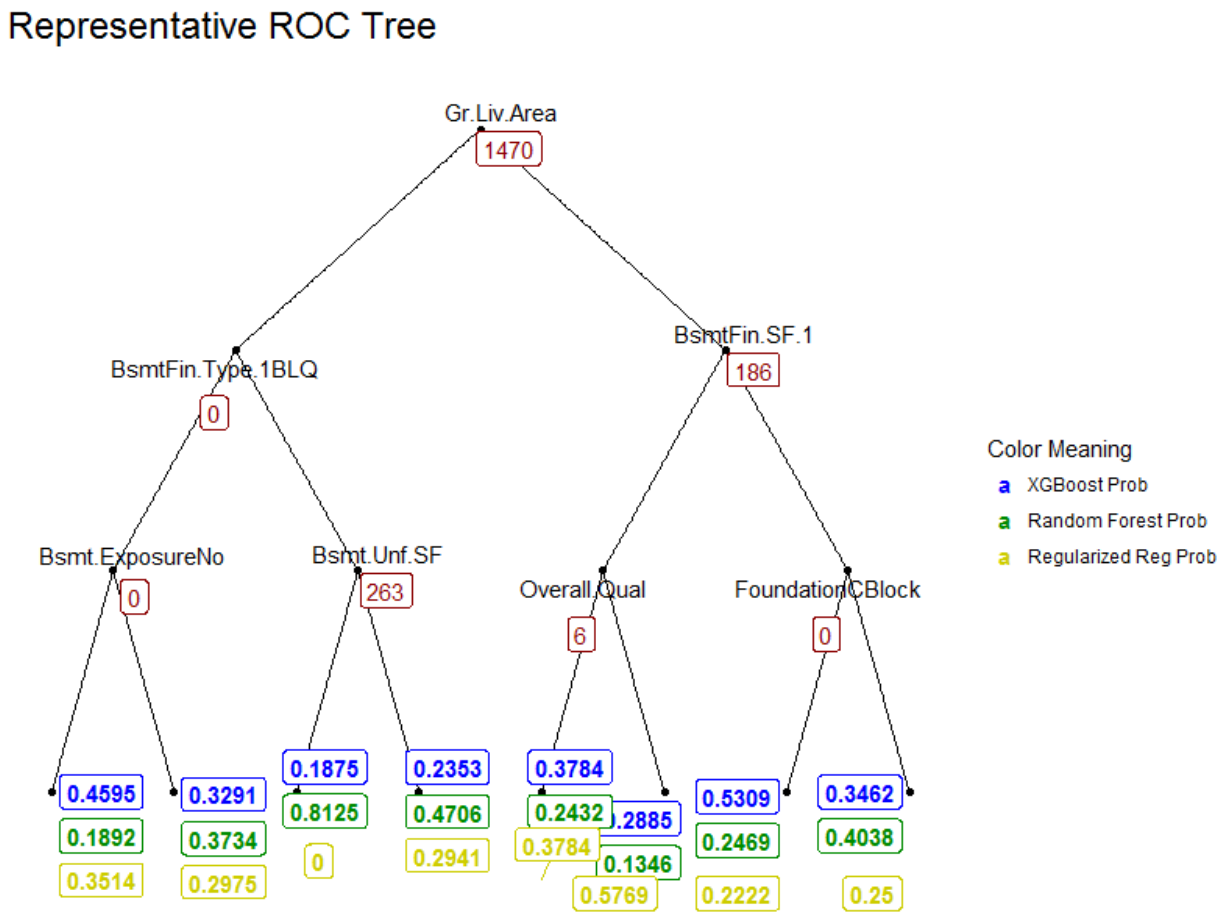

Figure 6: The Most Representative Tree in Split Train ROC Random Forest Ensemble. The red number is the split point on that feature, the blue number is the score for xgboost model, the green number is the score for random forest model, and the yellow number is score for regularized regression model.

6.4.4 Classification Random Forest Ensemble

A classification random forest is applied on the same meta training data with the ROC random forest described in Section 6.4.3 for comparison. The classification random forest uses the same parameters with the ROC random forest and achieves an average log rmse of 0.1006 on the testing data with a standard deviation of 0.00194.

The most representative tree in this ensemble is plotted below as Figure 7.

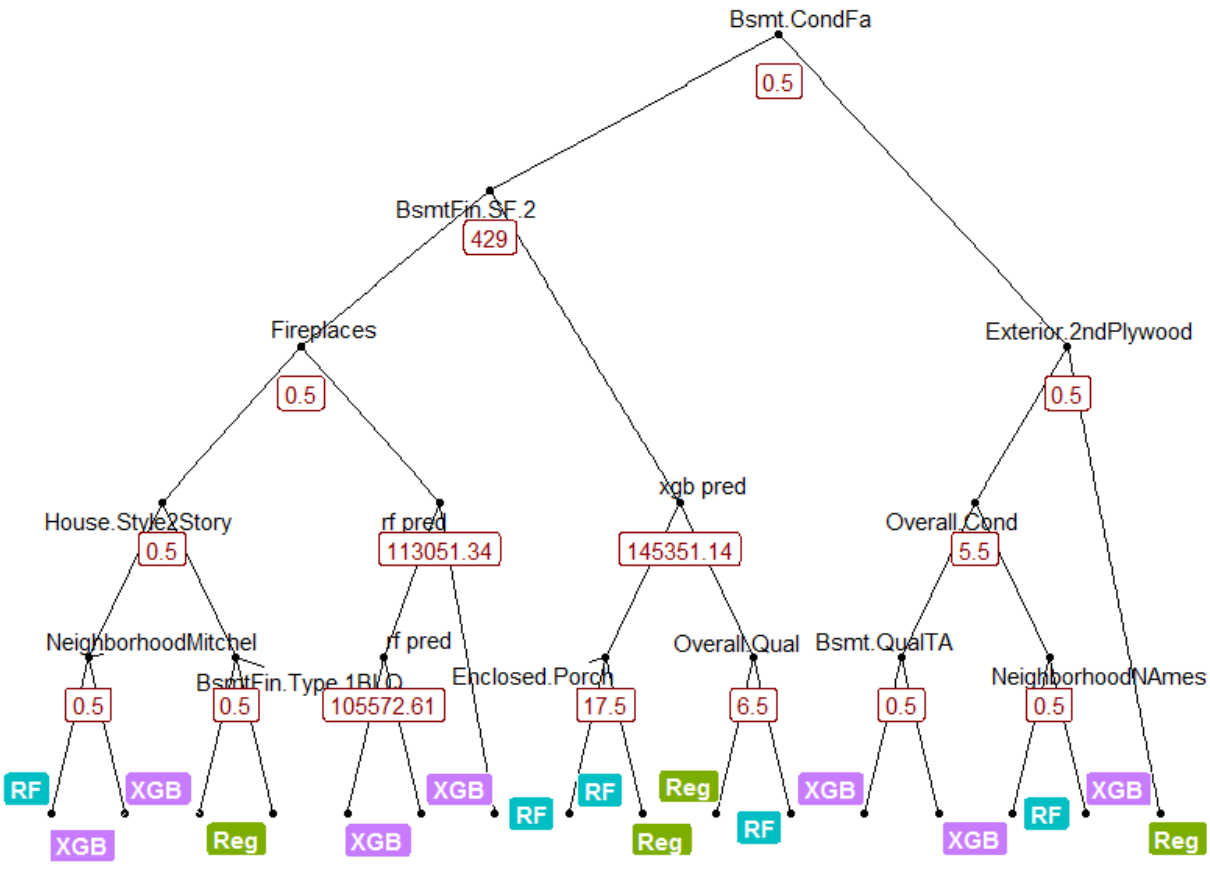

Figure 7: The Most Representative Tree in Split Train Classification Random Forest Ensemble. In the end nodes, RF indicates that the label for this node is the random forest base model, XGB indicates the label for this node is the XGBoost base model, and Reg indicates the label for this node is the regularized regression base model.

6.4.5 Regression Random Forest Ensemble

A traditional method to perform ensemble is to build a model using the predictions of the base models to predict the response. Here I use a regression random forest, which used all 283 features as well as the prediction of the three base models, to build a meta model to predict the sale price directly. This regression random forest ensemble achieves an average log rmse of 0.1063 with a standard deviation of 0.00186 on the testing data.

The most representative tree in the regression forest ensemble is plotted below.

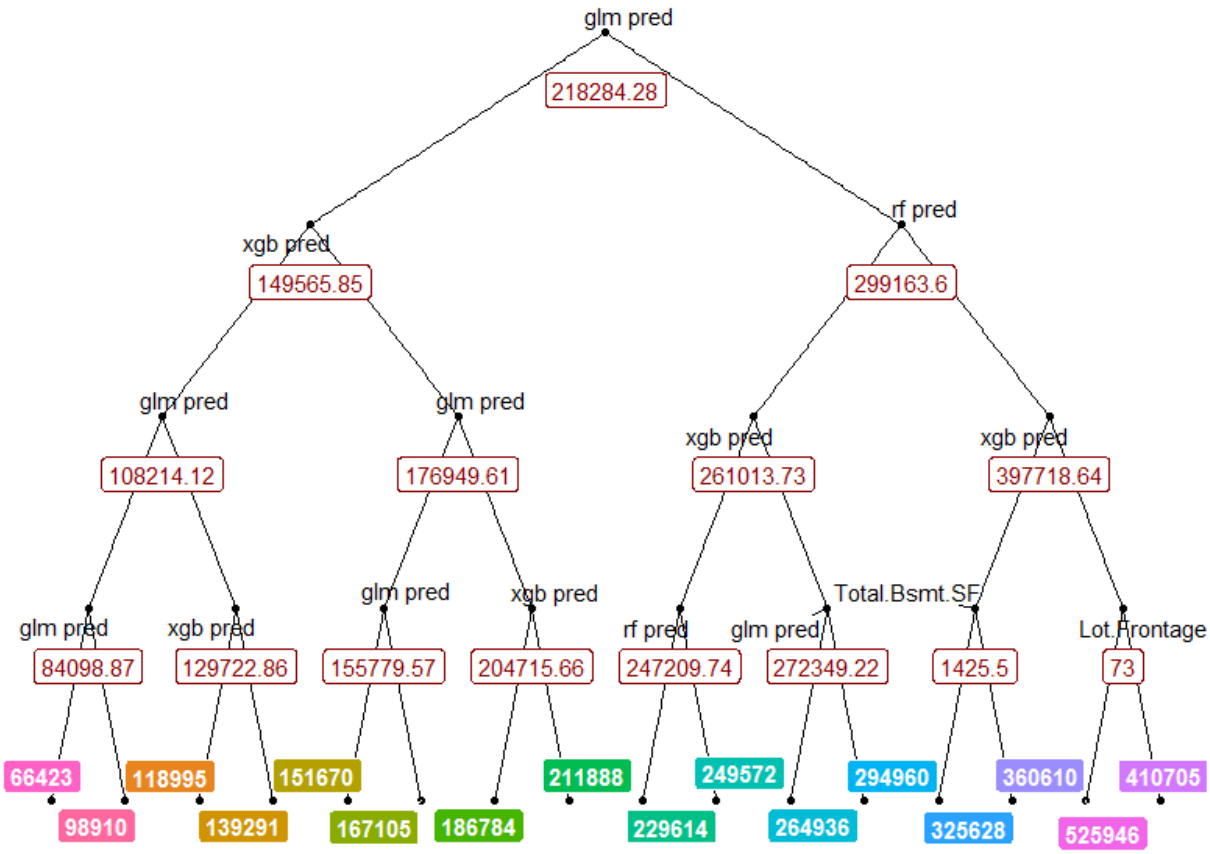

Figure 8: The Most Representative Tree in Split Train Regression Random Forest Ensemble. The number in the internal nodes indicates the split threshold, and the number in the terminal nodes indicates the response value for this node.

6.4.6 Comparison Summary

The performance of each base model and ensemble method is summarized in Table 14 below. The mean and standard deviation of the performance is calculated based on 10 different random splits of the base modeling training and meta model training data. Therefore a large part of the variance comes from the difference in training data.

Table 14: Performance of each model on the testing Data

Algorithm	Model Type	Test Log RMSE
Gradient Boosting Tree	Base Model	0.1072 ± 0.00243
Random Forest	Base Model	0.1144 ± 0.00138
Regularized Regression	Base Model	0.1078 ± 0.00497
Average Ensemble	Ensemble Model	0.1012 ± 0.00127
Regularized Regression	Ensemble Model	0.1030 ± 0.00342
ROC Forest Ensemble	Ensemble Model	0.1006 ± 0.00207
Classification Random Forest	Ensemble Model	0.1006 ± 0.00194
Regression Random Forest	Ensemble Model	0.1063 ± 0.00186

From the table it is easy to see that all ensemble method improve the performance on the testing data including the average ensemble. The ROC random forest ensemble has the lowest average on the testing data, followed by the classification random forest. Most ensemble methods excluding the regularized regression ensemble have smaller variance than base models, which agrees with the theory in Section 6.1.

The classification ensemble scheme performs better than the regression ensemble scheme. One reason behind is that the problem is very sparse and too many features in the

regression ensemble model will lead to overfitting. On the other hand, there might be overfitting in the classification ensemble scheme as well, but the classification ensemble scheme uses weighted average of the base models as final prediction, which increases the stability of the system.

The model performance across each run is shown in the figure below using a line plot.

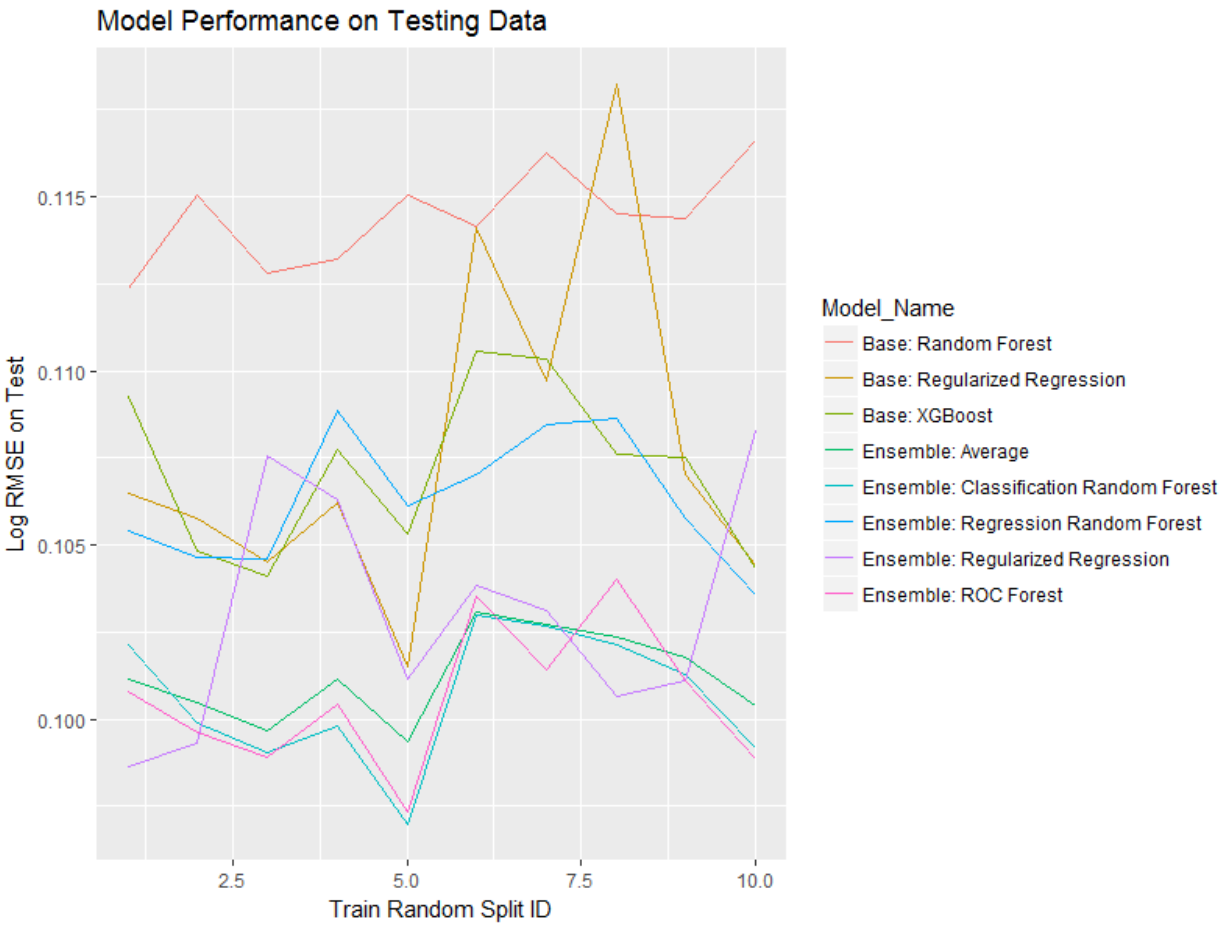

Figure 9: Model Performance on Testing Data. This figure shows the performance of separated trained base models and ensemble models on the testing data.

One of the advantages of ROC random forest over random forest is that it is much more stable, however it is not obvious in this setting because most of the variance comes from the

random split of the training data. It is also the reason that the regularized regression has such high variance, which is supposed to be more stable than random forest, as shown in Section 6.5.

6.5 Overlapping Ensemble

Note that this problem is very sparse, and it is hard for classifiers to achieve a good performance with such limited training data. As a result, none of the four ensemble methods we compared above has significant advantage over the average ensemble. In this case we would like to check the result when we use all the training data to train the base models and perform ensemble training on the same training data.

6.5.1 Fine Tune Gradient Boosting Tree Model

Before we move to the overlapping ensemble part, it is important to check the best performance a single layer model could achieve, since the models trained for ensemble have to minimize the performance difference between training data and testing data.

Therefore a fine-tuned XGBoost model is built using all the training data, which achieves a log rmse of 0.0629 on the training data and a log rmse of 0.0998 on the testing data. The single XGBoost model is selected because it performs better than single random forest and regularized regression model.

6.5.2 Full Train Base Models for Ensemble

In this situation that the base models and the ensemble models are trained with the same training data, it's very important to control the difference of base model performance on the

training data and their performance on the validation data. Therefore in this case, the target of the training is to control the variance of the base model instead of minimizing the total error.

The XGBoost base model is trained with 400 trees and it only considers at most 3 way interactions to avoid overfitting. This model achieves a log rmse of 0.1025 on the training data and a log rmse of 0.1051 on the testing data.

The random forest base model is trained with a max nodes restriction of 2^6, which means the depth of each tree is at most 6. If there are no restrictions on random forest, each tree would be fully grown and each leaf will have exactly one observation. In this case the performance on the training data will be much better than the performance on the testing data. Therefore it is important to place restriction on random forest. This random forest model with all the training data gives a log rmse of 0.1300 on the training data and a log rmse of 0.1254 on the testing data.

The regularized regression base model is using a smaller penalty and it has 136 non zero features. This model achieves a log rmse of 0.1082 on the training data and a log rmse of 0.1021 on the testing data.

The correlation of testing data prediction of all the base models is shown in the following Table 15. We can see that the correlation is less than those in Section 6.3.4, because the base models here are not optimizing the testing data performance but control the difference of performance on the training data and testing data.

Table 15: Correlation between Each Full Train Base Model

	XGboost	Random Forest	Regularized Reg
XGBoost	1	0.9883	0.9879

Random Forest	0.9883	1	0.9764
Regularized Reg	0.9879	0.9764	1

6.5.3 Full Train Ensemble Models

The ensemble models are trained similarly with Chapter 6.4. The performance of each base model and ensemble methods are summarized in the following Table 16. The mean and standard deviation is calculated by 10 repeated runs. The standard deviation here is much smaller than Section 6.3 and 6.4 because the training data remains the same for all models. Note that the performance of the base XGBoost model and the base regularized regression model are not changing because there is no randomness in regularized regression and the random seed in XGBoost model is the same.

Table 16: Performance of each model on the testing Data

Algorithm	Model Type	Test Log RMSE
Gradient Boosting Tree	Base Model	0.1051 ± 0
Random Forest	Base Model	0.1254 ± 0.00023
Regularized Regression	Base Model	0.1021 ± 0
Gradient Boosting Tree	Single Model	0.0998
Average Ensemble	Ensemble Model	0.1015 ± 0.00010
Regularized Regression	Ensemble Model	0.0994 ± 0.00018
ROC Random Forest	Ensemble Model	0.0987 ± 0.00008
Classification Random Forest	Ensemble Model	0.0992 ± 0.00041
Regression Random Forest	Ensemble Model	0.1046 ± 0.00113

The following Figure 10 shows the performance of each model across all runs.

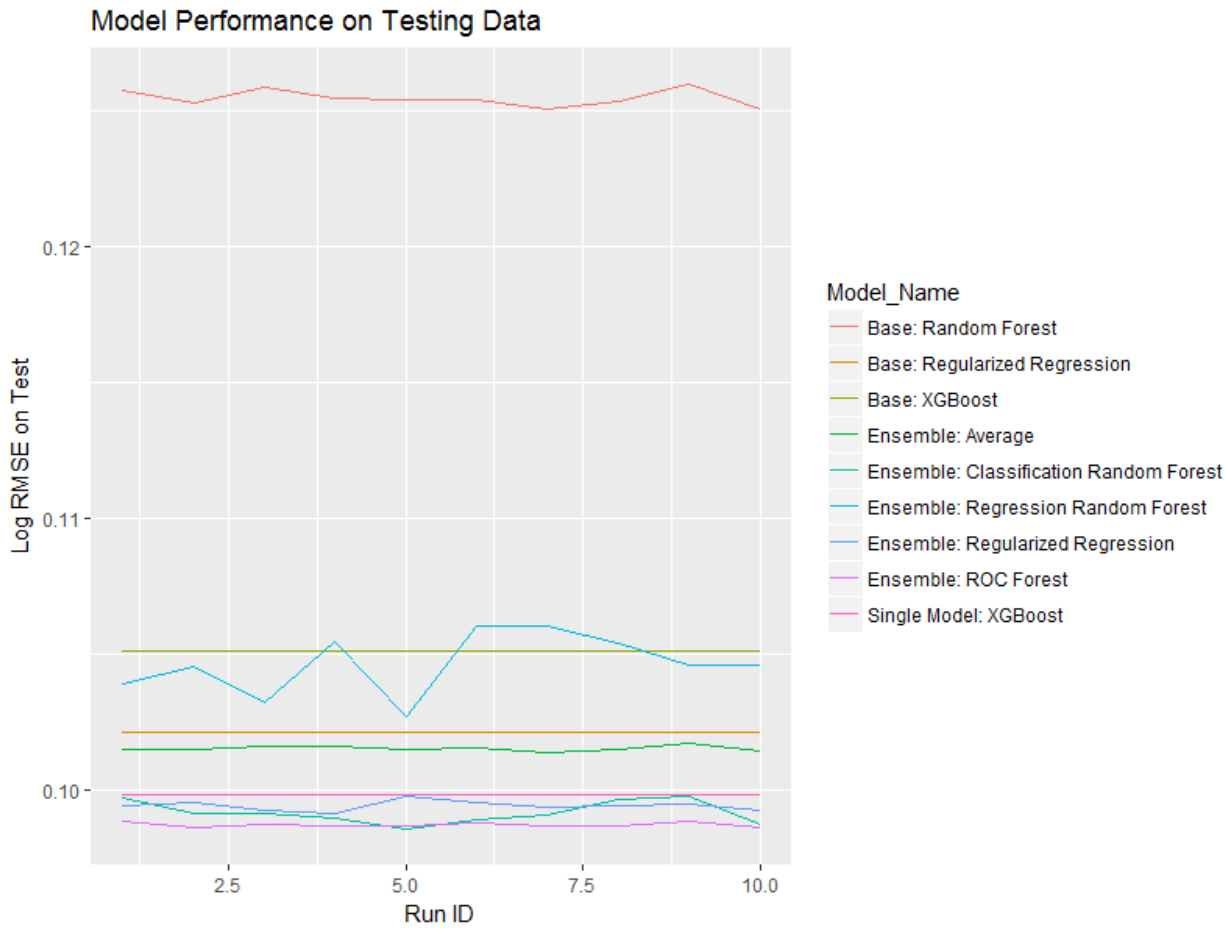

Figure 10: Model Performance on Testing Data. This figure shows the performance of overlapping trained base models and ensemble models on the testing data. The performance of random forest is dropped to control the performance difference on the testing data and training data.

The following Figure 11 enlarges the performance of the best four models, the ROC random forest ensemble, regularized regression ensemble, classification random forest ensemble and the single XGBoost model.

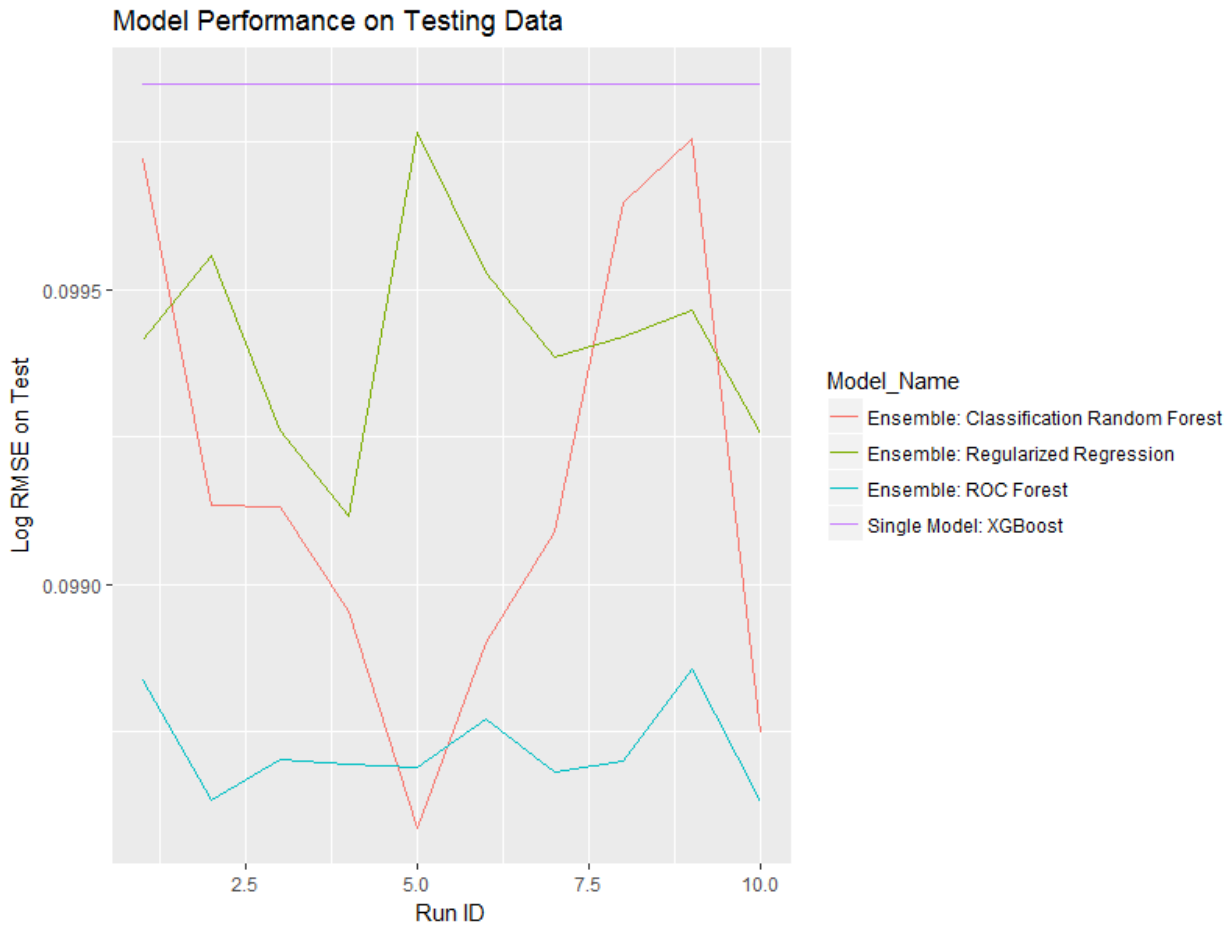

Figure 11: Enlarged Model Performance on Testing Data. The data shown in this figure is the same to Figure 7. The performance of the best models is enlarged in this figure.

From this figure we can see that ROC random forest has a large advantage over other ensemble methods, and most ensemble methods performs better than single fine-tuned model.

The classification random forest and ROC random forest are built using the same training scheme that they are all predicting the closest base model and use class probability to ensemble the base models. The result shows that ROC random forest has advantage over classification random forest in this ensemble application in both average performance and stability.

The regularized regression ensemble and regression random forest ensemble are built on same training scheme that they are all predicting the response directly using the features as well as the predictions of the base models. This training scheme is not performing as well as the classification one above. One reason for this is that the regression scheme may put too much weights in the base model outputs instead of the base features, as shown in the following representative trees.

The following figure shows the most representative tree in ROC random forest ensemble.

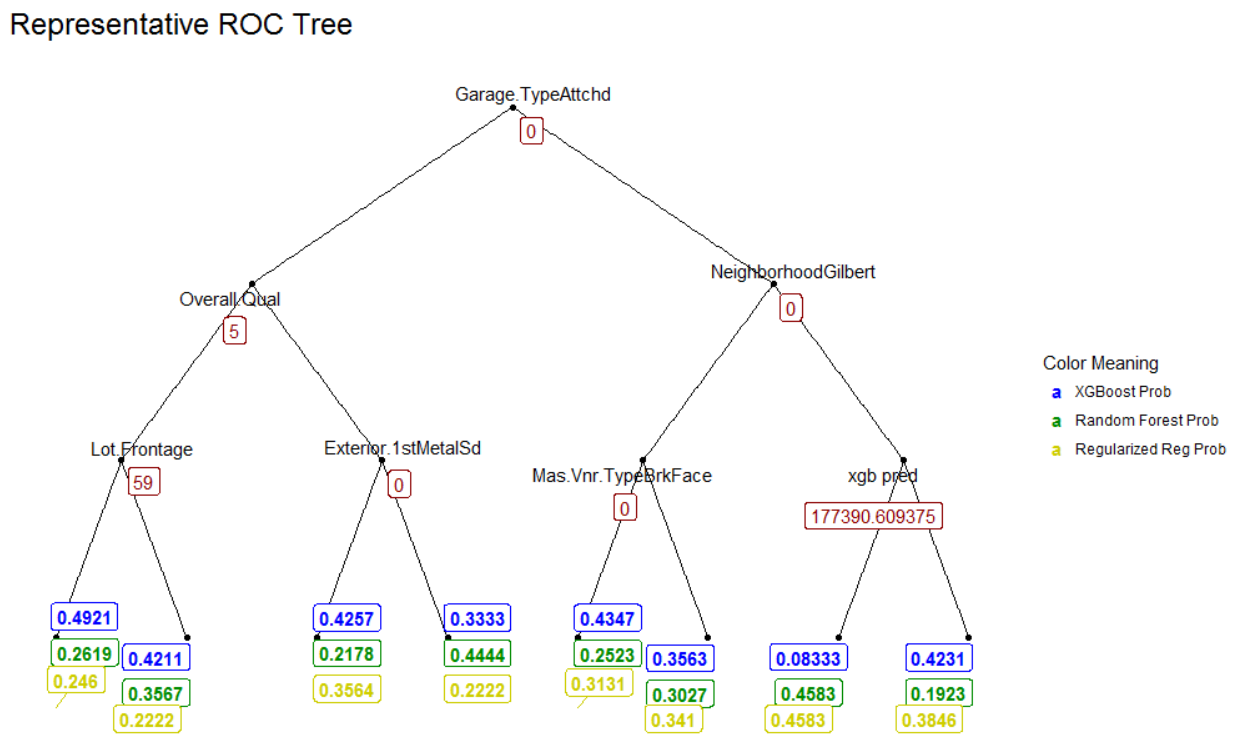

Figure 12 The Most Representative Tree in Overlapping Train ROC Random Forest Ensemble. The red number is the split point on that feature, the blue number is the score for xgboost model, the green number is the score for random forest model, and the yellow number is score for regularized regression model.

The following figure shows the most representative tree in classification random forest ensemble.

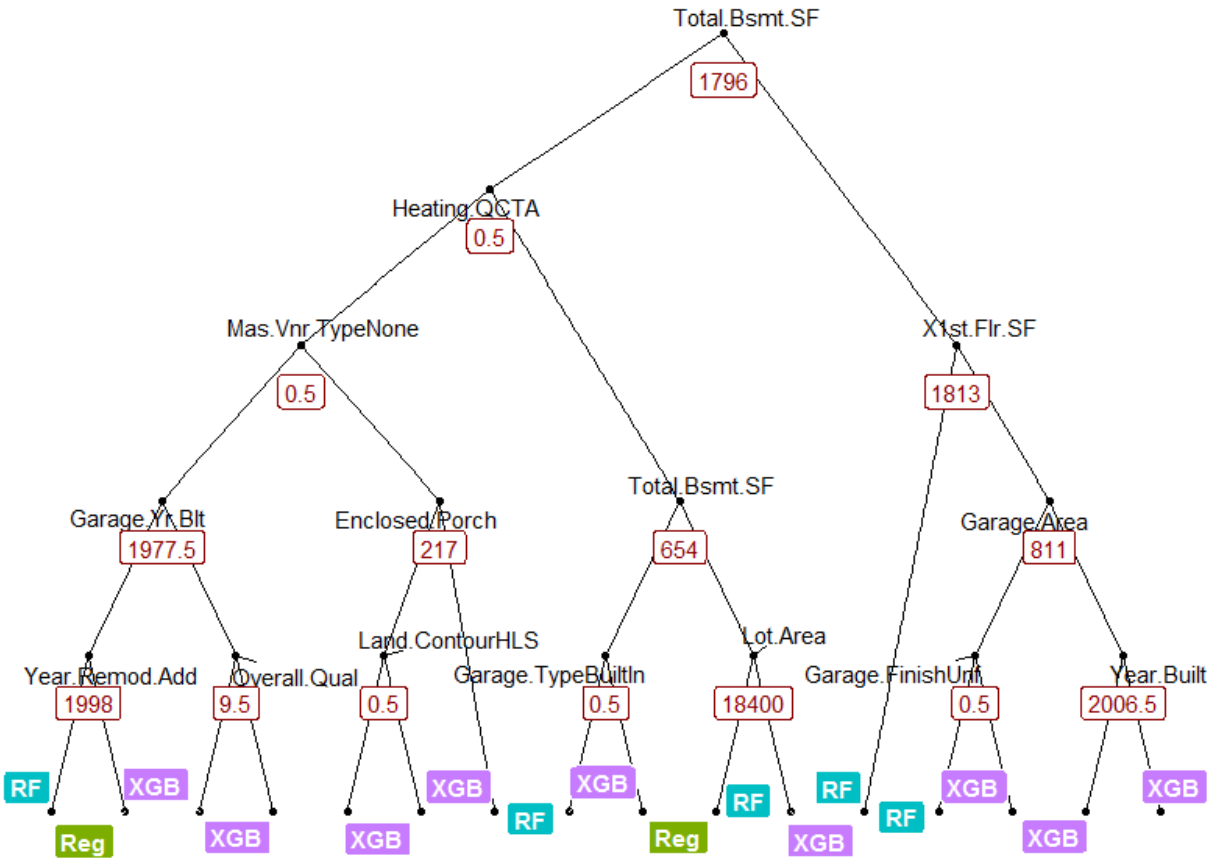

Figure 13:The Most Representative Tree in Overlappling Train Classification Random Forest Ensemble. In the end nodes, RF indicates that the label for this node is the random forest base model, XGB indicates the label for this node is the XGBoost base model, and Reg indicates the label for this node is the regularized regression base model.

The following figure shows the most representative tree in regression random forest ensemble.

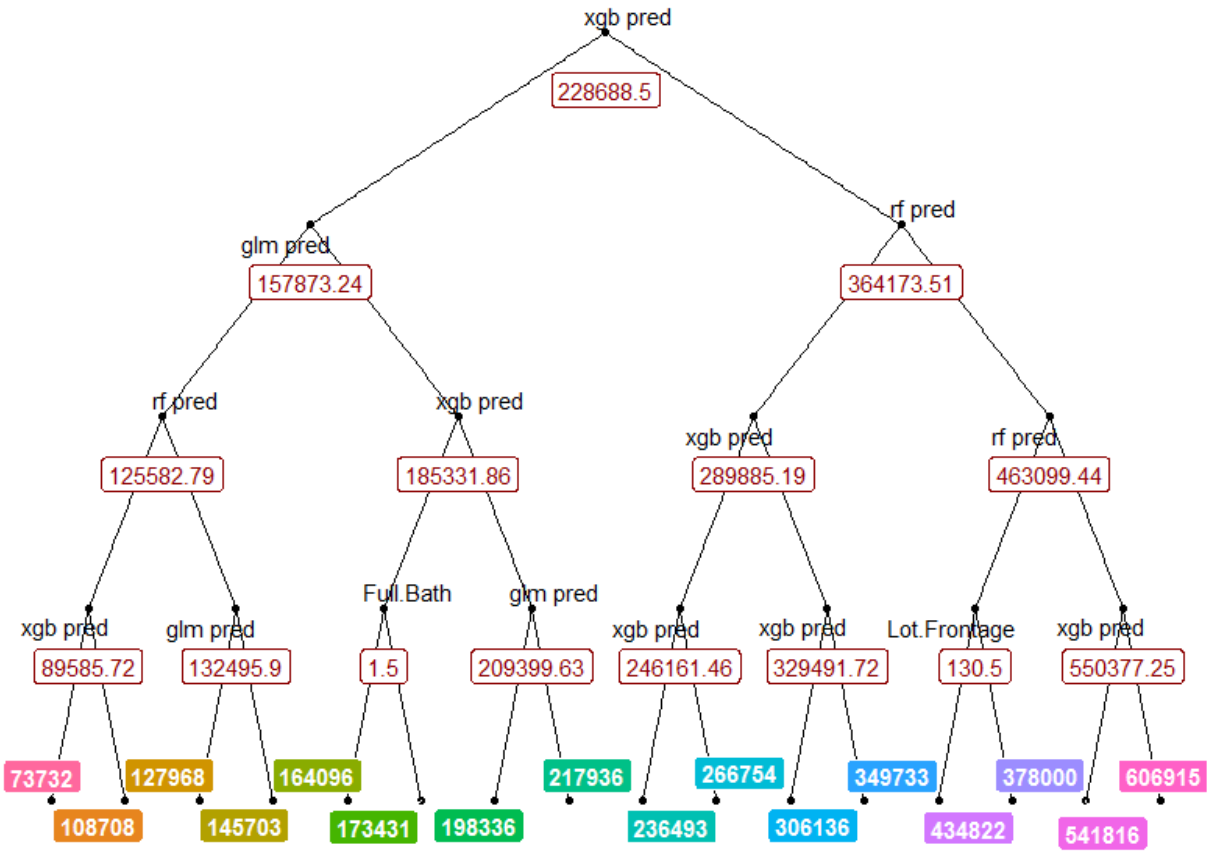

Figure 14 The Most Representative Tree in Overlapping Train Regression Random Forest Ensemble. The number in the internal nodes indicates the split threshold, and the number in the terminal nodes indicates the response value for this node.

From these representative trees along with representative trees in the split training scheme, we can see that the regression random forest ensemble is heavily using the prediction of the base models instead of the features. On the other hand, the ROC random forest and classification random forest is trying to ensemble the models based on the features of the data, which maybe the reason that they are more stable and accurate.

6.6 Boston Housing Data Application Summary

First of all, this application shows that ensemble methods are able to perform better and more stable than a fine-tuned single model, which agrees with the bias-variance tradeoff theory in Section 6.1. Among all the algorithms tested in this application, the ROC random forest is the best one in both splitting training scheme and overlapping training scheme. The top three models in this problem are overlapping trained ROC random forest, overlapping trained classification random forest and overlapping trained regularized regression. The main reason for this is that the ROC random forest is able to consider the sensitivity and specificity of all the classes and use the prior probability to adjust the prediction.

Secondly, this application shows that the ensemble methods could be used even in sparse problems. The Boston housing data has a lot of features and comparatively less training data. Therefore when we split the training data to base model training part and meta model training part, it is hard for us to achieve an excellent ensemble model. In this case, we can also use the training data twice for both base model and ensemble model, which is shown in Section 6.5. The key in this is to prevent overfitting in the base models, so that their performance on the training data is very close to their performance on the testing data. This is pretty hard for random forest, since random forest with loose regularization would achieve much better performance on the training data than testing data. Therefore there is a performance drop for random forest in the overlapping training part.

However, with these base models, the ensemble methods could achieve better performance than the split training part, and they are also performing better than a fine-tuned

single gradient boosting tree model. Therefore when the data is very sparse, we may want to use the whole training data for both base model training and ensemble model training.

Thirdly, this application shows that the classification ensemble scheme is better than the regression ensemble scheme, and ROC random forest is performing better than classification random forest in this scheme. The probability of the classification ensemble models is used to get a weighted average of all the base models, which could be the reason it is performing better, since weighted average could directly reduce the variance of the models.

Generally speaking, the last layer of an ensemble scheme should be a simple one like a weighted average to achieve robustness and reduce variance of the models. The classification scheme could help to utilize all the features but also keep the weighted average process, which is an advantage over simple weighted average and the regression scheme. On the other hand, although the regression scheme is also using all the features, it has a larger risk of overfitting as an ensemble model.

Finally, there are several key points in using the ensemble methods to improve the model performances, which are summarized below.

- The base models have similar performance but low correlation on testing data prediction. The lower the correlation is, the better performance the ensemble methods could achieve.
- The base models are performing similarly on the meta training data and testing data so that the risk of overfitting in ensemble model is reduced.
- Use robust ensemble method in the final layer of the ensemble scheme.

Chapter VII: Conclusion and Future Work

The multi-class ROC random forest proposed in this dissertation is developed from the ROC random forest proposed by B. Song [17]. It expands the binary ROC RF to multi-class cases by applying the one versus all idea. In the node attribute selection stage, it considers the AUC of all one versus all classifier and selects the attribute that provides the largest sum of the AUC. In the node threshold selection stage, it considers the harmonic mean of sensitivity and specificity of all the one versus all classifiers. By this design, the ROC random forest is able to (1) perform multi-class classification and (2) balance the performance on sensitivity and specificity for all classes. Furthermore, the prior probability of each class is considered in the random forest scheme, so that the minority class will have a reasonable weight adjustment. This is a parameter a user could tune during the modeling process. The multi-class ROC random forest shows its advantage on performance and speed in classification problems based on simulated data and UCI repository data.

Moreover, as a classification algorithm, the multi-class ROC random forest can also be used in regression problems as an ensemble methods, as shown in Chapter 6. The ensemble problem can be transformed into a classification problem by predicting which model is the best performing model, and the probability of each model could be used to obtain a weighted average of all the base models. This ensemble scheme combined with multi-class ROC random forest is performing better than all other ensemble methods tested and the best single XGBoost model in the Boston housing data.

The multi-class ROC random forest also shares some disadvantages with other algorithms. The first disadvantage is that the ROC random forest has to turn categorical features into dummy features, therefore it can only use one level of the categorical feature in one node. This solution is also used in linear regression, XGBoost, SVM, but the traditional random forest could utilize all the levels in one node. The second disadvantage is that the ROC random forest still has implicit regularization including max depth and minimum leaf size. This makes the model hard to tune. The Regularized Greedy Forest [35] provides a scheme to use direct penalty to control the regularization process, which could also be used on ROC random forest.

To summarize, the multi-class ROC random forest is recommended in highly imbalanced classification problems when the target is to maximize the AUC of the model prediction. It can also be considered as an alternative ensemble method in both classification and regression problems.

Reference:

[1] Guzella, Thiago S., and Walmir M. Caminhas. "A review of machine learning approaches to spam filtering." Expert Systems with Applications 36.7 (2009): 10206-10222.

[2] Rosenblatt, Frank. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

[3] Werbos, Paul John. "Beyond regression: new tools for prediction and analysis in the behavioral science." Ph. D. Thesis, Harvard University (1974).

[4] Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers." Proceedings of the fifth annual workshop on Computational learning theory. ACM, 1992.

[5] Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.

[6] Breiman, Leo. "Bagging predictors." Machine learning 24.2 (1996): 123-140.

[7] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[8] Federal Reserve System , The 2013 Federal Reserve Payments Study. https://www.frbservices.org/files/communications/pdf/general/2013_fed_res_paymt_study_detailed_rpt.pdf

[9] Yen, Show-Jane, and Yue-Shi Lee. "Cluster-based under-sampling approaches for imbalanced data distributions." Expert Systems with Applications 36.3 (2009): 5718-5727.

[10] Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.

[11] Núñez Castro, Haydemar, Luis González Abril, and Cecilio Angulo Bahón. "A post-processing strategy for SVM learning from unbalanced data." 19th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. 2011.

[12] Galar, Mikel, et al. "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42.4 (2012): 463-484.

[13] Elkan, Charles. "The foundations of cost-sensitive learning." International joint conference on artificial intelligence. Vol. 17. No. 1. Lawrence Erlbaum Associates Ltd, 2001.

[14] Swets, John A. "Detection theory and psychophysics: a review." Psychometrika 26.1 (1961): 49-63.

[15] Ferri, César, Peter Flach, and José Hernández-Orallo. "Learning decision trees using the area under the ROC curve." ICML. Vol. 2. 2002.

[16] Hossain, M. Maruf, Md Rafiul Hassan, and James Bailey. "ROC-tree: A novel decision tree induction algorithm based on receiver operating characteristics to classify gene expression data." Proceedings of the 2008 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2008.

[17] Bowen Song, “ROC Random Forest and Its Application”. Doctoral thesis, Stony Brook University, 2015.

[18] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." (2011).

[19] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006): 1-58.

[20] Sokolova, Marina, and Guy Lapalme. "A systematic analysis of performance measures for classification tasks." Information Processing & Management 45.4 (2009): 427-437.

[21] Landgrebe, Thomas, and R. Duin. "A simplified extension of the area under the ROC to the multiclass domain." Seventeenth annual symposium of the pattern recognition association of South Africa. 2006.

[22] Barber, C. Bradford, David P. Dobkin, and Hannu Huhdanpaa. "The quickhull algorithm for convex hulls." ACM Transactions on Mathematical Software (TOMS) 22.4 (1996): 469-483.

[23] http://www.qhull.org/

[24] Wolpert, David H. "Stacked generalization." Neural networks 5.2 (1992): 241-259.

[25] Mason, L., et al. "Boosting algorithms as gradient descent in function space (Technical Report)." RSISE, Australian National University (1999).

[26] Dupret, Georges, and Masato Koda. "Bootstrap re-sampling for unbalanced data in supervised learning." European Journal of Operational Research 134.1 (2001): 141-156.

[27] Cieslak, David, and Nitesh Chawla. "Learning decision trees for unbalanced data." Machine learning and knowledge discovery in databases (2008): 241-256.

[28] Liu, Wei, et al. "A robust decision tree algorithm for imbalanced data sets." Proceedings of the 2010 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2010.

[29] Loh, Wei-Yin, and Yu-Shan Shih. "Split selection methods for classification trees." Statistica sinica (1997): 815-840.

[30] Quinlan, J., and R. Cameron-Jones. "Oversearching and layered search in empirical learning." breast cancer 286 (1995): 2-7.

[31] Zeileis, Achim, Torsten Hothorn, and Kurt Hornik. "Model-based recursive partitioning." Journal of Computational and Graphical Statistics 17.2 (2008): 492-514.

[32] De Cock, Dean. "Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project." Journal of Statistics Education 19.3 (2011).

[33] Rokach, Lior. "Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography." Computational Statistics & Data Analysis 53.12 (2009): 4046-4072.

[34] Rokach, Lior. "Ensemble-based classifiers." Artificial Intelligence Review 33.1 (2010): 1-39.

[35] Johnson, Rie, and Tong Zhang. "Learning nonlinear functions using regularized greedy forest." IEEE transactions on pattern analysis and machine intelligence 36.5 (2014): 942-954.

[36] James, Gareth, et al. An introduction to statistical learning. Vol. 6. New York: springer, 2013.

[37] Fortmann-Roe, Scott. "Understanding the bias-variance tradeoff." (2012).

[38] https://www.kaggle.com/c/otto-group-product-classification-challenge/discussion/14335

[39] https://ww2.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt

[40] Dietterich, Thomas G. "Ensemble methods in machine learning." International workshop on multiple classifier systems. Springer Berlin Heidelberg, 2000.

[41] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

[42] Fawcett, Tom. "An introduction to ROC analysis." Pattern recognition letters 27.8 (2006): 861-874.

[43] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[44] Banerjee, Mousumi, Ying Ding, and Anne-Michelle Noone. "Identifying representative trees from ensembles." Statistics in medicine 31.15 (2012): 1601-1616.

[45] Chipman, Hugh A., Edward I. George, and Robert E. McCulloch. "Extracting representative tree models from a forest." IPT GROUP, IT DIVISION, CERN. 1998.

[46] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Vol. 1. Springer, Berlin: Springer series in statistics, 2001.

[47] Lichman, Moshe. "UCI machine learning repository, 2013." URL http://archive. ics. uci. edu/ml 8 (2015).

[48] Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas. "Supervised machine learning: A review of classification techniques." (2007): 3-24.

[49] Sill, Joseph, et al. "Feature-weighted linear stacking." arXiv preprint arXiv:0911.0460 (2009).

[50] Czepiel, Scott A. "Maximum likelihood estimation of logistic regression models: theory and implementation." Available at czep. net/stat/mlelr. pdf (2002).

[51] Archer, Kellie J., and Ryan V. Kimes. "Empirical characterization of random forest variable importance measures." Computational Statistics & Data Analysis 52.4 (2008): 2249-2260.

[52] Monteith, Kristine, et al. "Turning Bayesian model averaging into Bayesian model combination." Neural Networks (IJCNN), The 2011 International Joint Conference on. IEEE, 2011.

[53] Allwein, Erin L., Robert E. Schapire, and Yoram Singer. "Reducing multiclass to binary: A unifying approach for margin classifiers." Journal of machine learning research 1.Dec (2000): 113-141.

[54] Aly, Mohamed. "Survey on multiclass classification methods." Neural Netw (2005): 1-9.

[55] Loh, Wei-Yin. "Regression tress with unbiased variable selection and interaction detection." Statistica Sinica (2002): 361-386.

[56] Landgrebe, Thomas CW, and Robert PW Duin. "Approximating the multiclass ROC by pairwise analysis." Pattern recognition letters 28.13 (2007): 1747-1758.

[57] Hoens, T., et al. "Building decision trees for the multi-class imbalance problem." Advances in knowledge discovery and data mining (2012): 122-134.

[58] Kearns, Michael J., and Yishay Mansour. "A Fast, Bottom-Up Decision Tree Pruning Algorithm with Near-Optimal Generalization." ICML. Vol. 98. 1998.

[59] Bouckaert, Remco. "Efficient AUC learning curve calculation." AI 2006: Advances in Artificial Intelligence (2006): 181-191.