USING STATISTICAL LEARNING TO PREDICT SURVIVAL OF PASSENGERS ON THE RMS TITANIC

by

MICHAEL AARON WHITLEY

B.S., Benedictine University, Lisle, Illinois, 2011
M.S., Kansas State University, Manhattan, Kansas, 2013

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Approved by:

Major Professor
Christopher I. Vahl

Abstract

When exploring data, predictive analytics techniques have proven to be effective. In this report, the efficiency of several predictive analytics methods are explored. During the time of this study, Kaggle.com, a data science competition website, had the predictive modeling competition, "Titanic: Machine Learning from Disaster" available. This competition posed a classification problem to build a predictive model to predict the survival of passengers on the RMS Titanic. The focus of our approach was on applying a traditional classification and regression tree algorithm. The algorithm is greedy and can over fit the training data, which consequently can yield non-optimal prediction accuracy. In efforts to correct such issues with using the classification and regression tree algorithm, we have implemented cost complexity pruning and ensemble methods such as bagging and random forests. However, no improvement was observed here which may be an artifact associated with the Titanic data and may not be representative of those methods' performances. The decision trees and prediction accuracy of each method are presented and compared. Results indicate that the predictors sex/title, fare price, age, and passenger class are the most important variables in predicting survival of the passengers.

Table of Contents

List of Figures iv

List of Tables v

Acknowledgements vi

Chapter 1 - Introduction 1

- Background of Predictive Analytics 1
- Titanic: Machine Learning from Disaster 3
- Objective 4

Chapter 2 - Methods 6

- Regression Trees 9
- Cost Complexity Pruning 11
- Classification Trees 12
- Tree-Based Ensembles 13

Chapter 3 - Results 16

- First Analysis 16
- Second Analysis 27
- Connections between First and Second Analysis 31

Chapter 4 - Conclusion 32

Bibliography 34

Appendix A - First Application of Methods 35

Appendix B - Second Application of Methods 39

List of Figures

Figure 2.1 Example regression tree decision rules. 8
Figure 2.2 Example regression tree predictor space. 8
Figure 2.3 Example regression tree graphic. 8
Figure 3.1 First classification tree summary output from R. 22
Figure 3.2 First classification tree for response variable *Survived*. 23
Figure 3.3 Cross-validation output from R for the first classification tree. 23
Figure 3.4 Cross-validation error plotted against tree size and tuning parameter *k*. 24
Figure 3.5 First pruned classification tree for response variable *Survived*. 24
Figure 3.6 Bagging predictor variable importance values. 25
Figure 3.7 Bagging average decrease in Gini index for each predictor variable. 25
Figure 3.8 Random forest predictor variable importance values. 26
Figure 3.9 Random forest average decrease in Gini index for each predictor variable. 26
Figure 3.10 *Deck* and *Title* variable summaries. 29
Figure 3.11 Regression tree for response variable *Age*. 30
Figure 3.12 Second classification tree for response variable *Survived*. 30

List of Tables

Table 1.1 Kaggle's *Titanic: Machine Learning from Disaster* data 5

Acknowledgements

I would like to thank Kansas State University and the K-State Statistics department for giving me the opportunity to further achieve higher education and this research experience. I would like to acknowledge my advisor, Dr. Christopher Vahl, for his incredible patience, ability to motivate me and his willingness to help me with this research project and other graduate school endeavors. I would also like to acknowledge my committee members, Dr. Gary Gadbury and Dr. Leigh Murray, as they have played a crucial role in my success in the K-State Statistics program. Lastly, I would like to thank and acknowledge my family and friends for their support and belief in my success.

Chapter 1 - Introduction

Background of Predictive Analytics

Since the 1980s, companies have collected copious amounts of customer data to be stored in databases (Finlay, 2014). As the companies collected all of this data they began to think how they could use this data to improve operations or to provide additional benefits. This type of thinking formed "a natural progression toward using the data to improve estimates, forecasts, decisions, and ultimately, efficiency" (Abbott, 2014, p. 3). These databases grew to such an enormous size that in turn became too large for humans to analyze on their own (Finlay, 2014). Predictive analytics was an answer on how to handle such large databases. Predictive analytics is a procedure that incorporates the use of computational methods to determine important and useful patterns in large data. Furthermore, predictive analytics was devised from related fields of study such as, pattern recognition, statistics, machine learning, artificial intelligence, and data mining (Abbott, 2014).

Here are a few points that help to define predictive analytics. Predictive analytics is said to be "data-driven" in the sense that the algorithms used, generate a model from the patterns and characteristics of the data alone. Abbott (2014) states that these models are induced from the data. The "data-driven" algorithms used in predictive analytics may involve the "identification of variables to be included in the model, parameters that define the model, weights or coefficients in the model, or model complexity" (Abbott, 2014, p. 3). It also should be noted that the algorithms used in predictive analytics automatically determine patterns amongst the data. For many algorithms, this automation task will often transform individual variables in the data to be used appropriately when generating models (Abbott, 2014).

Predictive analytics incorporates the use of statistics, but is a subject that requires a different approach and one that has different ideals than statistics. First and foremost, statistics is a field that operates based on a uniquely defined set of rules and a foundation of theory, whereas with predictive analytics, that may not always be the case. For instance, there are algorithms used, e.g. from fields like machine learning and artificial intelligence, within predictive analytics that do not have a best possible solution or such a solution that can even be proved. Furthermore, professionals that use predictive analytics are more lenient when it comes to models and less particular with model parameters. That is, when fitting a predictive model to the data, the focus is on optimizing predictive accuracy of some target. Again, predictive models are "data-driven" or in other words, predictive models are formed from the given data on the basis of being able to make and influence decisions. In general, the methods used to create a predictive model tend to be less rigorous in comparison to many statistical analysis techniques (Abbott, 2014).

The algorithms that are used in predictive analytics can be categorized as either supervised learning methods or unsupervised learning methods. Supervised learning models aim to predict a target variable, represented by a single column in the dataset, by using the other variables or columns in the dataset. Supervised learning is also known as predictive modeling. The most common predictive modeling algorithms are classification when dealing with a categorical target variable or regression in the context of a continuous target variable. Unsupervised learning does not have a target variable, but rather builds a model using clusters of the data. Unsupervised learning models are referred to as descriptive modeling (Abbott, 2014). The next section provides a description of a supervised learning problem involving passengers on the RMS Titanic.

Titanic: Machine Learning from Disaster

If one is seeking an interesting and motivating, but introductory level problem involving statistical learning, predicting survival of the passengers on the RMS Titanic is a great place to start. It appears that this is somewhat of a common problem to work on and as an added benefit, the data set is publicly available. At the time of my study and the writing of this report, Kaggle (www.kaggle.com), the world's largest data science community, had an open predictive modeling competition titled, *Titanic: Machine Learning from Disaster*. This competition became available on Friday, September 8th 2012 and is to conclude on Thursday, December 31st 2015.

The data for the *Titanic: Machine Learning from Disaster* competition was provided by Kaggle and available via the competition dashboard. The Titanic passenger data consists of a training set and a test set, both of which are .csv files. The training set includes the response variable *Survived* and 11 other descriptive variables pertaining to 891 passengers. The test set does not include the response variable, but does contain the 11 other variables for 418 passengers. Note that the 418 passengers in the test dataset are different from the 891 passengers in the training dataset. It is to my knowledge that Kaggle did not specify the details as to how the training and test datasets were chosen. At least, one can assume that individuals were selected at random to form the training and test data sets. A description of the variables that are encountered in the Titanic dataset are given in Table 1.1.

A few additional notes were made on the competition page regarding specific details for some of the variables encountered in the Titanic data. It is first noted that the *Age* variable is measured in years, but can also appear as a fraction if the passenger is less than one year old. Furthermore, one will be able to tell if *Age* was estimated, that is if the age value is followed by .5, e.g. 28.5. There was also further explanation provided for the family relation variables, i.e.

SibSp and *Parch*. It may be of importance to note how these variables were defined and any possible exclusions based on these definitions. *SibSp* is an abbreviation for siblings and spouse. Siblings accounted for by the *SibSp* variable are brothers, sisters, stepbrothers, or stepsisters aboard the Titanic. The typical categorizations of a spouse, i.e. husband or wife, aboard the Titanic are captured by the *SibSp* variable as well. Thus, any fiancés, mistresses, or the like are not included in the *SibSp* variable. The *Parch* variable identifies both parents and children for each passenger aboard the Titanic. Parents are then considered to be either a mother or father. Children can be a son, daughter, stepson, or stepdaughter. Based on these definitions, one can conclude that family relatives such as, cousins, nephews/nieces, uncles/aunts, and in-laws are not captured by the family relation variables. Furthermore, if a child made the voyage with a nanny only, or neighbors, or friends of the family, then the *Parch* variable will be equal to 0.

Objective

The objective of this project is then to build a predictive model to predict which of the passengers survived the ship wreck. In particular, the response variable *Survived* will be modeled given ten possible predictors. The remainder of this report includes background on the methods used to build the predictive model, specifically classification and regression trees, cost complexity pruning, bagging and random forests. A case study based on the RMS Titanic data implementing the methods will be conducted.

Table 1.1 Kaggle's *Titanic: Machine Learning from Disaster* data

Variable Name	Variable Description	Possible Values	Categorical/Numerical
PassengerId	Observation Number	1, 2, …, 1309	Numerical
Survived	Survival	1 = Yes, 0 = No	Categorical
Pclass	Passenger Class	1 = 1st, 2 = 2nd, 3 = 3rd	Categorical
Name	Passenger Name	Braund, Mr. Owen Harris, Heikkinen, Miss. Laina, etc.	Categorical
Sex	Sex of Passenger	Female, Male	Categorical
Age	Age of Passenger	0.17 – 80	Numerical
SibSp	No. of Siblings/Spouses Aboard	0 – 8	Numerical
Parch	No. of Parents/Children Aboard	0 – 9	Numerical
Ticket	Ticket Number	680 – 3101298, A. 2. 39186, WE/P 5735, etc.	Categorical
Fare	Passenger Fare	0 – 512.3292	Numerical
Cabin	Passenger Cabin	A10, B101, C103, D, E12, F2, G6, etc.	Categorical
Embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton	Categorical

Chapter 2 - Methods

This chapter delves into tree-based methods or what are also commonly referred to as tree-based models or decision tree methods for regression and classification settings. The following definitions and terminology in this chapter are adapted from James et al. (2013) and Kuhn and Johnson (2013). Tree-based methods specify a set of conditions or rules that divide up the data. For an observed response, $\boldsymbol{Y_{nx1}}$, the Predictor Space, $\boldsymbol{X_{nxp}} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_p)$, is the set of p different predictors of $\boldsymbol{Y}$. Note that in literature it is common to see predictor variables being referred to as features. In particular, tree-based methods utilize nested if-then statements written in terms of the predictors to slice the predictor space into a number of sub-setting rectangular regions. The term "decision tree" was coined to describe these methods considering that the collection of if-then statements used to divide up the predictor space can be represented using a tree graphic. Tree-based methods are widely used in statistical modeling due to having several advantages. In general, the application of tree-based methods is simple and the results are easy to interpret. As an added benefit, these methods "can effectively handle many types of predictors (sparse, skewed, continuous, categorical, etc.) without the need to pre-process them" and without knowing the relationship that each predictor has with the response (Kuhn and Johnson, 2013, p. 174). Moreover, tree-based methods can accommodate missing data and automatically select the influential predictors for the model.

Now we consider a toy example to assist with learning the terminology and details of decision trees. Suppose we want to build a regression tree modeling a response $\boldsymbol{Y}$ given two numerical predictors $\boldsymbol{P1}$ and $\boldsymbol{P2}$. Also suppose that the data has already been divided into a training set and test set. A regression tree can be built from the training data and used to make predictions of the response $\boldsymbol{Y}$ that is contained in the test data set. This can be achieved using software that contains a package for building decision trees, such as the *tree* package in R.

Say the regression tree is defined by the nested if-then statements given in Figure 2.1. The set of if-then statements divide the $P1$ x $P2$ two-dimensional predictor space into three rectangular regions. These three regions can be expressed as $R_1 = \{X|P1 \geq 6.8\}$, $R_2 = \{X|P1 < 6.8, P2 \ \geq 33\}$, and $R_3 = \{X|P1 < 6.8, P2 < 33\}$. The two-dimensional predictor space containing regions R_1, R_2, and R_3 is displayed in Figure 2.2.

Every decision tree is comprised of internal and terminal nodes. Internal nodes are the points at which a splitting rule is displayed and a corresponding split occurs within a tree. Terminal nodes or leaves are the regions that divide up the predictor space. For the tree in this example and the trees that will follow in this report, terminal nodes are identified by a single constant value. Branches are the lines that connect the internal and terminal nodes of a tree.

Together, the if-then statements and corresponding partitions of the predictor space make up the regression tree shown in Figure 2.3. The regression tree in this example yields predictions of the response variable depending on which terminal node the corresponding test observations belong to. Note that the predicted outcomes are determined by taking the average of the training observations in each region. Predictions of the response variable are made as follows. Test observations belonging to the region $R_1 = \{X|P1 \geq 6.8\}$ will have a predicted outcome of 3.07. Test observations belonging to the region $R_2 = \{X|P1 < 6.8, P2 \ \geq 33\}$ receive the predicted outcome 4.85. Moreover, test observations captured by the region $R_3 = \{X|P1 < 6.8, P2 < 33\}$ will have a predicted outcome of 2.35.

Figure 2.1 Example regression tree decision rules.

```
if P1 < 6.8 then
   if P2 < 33 then Y_hat = 2.35
   else Y_hat = 4.85
else Y_hat = 3.07
```

Figure 2.2 Example regression tree predictor space.

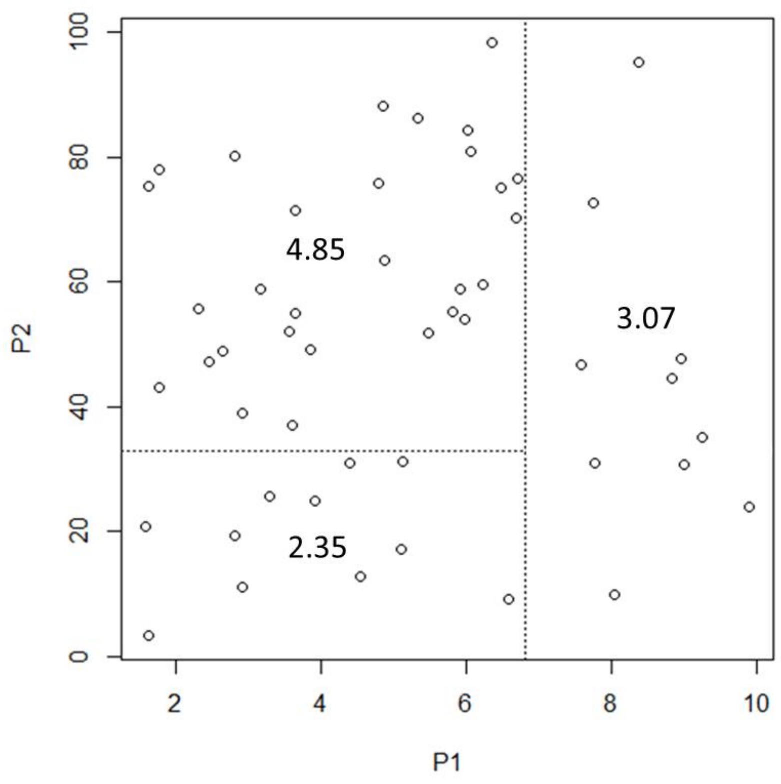
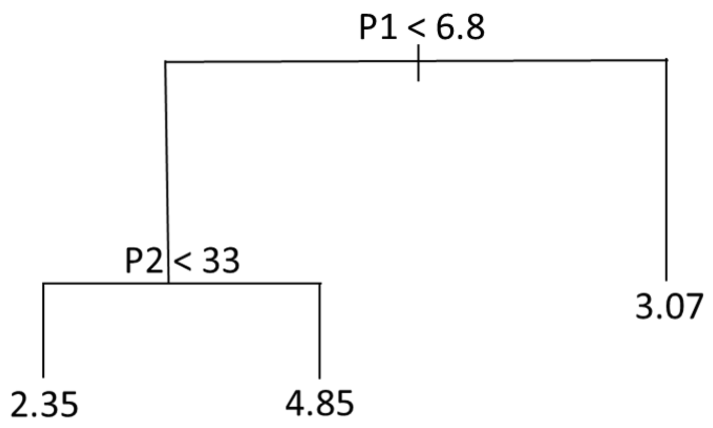


Figure 2.3 Example regression tree graphic.



Regression Trees

Regression trees have a non-categorical and numerical target variable and aim to divide up the data into subsetting rectangles that are homogeneous with respect to the response. In efforts to attain this homogeneity, regression tree algorithms will decide which predictors are important and are to be split, at which value of the predictor the split should occur, how deep the tree should be (i.e. how many layers of internal nodes are needed), how complex the tree should be (i.e. how many branches are needed), and provide a prediction equation for each terminal node. In this study, one of the goals is being able to understand and implement basic regression trees or trees that use constant functions to make predictions of some target from test data. The discussion that will follow on tree-based methods contains the ideology and methodology of Breiman et al. (1984) on classification and regression trees (CART).

When building a regression tree, the algorithm begins with the entire training data set, $\boldsymbol{X}$. The first step is to consider all of the predictors, $\boldsymbol{X_1}, \boldsymbol{X_2}, \dots, \boldsymbol{X_p}$, and all possible values or split points for each of the predictors. In doing this, the objective is to select a predictor $X_j, j = 1, 2, \dots, p$, and a split point s, such that the overall sums of squares error is minimized. That is, when the optimal predictor has been identified and the first split has been made, this results in obtaining two rectangular regions, R_1 and R_2. Thus, for some j and s, the regions $R_1(j, s) = \{X | X_j < s\}$ and $R_2(j, s) = \{X | X_j \geq s\}$ will have been formed to minimize the sums of squares error (SSE) where

$$\text{SSE} = \sum_{i:\, x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2,$$

where $\bar{y}_{R_1}$ and $\bar{y}_{R_2}$ are the means of the training observations in regions R_1 and R_2, respectively. After the first split has been made, the algorithm continues its search for preeminent predictors and corresponding values to split on in order to reduce the SSE. Next, the algorithm works to make another split within either region R_1 or R_2. Then after making the second split, the regression tree will contain three rectangular regions, say $R_1, R_2,$ and R_3. This process continues making additional splits on important predictors as long as there is a reduction in the SSE. The process will conclude when a final stopping criterion has been attained. A reasonable stopping criterion might be that the reduction in SSE for tree with the new split is above a certain (small) threshold or that the size of the resulting regions will be below a pre-specified minimum number. When the algorithm reaches the stopping criterion, the predictor space will have been divided into J disjoint rectangular regions, $R_1, R_2, \ldots, R_J$ and the final regression tree will have been formed. The tree is then used to make predictions of the response given test data. The predicted response corresponding to a given test observation will be the mean of the training observations from the region that captures that test observation. The formula to compute the overall sums of squares error over the J regions is given by

$$\text{SSE} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2,$$

where $\bar{y}_{R_j}$ is the mean of the training observations in the jth region (James et al., 2013; Kuhn and Johnson, 2013).

There are two quantities that are commonly used to assess model accuracy for regression trees. After a regression tree has been formed, the training error rate can be obtained. The training error rate is given by

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

where $\hat{y}_i$ is the predicted mean outcome for the *i*th training observation and $I(y_i \neq \hat{y}_i)$ is an indicator function which takes value one when $y_i \neq \hat{y}_i$ and value zero when $y_i = \hat{y}_i$. Simply put, the training error is the proportion of incorrect predictions of observations in the training data set. After a regression tree is used to make predictions of observations in the test data set, one can obtain the test error rate. The test error rate is similar in construction to the training error rate and gives the proportion of incorrect predictions of the test observations (James et al., 2013).

The algorithm that has been described above utilizes a tactic that is known as recursive binary splitting or recursive partitioning. James et al. (2013, p. 306) state that the algorithm has a "top-down, greedy approach." It is "top-down" in the sense that the starting point is at the top of the tree where the predictor space is whole and grows downward when making splits that divide up the predictor space. Furthermore, the algorithm is "greedy" since only the best possible split is made at each step in the process and it does not consider the overall growth of the tree. Therefore, the algorithm fails to consider splits that may yield a tree with better prediction accuracy. Moreover, the traditional regression tree algorithm that has been explained tends to overfit the training data. This means that the regression tree formed will yield good predictions of the training data, but such desirable prediction accuracy will fail to carry over to the test data. When a tree overfits the training data, this results in obtaining a tree that is large and complex.

Cost Complexity Pruning

A reasonable approach to prevent a regression tree from overfitting the training data is to allow the algorithm to produce a large complex tree and to then invoke pruning of that large tree. Tree pruning aims to reduce the size of the decision tree while selecting a subtree that has the smallest test error rate. We have chosen to use cost complexity pruning, an approach also taken by Breiman et al. (1984). The method first grows a large tree and then generates a "sequence of trees indexed by a nonnegative tuning parameter" k (James et al., 2013, p. 308). K-fold cross-validation is then used to identify the value of k corresponding to the smallest subtree T that minimizes the "penalized error rate,"

$$\mathrm{SSE}_k = SSE + k|T|,$$

where SSE is the overall sums of squares error and $|T|$ is the number of terminal nodes in subtree T (James et al., 2013; Kuhn and Johnson, 2013). Briefly, K-fold cross-validation randomly divides the training dataset into K equally-sized subsets. One subset is used as a validation dataset. The other K-1 subsets are used to fit the model and then a new estimate of test error is computed by averaging the test errors from the K-1 fits of the model.

One can observe that when k = 0, the subtree is equal to the large unpruned tree that was created at the beginning of the pruning process. As k increases, there is a penalty to pay when the subtrees are large. Therefore, the penalized error rate, SSE_k, tends to be minimized for smaller subtrees (James et al., 2013; Kuhn and Johnson, 2013).

Classification Trees

Classification trees are used for categorical target variables. The approach for building classification trees is analogous to the method applied when building regression trees except for a few different properties. The noticeable change is that the terminal nodes in a classification tree will contain one of the possible categories of the response variable. That is, predicted values in a classification tree are determined by selecting the category that has occurred the most amongst the training observations within each region of the predictor space.

There is also an adjustment in the criteria that is required to grow classification trees. Rather than using the SSE to determine the splits that should be made within a tree, the Gini index (*G*) is used. The Gini index measures the total variance across the *L* classes of the response variable and is computed using the formula,

$$G = \sum_{l=1}^{L} \hat{p}_{ml}(1 - \hat{p}_{ml}),$$

where $\hat{p}_{ml}$ is the proportion of training observations in the *m*th rectangular region from the *l*th class of the response variable. The Gini index is more commonly thought of as a numerical representation of node purity, i.e. an indication of the proportion of observations actually having the predicted response in each node. Obtaining a small *G* value indicates that a node primarily contains observations belonging to a single response category (James et al., 2013).

In order to achieve the best prediction accuracy when pruning a tree, the classification error rate needs be used as the criteria that guides the pruning procedure. The classification error rate is defined as the proportion of training observations that belong to a category other than the predicted category in a terminal node. Cost complexity pruning is also the method used when pruning classification trees.

Tree-Based Ensembles

Any single decision tree tends to have high variability. This means that subtle changes to the training data can have drastic effects on the form and prediction of a single decision tree. Bagging is the first method, which we present in this report that is constructed in order to reduce variance. Bagging is a method that relies on bootstrap sampling. A bootstrap sample is created from the training dataset by random sampling with replacement. In order to create multiple training datasets, B independent bootstrap samples are generated and a decision tree is built on each sample where B is a large integer, say B = 500. The procedure deliberately allows for each of the trees to be grown deep and unpruned. Bagging achieves lower variance by averaging the predictions coming from the trees on the B bootstrap samples.

Just as with bagging, random forests "build a number of decision trees on bootstrapped training samples" (James et al., 2013, p. 320). Random forests introduce an element of randomness into the building of decision trees. For every possible split point in a tree, a random subset of *m* predictor variables are considered from the total, *p*, number of predictors. Bagging is achieved when $m = p$, that is all predictors are considered at each possible split in a tree. Thus, if the data contains a single predictor that has a more prominent influence on the response, bagged trees will likely always select this predictor as the first split in each tree. This fact leads to the major disadvantage in bagging which is that it tends to generate a group of trees that are highly correlated.

Random forests mitigate this disadvantage by using $m = \sqrt{p}$. By using a random selection of $\sqrt{p}$ predictors, random forests will allow for the moderately important predictors to be considered when making the first split in a tree. Additionally, random forests will make use of a random selection of $\sqrt{p}$ predictors at subsequent splits in a tree. James et al. (2013, p.320) explain that the random forest procedure de-correlates decision trees and consequently makes the "average of the resulting trees less variable and hence more reliable" than bagged trees or a single decision tree.

When using bagging or random forests, one needs only to identify how many trees should be incorporated into the ensemble, i.e. to specify the value of B. For either method, choosing a larger number of trees will not cause overfitting the training data. It will suffice to use the number of trees at which the test error rate has hit a plateau. Note that bagging and random forests may yield an improvement of prediction accuracy over the prediction accuracy of a single decision tree.

Chapter 3 - Results

The following sections provide an overview of two analyses performed in effort to achieve the goal of predicting survival of the passengers on the RMS Titanic. The first analysis consists of very few changes in the data and incorporates methods such as regression and classification trees, cost complexity pruning, bagging, and random forests. The second analysis replicates the methods used in the first analysis while incorporating feature engineering. In these analyses, we have identified the important elements in the prediction, i.e. predictor variables, split points, etcetera. We also report the prediction accuracy of each method as provided by Kaggle.

First Analysis

In order to generate classification and regression trees in R, the *tree* package was used. The *tree* package utilizes the method of recursive partitioning in order to grow trees and a formula is provided when executing the tree command. Regression trees can be formed if the provided response variable given in the formula is numeric. Similarly, classification trees can be formed if the provided response variable given in the formula is a factor variable. In either case of regression or classification, the right-hand side of the formula needs to contain only numeric and/or factor predictor variables.

Given the Titanic data from Kaggle, we only needed to make minimal changes to the data before building the first classification tree predicting survival of the passengers. The first step was to add the response variable, *Survived*, to the test data set. This was achieved by appending an additional column to the test data containing “NA” in each cell. Adding the survived column to the test data set will allow one to make predictions of survival for each passenger. The training and test data sets were combined for consistency when making the following changes.

We reassigned the predictor variable’s class as either numeric or factor as needed. For example, variables originally classified as integer were reassigned as numeric. The exception to this reassignment was for binary variables, which also were originally classified as integer and needed to be reassigned as factor variables. This initial manipulation of the data provided by Kaggle also included cleaning up the variables: *Embarked, Fare,* and *Age*. There were two missing observations for the *Embarked* variable that were replaced with the most common embark point, “S”. The missing fare value was replaced with the median fare price. Twenty percent of the age observations were missing. Thus, the large number of missing age values presents a more prevalent problem. Initially, we replaced the missing age values with the median age. Later, in the second analysis, we will discuss another approach to manipulating the data that involves building a regression tree to predict the missing age values.

The formula that was used when building the first classification tree with response *Survived* incorporated seven predictor variables: *Pclass, Sex, Age, SibSp, Parch, Fare,* and *Embarked*. Recall that a survived value equal to one is the success, indicating that the passenger survived. From the tree summary given in Figure 3.1, we can see that the classification tree made used five of the seven variables: *Sex*, *Pclass*, *Fare*, *Age*, and *SibSp*. Notice that in Figure 3.1 we can also determine that the classification tree has nine terminal nodes and we are able to view the decision rules that were used to build the tree. The classification tree corresponding to the decision rules is shown in Figure 3.2.

When one wishes to interpret the results produced from a decision tree algorithm, such as the *tree* package in R, one can obtain the desired knowledge by first studying the tree graphic that has been produced. By studying the tree in Figure 3.2 we immediately are drawn to identifying which predictor variables were used in the tree, where in the tree these variables are incorporated, and how many terminal nodes were produced. We know that it is likely that the predictor variables closer to the top of the tree are those that have a more significant influence on the response variable. That is, one can claim that *Sex* is the most important variable in predicting survival of the passengers on the Titanic. The variables *Pclass* and *Fare* appear to be of the next most important variables in predicting survival of the passengers. Since there are nine terminal nodes, we know that the predictor space has been divided into nine rectangular regions. Again, these regions are formed by the rules given at each split point in the tree.

Now we will cover in detail each of the splits made and the resulting branches in the tree given in Figure 3.2. At each split point a rule is provided. Each rule identifies the predictor variable that is being split and the value of that variable where the split occurs. For example, the first decision rule given in the classification tree is "Sex: female." The tree is formatted such that the passengers in the training data are first divided by sex. So all females in the training data will make up the left branch and all males will make up the right branch. This is the format that is used throughout the tree. By this we mean that any training observations that are captured by the split value in the rule are represented by the left branch. In studying the tree we can see that there are three different branches that are part of the left half of the tree. These three branches contain female passengers only and are formed by incorporating the splits: Sex (female), Pclass (3), Fare (< 23.35). The right half of the tree contains six branches. These six branches contain male passengers only and are formed by the following splits: Sex (female), Fare (< 26.2688), Age (13.5), SibSp (< 2.5), Age (< 13.5), and Pclass (2). Furthermore, each branch concludes at a terminal node. Passengers that fall into a terminal node with value equal to one are classified as survivors. Likewise passengers that fall into a terminal node with value zero are classified as non survivors.

When using the *tree* package to build a classification or regression tree, additional useful information is produced and has been displayed in Figure 3.1. The details that are being referred to are those generated from the summary command and those contained in the set of decision rules. By calling the summary command, one also obtains the training error rate (i.e. misclassification error rate) and the residual mean deviance. The set of decision rules not only provides each split made, but also the number of individuals, *n*, that fall into each section after each split, the deviance, and the probability corresponding to the predicted class (i.e. survived or did not survive) of the individuals in each section. James et al. (2013) have indicated that obtaining a small deviance value implies that a tree is fitting the training data well. For completeness, when in the context of a classification tree, the formula for deviance (DEV) is

$$\mathrm{DEV} = -2\sum_{m}\sum_{l} n_{ml} log\hat{p}_{ml},$$

where $n_{ml}$is the number of training observations in the *m*th terminal node from the *l*th class and $\hat{p}_{ml}$ is the proportion of training observations in the *m*th rectangular region from the *l*th class of the response variable. It can be noted that the residual mean deviance is equal to the deviance over the difference of the total number of observations in the training data minus the number of terminal nodes in the pruned tree (James et al., 2013).

However, with all that being said, we would like to remind readers that the purpose of this project was to obtain an introduction to methods such as CART, bagging, and random forests. Thus, our focus was on being able to apply these methods and evaluate them by using the prediction or classification accuracy provided by Kaggle. The prediction accuracy coming from Kaggle was obtained by making a submission that predicts survival of passengers in the test data set. Kaggle then evaluates prediction accuracy on approximately fifty percent of the test data. Kaggle also states that when the competition concludes, final submissions will be evaluated on the remaining fifty percent of the test data. The prediction accuracy for the first classification tree was an astounding 0.79426.

Naturally the next step after generating a tree is to consider pruning that tree. Both of the following functions are also part of the *tree* package in R. We have chosen to use the cv.tree( ) function to perform K-fold cross-validation and the prune.misclass( ) function to prune the starting tree to the optimal size and complexity. Specifying the FUN = prune.misclass option when using the cv.tree( ) function allows for the classification error rate to be used as the criterion that directs the cross-validation and pruning procedure. Together these functions in R perform cost complexity pruning such that an arrangement of trees are considered and result in selecting a subtree that may improve performance (James et al., 2013).

The results from applying cost complexity pruning to our first classification tree are given in Figure 3.3. The output that is generated assists the user in selecting the final size of the decision tree. To do this, one will identify the tree size that has the smallest cross-validation error rate. In the output and for classification trees, the cross-validation error rate is identified by the name “dev” (James et al., 2013). The smallest cross-validation error rate is 161. Therefore, the best pruned tree is the one containing 8 terminal nodes. Plots of the cross-validation error rate as a function of both tree size and tuning parameter, *k*, are shown in Figure 3.4. The pruned classification tree containing 8 terminal nodes is given in Figure 3.5. The difference between the pruned tree and the original tree is that the rule “Pclass: 2” from the original tree has been removed and replaced with a single terminal node. The classification accuracy for the pruned tree remains the same as the unpruned tree, i.e. 0.79426.

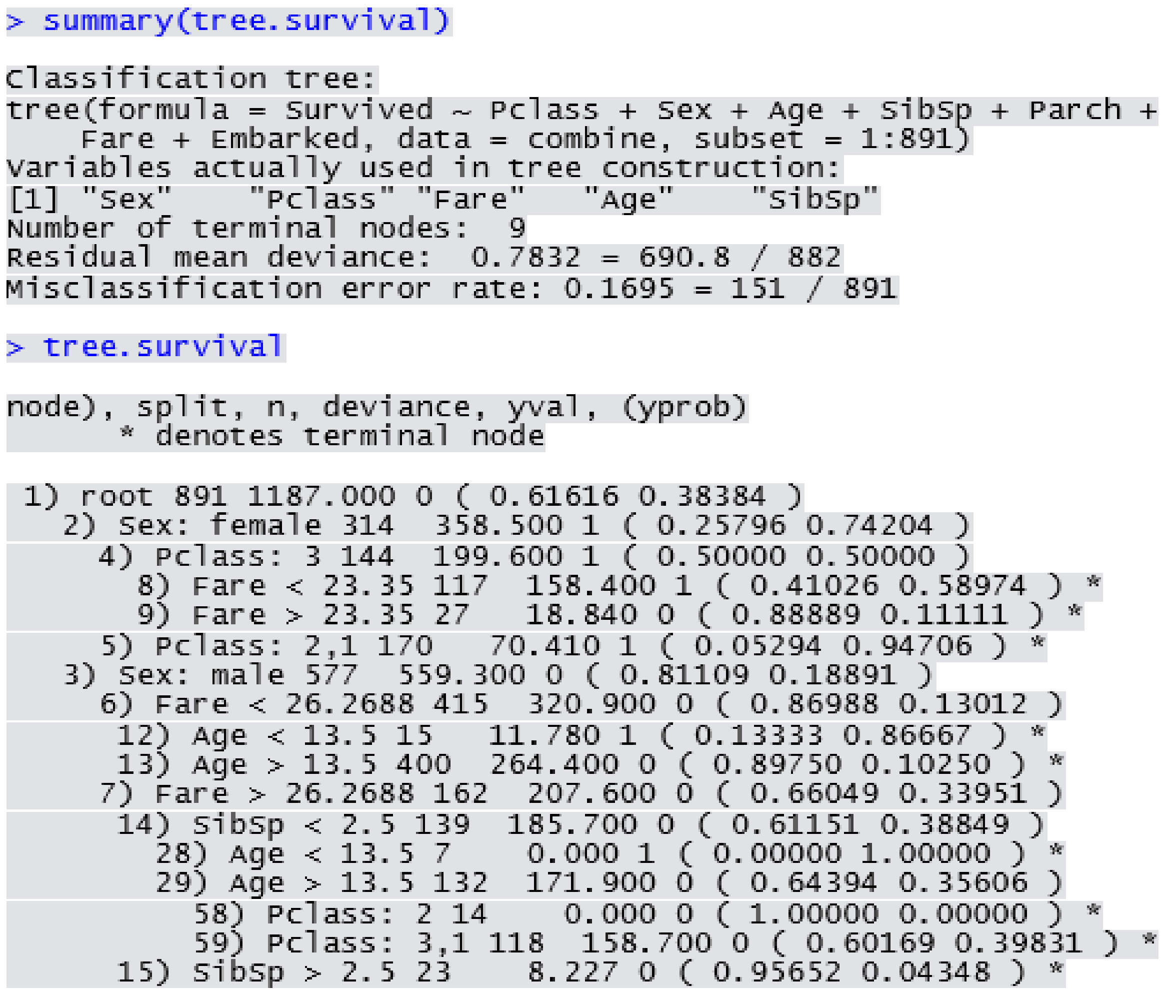
The *randomForest* package in R can be used for both bagging and random forest ensembles. This is achieved by correctly specifying the number of predictors that are to be considered at each split point in a tree by using the option mtry in the randomForest( ) function. By setting mtry = *p*, where *p* is the total number of predictor variables used when building a tree, the method of bagging is performed. A random forest ensemble is performed when the mtry option is left out and the default number of predictors considered for each split is equal to $\sqrt{p}$ (James et al., 2013). In each ensemble method, the default of 500 trees were generated.

Here the same seven predictor variables that were used when building the classification tree are incorporated into the bagging ensemble. These seven predictors are *Pclass*, *Sex*, *Age*, *SibSp*, *Parch*, *Fare*, and *Embarked*. When performing bagging, each of the seven predictors are considered at every split when building an individual tree for each tree in the ensemble. When applying random forests, a random choice of $\sqrt{7} \approx 2$ predictor variables are selected to be considered at each possible split point when building each tree in the ensemble. Note that the results of an ensemble method are too complex to be depicted using a single tree graphic. However, when using the *randomForest* package in R, there is the added bonus of obtaining variable importance measures and plots. The Gini index is used when assessing variable importance for classification trees. Therefore, the variable importance measures reported are the amount that the Gini index is decreased over the splits made on each predictor, averaged across all trees in the ensemble (James et al., 2013).

For the bagging method, the average decrease in Gini index values for each predictor variable is given in Figure 3.6 and are plotted in Figure 3.7. For the random forests method, the average decrease in Gini index values for each predictor variable is given in Figure 3.8 and are plotted in Figure 3.9. In both methods, the predictor variables yielding the largest decrease in

Gini index are *Sex*, *Fare*, *Age*, and *Pclass*, respectively. Kaggle reported a prediction accuracy of 0.76555 for the bagging approach and a prediction accuracy of 0.77033 for random forests. Neither of the ensemble methods were an improvement on prediction accuracy over the single classification tree.

Figure 3.1 First classification tree summary output from R.



```
> summary(tree.survival)

Classification tree:
tree(formula = Survived ~ Pclass + Sex + Age + SibSp + Parch +
    Fare + Embarked, data = combine, subset = 1:891)
Variables actually used in tree construction:
[1] "Sex"    "Pclass" "Fare"   "Age"    "SibSp"
Number of terminal nodes:  9
Residual mean deviance:  0.7832 = 690.8 / 882
Misclassification error rate: 0.1695 = 151 / 891

> tree.survival

node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 891 1187.000 0 ( 0.61616 0.38384 )
   2) Sex: female 314  358.500 1 ( 0.25796 0.74204 )
     4) Pclass: 3 144  199.600 1 ( 0.50000 0.50000 )
       8) Fare < 23.35 117  158.400 1 ( 0.41026 0.58974 ) *
       9) Fare > 23.35 27   18.840 0 ( 0.88889 0.11111 ) *
     5) Pclass: 2,1 170   70.410 1 ( 0.05294 0.94706 ) *
   3) Sex: male 577  559.300 0 ( 0.81109 0.18891 )
     6) Fare < 26.2688 415  320.900 0 ( 0.86988 0.13012 )
      12) Age < 13.5 15   11.780 1 ( 0.13333 0.86667 ) *
      13) Age > 13.5 400  264.400 0 ( 0.89750 0.10250 ) *
     7) Fare > 26.2688 162  207.600 0 ( 0.66049 0.33951 )
      14) SibSp < 2.5 139  185.700 0 ( 0.61151 0.38849 )
        28) Age < 13.5 7     0.000 1 ( 0.00000 1.00000 ) *
        29) Age > 13.5 132  171.900 0 ( 0.64394 0.35606 )
          58) Pclass: 2 14     0.000 0 ( 1.00000 0.00000 ) *
          59) Pclass: 3,1 118  158.700 0 ( 0.60169 0.39831 ) *
      15) SibSp > 2.5 23     8.227 0 ( 0.95652 0.04348 ) *
```

Figure 3.2 First classification tree for response variable *Survived*.

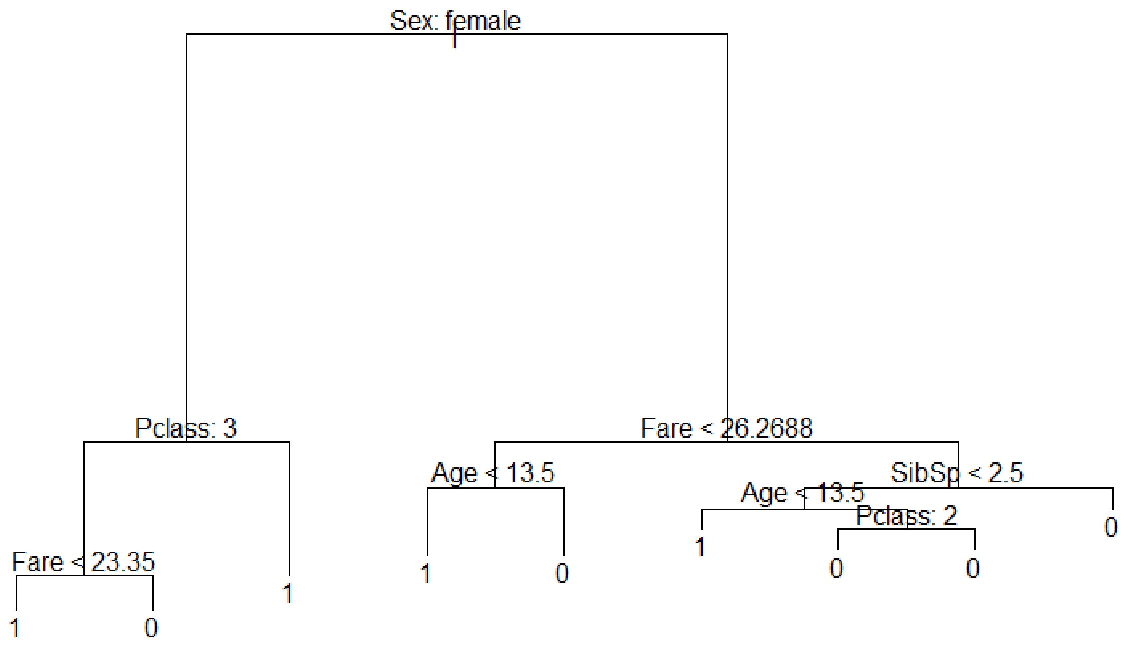
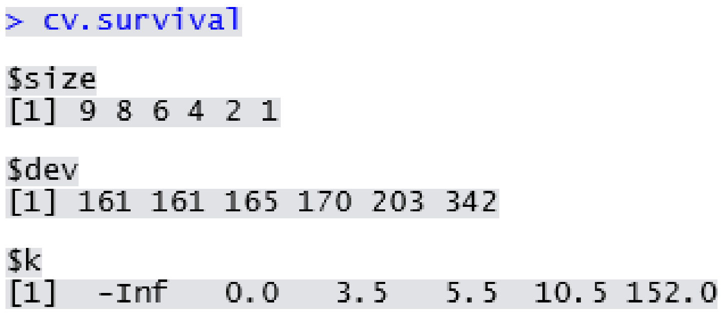


Figure 3.3 Cross-validation output from R for the first classification tree.



```
> cv.survival

$size
[1] 9 8 6 4 2 1

$dev
[1] 161 161 165 170 203 342

$k
[1]  -Inf    0.0    3.5    5.5   10.5 152.0
```

Figure 3.4 Cross-validation error plotted against tree size and tuning parameter *k*.

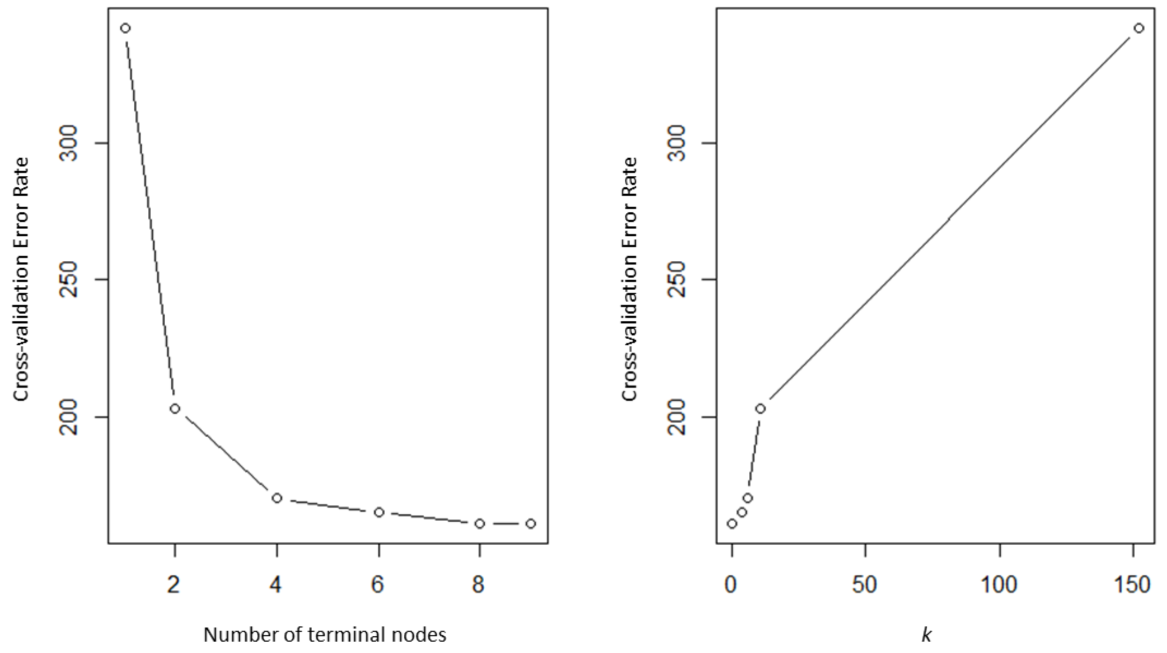


Figure 3.5 First pruned classification tree for response variable *Survived*.

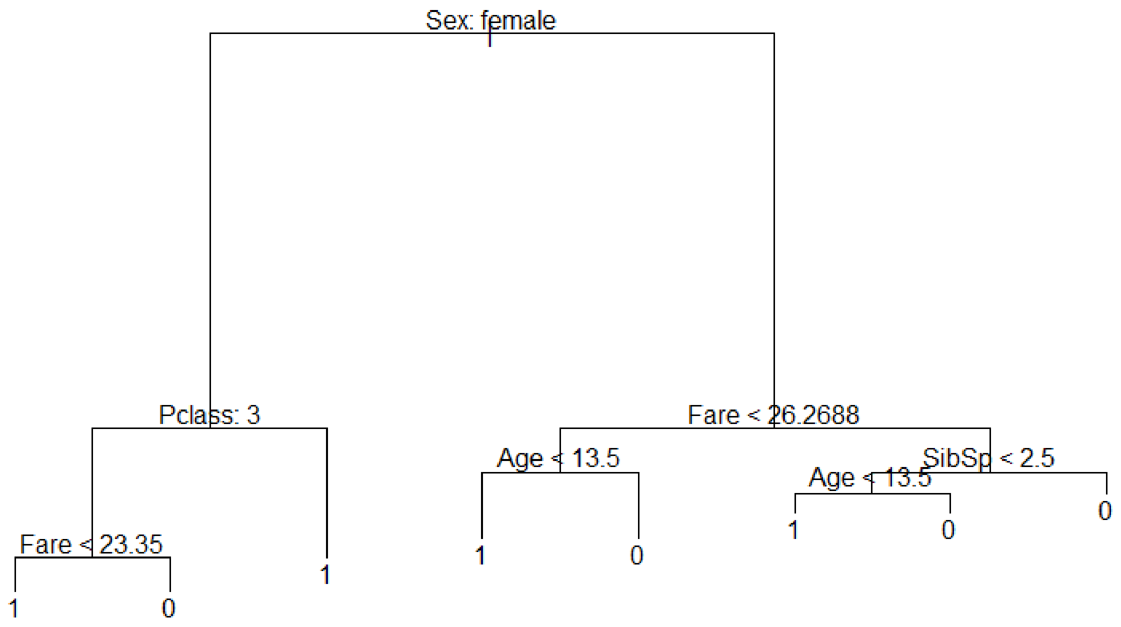
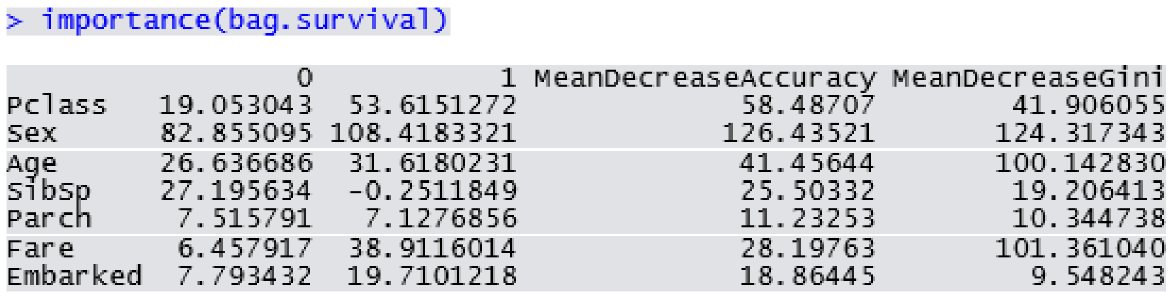


Figure 3.6 Bagging predictor variable importance values.



```
> importance(bag.survival)
```

```
                  0           1 MeanDecreaseAccuracy MeanDecreaseGini
Pclass    19.053043  53.6151272             58.48707        41.906055
Sex       82.855095 108.4183321            126.43521       124.317343
Age       26.636686  31.6180231             41.45644       100.142830
SibSp     27.195634  -0.2511849             25.50332        19.206413
Parch      7.515791   7.1276856             11.23253        10.344738
Fare       6.457917  38.9116014             28.19763       101.361040
Embarked   7.793432  19.7101218             18.86445         9.548243
```

Figure 3.7 Bagging average decrease in Gini index for each predictor variable.

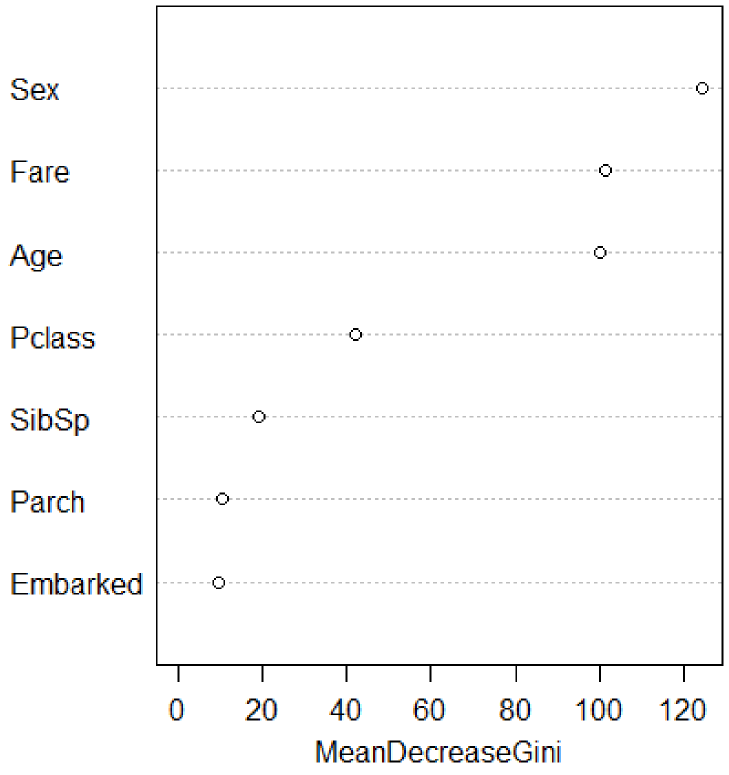
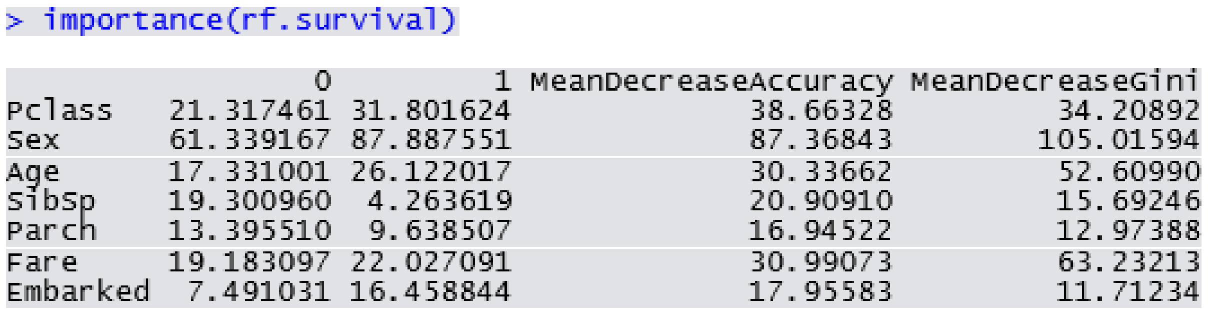


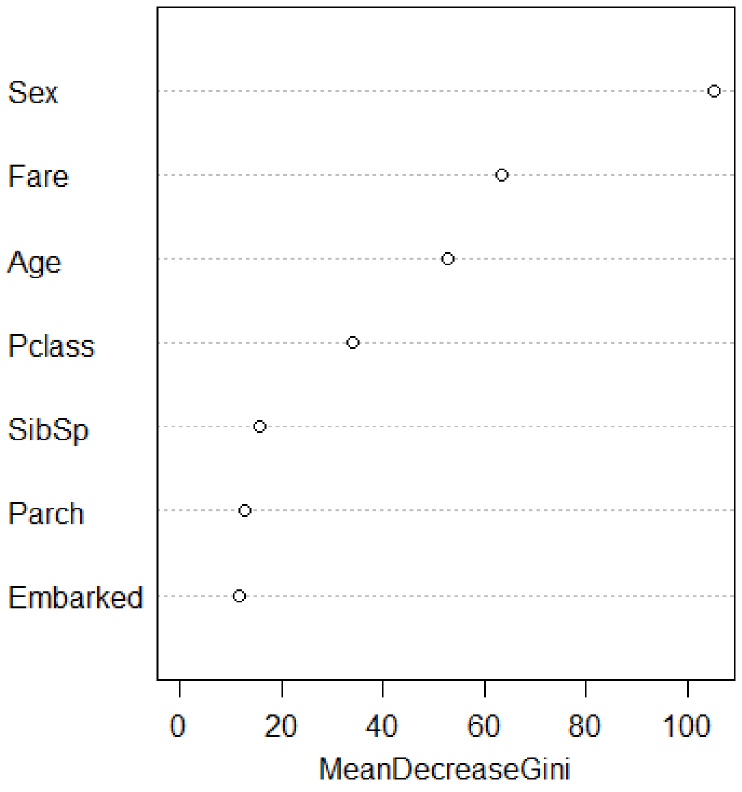
Figure 3.8 Random forest predictor variable importance values.



```
> importance(rf.survival)

                 0         1 MeanDecreaseAccuracy MeanDecreaseGini
Pclass   21.317461 31.801624             38.66328         34.20892
Sex      61.339167 87.887551             87.36843        105.01594
Age      17.331001 26.122017             30.33662         52.60990
SibSp    19.300960  4.263619             20.90910         15.69246
Parch    13.395510  9.638507             16.94522         12.97388
Fare     19.183097 22.027091             30.99073         63.23213
Embarked  7.491031 16.458844             17.95583         11.71234
```

Figure 3.9 Random forest average decrease in Gini index for each predictor variable.



Second Analysis

From the beginning we had identified predictor variables, such as *Cabin* and *Name*, to be more complex and involved variables that needed additional consideration. This thinking brought on the second approach to manipulating the data provided by Kaggle. Just as before, a column was created for the response variable, *Survived*, and appended to the test data set. Again, the training and test data sets were merged before making modifications. The same substitutions for missing values were made for the *Embarked* and *Fare* variables. On the Kaggle competition dashboard, there is a link to several tutorials to help one get started with solving the *Titanic: Machine Learning from Disaster* problem using R. We found two of these tutorials by T. Stephens (2014) and C. Wehrley (2014) to be useful in assisting with feature engineering of the more complex variables. In fact, the tutorial by T. Stephens inspired how we handled the *SibSp*, *Parch*, *Name*, and *Age* variables. Moreover, the tutorial by C. Wehrley had a significant influence on how the *Cabin* variable was worked with as well as the *Name* variable. Here we summarize the changes that were made involving the later variables mentioned. A new variable called *FamilySiz*e was created by adding the values from the *SibSp* and *Parch* variables plus one, i.e. *FamilySize* = *Sibsp* + *Parch* + 1. The *Deck* variable was created by separating and keeping the letters that were a part of each cabin observation. These letters actually correspond to which level or deck each passenger's cabin was located on the Titanic. To accommodate the *Name* variable, we created a new factor variable called *Title*. After careful consideration and research, we grouped the titles based on similar characteristics. The different levels of the new variables *Deck* and *Title* are displayed in Figure 3.10.

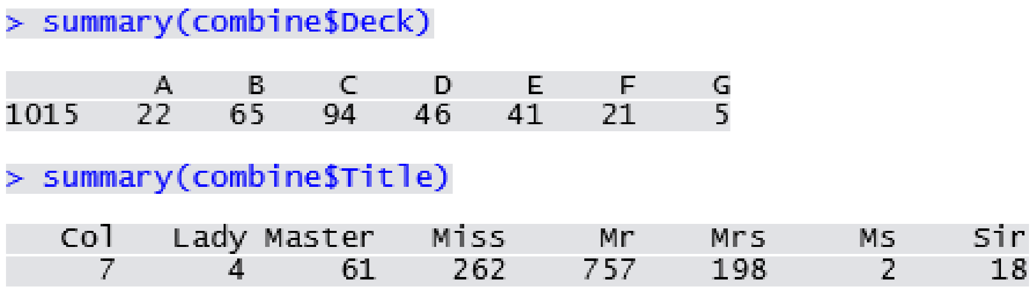
Considering that twenty percent of that age values were missing, a more creative solution needed to be applied in handling these missing values. This was a good opportunity to incorporate building a regression tree, since *Age* is a numeric variable. The formula that was used when building the regression tree for response variable *Age* incorporated nine predictor variables: *Pclass*, *Sex*, *Title*, *SibSp*, *Parch*, *FamilySize*, *Fare*, *Embarked*, *Deck*. The regression tree is depicted in Figure 3.11. We see that the most important variables in predicting age are *Title*, *Parch*, and *Pclass*. The regression tree formed consisted of seven terminal nodes. Thus, the regression tree found the predicted age values to be 22.310, 31.880, 7.124, 18.820, 28.860, 33.050, and 42.420. Now that this regression tree has been obtained, passengers with missing age values will be predicted as determined by the branch leading to the respected region or terminal node.

The purpose of the second analysis was to conduct feature engineering and replicate the methods applied in the first analysis, hoping to achieve higher prediction accuracy. However, in general there was no overall improvement in prediction accuracy as reported by Kaggle. In particular, the prediction accuracy provided for the second classification tree and corresponding pruned tree was 0.78947. Furthermore, the prediction accuracy when applying bagging was 0.72727 and later when applying random forest was 0.77033. With the exception of the performance with the bagging method, the performance from these methods could be considered acceptable in terms of predicting survival of the passengers on the RMS Titanic.

Even though, the prediction accuracy did not improve, it is important to report that this analysis is still significant for not only the feature engineering that was performed but also, the resulting classification tree produced from this analysis. The resulting classification tree is shown in Figure 3.12. Immediately one can notice that this classification tree is much cleaner and simpler. Thus, as a result of the feature engineering it has created a much more interpretable tree.

Upon further inspection of the classification trees from the first and second analysis, it is clear that there is a similar pattern present. For instance, the first split in the first classification tree is on sex while the first split in the second classification tree is on title, which ultimately depends on sex. By creating and making use of the *Title* variable in the second classification tree, young and old males and females were are distinguishable. In both trees the branches containing females have the next split on *Pclass* (refer to the left half of the first tree and right half of the second tree). Additionally, both trees indicate that older males are not likely to survive, and both trees indicate that younger males, children less than 13.5 years old, are more likely to survive. Moreover, the tree produced from the second analysis made use of the family size variable. This is a useful realization as it appears that members of smaller families, i.e. families comprised of 4 or less members, were more likely to survive.

Figure 3.10 *Deck* and *Title* variable summaries.



```
> summary(combine$Deck)

         A     B     C     D     E     F     G
1015    22    65    94    46    41    21     5

> summary(combine$Title)

    Col     Lady Master    Miss      Mr     Mrs      Ms     Sir
      7        4     61     262     757     198       2      18
```

Figure 3.11 Regression tree for response variable *Age*.

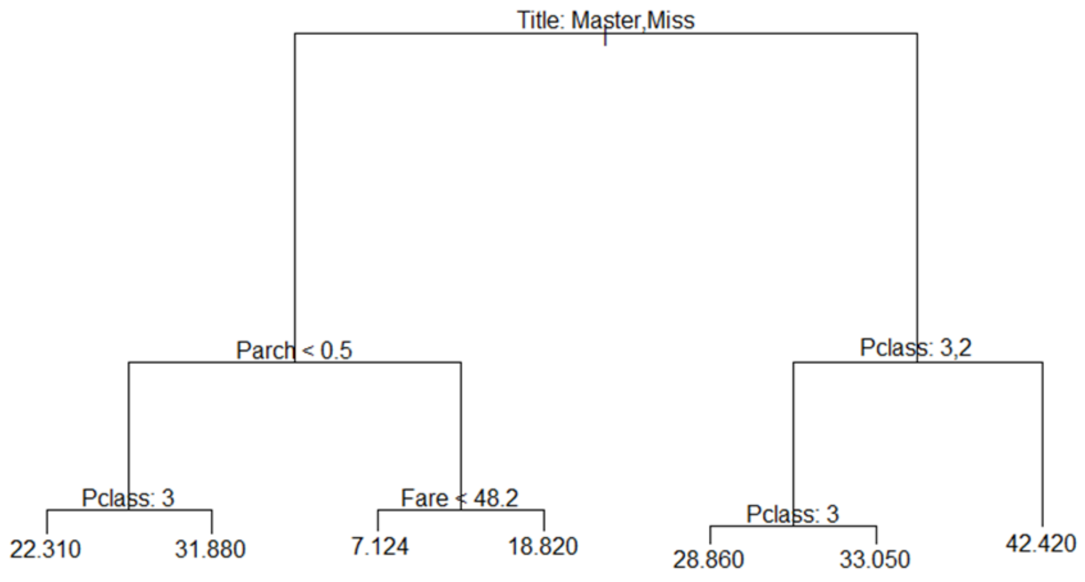
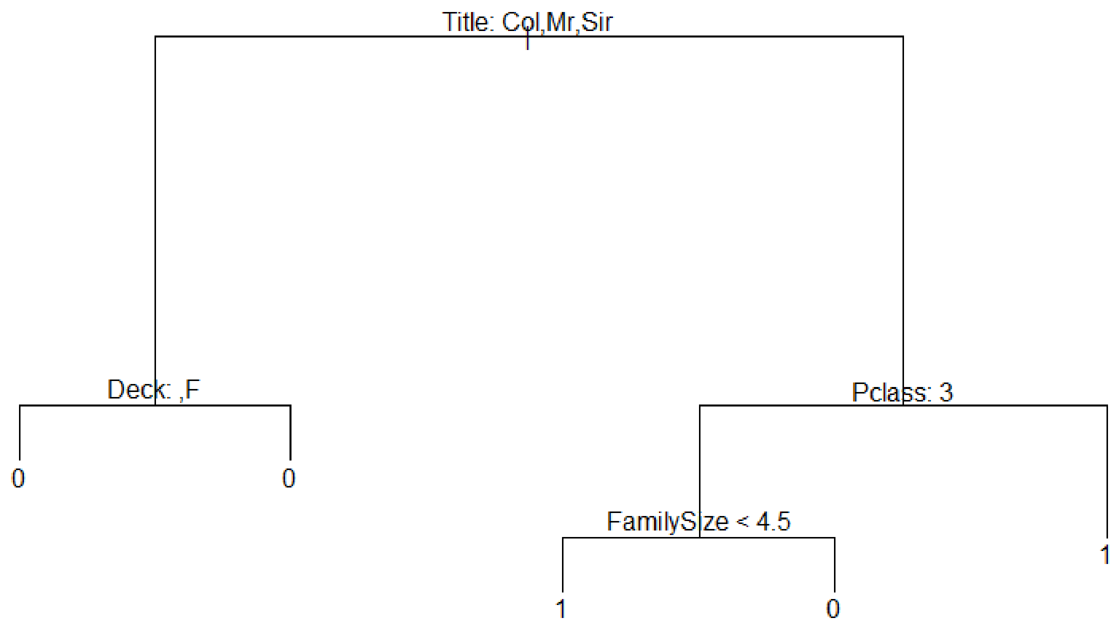


Figure 3.12 Second classification tree for response variable *Survived*.



Connections between First and Second Analysis

Alongside implementing the methods used and assessing these methods using prediction accuracy, we are able to summarize the important components of our solution when predicting survival of passengers on the RMS Titanic. All of the methods in the first analysis make use of the predictor variables, *Sex*, *Fare*, *Age*, *Pclass*, and *SibSp*. All of the methods in the second analysis make use of the predictor variables, *Title*, *Pclass*, *FamilySize*, and *Deck*. Of the predictor variables, we consider the most influential to be *Sex* or *Title*, *Fare*, *Age*, and *Pclass*. This is important because these findings support the claim that women and children, as well as the upper class, were given a priority of being rescued.

Chapter 4 - Conclusion

The motivation of this study developed from a desire to learn, understand, and apply the CART algorithm. Kaggle's predictive modeling competition, *Titanic: Machine Learning from Disaster*, served as a framework for introductory predictive analytic methods. The problem posed in this competition was to build a predictive model to predict the survival of passengers on the RMS Titanic. A great deal of time and effort was spent cleaning, organizing, and redefining variables in the data. We successfully implemented the CART methodology as well as advancements of this methodology such as bagging and random forests. We used the prediction accuracy, provided by Kaggle, to assess the efficacy of each method in correctly classifying the survival of passengers in the catastrophic ship wreck.

We obtained two sets of results coming from two different approaches to handling the data. We consider the first analysis to be the one which required minimal adjustments to the data provided by Kaggle. The second analysis was then the one which redefined the more complex variables such as *Cabin* and *Name* and used a regression tree to form predictions of the missing age values. It was determined that the first classification tree shown in Figure 3.2, provided the best prediction accuracy of all the methods. The prediction accuracy for the first classification tree was 0.79426. Cost complexity pruning was also implemented in both analyses and did not yield any improvement in prediction accuracy, but yielded a more interpretable tree. We also found that when applying bagging, the application in the first analysis gave a higher prediction accuracy than in the second analysis. The better bagging prediction accuracy provided by Kaggle was 0.76555. The application of random forests yielded an equivalent prediction accuracy of 0.77033 in both analyses.

It came at quite a surprise that there was no improvement in the prediction accuracy of survival of the passengers when applying the bagging and random forest ensemble methods. Especially since the research that we conducted on ensemble methods seemed to indicate that ensembles were certainly an improvement on any single decision tree method. However, looking back at the tutorial by T. Stephens (2014), this same phenomenon was observed. We hypothesize that this phenomenon could be attributed to the structure of the data. That is, we believe there is more to the data than what we have discovered and the data may require additional restructuring to obtain further improvement when using the methods that have been covered.

It is apparent that the effort put forth when working on the *Titanic: Machine Learning from Disaster* problem has achieved our aims and goals of this study. In attempts to provide a superior solution and further our knowledge of predictive analytics, future work that can be applied to this problem involves learning how to apply methods such as conditional random forests and boosting ensembles. Additionally, to aid in our improvement, it is crucial to spend time on optimizing and tuning parameters that appear in such methods.

Bibliography

Abbott, D. (2014). *Applied Predictive Analytics: Principles and Techniques for the Professional Data Analyst*, 1st edn. Indianapolis: John Wiley & Sons, Inc.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. New York: Chapman and Hall.

Finlay, S. (2014). *Predictive Analytics, Data Mining and Big Data: Myths, Misconceptions and Methods*, 1st edn. New York: Palgrave Macmillan.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*, 1st edn. New York: Springer.

Kaggle. (2012). *Titanic: Machine Learning from Disaster*. Available from: https://www.kaggle.com/c/titanic (accessed 26 January 2015).

Kuhn, J., and Johnson, K. (2013). *Applied Predictive Modeling*, 1st edn. New York: Springer.

Stephens, T. (2014). *Titanic: Getting Started with R*. Available from: http://trevorstephens.com/post/72916401642/titanic-getting-started-with-r (accessed 9 February 2015).

Wehrley, C. (2014). *Titanic Survival Prediction*. Available from: https://github.com/wehrley/wehrley.github.io/blob/master/SOUPTONUTS.md (accessed 9 February 2015).

Appendix A - First Application of Methods

```
setwd("K:/Statistics Research/Kaggle_Titanic Machine Learning from Disaster/Analysis 11515")

# Read in and look at training data
train <-read.csv("K:/Statistics Research/Kaggle_Titanic Machine Learning from
Disaster/Analysis 11515/train.csv")
View(train)
str(train)

# Read in and look at test data
test <- read.csv("K:/Statistics Research/Kaggle_Titanic Machine Learning from
Disaster/Analysis 11515/test.csv")
View(test)
str(test)

# Create a column for the response in the test data
test$Survived <- NA

# Combine train and test data sets (training observations come before test observations)
combine <- rbind(train, test)

# Change variable class to either numeric or factor
combine[,2] <- sapply(combine[,2], as.factor)
combine$Pclass <- factor(combine$Pclass, levels = c("3", "2", "1"), ordered = TRUE)
combine[,7] <- sapply(combine[,7], as.numeric)
combine[,8] <- sapply(combine[,8], as.numeric)

# Replace missing age values with the median age
summary(combine$Age)
combine$Age[is.na(combine$Age)] <- median(combine$Age, na.rm = TRUE)

# Replace 2 missing embark locations with the most common category, i.e. S
summary(combine$Embarked)
which(combine$Embarked == '')
combine$Embarked[c(62, 830)] = "S"

# Replace only missing fare with the median fare
summary(combine$Fare)
which(is.na(combine$Fare))
combine$Fare[1044] <- median(combine$Fare, na.rm = TRUE)

# Look at the combined data set to ensure format is correct
View(combine)
str(combine)
```

```
# Separate combined data set back into training and test data sets
train <- combine[1:891,]
test <- data.frame(combine[892:1309,],row.names = NULL)

#################################################################################

# Build a classification tree to classify Survival values using training data
attach(combine)
install.packages('tree')
library(tree)
tree.survival <- tree(Survived ~ Pclass + Sex
                + Age + SibSp + Parch
                + Fare + Embarked, combine, subset = 1:891)
summary(tree.survival)
tree.survival
plot(tree.survival)
text(tree.survival, pretty = 0)

# Make predictions of Survival values in test data
tree.predict <- predict(tree.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit <- data.frame(PassengerId = test$PassengerId, Survived = tree.predict)
write.csv(submit, file = "firstclasstree.csv", row.names = FALSE)
# Submission result 0.79426

#################################################################################

# Conduct cross-validation and cost complexity pruning
set.seed(100)
cv.survival <- cv.tree(tree.survival, FUN = prune.misclass)
names(cv.survival)
cv.survival

par(mfrow =c(1,2))
# Plot cross-validation error as a function of tree size
plot(cv.survival$size ,cv.survival$dev ,type="b")
# Plot cross-validation error as a function of the cost complexity parameter, k
plot(cv.survival$k ,cv.survival$dev ,type="b")

# Form the pruned survival classification tree
prune.survival <- prune.misclass(tree.survival, best = 8)
summary(prune.survival)
prune.survival
plot(prune.survival)
text(prune.survival, pretty = 0)
```

```
# Make predictions of Survival values in test data
tree.predict <- predict (prune.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit0 <- data.frame(PassengerId = test$PassengerId, Survived = tree.predict)
write.csv(submit0, file = "firstprunedtree.csv", row.names = FALSE)
# Submission result 0.79426

##################################################################################

# Utilize bagging to classify Survival values using training data
install.packages('randomForest')
library(randomForest)
set.seed (200)
bag.survival <- randomForest(Survived ~ Pclass + Sex
                            + Age + SibSp + Parch
                            + Fare + Embarked, data = combine, subset = 1:891,
                            mtry=7, importance =TRUE)

bag.survival
plot(bag.survival)

# Make predictions of Survival values in test data
yhat.bag <- predict(bag.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit2 <- data.frame(PassengerId = test$PassengerId, Survived = yhat.bag)
write.csv(submit2, file = "firstbaggedtree.csv", row.names = FALSE)
# Submission result 0.76555

# View the importance values of each variable
importance(bag.survival)
# Plot the importance values
varImpPlot(bag.survival)

##################################################################################

# Utilize random forest to classify Survival values using training data
set.seed (200)
rf.survival <- randomForest(Survived ~ Pclass + Sex
                           + Age + SibSp + Parch
                           + Fare + Embarked, data = combine, subset = 1:891,
                           importance =TRUE)
rf.survival
plot(rf.survival)
```

```
# Make predictions of Survival values in test data
yhat.bag2 <- predict(rf.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit3 <- data.frame(PassengerId = test$PassengerId, Survived = yhat.bag2)
write.csv(submit3, file = "firstrandomforest.csv", row.names = FALSE)
# Submission result 0.77033

# View the importance values of each variable
importance(rf.survival)
# Plot the importance values
varImpPlot(rf.survival)
```

Appendix B - Second Application of Methods

```
setwd("K:/Statistics Research/Kaggle_Titanic Machine Learning from Disaster/Analysis 11515")

# Read in and look at training data
train <-read.csv("K:/Statistics Research/Kaggle_Titanic Machine Learning from
Disaster/Analysis 11515/train.csv")
View(train)
str(train)

# Read in and look at test data
test <- read.csv("K:/Statistics Research/Kaggle_Titanic Machine Learning from
Disaster/Analysis 11515/test.csv")
View(test)
str(test)

# Create a column for the response in the test data
test$Survived <- NA

# Combine train and test data sets (training observations come before test observations)
combine <- rbind(train, test)

# Change variable class to either numeric or factor
combine[,2] <- sapply(combine[,2], as.factor)
combine$Pclass <- factor(combine$Pclass, levels = c("3", "2", "1"), ordered = TRUE)
combine[,7] <- sapply(combine[,7], as.numeric)
combine[,8] <- sapply(combine[,8], as.numeric)

# Create family size variable
combine$FamilySize <- combine$SibSp + combine$Parch + 1

# Replace 2 missing embark locations with the most common category, i.e. S
summary(combine$Embarked)
which(combine$Embarked == '')
combine$Embarked[c(62, 830)] = "S"

# Replace only missing fare with the median fare
summary(combine$Fare)
which(is.na(combine$Fare))
combine$Fare[1044] <- median(combine$Fare, na.rm = TRUE)

# Create the Deck variable by separating and pulling off the deck letter contained in the Cabin
values
str(combine$Cabin)
combine$Cabin <- as.character(combine$Cabin)
combine$Deck <- substring(combine$Cabin, 1, 1)
```

```
combine$Deck <- as.factor(combine$Deck)
summary(combine$Deck)
which(combine$Deck == 'T')
combine$Deck[340] = ""
str(combine$Deck)

# Create the Title variable by separating and pulling off the name prefix
combine$Name <- as.character(combine$Name)
combine$Title <- sapply(combine$Name, FUN = function(x) {strsplit(x, split = '[,.]')[[1]][2]})
combine$Title <- sub(' ', '', combine$Title)
table(combine$Title)
combine$Title[combine$Title %in% c('Capt', 'Col', 'Major')] <- 'Col'
combine$Title[combine$Title %in% c('Miss', 'Mlle')] <- 'Miss'
combine$Title[combine$Title %in% c('Mrs', 'Mme')] <- 'Mrs'
combine$Title[combine$Title %in% c('Don', 'Dr', 'Rev', 'Sir')] <- 'Sir'
combine$Title[combine$Title %in% c('Dona', 'Jonkheer', 'Lady', 'the Countess')] <- 'Lady'
combine$Title <- as.factor(combine$Title)

# Build a regression tree with response Age using training data
summary(combine$Age)
install.packages('tree')
library(tree)
attach(combine)
tree.age <- tree(Age ~ Pclass + Sex + Title
            + SibSp + Parch + FamilySize
            + Fare + Embarked + Deck, combine[!is.na(combine$Age),])
summary(tree.age)
tree.age
plot(tree.age)
text(tree.age, pretty = 0)

# Make predictions of Age values in test data
combine$Age[is.na(combine$Age)] <- predict(tree.age, newdata =
combine[is.na(combine$Age),])

# Look at the combined data set to ensure format is correct
View(combine)
str(combine)

# Separate combined data set back into training and test data sets
train <- combine[1:891,]
test <- data.frame(combine[892:1309,],row.names = NULL)

##################################################################################

# Build a classification tree to classify Survival values using training data
```

```
attach(combine)
tree.survival <- tree(Survived ~ Pclass + Sex + Title
                + Age + SibSp + Parch + FamilySize
                + Fare + Embarked + Deck, combine, subset = 1:891)
summary(tree.survival)
tree.survival
plot(tree.survival)
text(tree.survival, pretty = 0)

# Make predictions of Survival values in test data
tree.predict <- predict(tree.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit <- data.frame(PassengerId = test$PassengerId, Survived = tree.predict)
write.csv(submit, file = "secondclasstree.csv", row.names = FALSE)
# Submission result 0.78947

##################################################################################

# Conduct cross-validation and cost complexity pruning
set.seed(100)
cv.survival <- cv.tree(tree.survival, FUN = prune.misclass)
names(cv.survival)
cv.survival

par(mfrow =c(1,2))
# Plot cross-validation error as a function of tree size
plot(cv.survival$size ,cv.survival$dev ,type="b")
# Plot cross-validation error as a function of the cost complexity parameter, k
plot(cv.survival$k ,cv.survival$dev ,type="b")

# Form the pruned survival classification tree
prune.survival <- prune.misclass(tree.survival, best = 4)
summary(prune.survival)
prune.survival
plot(prune.survival)
text(prune.survival, pretty = 0)

# Make predictions of Survival values in test data
tree.predict <- predict (prune.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit0 <- data.frame(PassengerId = test$PassengerId, Survived = tree.predict)
write.csv(submit0, file = "secondprunedtree.csv", row.names = FALSE)
# Submission result 0.78947
```

```
################################################################################

# Utilize bagging to classify Survival values using training data
install.packages('randomForest')
library(randomForest)
set.seed (5000)
bag.survival <- randomForest(Survived ~ Pclass + Sex + Title
                        + Age + FamilySize
                        + Fare + Embarked + Deck, data = combine, subset = 1:891,
                        mtry=8, importance = TRUE)
bag.survival
plot(bag.survival)

# Make predictions of Survival values in test data
yhat.bag <- predict(bag.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit2 <- data.frame(PassengerId = test$PassengerId, Survived = yhat.bag)
write.csv(submit2, file = "secondbaggedtree.csv", row.names = FALSE)
# Submission result 0.72727

# View the importance values of each variable
importance(bag.survival)
# Plot the importance values
varImpPlot(bag.survival)

################################################################################

# Utilize random forest to classify Survival values using training data
set.seed (5000)
rf.survival <- randomForest(Survived ~ Pclass + Sex + Title
                        + Age + FamilySize
                        + Fare + Embarked + Deck, data = combine, subset = 1:891,
                        importance =TRUE)
rf.survival
plot(rf.survival)

# Make predictions of Survival values in test data
yhat.bag2 <- predict(rf.survival, newdata = combine[892:1309,], type = "class")

# Create a csv file to submit to Kaggle
submit3 <- data.frame(PassengerId = test$PassengerId, Survived = yhat.bag2)
write.csv(submit3, file = "secondrandomforest.csv", row.names = FALSE)
# Submission result 0.77033

# View the importance values of each variable
```

```
importance(rf.survival)
# Plot the importance values
varImpPlot(rf.survival)
```