Roughened Random Forests

for Binary Classification

by

Kuangnan Xiong

A Dissertation

Submitted to the University at Albany, State University of New York

in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

School of Public Health

Department of Epidemiology and Biostatistics

May 2014

Roughened Random Forests

for Binary Classification

by

Kuangnan Xiong

Abstract

Binary classification plays an important role in many decision-making processes. Random forests can build a strong ensemble classifier by combining weaker classification trees that are de-correlated. The strength and correlation among individual classification trees are the key factors that contribute to the ensemble performance of random forests. We propose roughened random forests, a new set of tools which show further improvement over random forests in binary classification. Roughened random forests modify the original dataset for each classification tree and further reduce the correlation among individual classification trees. This data modification process is composed of artificially imposing missing data that are missing completely at random and subsequent missing data imputation.

Through this dissertation we aim to answer a few important questions in building roughened random forests: (1) What is the ideal rate of missing data to impose on the original dataset? (2) Should we impose missing data on both the training and testing datasets, or only on the training dataset? (3) What are the best missing data imputation methods to use in roughened random forests? (4) Do roughened random forests share the same ideal number of covariates selected at each tree node as the original random forests? (5) Can roughened random forests be used in medium- to high- dimensional datasets?

Acknowledgements

First, I would like to express my deepest gratitude to my dissertation committee chairs, Dr. Robert Pruzek and Dr. Recai Yucel. Dr. Pruzek encouraged and helped with my transition from the Department of Biological Sciences to the Department of Epidemiology and Biostatistics. He was extremely supportive during my long search for a dissertation topic. From Dr. Pruzek, I learned the importance of patience and determination in both research and life. Dr. Yucel guided me through the writing process for this dissertation and I benefited tremendously from his numerous constructive suggestions. Dr. Yucel has had a huge influence on the way I think and write about research.

I would also like to sincerely thank the members of my dissertation committee, Dr. A. Gregory DiRienzo and Dr. Tao Lu. I enjoyed all of my discussions with them, and I appreciate their insightful feedback.

I owe many thanks to the faculty and staff members at the University at Albany School of Public Health. In particular, I would like to thank Dr. Howard Stratton, Dr. Igor Zurbenko, Dr. Yuchi Young, Dr. Edward Fitzgerald, Michael Zdeb, Lenore Gensburg, Judith Pelton and Nikki Malachowski. I would also like to acknowledge Dr. Ing-Nang Wang from the Department of Biological Sciences.

During my doctoral studies, I have been very fortunate to work with a terrific group of people at the New York State Department of Health. In particular, I would like to thank

Dr. Marilyn Kacica, Dr. Valerie Haley, Dr. Donna Noyes, Kristen Lawless, Todd Gerber and Dr. Mycroft Sowizral. I would also like to acknowledge Kristen Lawless and Megan Gallagher who kindly helped me with editing this dissertation.

I am indebted to the amazing classmates and friends I met over the years at the University at Albany. I would particularly like to thank Yi Sun, Xuelin Weng, Changning Xu, Ming Liu, Fangtao He, Meng Wu, Yan Wu, Yan Xu, Enxu Zhao, Dr. Lei Chen, Dr. Mingzeng Sun, Dr. Zhen Huang, Dr. Qingmei Weng, Dr. Hong Wu, Xiaomian Zheng, Daniel Reynolds, Diana Nadler, Sairam Chinnam, Dr. Yongping Shao, Dr. Qinglu Zeng, Dr. Daiying Xu, Mingliang Wan, Dr. Jingjing Xie, Dr. Yuanyuan Liu, Dr. Shengchun Wang, Dr. Yan Han, Dr. Tao He, Dr. Quan He, Dr. Congzhou Wang and Jianchao Zhang.

I would like to thank my family for their unwavering love. My parents, Shuyuan Xiong and Zhifang Bo, have always been eager and excited to hear about my research and studies at the University at Albany. Their encouragement of my doctoral pursuit has meant so much to me. Last, and most importantly, I would like to thank my wife, Jin, who has willingly stood by my side through this seemingly endless PhD journey. I would not have finished my doctoral studies without Jin’s support.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

The Oxford Advanced Learner's Dictionary defines "wisdom" as "the ability to make sensible decisions and give good advice because of the experience and knowledge that you have". In statistical terms, making decisions based on our past experience and knowledge is called "supervised learning". In addition, a decision can be usually simplified as the answer to a clearly stated "Yes or No" question, or in statistical terms, a binary classification problem. Therefore, statistically speaking, a good supervised learning algorithm for binary classification is an important part of "wisdom".

In the past decades we have seen the exponential increase of information storage capacity, which makes it impossible for us to solely rely on the human brain for information processing. Developing computer software with advanced supervised learning algorithms for binary classification is of paramount importance, as it can help us gain analytical "wisdom" and make better decisions.

1.1 Brief Summary of Binary Classification Methods

In the following discussion, we let N denote a number of sampled units which are observed on their M covariates. We denote the covariate matrix as X. For each observation x_i, $x_i = (x_{i1}, x_{i2}, x_{i3}, .. x_{iM})$, where i = 1, 2, ..., N. Let X_1, X_2, X_3, $\ldots X_M$ denote each of the M covariates. Further, we will use Y to denote a binary outcome variable Y. For each individual y_i, there are two classes labeled as 0 and 1, and we let

p_i denote $P(y_i = 1|x_i)$. In supervised learning literature, we usually use a training dataset to find the function that defines the relationship between X and Y, and then apply the function in the testing dataset to test the effectiveness of the learning algorithm. In our case, these learning algorithms are binary classification methods. The training dataset and testing dataset are assumed to be from the same data sample. The Y values of the testing dataset are predicted from the X values in the testing dataset based on the relationship between X and Y in the training dataset. The binary classification method is assessed by comparing the predicted Y values and the observed Y values in the testing dataset.

1.1.1 Logistic Regression

Logistic regression is one of the most widely used parametric methods for binary classification. Logistic regression models the probability that Y=1 given X. For $p_i = P(y_i = 1|x_i)$, the logit transformation of p_i is a linear function of x_i:

$$\log\left(\frac{p_i}{1-p_i}\right) = x_i{}^T\beta + \beta_0 \qquad (1.1)$$

where β is the coefficient vector and β_0 is the intercept. Maximum Likelihood methods (ML) can be used to estimate β and β_0 (Hosmer & Lemeshow, 2000). After obtaining the estimated β and β_0 values from the training dataset, we can directly apply them in the testing dataset using the above equation (1.1) to predict the corresponding p_i value. Conventionally, we use $p_i \leq 0.5$ and $p_i > 0.5$ to separate both classes of 0 and 1.

1.1.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) uses the linear combination of covariates to separate the two classes of 0 and 1 (Ripley, 1996; Hastie, et al., 2009). Let C_i denote a linear combination of x_i as shown below.

$$C_i = x_i^T\beta + \beta_0 \qquad (1.2)$$

Let S_w denote the variance of C_i within both classes, and let S_b denote the variance of C_i between both classes. LDA aims to find the β vector which maximizes the ratio of S_b/S_w. After getting the estimated β and β_0 values from the training dataset, we can directly apply them in the testing dataset using the above equation (1.2) to predict the corresponding C_i value. Conventionally, we use $C_i \le 0$ and $C_i > 0$ to separate both classes of 0 and 1.

1.1.3 Naive Bayes Classifier

The naive Bayes classifier assumes the independence between covariates within each class. Therefore, based on Bayes' rule, we can get

$$P(Y = 1|X) = \frac{\pi_1 \prod_1^M f_1(X_j)}{\pi_1 \prod_1^M f_1(X_j) + \pi_0 \prod_1^M f_0(X_j)} \qquad (1.3)$$

In the above equation (1.3), π_1 and π_0 denote the proportion of observations with Y=1 and Y=0, or the default prior. Further, $f_1(X_j)$ denotes the marginal density for covariate X_j when Y=1, $f_0(X_j)$ denotes the marginal density for covariate X_j when Y=0 (Hastie, et al., 2009). The likelihood function for Y=1 is $\prod_1^M f_1(X_j)$, and the likelihood function for Y=0 is $\prod_1^M f_0(X_j)$. For each observation in the testing dataset, we can get the estimated p_i using the above equation (1.3). Conventionally, we use $p_i \le 0.5$ and $p_i > 0.5$ to separate both classes of 0 and 1. The naive Bayes classifier is easy and quick to build due to its simplicity, and it is not sensitive to irrelevant covariates. It does not, however, take covariate interactions into consideration due to its assumption of independence between covariates within each class.

1.1.4 K-Nearest Neighbor

K-Nearest Neighbor (KNN) does not train a model for prediction. For each new observation $\mathrm{u_i}$ in the testing dataset, KNN calculates the Euclidean distance $|\mathrm{x_i} - \mathrm{u_i}|$ between $\mathrm{u_i}$ and all current observations $\mathrm{x_i}$ in the training dataset (Hastie, et al., 2009).

$$|\mathrm{x_i} - \mathrm{u_i}| = \sqrt{\sum_{\mathrm{j=1}}^{\mathrm{M}} (\mathrm{x_{ij}} - \mathrm{u_{ij}})^2} \qquad (1.4)$$

Based on the above equation (1.4), we can find the K observations with the shortest Euclidean distances to $\mathrm{u_i}$. Then, the frequency of classes within these K observations are used to predict the class for $\mathrm{u_i}$. KNN does not require model building, but it is computationally expensive as it needs to calculate the Euclidean distances between each new observation and all observations in the training dataset. Therefore, it is difficult to apply KNN in high-dimensional datasets.

1.1.5 Neural Networks

Neural networks are created by imitating the function of neural systems in living organisms. A model based on neural networks can usually include an input layer, a hidden layer and an output layer as shown in Figure 1 (Ripley, 1996; Hastie, et al., 2009).

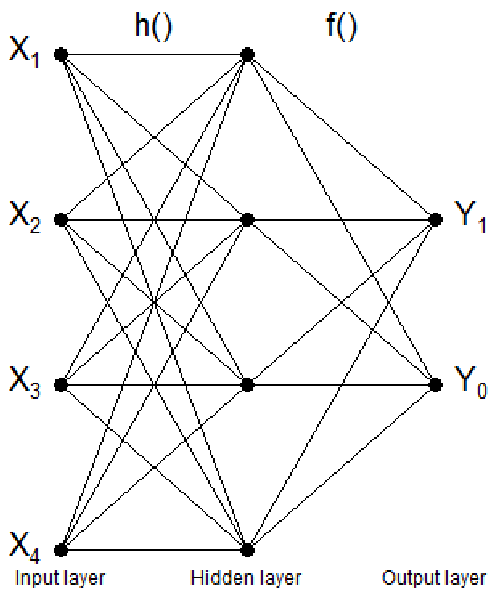


Figure 1: An illustration of neural networks

Mathematically, it can be written as:

$$g(x_i) = f(h(x_i^T w + w_0)^T \beta + \beta_0) \qquad (1.5)$$

The two linking functions h() and f() are usually both non-linear functions, which help transform the original data from the input layer to the hidden layer and then to the output layer. The output layer gives out the probability for each class. w and β are weights for the input layer and the hidden layer. w_0 and β_0 are intercepts. For each observation in the testing dataset, the estimated probability for both classes of 0 and 1 can be obtained using the above equation (1.5). Then, the class with the larger probability is selected as the final prediction. Neural networks are good at handling non-linearity in binary classification, but they often suffer from local minima and they tend to overfit the training data.

1.1.6 Support Vector Machines

Support vector machines separate the two classes within the dataset by a hyperplane. For a vector space of dimension M, a hyperplane has a dimension of (M-1). If the weighted average of covariates cannot separate the two classes, quadratic terms are added. If separation still does not occur, cubic terms are added. As the dimensionality increases, a separating hyperplane can usually be found between these two classes. The optimal hyperplane should have the maximum distance to the closest vectors from both classes. It can be written as in the equation below.

$$g(x_i) = h(x_i)^T\beta + \beta_0 \qquad (1.6)$$

Figure 2 illustrates an example of optimal linear hyperplane that separates two classes in a dataset with two covariates. The dots and circles represent vectors for two different classes. The vectors in each class which have minimum distances to the optimal hyperplane are called support vectors (Cortes & Vapnik, 1995).

For each observation in the testing dataset, we can obtain the corresponding $g(x_i)$ using the above equation (1.6). The sign of $g(x_i)$ can help us decide the predicted class of the new observation based on which side of the optimal hyperplane it falls in. Support vector machines and neural networks are both good at handling non-linearity in data. Support vector machines, however, do not suffer from local minima and they are less likely to overfit the training data (Hastie, et al., 2009).

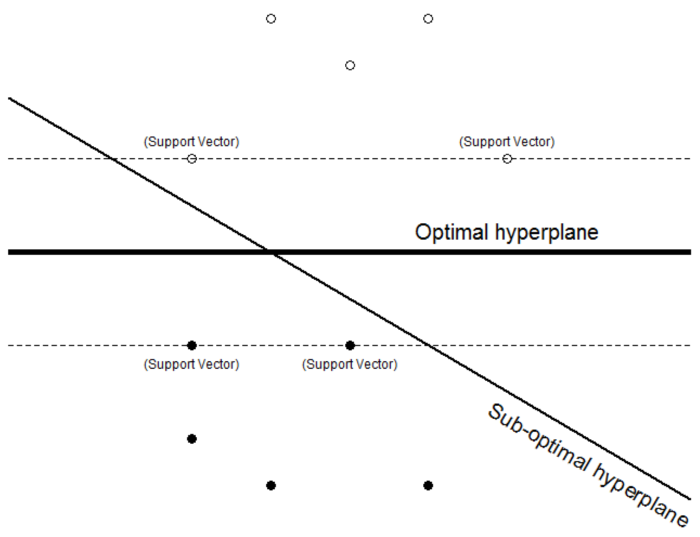


Figure 2: An illustration of support vector machines

1.1.7 Classification Tree

A classification tree is composed of a sequence of tree nodes. Each tree node applies a certain rule to split the sample into two subsamples (Breiman, et al., 1984). Figure 3 illustrates a simple classification tree which answers the question “Is your BMI in the normal range?”. First, we use “BMI < 18.5 or BMI >= 18.5” to split the sample into two subsamples. The subsample on the left, with BMI < 18.5, is not in the normal BMI range. For the subsample on the right with BMI >= 18.5, we use “BMI >= 25 or BMI < 25” to further split it into two subsamples. The subsample on the left, with BMI >= 25, is again not in the normal BMI range. The subsample on the right, with BMI < 25 and BMI >= 18.5, is in the normal BMI range.

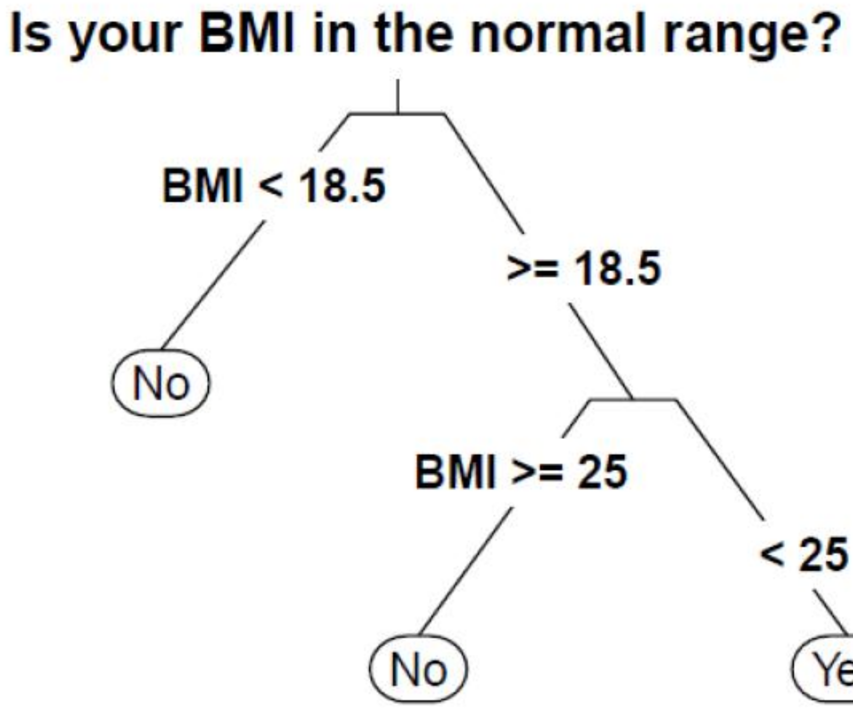


Figure 3: An illustration of classification tree

There are different splitting criteria to build the nodes for a classification tree, Leo Breiman (1984) originally proposed to use the Gini impurity criterion (G) at each node inside the classification tree. For each subset after splitting, if the proportion for each type of binary response is P_0 and P_1, Gini impurity criterion in this subset can be computed as ($G=1- P_0^2- P_1^2$). Gini impurity criterion reaches the minimum value of 0 when all observations in the subset belong to a single type of binary response, and it reaches the maximum value of 0.5 when all observations in the subset are evenly distributed between two binary responses. Each split should minimize the weighted (by proportion of sample in the subset) sum of Gini impurity, and the decrease in Gini impurity should at least reach a threshold value that is set as penalization parameter on complexity. For each terminal node, the majority class is used as the predicted class for all observations that fall within this terminal node.

Figure 4 illustrates the difference between a parametric logistic regression model and a non-parametric classification tree with a simple example. The original function of Y=f(X) is as below: Y=1 when X is between 25 and 50, or between 75 and 100; Y=0 when X is between 0 and 25, or between 50 and 75. When we directly apply logistic regression on the data, it does not display the relationship between X and Y correctly. When we apply a classification tree on the data, however, it perfectly explains the relationship between X and Y in a tree structure.

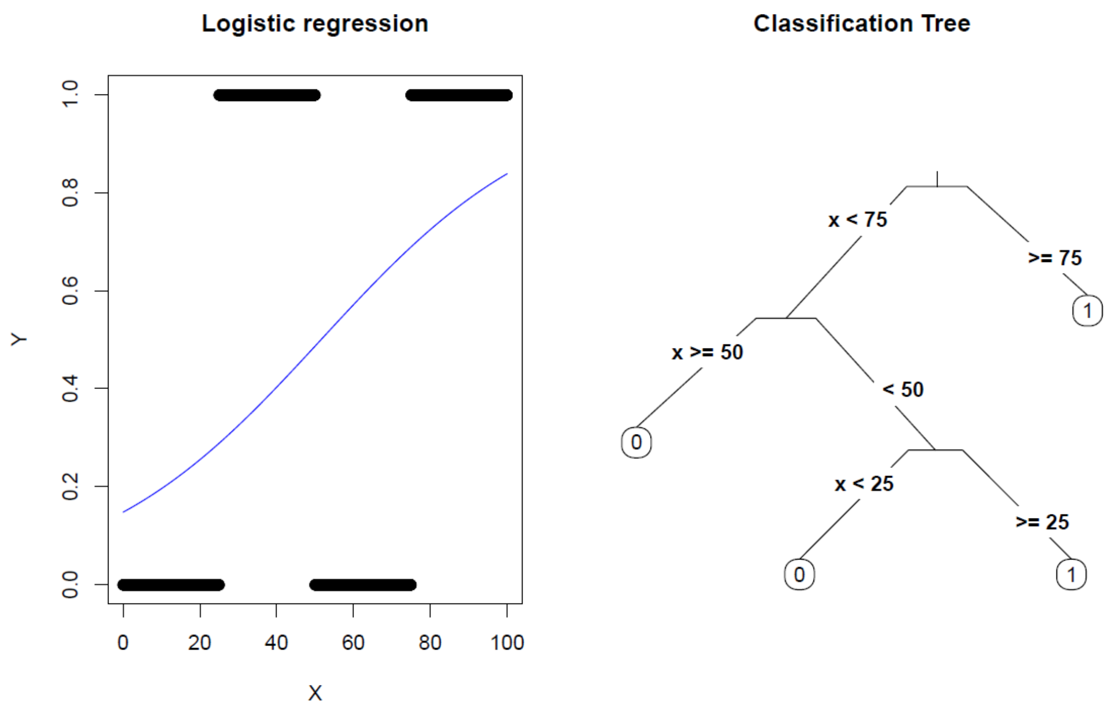


Figure 4: A comparison of logistic regression and classification tree

A single classification tree can be intuitive to understand and easy to apply in datasets with simple patterns. However, for datasets with a large number of covariates and high-order interactions, a simple classification tree can be ineffective for prediction. If a single tree seems problematic, many trees might be combined to improve the prediction of responses.

1.1.8 Bagging, Random Forests and Boosting

Leo Breiman introduced the concept of bagging (short for “bootstrap aggregating”), which is able to generate a wide variety of individual classifiers when data are perturbed (Breiman, 1996). Bagging was found to improve the performance of classification trees. Tin Ho proposed the idea of random decision forests (Ho, 1995) built with multiple classification trees by sub-sampling of covariates at each tree node. Leo Breiman further combined bagging and random decision forests into random forests (Breiman, 2001). For a dataset of size N with M covariates, the algorithm to build the random forests can be briefly explained as below.

First, randomly sample N data records with replacement; second, find a split among m ($m<M$) randomly chosen covariates at each tree node, and each tree is grown to the largest extent without pruning; third, repeat the above process to build multiple classification trees. Each individual tree produces a vote, and the final classification is based on averaging the votes from all individual trees (Breiman, 2001). Random forests have been widely used across different research disciplines since then due to their

excellent performance in classification and regression (Palmer, et al., 2007; Touw, et al., 2013; Cutler, et al., 2007; Cordell, 2009; Gislason, et al., 2006).

Along with the success of bagging and random forests, another widely acclaimed ensemble algorithm called adaptive boosting (Adaboost) was developed (Freund & Schapire, 1997). In contrast to the parallel approach of building individual classifiers in random forests, boosting takes the iterative approach. Instead of building a large un-pruned individual classification tree in a bootstrap sample, a simple decision stump (one-split tree) is built at each step with the whole dataset. Then, the data is reweighted by putting more weights on the wrongly classified data points before another stump is built. At every iterative step, the data is reweighted by putting more emphasis on previously misclassified data points. Each individual classifier is also given a weight based on its performance. The final ensemble learner is a weighted combination of stumps that have superb performance, especially when noise level is low.

Leo Breiman praised Adaboost with trees as the “best off-the-shelf classifier in the world”. Discussions regarding the theories behind the success of boosting have not ceased since its invention (Mease & Wyner, 2008). In the meantime, many variants of Adaboost have been developed by tweaking the weights for wrongly classified data points, the tree depths and regularization for each individual classifier. The popular variants include gradient boosting (Friedman, 2001) and Bayesian additive regression trees (Chipman, et al., 2010). To restrict the contribution of individual classification trees, gradient boosting uses a shrinkage parameter and Bayesian additive regression trees use a regularization prior. Gradient boosting can handle categorical variables directly, while Bayesian additive regression trees need to transform categorical variables into dummy variables before use.

1.2 Comparison of Binary Classification Methods

Caruana (2006) did an extensive empirical comparison of the ten most popular supervised learning algorithms: support vector machines, neural networks, logistic regression, naive Bayes classifier, KNN, random forests, classification trees, bagged decision trees, boosted decision trees, and boosted stumps. Eleven binary classification problems were tested with these ten algorithms with regards to eight different performance measures. The results suggested that boosted decision trees ranked first overall with a 58% chance of being the best algorithm, and random forests ranked second overall with a 39% chance of being the best algorithm (Caruana & Niculescu-Mizil, 2006). Even though the "no free lunch" theorem (Wolpert & Macready, 1997) suggests that there is no universally best supervised learning algorithm, boosted decisions trees and random forests often end up as the top choices in binary classification problems. Past winners of data prediction competitions on the Kaggle website (Kaggle, 2010) also often included boosted decision trees and random forests in their final winning models to make binary classification predictions.

Boosted decision trees and random forests are both ensemble classifiers based on classification trees. While boosted decision trees build classification trees step-by-step with each following classification tree dependent on the previous tree, the classification trees within random forests are independent of each other. Boosted decision trees usually take more calibration and longer computation time than random forests. In this dissertation, we will focus on further improving random forests in binary classification.

1.3 Dissertation Outline

In Chapter 2, an overview of random forests (RF) is given and previous attempts to improve random forests are summarized. Then, a new kind of random forests called "Roughened Random Forests (RRF)" is introduced. And an algorithm, which we call "Roughened Random Forests - A (RRFA)", is implemented. RRFA intends to improve the diversity within random forests by imposing missing data and then replacing them with median/mode imputation. RRFA shows improvements over the original random forests in binary classification with respect to misclassification error and area under the curve (AUC).

In Chapter 3, three new RRF algorithms are further explored. They are called "Roughened Random Forests - B (RRFB)", "Roughened Random Forests - C (RRFC)" and "Roughened Random Forests - D (RRFD)". RRFB restricts the introduction of missing data to the training dataset. RRFC explores different imputation methods for missing data. RRFD seeks to find the ideal number of variables selected at each tree node within RRFB. RRFB leads to better AUC performance as well as shorter computation time than RRFA. RRFC and RRFD can be both better than RRFB with regards to AUC at the expense of longer computation time.

In Chapter 4, the RRFE algorithm is presented for binary classification in medium- to high-dimensional datasets. The major difference between RRFE and RRFB is the selective introduction of missing data based on variable importance according to the original random forests. RRFE can lead to improved AUC over the original random forests in both medium- to high-dimensional datasets. For high-dimensional microarray datasets, after applying variable selection based on variable importance from the original random forests, RRFB and RRFE can both lead to improvement with regards to AUC.

The conclusion and future research topics are discussed in Chapter 5.

Chapter 2. Roughened Random Forests (RRF)

An Improved Random Forests Approach in Binary Classification

Abstract

Random forests (RF) can build a strong ensemble classifier by combining a diverse set of weaker classifiers. The strength of individual classifiers and the correlations among them are the key factors of the random forests' ensemble classification performance. Our work aims to improve the binary classification performance of random forests by modifying the original dataset before building each individual classifier. This modification decreases correlations among individual classifiers by imposing missing values under a mechanism that is missing completely at random (MCAR). These missing values are then replaced by single imputation or multiple imputation. We call this new method "Roughened Random Forests (RRF)". An algorithm which we call "Roughened Random Forests - A (RRFA)" is introduced to implement RRF. We demonstrate and contrast performance of RRFA with RF in real-life datasets as well as simulated datasets. We observe significant improvements in RRFA over RF with respect to measures including misclassification error and area under the curve (AUC).

2.1 Introduction

2.1.1 Random Forests

Random forests are based on un-pruned growth of individual classification trees through bagging of data as well as sub-sampling of covariates. Each individual classification tree starts by sampling the original dataset with replacement using the same sample size. For sampling with replacement using N observations, the probability that one particular observation is selected can be expressed as $1-\left(1-\frac{1}{N}\right)^{N}$, which is in the value range of (0.632, 0.635) for $N \geq 65$. Therefore, about 63% of the observations are used to build each individual classification tree, the remaining 37% of the sample, or out-of-bag (OOB) sample, can be used for internal validation.

The instability of individual classification trees, combined with the variety of combinations in bagging of data as well as sub-sampling of covariates, can help build diverse individual classifiers in binary classification. There is usually a trade-off relationship between strength and correlation of individual classification trees, which together determine the random forests' classification performance.

There are different splitting criteria to build the nodes for a classification tree, Leo Breiman originally proposed to use the Gini impurity criterion (G) at each node inside the classification tree (Breiman, et al., 1984). For a single classification tree, the decrease in Gini impurity criterion should at least reach a threshold value that is set as the penalization parameter on complexity. For random forests, there is usually no penalization parameter on complexity, and an individual classification tree can grow as complex as possible.

Median/mode imputation and proximity-based imputation were both used in the original random forests to deal with missing data (Breiman & Cutler, 2004). Median/mode imputation, implemented by the R (R Development Core Team, 2012) function `na.roughfix` in `randomForest` package (Liaw & Wiener, 2002), performs median imputation on continuous variables and mode imputation on categorical variables. For each pair of cases, the proximity is defined as the proportion of trees where these two cases occupy the same terminal node. The proximity matrix quantifies the similarity between each pair of cases based on their location in the individual tree terminals, and the similarities can be used as weights for missing data imputation. The proximity-based imputation is dependent on the binary outcome and therefore it cannot be used when the binary outcome is unknown. The median/mode imputation is not dependent on the binary outcome and therefore it can be used for predicting new classifications when there are missing data in the covariates.

2.1.2 Related work

Rotation forests (Rodriguez, et al., 2006) use a combination of covariate sub-sampling and rotation of covariate axes by principal component analysis to improve the diversity and accuracy within individual classifiers. The rotation forest was favored over random forests on a random selection of 33 benchmark datasets from University of California Irvine (UCI) Machine Learning Repository (Bache & Lichman, 2013).

Robnik-Sikonja (2004) presented two new approaches aimed to improve performance of random forests. The first approach was to increase the diversity of classifiers by using a combination of different node splits measures called Gain ratio, MDL, ReliefF in addition to the original Gini impurity criterion. This method was found to significantly improve prediction accuracy but not the area under the curve (AUC). The second approach was to use weighted voting instead of un-weighted voting as adopted by the original random forests. For each case to be classified, the 30 cases most similar to this case were selected. The final prediction for this case was a weighted average of select trees from the trained random forests based on their performances on these 30 cases while they were not used to build these individual trees. The weighted voting method was found to significantly improve accuracy and AUC (Robnik-Sikonja, 2004).

Conditional inference forests are based on conditional inference trees (Hothorn, et al., 2006) instead of regular classification trees. The main difference between a conditional inference tree and a regular classification tree is that a conditional inference tree first picks the best variable based on statistical testing and then picks the best split within this variable, while a regular classification tree picks the best split among all available variables which can lead to biased selection for variables with more categories. Conditional inference forests were found to be less biased than the original random forests in assessing variable importance.

Oblique random forests (Menze, et al., 2011) are based on oblique splits instead of orthogonal splits used by the original random forests. The main difference between an oblique split and an orthogonal split is that the oblique split is based on a combination of variables while an orthogonal split is based on a single variable. In oblique random forests, linear discriminant models can be used to find the optimum splits. Menze (2011) found oblique random forests to outperform the original random forests in numerical and spectral data.

Voting on Classifications from Imputed learning sets (VCI) was introduced to improve classification accuracy in different supervised learning algorithms (Su, et al., 2009). For a classification dataset, VCI randomly imposes 30% of missing data nine times, then missing values are imputed for each of them. A classifier is used in each of these nine imputed datasets and the resulting predictions are combined by majority rule. Both single imputation and multiple imputation are used by VCI. Ten different classifiers, including the naive Bayes classifier, support vector machines , neural networks, random forests and several others, were each applied in ten different complete datasets. VCI was found to improve classification accuracies in KNN, naive Bayes classifier, support vector machines and neural networks, but not in random forests. The original random forests were able to achieve better classification accuracy than all other classifiers with or without using VCI.

2.2 Background

2.2.1 Notations and Assumptions

We will let X denote a covariate matrix with N rows corresponding to observational units and M columns corresponding to a set of covariates used in forming the classification trees. Our particular use of the random forests is to predict an outcome variable that is of binary nature. We will use Y to denote this binary outcome variable. We assume that

$Y \sim \text{Bernoulli}\,(P(Y = 1))$. A value of 0 (or negative) and a value of 1 (or positive) are used to refer to the two different values of Y. We will refer to these as "classes" in Y. Prediction of the probability of a positive class using available covariates will be denoted as $\widehat{P}(Y = 1|X)$. Similarly, $\widehat{Y}$ will indicate the predicted outcome for Y. Y_c will be used to indicate a cut off value to predict Y, i.e. $\widehat{Y}$, based on $\widehat{P}(Y = 1|X)$. Note that Y_c is in the range of 0 and 1. The default value of Y_c is usually set as 0.5 in binary classification. The rule to reach $\widehat{Y}$ based on $\widehat{P}(Y = 1|X)$ and Y_c is as below.

$$\widehat{Y} = \begin{cases} 1, & \widehat{P}(Y = 1|X) > Y_c \\ 0, & \widehat{P}(Y = 1|X) \leq Y_c \end{cases}.$$

We will let FP refer to false positive, a case with $\widehat{Y} = 1$ and $Y = 0$. And let FN refer to false negative, a case with $\widehat{Y} = 0$ and $Y = 1$. Similarly, we will let TP refer to true positive, a case with $\widehat{Y} = 1$ and $Y = 1$ and let TN refer to true negative, a case with $\widehat{Y} = 0$ and $Y = 0$.

We will let FPR refer to false positive rate, or $P(\widehat{Y} = 1 \mid Y = 0)$. And let FNR refer to false negative rate, or $P(\widehat{Y} = 0 \mid Y = 1)$. Similarly, we will let TPR refer to true positive rate, or $P(\widehat{Y} = 1 | Y = 1)$ and let TNR refer to true negative rate, or $P(\widehat{Y} = 0 | Y = 0)$.

We will generate receiver operating characteristic (ROC) curve, by plotting FPR on the X axis and TPR on the Y axis as shown in Figure 5. ROC curve covers all possible Y_c values when Y_c moves from 0 to 1. Let AUC denote the size of the area under the ROC Curve. As both TPR and FPR are within the range of 0 and 1, the maximum value of

AUC is 1. Let t denote the Y_c value between 0 and 1, we can derive the AUC value using $\mathrm{AUC} = \int \mathrm{TPR}(t)\mathrm{FPR}'(t)dt$.

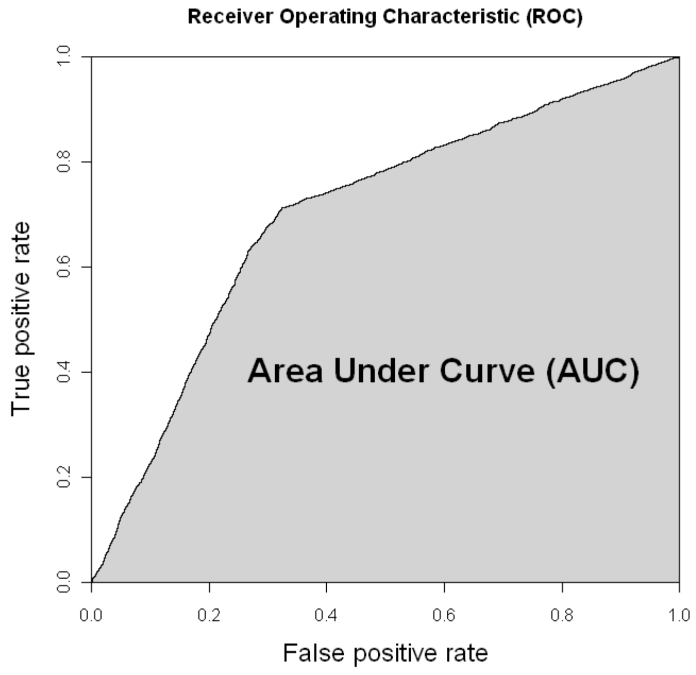


Figure 5: Area under the curve (AUC)

We will let Z denote the variable of interest for which missingness will be imposed. We will use R as the missingness indicator variable for Z, i.e. R = 1 if Z is observed and R = 0 if Z is missing. Missing values will be imposed under the missing completely at random (MCAR) mechanism (Little & Rubin, 2002). MCAR means that the mechanism creating missingness is independent of both observed and missing values. It can be written as $P(R \mid Z_{obs}, Z_{mis}, X) = P(R)$.

We will let RF refer to random forests, a combination of independently constructed classification trees. RF is usually called an ensemble classifier. We will use $\mathrm{N_{tree}}$ to denote the number of classification trees inside the random forests. The default value of $\mathrm{N_{tree}}$ is usually set as 500. Let m denote the number of randomly chosen covariate candidates at each node of a single classification tree within the random forests. Let $\lfloor\sqrt{M}\rfloor$ denote the integer part of the square root of M. The default value of m is usually set as $\lfloor\sqrt{M}\rfloor$. We will use $\mathrm{N_{tr}}$ to denote the sample size in the training dataset, and $\mathrm{N_{te}}$ to denote the sample size in the testing dataset. Let $\mathrm{C_n^k}$ denote the number of possible combinations of n items taken k at a time without repetition.

We will use RRF to refer to roughened random forests, a new random forests approach proposed in this dissertation. As missing data are imposed in RRF, we will use $\mathrm{MIS_{pct}}$ to denote the percentage of missing data. We will use WTL ("Win/Tie/Loss") to compare RRF and RF. For example, a simulation experiment is repeated 50 times using RRF and RF. If RRF is better than RF in 30 experiments, equal to RF in 5 experiments, and worse than RF in 15 experiments, we can say that the RRF has a WTL value of "30/5/15" over RF.

2.2.2 Definitions

Throughout this paper, we will make use of the following definitions. These definitions are commonly used in the context of binary classification and missing data analysis.

Misclassification error: We will denote misclassification error as e which indicates the percentage of false positive cases and false negative cases among all cases, or P $(\hat{Y} \neq Y)$.

Accuracy: Percentage of true positive cases and true negative cases among all cases, or P $(\hat{Y} = Y)$. When misclassification error is e, accuracy is just (1-e).

Bayes error: Statistically the lowest possible misclassification error rate for a given classification problem. Bayes error exists due to the overlap between different classes' statistical distributions.

Median/mode imputation: Impute the missing values in a continuous variable by its median value and impute the missing values in a categorical variable by its mode value.

Mean/mode imputation: Impute the missing values in a continuous variable by its mean value and impute the missing values in a categorical variable by its mode value.

Minimum-value/mode imputation: Impute the missing values in a continuous variable by its minimum value and impute the missing values in a categorical variable by its mode value.

Maximum-value/mode imputation: Impute the missing values in a continuous variable by its maximum value and impute the missing values in a categorical variable by its mode value.

Variable importance: A variable's importance is measured by the average decrease in Gini impurity criterion due to splitting on this variable among all classification trees.

Relative importance: A variable's relative importance is defined by its variable importance divided by the maximum variable importance among all covariates in this dataset.

2.2.3 Metrics for Performance Assessment

Misclassification error and AUC are two highly used performance measures in binary classification. AUC is also known as "c statistics" and it is similar to the Mann–Whitney U statistic or Wilcoxon rank-sum test (Hastie, et al., 2009). In the first ten Kaggle competitions focusing on binary classifications, eight of them used AUC or equivalent as the measure to assess binary classification performance (Kaggle, 2010). The maximum value of AUC is 1 and the minimum value of AUC is 0. Statistically speaking the minimum value of AUC should be 0.5, which can be achieved by random guesses. In practice, AUC value can get below 0.5. For AUC value under 0.5, we can make it reach over 0.5, or (1 - AUC), by flipping the binary predictions from negative to positive (0 to 1), and vice versa, positive to negative (1 to 0).

2.2.4 Pearson Correlation Coefficient (or Phi Coefficient)

For any two binary classifiers (C_i, C_j), predictions of any observation can have four distinct combinations: (0,1), (1,1), (1,0) and (0,0). Suppose that we have ***a*** pairs of (0,1), ***b*** pairs of (1,1), ***c*** pairs of (1,0) and ***d*** pairs of (0,0) as shown in Table 1.

Table 1: Four possible combinations of predictions by two binary classifiers

Predictions by C_i	Predictions by C_j	Number of pairs
0	1	***a***
1	1	***b***
1	0	***c***
0	0	***d***

The Pearson correlation coefficient (r) for two sets of binary predictions is also called phi coefficient. The phi coefficient assumes that the two sets of binary predictions follow bivariate discrete distribution, while tetrachoric correlation coefficient assumes that the two sets of binary predictions follow bivariate normal distribution (Ekstrom, 2011). In our binary classification problems, the predictions are more likely to be following bivariate discrete distribution. Therefore, we will choose the phi coefficient over the tetrachoric correlation coefficient. And the phi coefficient (r) can be calculated as below.

$$\mathrm{r} = \mathrm{cor}(\mathrm{C_i}, \mathrm{C_j}) = \frac{\mathrm{cov}(\mathrm{C_i}, \mathrm{C_j})}{\sqrt{\mathrm{var}(\mathrm{C_i})\mathrm{var}(\mathrm{C_j})}} = \frac{\mathrm{bd} - \mathrm{ac}}{\sqrt{(\mathrm{b}+\mathrm{c})(\mathrm{a}+\mathrm{d})(\mathrm{a}+\mathrm{b})(\mathrm{c}+\mathrm{d})}}$$

2.2.5 Leo Breiman's Generalization Error Bound

Margin is the probability of correct classification minus the maximum probability of incorrect classification of a case. For binary classification using random forests, margin can be calculated as the proportion of votes for the correct class minus proportion of votes for the wrong class. Strength (s) is the average of margins in the testing dataset.

Correlation (ρ) is the average correlation between the margins of any two different classification trees of the random forests (Breiman, 2001; Liu, et al., 2008)

$$\text{Margin}: \ s_i = P(\hat{Y} = Y) - P(\hat{Y} \neq Y)$$

$$\text{Strength}: \ s = \frac{1}{N_{te}} \sum_1^{N_{te}} s_i \quad (2.1)$$

$$\text{Correlation}: \ \rho = \frac{\frac{1}{N_{te}} \sum_1^{N_{te}} s_i^2 - \hat{s}^2}{\frac{2}{N_{tree}} \sum_1^{N_{tree}} \sqrt{P(\hat{Y} = Y) P(\hat{Y} \neq Y)}} \quad (2.2)$$

PE is the generalization error of random forests. Based on Leo Breiman's inference, PE has an upper bound conditional on s and ρ in equations (2.1) and (2.2). Mathematically, it can be written as below.

$$PE \leq \frac{\rho\,(1-s^2)}{s^2} \quad (2.3)$$

For simplicity, we can also use the average phi coefficient between any two sets of binary predictions within random forests to substitute ρ when we are comparing different error bounds of PE based on equation (2.3).

Throughout this dissertation, we also use "Breiman's error bound" or "Breiman's generalization error bound" to refer to Leo Breiman's generalization error bound for random forests.

2.2.6 Ensemble Classification

For a given number of observations in binary classification, an individual classifier can give one set of predictions for the possible outcomes. The most common way to combine predictions from individual classifiers is by majority rule. For an observation, if 5 out of

9 classifiers predict the binary outcome as "1" and 4 out of 9 classifiers predict the binary outcome as "0", the majority rule will infer that the ensemble classifier's prediction is "1". Also, this ensemble classifier can be used to predict that $\widehat{\mathrm{P}}(\mathrm{Y}=1) = 5/9$.

When we have completely independent individual classifiers, we can easily compute the accuracy of the ensemble classifier through knowledge of the accuracy of the individual classifiers. For example, when three independent individual classifiers are combined to make a binary classification, and each classifier comes with an error rate *e*, the cumulative probability that at least two classifiers make the correct decision is

$$\mathrm{P}_3 = 1 - \mathrm{C}_3^0(1-\mathrm{e})^0\mathrm{e}^3 - \mathrm{C}_3^1(1-\mathrm{e})^1\mathrm{e}^2$$

Figure 6 shows the ensemble classifier accuracy for 9, 99 and 999 independent classifiers. As long as individual classifiers have accuracy levels higher than 0.5 and they are independent of each other, we can improve the accuracy of the ensemble classifier by adding more individual classifiers. The equation (2.4) below is used to calculate cumulative accuracy for 999 independent classifiers as P_{999}. Given that individual independent classifiers have an accuracy of at least 0.6, $\mathrm{P}_{999} \geq (1-10^{-10})$.

$$\mathrm{P}_{999} = 1 - \mathrm{C}_{999}^0(1-\mathrm{e})^0\mathrm{e}^{999} - \mathrm{C}_{999}^1(1-\mathrm{e})^1\mathrm{e}^{998} - \cdots - \mathrm{C}_{999}^{499}(1-\mathrm{e})^{499}\mathrm{e}^{500} \quad (2.4)$$

When individual independent classifiers have accuracies below 0.5, one can interchange binary predictions (from 1 to 0, or from 0 to 1) to achieve an accuracy larger than 0.5. Hence, Figure 6 is symmetrical at (0.5, 0.5).

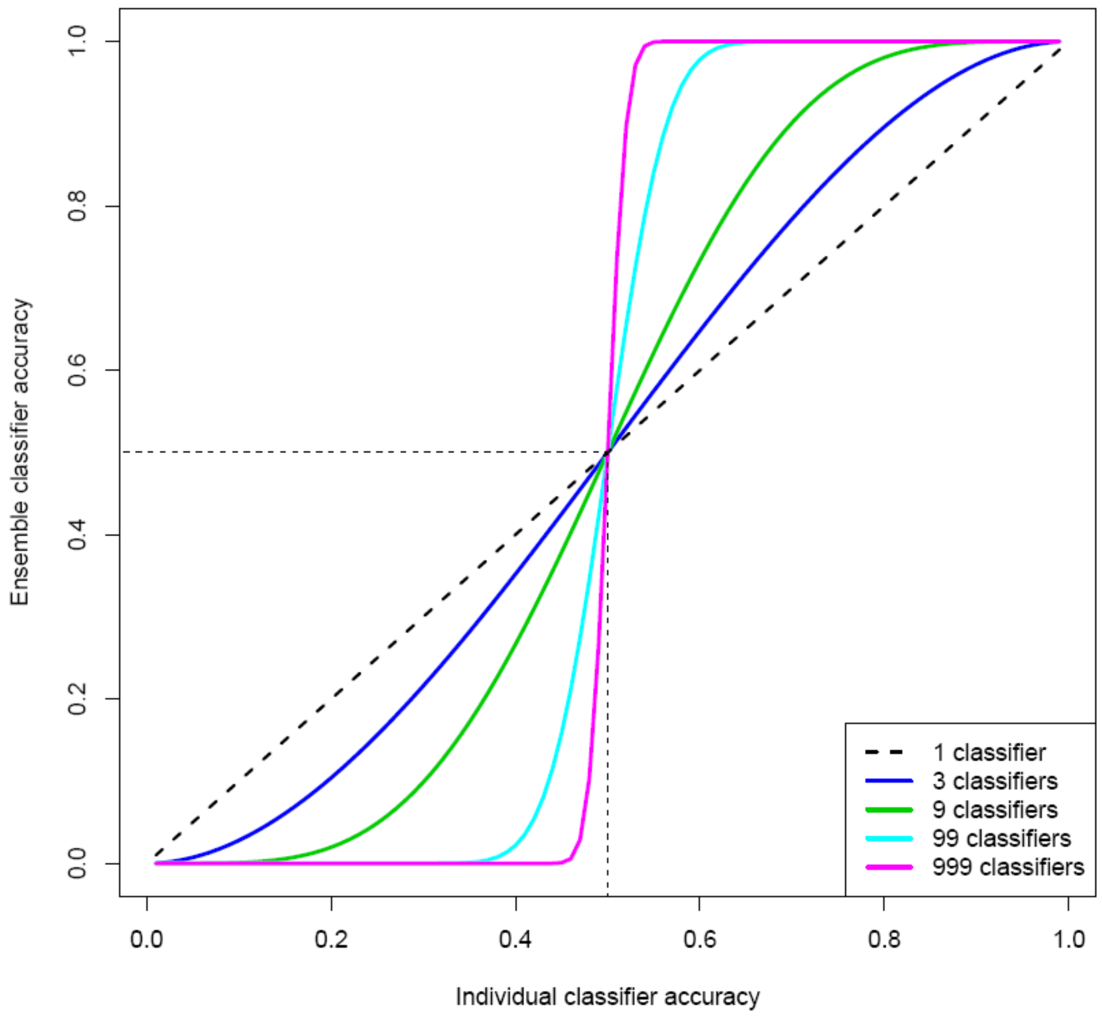


Figure 6: The Ensemble classifier accuracy and individual classifiers' accuracy by number of classifiers (theoretical results with r = 0)

However, in applications it would be unrealistic to expect wholly independent individual classifiers, especially if there were many of them. In a new simulation experiment using binary classification, we aim at building an ensemble classifier by combining 500 correlated classifiers through majority rule to make predictions for 2000 observations. The accuracy of the ensemble classifier is plotted against the average accuracy of individual classifiers across different levels of individual classifier correlation from 0.1 to 0.9 in Figure 7.

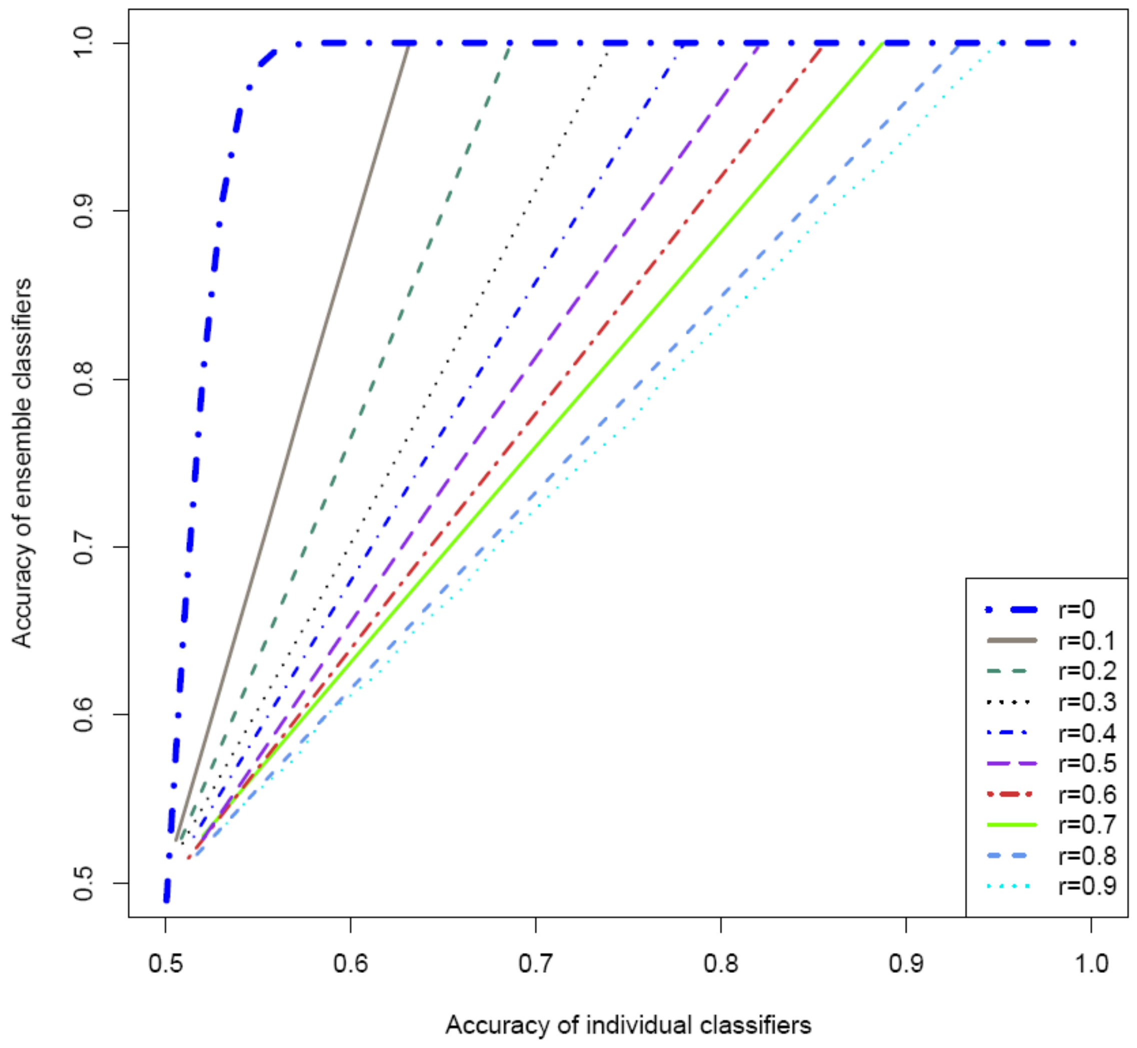


Figure 7: The Ensemble classifier accuracy and individual classifiers' accuracy by levels of correlations (simulated results with 500 classifiers)

As complete independence cannot be simulated empirically, results for r=0 in Figure 7 are theoretically derived similar to Figure 6. Results for r=0.1 to r=0.9 are based on empirical simulations using functions in the R (R Development Core Team, 2012) package `bindata` (Leisch, et al., 2011), and smoothed by locally-weighted polynomial regression.

As seen in Figure 7, the accuracies of ensemble classifiers decrease with increasing correlations. When the correlation coefficient is 0.1, we can still build an ensemble classifier with accuracy over 0.9 based on individual classifiers at an accuracy level around 0.6 or more. When the correlation coefficient reaches 0.9, the accuracy gain from combining 500 individual classifiers is marginal.

2.3 Roughened Random Forests - A (RRFA)

The successes of random forests are largely due to the subtle balance of accuracy and correlation among individual classification trees. We propose to further decrease the correlation within random forests by modifying the original dataset before building each classification tree. The resulting roughened random forests should have a decrease in both correlation among individual classification trees and the accuracy of individual classification trees. In our first proposed algorithm, RRFA, we modify the original dataset through imposing missing data followed by missing data imputation. RRFA algorithm is composed of four steps as below.

1. Impose missing values under the mechanism of missing completely at random on all covariates of both training and testing datasets.
2. Impute the missing data by median imputation for continuous variables and mode imputation for categorical variables.

3. Build one tree in random forests using the above imputed training dataset, and then use it to predict the binary outcomes in the imputed testing dataset.
4. Repeat 1 to 3 for N_{tree} times, in total N_{tree} different trees are built, and N_{tree} different sets of predictions are made for the binary outcomes in the imputed testing dataset.

During the N_{tree} repeats, different sets of predictions in RRFA are averaged and RRFA's performances are compared with performances of the original random forests (RF) which directly use the complete dataset. The major differences between RF and RRFA are illustrated in Figure 8.

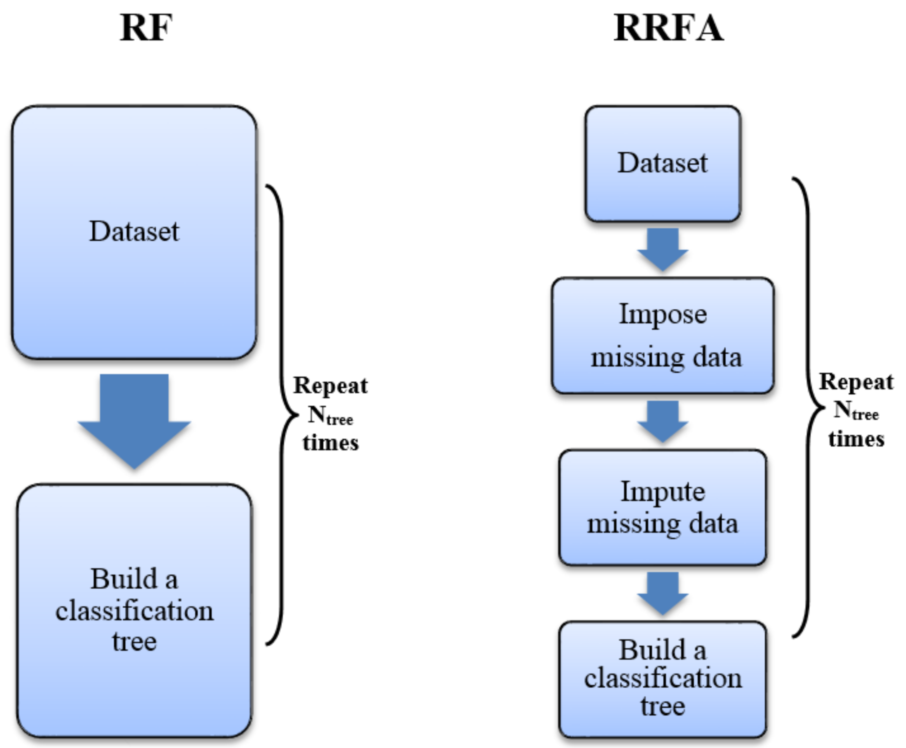


Figure 8: The differences between RF and RRFA

2.4 Datasets

2.4.1 Pima Indians Dataset

The Pima Indians dataset is the combination of `Pima.tr` and `Pima.te` datasets in R package `MASS` (Ripley, et al., 2014). `Pima.tr` and `Pima.te` were originally collected by the United States' National Institute of Diabetes and Digestive and Kidney Diseases. The combined dataset includes 532 Pima Indian women who are at least 21 years old. Among them, 177 women have diabetes, and 355 women do not have diabetes. In this dataset there are seven covariates, which are number of pregnancies, plasma glucose concentration in an oral glucose tolerance test, diastolic blood pressure, triceps skin fold thickness, body mass index, diabetes pedigree function, and age.

2.4.2 Blowdown Dataset

The blowdown dataset is from R package `alr3` (Weisberg, 2011). This dataset comes from the Boundary Waters canoe area wilderness in northern Minnesota, USA. A major storm hit this area on July 4, 1999. After the storm, 3666 trees were examined for survival. 1684 trees died and 1982 trees survived during the follow-up. There are three variables in this dataset, tree diameter, local severity of the storm, and tree species.

2.4.3 Simulated Datasets (Mease1 and Mease2)

We will simulate two datasets using two different simulation rules (Mease & Wyner, 2008). These two simulated datasets will be called Mease1 and Mease2. For the first simulated dataset Mease1, it is based on the equation below:

$$P(Y = 1|X) = q + (1 - 2q)\, I\left[\sum_{j=1}^{J} X^j > J/2\right]$$

X is distributed *iid* uniform on the d-dimensional unit cube $[0,1]^d$. We will set q (Bayes error) at 0.1, d (total number of available variables) at 20, J is set at 5. Also, 1000 observations will be simulated for use. There should be around 500 observations each for Y=1 and Y=0.

For the second simulated dataset Mease2, it is based on the equation below:

$$P\,(Y = 1|X) = 1/(1 + e^{k\left(\sum_{j=1}^{J} X^j - J/2\right)})$$

X is distributed *iid* uniform on the d-dimensional unit cube $[0,1]^d$. Here we also set d at 20, J at 5, and k is set at 8 so that Bayes error is also at around 0.1 here. Also, 1000 observations will be simulated for use. There should be around 500 observations each for Y=1 and Y=0.

2.5 Experiments

For N observations in a given dataset, we randomly draw N_{tr} observations as training dataset, and the rest of N_{te} ($N - N_{tr}$) observations are used as testing dataset. Besides using the original RF in the complete dataset, we apply the above new random forests algorithm RRFA using five different rates of missing data (MIS_{pct}) that are MCAR. The five different rates of missing data are initially set as: 10%, 20%, 30%, 40% and 50%, and they can also be adjusted in different scenarios. The same experiment will be also repeated with different values of N_{tr} and N_{te}. For the number of trees (N_{tree}), we will use the default value of N_{tree}=500.

To address the simulation error, we divide the existing dataset into training and testing datasets at different data size ratios (N_{tr}/N_{te}) and randomly sample training and testing datasets 50 times for each N_{tr}/N_{te}. We will compare the overall performance of RRFA and RF across different datasets at different N_{tr}/N_{te} and different MIS_{pct}. We will compare RRFA with the original RF by their relative performance using W/T/L tables. Also, we will calculate the average misclassification errors and average AUC values with each additional tree in the random forests.

2.6 Results

2.6.1 Pima Indians Dataset

2.6.1.1 N_{tr}=200 and N_{te}=332

The results for the first experiment, with N_{tr}=200 and N_{te}=332, are presented in Table 2. The format of Table 2 can be explained using the first row of results as an example. When RRFA with 10% of missing data is compared with the original RF in misclassification errors, RRFA has 33 wins, 4 ties and 13 losses, with a W/T/L value of "33/4/13". When RRFA with 10% of missing data is compared with the original RF in AUC, RRFA has 47 wins, 0 ties and 3 losses, with a W/T/L value of "47/0/3". As misclassification error reaches the best at the minimum and AUC reaches the best at the maximum, a smaller value means "win" for misclassification error and "loss" for AUC. On the contrary, a larger value means "loss" for misclassification error and "win" for

AUC. Also, we make the cell value **bold** if the number of "win" in that cell is more than the number of "loss", and we make the cell value **<u>bold</u>** (bold and underlined) if the proportion of "win" is significantly higher than 0.5. The cell values in Table 2 are all bold, therefore, RRFA beats the original RF in both misclassification error and AUC when 10%, 20%, 30%, 40% and 50% of missing data are imposed. Among all of them, RRFA with 20% of missing data has the most wins (**37/6/7**) in misclassification error comparison and RRFA with 30% of missing data has the most wins (**49/0/1**) in AUC comparison.

Table 2: The performance comparison of RRFA and RF in the Pima Indians dataset with a training/testing data size ratio of 200/332 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	**33/4/13**	**47/0/3**
20%	**37/6/7**	**48/0/2**
30%	**31/8/11**	**49/0/1**
40%	**28/4/18**	**46/0/4**
50%	**25/4/21**	**42/0/8**

Results are also presented in Figure 9 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10%, 20%, 30% and 40% of missing data start to show improvement over the original RF in average misclassification error when N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 50% of missing data has similar performance as the original RF. RRFA with 20% and 30% of missing data show the best performances in average misclassification error. For AUC values, RRFA with 10%, 20%, 30%, 40% and 50% of missing data start to show improvements over the original RF before N_{tree} reaches 100 and the improvements continue untile N_{tree} reaches 500. RRFA with 20%, 30% and 40% of missing data show the best performances in average AUC value.

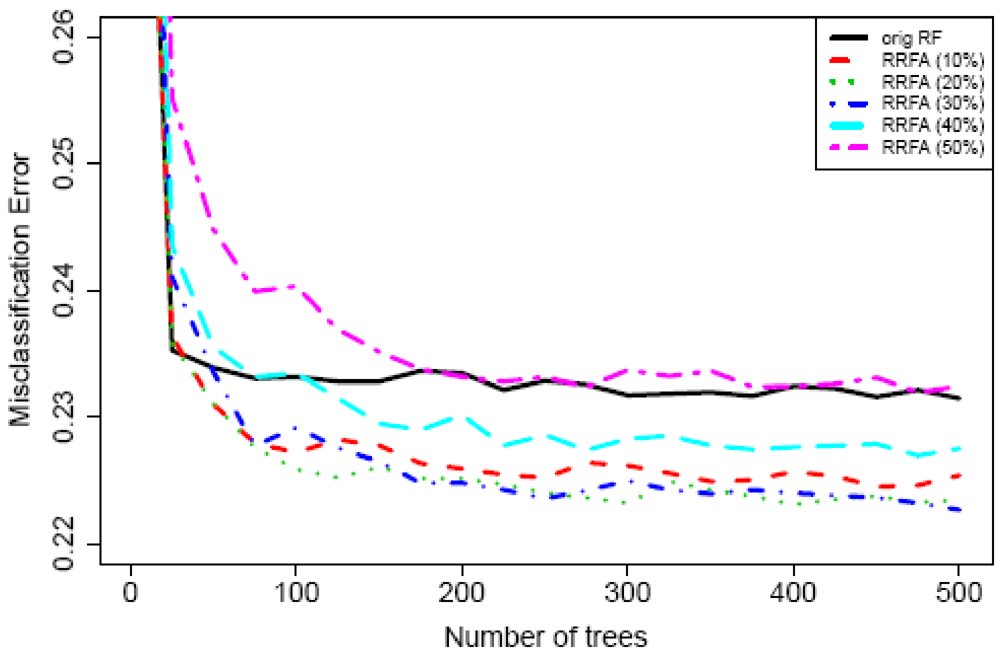
2.6.1.2 N_{tr} =266 and N_{te}=266

For N_{tr} =266 and N_{te}=266, RRFA also beats the original RF in both misclassification error and AUC when 10%, 20%, 30%, 40% and 50% of missing data are imposed as shown in Table 3. Among all of them, RRFA with 30% of missing data has the most wins in misclassification error comparison (**32/8/10**) and RRFA with 10% of missing data has the most wins in AUC comparison (**49/0/1**).

Table 3: The performance comparison of RRFA and RF in the Pima Indians dataset with a training/testing data size ratio of 266/266 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	**32/5/13**	**49/0/1**
20%	**28/7/15**	**47/0/3**
30%	**32/8/10**	**47/0/3**
40%	**31/3/16**	**45/0/5**
50%	**28/2/20**	**43/0/7**

Results are also presented in Figure 10 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10%, 20%, 30% and 40% of missing data start to show improvement over the original RF in average misclassification error when N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 50% of missing data starts to outperform the original RF when N_{tree} reaches 400. RRFA with 30% of missing data shows the best performances in average misclassification error. While for average AUC values, RRFA with 10%, 20%, 30%, 40% and 50% of missing data start to show improvements over the original RF before N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 30% and 40% of missing data show the best performances in average AUC value.



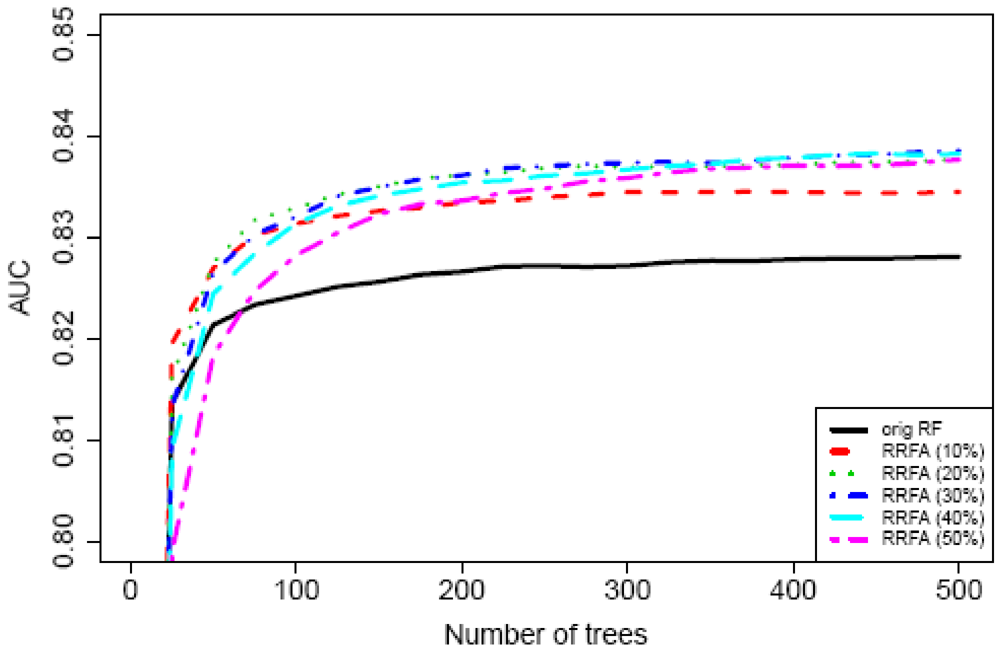
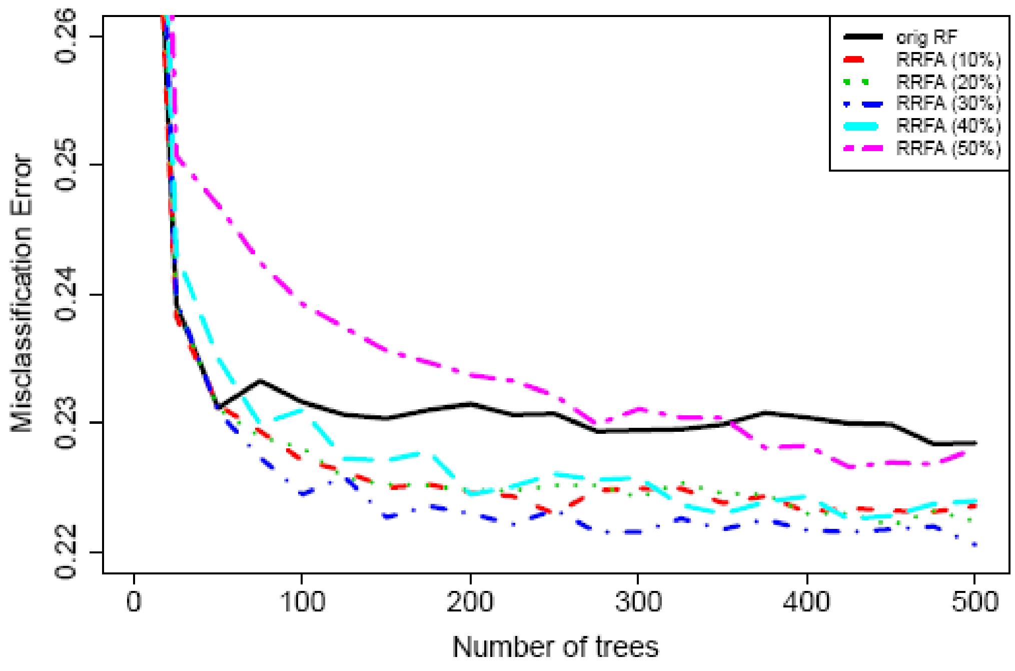


Figure 9: The average performance comparison of RRFA and RF in the Pima Indians dataset with a training/testing data size ratio of 200/332 over 50 trials



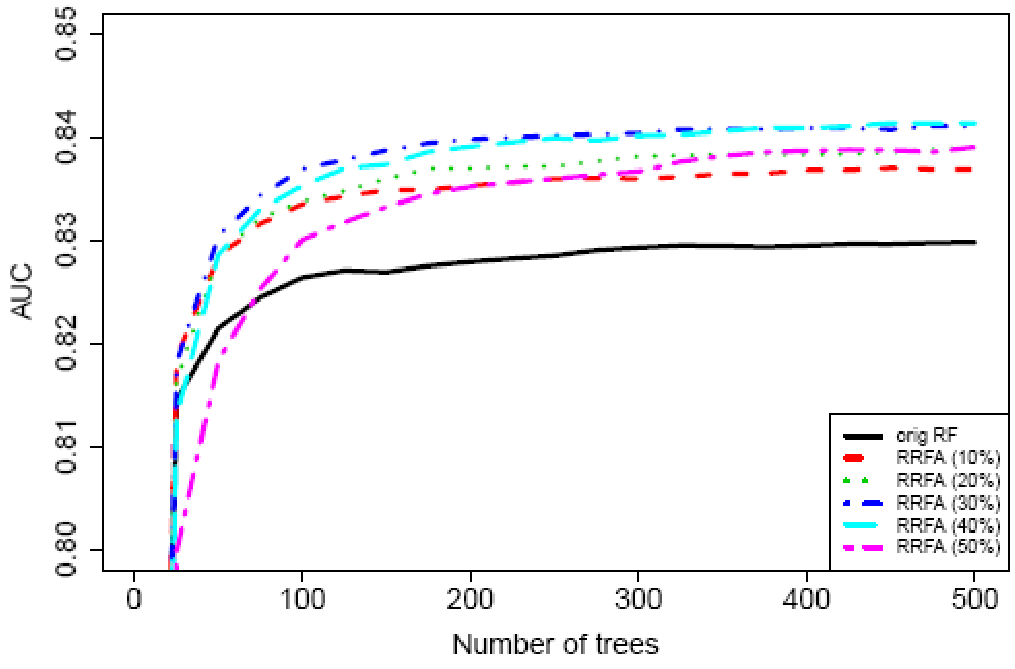


Figure 10: The average performance comparison of RRFA and RF in the Pima Indians dataset with a training/testing data size ratio of 266/266 over 50 trials

2.6.2 Blowdown Dataset

2.6.2.1 N_{tr} =200 and N_{te}=3466

For N_{tr} =200 and N_{te}=3466, the results are first shown in Table 4. RRFA beats the original RF in misclassification error when 10%, 20% and 30% of missing data are imposed, and RRFA beats the original RF in AUC when 10%, 20%, 30%, 40% and 50% of missing data are imposed. Among all of them, RRFA with 10% of missing data has the most wins in both misclassification error comparison (**<u>38/1/11</u>**) and AUC comparison (**<u>50/0/0</u>**).

Table 4: The performance comparison of RRFA and RF in the blowdown dataset with a training/testing data size ratio of 200/3466 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	**<u>38/1/11</u>**	**<u>50/0/0</u>**
20%	**30/1/19**	**<u>47/0/3</u>**
30%	**26/0/24**	**<u>45/0/5</u>**
40%	13/1/36	**<u>39/0/11</u>**
50%	9/0/41	**27/0/23**

Results are also presented in Figure 11 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10%, and 20% of missing data start to show improvement over the original RF in misclassification error when N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 30% of missing data has similar performance as the original RF. RRFA with 40% and 50% of missing data show worse performances in average misclassification error than the original RF. As for AUC values, RRFA with

10%, 20%, 30% and 40% of missing data start to show improvements over the original RF before N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 20% and 30% of missing data show the best performances in average AUC value.

2.6.2.2 N_{tr} =666 and N_{te}=3000

Due to poor performance associated with 40% and 50% of missing data for N_{tr} =200 and N_{te}=3466 in the previous experiment, we will use five lower rates (5%, 10%, 15%, 20% and 30%) of missing data instead for N_{tr} =666 and N_{te}=3000. The results are first shown in Table 5. RRFA beats the original RF in misclassification error when 5%, 10%, 15% and 20% of missing data are imposed, and RRFA beats the original RF in AUC when 5%, 10%, 15%, 20% and 30% of missing data are imposed. Among all of them, RRFA with 10% of missing data has the most wins in both misclassification error comparison (**35/1/14**) and AUC comparison (**49/0/1**).

Table 5: The performance comparison of RRFA and RF in the blowdown dataset with a training/testing data size ratio of 666/3000 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
5%	**31/3/16**	**48/0/2**
10%	**35/1/14**	**49/0/1**
15%	**29/1/20**	**46/0/4**
20%	**30/0/20**	**45/0/5**
30%	21/0/29	**40/0/10**

Results are also presented in Figure 12 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 5%, 10%, 15% and 20% of missing data start to show improvement over the original RF in misclassification error when N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 30% of missing data has worse performance than the original RF. RRFA with 10% of missing data show the best performances in average misclassification error. While for AUC values, RRFA with 5%, 10%, 15%, 20% and 30% of missing data start to show improvements over the original RF before N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 15% and 20% of missing data show the best performances in average AUC value.

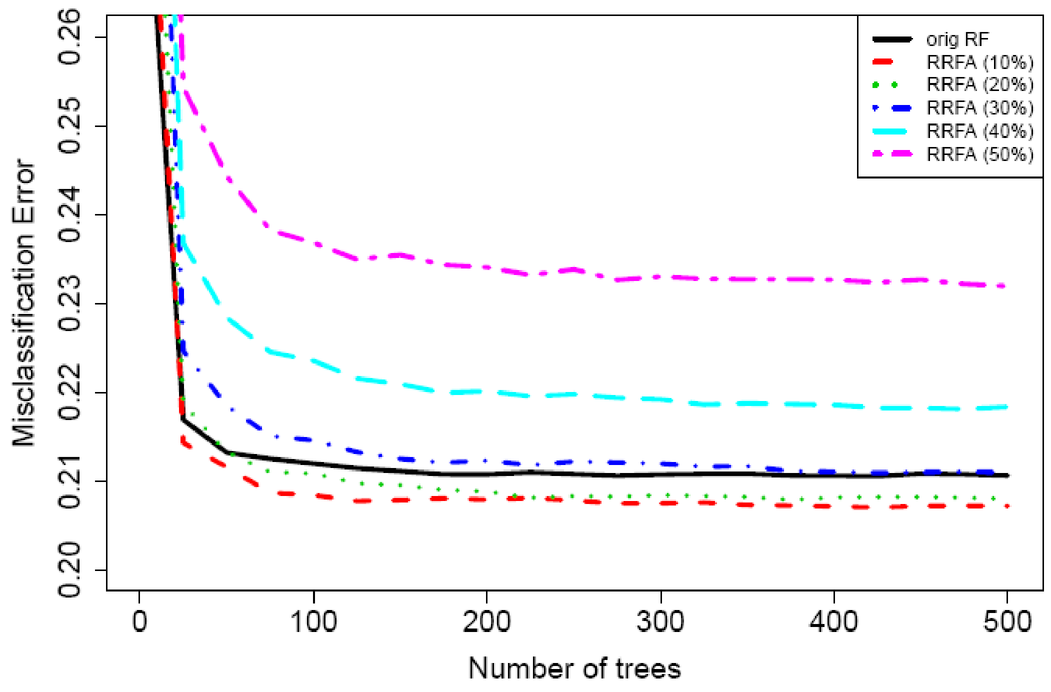
2.6.2.3 N_{tr} =1833 and N_{te}=1833

For N_{tr} =1833 and N_{te}=1833, the results are first shown in Table 6. We also use five lower rates (5%, 10%, 15%, 20% and 30%) of missing data instead. RRFA beats the original RF in misclassification error when 5%, 10%, 15%, 20% and 30% of missing data are imposed, and RRFA beats the original RF in AUC when 5%, 10%, 15% and 20% of missing data are imposed. Among all of them, RRFA with 10% of missing data has the most wins (**42/0/8**) in misclassification error comparison and RRFA with 5% of missing data has the most wins (**47/0/3**) in AUC comparison.

Table 6: The performance comparison of RRFA and RF in the blowdown dataset with a training/testing data size ratio of 1833/1833 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
5%	**36/5/9**	**47/0/3**
10%	**42/0/8**	**45/0/5**
15%	**38/2/10**	**42/0/8**
20%	**30/1/19**	**36/0/14**
30%	**27/0/23**	16/0/34

Results are also presented in Figure 13 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 5%, 10%, 15% and 20% of missing data start to show improvement over the original RF in misclassification error when N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 30% of missing data has better performance than the original RF when N_{tree} gets over 200 until N_{tree} reaches 500. RRFA with 15% of missing data shows the best performances in average misclassification error. While for AUC values, RRFA with 5%, 10%, 15% and 20% of missing data start to show improvements over the original RF before N_{tree} reaches 100 and the improvements continue until N_{tree} reaches 500. RRFA with 10% of missing data shows the best performances in average AUC value.



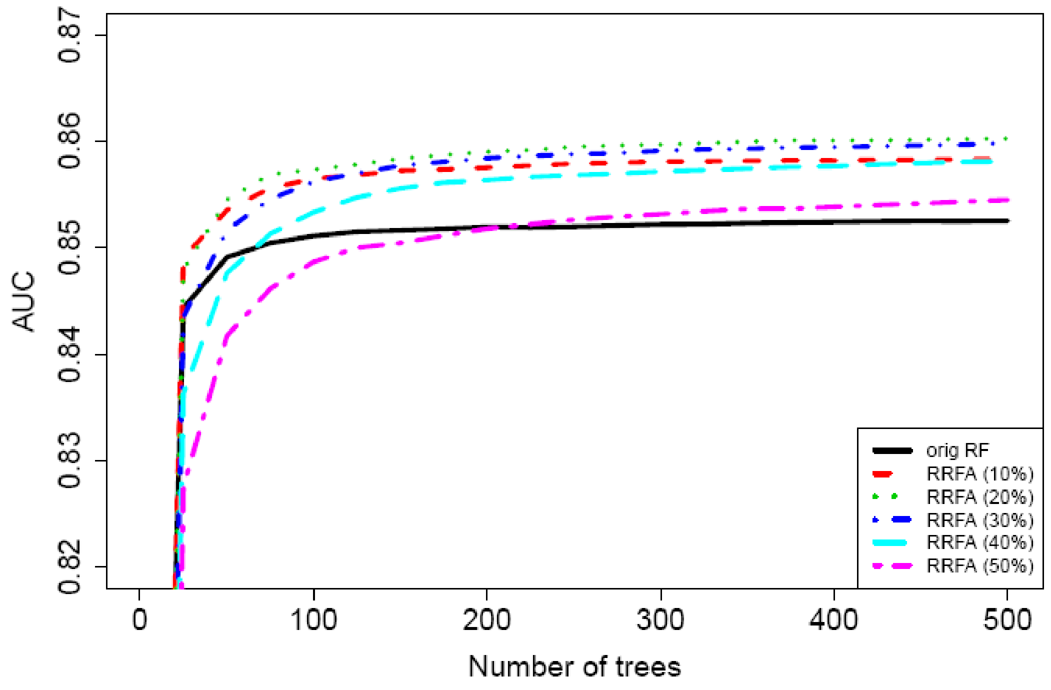
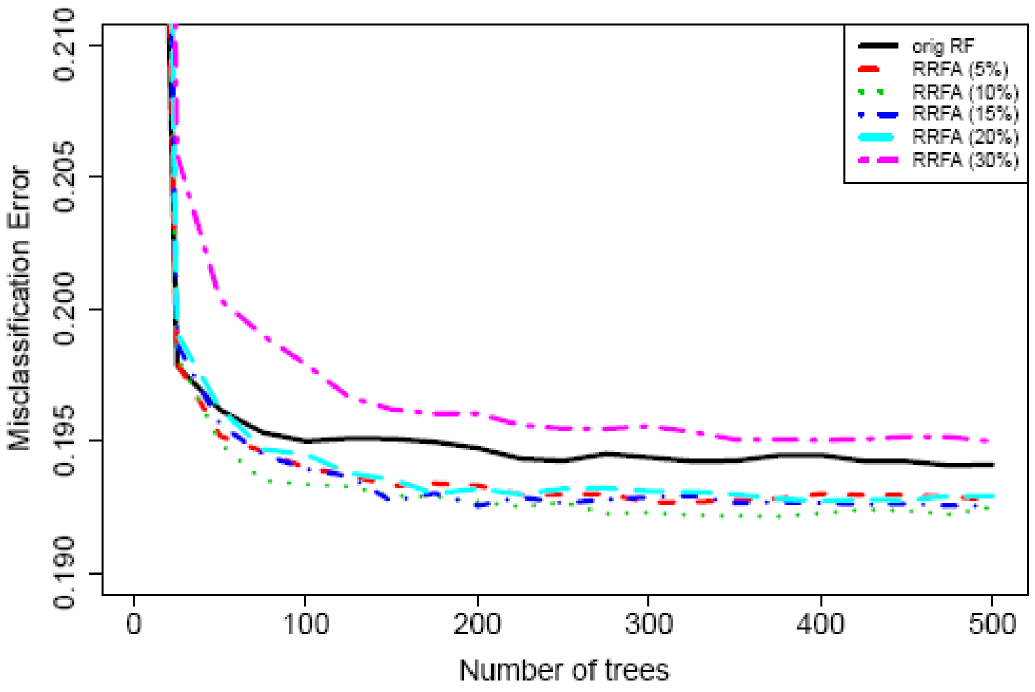


Figure 11: The average performance comparison of RRFA and RF in the blowdown dataset with a training/testing data size ratio of 200/3466 over 50 trials



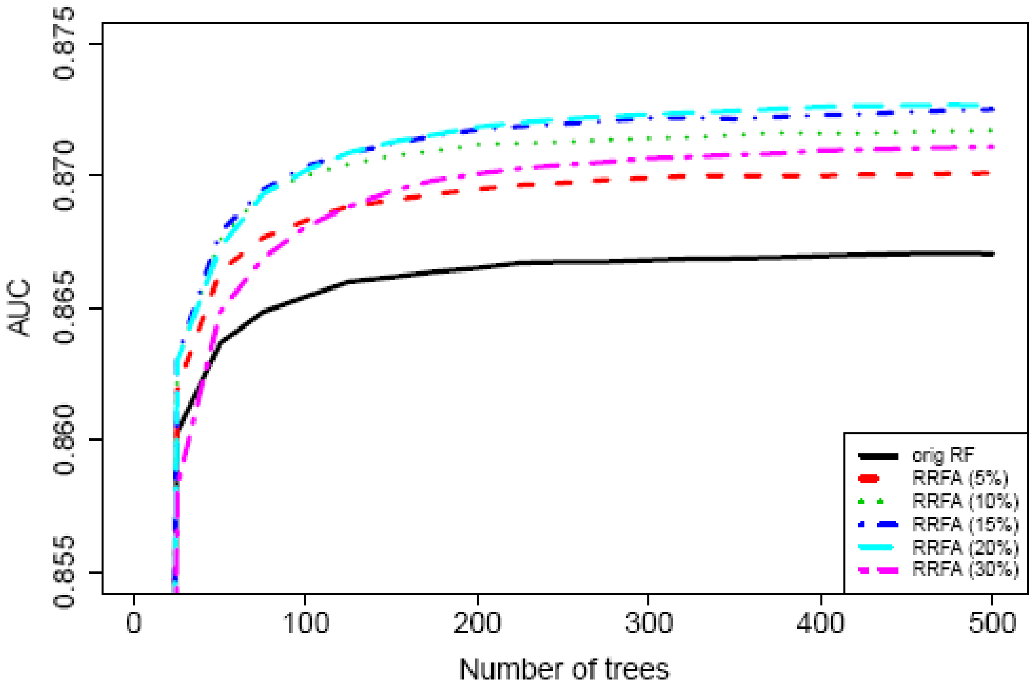
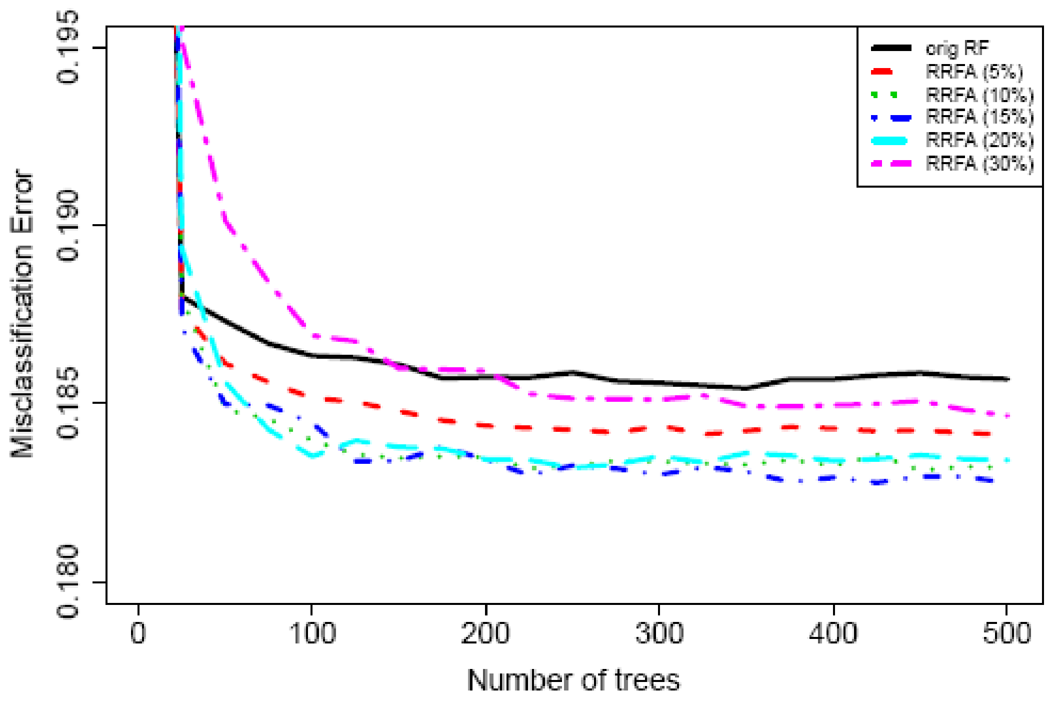


Figure 12: The average performance comparison of RRFA and RF in the blowdown dataset with a training/testing data size ratio of 666/3000 over 50 trials



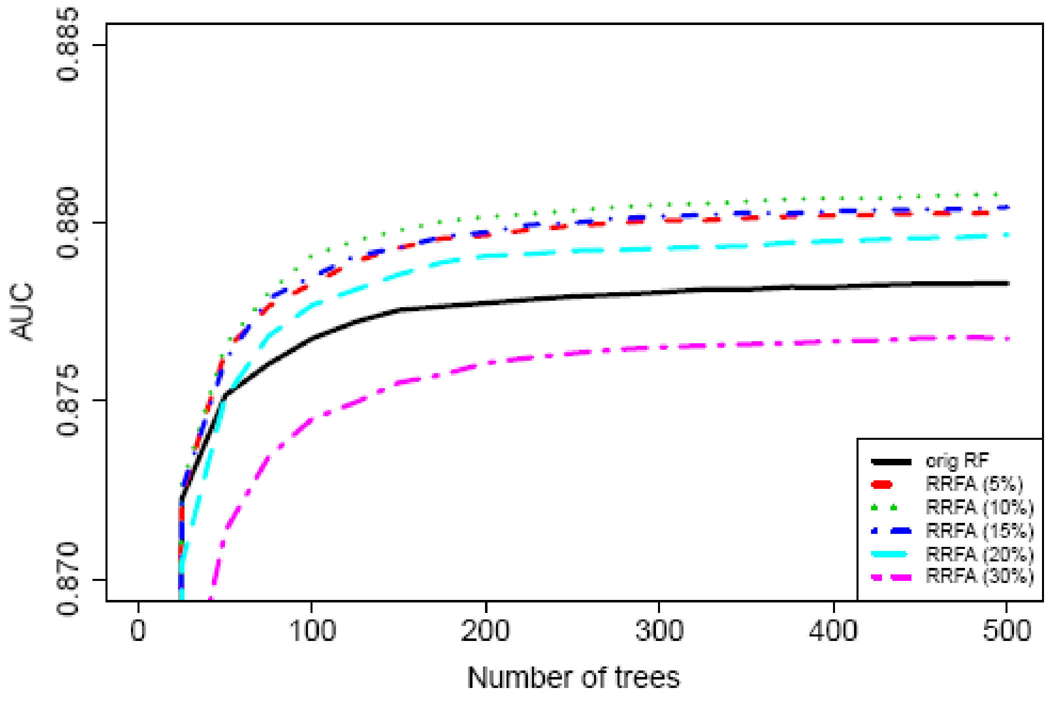


Figure 13: The average performance comparison of RRFA and RF in the blowdown dataset with a training/testing data size ratio of 1833/1833 over 50 trials

2.6.3 Mease1 Dataset

2.6.3.1 N_{tr} =200 and N_{te}=800

For N_{tr} =200 and N_{te}=800, the results are first shown in Table 7. RRFA beats the original RF in misclassification error when 10%, and 20% of missing data are imposed, and RRFA beats the original RF in AUC when 10%, 20%, and 30% of missing data are imposed. Among all of them, RRFA with 10% of missing data has the most wins (**29/3/18**) in misclassification error comparison and RRFA with 20% of missing data has the most wins (**34/0/16**) in AUC comparison.

Table 7: The performance comparison of RRFA and RF in the Mease1 dataset with a training/testing data size ratio of 200/800 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	**29/3/18**	**33/0/17**
20%	**29/1/20**	**34/0/16**
30%	23/4/23	**28/0/22**
40%	15/2/33	25/0/25
50%	12/1/37	18/0/32

Results are also presented in Figure 14 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10% and 20% of missing data start to show improvement over the original RF in average misclassification error when N_{tree} gets over 400. While for average AUC values, RRFA with 10%, 20% and 30% of missing data start to show improvements over the original RF when N_{tree} gets over 300. RRFA with 10% of missing data shows the best performance in average misclassification error. RRFA with 20% of missing data shows the best performances in average AUC value.

2.6.3.2 N_{tr} =400 and N_{te}=600

For N_{tr} =400 and N_{te}=600, the results are first shown in Table 8. RRFA beats the original RF in misclassification error when 10% and 20% of missing data are imposed, and RRFA beats the original RF in AUC when 10%, 20%, 30% and 40% of missing data are imposed. Among all of them, RRFA with 10% of missing data has the most wins in both misclassification error comparison (**30/4/16**) and AUC comparison (**<u>40/0/10</u>**).

Table 8: The performance comparison of RRFA and RF in the Mease1 dataset with a training/testing data size ratio of 400/600 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	**30/4/16**	**<u>40/0/10</u>**
20%	**25/4/21**	**<u>37/0/13</u>**
30%	19/1/30	**30/0/20**
40%	14/2/34	**26/0/24**
50%	12/2/36	17/0/33

Results are also presented in Figure 15 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10% of missing data start to show improvement over the original RF in average misclassification error when N_{tree} gets over 300. While for average AUC value, RRFA with 10%, 20% and 30% of missing data start to show improvements over the original RF when N_{tree} gets over 300. RRFA with 10% and 20% of missing data show the best performances in average AUC value.

2.6.4 Mease2 Dataset

2.6.4.1 N_{tr} =200 and N_{te}=800

For N_{tr} =200 and N_{te}=800, the results are first shown in Table 9. RRFA beats the original RF in misclassification error when 10% and 20% of missing data are imposed, and RRFA beats the original RF in AUC when 10%, 20%, 30% and 40% of missing data are imposed. Among all of them, RRFA with 10% of missing data has the most wins (**38/0/12**) in misclassification error comparison and RRFA with 20% of missing data has the most wins (**43/0/7**) in AUC comparison.

Table 9: The performance comparison of RRFA and RF in the Mease2 dataset with a training/testing data size ratio of 200/800 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	**38/0/12**	**41/0/9**
20%	**30/1/19**	**43/0/7**
30%	22/2/26	**32/0/18**
40%	20/4/26	**33/0/17**
50%	16/2/32	25/0/25

Results are also presented in Figure 16 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10% and 20% of missing data start to show improvement over the original RF in average misclassification error when N_{tree} gets over 350. While for average AUC values, RRFA with 10%, 20%, 30% and 40% of missing data start to show improvements over the original RF when N_{tree} gets over 300. RRFA with 10% of missing data shows the best performance in average misclassification error. RRFA with 10% and 20% of missing data show the best performances in average AUC value.

2.6.4.2 N_{tr} =400 and N_{te}=600

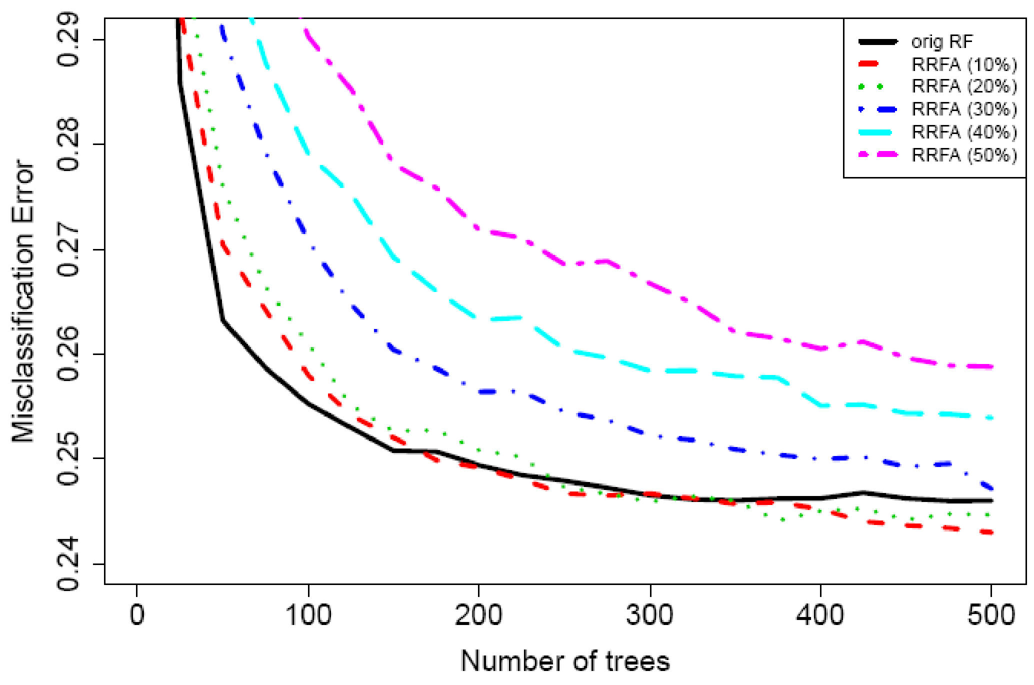
For N_{tr} =400 and N_{te}=600, the results are first shown in Table 10. RRFA beats the original RF in misclassification error when 20% and 30% of missing data are imposed, and RRFA beats the original RF in AUC when 10%, 20%, and 30% of missing data are imposed. Among all of them, RRFA with 20% of missing data has the most wins (**30/5/15**) in misclassification error comparison and RRFA with 10% as well as 30% of missing data both have the most wins (<u>**35/0/15**</u>) in AUC comparison.

Table 10: The performance comparison of RRFA and RF in the Mease2 dataset with a training/testing data size ratio of 400/600 over 50 trials in a W/T/L/ table

MIS_{pct}	Misclassification Error	AUC
10%	24/2/24	<u>**35/0/15**</u>
20%	**30/5/15**	<u>**33/0/17**</u>
30%	**26/7/17**	<u>**35/0/15**</u>
40%	18/2/30	22/0/28
50%	15/5/30	12/0/38

Results are also presented in Figure 17 showing how the average misclassification errors and average AUC values evolve when the number of trees increases from 1 to 500. RRFA with 10%, 20% and 30% of missing data start to show improvement over the original RF in average misclassification error when N_{tree} reaches 400. While for average AUC values, RRFA with 10%, 20% and 30% of missing data start to show similar improvements over the original RF when N_{tree} reaches 350. RRFA with 20% and 30% of missing data show the best performance in average misclassification error.

For all of the above experiments in four different datasets, RRFA achieves better binary classification performance than RF when the rates of imposed missing data are 10% or 20%. Using 200 trees can be enough to show the superior performance of RRFA in two real-life datasets. Using 400 trees can be enough to show the superior performance of RRFA in two simulated datasets. And using 500 trees always lead to even better improvements. As random forests generally do not overfit the data (Breiman & Cutler, 2004), more trees should be used when computational resources are not limited.



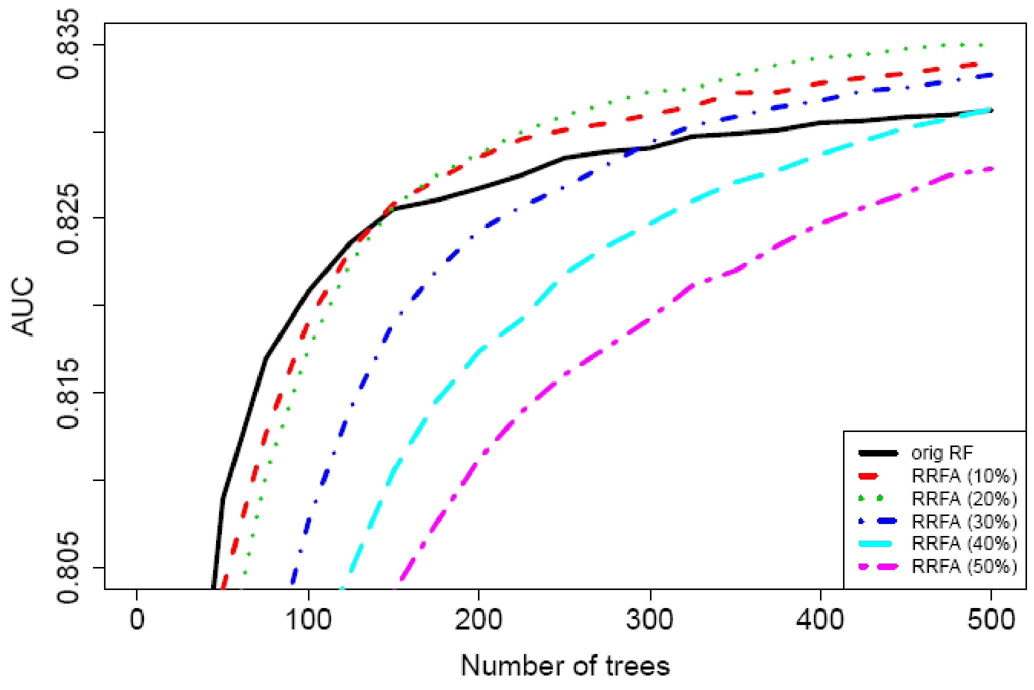
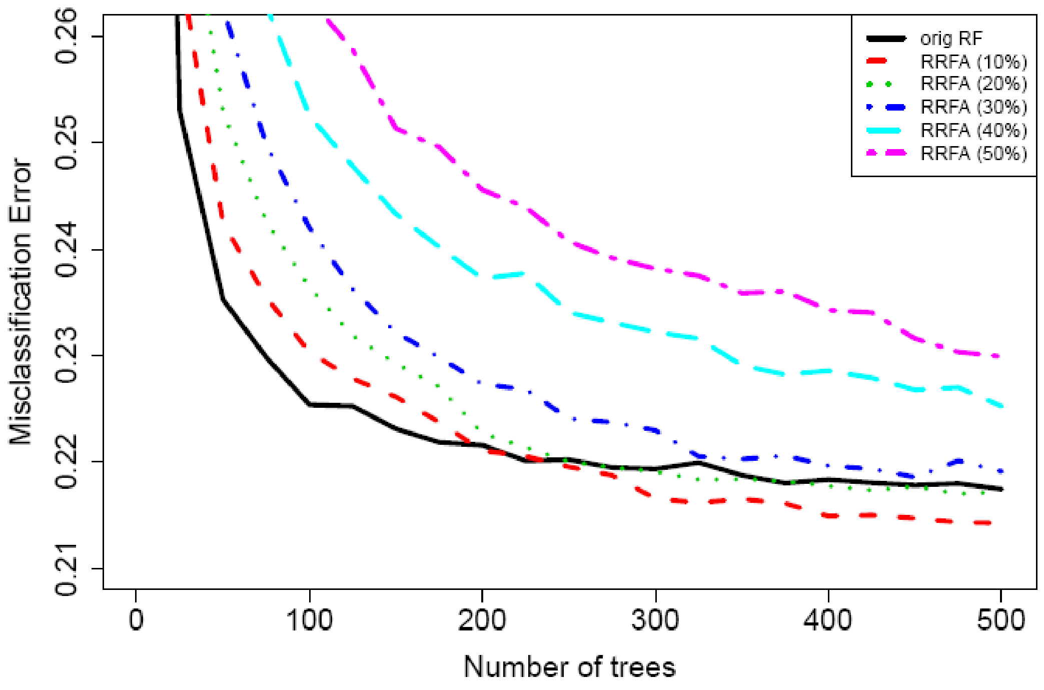


Figure 14: The average performance comparison of RRFA and RF in the Mease1 dataset with a training/testing data size ratio of 200/800 over 50 trials



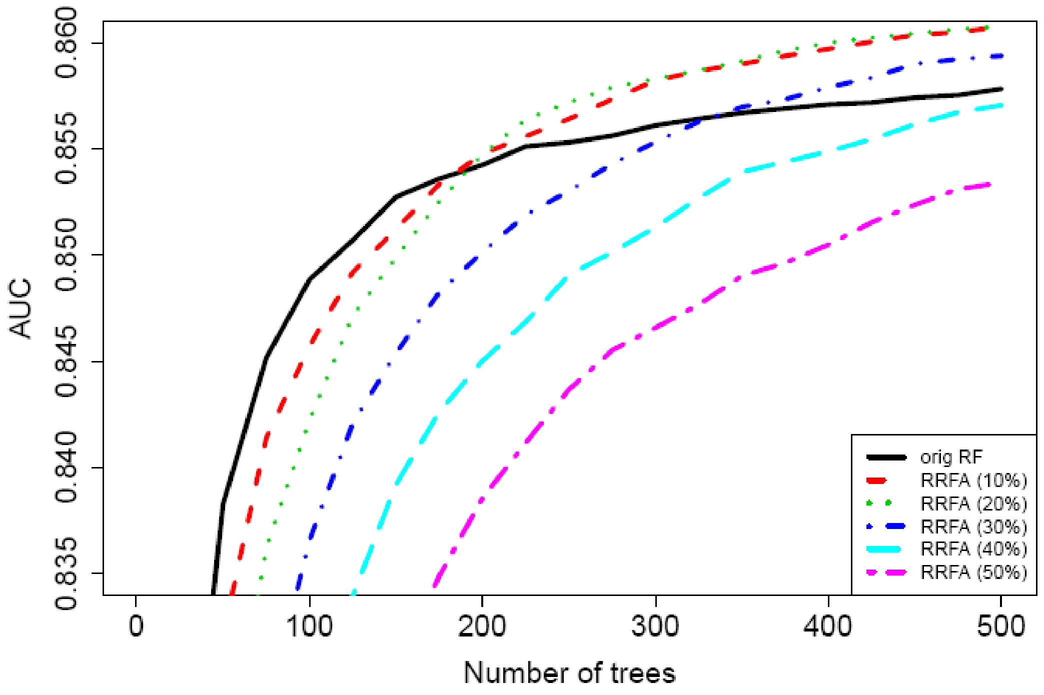
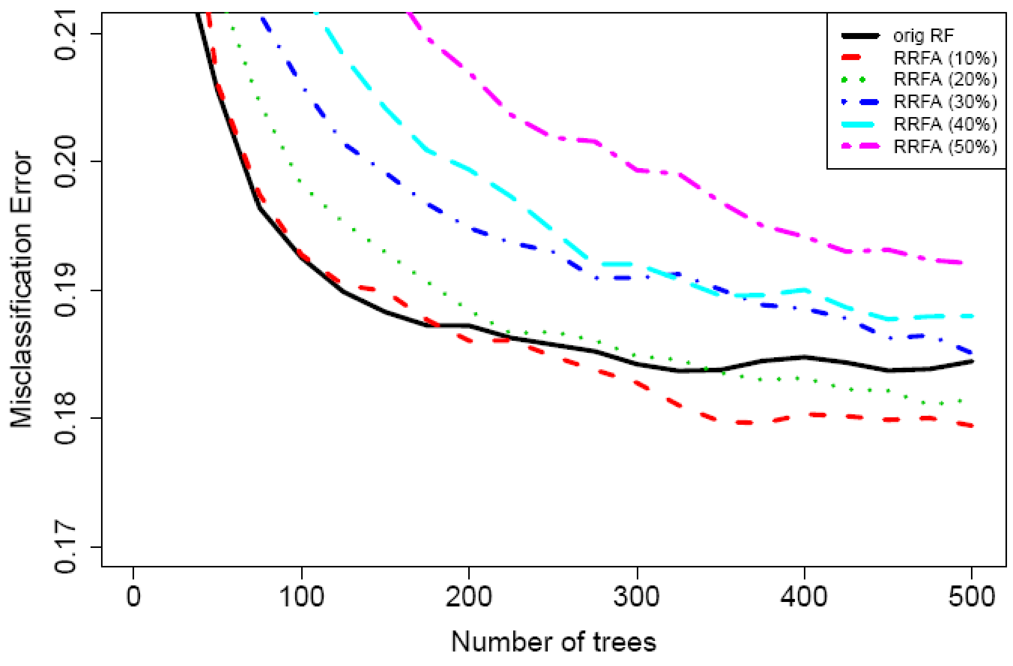


Figure 15: The average performance comparison of RRFA and RF in the Mease1 dataset with a training/testing data size ratio of 400/600 over 50 trials



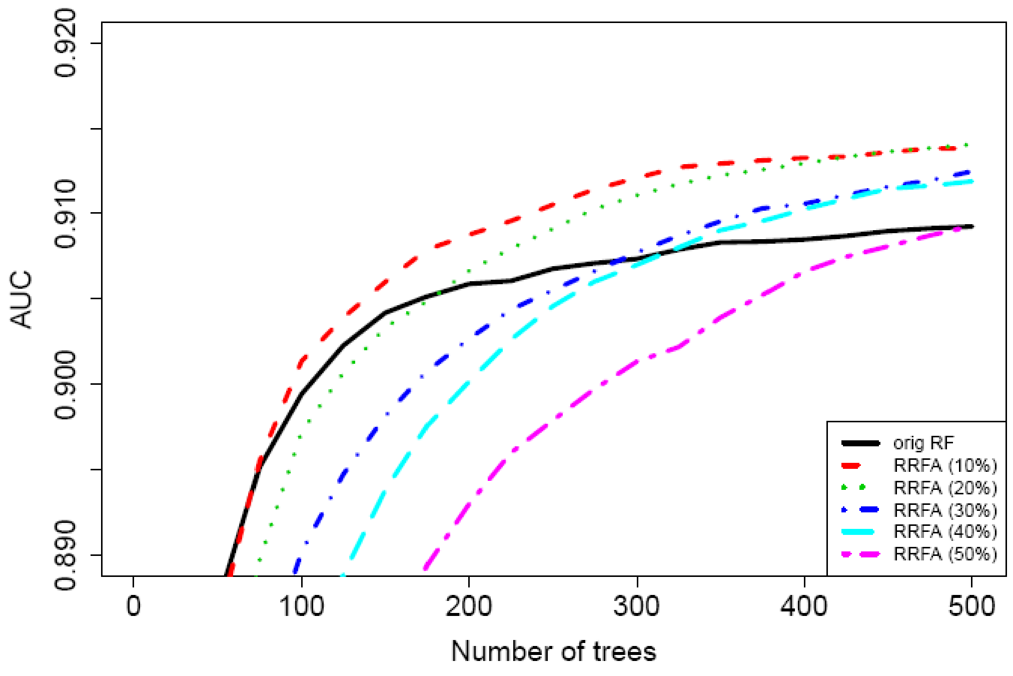
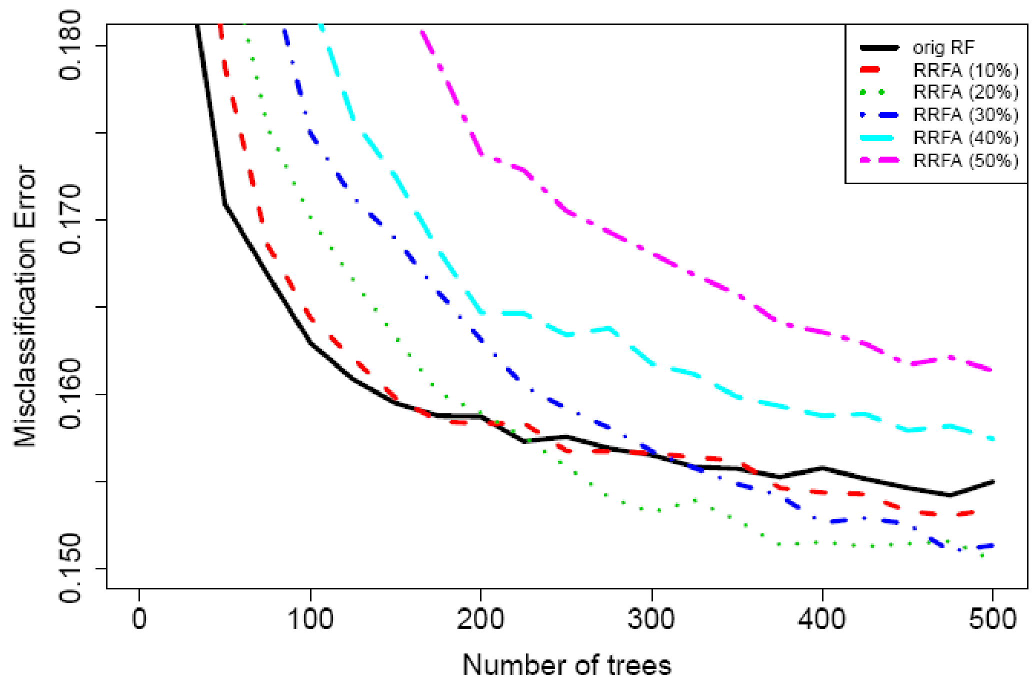


Figure 16: The average performance comparison of RRFA and RF in the Mease2 dataset with a training/testing data size ratio of 200/800 over 50 trials



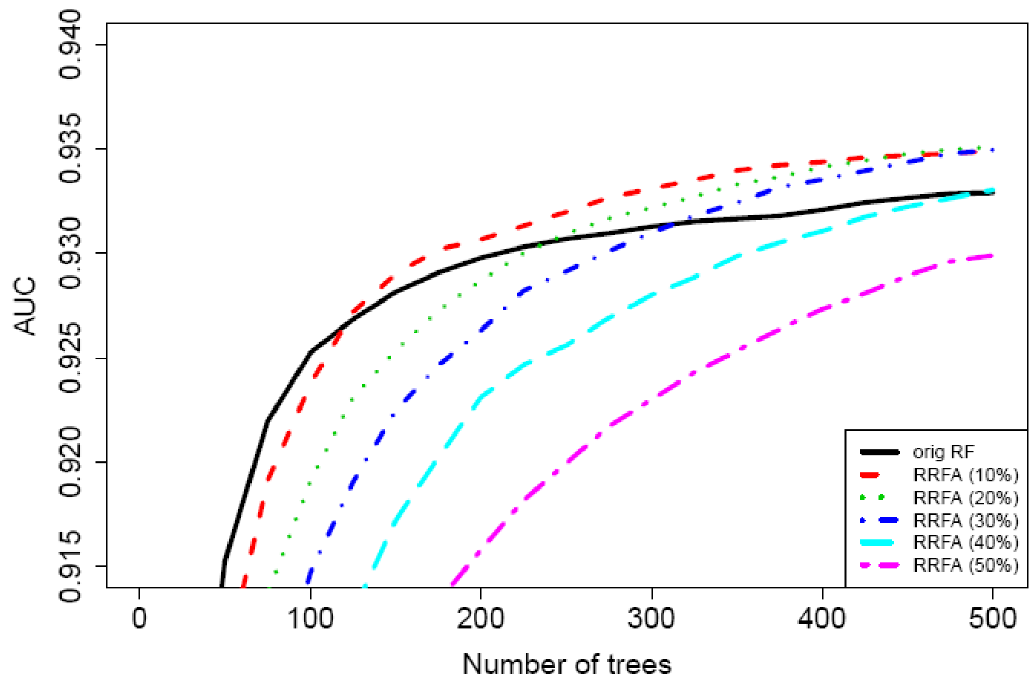


Figure 17: The average performance comparison of RRFA and RF in the Mease2 dataset with a training/testing data size ratio of 400/600 over 50 trials

2.7 Discussion

Our primary goal in this chapter is to compare the binary classification performance of RRFA with RF. For the above two real-life datasets and two simulated datasets, RRFA can have superior performance than the original RF in both misclassification error and AUC. RRFA works the best when the rates of imposed missing data under MCAR are 10% and 20%. In addition, the improvements in AUC are much more significant than misclassification error. Furthermore, the performance improvements are more significant in real-life datasets than simulated datasets. Fewer trees are also needed to show performance improvements in real-life datasets than simulated datasets. This is likely due to the higher number of covariates in our two simulated datasets.

There are multiple techniques which could potentially further improve the classification performance of RRFA. Missing data are imposed in both training and testing datasets in RRFA. One potential technique would be to limit the missing data introduction to the training dataset. Median/mode imputation is the only imputation method used in RRFA. Another potential technique would be to use different single imputation and multiple imputation methods.

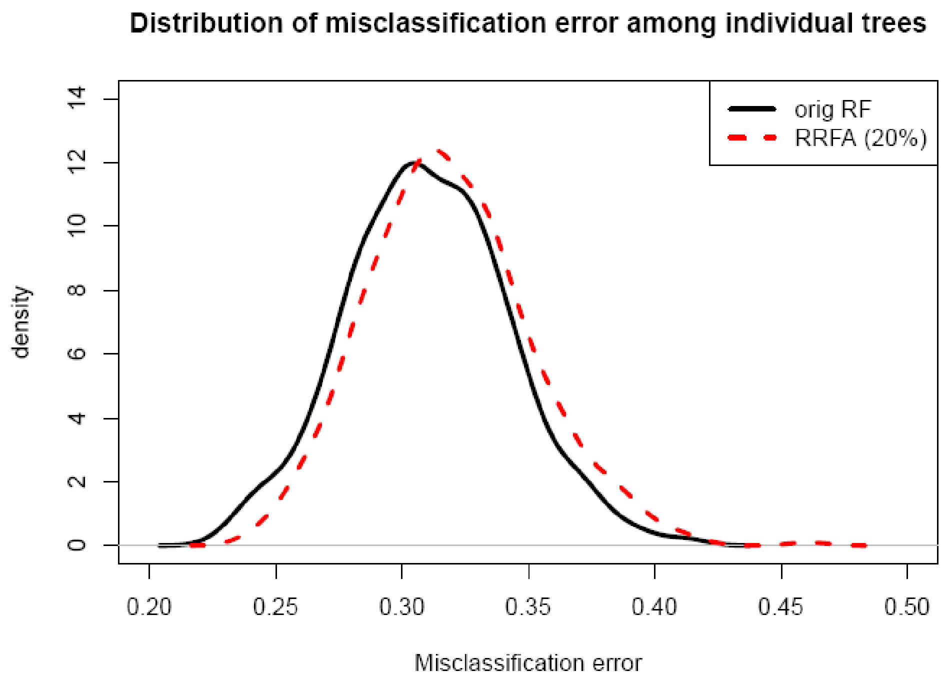
2.7.1 Misclassification Error and Pearson Correlation Coefficient

To learn the underlying reason for the improvement in RRFA, we can look at the individual tree performances with an example. We will use `Pima.tr` (N_{tr}=200) and `Pima.te` (N_{te}=332) from R package `MASS` as the training dataset and the testing dataset. Instead of using the default value of N_{tree} = 500, we further increase it to N_{tree} = 2000 to have more stable results. We look at the individual trees' misclassification errors and pairwise correlations in both the original random forests and RRFA with 20% of missing data.

As seen in the probability density distribution plots in Figure 18, even though the individual trees' misclassification errors increase, the pairwise correlations between trees decrease substantially. When the same example is repeated using different datasets and with different rates of missing data, we find the only common characteristic among all successful improvements is the increase of misclassification error and the decrease of pairwise correlation, while the shape and span of the probability density plots vary largely.

2.7.2 AUC and Leo Breiman's Generalization Error Bound

Leo Breiman proposed the generalization error for random forests with an upper bound as given in equation (2.3). Here we use the same example related to the Pima Indian dataset. When rates of imposed missing data (MIS_{pct}) range from 0 to 50% by 10%, we generate 100 random forests with the same size of N_{tree} =2000 for each MIS_{pct}. RRFA with 0% of missing data is just the original RF. For computational simplicity, here we will use average Pearson correlation coefficient r (the same as phi coefficient) to replace ρ while calculating Breiman's error bounds. As shown in Figure 19, AUC is inversely related to Breiman's error bound. As the percentage of missing data increases from 0 to 50%, the error bound becomes smaller, and the AUC values gets bigger.



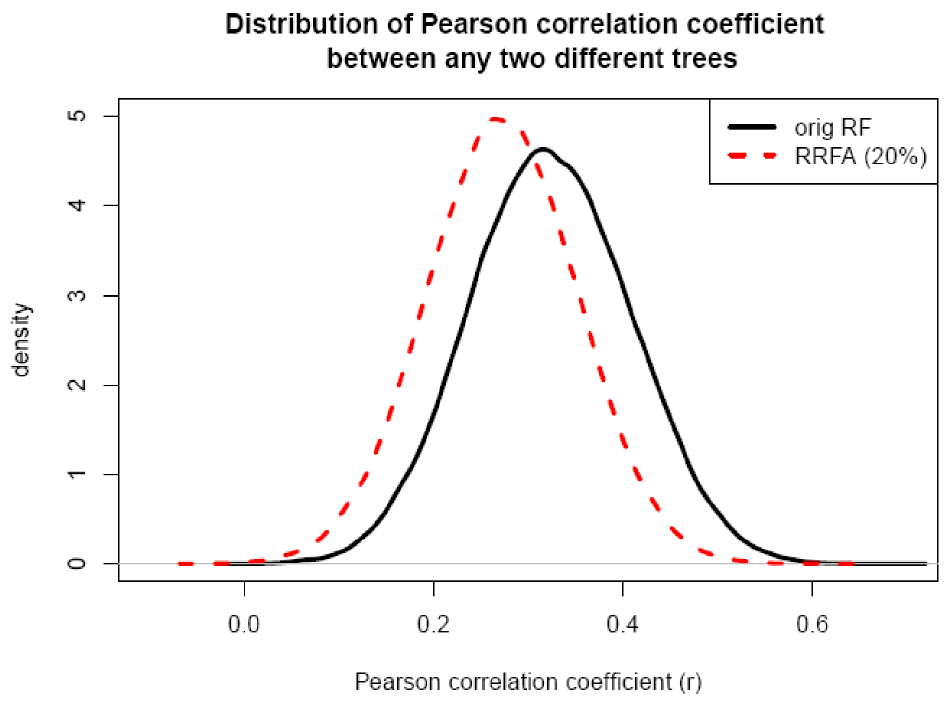


Figure 18: Comparison of misclassification error and Pearson correlation coefficient in RF and RRFA in the Pima Indians dataset with $N_{tr} = 200$ and $N_{te} = 332$

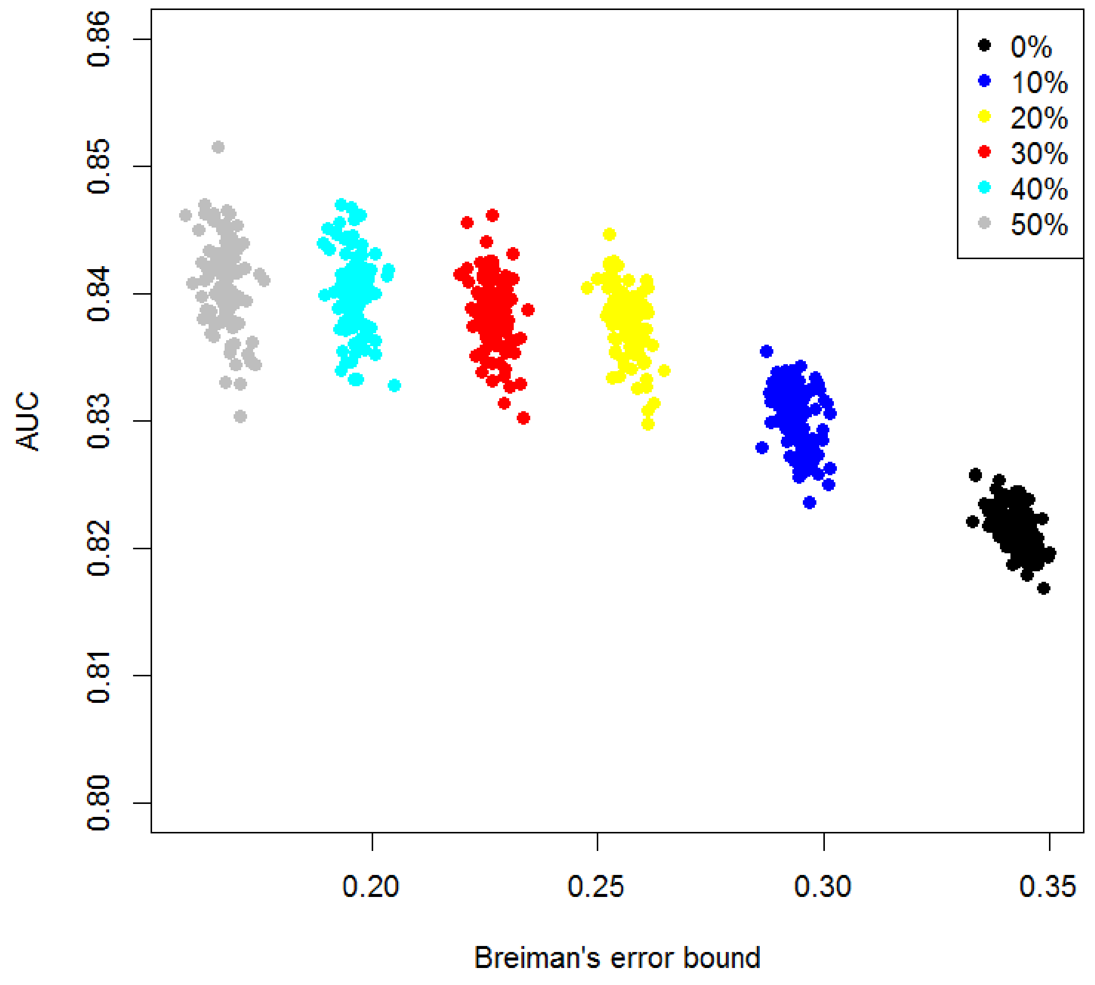


Figure 19: AUC vs. Breiman's error bound in both RF and RRFA in the Pima Indians dataset with $N_{tr} = 200$ and $N_{te} = 332$

2.7.3 Assessment Metrics for Binary Classification Performance

AUC is inversely associated with the misclassification error and it is more sensitive than misclassification error in detecting performance improvements. For all of the above four datasets, the Hand index (Hand, 2009), and the Kolmogorov-Smirnov test statistic can also be used to assess the predictive performance in testing datasets. Similar improvements are also observed and therefore results are not shown here.

Chapter 3. Improved Roughened Random Forests Algorithms

Abstract

Based on the RRFA algorithm from Chapter 2, here we further develop three different versions of RRF algorithms. These three new RRF algorithms are called “Roughened Random Forests - B (RRFB)”, “Roughened Random Forests - C (RRFC)” and “Roughened Random Forests - D (RRFD)”. RRFB restricts the introduction of missing data to the training dataset. RRFC allows users to choose one of the seven imputation methods and they are separately named as RRFC1 to RRFC7. These imputation methods are mean/mode imputation, minimum-value/mode imputation, maximum-value/mode imputation, hot-deck imputation, regression-based imputation, multiple imputation by chained equations (MICE) and proximity-based imputation. RRFD uses different numbers of variables at each tree node split. These three new RRF algorithms (RRFB, RRFC and RRFD) are compared with RRFA and RF with respect to AUC values. RRFB leads to better classification as well as shorter computation time than RRFA. RRFC6 (with MICE as the imputation method) produces slightly better overall performances than RRFB, but RRFC6 requires much longer computation time than RRFB. RRFD also provides better overall performance than RRFB, but computational cost is also higher in RRFD. When there are limited computational resources, RRFB is preferred. When computational resources are not a problem, RRFC and RRFD can potentially provide better performance than RRFB.

3.1 Introduction

In Chapter 2, we discussed that RRFA can help boost the performance of random forests, and that this effect can be used even when complete data are already available for classification. It appears that the virtues of the RRFA can be explained mostly by the reductions in pairwise correlations among all individual trees. Thus, we further propose several new methods that improve random forests' performance in binary classification by modifying pairwise correlations among all individual trees.

3.2 Motivations for Improving RRFA

We will use one example from Chapter 2 to demonstrate the subpar performance of RRFA when the imposed missing values exceed certain percentages. While these algorithms performed satisfactorily in moderate amounts of artificially imposed missing data, their classification performance measured by AUC does not show an inverse linear relationship with Breiman's error bound across the rates of missing data. Below, we summarize our empirical evidence that motivates the further modifications in our original roughened random forest algorithm RRFA.

We use `Pima.tr` (N_{tr}=200) and `Pima.te` (N_{te}=332) from R package `MASS` as the training dataset and the testing datasets. We apply RRFA in the Pima Indians dataset with different rates of missing data (MIS_{pct}) from 10% to 90% by 10%. Instead of using the default value of N_{tree} = 500, we further increase it to N_{tree} = 2000 to have more stable results. We repeat both the original random forests (RF) and RRFA 100 times with the same training dataset and the same testing dataset. In addition, we derive the 2.5th percentile (2.5%), median, mean and 97.5th percentile (97.5%) for AUC based on these 100 repeated experiments as shown in Table 11. The improvement peaks at $MIS_{pct} = 50\%$ and starts going down between 60% and 90%. For the original complete dataset, RF generated an AUC of 0.822. RRFA has the best AUC value of 0.841 with $MIS_{pct} = 50\%$, a 2.3% improvement ($p < 0.05$).

Table 11: AUC comparison of RF vs. RRFA with MIS_{pct} between 10% and 90% in the Pima Indians dataset

MIS_{pct}	2.5%	median	mean	97.5%
0% (RF)	0.818	0.822	0.822	0.826
10%	0.826	0.831	0.831	0.835
20%	0.833	0.837	0.838	0.843
30%	0.833	0.839	0.839	0.844
40%	0.834	0.840	0.840	0.846
50%	**0.835**	**0.841**	**0.841**	**0.848**
60%	0.830	0.838	0.838	0.844
70%	0.823	0.832	0.832	0.843
80%	0.804	0.815	0.815	0.826
90%	0.784	0.797	0.797	0.815

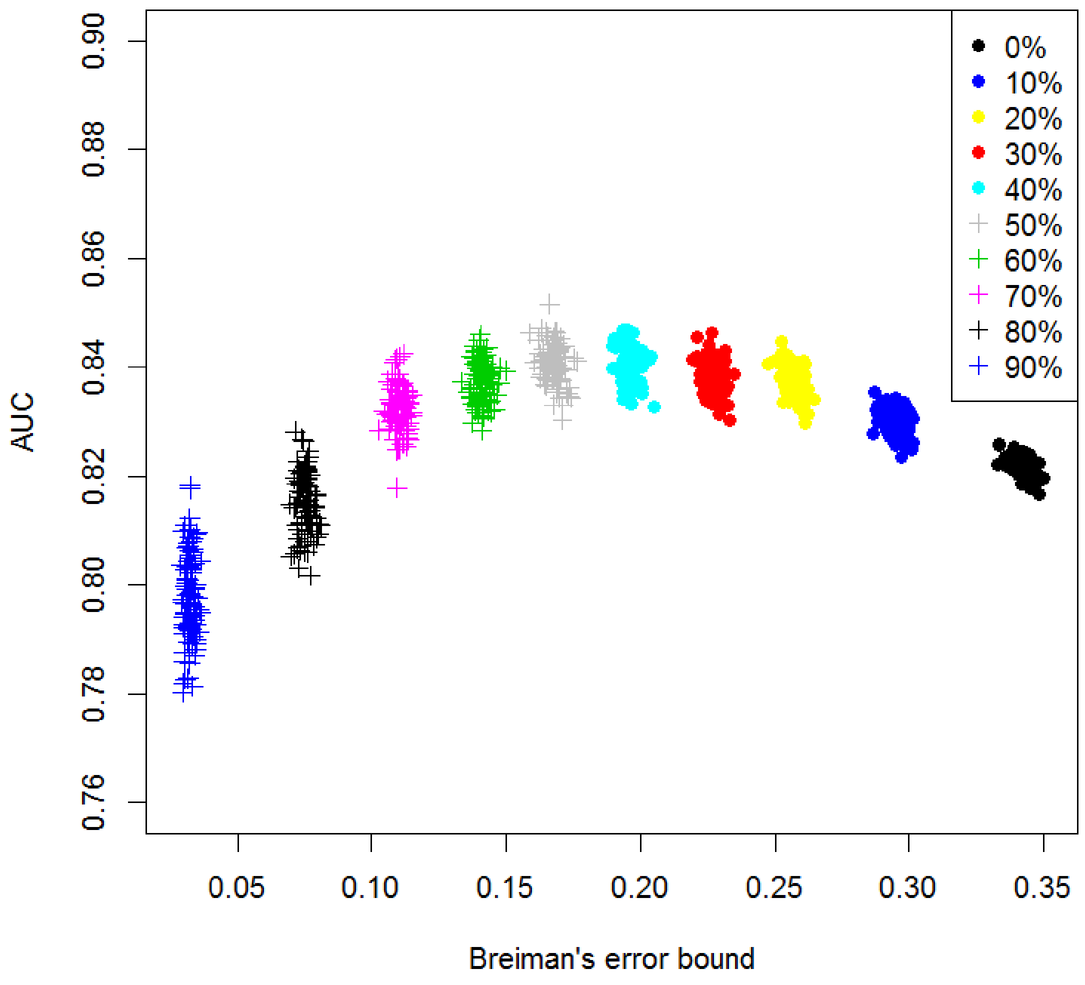


Figure 20: AUC vs. Breiman's error bound in both RF and RRFA (with MIS_{pct} between 10% and 90%) in the Pima Indians dataset with N_{tr}=200 and N_{te}=332

As shown in Figure 20, even though the error bound decreases between 50% and 90% of MIS_{pct}, the AUC does not increase any more. This is mostly because the RRFA algorithm imposes missing data into both the training and testing dataset, but Breiman's error bound does not take into consideration of the modification of the testing dataset.

3.3 Roughened Random Forests - B (RRFB)

Next, we introduce the RRFB algorithm which only imposes missing data on the training dataset. The RRFB algorithm is implemented in the following four steps.

1. Impose missing values under the mechanism of missing completely at random on all covariates of the training dataset.
2. Impute the missing data by median imputation for continuous variables and mode imputation for categorical variables.
3. Build one tree in random forests using the above imputed training dataset, and then use it to predict the binary outcomes in the original testing dataset.
4. Repeat 1 to 3 for N_{tree} times, in total N_{tree} different trees are built, and N_{tree} different sets of predictions are made for the binary outcomes in the testing dataset.

We apply RRFB in the same experiment as in 3.2. For the original complete dataset, random forests (RF) generate a mean AUC value of 0.822, with a 2.5th percentile of 0.818 and a 97.5th percentile of 0.826 in 100 repeated experiments. When MIS_{pct} = 70%, RRFB achieves a mean value of AUC value at 0.845 with a 2.5th percentile of 0.840 and a 97.5th percentile of 0.850 in 100 repeated experiments. When MIS_{pct} = 70%, RRFB achieves a 2.8% improvement over the original RF in average AUC. In Table 12, we report more detailed results for each different MIS_{pct}. We also observe that RRFB leads to similar performance with respect to width of the underlying empirical 95% confidence interval.

RRFB and RRFA both have significant improvements ($p<0.05$) over the original RF. RRFB is also better than RRFA with a 0.5% improvement over RRFA in average AUC, but the improvement from RRFA to RRFB is not significant ($p>0.05$). We further investigate the relationship between AUC and Breiman's error bound in Figure 21 for RRFB. As the testing dataset is kept complete under RRFB, Figure 21 shows a negative linear relationship between AUC and Breiman's error bound. MIS_{pct} = 70% helps achieve the lowest Breiman's error bound and the highest AUC value.

Table 12: AUC comparison of RF vs. RRFB with MIS_{pct} between 10% and 90% in the Pima Indians dataset

MIS_{pct}	2.5%	median	mean	97.5%
0% (RF)	0.818	0.822	0.822	0.826
10%	0.822	0.826	0.826	0.830
20%	0.826	0.830	0.830	0.834
30%	0.829	0.834	0.834	0.838
40%	0.835	0.839	0.839	0.845
50%	0.836	0.842	0.842	0.846
60%	0.838	0.844	0.844	0.851
70%	**0.840**	**0.845**	**0.845**	**0.850**
80%	0.839	0.844	0.845	0.850
90%	0.827	0.835	0.835	0.843

In real-life applications, imputing only the training dataset can save users from repeating the lengthy random forests building process when we apply it on a different testing dataset. These results imply that, users may prefer RRFB when the computation time is deemed to be a factor in decision-making as RRFB significantly reduces computation time by only imposing missing data in the training data.

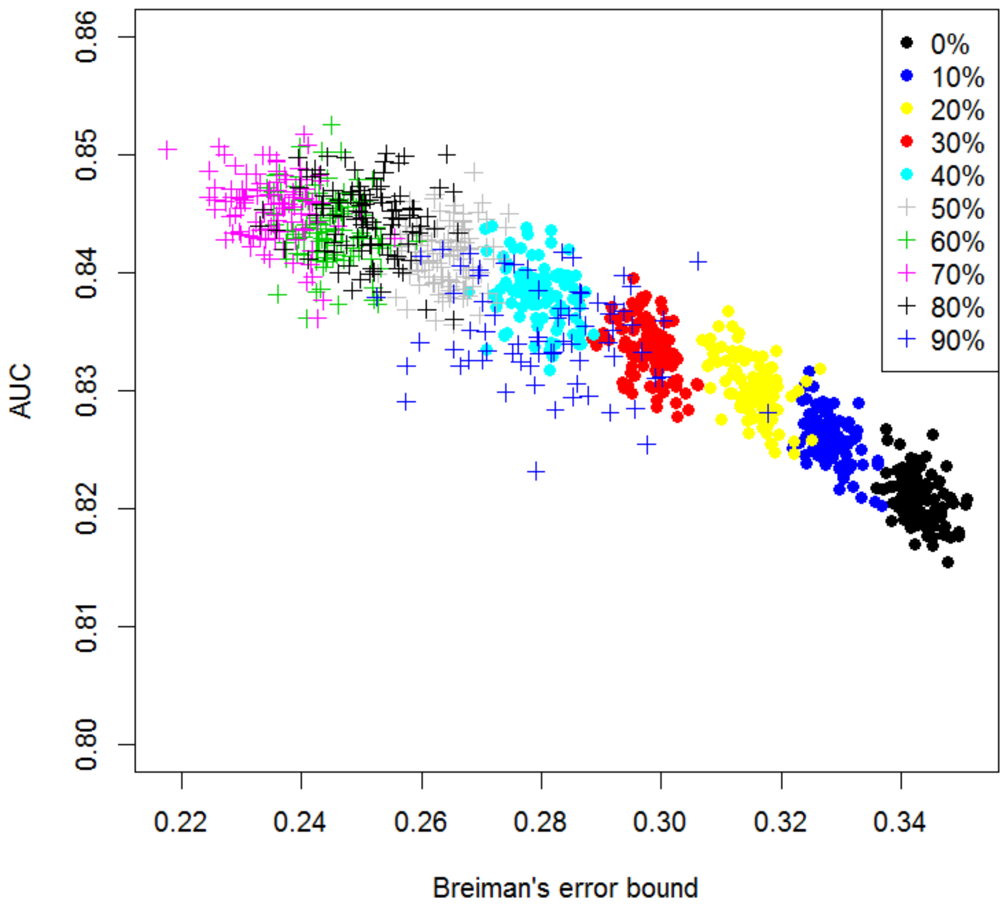


Figure 21: AUC vs. Breiman's error bound in both RF and RRFB (with MIS_{pct} between 10% and 90%) in the Pima Indians dataset with $N_{tr}=200$ and $N_{te}=332$

3.4 Roughened Random Forests - C (RRFC)

We used single imputation (median/mode imputation) in RRFA and RRFB because it is the faster one between the two missing data imputation methods used in the original random forests (Breiman & Cutler, 2004). Single imputation has been criticized extensively in the statistical literature (Rubin, 1987; Schafer & Graham, 2002; Little & Rubin, 2002) as it potentially underestimates uncertainty measures, RRFC is motivated to include multiple imputation methods (Little & Rubin, 2002; Yucel, 2011) which can circumvent this problem. The RRFC algorithm is described below in four steps.

1. Impose missing values under the mechanism of missing completely at random on all covariates of the training dataset.
2. Impute the missing data by one of the 7 listed* imputation methods other than median/mode imputation (called RRFC1, RRFC2, … , RRFC7).
3. Build one tree using the above imputed training dataset, and then use it to predict the binary outcomes in the original testing dataset.
4. Repeat 1 to 3 for N_{tree} times, in total N_{tree} different trees are built, and N_{tree} different sets of predictions are made for the binary outcomes in the testing dataset.

*The imputation methods used in Step 2 of RRFC include:

RRFC1. Impute the missing values in a continuous variable by its mean value and impute the missing values in a categorical variable by its mode value (Mean/mode imputation).

RRFC2. Impute the missing values in a continuous variable by its minimum value and impute the missing values in a categorical variable by its mode value (Minimum-value /mode imputation).

RRFC3. Impute the missing values in a continuous variable by its maximum value and impute the missing values in a categorical variable by its mode value (Maximum-value /mode imputation).

RRFC4. Hot-deck imputation for all variables. For each variable, observed values are randomly selected to impute missing values.

RRFC5. Regression-based imputation for all variables. Linear regression is used to impute continuous variables. Logistic regression is used to impute binary variables. And multinomial logistic regression is used to impute categorical variables with three or more categories.

RRFC6. Multiple imputation by chained equation (Van Buuren & Groothuis-Oudshoorn, 2011) is used to produce the imputed dataset (implemented by `mice` function in R package `mice`).

RRFC7. Missing data is imputed based on proximity from random forests (implemented by `rfImpute` function in R package `randomForest`).

3.4.1 Differences Between RRFC5, RRFC6 and RRFC7

RRFC5 and RRFC6 are both regression-based imputation methods. Missing data are imposed to each variable in both RRFC5 and RRFC6. In RRFC5, for each variable X_i, a regression model is built using all other variables in the complete dataset before any missing data is imposed. Therefore, the regression model can produce fitted values for each variable in each observation. Then, these fitted values are used to impute missing data that are imposed in RRFC5. RRFC6 uses multiple imputation by chained equation (MICE) which is based on fully conditional specification. In RRFC6, each variable with missing data is imputed sequentially. This process repeats for a few iterations. For a variable X_i with missing values, we denote the observed and missing values as X_i^{obs} and X_i^{mis}. A regression model is applied using X_i^{obs} as the outcome and all other variables as predictors, in which missing values are initially replaced by single imputations. Then X_i^{mis} is replaced by predicted values based on the above fitted regression model. This process cycles through all variables with missing values until the convergence of the sampling distribution of all variables with imputed values, or reaching the maximum number of iterations (Van Buuren & Groothuis-Oudshoorn, 2011).

RRFC7 adopts a completely different method to impute missing values. After filling in missing values by median/mode imputation, it builds a random forest to find the proximity matrix between observations. The proximity matrix is then used to fill in the originally missing values to create a newly imputed dataset, which is again used to build a new random forest to find a new proximity matrix. This process is usually repeated for a few iterations and the last imputed dataset is kept for use. The proximity matrix is used differently for imputation of continuous variables and categorical variables. For continuous variables, missing values are imputed based on the proximity-based weighted average of observed values; for categorical variables, missing values are imputed based on the proximity-based weighted mode of observed values (Breiman & Cutler, 2004; Liaw & Wiener, 2002).

3.4.2 Datasets

The following 12 different datasets are mainly from the University of California Irvine (UCI) Machine Learning Repository (Bache & Lichman, 2013). They are a diverse combination of datasets that can be used to compare our RRFC algorithms with RRFB and RF. After data cleaning, there are no missing data or duplicated records. We briefly describe these datasets below.

Balance

This data set is from the UCI Machine Learning Repository with the original name "Balance Scale Data Set". It has four continuous variables besides the outcome variable. The outcome variable includes: 49 "B", 288 "L" and 288 "R". We will only keep the 288 "L" and 288 "R" for our analysis.

Blood

This data set is from the UCI Machine Learning Repository with the original name "Blood Transfusion Service Center Data Set" (Yeh, et al., 2009). It has four continuous variables besides the outcome variable. There were originally 748 cases. After deduplication, the outcome variable has 149 donors and 384 non-donors.

Blowdown

This data set is from R package `alr3` (Weisberg, 2011). The dataset has two continuous variables and one categorical variable besides the outcome variable. There were originally 3666 cases. After deduplication, the outcome variable has 1582 live trees and 1736 dead trees.

Contraceptive

This data set is from the UCI Machine Learning Repository with the original name "Contraceptive Method Choice Data Set". It has two continuous variables and seven categorical variables besides the outcome variable. There were originally 1473 cases. After deduplication, the outcome variable has 811 long-term and short-term contraceptive users plus 614 non-users.

Credit

This data set is from the UCI Machine Learning Repository with the original name "Credit Approval Data Set". It has six continuous variables and nine categorical variables besides the outcome variable. There were originally 690 cases. After deduplication and taking out missing values, the outcome variable has 296 positive cases and 357 negative cases.

CTG

This data set is from the UCI Machine Learning Repository with the original name "Cardiotocography Data Set". It has 21 continuous variables besides the outcome variable. There were originally 2126 cases. After deduplication and taking out the "Normal" cases, the outcome variable has 175 "Pathologic" and 292 "Suspect" cases in our analysis.

Haberman

This data set is from the UCI Machine Learning Repository with the original name "Haberman's Survival Data Set". It has three continuous variables besides the outcome variable. There were originally 306 cases. After deduplication, the outcome variable has 79 deaths and 210 survivors.

ILPD

This data set is from the UCI Machine Learning Repository with the original name "ILPD (Indian Liver Patient Dataset) Data Set". It has nine continuous variables and one categorical variable besides the outcome variable. There were originally 583 cases. After deduplication, the outcome variable has 162 positive cases and 404 negative cases.

Mammography

This data set is from the UCI Machine Learning Repository with the original name "Mammographic Mass Data Set" (Elter, et al., 2007). It has two continuous variables and two categorical variables besides the outcome variable. There were originally 961 cases. After deduplication and taking out missing data, the outcome variable has 230 malignant cases and 257 benign cases.

Statlog

This data set is from the UCI Machine Learning Repository with the original name "Statlog (Heart) Data Set". It has seven continuous variables and six categorical variables besides the outcome variable. The outcome variable has 120 positive cases and 150 negative cases.

Titanic

This dataset is from the Vanderbilt University Department of Biostatistics website with the original name “titanic3” (Wang, 2014). We will use three continuous variables (“Age”, “Number of Siblings/Spouses Aboard”, “Number of Parents/Children Aboard”) and three categorical variables (“Passenger class as 1st”, “Passenger class as 2nd”, “sex”) besides the outcome variable. There were originally 1309 cases. After deduplication and taking out missing data, the outcome variable has 324 survivors and 356 deaths.

Yeast

This data set is from the UCI Machine Learning Repository with the original name “Yeast Data Set”. It has eight continuous variables besides the outcome variable. There were originally 1484 cases. We will only keep two major categories “CYT” and “NUC” for our analysis. After deduplication, the outcome variable has 438 “CYT” and 425 “NUC”.

3.4.3 Experiments

Our experiments aim to compare RF with RRFB and RRFC in the above listed 12 datasets as well as the Pima Indians dataset. After data cleaning, all these datasets are complete and have no duplicated records. Also, we only use the same amount of observations from each class to avoid the influence of class imbalance (Chen, et al., 2004).

We propose to use a special 10-fold cross-validation method that can deal with datasets with small sample sizes. For each dataset containing $N_{(Y=1)}$ positive cases and $N_{(Y=0)}$ negative cases, the first Q pairs of positive and negative cases are selected for use. Q is set as the minimum value among $N_{(Y=1)}$, $N_{(Y=0)}$ and 500. These Q pairs of cases are then divided into five subsamples with equal sizes, and each subsample should have 50% of positive cases and 50% of negative cases. We randomly choose two subsamples out of the five total subsamples to create a testing dataset, and the rest of three subsamples should form a training dataset. The training dataset should have 60% of the total data and the testing dataset should have 40% of the total data. In total, we should have $C_5^2 = 10$ different pairs of training and testing datasets. RF, RRFB and RRFC (RRFC1 to RRFC7) are simultaneously built on each training dataset with MIS_{pct} ranging from 10% to 90% by 10% and N_{tree}=1000. The resulting random forests are applied on the testing datasets for predictions.

3.4.4 Results

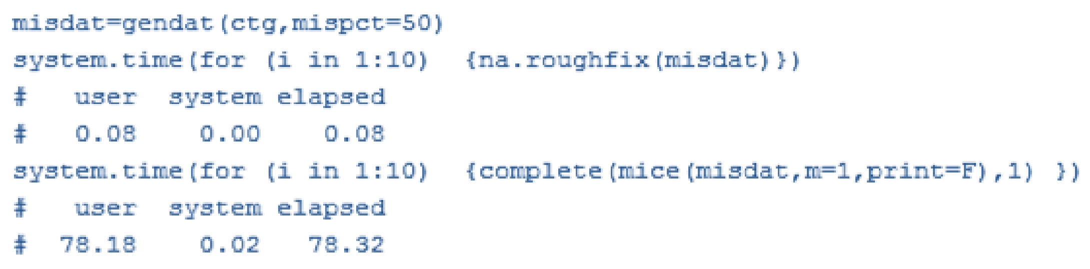
For each of the 13 datasets, the AUC values are averaged across the 10-fold cross-validation for each of the nine different MIS_{pct}. The highest average AUC values among these nine different MIS_{pct} are recorded for each dataset and each algorithm in the first 13 rows of Table 13. The best AUC value in each row is shown in **bold** font. RF does not provide the best performance in any of the 13 datasets.

The 14th to 17th rows of Table 13 show relative performance of RRFB and RRFC using RF as the reference. The 14th row, “#imp”, shows the number of times each algorithm outperforms RF. RRFB and RRFC6 both outperform RF in 12 of the 13 datasets except the CTG dataset. The 15th row, “Max imp%”, shows the highest percentage of AUC improvement among the 13 datasets for each algorithm. RRFC1 has the best performance in this row with 9.1% improvement in “Mammography” dataset. The 16th row, “Median imp%”, shows the best median percentage of AUC improvement among the 13 datasets for each algorithm. RRFC6 has the best performance in this row with 1.4% improvement. The 17th row, “Mean imp%”, shows the best mean percentage of AUC improvement among the 13 datasets for each algorithm. RRFC6 has the best performance in this row with 1.7% improvement.

Overall, RRFC6 provides slightly better performance than RRFB. However, RRFC6 takes much longer computation time than RRFB, as shown in Figure 22. For the CTG dataset, when $MIS_{pct} = 50\%$, RRFB only takes 0.8 seconds to finish ten imputations, but RRFC6 takes 78.32 seconds to finish ten imputations. The above imputations were done in R 2.15.1 on a computer with Intel Core i5-3570 CPU @ 3.40 GHz and 4.00G RAM in a 32-bit Windows 7 Enterprise operation system.

Table 13: AUC comparison in 13 different datasets using RF, RRFB and RRFC

(AUC)	RF	RRFB	RRFC1	RRFC2	RRFC3	RRFC4	RRFC5	RRFC6	RRFC7
	Original	Median	Mean	Min	Max	Hot-deck	Regression	MICE	proximity
Balance	0.9899	0.9967	0.9968	0.9814	0.9864	0.9968	0.9919	**0.9980**	0.9959
Blood	0.5679	0.5799	**0.5893**	0.5869	0.5719	0.5693	0.5882	0.5761	0.5745
Blowdown	0.9690	0.9697	0.9686	0.9675	0.9694	0.9689	0.9696	**0.9712**	0.9696
Contraceptive	0.7537	0.7655	0.7651	0.7506	0.7627	0.7609	0.7710	**0.7734**	0.7649
Credit	0.9263	0.9271	0.9249	0.9253	**0.9272**	0.9263	0.9257	0.9267	0.9251
CTG	0.9895	0.9877	0.9878	**0.9898**	0.9816	0.9877	0.9891	0.9889	0.9890
Haberman	0.8768	0.8910	0.8925	0.8730	0.8744	0.8820	0.8846	**0.8961**	0.8744
ILPD	0.7746	0.7902	0.7819	0.7906	0.7747	0.7862	0.7830	**0.7962**	0.7749
Mammography	0.7240	0.7862	**0.7901**	0.7861	0.7672	0.7736	0.7484	0.7837	0.7760
Statlog	0.9047	0.9073	0.9124	0.9049	0.9101	0.9088	0.9117	0.9089	**0.9137**
Titanic	0.8031	0.8060	0.8187	0.8070	0.8129	0.8127	0.8137	0.8151	**0.8189**
Yeast	0.7098	**0.7172**	0.7146	0.7067	0.7133	0.7152	0.7071	0.7165	0.7133
Pima Indians	0.8335	0.8425	0.8410	0.8359	0.8340	0.8411	0.8360	**0.8461**	0.8449
#imp	*ref*	**12**	10	7	10	10	10	**12**	10
Max imp%	*ref*	8.6%	**9.1%**	8.6%	6.0%	6.9%	3.6%	8.2%	7.2%
Median imp%	*ref*	1.0%	0.9%	0.0%	0.1%	0.7%	0.8%	**1.4%**	0.6%
Mean imp%	*ref*	1.5%	1.7%	1.0%	0.7%	1.1%	1.0%	**1.7%**	1.1%



```
misdat=gendat(ctg,mispct=50)
system.time(for (i in 1:10)  {na.roughfix(misdat)})
#   user  system elapsed
#   0.08    0.00    0.08
system.time(for (i in 1:10)  {complete(mice(misdat,m=1,print=F),1) })
#   user  system elapsed
#  78.18    0.02   78.32
```

Figure 22: Difference in computation time between RRFB and RRFC6 with regards to missing data imputation.

3.5 Roughened Random Forests - D (RRFD)

3.5.1 RRFD Algorithm

RRFD is based on RRFB, with further exploration on the ideal choice of m in Step 3. The four steps of RRFD algorithm are listed below.

1. Impose missing values under the mechanism of missing completely at random on all covariates of the training dataset.
2. Impute the missing data by median imputation for continuous variables and mode imputation for categorical variables.
3. Build one tree with a certain m (between 1 and M) value using the above imputed training dataset, and then use it to predict the binary outcomes in the original testing dataset.
4. For each m value, repeat 1 to 3 for N_{tree} times, in total N_{tree} different trees are built, and N_{tree} different sets of predictions are made for the binary outcomes in the testing dataset. The m value with the best final AUC is selected for use.

3.5.2 RRFD in the Pima Indians Dataset

We apply RRFD in the same experiment as in 3.2. The results are shown in Figure 23.

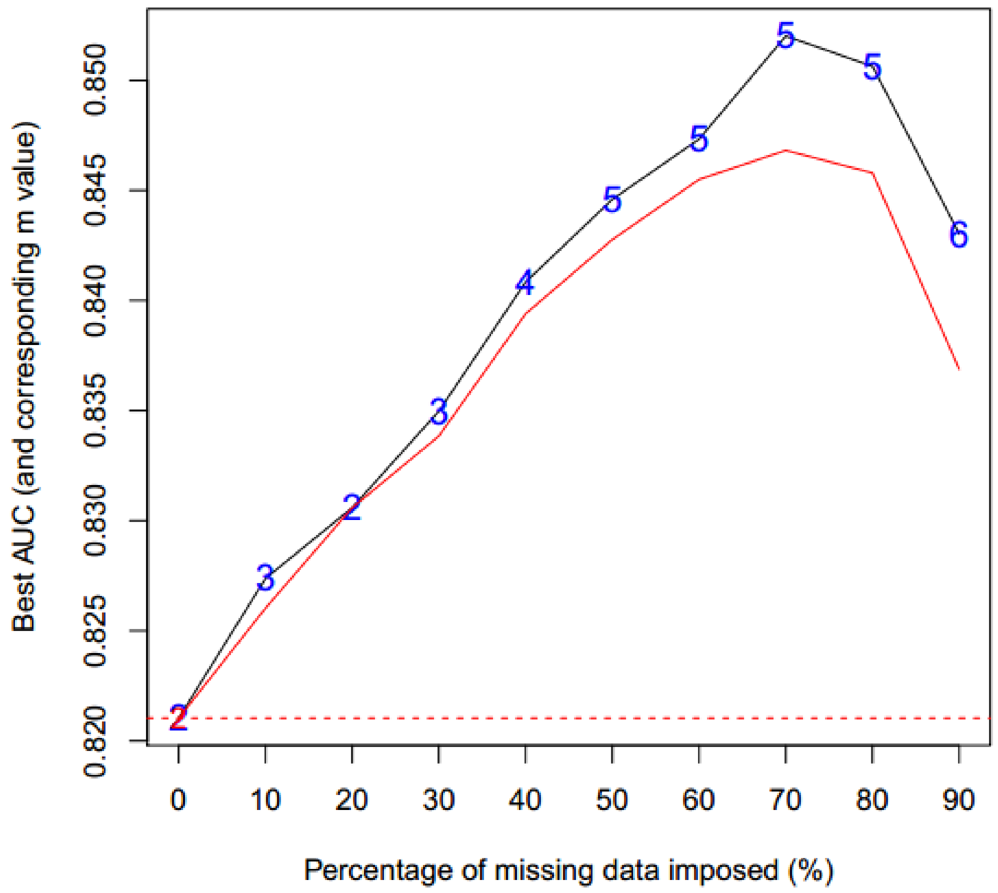


Figure 23: RF, RRFB and RRFD in the Pima Indians dataset

In Figure 23, the red dashed line represents the AUC of the original random forests (RF) with the default m value. For the Pima Indians dataset, m=2, or the integer part of the square root of 7. The red solid line, or RRFB, shows how AUC changes with different rates of missing data when m is set to the default value. The black solid line, or RRFD, shows how the best AUC changes with different rates of missing data when m is chosen from all the possible values. For the Pima Indians dataset, m is chosen from 1, 2, 3, 4, 5, 6 and 7. The blue numbers on the black line show the ideal m values at each rate of missing data in RRFD.

For the original complete dataset, m = 2 is the ideal choice. However, when we impose more missing data, the ideal m value also goes up. The best AUC value of 0.852 is achieved with MIS_{pct} = 70% and m = 5. In this case, RRFD produces a 3.6% increase over the original random forests while RRFB only produces a 2.8% increase at m = 2.

3.5.3 RRFD in 12 Different Datasets

We further test RRFD using the same 12 datasets from 3.4.2. Due to the computation time requirement, we will not do 10-fold cross-validation using RRFD. Instead, we aim to illustrate the performance of RRFD for each dataset across different rates of imposed missing data. For each one of the 12 datasets, we divide the original dataset evenly into the training dataset and the testing dataset using the same rule. For example, if there are 200 observations, all the even-numbered observations (2nd, 4th, 6th until 200th) will be used as the training dataset, and all the odd-numbered observations (1st, 3rd, 5th until 199th) will be used as the testing dataset. We apply RF, RRFB and RRFD in each dataset and the AUC results are shown in Figure 24 and Table 14.

In each of the 12 graphs in Figure 24, the red dashed line represents the AUC of the original RF using the default m value. The red solid line, or RRFB, shows how AUC changes with different rates of missing data using the default m value. The black solid line, or RRFD, shows the best AUC with different rates of missing data using the ideal m values. The blue numbers on the black line show the ideal m values for each rate of missing data.

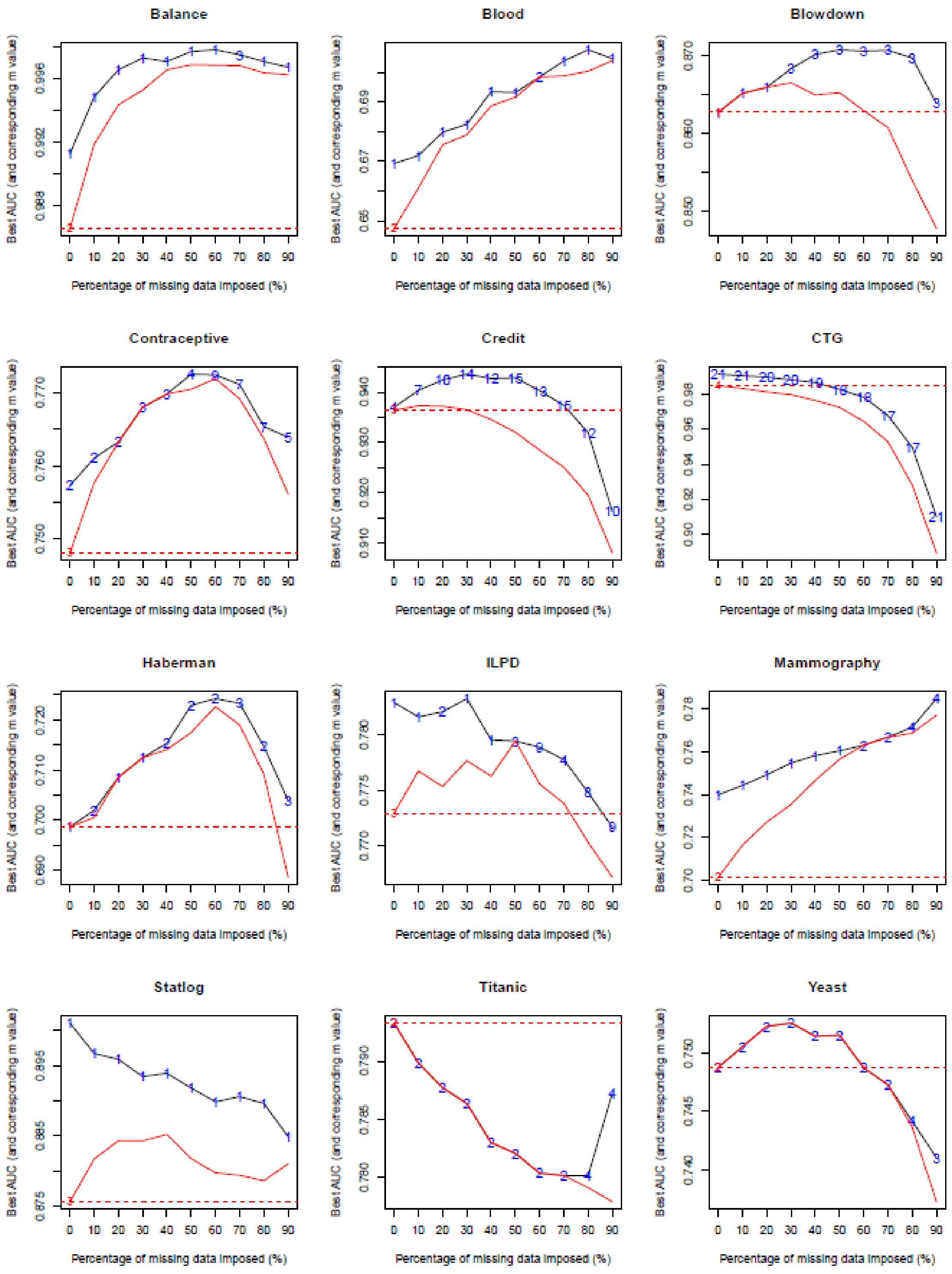


Figure 24: RF, RRFB and RRFD in 12 different datasets with $N_{tr}/N_{te} = 1$

Table 14: Summary of RF, RRFB and RRFD in 12 different datasets

	Covariates		RRFB		RRFD	
	#cont	#cat	Improve%	m (MIS_{pct})	Improve%	m (MIS_{pct})
Balance	4	0	1.0%	2 (50%)	1.1%	1 (60%)
Blood	4	0	8.7%	2 (90%)	9.3%	1 (80%)
Blowdown	2	1	0.4%	1 (30%)	0.9%	3 (50%)
Contraceptive	2	7	3.2%	3 (60%)	3.3%	4 (50%)
Credit	6	9	0.1%	3 (10%)	0.8%	14 (30%)
CTG	21	0	0%	4 (0%)	0.7%	21 (0%)
Haberman	3	0	3.4%	1 (60%)	3.7%	2 (60%)
ILPD	9	1	0.8%	3 (50%)	1.3%	1 (30%)
Mammography	2	2	10.8%	2 (90%)	11.9%	4 (90%)
Statlog	7	6	1.1%	3 (40%)	2.9%	1 (0%)
Titanic	3	3	0%	2 (0%)	0%	2 (0%)
Yeast	8	0	0.5%	2 (30%)	0.5%	2 (30%)

As shown in Figure 24 and Table 14, RRFB on average produces a 2.5% increase in AUC with a range between 0% and 10.8% over the original random forests. RRFD on average produces a 3.0% increase in AUC with a range between 0% and 11.9% over the original random forests. The ideal m value can be larger or smaller than the default m value. The best MIS_{pct} varies across different datasets Also, the improvements do not seem to correlate with the ratio of the number of continuous variables (#cont) and the number of categorical variables (#cat).

For the two least improved datasets (CTG and Titanic datasets) above, we change the size ratio of training and testing datasets a few times and rerun the same experiments, the results are shown in Figure 25.

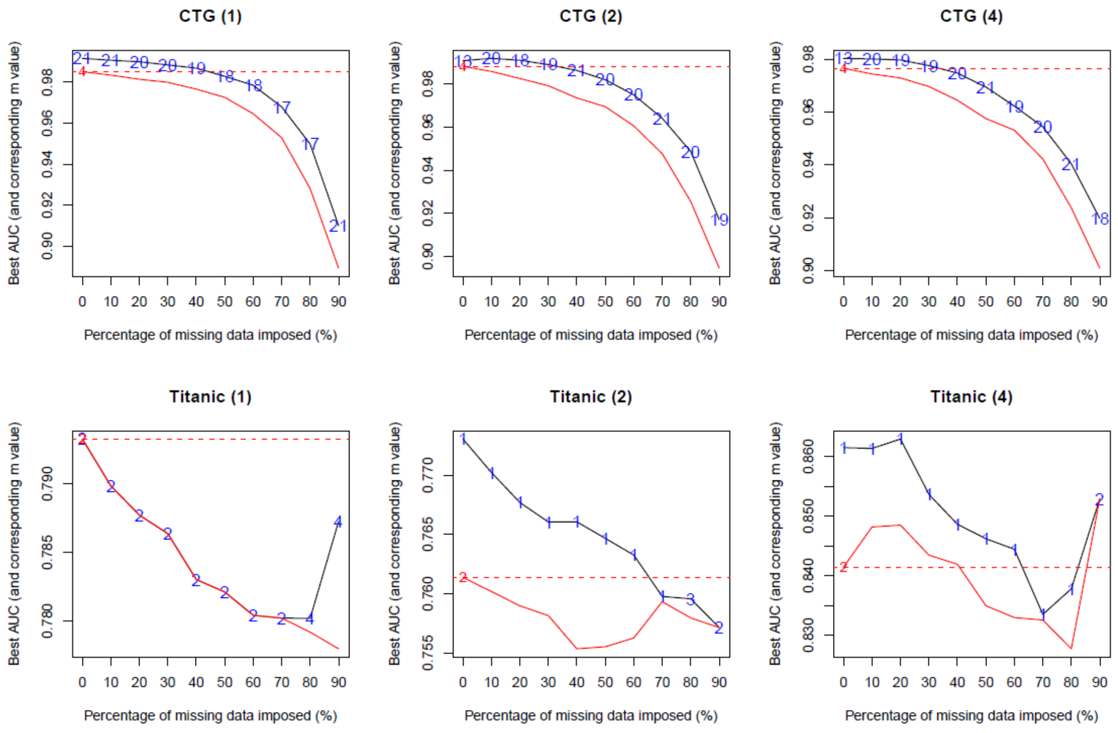


Figure 25: AUC comparison of RF, RRFB and RRFD in the CTG and Titanic datasets with size ratios between training and testing datasets at 1, 2 and 4

For the CTG dataset, RRFB does not show any improvement over RF even after changing the size ratio of training and testing data. RRFD can achieve slightly better performance than RF when no missing data is imposed. We just need to change the m value in the original RF to improve the AUC performance in the CTG dataset. For the Titanic dataset, after changing the size ratio of the training and testing data, RRFB and RRFD can both produce improvements over RF.

3.6 Discussion

In this chapter, we investigate three different algorithms (RRFB, RRFC and RRFD) that aim at improving RRFA's binary classification performance. AUC is used as the only assessment metric in this chapter as it is more sensitive than accuracy in detecting performance improvement based on the discussion in Chapter 2.

RRFB achieves better AUC performance than RRFA with shorter computation time in the Pima Indians dataset. Also, there is an inverse linear relationship between AUC and Breiman's error bound when using RRFB. RRFC6 has slightly better classification performance than RRFB but RRFC6 requires much longer computation time. None of the other RRFC algorithms match RRFB in overall performance.

The purpose of imposing missing data and subsequent imputation in any RRF algorithm is mainly to create diversion from the original data. The main advantage of multiple imputation over single imputation is its reflection of uncertainty in imputing missing values. Therefore, conceptually multiple imputation is not more useful than single imputation in roughened random forests. RRFC7 uses proximity-based imputation which is usually better than RRFB's median/mode imputation in handling regular missing data analysis. However, RRFC7 is not any better than RRFB in AUC improvement. On the other hand, with the help of multiple imputation by MICE, RRFC6 provides slightly better AUC performance than RRFB at the cost of significantly longer computation time.

Leave-one out cross validation was found to be asymptotically inconsistent in selecting the best linear model in terms of predictive ability (Shao, 1993). Shao (1993) concluded that the sample size used for validation should be as large as possible. In another words, N_{te}/N_{tr} should be as large as possible in cross-validation for linear model selection. To compare different RRFC algorithms, we used a special 10-fold cross-validation so that N_{te}/N_{tr} is equal to 4/6 instead of 1/9 in each fold. We did not further increase N_{te}/N_{tr} so that we have enough sample size for training the roughened random forests.

Finally, we look at the most important parameter m value within RF. The default m value can usually achieve good classification performance in RF, but RRF algorithms do not necessarily share the same default m values as RF. RRFD can produce further improvements over RRFB by using different m values. However, there is no golden rule as to what should be the ideal m value used in RRFD. Consequently, RRFD needs to test different m values and requires much longer computation time than RRFB.

Among the 13 datasets used in this chapter, the CTG dataset which has 21 continuous variables is least improved by any above RRF algorithms. We might need to further modify RRF algorithms when we are dealing with datasets with higher dimensions in order to achieve AUC improvement over the original RF.

Chapter 4. Roughened Random Forests - E (RRFE) Algorithm in Medium- to High-dimensional Datasets

Abstract

The RRFE algorithm is created for implementing roughened random forests in medium- to high-dimensional datasets. The major difference between RRFE and RRFB is the selective introduction of missing data based on variable importance according to the original random forests. In five medium-dimensional datasets with between 20 to 200 covariates, experimental results show that RRFB generally does not outperform the original random forests with respect to AUC, but RRFE can lead to improved AUC over the original random forests. The improvement in RRFE mainly comes from the uneven decreases in strength and correlations among individual classification trees, and consequently the decrease in Breiman's generalization error bound. RRFE is also computationally more efficient than RRFB. In two high-dimensional microarray datasets with 2000 and 2905 covariates and under 200 observations, experimental results show that RRFB and RRFE can both improve AUC over the original RF after variable selection, with RRFB slightly better than RRFE.

4.1 Introduction

Previously introduced RRF algorithms (RRFA, RRFB, RRFC and RRFD) achieve AUC improvement in binary classification over RF mainly by unevenly reducing both strength and correlation among individual classification trees, which lead to a lower Breiman's generalization error bound. All of these RRF algorithms non-discriminately impose the same amount of missing data to each variable, which work relatively well in datasets with under 20 covariates. However, RRFB and RRFC6 algorithms did not lead to any AUC improvement in the CTG dataset with 21 continuous variables. In this chapter, we further propose to show how RRFE can be used to improve the performance of random forests in medium- to high-dimensional datasets with between 20 to 3000 covariates.

4.2 RRFE Algorithm

In random forests, a value is given to represent each variable's importance based on its role in reducing Gini impurity among all individual classification trees. In RRFE, the probability that missing data is imposed on a certain variable is conditional on the variable's relative importance. The RRFE algorithm is listed as below.

1. Impose missing values under the mechanism of missing completely at random on selected covariates of the training dataset, and the probability that missing data is imposed on a certain variable is based on the k-th power of the variable's relative importance. A variable's relative importance is defined as its variable importance divided by the maximum variable importance among all available covariates according to the original random forests.
2. Impute the missing data by median imputation for continuous variables and mode imputation for categorical variables.

3. Build one tree in random forests using the above imputed training dataset, and then use it to predict the binary outcomes in the original testing dataset.
4. Repeat 1 to 3 for N_{tree} times, in total N_{tree} different trees are built, and N_{tree} different sets of predictions are made for the binary outcomes in the testing dataset.

4.3 Datasets

CTG is the same dataset from 3.4.2

Dermatology

This data set is from the UCI Machine Learning Repository with the original name "Dermatology Data Set". It has 33 continuous variables besides the outcome variable. There were originally 366 cases. After deduplication and taking out missing values, the outcome variable has 187 positive cases and 171 negative cases.

Ionosphere

This data set is from the UCI Machine Learning Repository with the original name "Ionosphere Data Set". It has 34 continuous variables besides the outcome variable. There were originally 351 cases. After deduplication, the outcome variable has 225 positive cases and 125 negative cases.

Musk

This data set is from the UCI Machine Learning Repository with the original name "Musk (Version 1) Data Set". It has 166 continuous variables besides the outcome variable. All 476 unique cases are unique and complete. The outcome variable contains 207 musks and 269 non-musks.

Steel

This data set is from the UCI Machine Learning Repository with the original name "Steel Plates Faults Data Set" (It is provided by Semeion, Research Center of Sciences of Communication, Via Sersale 117, 00128, Rome, Italy). It has 27 continuous variables besides the outcome variable. All 1941 unique cases are unique and complete. The outcome variable contains 673 positive cases and 1268 negative cases.

Alon

This microarray dataset is from R package `datamicroarray` (Ramey, 2013; Alon, et al., 1999). It has 2000 continuous variables besides the binary outcome variable. The outcome variable contains 40 positive cases and 22 negative cases.

Gravier

This microarray dataset is from R package `datamicroarray` (Ramey, 2013; Gravier, et al., 2010). It has 2905 continuous variables besides the binary outcome variable. The outcome variable contains 111 positive cases and 57 negative cases.

4.4 Experiments

We aim to compare RF with RRFB and RRFE in the above listed datasets with the same 10-fold cross-validation experiment used in 3.4.3. For each dataset containing $N_{(Y=1)}$ positive cases and $N_{(Y=0)}$ negative cases, the first Q pairs of positive and negative cases are selected for use. Q is set as the minimum value among $N_{(Y=1)}$, $N_{(Y=0)}$ and 500. These Q pairs of cases are then divided into five subsamples with equal sizes, and each subsample should have 50% of positive cases and 50% of negative cases. We randomly choose two subsamples out of the five total subsamples to create a testing dataset, and the rest of three subsamples should form a training dataset. The training dataset should have 60% of the total data and the testing dataset should have 40% of the total data. In total, we should have $C_5^2 = 10$ different pairs of training and testing datasets. RF, RRFB and RRFE are simultaneously built on each training dataset with k values ranging from 1 to 5 by 1, MIS_{pct} ranging from 10% to 50% by 10%, as well as MIS_{pct} ranging from 1% to 5% by 1%. N_{tree}=1000 is used in all experiments. The resulting random forests are applied on the testing datasets. All AUC performance for RF, RRFB and RRFE are averaged across these 10-fold cross-validations.

4.5 Results in Medium-dimensional Datasets

We show all results in the following tables. Those results better than the original RF are in **bold** font. The best results for each dataset among all listed results (Table 15, Table 16, Table 17, Table 18, Table 19 and Table 20) are both **bold** and <u>underlined</u>.

In Table 15, for the dermatology and musk datasets, RRFE shows consistent improvement over RF for all five different k values (1, 2, 3, 4 and 5) when MIS_{pct} changes from 10% to 50% by 10%. But RRFB does not show any improvement over RF when MIS_{pct} changes from 10% to 50% by 10%. No improvement is observed by either RRFB or RRFE in three other datasets. In Table 16, RRFE with MIS_{pct} =1% and k=3 produces improvements over the original RF in all five listed datasets. However, the improvements in dermatology and musk datasets using MIS_{pct} ranging from 1% to 5% by 1% are not as good as using MIS_{pct} ranging from 10% to 50% by 10%.

In Table 17, when we change m value from $\lfloor\sqrt{M}\rfloor$ to $\lfloor 2\sqrt{M}\rfloor$ and use MIS_{pct} ranging from 10% to 50% by 10%, RRFE with MIS_{pct} =10% and k=3 produces improvements in AUC in all five datasets, and RRFE produces AUC improvements in four out of five datasets in 16 other scenarios. In Table 18, when we change m value from $\lfloor\sqrt{M}\rfloor$ to $\lfloor 2\sqrt{M}\rfloor$ and use MIS_{pct} ranging from 1% to 5% by 1%, RRFE produces AUC improvements in all five datasets in 4 different scenarios, and RRFE produces AUC improvements in four out of five datasets in 7 other scenarios.

In Table 19, when we change m value from $\lfloor\sqrt{M}\rfloor$ to $\lfloor\sqrt{M}/2\rfloor$ and use MIS_{pct} ranging from 10% to 50% by 10%, again we only see AUC improvements by RRFE in the dermatology and musk datasets. In Table 20, when we change m value from $\lfloor\sqrt{M}\rfloor$ to $\lfloor\sqrt{M}/2\rfloor$ and use MIS_{pct} ranging from 1% to 5% by 1%, the improvements in AUC are much less frequent than Table 16 and Table 18 where m values are set as $\lfloor\sqrt{M}\rfloor$ and $\lfloor 2\sqrt{M}\rfloor$.

Table 15: AUC comparison of RF, RRFB and RRFE in medium-dimensional datasets with m set as square root of M and MIS_{pct} ranging from 10% to 50% by 10%

	MIS_{pct}	k	CTG	Dermatology	Ionosphere	Musk	Steel	IMP
RF	0%	0	0.989428	0.997451	0.968297	0.951957	0.934287	ref
RRFE	10%	1	0.988235	**0.997594**	0.966380	**0.952203**	0.931516	2/5
RRFE	10%	2	0.988704	**0.997637**	0.967260	**0.952023**	0.932391	2/5
RRFE	10%	3	0.988878	**0.997753**	0.967100	**0.952944**	0.932773	2/5
RRFE	10%	4	0.988704	**0.997724**	0.967020	**0.953238**	0.933183	2/5
RRFE	10%	5	0.989418	**0.997647**	0.966600	**0.952564**	0.933531	2/5
RRFE	20%	1	0.987439	**0.997488**	0.964560	**0.952638**	0.929233	2/5
RRFE	20%	2	0.987969	**0.997872**	0.964960	**0.953833**	0.931885	2/5
RRFE	20%	3	0.988306	**0.997693**	0.966340	**0.953090**	0.932873	2/5
RRFE	20%	4	0.988510	**0.998010**	0.965220	**0.952857**	0.933143	2/5
RRFE	20%	5	0.989082	**0.997936**	0.965800	**0.953195**	0.933543	2/5
RRFE	30%	1	0.987286	**0.997606**	0.963800	**0.952961**	0.929409	2/5
RRFE	30%	2	0.987878	**0.997840**	0.965020	**0.954514**	0.931060	2/5
RRFE	30%	3	0.988490	**0.998078**	0.964460	**0.953675**	0.931925	2/5
RRFE	30%	4	0.988571	**0.998143**	0.965340	**0.953603**	0.932651	2/5
RRFE	30%	5	0.989010	**0.998009**	0.965820	**0.953501**	0.932916	2/5
RRFE	40%	1	0.986755	**0.997798**	0.961600	**0.954071**	0.928116	2/5
RRFE	40%	2	0.987112	**0.998056**	0.963260	**0.954691**	0.931015	2/5
RRFE	40%	3	0.988102	**0.998054**	0.963640	**0.954058**	0.931425	2/5
RRFE	40%	4	0.987837	**0.997969**	0.963980	**0.953485**	0.932480	2/5
RRFE	40%	5	0.988643	**0.998205**	0.963820	**0.953193**	0.932624	2/5
RRFE	50%	1	0.985980	**0.998013**	0.960780	<u>**0.956217**</u>	0.926246	2/5
RRFE	50%	2	0.987031	**0.997948**	0.962860	**0.955097**	0.930105	2/5
RRFE	50%	3	0.987520	**0.998171**	0.962700	**0.954590**	0.931200	2/5
RRFE	50%	4	0.987796	**0.998085**	0.963280	**0.953427**	0.931968	2/5
RRFE	50%	5	0.988224	<u>**0.998215**</u>	0.963200	**0.953847**	0.933053	2/5
RRFB	10%	0	0.987163	0.996795	0.965540	0.948493	0.927493	0/5
RRFB	20%	0	0.986347	0.996045	0.961920	0.945333	0.921015	0/5
RRFB	30%	0	0.984398	0.995143	0.959580	0.941416	0.915219	0/5
RRFB	40%	0	0.981878	0.993767	0.956060	0.935001	0.909144	0/5
RRFB	50%	0	0.978398	0.992442	0.954000	0.930389	0.900593	0/5

Table 16: AUC comparison of RF, RRFB and RRFE in medium-dimensional datasets with m set as square root of M and MIS_{pct} ranging from 1% to 5% by 1%

	MIS_{pct}	k	CTG	Dermatology	Ionosphere	Musk	Steel	IMP
RF	0%	0	0.989428	0.997451	0.968297	0.951957	0.934287	ref
RRFE	1%	3	**0.989429**	**0.997593**	**0.968960**	**0.952302**	**0.934430**	5/5
RRFE	1%	2	0.988939	**0.997579**	**0.968540**	**0.952795**	**0.934408**	4/5
RRFE	1%	5	**0.989561**	**0.997474**	**0.968380**	**0.952539**	0.933488	4/5
RRFE	4%	3	0.989112	**0.997530**	**0.968320**	**0.952081**	**0.934341**	4/5
RRFE	1%	4	0.989357	**0.997592**	**0.969600**	0.951869	**0.934674**	3/5
RRFE	2%	3	0.989071	**0.997569**	**0.968820**	0.951578	**0.934355**	3/5
RRFE	1%	1	0.989112	**0.997538**	**0.968500**	0.951307	0.933825	2/5
RRFE	2%	2	0.989092	**0.997611**	0.967940	**0.952904**	0.933814	2/5
RRFE	4%	2	0.988847	**0.997551**	0.967320	**0.951981**	0.934110	2/5
RRFE	4%	4	0.989357	**0.997583**	0.967760	**0.952549**	0.933985	2/5
RRFE	4%	5	0.989163	**0.997485**	0.967300	**0.952956**	0.933921	2/5
RRFE	5%	2	0.988388	**0.997561**	0.967100	**0.952620**	0.932889	2/5
RRFE	5%	5	0.989367	**0.997625**	0.967420	**0.952066**	0.933624	2/5
RRFB	1%	0	0.989153	**0.997453**	0.968160	**0.952740**	0.933570	2/5
RRFE	2%	1	0.989347	**0.997656**	0.967280	0.951320	0.933553	1/5
RRFE	2%	5	0.989265	**0.997589**	0.968240	0.951833	0.933435	1/5
RRFE	3%	1	0.988929	**0.997570**	0.967480	0.951650	0.933461	1/5
RRFE	3%	2	0.989092	**0.997540**	0.968260	0.951701	0.933241	1/5
RRFE	3%	3	0.989010	**0.997623**	0.967520	0.950730	0.934000	1/5
RRFE	3%	4	0.989255	0.997283	0.967960	**0.953325**	0.934218	1/5
RRFE	3%	5	0.989224	0.997325	0.968060	**0.952631**	0.934223	1/5
RRFE	5%	1	0.988592	0.997211	0.966600	**0.952098**	0.932668	1/5
RRFE	5%	3	0.988959	0.997346	0.967460	**0.952870**	0.933931	1/5
RRFE	5%	4	0.989122	0.997369	0.967660	**0.952974**	0.933786	1/5
RRFE	2%	4	0.989071	0.997315	0.968280	0.950935	0.933914	0/5
RRFE	4%	1	0.988663	0.997412	0.968000	0.951827	0.932825	0/5
RRFB	2%	0	0.989041	0.997440	0.967480	0.951356	0.931986	0/5
RRFB	3%	0	0.988714	0.997260	0.966800	0.951719	0.931999	0/5
RRFB	4%	0	0.988449	0.997090	0.966520	0.950571	0.930853	0/5
RRFB	5%	0	0.988214	0.997037	0.966660	0.949947	0.931218	0/5

Table 17: AUC comparison of RF, RRFB and RRFE in medium-dimensional datasets with m set as twice the square root of M and MIS_{pct} ranging from 10% to 50% by 10%

	MIS_{pct}	k	CTG	Dermatology	Ionosphere	Musk	Steel	IMP
RF	0%	0	0.989067	0.996524	0.968378	0.947438	0.932779	ref
RRFE	10%	3	**0.989112**	**0.996914**	**0.968600**	**0.949233**	**0.933065**	5/5
RRFE	10%	5	**0.989194**	**0.997107**	0.967440	**0.948641**	**0.933479**	4/5
RRFE	20%	3	**0.989092**	**0.997195**	0.966460	**0.949929**	**0.934124**	4/5
RRFE	20%	4	**0.989071**	**0.997394**	0.966080	**0.950618**	**0.933926**	4/5
RRFE	20%	5	**0.989071**	**0.997490**	0.965560	**0.949239**	**0.934519**	4/5
RRFE	30%	2	**0.989633**	**0.997612**	0.964380	**0.951975**	**0.933181**	4/5
RRFE	30%	3	**0.989173**	**0.997547**	0.964980	**0.950587**	**0.934191**	4/5
RRFE	30%	4	**0.989469**	**0.997576**	0.964620	**0.950490**	**0.934278**	4/5
RRFE	30%	5	**0.989480**	**0.997694**	0.965100	**0.950713**	**0.934959**	4/5
RRFE	40%	2	**0.989490**	**0.997878**	0.963100	**0.953425**	**0.933649**	4/5
RRFE	40%	3	**0.989602**	**0.997867**	0.964260	**0.953308**	**0.933366**	4/5
RRFE	40%	4	**0.989388**	**0.997811**	0.964140	**0.952571**	**0.934750**	4/5
RRFE	40%	5	**0.989214**	**0.997639**	0.964040	**0.951347**	**0.934479**	4/5
RRFE	50%	2	**0.989194**	**0.997976**	0.961980	**0.953834**	**0.933198**	4/5
RRFE	50%	3	**0.989582**	**0.997870**	0.963320	**0.953895**	**0.933325**	4/5
RRFE	50%	4	**0.989643**	**0.997794**	0.963060	**0.952713**	**0.933949**	4/5
RRFE	50%	5	**0.989633**	**0.997770**	0.963620	**0.952962**	**0.933923**	4/5
RRFE	10%	4	0.988888	**0.997063**	0.968240	**0.949402**	**0.933733**	3/5
RRFE	20%	2	**0.989408**	**0.997063**	0.966880	**0.951096**	0.932718	3/5
RRFE	30%	1	**0.989092**	**0.997539**	0.963980	**0.951634**	0.931064	3/5
RRFE	50%	1	**0.989092**	**0.998044**	0.961160	**0.954487**	0.930680	3/5
RRFE	10%	1	0.989000	**0.996669**	0.967180	**0.948287**	0.931045	2/5
RRFE	10%	2	0.988857	**0.996871**	0.967560	**0.948873**	0.932236	2/5
RRFE	20%	1	0.988684	**0.997074**	0.965860	**0.950436**	0.931835	2/5
RRFE	40%	1	0.989010	**0.998021**	0.961960	**0.952230**	0.931045	2/5
RRFB	10%	0	0.988051	0.995973	0.966640	0.946589	0.926559	0/5
RRFB	20%	0	0.988000	0.995060	0.963760	0.942250	0.920983	0/5
RRFB	30%	0	0.987276	0.994447	0.961060	0.941287	0.915086	0/5
RRFB	40%	0	0.985398	0.993350	0.957200	0.935024	0.909379	0/5
RRFB	50%	0	0.983235	0.991918	0.953580	0.926977	0.903255	0/5

Table 18: AUC comparison of RF, RRFB and RRFE in medium-dimensional datasets with m set as twice the square root of M and MIS_{pct} ranging from 1% to 5% by 1%

	MIS_{pct}	k	CTG	Dermatology	Ionosphere	Musk	Steel	IMP
RF	0%	0	0.989067	0.996524	0.968378	0.947438	0.932779	ref
RRFE	2%	2	**0.989082**	**0.996551**	**0.968420**	**0.948823**	**0.932853**	5/5
RRFE	2%	3	**0.989306**	**0.996604**	**0.968740**	**0.947963**	**0.933184**	5/5
RRFE	3%	4	**0.989122**	**0.996699**	**0.968520**	**0.947950**	**0.932809**	5/5
RRFE	4%	5	**0.989184**	**0.996678**	**0.968420**	**0.948311**	**0.932808**	5/5
RRFE	2%	4	**0.989102**	**0.996646**	0.967620	**0.948337**	**0.933068**	4/5
RRFE	2%	5	**0.989194**	**0.996625**	0.968060	**0.948317**	**0.933424**	4/5
RRFE	4%	3	0.988745	**0.996686**	**0.968680**	**0.948551**	**0.933116**	4/5
RRFE	5%	2	**0.989071**	**0.996751**	0.967980	**0.948422**	**0.932976**	4/5
RRFE	5%	3	**0.989194**	**0.996755**	0.967880	**0.948343**	**0.932899**	4/5
RRFE	5%	4	**0.989102**	**0.996699**	**0.968880**	**0.948885**	0.932475	4/5
RRFE	5%	5	0.988347	**0.996934**	**0.968800**	**0.947939**	**0.932911**	4/5
RRFE	1%	1	0.988888	**0.996550**	**0.969540**	0.947287	**0.932883**	3/5
RRFE	1%	2	0.988602	**0.996604**	0.968160	**0.948529**	**0.933483**	3/5
RRFE	1%	5	**0.989500**	**0.996592**	0.968340	0.947319	**0.933699**	3/5
RRFE	3%	2	0.988867	**0.996647**	**0.968480**	**0.948792**	0.932713	3/5
RRFE	3%	3	0.988520	**0.996740**	**0.968660**	**0.948151**	0.932644	3/5
RRFE	3%	5	0.989041	**0.996841**	0.968160	**0.947555**	**0.932785**	3/5
RRFE	4%	2	0.988388	**0.996529**	0.967940	**0.949391**	**0.933244**	3/5
RRFE	1%	4	**0.989296**	0.996509	0.967480	**0.947613**	0.932316	2/5
RRFE	2%	1	0.988867	**0.996656**	0.968340	**0.948376**	0.932394	2/5
RRFE	4%	1	0.988694	**0.996667**	0.967900	**0.948239**	0.931724	2/5
RRFE	4%	4	0.988694	**0.996699**	0.968120	**0.947834**	0.932549	2/5
RRFE	5%	1	0.988561	**0.996658**	0.967800	**0.949153**	0.932031	2/5
RRFE	1%	3	0.988990	0.996497	0.968300	**0.948246**	0.932651	1/5
RRFE	3%	1	0.988612	**0.996614**	0.968300	0.946953	0.932141	1/5
RRFB	1%	0	0.988949	0.996334	**0.968940**	0.947342	0.932594	1/5
RRFB	3%	0	0.988592	**0.996531**	0.967880	0.946768	0.930345	1/5
RRFB	4%	0	0.988878	0.996486	0.968180	**0.947595**	0.930599	1/5
RRFB	2%	0	0.988786	0.996485	0.967920	0.947433	0.931244	0/5
RRFB	5%	0	0.988367	0.996314	0.967540	0.945920	0.929269	0/5

Table 19: AUC comparison of RF, RRFB and RRFE in medium-dimensional datasets with m set as half of the square root of M and MIS_{pct} ranging from 10% to 50% by 10%

	MIS_{pct}	k	CTG	Dermatology	Ionosphere	Musk	Steel	IMP
RF	0%	0	0.987950	0.997306	0.966285	0.953610	0.932301	ref
RRFE	10%	5	0.987673	**0.997330**	0.965140	**0.953844**	0.931671	2/5
RRFE	20%	3	0.986480	**0.997434**	0.964400	**0.954533**	0.929580	2/5
RRFE	20%	4	0.986398	**0.997596**	0.964040	**0.954224**	0.929998	2/5
RRFE	20%	5	0.986755	**0.997383**	0.964740	**0.954697**	0.930821	2/5
RRFE	30%	2	0.985051	**0.997584**	0.963300	**0.955403**	0.928018	2/5
RRFE	30%	3	0.986061	**0.997478**	0.963860	**0.954883**	0.927979	2/5
RRFE	30%	4	0.986122	**0.997658**	0.963300	**0.954272**	0.929188	2/5
RRFE	30%	5	0.986235	**0.997467**	0.963900	**0.954251**	0.929729	2/5
RRFE	40%	2	0.984490	**0.997524**	0.962060	**0.955783**	0.927004	2/5
RRFE	40%	3	0.985306	**0.997640**	0.962920	**0.955058**	0.926944	2/5
RRFE	40%	4	0.985306	**0.997573**	0.962660	**0.954556**	0.928336	2/5
RRFE	40%	5	0.985806	**0.997649**	0.963360	**0.954806**	0.928724	2/5
RRFE	50%	2	0.983949	**0.997616**	0.961640	**0.955086**	0.924815	2/5
RRFE	50%	3	0.984561	**0.997711**	0.962480	**0.955042**	0.926361	2/5
RRFE	50%	4	0.984867	**0.997668**	0.962940	**0.955004**	0.926934	2/5
RRFE	50%	5	0.985786	**0.997607**	0.963620	**0.954924**	0.927325	2/5
RRFE	10%	1	0.986847	**0.997369**	0.964340	0.953257	0.929788	1/5
RRFE	10%	2	0.986745	0.997276	0.965140	**0.954885**	0.931254	1/5
RRFE	10%	3	0.986898	**0.997434**	0.965340	0.953484	0.930099	1/5
RRFE	10%	4	0.987224	0.997294	0.964960	**0.953801**	0.931240	1/5
RRFE	20%	2	0.986051	0.997253	0.964520	**0.954820**	0.928846	1/5
RRFE	30%	1	0.984633	0.997198	0.961620	**0.955168**	0.925164	1/5
RRFE	50%	1	0.981194	0.996964	0.960500	**0.955362**	0.920734	1/5
RRFE	20%	1	0.985582	0.997006	0.963580	0.952751	0.927519	0/5
RRFE	40%	1	0.983408	0.997097	0.961320	0.953080	0.922604	0/5
RRFB	10%	0	0.985704	0.996281	0.963840	0.950643	0.926774	0/5
RRFB	20%	0	0.983327	0.995211	0.961240	0.945740	0.921079	0/5
RRFB	30%	0	0.980357	0.994017	0.958280	0.941863	0.914679	0/5
RRFB	40%	0	0.977265	0.992506	0.955360	0.935903	0.908853	0/5
RRFB	50%	0	0.972735	0.989281	0.952000	0.930563	0.898846	0/5

Table 20: AUC comparison of RF, RRFB and RRFE in medium-dimensional datasets with m set as half of the square root of M and MIS_{pct} ranging from 1% to 5% by 1%

	MIS_{pct}	k	CTG	Dermatology	Ionosphere	Musk	Steel	IMP
RF	0%	0	0.987950	0.997306	0.966285	0.953610	0.932301	ref
RRFE	1%	4	0.987684	**0.997314**	**0.966300**	**0.953699**	**0.932773**	4/5
RRFE	1%	5	0.987582	**0.997488**	**0.966460**	0.952695	**0.932699**	3/5
RRFE	5%	5	0.987265	**0.997478**	**0.966440**	**0.953811**	0.931568	3/5
RRFE	1%	2	0.987622	0.997299	**0.966340**	**0.953829**	0.931540	2/5
RRFE	2%	3	0.987418	**0.997392**	0.965900	**0.954412**	0.931700	2/5
RRFE	2%	4	0.987408	**0.997391**	0.966000	**0.954070**	0.932273	2/5
RRFE	3%	5	0.987755	**0.997399**	0.965980	**0.954162**	0.931923	2/5
RRFE	4%	2	0.987602	**0.997369**	0.966000	**0.954213**	0.931441	2/5
RRFE	4%	3	0.987265	**0.997376**	0.965360	**0.953868**	0.931805	2/5
RRFE	5%	3	0.987510	**0.997422**	0.965800	**0.954169**	0.931936	2/5
RRFE	1%	1	0.987765	0.997220	**0.966380**	0.953298	0.931860	1/5
RRFE	1%	3	0.987888	0.997230	0.965760	0.953537	**0.932913**	1/5
RRFE	2%	1	0.987276	0.997232	**0.966780**	0.952449	0.931198	1/5
RRFE	3%	1	0.987214	**0.997337**	0.966160	0.953450	0.931143	1/5
RRFE	3%	2	0.987286	0.997275	**0.966340**	0.953433	0.932023	1/5
RRFE	3%	3	0.987571	**0.997320**	0.965840	0.953386	0.931665	1/5
RRFE	3%	4	0.987408	0.997187	0.965760	**0.954236**	0.931440	1/5
RRFE	4%	4	0.987663	0.997134	**0.967340**	0.953112	0.931578	1/5
RRFE	4%	5	0.987602	**0.997474**	0.966100	0.953130	0.931720	1/5
RRFE	5%	2	0.986765	**0.997499**	0.965900	0.953342	0.931108	1/5
RRFE	2%	2	0.987357	0.997231	0.965960	0.952663	0.931930	0/5
RRFE	2%	5	0.987633	0.997125	0.966200	0.952825	0.931739	0/5
RRFE	4%	1	0.987143	0.997178	0.964840	0.953110	0.930961	0/5
RRFE	5%	1	0.987082	0.997283	0.965700	0.952936	0.931010	0/5
RRFE	5%	4	0.987531	0.997126	0.965460	0.953038	0.931328	0/5
RRFB	1%	0	0.987827	0.997178	0.966140	0.953175	0.931283	0/5
RRFB	2%	0	0.987388	0.997078	0.965380	0.953516	0.930151	0/5
RRFB	3%	0	0.987102	0.996996	0.964500	0.952944	0.930108	0/5
RRFB	4%	0	0.987010	0.997082	0.965080	0.952648	0.929175	0/5
RRFB	5%	0	0.987071	0.996869	0.965020	0.952643	0.929400	0/5

Table 21: Summary of the best AUC performance among RF, RRFB and RRFE in five medium-dimensional datasets

Dataset	m	MIS_{pct}	k	Algorithm	AUC	AUC (RF)	IMP%
CTG	$\lfloor 2\sqrt{M} \rfloor$	50%	4	RRFE	0.989643	0.989067	0.06%
dermatology	$\lfloor \sqrt{M} \rfloor$	50%	5	RRFE	0.998215	0.997451	0.08%
ionosphere	$\lfloor \sqrt{M} \rfloor$	1%	4	RRFE	0.969600	0.968297	0.13%
musk	$\lfloor \sqrt{M} \rfloor$	50%	1	RRFE	0.956217	0.951957	0.45%
steel	$\lfloor 2\sqrt{M} \rfloor$	30%	5	RRFE	0.934959	0.932779	0.23%

The best AUC performances of RRFE and RRFB in all five datasets are listed in Table 21 together with the AUC performances provided by the original RF. The best performances among RF, RRFB and RRFE are always provided by RRFE rather than RRFB. Three out of five algorithms use the default m value of $\lfloor \sqrt{M} \rfloor$. Four out of five algorithms use $MIS_{pct} = 50\%$. Four out of five algorithms used k values at either 4 or 5. The percentage of improvement in AUC varies from 0.06% to 0.45%.

We will use one of the 10-fold cross-validations within the steel dataset at $MIS_{pct} = 30\%$ to demonstrate how RRFE achieves improvement over RF while RRFB does not. The average strength of individual classification trees, the average correlation among individual classification trees, the accuracy and AUC as well as Breiman's generalization error bound are together listed for RF, RRFB and RRFE algorithms in steel dataset in Table 22 and Figure 26. The algorithm with the lowest accuracy and AUC is RRFB, which also has the lowest average strength, the lowest average correlation as well as the highest Breiman's error bound. For RRFE with k=5, it does not have the lowest correlation among all five RRFE algorithms, but it has the highest average strength among all five RRFE algorithms. Among all listed algorithms, RRFE with k=5 has the lowest Breiman's error bound, the highest AUC value as well as the highest accuracy.

Table 22: AUC vs. Breiman's error bound in the steel dataset for RF, RRFB and RRFE

Algorithm	Strength	Correlation	Accuracy	AUC	Breiman's error bound
RF	0.495150	0.208246	0.830000	0.935425	0.641137
RRFB	0.376300	0.161005	0.817500	0.908263	0.976025
RRFE (k=1)	0.441225	0.166230	0.852500	0.935338	0.687635
RRFE (k=2)	0.463240	0.175527	0.850000	0.939225	0.642431
RRFE (k=3)	0.473585	0.180483	0.852500	0.938850	0.624230
RRFE (k=4)	0.479285	0.182693	0.855000	0.939700	0.612613
RRFE (k=5)	0.485735	0.182699	0.862500	0.942038	0.591650

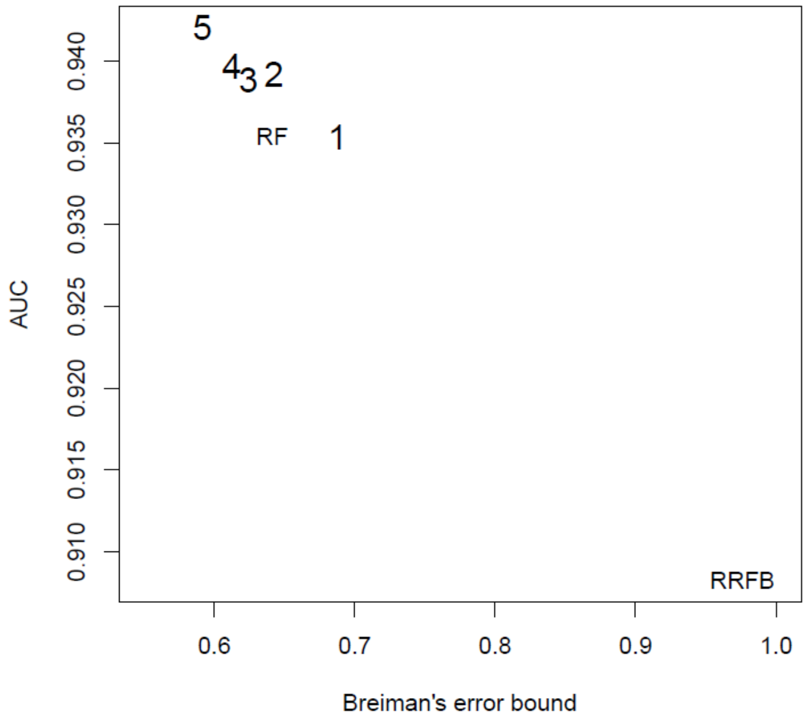


Figure 26: AUC vs. Breiman's error bound in the steel dataset for RF, RRFB and RRFE

4.6 Results in High-dimensional Microarray Datasets

In high-dimensional microarray datasets, variable selection is an important step in data analysis (Lu, et al., 2011; Schumi, et al., 2008). Here, we rank all variables by their variable importance according to random forests for the two microarray datasets. The criterion for assessing variable importance is the mean decrease in Gini impurity criterion. For both microarray datasets, we run the same experiment as listed in 4.4 first with all variables (AUC2000/AUC2905), and then with the best 800 variables (AUC800), the best 400 variables (AUC400), the best 200 variables (AUC200) and the best 100 variables (AUC100).

4.6.1 Alon Dataset

Figure 27 shows the sorted variable importance in the Alon dataset (2000 variables) according to the original random forests (N_{tree} = 2000).

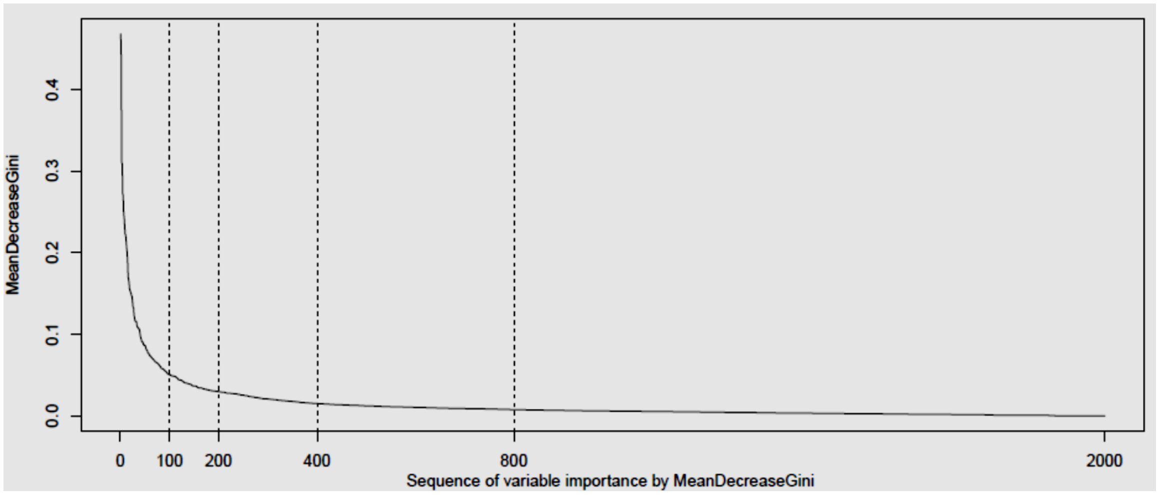


Figure 27: Variable importance in the Alon dataset based on random forests

The variable importance values for the most important 100 variables vary between 0.5 and around 0.05. The least important 1900 variables have variable importance values between 0 and 0.05. The experimental results for AUC2000, AUC800, AUC400, AUC200 and AUC100 are listed in Table 23.

In Table 23, the best AUC performance for RF is underlined in the first row, the best AUC performance for each column is underlined and **bold**. AUC800, AUC400, AUC200 and AUC100 are all much better than the original AUC2000. Both RRFE and RRFB can achieve improvements over the original random forests. The best AUC performance is produced by the best 100 variables (AUC100) using RRFB with MIS_{pct} =30%.

4.6.2 Gravier Dataset

Figure 28 shows the sorted variable importance in the Gravier dataset (2905 variables) according to the original random forests (N_{tree} = 2000).

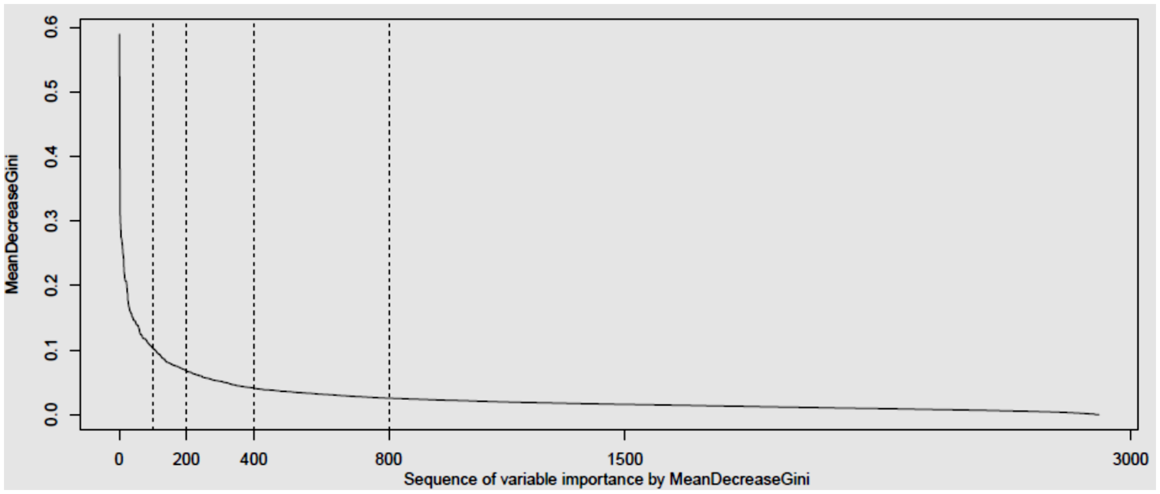


Figure 28: Variable importance in the Gravier dataset based on random forests

The variable importance values for the most important 400 variables vary between 0.6 and around 0.05. The least important 2505 variables have variable importance values between 0 and around 0.05. The experimental results for AUC2905, AUC800, AUC400, AUC200 and AUC100 are listed in Table 24.

In Table 24, the best AUC performance for RF is underlined in the first row, the best AUC performance for each column is underlined and **bold**. AUC800, AUC400, AUC200 and AUC100 are all much better than the original AUC2905. Both RRFE and RRFB can achieve improvements over the original random forests. The best AUC performance is produced by the best 400 variables (AUC400) using RRFB with MIS_{pct} =40%.

Table 23: AUC comparison of RF, RRFB and RRFE with MIS$_{pct}$ ranging from 10% to 50% by 10% in the Alon dataset

	MIS$_{pct}$	k	AUC2000	AUC800	AUC400	AUC200	AUC100
RF	0%	0	0.939151	0.970725	0.972306	0.971016	0.969073
RRFB	10%	0	0.919524	0.977530	0.973006	0.977514	0.978311
RRFB	20%	0	0.911351	0.977951	0.976045	**0.983201**	0.973576
RRFB	30%	0	0.913560	0.971559	0.978850	0.976607	**0.985671**
RRFB	40%	0	0.853735	0.950135	0.976591	0.971762	0.981388
RRFB	50%	0	0.834257	0.950450	0.977045	0.976045	0.981600
RRFE	10%	1	0.942215	0.978600	0.978163	0.979983	0.972225
RRFE	20%	1	0.934596	0.976061	0.978530	0.981327	0.977686
RRFE	30%	1	0.934321	0.972796	0.983671	0.976710	0.978733
RRFE	40%	1	0.934470	0.976186	0.980858	0.979022	0.980421
RRFE	50%	1	0.943894	0.974990	0.980202	0.983092	0.978279
RRFE	10%	2	0.934846	0.967287	0.973061	0.971592	0.970991
RRFE	20%	2	0.944074	0.977694	0.977093	0.971428	0.974514
RRFE	30%	2	0.939471	**0.979311**	0.979093	0.974662	0.972764
RRFE	40%	2	0.941096	0.971521	0.979327	0.978655	0.977951
RRFE	50%	2	0.929339	0.976373	0.982359	0.975733	0.977623
RRFE	10%	3	0.932236	0.975803	0.974296	0.967232	0.972671
RRFE	20%	3	0.947448	0.970248	0.976639	0.977655	0.975241
RRFE	30%	3	0.940035	0.973389	0.984069	0.976030	0.972029
RRFE	40%	3	0.922446	0.969068	0.980546	0.978437	0.976796
RRFE	50%	3	0.933245	0.974295	0.982905	0.976296	0.977623
RRFE	10%	4	**0.954145**	0.977983	0.974968	0.973217	0.972264
RRFE	20%	4	0.931799	0.972389	0.977546	0.973843	0.975749
RRFE	30%	4	0.944980	0.972171	0.984687	0.975530	0.979921
RRFE	40%	4	0.941620	0.970444	0.977257	0.972936	0.977006
RRFE	50%	4	0.930291	0.971482	**0.985015**	0.972357	0.977296
RRFE	10%	5	0.938666	0.970045	0.975623	0.973280	0.973061
RRFE	20%	5	0.947660	0.972389	0.974968	0.979546	0.973498
RRFE	30%	5	0.937112	0.972678	0.979093	0.977123	0.972881
RRFE	40%	5	0.940354	0.975779	0.978530	0.975765	0.977045
RRFE	50%	5	0.947339	0.973662	0.978546	0.977623	0.975154

Table 24: AUC comparison of RF, RRFB and RRFE with MIS_{pct} ranging from 10% to 50% by 10% in the Gravier dataset

	MIS_{pct}	k	AUC2905	AUC800	AUC400	AUC200	AUC100
RF	0%	0	0.889785	0.924021	0.933349	0.922233	0.918333
RRFB	10%	0	0.892035	0.929214	0.936669	0.923356	0.922037
RRFB	20%	0	0.892163	**0.937507**	0.937805	0.920334	0.916377
RRFB	30%	0	0.879934	0.927562	0.940217	0.926042	0.919480
RRFB	40%	0	0.885349	0.933591	**0.945776**	0.924673	0.920810
RRFB	50%	0	0.871786	0.931683	0.940454	0.925417	0.909974
RRFE	10%	1	0.891004	0.925117	0.934732	0.924742	0.916842
RRFE	20%	1	0.893667	0.934079	0.932559	0.924173	0.919757
RRFE	30%	1	0.890101	0.924498	0.941722	0.923756	0.920221
RRFE	40%	1	0.887674	0.929960	0.935077	0.926061	0.914231
RRFE	50%	1	0.888533	0.920026	0.935680	0.924959	0.921122
RRFE	10%	2	0.888433	0.924755	0.940791	0.925698	0.919322
RRFE	20%	2	0.891797	0.930520	0.937528	0.920230	0.918518
RRFE	30%	2	0.892968	0.931070	0.935172	0.920278	0.920037
RRFE	40%	2	**0.895718**	0.925144	0.933396	0.927003	0.922259
RRFE	50%	2	0.895602	0.924993	0.935749	0.923040	0.921105
RRFE	10%	3	0.886558	0.928336	0.932929	0.926710	0.919550
RRFE	20%	3	0.893690	0.930641	0.932620	0.925356	0.919408
RRFE	30%	3	0.892806	0.932040	0.937093	0.922291	0.916572
RRFE	40%	3	0.890163	0.929064	0.935075	0.919683	0.916946
RRFE	50%	3	0.892059	0.925915	0.929957	0.926929	0.916720
RRFE	10%	4	0.888338	0.923128	0.932727	0.925843	0.919248
RRFE	20%	4	0.890978	0.928639	0.932466	0.926026	**0.922285**
RRFE	30%	4	0.889675	0.927168	0.939355	0.921029	0.918905
RRFE	40%	4	0.890158	0.925295	0.936021	0.922657	0.918718
RRFE	50%	4	0.887385	0.922091	0.934694	0.927020	0.917575
RRFE	10%	5	0.891045	0.925559	0.932133	0.924483	0.916788
RRFE	20%	5	0.890246	0.929859	0.936622	0.922387	0.915720
RRFE	30%	5	0.886487	0.927792	0.934548	0.921660	0.919245
RRFE	40%	5	0.893580	0.927530	0.939836	0.927406	0.920229
RRFE	50%	5	0.892611	0.923032	0.931449	**0.927665**	0.921173

4.7 Discussion

When dealing with medium-dimensional datasets, in most cases RRFB does not perform better than RF with regards to AUC, but RRFE takes advantage of the variable importance information provided by RF and RRFE can further improve RF with regards to AUC.

When MIS_{pct} changes from 10% to 50% by 10%, RRFE is more likely to produce the best improvement in an individual medium-dimensional dataset. When MIS_{pct} changes from 1% to 5% by 1%, RRFE is more likely to produce consistent improvements across all five medium-dimensional datasets. However, in both MIS_{pct} ranges, the maximum AUC improvement is below 0.5%. The margin for improvement by RRFE seems relatively small for these five medium-dimensional datasets.

RRFE is also slightly faster than RRFB, as RRFE only imposes missing data in selected variables instead of all variables. For part of the simulation experiments using the steel dataset, we carried it out in R 2.15.1 on a computer with Intel Core i5-3570 CPU @ 3.40 GHz and 4.00G RAM in a 32-bit Windows 7 Enterprise operation system. It took one minute using RF, 16 minutes using RRFB and 13 minutes using RRFE.

Besides the default value of m= $\lfloor\sqrt{M}\rfloor$, here we only experimented with two alternative m values, m= $\lfloor 2\sqrt{M}\rfloor$ and m=$\lfloor\sqrt{M}/2\rfloor$. m=$\lfloor 2\sqrt{M}\rfloor$ produced better improvement than m= $\lfloor\sqrt{M}\rfloor$ in 2 out of 5 datasets. RRFE with other m values might provide further improvements for these medium- dimensional datasets.

When dealing with high-dimensional microarray datasets using random forests, variable selection is an important step. After variable selection, both RRFB and RRFE can achieve AUC performance improvement over the original RF. For these two microarray datasets, RRFB is slightly better than RRFE in terms of AUC improvement after variable selection.

Chapter 5. Conclusion and Future Directions

5.1 Conclusion

This dissertation has introduced the roughened random forests (RRF), a simple and effective method to improve the original random forests (RF) in binary classification. By imposing missing data and then imputing missing data in the originally complete dataset before building each individual classification tree, RRF algorithms are able to unevenly decrease strength and correlation and lead to decreased Breiman's error bound.

We first examined RRFA, an algorithm which imposes missing data in both the training and testing datasets. RRFA uses a quick and easy method, median/mode imputation, to replace the missing data. Experimental results show that RRFA can produce consistent improvements over RF with regards to accuracy and AUC, especially AUC.

However, when the percentage of imposed missing data gets over 50% in the Pima Indians dataset, we find that Breiman's error bound and AUC no longer have a negative linear relationship in RRFA. This problem is solved by RRFB algorithm, which only imposes missing data in the training dataset. RRFB shows further improvement over RRFA with regards to AUC in the Pima Indians dataset. Also, RRFB is much quicker than RRFA in making predictions for new testing datasets.

In addition to RRFA and RRFB which use median/mode imputation, we further investigated RRFC algorithms using seven different imputation methods. These seven

RRFC algorithms are named as RRFC1, RRFC2, RRFC3, RRFC4, RRFC5, RRFC6 and RRFC7. RRFC6 uses multivariate imputation by chained equations (MICE) as the imputation method and RRFC6 produces slightly better overall AUC performances than RRFB, but RRFC6 requires much more computational resources than RRFB. None of the other six RRFC algorithms match RRFB in overall AUC performance.

One of the most important parameter within random forests is the m value, or `mtry` value as in R package `randomForest`. The default value of m usually works well for the original random forests. However, it is not necessarily the best choice for roughened random forests. RRFD tests all possible m values instead of using the default m value. RRFD can provide considerable improvement over RRFB with regards to AUC. However, RRFD is computationally expensive, especially when the dataset's dimension gets higher.

For medium-dimensional datasets with between 20 and 200 covariates, we further proposed RRFE algorithm which selectively imposes missing data in the more important variables. A variable's importance value is based on its average contribution to reducing Gini impurity criterion in the original random forests. RRFE demonstrates better AUC performance than RF in medium-dimensional datasets while RRFB fails to do so.

For high-dimensional microarray datasets with 2000 or more covariates, variable selection is an important step. For the two tested microarray datasets, experimental results demonstrate that RRFB and RRFE can both improve AUC performance over RF after variable selection. RRFB is slightly better than RRFE after variable selection in these two microarray datasets.

The original RF's computation time can be represented as $cN_{tree}N\sqrt{M}\log N$ (Breiman, 2003). The value of the constant c is restricted by the hardware configuration. For RRF algorithms, the additional computation time is just the time used for missing data introduction (T_{mis}) and missing data imputation (T_{imp}) in each classification tree. Therefore, it can be written as $c(N_{tree}N\sqrt{M}\log N + N_{tree}T_{mis} + N_{tree}T_{imp})$. As we impose missing data completely at random in all our RRF algorithms, T_{mis} is usually negligible. T_{imp} depends largely on the selection of imputation methods. For single imputation, such as median/mode imputation used in RRFB, T_{imp} is also negligible. However, for multiple imputation such as MICE used in RRFC6, T_{imp} can get very big for larger datasets with high rates of missing data.

Both RRF and RF build classification trees independently. Therefore, both can improve computational efficiency through parallel computing.

5.2 Future Directions

5.2.1 Expand RRF to Alternative Random Forests

As the RRF algorithms mostly focus on modifying the original dataset instead of the inner workings of random forests, they can be easily extended to alternative random forests. Preliminary results show that RRFB and RRFD can both significantly improve oblique random forests and conditional inference forests in binary classification. We will further test RRF algorithms in oblique random forests and conditional inference forests.

5.2.2 Other Applications

Our current RRF algorithms are all used in binary classification with complete datasets. They will be further applied in binary classification with incomplete datasets.

We tested RRFB and RRFE in high-dimensional microarray datasets with around 2000 to 3000 covariates. We will further test them in microarray datasets with ultra-high dimensions.

As random forests use the same algorithm for all classifications, we will also apply RRF similarly in multiclass classification. Also, we will further extend RRF to regression analysis based on random forests.

Bibliography

Alon, U. et al., 1999. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences,* 96(12), pp. 6745--6750.

Bache, K. & Lichman, M., 2013. *UC Irvine Machine Learning Repository.* [Online] Available at: http://archive.ics.uci.edu/ml/ [Accessed 21 April 2014].

Breiman, L., 1996. Bagging predictors. *Machine learning,* 24(2), pp. 123-140.

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

Breiman, L., 2003. *RF/tools : A class of two-eyed algorithms,* San Francisco: SIAM WORKSHOP.

Breiman, L. & Cutler, A., 2004. *Random forests.* [Online] Available at: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm [Accessed 18 April 2014].

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J., 1984. *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

Caruana, R. & Niculescu-Mizil, A., 2006. *An empirical comparison of supervised learning algorithms.* Pittsburgh, Pennsylvania, The Proceedings of the 23rd International Conference on Machine Learning (ICML2006), pp. 161-168.

Chen, C., Liaw, A. & Breiman, L., 2004. *Using random forest to learn imbalanced data,* Berkeley, CA: University of California at Berkeley, Mathematics Statistics Library.

Chipman, H. A., George, E. I. & McCulloch , R. E., 2010. BART: Bayesian Additive Regression Trees. *Annals of Applied Statistics,* 4(1), p. 266–298.

Cordell, H. J., 2009. Detecting gene–gene interactions that underlie human diseases. *Nature Reviews Genetics,* Volume 10, pp. 392-404.

Cortes, C. & Vapnik, V., 1995. Support-vector networks. *Machine Learning,* Volume 20, pp. 273-297.

Cutler, D. R. et al., 2007. Random forests for classification in ecology. *Ecology,* Volume 88, p. 2783–2792.

Ekstrom, J., 2011. *The Phi-coefficient, the tetrachoric correlation coefficient, and the Pearson-Yule Debate.* [Online]
Available at: http://escholarship.org/uc/item/7qp4604r
[Accessed 18 April 2014].

Elter, M., Schulz-Wendtland, R. & Wittenberg , T., 2007. The prediction of breast cancer biopsy outcomes using two CAD approaches that both emphasize an intelligible decision process. *Medical Physics,* 34(11), pp. 4164-4172.

Freund, Y. & Schapire, R. E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences,* 55(1), p. 119–139.

Friedman, J. H., 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics,* 29(5), pp. 1189-1232.

Gislason, P. O., Benediktsson, J. A. & Sveinsson, J. R., 2006. Random forests for land cover classification. *Pattern Recognition Letters,* 27(4), p. 294–300.

Gravier, E. et al., 2010. A prognostic DNA signature for T1T2 node-negative breast cancer patients. *Genes, Chromosomes and Cancer,* 49(12), pp. 1125-1134.

Hand, D., 2009. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning,* 77(1), pp. 103-123.

Hastie, T., Tibshirani, R. & Friedman, J., 2009. *The elements of statistical learning: data mining, inference, and prediction.* 2nd ed. s.l.:Springer.

Hosmer, D. W. & Lemeshow, S., 2000. *Applied logistic regression.* 2nd ed. s.l.:Wiley-Interscience Publication.

Hothorn, T., Hornik, K. & Zeileis, A., 2006. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics,* 15(3), pp. 651--674.

Ho, T. K., 1995. *Random decision forest.* Montreal, QC, Proceedings of the 3rd International Conference on Document Analysis and Recognition, pp. 278-282.

Kaggle, 2010. *Kaggle competitions.* [Online]
Available at: http://www.kaggle.com/competitions
[Accessed 18 April 2014].

Leisch, F., Weingessel, A. & Hornik, K., 2011. *bindata: generation of artificial binary data.* [Online]
Available at: http://CRAN.R-project.org/package=bindata

Liaw, A. & Wiener, M., 2002. Classification and regression by randomForest. *R News,* 2(3), pp. 18-22.

Little, R. J. & Rubin, D. B., 2002. *Statistical analysis with missing data.* Hoboken, New Jersey: John Wiley & Sons, Inc..

Liu, F. T., Ting, K. M., Yu, Y. & Zhou, Z.-H., 2008. Spectrum of variable-random trees. *Journal of Artificial Intelligence Research,* Volume 32, pp. 355-384.

Lu, T., Liang, H., Li, H. & Wu, H., 2011. High dimensional ODEs coupled with mixed-effects modeling techniques for dynamic gene regulatory network identification. *Journal of the American Statistical Association,* 106(496), pp. 1242-1258.

Mease, D. & Wyner, A., 2008. Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research,* Volume 9, pp. 131-156.

Menze, B. H. et al., 2011. On oblique random forests. *Machine Learning and Knowledge Discovery in Databases · Lecture Notes in Computer Science,* Volume 6912, pp. 453-469.

Palmer, D. S., O'Boyle, N. M., Glen, R. C. & Mitchell, J. B. O., 2007. Random forest models to predict aqueous solubility. *Journal of Chemical Information and Modeling,* 47(1), pp. 150-158.

R Development Core Team, 2012. *R: A Language and Environment for Statistical Computing.* [Online]
Available at: http://www.R-project.org

Ramey, J., 2013. *datamicroarray.* [Online]
Available at: https://github.com/ramhiser/datamicroarray
[Accessed 21 April 2014].

Ripley, B. D., 1996. *Pattern recognition and neural networks.* s.l.:Cambridge University Press.

Ripley, B. et al., 2014. *MASS: Support Functions and Datasets for Venables and Ripley's MASS.* [Online]
Available at: http://cran.r-project.org/web/packages/MASS/index.html
[Accessed 18 April 2014].

Robnik-Sikonja, M., 2004. *Improving random forests.* s.l., ECML 2004 Proceedings, pp. 359-370.

Rodriguez, J. J., Kuncheva, L. I. & Alonso, C. J., 2006. Rotation forest: A new classifier ensemble method.. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 28(10), pp. 1619-30.

Rubin, D. B., 1987. *Multiple imputation for nonresponse in surveys.* New York: Wiley.

Schafer, J. L. & Graham, J. W., 2002. Missing data: our view of the state of the art. *Psychological Methods,* 7(2), pp. 147-177.

Schumi, J., DiRienzo, A. G. & DeGruttola, V., 2008. Testing for associations with missing high-dimensional categorical covariates. *The International Journal of Biostatistics,* 4(1).

Shao, J., 1993. Linear model selection by cross-validation. *Journal of the American Statistical Association,* 88(422), pp. 486-494.

Su, X., Khoshgoftaar, T. M. & Greiner, R., 2009. Making an accurate classifier ensemble by voting on classifications from imputed learning sets. *International Journal of Information and Decision Sciences,* 1(3), pp. 301-22.

Touw, W. G. et al., 2013. Data mining in the life sciences with random forest: a walk in the park or lost in the jungle?. *Briefings in Bioinformatics,* 14(3), pp. 315-326.

Van Buuren, S. & Groothuis-Oudshoorn, K., 2011. mice: Multivariate Imputation by Chained Equations in R.. *Journal of Statistical Software,* 45(3), pp. 1-67.

Wang, L., 2014. *Vanderbilt Biostatistics Wiki.* [Online]
Available at: http://biostat.mc.vanderbilt.edu/wiki/Main/DataSets
[Accessed 30 April 2014].

Weisberg, S., 2011. *alr3: Data to accompany Applied Linear Regression 3rd edition.* [Online]
Available at: http://cran.r-project.org/web/packages/alr3/index.html
[Accessed 18 April 2014].

Wolpert, D. H. & Macready, W. G., 1997. No free lunch theorems for optimization. *EEE Transactions on Evolutionary Computation,* 1(1), pp. 67-82.

Yeh, I.-C., Yang, K.-J. & Ting, T.-M., 2009. Knowledge discovery on RFM model using Bernoulli sequence. *Expert Systems with Applications,* 36(3), p. 5866–5871.

Yucel, R. M., 2011. State of the multiple imputation software. *Journal of Statistical Software,* 45(1).

Appendix : R Functions and Example R Codes with Output

```
##################################################
###############The start of R functions  #############
##################################################

###The following three R packages are requied
library(randomForest)   #Required in all RRF-related functions
library(nnet)     #Required in rrfc5()
library(mice)     #Required in rrfc6()


##### The following eleven functions are based on the dissertation
## rrfa() is the function for RRFA algorithm
## rrfb() is the function for RRFB algorithm
## rrfc1() is the function for RRFC1 algorithm
## rrfc2() is the function for RRFC2 algorithm
## rrfc3() is the function for RRFC3 algorithm
## rrfc4() is the function for RRFC4 algorithm
## rrfc5() is the function for RRFC5 algorithm
## rrfc6() is the function for RRFC6 algorithm
## rrfc7() is the function for RRFC7 algorithm
## rrfd() is the function for RRFD algorithm
## rrfe() is the function for RRFE algorithm


###parameters used in all above functions are listed below
#dat : a dataframe containing both training and testing datasets.
##the last column of 'dat' should be the binary outcome variable (factor)
#yvar : the column number for the binary outcome variable,
##defaulted value of yvar is ncol(dat)
#tr : row numbers of all training data
#te : row numbers of all testing data
#mispct : proportion of missing data, ranging from 0 to 1
#number.trees : number of trees used in roughened random forests

##the following parameter is specifically used in RRFD
#m : the number of covariates selected at each tree node in RRFD
##the following parameter is specifically used in RRFE
#k : the power to be used on each variable's relative importance

#mfix() is the function used in RRFA,RRFB, RRFC1, RRFC2 and RRFC3
#mfix() is based on na.roughfix() function from randomForest package


mfix=function(x,mmmm){
```

```
##x is the name of the dataset with missing data
##mode imputation is always used on categorical variables
##mmmm is used to set imputation method for continuous variables
##mmmm = 1/2/3/4 refers to median/mean/min/max imputation
##when mmmm=1, mfix() is just the same as na.roughfix()

m4m=function(x){
c(median(x,na.rm=T),mean(x,na.rm=T),
min(x,na.rm=T),max(x,na.rm=T))[mmmm]}

mmfix <- function(object, ...)
  UseMethod("mmfix")

mmfix.data.frame <- function(object, ...) {
  isfac <- sapply(object, is.factor)
  isnum <- sapply(object, is.numeric)
  if (any(!(isfac | isnum)))
      stop("mfix only works for numeric or factor")
  roughfix <- function(x) {
      if (any(is.na(x))) {
          if (is.factor(x)) {
              freq <- table(x)
              x[is.na(x)] <- names(freq)[which.max(freq)]
          } else {
              x[is.na(x)] <- m4m(x)
          }
      }
      x
  }
  object[] <- lapply(object, roughfix)
  object
}

mmfix.default <- function(object, ...) {
  if (!is.atomic(object))
    return(object)
  d <- dim(object)
  if (length(d) > 2)
    stop("can't handle objects with more than two dimensions")
  if (all(!is.na(object)))
    return(object)
  if (!is.numeric(object))
    stop("mfix can only deal with numeric data.")
  if (length(d) == 2) {
      hasNA <- which(apply(object, 2, function(x) any(is.na(x))))
      for (j in hasNA)
```

```
        object[is.na(object[, j]), j] <- m4m(object[, j])
  } else {
      object[is.na(object)] <- m4m(object)
  }
  object
}
mmfix(x)
}

################## RRFA #############################
rrfa=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,1)}


fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat,mispct,yvar)
rf=randomForest(pmr[tr,-yvar],pmr[tr,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, pmr[te,-yvar]))-1}
list(pred=fin)}


################## RRFB #############################
rrfb=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,1)}


fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}
```

```
################## RRFC1 ##############################
rrfc1=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,2)}
fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFC2 ##############################
rrfc2=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,3)}

fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFC3 ##############################
rrfc3=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,4)}
```

```
fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFC4 ##############################
rrfc4=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

hotdeckimp=function(dt,mispct,yvar=ncol(dt)){
nr=nrow(dt);nc=ncol(dt);
nd=dt; nm=floor(nr*mispct)
for (z in 1:nc){
if(!(z %in% yvar)) nd[sample(nr,nm,rep=F),z]=nd[sample(nr,nm,rep=F),z]}
nd}

fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=hotdeckimp(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFC5 ##############################
rrfc5=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

library(nnet)
regdat=function(dat){datmis=dat;  nc=ncol(dat)
for (i in 1:nc){
if(is.factor(dat[,i]))  datmis[,i]=predict(multinom(dat[,i] ~ ., data=dat[,-i]))
if(!is.factor(dat[,i])) datmis[,i]=predict(glm(dat[,i] ~ ., data=dat[,-i]))}
datmis}
regdat=regdat(dat[tr,])

regimp=function(dt,mispct,yvar=ncol(dt)){
nr=nrow(dt);nc=ncol(dt);
nd=dt; nm=floor(nr*mispct)
for (z in 1:nc){
smpcs=sample(nr,nm,rep=F)
if(!(z %in% yvar)) nd[smpcs,z]=regdat[smpcs,z]}
nd}

fin=matrix(0,length(te),number.trees)
```

```
for (i in 1:number.trees){
pmr=regimp(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFC6 ##############################
rrfc6=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

library(mice)

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
compdat=complete(mice(nd,m=1,print=F),1)
na.roughfix(compdat)}

fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFC7 ##############################
rrfc7=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
nd}

fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
misdat=rdms(dat[tr,],mispct,yvar)
pmr=rfImpute(misdat[,-yvar],misdat[,yvar])
rf=randomForest(pmr[,-1],pmr[,1],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}
```

```
################## RRFD #############################
rrfd=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees, m){

rdms=function(dt,pct,kpc){
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*pct)
for (z in 1:nc){
if(!(z %in% kpc)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,1)}

fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1,mtry=m)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}


################## RRFE #############################
rrfe=function(dat,yvar=ncol(dat),tr,te,mispct,number.trees,k){

rf=randomForest(dat[tr,-yvar],dat[tr,yvar])
vp=varImpPlot(rf)

rdms=function(dt,pct,kpc,k){
colmis=c(1:length(vp))[rbinom(length(vp),1,(vp/max(vp))^(k))==1]
nr=nrow(dt);nc=ncol(dt)
nd=dt; nm=floor(nr*.01)
for (z in 1:nc){
if((z %in% colmis)) nd[sample(nr,nm,rep=F),z]=NA}
mfix(nd,1)}

fin=matrix(0,length(te),number.trees)
for (i in 1:number.trees){
pmr=rdms(dat[tr,],mispct,yvar,k)
rf=randomForest(pmr[,-yvar],pmr[,yvar],ntree=1)
fin[,i]=as.numeric(predict(rf, dat[te,-yvar]))-1}
list(pred=fin)}



#RFvsRRF() is a wrapper function
#RFvsRRF() is used to compare RF with all above RRF algorithms

RFvsRRF=function(dat,tr,te,yvar,mispct,number.trees){
```

```
rrfres=NULL
rf=randomForest(dat[tr,-yvar],dat[tr,yvar],dat[te,-yvar],ntree=number.trees)
rrfres[1]=colAUC(rf$test$votes[,2],dat[te,yvar])

r=rrfa(dat,yvar,tr,te,mispct,number.trees)
rrfres[2]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfb(dat,yvar,tr,te,mispct,number.trees)
rrfres[3]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc1(dat,yvar,tr,te,mispct,number.trees)
rrfres[4]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc2(dat,yvar,tr,te,mispct,number.trees)
rrfres[5]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc3(dat,yvar,tr,te,mispct,number.trees)
rrfres[6]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc4(dat,yvar,tr,te,mispct,number.trees)
rrfres[7]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc5(dat,yvar,tr,te,mispct,number.trees)
rrfres[8]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc6(dat,yvar,tr,te,mispct,number.trees)
rrfres[9]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

r=rrfc7(dat,yvar,tr,te,mispct,number.trees)
rrfres[10]=colAUC(apply(r$pred,1,mean),dat[te,yvar])

rrfdres=NULL
for (k in 1:(yvar-1)){
r=rrfd(dat,yvar,tr,te,mispct,number.trees,k)
rrfdres[k]=colAUC(apply(r$pred,1,mean),dat[te,yvar])}

r=rrfe(dat,yvar,tr,te,mispct,number.trees,1)
rrfe1=colAUC(apply(r$pred,1,mean),dat[te,yvar])
r=rrfe(dat,yvar,tr,te,mispct,number.trees,2)
rrfe2=colAUC(apply(r$pred,1,mean),dat[te,yvar])
r=rrfe(dat,yvar,tr,te,mispct,number.trees,3)
rrfe3=colAUC(apply(r$pred,1,mean),dat[te,yvar])
r=rrfe(dat,yvar,tr,te,mispct,number.trees,4)
rrfe4=colAUC(apply(r$pred,1,mean),dat[te,yvar])
r=rrfe(dat,yvar,tr,te,mispct,number.trees,5)
rrfe5=colAUC(apply(r$pred,1,mean),dat[te,yvar])
```

```
finres=c(rrfres,rrfdres,rrfe1,rrfe2,rrfe3,rrfe4,rrfe5)
names(finres)=c("RF","RRFA","RRFB","RRFC1","RRFC2","RRF3","RRFC4",
"RRFC5","RRFC6","RRFC7",paste("RRFD_m",1:(yvar-1),sep=""),
"RRFE_k1","RRFE_k2","RRFE_k3","RRFE_k4","RRFE_k5")
finres}

################################################
###############The end of R functions###############
################################################




####################################################
############ Example R codes to test above R functions  ##
####################################################

library(MASS)    #Pima Indians dataset is from this package
library(alr3)         #Blowdown dataset is from this package
library(caTools)  #AUC calculation function  is from this package

###Pima Indians dataset is prepared
dat=rbind(Pima.tr,Pima.te)
number.trees=3
tr=1:200
te=201:532
mispct=0.1
yvar=ncol(dat)
###Pima Indians dataset is compared between RF and RRF
###using 10% of imposed missing data and 3 trees
pima.3trees.auc=RFvsRRF(dat,tr,te,yvar,mispct,number.trees)


###Blowdown dataset is prepared
dat=blowdown[,c(1,2,4,3)]
dat$y=factor(dat$y)
tr=seq(1,3666,2)
te=seq(2,3666,2)
yvar=ncol(dat)
mispct=.1
number.trees=3
###Blowdown dataset is compared between RF and RRF
###using 10% of imposed missing data and 3 trees
```

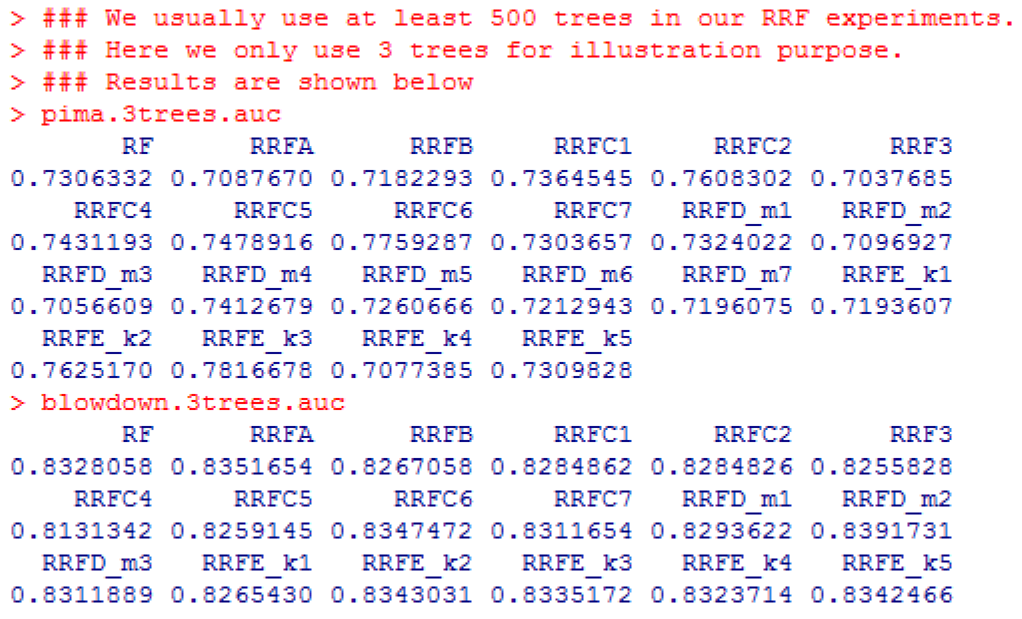
blowdown.3trees.auc=RFvsRRF(dat,tr,te,yvar,mispct,number.trees)

We usually use at least 500 trees in our RRF experiments.
Here we only use 3 trees for illustration purpose.
Results are shown below
pima.3trees.auc
blowdown.3trees.auc

###
################The end of Example R codes ########
###

###
################ R output ########
###

We tested all above example R codes on 4/28/2014 using R 2.15.1, below is the output screenshot in R. The AUC value for each RRF algorithm is listed. For example, RRFD_m1 refers to using RRFD algorithm with m value set as 1. RRFE_k1 refers to using RRFE algorithm with k value set as 1.



```
> ### We usually use at least 500 trees in our RRF experiments.
> ### Here we only use 3 trees for illustration purpose.
> ### Results are shown below
> pima.3trees.auc
       RF      RRFA      RRFB     RRFC1     RRFC2      RRF3 
0.7306332 0.7087670 0.7182293 0.7364545 0.7608302 0.7037685 
    RRFC4     RRFC5     RRFC6     RRFC7   RRFD_m1   RRFD_m2 
0.7431193 0.7478916 0.7759287 0.7303657 0.7324022 0.7096927 
  RRFD_m3   RRFD_m4   RRFD_m5   RRFD_m6   RRFD_m7   RRFE_k1 
0.7056609 0.7412679 0.7260666 0.7212943 0.7196075 0.7193607 
  RRFE_k2   RRFE_k3   RRFE_k4   RRFE_k5 
0.7625170 0.7816678 0.7077385 0.7309828 
> blowdown.3trees.auc
       RF      RRFA      RRFB     RRFC1     RRFC2      RRF3 
0.8328058 0.8351654 0.8267058 0.8284862 0.8284826 0.8255828 
    RRFC4     RRFC5     RRFC6     RRFC7   RRFD_m1   RRFD_m2 
0.8131342 0.8259145 0.8347472 0.8311654 0.8293622 0.8391731 
  RRFD_m3   RRFE_k1   RRFE_k2   RRFE_k3   RRFE_k4   RRFE_k5 
0.8311889 0.8265430 0.8343031 0.8335172 0.8323714 0.8342466 
```