Safety Analysis of Automobiles

Graham Marlow

June 8, 2015

Abstract

The purpose of this project is to classify risky and safe cars based off of a number of attributes acquired from the "Automobile" data set. To do this, I use four different classification trees and judge by the misclassification error which tree method is the best one to use. I begin with a standard binary recursive partitioning method and examine the tree based off of its most basic merits. From there, I partition the data into a training set and test set and create a new tree, and test its accuracy. I then proceed to use K-fold cross validation to make a newly pruned tree, and examine the error of this tree compared to the original. The next two methods that I use involve simulating many trees, utilizing random forests and boosting, which I compare to the pruned tree achieved during cross-validation.

At the end of all of the methods, it turns out that the random forests method produces the least amount of misclassification error. After examining the cross-validation error, it seems that the optimal amount of nodes for the classification tree is six. The variables that the random forest method declared most important for classifying risky vehicles are number of doors, normalized losses, wheel base, height, and length. The random forest method disproves my hypothesis that price would be the most important factor for determining the riskiness of a vehicle.

Introduction

In this project I will be analyzing the safety ratings of various brands of cars and determine what variables are most important when classifying whether or not a car is risky. I will be using the automobile data set from the UC Irvine Machine Learning Repository, which consists of 205 observations and 26 variables. 24 of the variables account for a particular attribute of the vehicle (fuel-type, weight, price, stroke, etc.). The other two variables are the safety rating (called symboling) and the make of the vehicle (Audi, BMW, etc.).

Goal

The purpose of this project and its implications lies in the classification of risky and safe cars. By discovering which variables contribute most towards the classification of riskiness, we can help make our cars safer across the board by knowing what attributes of vehicles to change.

Data Cleanup

To properly work with this dataset, I removed the `make` variable from the data as there are too few observations in the dataset to classify such a wide variety of different brands of car. Additionally, `make` is the only variable that does not contribute to a physical attribute of a car.

To use the Random Forest method of classification, I needed to remove all blank or `NA` attributes from the dataset. In order to have results that are comparable across all classification techniques, I removed these observations from the data completely for all methods.

Attributes

Symboling is represented as an integer that falls within the set [-3, 3]. A value of -3 indicates that the vehicle is considered safe, whereas a value of +3 indicates that the vehicle is "risky". Since this is the response variable that we wish to classify, I changed it into a binary variable to make it easier to work with. I split the variable into "safe" cars, or cars that have a symboling within [-3, 0] and "risky" cars, or cars that have a symboling within [1, 3]. Due to the large amount of cars that are classified as "risky", I decided to classify neutral cars (cars with a symboling of 0) as safe. This will help balance out the data and improve the accuracy of my results.

Normalized losses is another variable that is very important to the dataset, as it represents the relative average loss per payment per insured vehicle year. This loss is normalized for all vehicles that fall within a particular size classification.

In an attempt to achieve the best possible classification given the dataset, `make` was the only variable that I removed.

Methods

I use several different tree-based classification methods.

1. Binary recursive partitioning. This is to provide a very basic tree at the beginning of the analysis and gain a broad knowledge of the data.
2. K-fold cross validation. I use this method to define a more accurate training and testing procedure in an attempt to minimize misclassification error.
3. Random forests. This technique, while computationally expensive, is very powerful and typically provides better results than cross validation.
4. Boosting. Another powerful technique, this method and random forests both provide a very strong analysis. I use this technique to compare important variables attained during the random forests method.

Results

The random forest method of classification ended up giving the lowest misclassification error. The method misclassified only `5%` of observations, which is very low. Therefore, I believe that the mean tree produced by the random forest method provides a good method of classification. The boosting method came in a close second, with a misclassification error of `5.7%`. However, as I will note later in the project, the variables that the boosting method classified as important are vastly different from those of the random forest method. K-fold cross validation yielded a misclassification error of `5.8%`.

The variables that were classified as most important by the random forest method were: number of doors, wheel base, normalized losses, height, length, and curb weight. There is a significant drop in importance rating when advancing after normalized losses; number of doors, wheel base, and normalized losses all of a high importance rating of about 9, whereas the other notable variables in the dataset hover around 5. All other variables have ratings that are 2 or lower.

Therefore, it seems that the number of doors, wheelbase, and normalized losses are the three most important attributes to keep in mind when determining the riskiness of an automobile.

Software and Packages Used

For the analysis within this project, I used R. For the methods detailed above I used several different packages. These packages are `tree`, for the binary recursive tree method, `randomForest`, for the random forests method, and `ada`, for the boosting method.

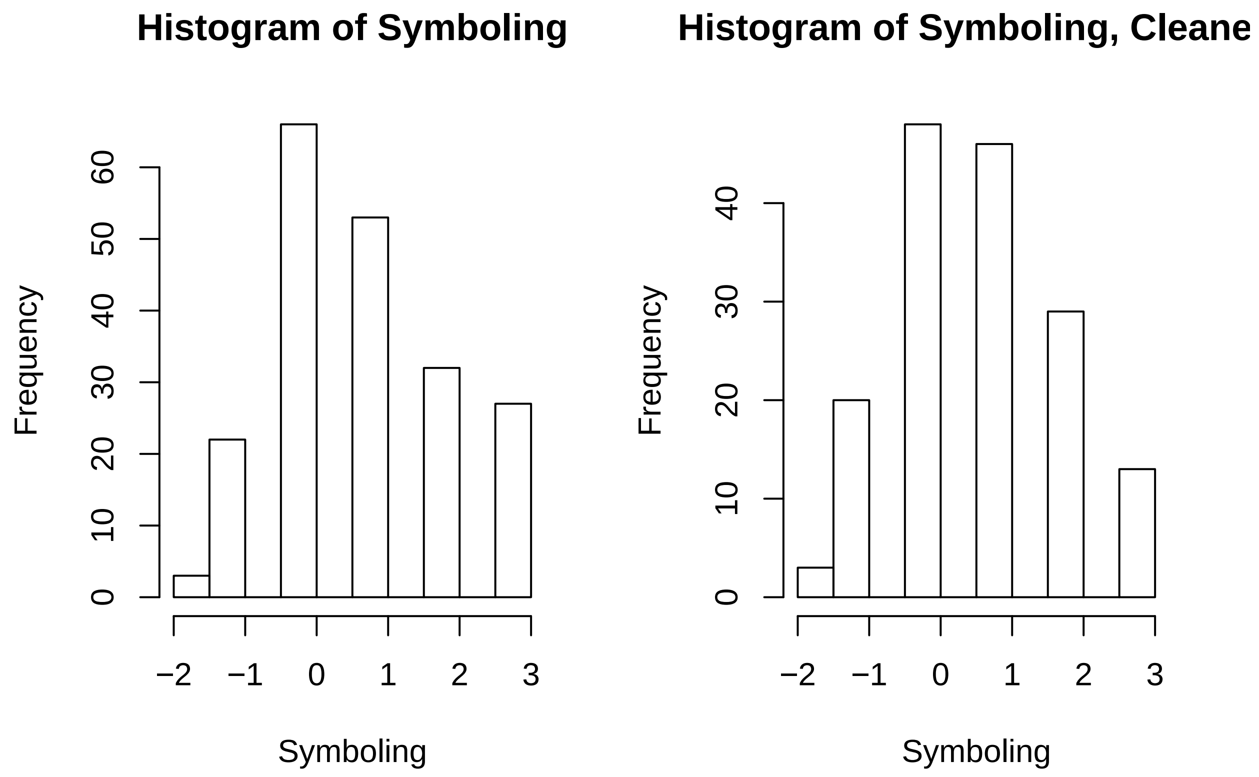
My dataset is found on UC Irvine's Machine Learning Repository, and is accredited to Jeffery C. Schlimmer. The data was donated in 1985. The sources for this data are outlined within my R code, as well as on the UCI page for the dataset.

Discussion

Because this dataset is from 1985, it may not be representative of modern cars, however, it should still present a good indication of important attributes that can be used for modern cars. Furthermore, the fact that I removed `44` attributes in order to use the random forest method makes my data very sparse and not as strong.

Data Description

The data that I used for this project comes from the UCI Machine Learning Repository. The dataset is called `automobile` and contains `25` variables, the most important of which is `symboling`, the index of safety that this project will use to classify the data.



Left is a histogram of symboling from the raw dataset. As we can see from the distribution of frequencies, most of the observations are either 0 or positive, with only a very few vehicles being classified as completely safe in the negative values. Because 0 is considered "neutral" and not "risky", I group it together with the other "safe" variables. This also balances out the data into risky and safe groups.

Additionally, due to the fact that the random forest method will not work with any blank or NA values, I removed all observations that contained such values. The number of observations removed is 44. I removed these observations for every method so that I could compare the results for each method using the same data and provide consistency.

The right histogram is the new distribution of symboling values, in a dataset with all observations containing NA removed. There is not a whole lot of change happening between the two distributions, so I think that removing the observations that contain blank values is justified.

Tree-based Methods

In this section, I will be performing a number of different classification trees using different classification methods.

Binary Recursive Partitioning

The first technique I will use is binary recursive partitioning. This is the most basic method out of the four that I will cover, and therefore results in the greatest misclassification error.

```r
# Load required packages
library(tree)
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```r
library(ada)
```

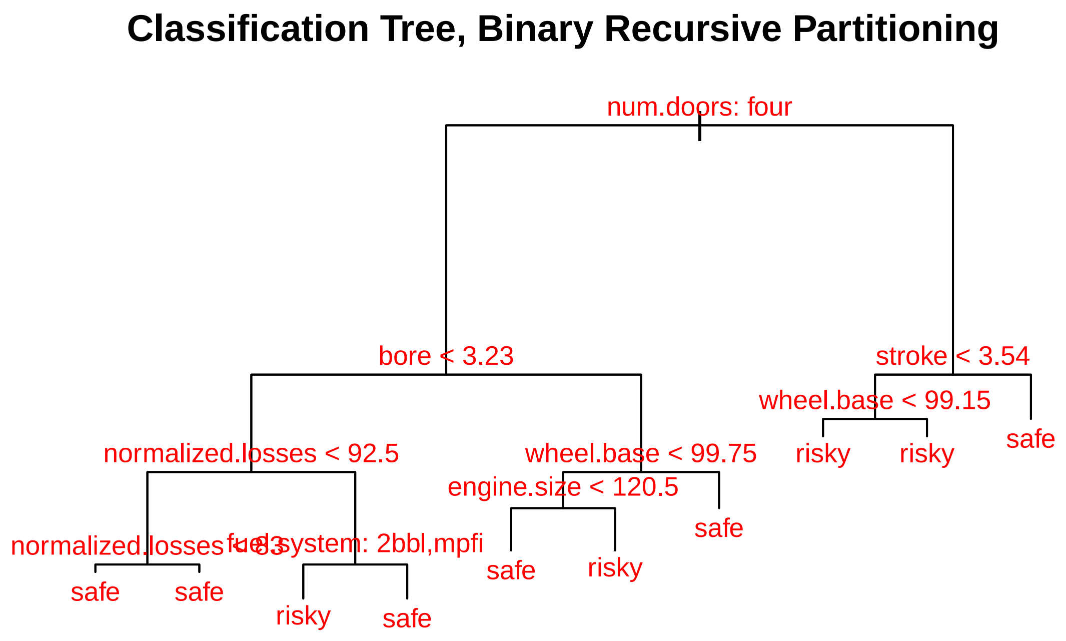
```
## Loading required package: rpart
```

```r
# Attach binary variable to dataset, remove symboling
new.data <- subset(new.data, select = -c(symboling))
tree.data <- data.frame(new.data, biSym)

# Simple Classification Tree, no Model Evaluation
####################################################
# Generate tree using defaults
car.tree <- tree(biSym ~., data=tree.data)
summary(car.tree)
```

```
##
## Classification tree:
## tree(formula = biSym ~ ., data = tree.data)
## Variables actually used in tree construction:
## [1] "num.doors"         "bore"               "normalized.losses"
## [4] "fuel.system"       "wheel.base"         "engine.size"
## [7] "stroke"
## Number of terminal nodes:  10
## Residual mean deviance:  0.2392 = 35.64 / 149
## Misclassification error rate: 0.06289 = 10 / 159
```

```
# Plot the tree and labels
par(mfrow=c(1, 1))
plot(car.tree)
text(car.tree, pretty=0, cex=0.8, col='red')
title('Classification Tree, Binary Recursive Partitioning')
```



The binary recursive partitioning method involves partitioning the response variable (`symboling`) and choosing splits from the rest of the variables in the dataset. Splits are chosen based off of their impurity – we wish to maximize the reduction in impurity based off of the variables. The splitting halts when the terminal nodes are too small or too few to be split.

The classification tree in this case has 10 nodes, detailing these 7 variables: number of doors, bore, normalized losses, fuel system, wheel base, engine size, and stroke. Number of doors seems to be the most important variable, as it is the root of the tree. If the number of doors of a given car is *not* four, the car is very likely to be risky.

Next, I want to use model validation techniques to test the previous modelling technique and see whether or not we can prune it. This is accomplished through the use of a training set and a test set. I will use the training set, consisting of 75% of the data, to model the data and a test set, consisting of the remaining data, to test the model.

```r
# Model Validation
set.seed(1)
# Training and Test sets
# Training set, sample 75% of the data
index <- sample(dim(new.data)[1], size=floor(dim(new.data)[1])*.75)
# Take remaining for test set
index.test <- setdiff(1:dim(new.data)[1], index)

# Our two sets:
train.set <- new.data[index,]

symbol.test <- biSym[index.test]
test.set <- new.data[index.test,]

# Model Validation
# Use training set and test set to evaluate our classification tree
car.tree <- tree(biSym ~., data=tree.data, subset=index)

# Predict on test set
tree.pred <- predict(car.tree, test.set, type='class')

# Confusion matrix
error <- table(tree.pred, symbol.test)

# Classification Error
1 - sum(diag(error))/sum(error)
```

```
## [1] 0.1
```

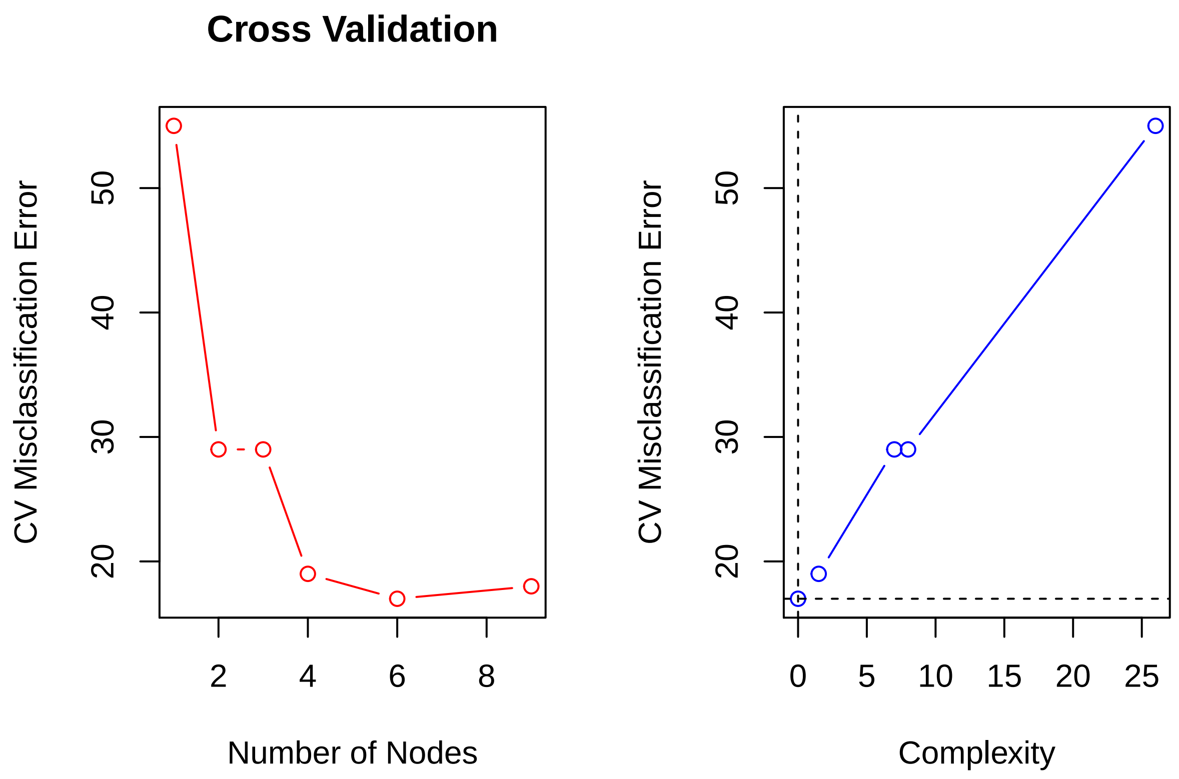
In the above code, I separated the original dataset into a training set and test set, modeled a tree from the training set, and compared it to a tree that is predicted based off of the test set. Error is calculated based off of comparing the predicted model (from the test set) to the response variable (in our case, `symboling`.)

After validating the model using a training set and test set, we are left with a misclassification error of `10%`.

K-fold Cross Validation

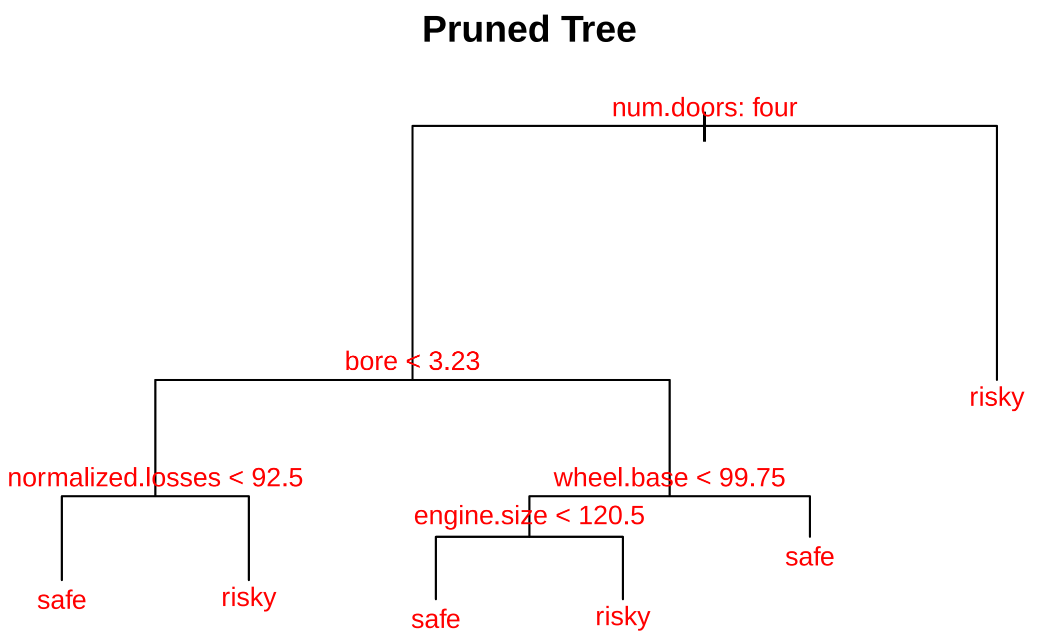
The second technique that I will use is k-fold cross validation. This method will automatically create a training set and test set, based off of our original data. In effect, the process of training a tree and testing it will be repeated multiple times on slightly different subsets of our automobile dataset. The advantage to using this technique over recursive partitioning is reduced variability, due to the fact that we are predicting the fit of a model based off of a hypothetical validation set.

However, there is also a downside to using cross validation: one must declare a parameter `K` to judge their prediction. `K` is the number of folds that take place in the cross validation, and my choice of `K = 10` is fairly arbitrary, as it seems to be a good spot between having a large number of sample combinations (low `K`), and a low number of sample combinations (high `K`).



Before continuing with the cross validation technique, it is important to examine an optimal level of `k`, or the number of nodes that the tree should have. Given the first plot, number of nodes vs. error, we can see that we attain the minimum error at `k = 6`. Therefore, we should use `6` nodes when we construct our tree. The importance of reducing the complexity of the model is demonstrated through the complexity vs. error plot, where a minimum error is attained by having the minimum amount of complexity.

```r
# Prune tree
par(mfrow=c(1, 1))
prune.car <- prune.misclass(car.tree, best=6)
plot(prune.car)
text(prune.car, pretty=0, col='red', cex=0.8)
title('Pruned Tree')
```



```r
summary(prune.car)
```

```
##
## Classification tree:
## snip.tree(tree = car.tree, nodes = c(3L, 9L))
## Variables actually used in tree construction:
## [1] "num.doors"         "bore"              "normalized.losses"
## [4] "wheel.base"        "engine.size"
## Number of terminal nodes:  6
## Residual mean deviance:  0.4128 = 46.64 / 113
## Misclassification error rate: 0.05882 = 7 / 119
```

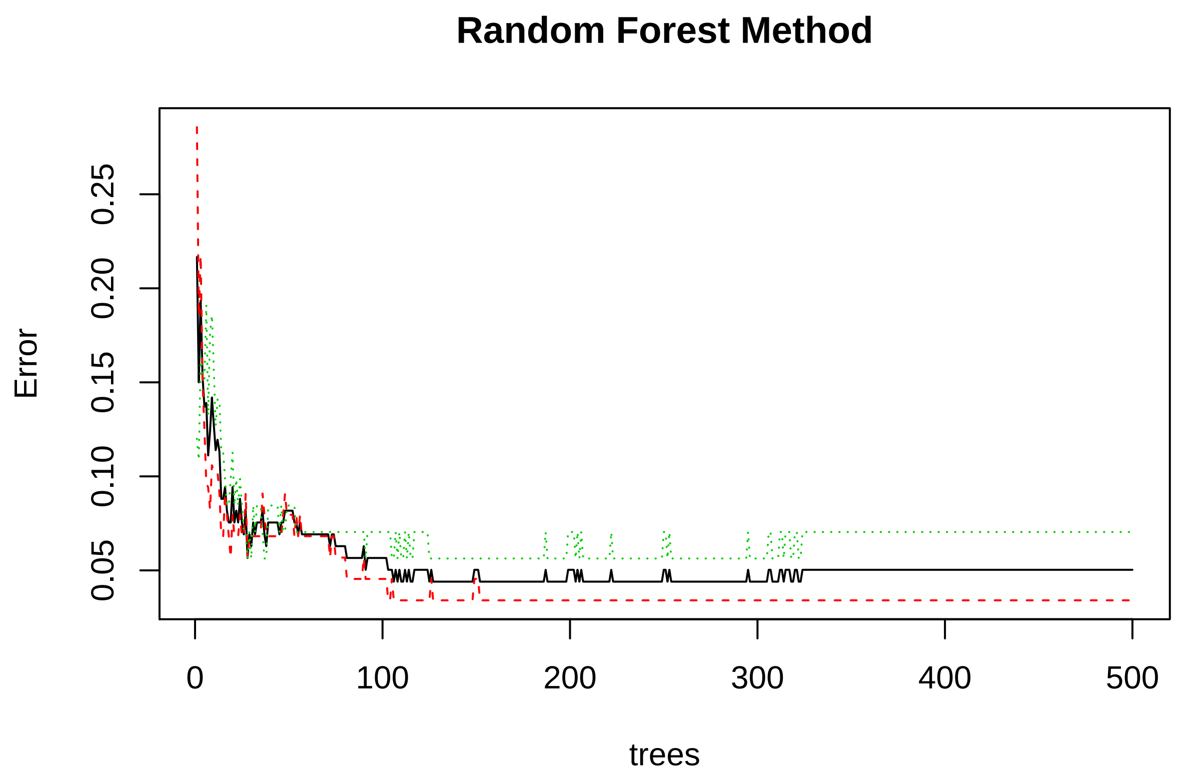
Given these estimations, we now form a new tree based upon the optimal number of nodes, `6`. This tree cuts out the less important variables from the model, while maintaining the ones that contribute most towards classifying riskiness. This tree demonstrates that all observations that have a number of doors different from four are classified as risky, which is fairly significant. The other important classifying variables include bore, normalized losses, wheel base, and engine size.

The error for this tree is `5.8%`, better than the binary recursive tree.

Random Forest

The next method that I will use to classify riskiness is the random forest method. This method involves boostrapping a large number of tree simulations, and using the mean result as the final product. This usually results in a better model than single-based tree models.

```r
# Classification Tree using Random Forest
fit <- randomForest(biSym ~., data=tree.data)
plot(fit,
      main = 'Random Forest Method')
```



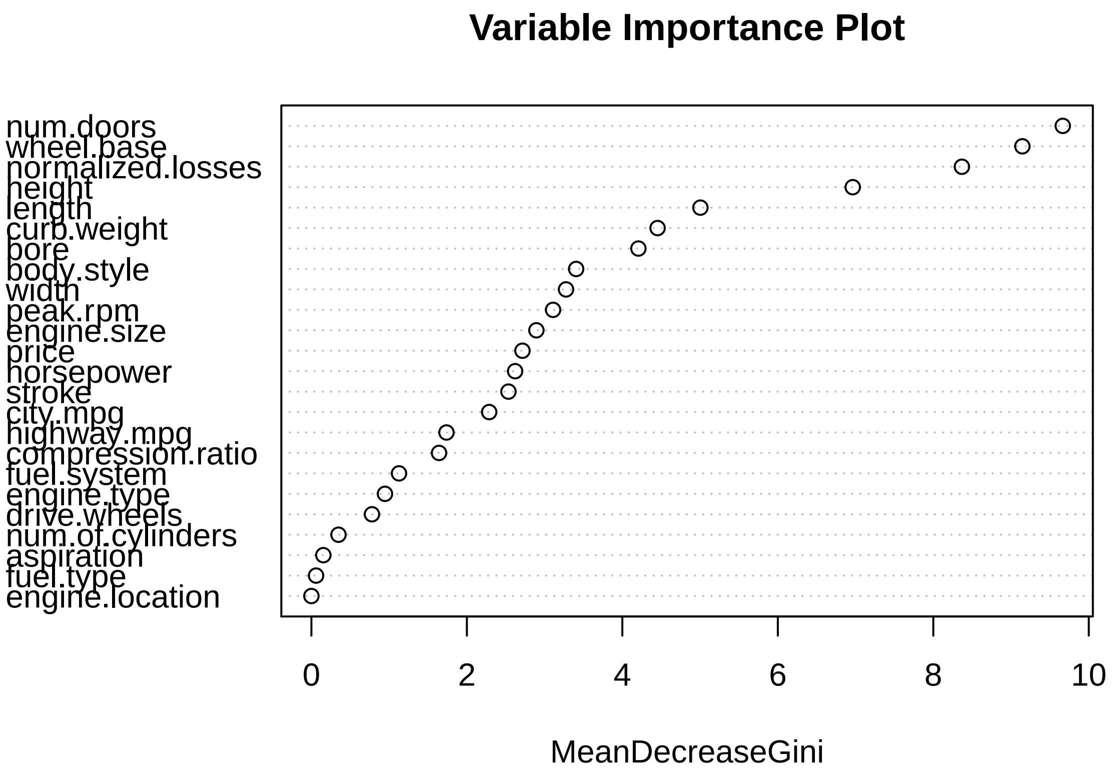
```r
print(fit)
```

```
##
## Call:
##  randomForest(formula = biSym ~ ., data = tree.data)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 5.03%
## Confusion matrix:
##       risky safe class.error
## risky    85    3  0.03409091
## safe      5   66  0.07042254
```

The plot above gives an illustration of all of the tree simulations and their respective errors. As the number of simulations increases, the amount of error decreases until it begins to normalize around a specific value, in this case, `5.03%`. The red dashed line represents the lower bound of the confidence interval around the error, and the green dashed line represents the upper bound.

The end misclassification error of `5.03%` is better than the cross validation error, and supports the reasoning that simulating many trees gives a better result than using just one. Therefore, our preferred model in this project is the random forest model.

```r
# List importance
varImpPlot(fit,
           main = 'Variable Importance Plot')
```



The above list shows all of the variables of the dataset and their respective importance values. A higher importance indicates that the variable is more useful in classifying whether or not an auto is risky or safe. The results from the random forest method are actually very similar to the results from the cross validation method, where number of doors, wheel base, and normalized losses are the most important variables to consider. However, unlike the cross validation model, our random forest model does not think that engine size is an important factor in determining riskiness. To further test the variables chosen, I will use the boosted method of classification.

Boosting Method

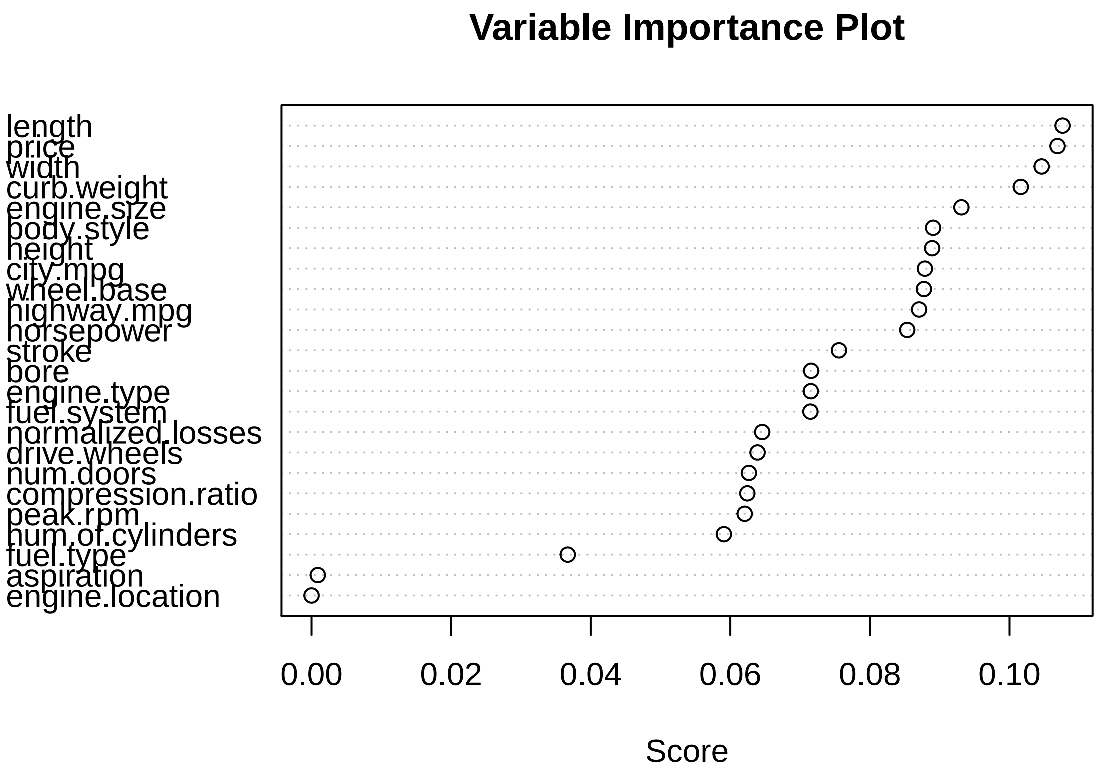
The boosting method is similar to random forests, however, is a supervised learning technique. Unlike the random forest method, boosting gradually refits the data on each iteration, effectively improving it. Therefore, each new iteration is dependent on the previous. This makes for a computationally intensive, yet accurate classification.

```r
# Classification with Boosting
fit.boost <- ada(biSym ~., data=tree.data)
fit.boost
```

```
## Call:
## ada(biSym ~ ., data = tree.data)
##
## Loss: exponential Method: discrete   Iteration: 50
##
## Final Confusion Matrix for Data:
##           Final Prediction
## True value risky safe
##      risky    88    0
##      safe      3   68
##
## Train Error: 0.019
##
## Out-Of-Bag Error:  0.057  iteration= 50
##
## Additional Estimates of number of iterations:
##
## train.err1 train.kap1
##         46         46
```

The misclassification error for the boosting method is `5.7%`. This is not as good as the random forest method, but is slightly better than CV. However, it is important to note that the variables that the boosting method declares most important are very different from the random forest method.

```
# Plot variable importance
varplot(fit.boost)
```



In the boosting method, length, price, and weight are determined the three most important variables to classifying the data. This contrasts the other methods, where number of doors and normalized losses are among the most important variables.

Conclusion

When classifying the automobile dataset by `symboling`, the random forest method provides the best results. The random forest classification tree resulted in a misclassification error of about `5%`, the lowest of the trees. Furthermore, the random forest method reduces variability due to bootstrapping, making it favorable to cross validation and binary partitioning methods.

The most important variables when classifying riskiness were number of doors, normalized losses, and wheel base.

To further the results from this project, more analysis could be conducted on what aspects of each of the major variables could be changed in order to yield safer results. For example, because we now know that normalized losses, wheel base, and number of doors are the most important variables when it comes to classifying an auto as safe, we could test what qualities of each of these variables should be changed in order to achieve a smaller risk rating. If one wants to make a family-safe vehicle, the results of this project suggest that the vehicle should have four doors.

Appendix

```r
setwd('C:/Users/Graham/Desktop/FinalProject131/data')
auto.data <- read.csv('C:/Users/Graham/Desktop/FinalProject131/data/data_edited.csv',
                      header=F)

colnames(auto.data) = c(
  'symboling',
  'normalized.losses',
  'make',
  'fuel.type',
  'aspiration',
  'num.doors',
  'body.style',
  'drive.wheels',
  'engine.location',
  'wheel.base',
  'length',
  'width',
  'height',
  'curb.weight',
  'engine.type',
  'num.of.cylinders',
  'engine.size',
  'fuel.system',
  'bore',
  'stroke',
  'compression.ratio',
  'horsepower',
  'peak.rpm',
  'city.mpg',
  'highway.mpg',
  'price'
  )

# Preliminaries
# load and modify data; assign training set and test set

# load data from 'data.R'
setwd('C:/Users/Graham/Desktop/FinalProject131/src')
source('data.R', local=TRUE)
head(auto.data)
dim(auto.data)


# Data Modification
####################
# Remove make variable from data
safety.data <- subset(auto.data, select = -c(make))
dim(safety.data)

# Remove unknown/blank observations from the data
# We remove them because we cannot have blank observations when using Random Forest
# Classification
new.data <- na.omit(safety.data) # remove all NA obs.
dim(new.data)
# We do it immediately so we can compare results across all tree methods

# Create binary variable out of symboling, classifying risky and safe
biSym <- ifelse(new.data$symboling <= 0, c('safe'), c('risky') )
num.binary <- ifelse(new.data$symboling <= 0, 1, 0 )


# Exploratory Analysis
#######################

# Load in the data
setwd('C:/Users/Graham/Desktop/FinalProject131/src')
source('prelims.R', local=TRUE)

# safety.data is our vanilla dataset
dim(safety.data)

# data is our dataset with the binary variable attached
data <- data.frame(new.data, num.binary)

# Show the distribution of symboling parameter
hist(safety.data$symboling)
title('Histogram of Symboling')

# Distribution of symboling after removing blank observations
hist(data$symboling)
title('Histogram of Symboling, Removed Obs.')
```

```r
# Distribution of Binary variable
hist(data$num.binary)
title('Histogram of risky (0) and safe (1)')

# Plot price vs. symboling
plot(data$price, data$num.binary,
     xlab = 'Price',
     ylab = 'Safety')
title('Price vs. Safety')

# Simple linear regression, price vs. safety
fit.price <- lm(data$num.binary ~ data$price)
summary(fit.price)
# It seems that price has a statistically significant impact on symboling


# Classification Tree

# Load required packages
library(tree)
library(randomForest)
library(ada)

# load data from 'prelims.R'
setwd('C:/Users/Graham/Desktop/FinalProject131/src')
source('prelims.R', local=TRUE)
dim(new.data)

# Attach binary variable to dataset, remove symboling
new.data <- subset(new.data, select = -c(symboling))
tree.data <- data.frame(new.data, biSym)


# Simple Classification Tree, no Model Evaluation
##################################################
# Generate tree using defaults
car.tree <- tree(biSym ~., data=tree.data)
summary(car.tree)

# Plot the tree and labels
plot(car.tree)
text(car.tree, pretty=0, cex=0.8, col='red')
title('Classification Tree')


# Model Validation
###################

set.seed(1)

# Training and Test sets
# Training set, sample 75% of the data
index <- sample(dim(new.data)[1], size=floor(dim(new.data)[1])*.75)
# Take remaining for test set
index.test <- setdiff(1:dim(new.data)[1], index)
length(index) + length(index.test) # 203, equal to number of obs.

# Our two sets:
train.set <- new.data[index,]
dim(train.set)

symbol.test <- biSym[index.test]
test.set <- new.data[index.test,]
dim(test.set)


# Model Validation
# Use training set and test set to evaluate our classification tree
car.tree <- tree(biSym ~., data=tree.data, subset=index)

# Predict on test set
tree.pred <- predict(car.tree, test.set, type='class')

# Confusion matrix
error <- table(tree.pred, symbol.test)

# Classification Error
1 - sum(diag(error))/sum(error)


# K-fold Cross Validation
##########################
# Note that Cross Validation automatically declares training and test sets
cv.car <- cv.tree(car.tree, FUN=prune.tree, K=10, method='misclass')
cv.car

# Plot
# Misclassification error vs. number nodes
plot(cv.car$size, cv.car$dev, type='b',
     xlab='Number of Nodes',
     ylab='CV Misclassification Error',
     col='red')

# Complexity and Min. Error
plot(cv.car$k, cv.car$dev, type='b',
     xlab='Complexity',
     ylab='CV Misclassification Error',
     col='blue')
# Get minimum error
min.error <- which.min(cv.car$dev) # index of min. error
abline(h = cv.car$dev[min.error], lty=2)
abline(v = cv.car$k[min.error], lty=2)
```

```r
# Prune tree
prune.car <- prune.misclass(car.tree, best=6)
plot(prune.car)
text(prune.car, pretty=0, col='red', cex=0.8)
title('Pruned Tree')


# Classification Tree using Random Forest
###########################################
fit <- randomForest(biSym ~., data=tree.data)
print(fit)

# Error = 5%, Random forest is a much better technique
importance(fit)

# Plot the bootstrapped trees
plot(fit)


# Classification with Boosting
################################
fit.boost <- ada(biSym ~., data=tree.data)
fit.boost

# Plot variable importance
varplot(fit.boost)
```