stat154_finalproject

Chase Enzweiler

11/14/2017

Final Project Stat 154

The goal of this project is to predict whether a person makes more than 50k a year.

## Preprocessing and Exploratory Data Analysis

First we want to load in our needed packages

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.3.2
```

```r
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```r
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.2
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.3.2
```

Now we load in our adult census data.

```r
# load in our training data
adult_data <- read.csv("~/Desktop/R_projects/Stat154_finalproject/adult.data.csv", header = FALSE)

# load in our test data
adult_test <- read.csv("~/Desktop/R_projects/Stat154_finalproject/adult.test.csv", header = FALSE)
```

We add column names to our data frames.

```
# name the columns
colnames(adult_data) <- c("age", "workclass", "fnlwgt", "education", "education.num", "marital.status",

colnames(adult_test) <- c("age", "workclass", "fnlwgt", "education", "education.num", "marital.status",
```

There are missing values in the data currently represented by " ¿‘. Data with missing values will be removed from both the training and the test data.

```
# set missing values to NA
adult_data[adult_data == " ?"] <- NA

adult_test[adult_test == " ?"] <- NA

# remove observations with missing values

adult_data <- na.omit(adult_data)

adult_test <- na.omit(adult_test)

# drop the unused level of " ?" from factor variables

adult_data <- droplevels(adult_data)

adult_test <- droplevels(adult_test)
```

Other issues we run into with our training and test data is that our response variable income has inconsistent spelling between our two datasets. In our training dataset income is represented by " <=50K" or " >50K" and in the testing dataset it is represented by " <=50K." and " >50K.“. Therefore we will change the class names in our testing data set to match our training dataset.

```
# change " <=50K." to " <=50K" etc.
adult_test$income <- ifelse(adult_test$income == " >50K.", " >50K", " <=50K")
```

Thankfully, classification tree based methods don't need too much data preprocessing. Classification trees should be consistent regarding scaling and can handle categorical variables without having to dummy code them. However, we notice that our predictor education-num and education are the same just one as an integer and one as a factor variable. Therefore we will drop the education variable from our data set and keep education-num.

```
# drop education and create a copy of the data we will use
train <- adult_data[,-4]
test <- adult_test[,-4]
```

We can take a look at the structure of our predictors and a summary of our predictors

```
str(train)
```

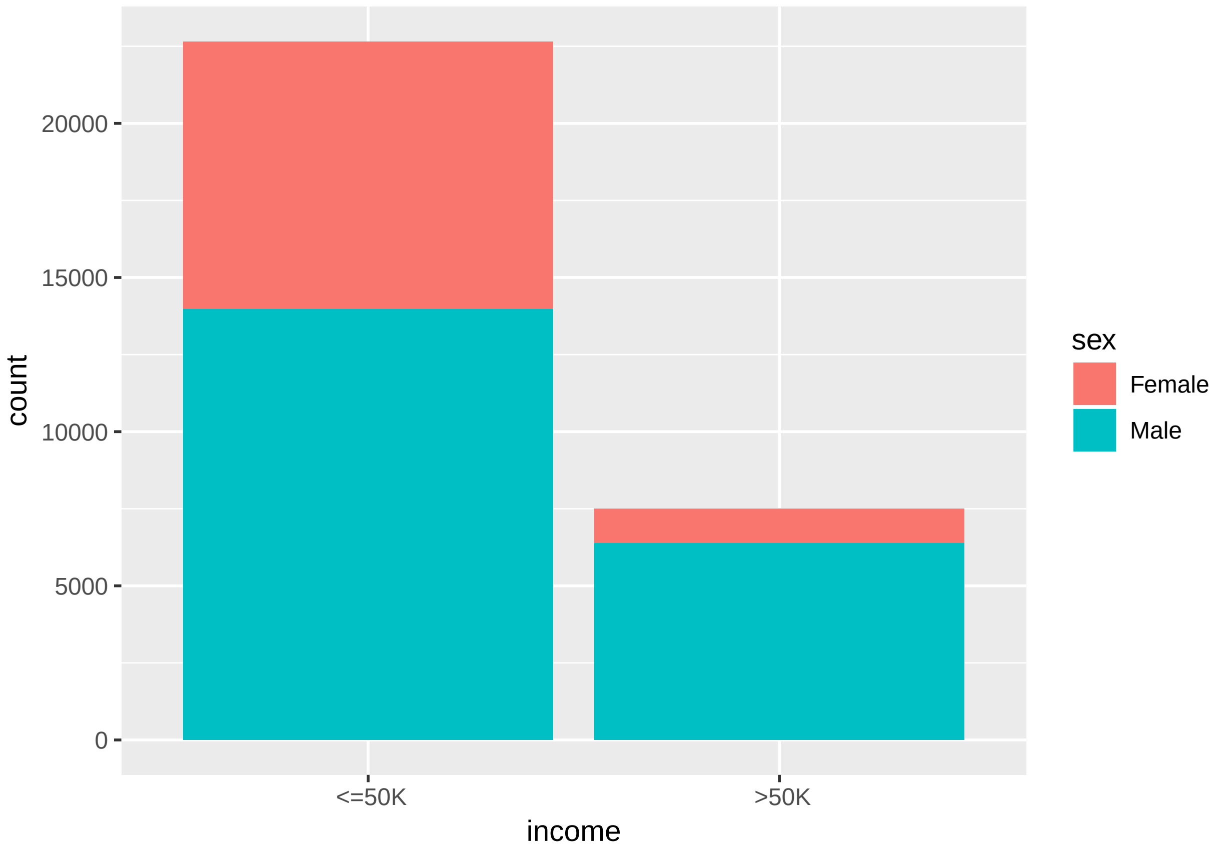
```
## 'data.frame':    30162 obs. of  14 variables:
##  $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
##  $ workclass     : Factor w/ 7 levels " Federal-gov",..: 6 5 3 3 3 3 3 5 3 3 ...
##  $ fnlwgt        : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
##  $ education.num : int  13 13 9 7 13 14 5 9 14 13 ...
##  $ marital.status: Factor w/ 7 levels " Divorced"," Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...
##  $ occupation    : Factor w/ 14 levels " Adm-clerical",..: 1 4 6 6 10 4 8 4 10 4 ...
##  $ relationship  : Factor w/ 6 levels " Husband"," Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
##  $ race          : Factor w/ 5 levels " Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
##  $ sex           : Factor w/ 2 levels " Female"," Male": 2 2 2 2 1 1 1 2 1 2 ...
##  $ capital.gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
##  $ capital.loss  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ hours.per.week: int  40 13 40 40 40 40 16 45 50 40 ...
##  $ native.country: Factor w/ 41 levels " Cambodia"," Canada",..: 39 39 39 39 5 39 23 39 39 39 ...
##  $ income        : Factor w/ 2 levels " <=50K"," >50K": 1 1 1 1 1 1 1 2 2 2 ...
```

```r
summary(train)
```

```
##       age                    workclass         fnlwgt       
##  Min.   :17.00    Federal-gov     :  943   Min.   :  13769  
##  1st Qu.:28.00    Local-gov       : 2067   1st Qu.: 117627  
##  Median :37.00    Private         :22286   Median : 178425  
##  Mean   :38.44    Self-emp-inc    : 1074   Mean   : 189794  
##  3rd Qu.:47.00    Self-emp-not-inc: 2499   3rd Qu.: 237628  
##  Max.   :90.00    State-gov       : 1279   Max.   :1484705  
##                   Without-pay     :   14                    
##  education.num                 marital.status            occupation  
##  Min.   : 1.00    Divorced             : 4214    Prof-specialty :4038  
##  1st Qu.: 9.00    Married-AF-spouse    :   21    Craft-repair   :4030  
##  Median :10.00    Married-civ-spouse   :14065    Exec-managerial:3992  
##  Mean   :10.12    Married-spouse-absent:  370    Adm-clerical   :3721  
##  3rd Qu.:13.00    Never-married        : 9726    Sales          :3584  
##  Max.   :16.00    Separated            :  939    Other-service  :3212  
##                   Widowed              :  827   (Other)         :7585  
##           relationship                   race           sex       
##   Husband       :12463    Amer-Indian-Eskimo:  286    Female: 9782  
##   Not-in-family : 7726    Asian-Pac-Islander:  895    Male  :20380  
##   Other-relative:  889    Black             : 2817                  
##   Own-child     : 4466    Other             :  231                  
##   Unmarried     : 3212    White             :25933                  
##   Wife          : 1406                                              
##                                                                     
##   capital.gain    capital.loss     hours.per.week         native.country 
##  Min.   :    0   Min.   :   0.00   Min.   : 1.00    United-States:27504  
##  1st Qu.:    0   1st Qu.:   0.00   1st Qu.:40.00    Mexico       :  610  
##  Median :    0   Median :   0.00   Median :40.00    Philippines  :  188  
##  Mean   : 1092   Mean   :  88.37   Mean   :40.93    Germany      :  128  
##  3rd Qu.:    0   3rd Qu.:   0.00   3rd Qu.:45.00    Puerto-Rico  :  109  
##  Max.   :99999   Max.   :4356.00   Max.   :99.00    Canada       :  107  
##                                                    (Other)       : 1516  
##     income     
##   <=50K:22654  
##   >50K : 7508  
##                
##                
##                
##                
## 
```

From the summarized data we can see that our sample predominantly makes less than 50k. This tells us that our data is imbalanced as there are far more observations with income less than or equal to 50K, 22654,than there are observations that make greater than 50K ,only 7508. This could prove to be a problem when we fit some of our classifiers as our classifiers might become bias towards the class " <=50K". However for this project we are only concerned with the overall accuracy of our classifiers and not the accuracy for each particular class   Now we can take a quick look at the relationships between some of our predictors and our response

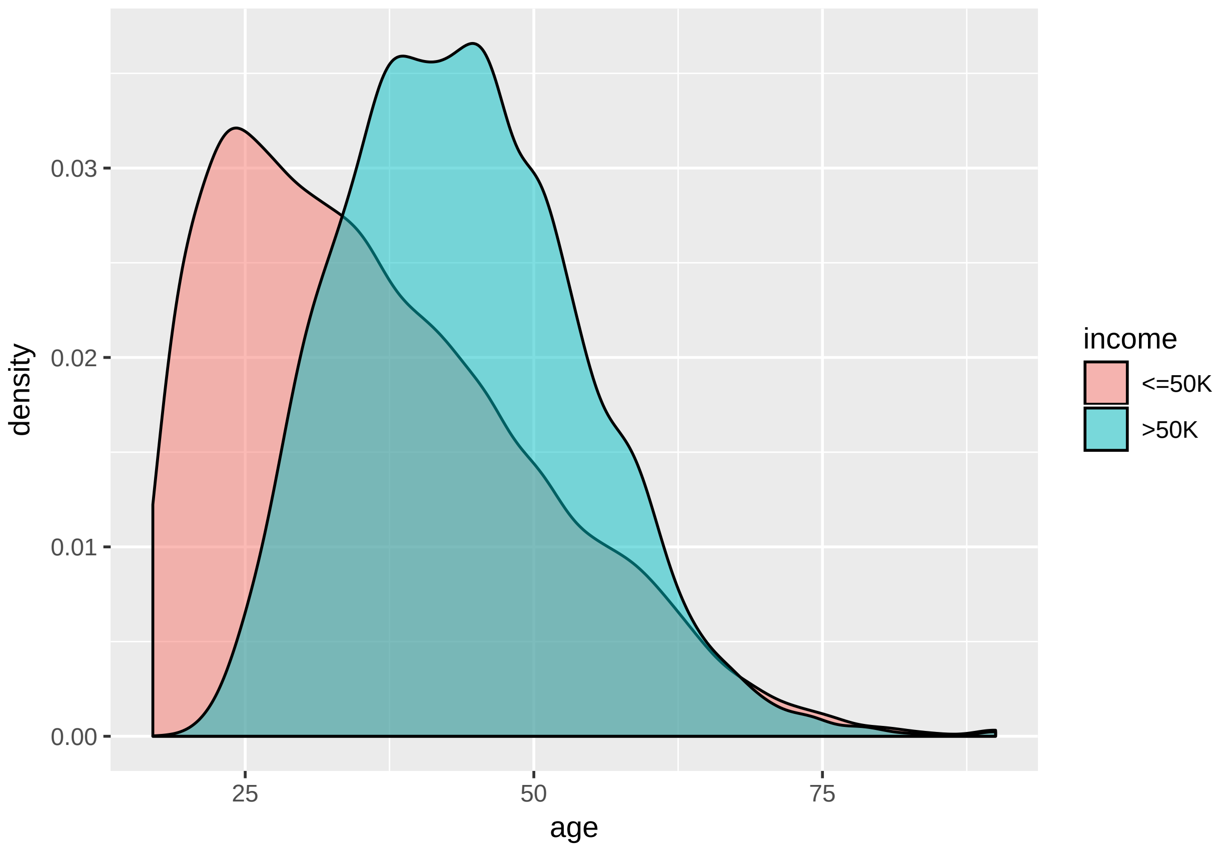
```
# check relationship between sex and income
ggplot(train, aes(income, fill = sex)) + geom_bar()
```



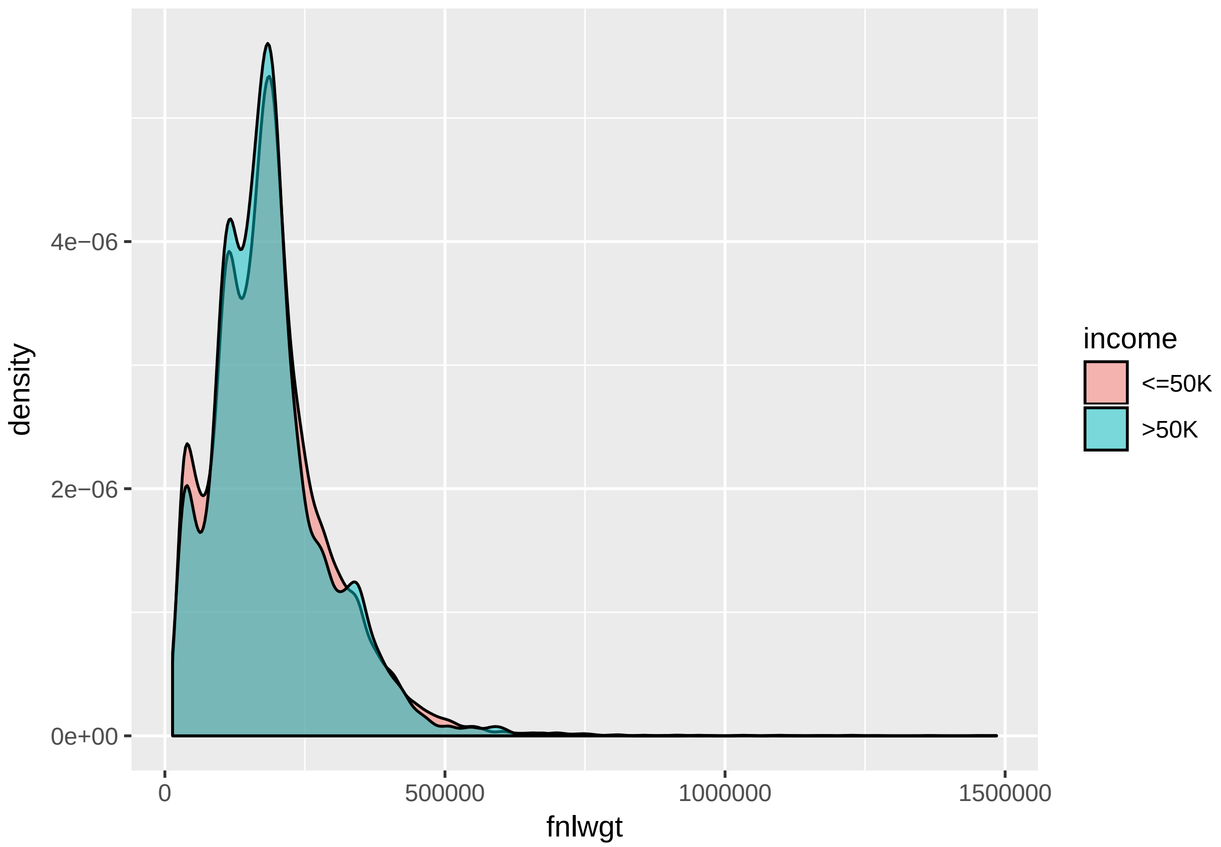
We see that overall from predictor sex the level "Male" is more likely to make more than 50k than "Female".

We can also take a look at the densities of a few predictors with respect to the income class.

```
# age density
ggplot(train, aes(x = age, fill = income)) + geom_density(alpha = .5)
```



```r
ggplot(train, aes(x = fnlwgt, fill = income)) + geom_density(alpha = .5)
```



From looking at the densities we can see that younger people are more likely to make less than 50k than older people and the distribution of age with in come less than 50k is skewed right. Also we see that the densitites of fnlwgt for each income class are very similiar and fnlwgt may not be a strong predictor of a persons income.

We can also check out the correlations among numerical predictors.

```r
# correlation matrix
cor(train[,c(1, 3, 4, 10,11,12)])
```

```
##                        age        fnlwgt education.num  capital.gain
## age             1.00000000 -0.0765108361    0.04352609  0.0801542263
## fnlwgt         -0.07651084  1.0000000000   -0.04499174  0.0004215674
## education.num   0.04352609 -0.0449917421    1.00000000  0.1244159953
## capital.gain    0.08015423  0.0004215674    0.12441600  1.0000000000
## capital.loss    0.06016548 -0.0097495278    0.07964641 -0.0322293265
## hours.per.week  0.10159876 -0.0228857516    0.15252207  0.0804318007
##                capital.loss hours.per.week
## age             0.060165480     0.10159876
## fnlwgt         -0.009749528    -0.02288575
## education.num   0.079646410     0.15252207
## capital.gain   -0.032229327     0.08043180
## capital.loss    1.000000000     0.05241705
## hours.per.week  0.052417049     1.00000000
```

We see that there is little correlation among our numerical predictors.

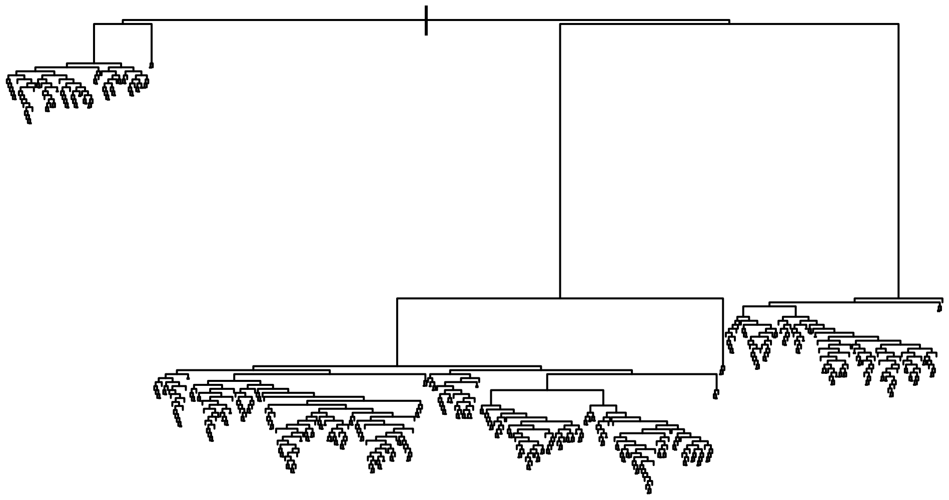
Building a Classification Tree

Using the R package "rpart" we will fit a classification tree.

```
# fit a classification tree to the training data, grown to maximmum depth
set.seed(1)

class_tree <- rpart(income ~ ., data = train, method = "class", control = rpart.control(minsplit = 20,
```

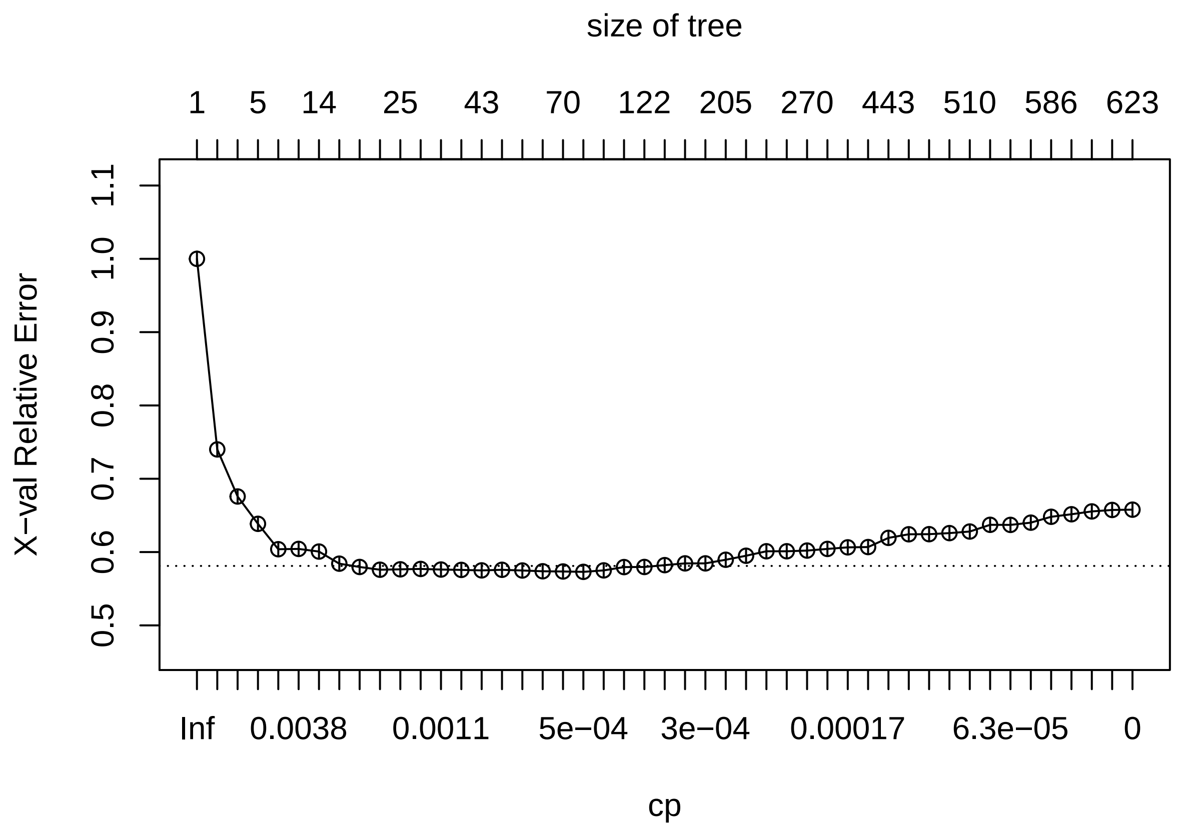
Since we grew our classification tree to its maximum depth, we expect that our tree will be extremely large. However, we can still plot and view our tree although we will leave out the labels so that the tree is still visible.

```
# full classification tree
plot(class_tree)
```



Now that we have grown a large classification tree we have to prune the tree to avoid overfitting our data. We then look for a subtree with the lowest cross-validation error and find the cost complexity parameter that is associated with the smallest cross-validation error.

```
# can plot the relative error against the cp to visualize amount of splits in best subtree
plotcp(class_tree)
```



```r
# table containing cross validation errors and cp
printcp(class_tree)
```

```
##
## Classification tree:
## rpart(formula = income ~ ., data = train, method = "class", control = rpart.control(minsplit = 20,
##     cp = 0))
##
## Variables actually used in tree construction:
##  [1] age            capital.gain   capital.loss   education.num
##  [5] fnlwgt         hours.per.week marital.status native.country
##  [9] occupation     race           relationship   sex
## [13] workclass
##
## Root node error: 7508/30162 = 0.24892
##
## n= 30162
##
##             CP nsplit rel error  xerror      xstd
## 1   1.2999e-01      0   1.00000 1.00000 0.0100018
## 2   6.4198e-02      2   0.74001 0.74001 0.0089670
## 3   3.7294e-02      3   0.67581 0.67581 0.0086527
## 4   5.0613e-03      4   0.63852 0.63852 0.0084574
## 5   4.3953e-03      9   0.60202 0.60389 0.0082669
## 6   3.2854e-03     10   0.59763 0.60429 0.0082692
## 7   3.1966e-03     13   0.58777 0.60069 0.0082489
## 8   2.2199e-03     17   0.57352 0.58418 0.0081543
## 9   1.7315e-03     20   0.56686 0.57965 0.0081280
## 10 1.3319e-03     22   0.56340 0.57592 0.0081062
## 11 1.1987e-03     24   0.56074 0.57645 0.0081093
## 12 1.0655e-03     27   0.55714 0.57698 0.0081125
## 13 1.0389e-03     28   0.55607 0.57619 0.0081078
## 14 9.3234e-04     35   0.54875 0.57565 0.0081047
## 15 6.9259e-04     42   0.54222 0.57499 0.0081007
## 16 6.8815e-04     51   0.53396 0.57579 0.0081054
## 17 6.6596e-04     58   0.52810 0.57485 0.0081000
## 18 5.9936e-04     65   0.52278 0.57379 0.0080937
## 19 5.3277e-04     69   0.52038 0.57352 0.0080921
## 20 4.6617e-04     78   0.51558 0.57299 0.0080890
## 21 3.9957e-04     93   0.50839 0.57499 0.0081007
## 22 3.6628e-04    115   0.49907 0.57952 0.0081272
## 23 3.3298e-04    121   0.49627 0.57965 0.0081280
## 24 3.1078e-04    143   0.48815 0.58218 0.0081427
## 25 2.9968e-04    150   0.48575 0.58458 0.0081566
## 26 2.9598e-04    154   0.48455 0.58458 0.0081566
## 27 2.6638e-04    204   0.46137 0.58950 0.0081851
## 28 2.2199e-04    243   0.45045 0.59497 0.0082163
## 29 2.1311e-04    252   0.44832 0.60109 0.0082511
## 30 2.1089e-04    257   0.44726 0.60109 0.0082511
## 31 1.9979e-04    269   0.44473 0.60202 0.0082564
## 32 1.7759e-04    315   0.43447 0.60429 0.0082692
## 33 1.6649e-04    340   0.42981 0.60642 0.0082812
## 34 1.3319e-04    344   0.42914 0.60682 0.0082834
## 35 1.1099e-04    442   0.41489 0.61934 0.0083530
## 36 1.0655e-04    448   0.41422 0.62427 0.0083801
## 37 9.9893e-05    463   0.41263 0.62440 0.0083808
## 38 8.8794e-05    494   0.40823 0.62587 0.0083889
## 39 7.9915e-05    509   0.40690 0.62800 0.0084005
## 40 6.6596e-05    514   0.40650 0.63719 0.0084503
## 41 6.0541e-05    554   0.40370 0.63705 0.0084495
## 42 4.4397e-05    565   0.40304 0.64012 0.0084660
## 43 3.3298e-05    585   0.40210 0.64811 0.0085086
## 44 2.9598e-05    593   0.40184 0.65157 0.0085269
## 45 2.6638e-05    606   0.40117 0.65543 0.0085472
## 46 2.2199e-05    616   0.40091 0.65743 0.0085577
## 47 0.0000e+00    622   0.40077 0.65783 0.0085598
```

```r
# obtain the optimal cost complexity parameter

# returns named int, so use as.integer() to get just int
cv_index <- as.integer(which.min(class_tree$cptable[,"xerror"]))

class_best_cp <- class_tree$cptable[20,"CP"]

# optimal cp
class_best_cp
```

```
## [1] 0.0004661694
```

Using our optimal cost complexity parameter, we can now prune our tree to get the lowest test error rate that we estimated using the cross validation error.

```r
# prune the tree
class_tree_prune <- prune(class_tree, class_best_cp)
```

Variable importance is measured by the sum of goodness of split. Since our tree was grown using the Gini index to determine the goodness of split, the importance of each of our variables is then determined by the sum of the amount that the Gini index is decreased by splits over that variable.

```
# variable importance statistic output
class_tree_prune$variable.importance
```

```
##   relationship marital.status  education.num   capital.gain     occupation
##    2313.364626    2252.680475    1136.866505    1096.591867     972.760153
##            sex            age hours.per.week   capital.loss native.country
##     768.345034     693.672458     403.041274     324.070096      93.539787
##      workclass         fnlwgt           race
##      79.591378      36.568470       6.848578
```

Our 5 most important variables are therefore relationship, marital.status, education.num, capital.gain, and occupation.   We can then calculate the training accuracy rate of our model.

```
# predict the pruned tree (class_tree_prune) on the training data
set.seed(1)
train_tree_pred <- predict(class_tree_prune, newdata = train)

head(train_tree_pred)
```

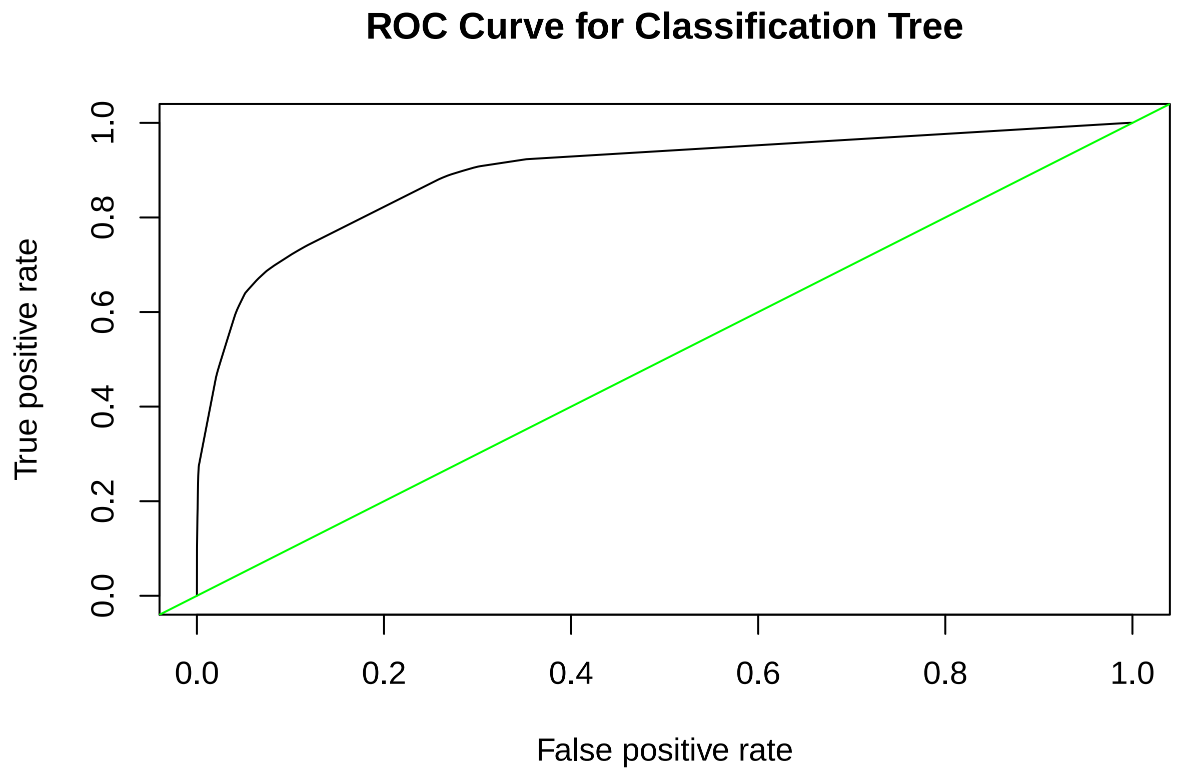
```
##       <=50K       >50K
## 1 0.9619168 0.03808321
## 2 0.6239316 0.37606838
## 3 0.9619168 0.03808321
## 4 0.9098361 0.09016393
## 5 0.2500000 0.75000000
## 6 0.2284409 0.77155911
```

```
# we want 1 for >50k and 0 for <= 50k

# we select the second column here corresponding to probabilities that income is >50k

train_tree_pred <- train_tree_pred[,2]

# create a logical vector that is true when the estimated probability of income >50k is > .5

train_tree_pred_tf <- train_tree_pred > .5

# original observations class_tree_prune$y are classified as 1 for <= 50k and 2 for >50k so we subtract

training_acc_tree_prune <- mean(train_tree_pred_tf == (class_tree_prune$y - 1))

# Training Accuracy for pruned classification tree

training_acc_tree_prune
```

```
## [1] 0.8716597
```

The training accuracy of our pruned classification tree(class_tree_prune) is about 87 percent. We can also display the ROC curve for our classification tree which will show the true positive and false positive rates over all possible cutoffs/thresholds.

```r
# ROCR prediction object
class_prediction <- prediction(train_tree_pred, class_tree_prune$y)

# plot roc curve

class_roc <- performance(class_prediction, measure = "tpr", x.measure = "fpr")

plot(class_roc, main = "ROC Curve for Classification Tree")
abline(a = 0, b = 1, col = "green")
```



```r
# report the area under curve

class_auc <- performance(class_prediction, measure = "auc")

class_auc@y.values
```

```
## [[1]]
## [1] 0.8929411
```

Our ROC curve for our classification tree shows that our tree performs much better on our data than a random classifier would. A random classifier given by the green line in our ROC curve gives an area under curve (AUC) of .5 where our classification tree performs much better with area under the curve of AUC = .8929411.

One note I would like to make however is that we have a good training accuracy rate at around 87 percent but our classification tree is more accurate classifying observations that income is <= 50K rather than >50K. For example we can look at the confusion matrix where the left side of the table is predicted class with 0 equal to <= 50K and 1 equal to >50K.

```r
# confusion matrix
predictions_of_class <- ifelse(train_tree_pred_tf, " >50K", " <=50K")
```

```
table(predictions_of_class, ifelse(class_tree_prune$y-1 == 0, " <=50K", " >50K"))
```

```
##
## predictions_of_class  <=50K  >50K
##                <=50K  21488  2705
##                >50K    1166  4803
```

Therefore to get better predictions of >50K we can change our cutoff/threshold values, although this will affect our overall training accuracy. We could also, as mentioned earlier, train this classifier on set that is less imbalanced than our current training data set. However, we get a good enough training accuracy rate for our classification tree that we are ok with more error classifying our " >50K" class to have a lower overall training accuracy rate.

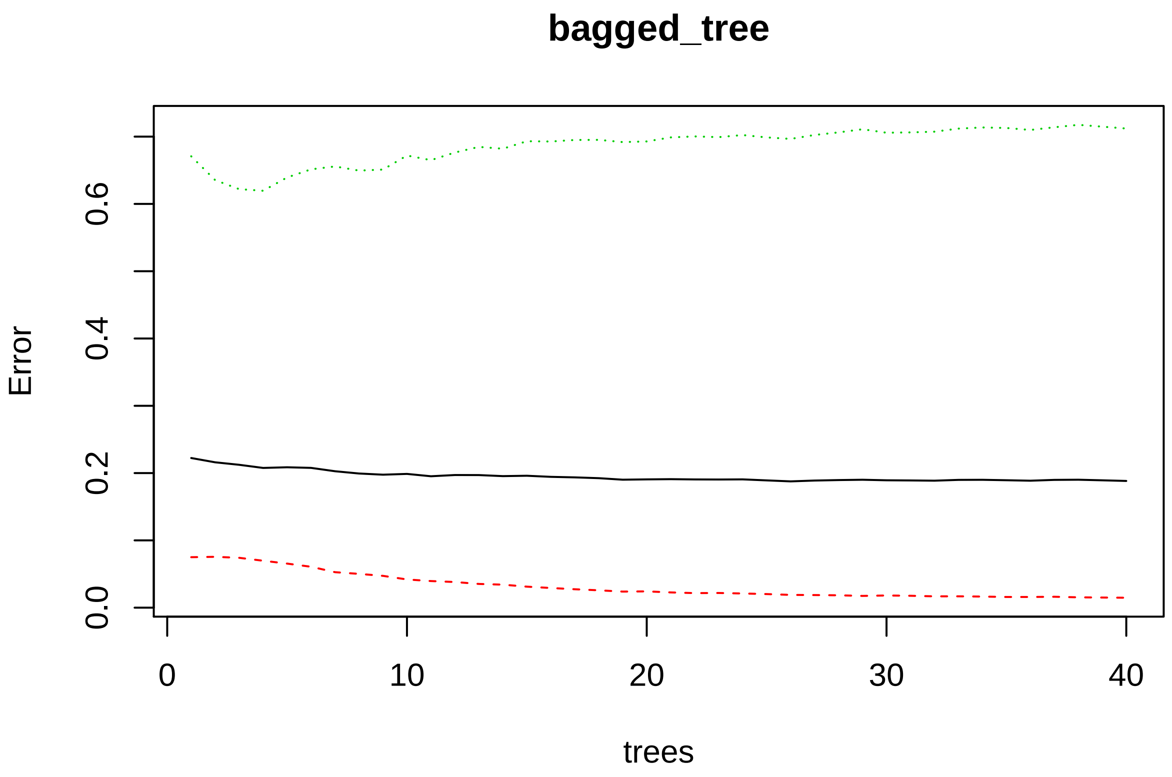
Building a Bagged Tree

Using the package randomForest we can fit a bagged tree to our training data. The bagged tree is made by fitting trees on bootstrapped training sets. The parameter to choose is the number of trees we want constructed (ntree), but since the bagged tree will not overfit the data as the parameter ntree grows we will choose a number of trees (ntree) that is not to computationally expensive.

```
# fit a bagged tree
set.seed(1)
bagged_tree <- randomForest(income ~ ., data = train, ntree = 40, mtry = 13, importance = TRUE)

bagged_tree
```

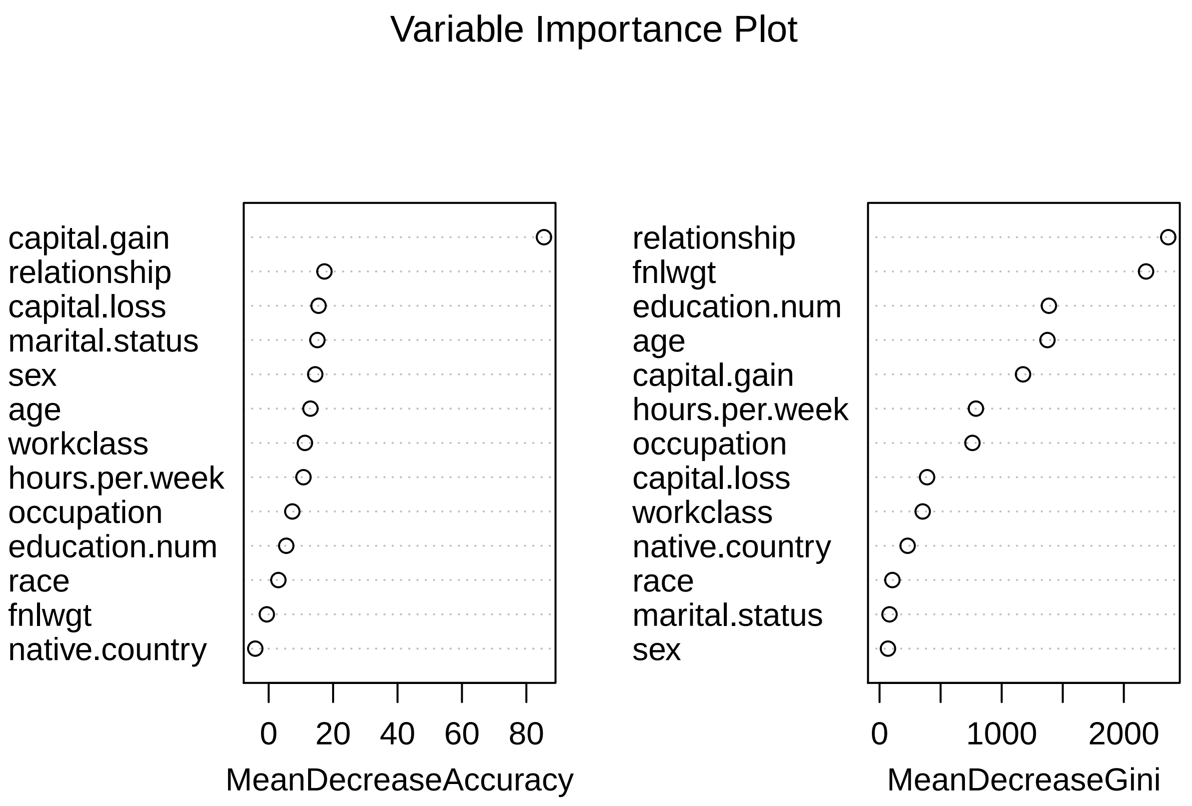
```
##
## Call:
##  randomForest(formula = income ~ ., data = train, ntree = 40,      mtry = 13, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 40
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 18.83%
## Confusion matrix:
##         <=50K  >50K class.error
##  <=50K  22320   334  0.01474353
##  >50K    5346  2162  0.71204049
```

```
# plot the error with respect to the number of trees
plot(bagged_tree)
```



The plot above shows the error rates for both classes with green being the error rate for class >50K and the red being the error rate for class <= 50K. The black solid line represents the out-of-bag error estimation. We see that the out-of-bag error levels off at about 40 trees, so we find that 40 trees is an acceptable amount for this model. We can now check which variables are the most important

```r
# variable importance
varImpPlot(bagged_tree, main = "Variable Importance Plot")
```



```
bagged_tree$importance
```

```
##                        <=50K          >50K MeanDecreaseAccuracy
## age             6.739774e-03  1.406715e-02         0.0085726662
## workclass       2.522354e-03  4.800002e-03         0.0030853724
## fnlwgt         -2.089579e-05 -6.888206e-04        -0.0001917342
## education.num   9.728082e-03 -1.045774e-03         0.0070237850
## marital.status  3.122695e-02 -4.388541e-03         0.0223269121
## occupation      9.358773e-03  5.952545e-03         0.0084970019
## relationship    3.964567e-02 -1.257459e-02         0.0266025719
## race            4.506922e-04  6.438003e-05         0.0003538572
## sex             4.276518e-03 -8.569389e-04         0.0029936530
## capital.gain    3.705100e-02  1.119234e-01         0.0557652189
## capital.loss    5.548903e-03  3.188626e-02         0.0121244157
## hours.per.week  5.073945e-03  6.319634e-03         0.0053847417
## native.country -2.247311e-04 -2.153804e-04        -0.0002228718
##                MeanDecreaseGini
## age                  1374.78728
## workclass             353.11235
## fnlwgt               2181.99691
## education.num        1386.50982
## marital.status         81.50234
## occupation            760.47490
## relationship         2363.81505
## race                  105.08698
## sex                    68.83435
## capital.gain         1174.75966
## capital.loss          389.06701
## hours.per.week        789.07421
## native.country         229.75037
```

Paying attention to the mean decrease in Gini index for each variable in our variable importance plot we see that our most important variables are relationship, fnlwgt, education.num, age, capital.gain, hours.per.week, and occupation with mean decrease of Gini 2363.81505, 2181.99691, 1386.50982, 1374.78728, 1174.75966, 789.07421, and 760.47490 respectively.

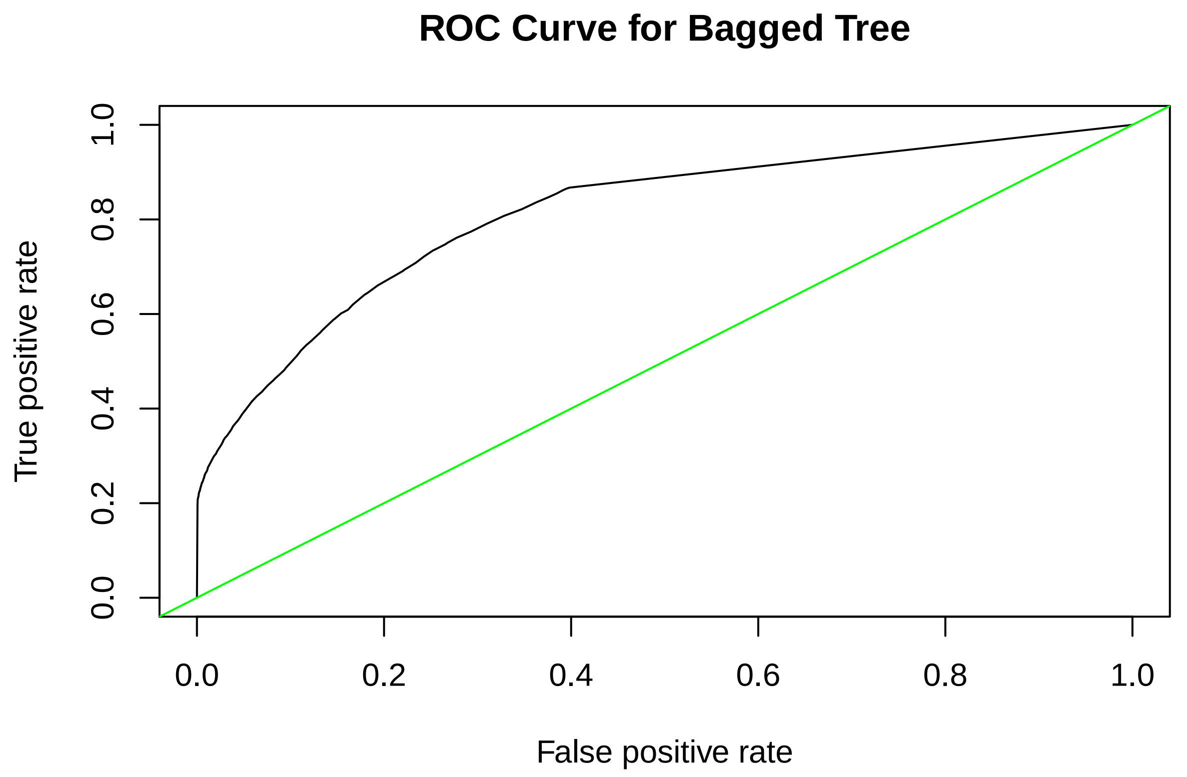
Now to calculate the training accuracy rate.

```
# generate predictions on the training set
set.seed(1)
bagged_train_pred <- predict(bagged_tree, newdata = train)

# training accuracy
training_acc_bagged <- bagged_train_pred == bagged_tree$y

mean(training_acc_bagged)
```

```
## [1] 0.8630064
```

The training accuracy rate of our bagged tree is about 86 percent which is very close to our training accuracy rate for our classification tree. The training accuracy of our bagged tree is around 1 percent less accurate than our classification tree. We will also display the ROC curve for our bagged tree.

```
# ROCR prediction object

bagged_prediction <- prediction(bagged_tree$votes[,2], bagged_tree$y)

# plot roc curve

bagged_roc <- performance(bagged_prediction, measure = "tpr", x.measure = "fpr")

plot(bagged_roc, main = "ROC Curve for Bagged Tree")
abline(a = 0, b = 1, col = "green")
```



```r
# report the area under curve

bagged_auc <- performance(bagged_prediction, measure = "auc")

bagged_auc@y.values
```

```
## [[1]]
## [1] 0.8126158
```

The ROC curve above was created using proportions from the votes of each of the trees in the bagged tree. Our area under curve(AUC) value is good and is much better than the "no information" classifier which is represented by the green line with AUC = .5. The AUC of our bagged tree is 0.8126158.

Lastly just to get a better idea on how our bagged tree is performing on our training set we can take a look at the confusion matrix for our bagged tree on our training set.

```r
# confusion matrix
bagged_tree_predictions <- bagged_train_pred
table(bagged_tree_predictions, bagged_tree$y)
```

```
##
## bagged_tree_predictions  <=50K  >50K
##                    <=50K  22639  4117
##                    >50K      15  3391
```

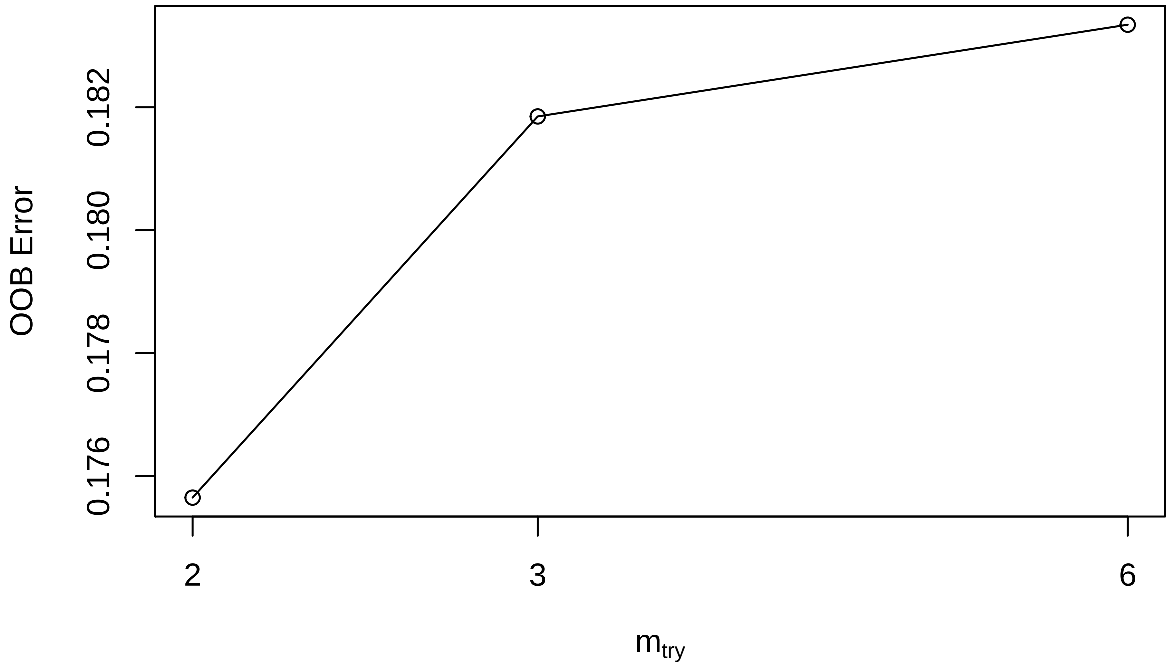
Looking at the confusion matrix for our bagged tree on the training data set we see that our bagged tree performs extremely well correctly classifying observations that have income <=50K, but performs poorly trying to classify observations with income >50K. Some of this bias towards the class " <=50K" could be due to the fact that the bagged tree was trained on an imbalanced training set. However, we still get a good overall training accuracy at about 86 percent just around 1 percent worse compared to our classification tree.

Building a Random Forest

We can now use the R package randomForest to fit a random forest to our training data. Random forests are similiar to bagged trees except random forests decorrelate the bootstrapped trees by restricting the variables available to perform each split in each tree. Usually the amount of random variables selected is restricted to the square root of the total amount of variables, but here we will consider whichever restriction is optimal. To find the optimal restriction of randomly selected variables we can use the function tuneRF() from the randomForest package. tuneRF will search for the optimal amount of variables to choose from for each split of the random forest and provide the out-of-bag errors for each amount used.

```
# find optimal mtry parameter
set.seed(1)
tuneRF(train[,-14], train$income)
```

```
## mtry = 3  OOB error = 18.19%
## Searching left ...
## mtry = 2     OOB error = 17.57%
## 0.03409298 0.05
## Searching right ...
## mtry = 6     OOB error = 18.33%
## -0.008204193 0.05
```



```
##       mtry  OOBError
## 2.OOB    2 0.1756515
## 3.OOB    3 0.1818513
## 6.OOB    6 0.1833433
```

from our tuneRF results we see that our optimal amount of variables (mtry paramter in randomforest) is 2. Using mtry = 2 gives us the lowest out-of-bag error with .1756515. Therefore we will fit our random forest classifier with parameter mtry = 2.

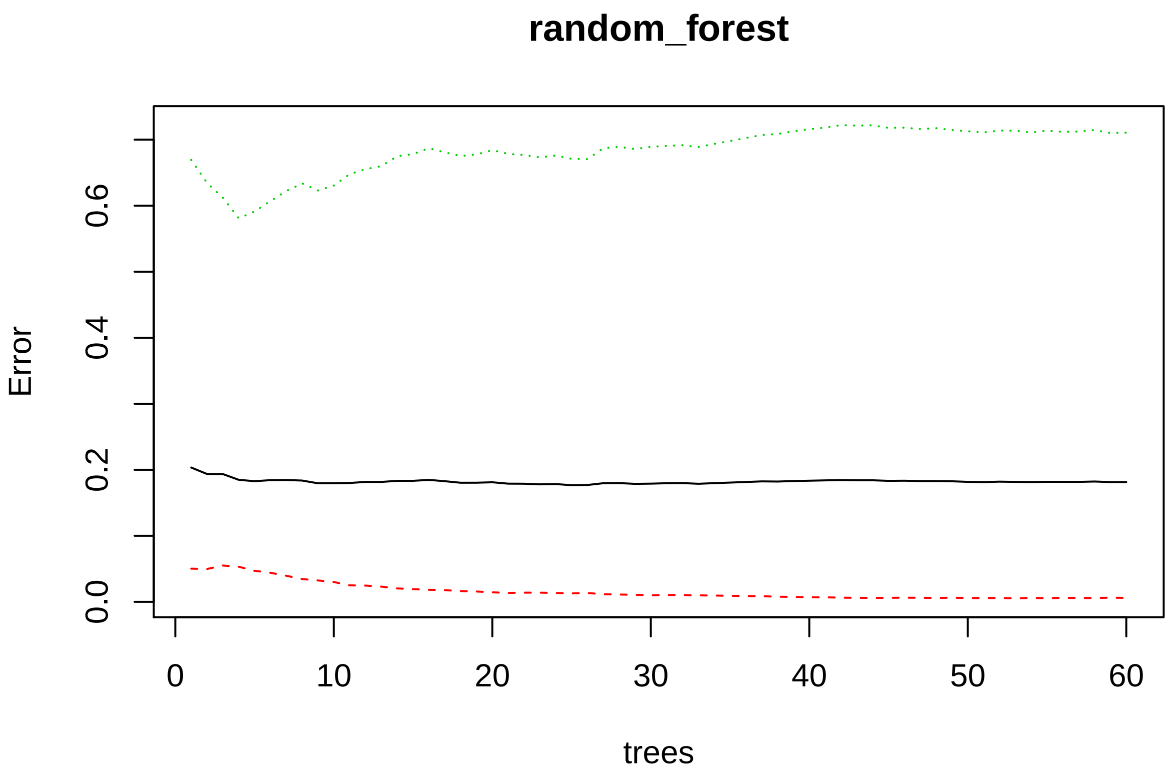
```
# fit our random forest classifier
set.seed(1)
random_forest <- randomForest(income ~ ., data = train, mtry = 2, ntree = 60, importance = TRUE)
```

```
random_forest
```

```
##
## Call:
##  randomForest(formula = income ~ ., data = train, mtry = 2, ntree = 60,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 60
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 18.14%
## Confusion matrix:
##         <=50K  >50K class.error
##  <=50K  22519   135 0.005959213
##  >50K    5335  2173 0.710575386
```

We fit our random forest with a total of 60 trees. We can then plot the error of our random forest to determine whether 60 trees is sufficient for our model.

```
# plot random forest oob error
plot(random_forest)
```

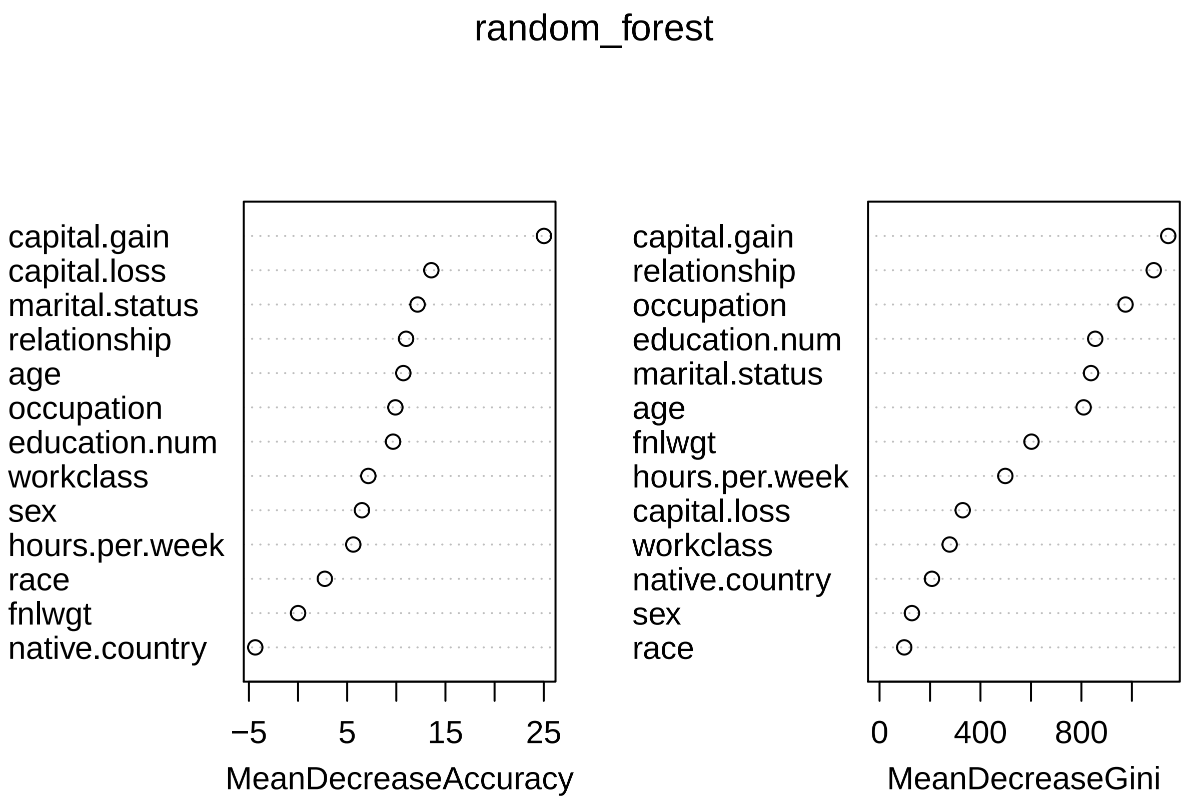


We see that the solid black line representing the out-of-bag error rate remains level from about the 40 tree mark and therefore 60 trees will be sufficient for our random forest. We can now check which variables in our random forest are most important.

```
# variable importance measures
random_forest$importance
```

```
##                        <=50K          >50K MeanDecreaseAccuracy
## age             0.0027021474  0.0162993736         6.072569e-03
## workclass       0.0016237211  0.0061251882        2.732272e-03
## fnlwgt          0.0001766804 -0.0005264667        8.646398e-07
## education.num   0.0063759714  0.0315064468        1.258682e-02
## marital.status  0.0231812829  0.0204169892        2.247360e-02
## occupation      0.0061558742  0.0338480521        1.302099e-02
## relationship    0.0219496393  0.0118670094        1.944172e-02
## race            0.0003391067  0.0007982527        4.519065e-04
## sex             0.0057133028  0.0005274600        4.434495e-03
## capital.gain    0.0278964571  0.0780089837        4.031618e-02
## capital.loss    0.0034818400  0.0220595778        8.095835e-03
## hours.per.week  0.0007735414  0.0124978873        3.688931e-03
## native.country -0.0001197369 -0.0003251742       -1.712402e-04
##                MeanDecreaseGini
## age                   808.82208
## workclass             277.87435
## fnlwgt                602.08867
## education.num         854.73979
## marital.status        837.99294
## occupation            974.86970
## relationship         1086.72896
## race                   97.79911
## sex                   128.21681
## capital.gain         1144.10785
## capital.loss          329.57609
## hours.per.week        498.38253
## native.country        207.51235
```

```
varImpPlot(random_forest)
```



We consider important variables to be those that have largest decrease in Gini index averaged across all trees. Therefore using the measure of mean decrease in Gini, our most important variables in our random forest are capital.gain, relationship, occupation, education.num, marital.status, and age each with mean decrease of Gini index of 1144.10785, 1086.72896, 974.86970, 854.73979, 837.99294, and 808.82208 respectively.

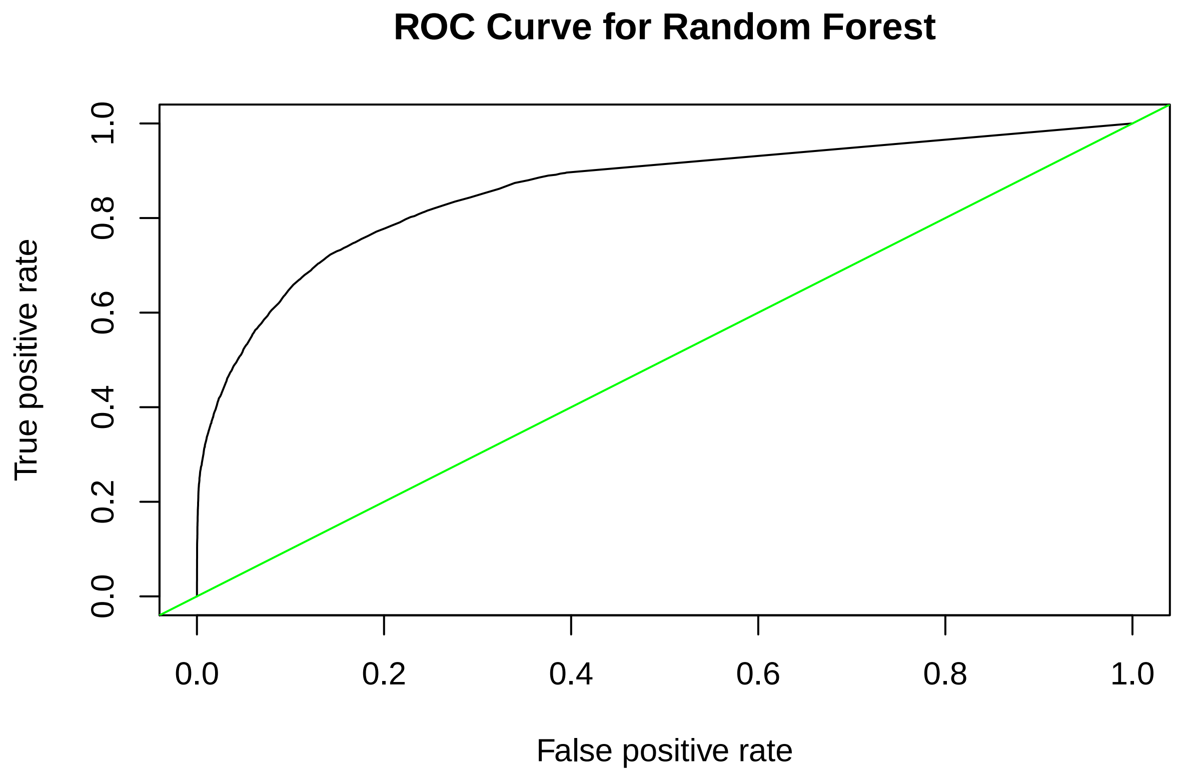
Now we can calculate the training accuracy rate using our random forest classifier.

```
# training accuracy rate

rf_pred <- predict(random_forest, newdata = train)

mean(rf_pred == random_forest$y)
```

```
## [1] 0.8275976
```

Our random forest has a training accuracy rate of about 82.75645 percent which is the lowest of all our classifiers. We can also display the ROC curve for our random forest.

```
# prediction object
rf_prediction <- prediction(random_forest$votes[,2], random_forest$y)

# plot roc curve

rf_roc <- performance(rf_prediction, measure = "tpr", x.measure = "fpr")

plot(rf_roc, main = "ROC Curve for Random Forest")
abline(a = 0, b = 1, col = "green")
```



```
# report the area under curve

rf_auc <- performance(rf_prediction, measure = "auc")
```

```
rf_auc@y.values
```

```
## [[1]]
## [1] 0.8606042
```

Dispayling the Roc curve we see that our random forest performs much better than would a "no information" classifier represented by the green line. The area under the curve for our random forest is 0.8606042. We can construct a confusion matrix that will give us an idea of how well our classifier classifies observations of a certain class. We see below that our random forest classifier trained on the training set (train) is near perfect classifying observations with income less than 50K and is bad at classifying observations who's true income is greater than 50K.

```
# confusion matrix
random_forest_predictions <- rf_pred
table(random_forest_predictions, random_forest$y)
```

```
##
## random_forest_predictions  <=50K  >50K
##                     <=50K  22635  5181
##                     >50K      19  2327
```

Looking at our random forest confusion matrix we see that maybe training our random forest on an imbalanced training data set caused our classifier to be bias towards the class " <=50K". We didn't care about having relatively high error rates for the class" >50K" before in our other classifiers because the decrease in error for our other class was good enough to offset our high errors and still gave us good training accuracy for our other two classifiers. However with our random forest, the high error rates for " >50K" are not being offset enough with low error rates for the other class and the training accuracy takes a big hit. The training accuracy for our random forest is 82.75976 which is lower than all our other classifiers.

A way we can combat our random forest's high bias towards the class " <=50K" is to train our random forest on a balanced training dataset and see if we can improve our training accuracy rate for our random forest.

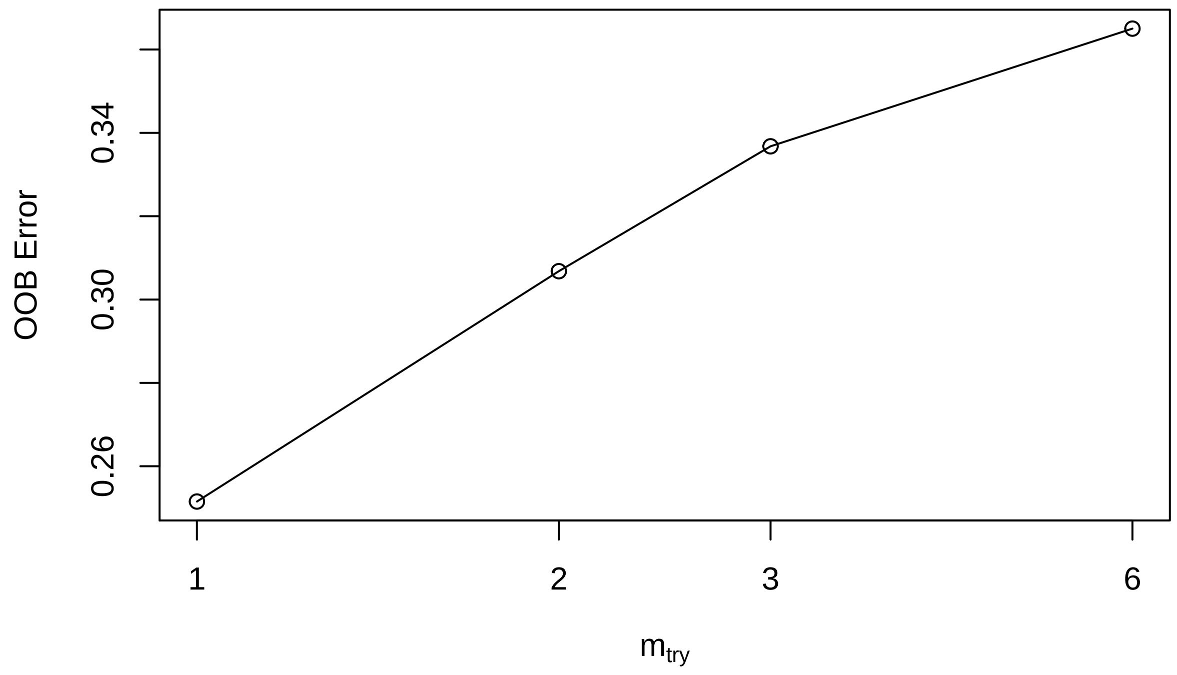
We will create a new training data set by keeping observations with income " >50K" and sampling from the remaining observations with income "<=50K" to create a new dataset that has equal amounts of observations with each income class.

```
# create new undersampled training set
set.seed(1)

less_index <- which(train$income == " <=50K")
greater_index <- which(train$income == " >50K")

# sample from the class with way more observations
samp <- sample(less_index, size = length(greater_index))

# the dataset
undersampled <- train[append(samp, greater_index),]
```

Now we can retrain a random forest on this new balanced training dataset. Similiar as above we search for an optimal amount of variables to try at each split.

```
# tune mtry for new balanced random forest
set.seed(1)
tuneRF(undersampled[,-14], undersampled$income)
```

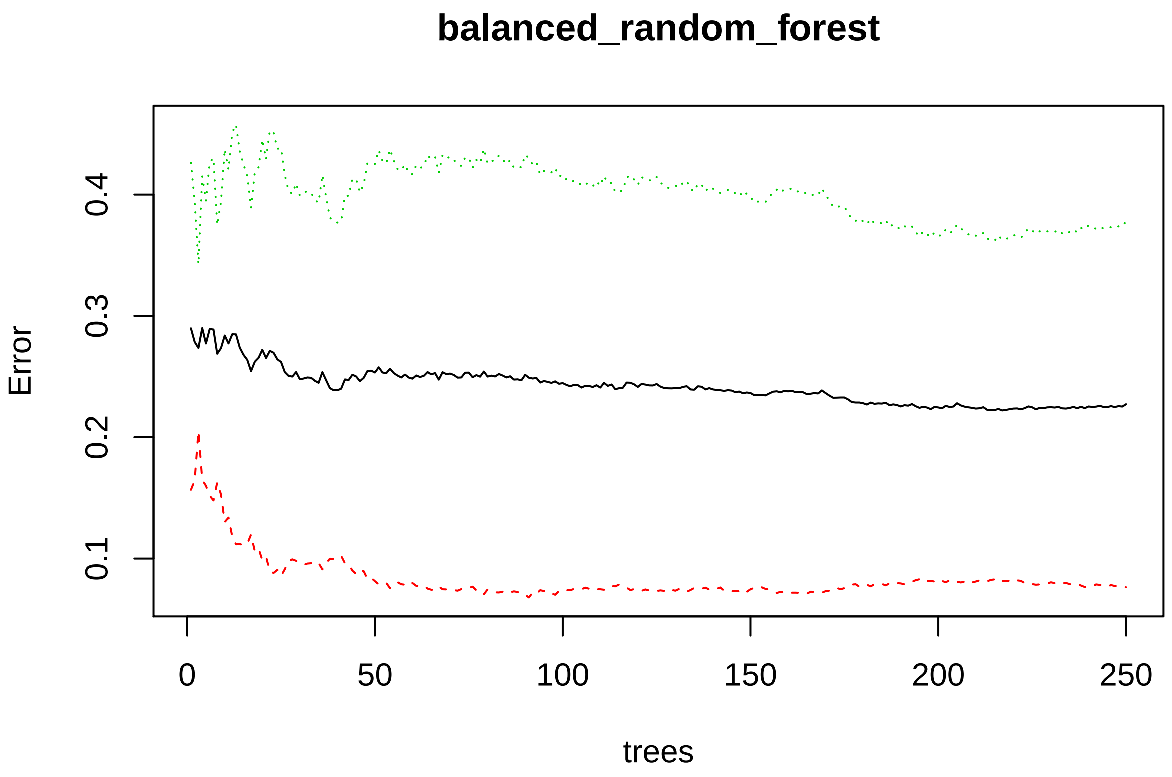
```
## mtry = 3  OOB error = 33.68%
## Searching left ...
## mtry = 2     OOB error = 30.68%
## 0.08898556 0.05
## mtry = 1     OOB error = 25.15%
## 0.1801606 0.05
## Searching right ...
## mtry = 6     OOB error = 36.5%
## -0.4511517 0.05
```



```
##       mtry  OOBError
## 1.OOB    1 0.2515317
## 2.OOB    2 0.3068061
## 3.OOB    3 0.3367741
## 6.OOB    6 0.3650107
```

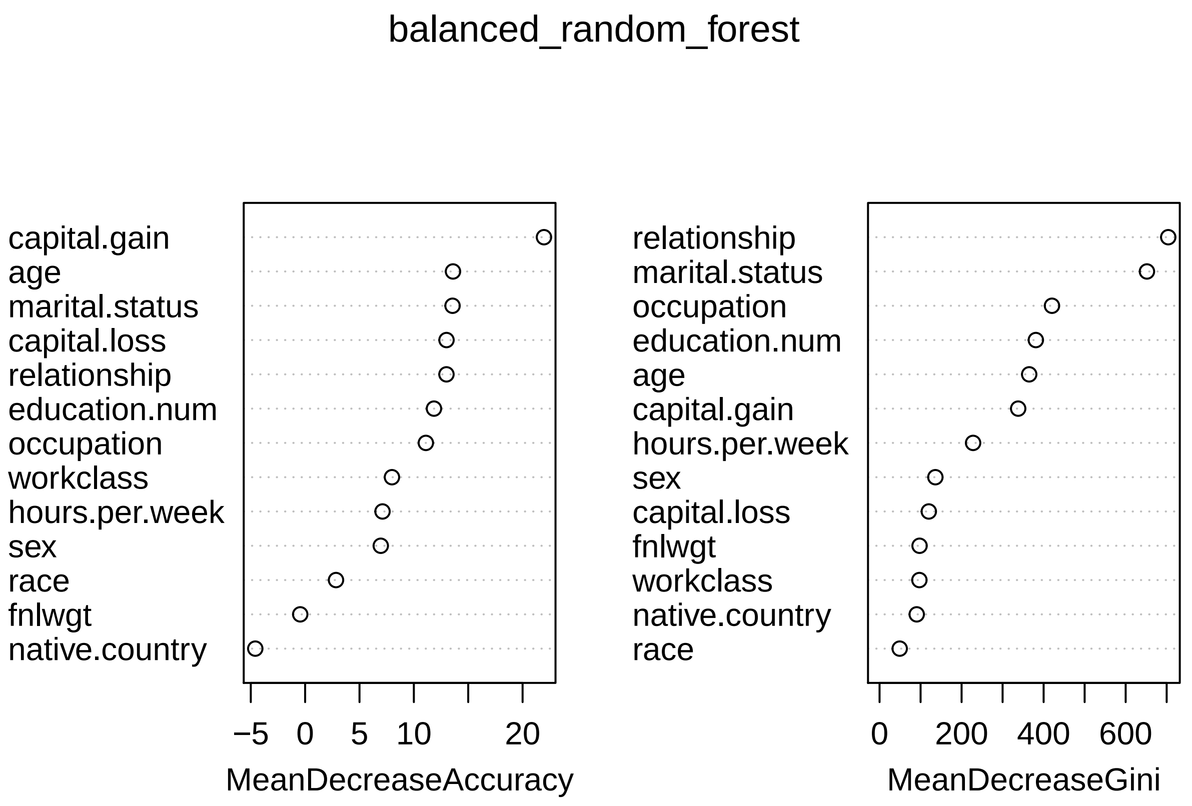
We get that the optimal number of variables to choose from at each split is 1. Therefore when we fit our random forest parameter mtry = 1.

```
# fit our new random forest classifier on the balanced data
set.seed(1)
balanced_random_forest <- randomForest(income ~ ., data = undersampled, mtry = 1, ntree = 250, importan

plot(balanced_random_forest)
```



We fit our new random forest called balanced_random_forest with 250 trees and from looking at the above plot of the out-of-bag error(the black line), it seems the oob error seems to level after 200 trees, so our random forest of 250 trees will be sufficient. We can next check what variables are most important for our new random forest.

```
# important variables
varImpPlot(balanced_random_forest)
```



```
balanced_random_forest$importance
```

```
##                        <=50K          >50K MeanDecreaseAccuracy
## age             3.343805e-03  2.023100e-02         1.179741e-02
## workclass       5.072053e-03  2.392554e-03         3.729906e-03
## fnlwgt         -3.339198e-04  1.320553e-04        -1.035760e-04
## education.num   2.226173e-02  1.873180e-02         2.049341e-02
## marital.status  3.917817e-02  3.202099e-02         3.560233e-02
## occupation      1.525616e-02  1.632010e-02         1.579823e-02
## relationship    4.260763e-02  3.570424e-02         3.916493e-02
## race           -1.337147e-05  1.298220e-03         6.426612e-04
## sex             1.134900e-02  3.544183e-03         7.451271e-03
## capital.gain    3.290811e-02  2.439600e-02         2.864594e-02
## capital.loss    7.288452e-03  7.324612e-03         7.302762e-03
## hours.per.week  4.648860e-03  8.154835e-03         6.402025e-03
## native.country -8.008671e-05 -7.826872e-05        -7.896408e-05
##                MeanDecreaseGini
## age                   364.83827
## workclass              97.09491
## fnlwgt                 97.53456
## education.num         380.99798
## marital.status        651.84948
## occupation            420.33837
## relationship          703.82267
## race                   49.10751
## sex                   136.04648
## capital.gain          337.92745
## capital.loss          120.06155
## hours.per.week        228.01389
## native.country         90.58654
```

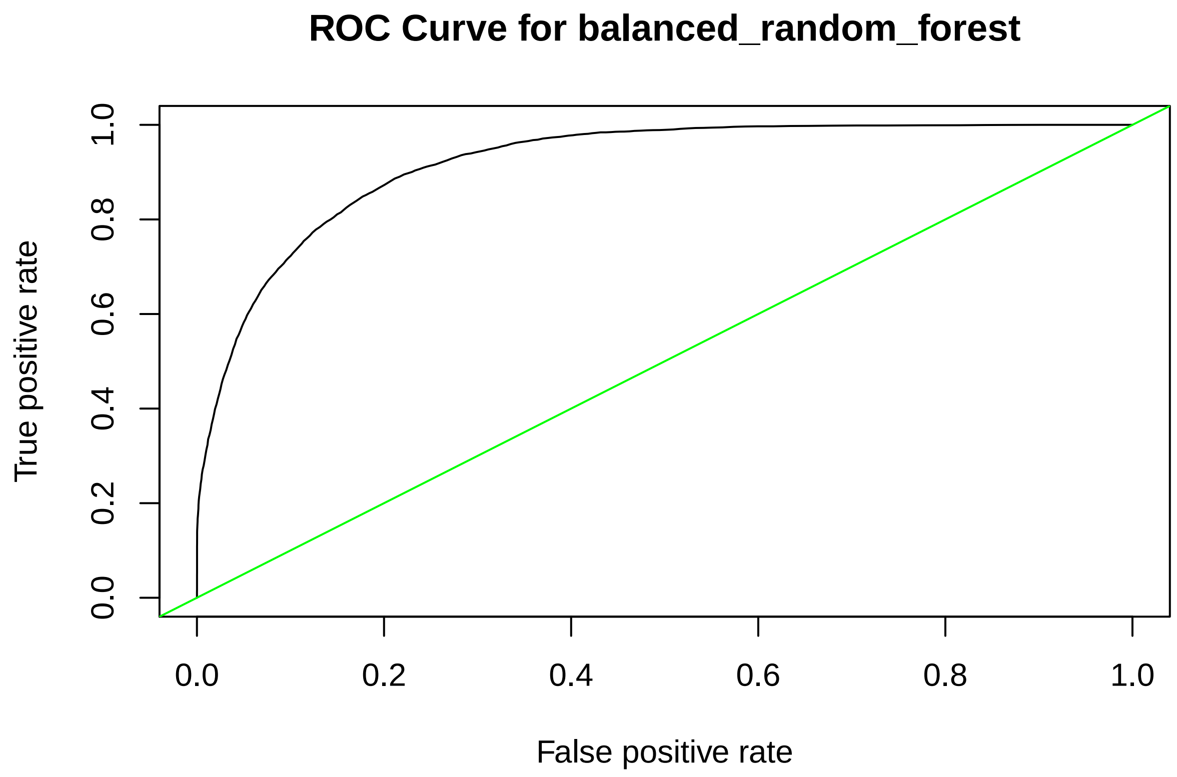
We consider important variables to be those that have largest decrease in Gini index averaged across all trees. Therefore using the measure of mean decrease in Gini, our most important variables in our new random forest are relationship, marital.status, ocupation, education.num, age, and capital.gain with mean decreas in Gini 703.82267, 651.84948, 420.33837, 380.99798, 364.83827, and 337.92745 respectively. Next we can calculate the training accuracy on our full training data(train) that was used to fit our other random forest to see if our new random forest trained on balanced data performs better.

```r
# calculate training accuracy rate on original unbalanced training set
set.seed(1)
balanced_random_forest_pred <- predict(balanced_random_forest, newdata = train)

# can use random_forest$y because is y from data set train
mean(balanced_random_forest_pred == random_forest$y)
```

```
## [1] 0.8612161
```

Our training accuracy rate for our new random forest trained on undersampled balanced training data is 86.12161 percent. Therefore our new random forest classifier(balanced_random_forest) performs better than our original random forest that was trained on the original training dataset(random_forest). Although our new random forest has a better training accuracy rate than the original random forest(random_forest), our new random forest still has a lower training accuracy rate than our bagged tree and our classification tree. Lastly we will plot the ROC curve for our new random classifier.

```r
# prediction object
set.seed(1)
balanced_prediction <- prediction(predict(balanced_random_forest, type = "vote", newdata = train)[,2], 

# plot roc curve

bal_roc <- performance(balanced_prediction, measure = "tpr", x.measure = "fpr")

plot(bal_roc, main = "ROC Curve for balanced_random_forest")
abline(a = 0, b = 1, col = "green")
```



```r
# report the area under curve

bal_auc <- performance(balanced_prediction, measure = "auc")

bal_auc@y.values
```

```
## [[1]]
## [1] 0.919952
```

The area under the curve for our new random forest is AUC = 0..919952

Model Selection

Now we will validate our supervised classifier with the best training error on the test set. Our best classifier is our classification tree(class_tree_prune). However before we validate our classifier we have to fix our test set first because the variable native.country has 40 levels in our test set and 41 in our training set. This leads to errors when we try to predict on the test set so we will add the missing level "Holand-Netherlands" to the test set.

```r
# fix our test set to have 41 levels for variable native.country

# we can first rbind both sets
all_data <- rbind(train, test)

# we can take out the training data and our test data should be left with 41 levels for native.country

adjusted_test <- all_data[-seq(1,30162, 1),]
```

Now we can use our classification tree to calculate the test accuracy rate.

```r
# predict on test
set.seed(1)
# probabilities of greater than 50k
class_test_pred <- predict(class_tree_prune, adjusted_test)[,2]

# boolean of greater than 50K
class_test_pred_char <- ifelse(class_test_pred > .5, " >50K", " <=50K")


mean(adjusted_test$income == class_test_pred_char)
```

```
## [1] 0.8555777
```

Our classification tree has a test accuracy rate of 85.55777 percent. We can calculate the confusion matrix for our classification tree predicted on our test set.

```r
# confusion matrix
Predictions <- class_test_pred_char

table(Predictions, adjusted_test$income)
```

```
##
## Predictions  <=50K  >50K
##       <=50K  10646  1461
##       >50K     714  2239
```

Using the positive event " >50K" we can calculate the true positive rate and the true negative rate.

```r
# true positive rate tpr = tp/p
tpr <- 2239 / (2239 + 1461)

tpr
```

```
## [1] 0.6051351
```

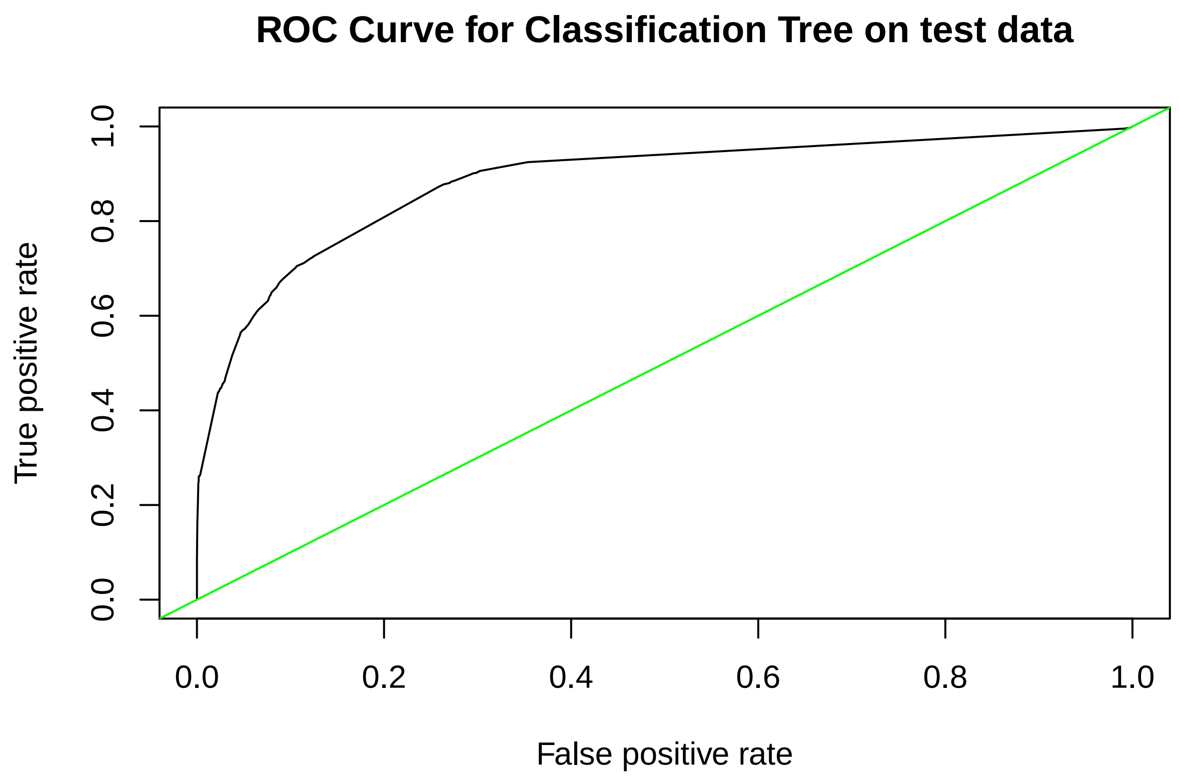
```r
# true negative rate = specificity, fpr = 1 - specificity = fp/n

fpr <- 714 / (714 + 10646)

#specificity
1 - fpr
```

```
## [1] 0.9371479
```

The true positive rate/ sensitivity is 0.6051351 and the specificity/ 1 - false positive rate is 0.9371479. lastly we will plot the roc curves of all our classifiers on the testing data. First for our classification tree on the test data.

```r
# classification tree roc curve
set.seed(1)
# ROCR prediction object
class_test_prediction <- prediction(class_test_pred, adjusted_test$income)

# plot roc curve

class_test_roc <- performance(class_test_prediction, measure = "tpr", x.measure = "fpr")

plot(class_test_roc, main = "ROC Curve for Classification Tree on test data")
```

```r
abline(a = 0, b = 1, col = "green")
```

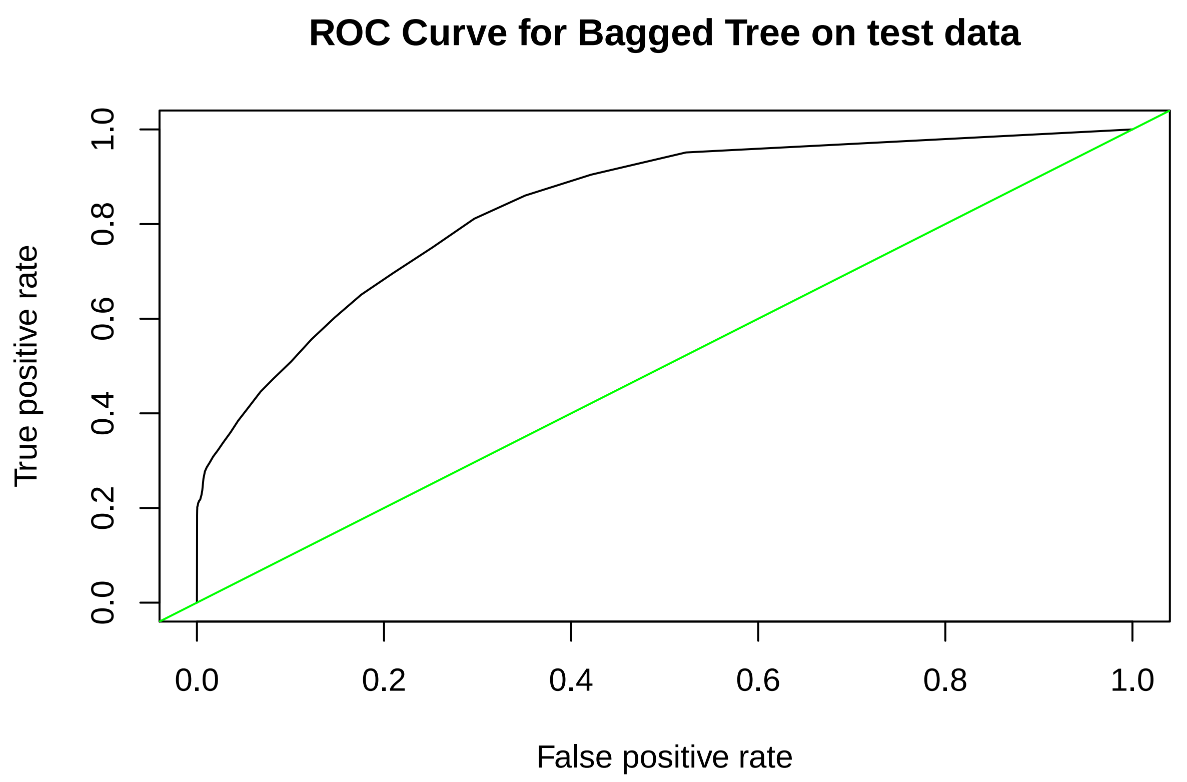


```r
# report the area under curve

class_test_auc <- performance(class_test_prediction, measure = "auc")

class_test_auc@y.values
```

```
## [[1]]
## [1] 0.8842563
```

ROC curve for bagged tree on test data

```r
# roc for bagged tree on test data
set.seed(1)
# ROCR prediction object

bagged_test_prediction <- prediction(predict(bagged_tree, adjusted_test, type = "vote")[,2], adjusted_te

# plot roc curve

bagged_test_roc <- performance(bagged_test_prediction, measure = "tpr", x.measure = "fpr")

plot(bagged_test_roc, main = "ROC Curve for Bagged Tree on test data")
abline(a = 0, b = 1, col = "green")
```

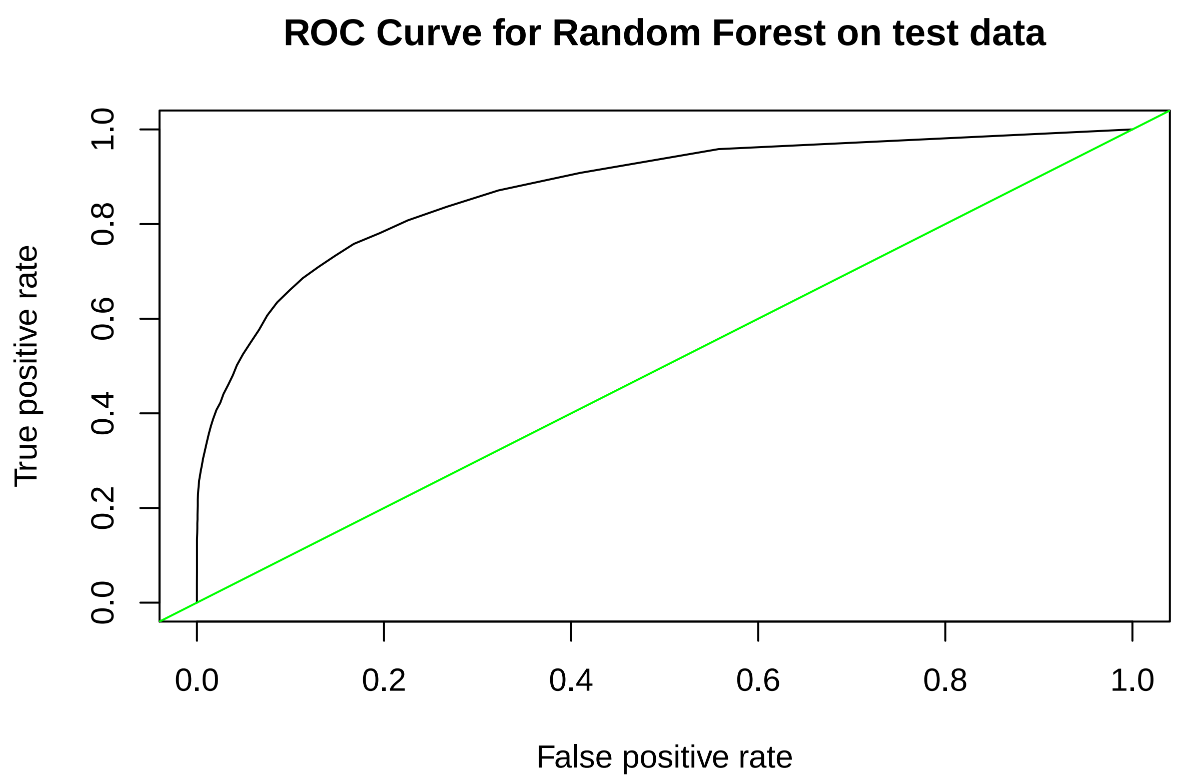


```
# report the area under curve

bagged_test_auc <- performance(bagged_test_prediction, measure = "auc")

bagged_test_auc@y.values
```

```
## [[1]]
## [1] 0.8387625
```

ROC curve for our random forest classifier fit on the original training data.

```
# roc for random forest
# prediction object
set.seed(1)
rf_test_prediction <- prediction(predict(random_forest, newdata = adjusted_test, type = "vote")[,2], ad

# plot roc curve

rf_test_roc <- performance(rf_test_prediction, measure = "tpr", x.measure = "fpr")

plot(rf_test_roc, main = "ROC Curve for Random Forest on test data")
abline(a = 0, b = 1, col = "green")
```

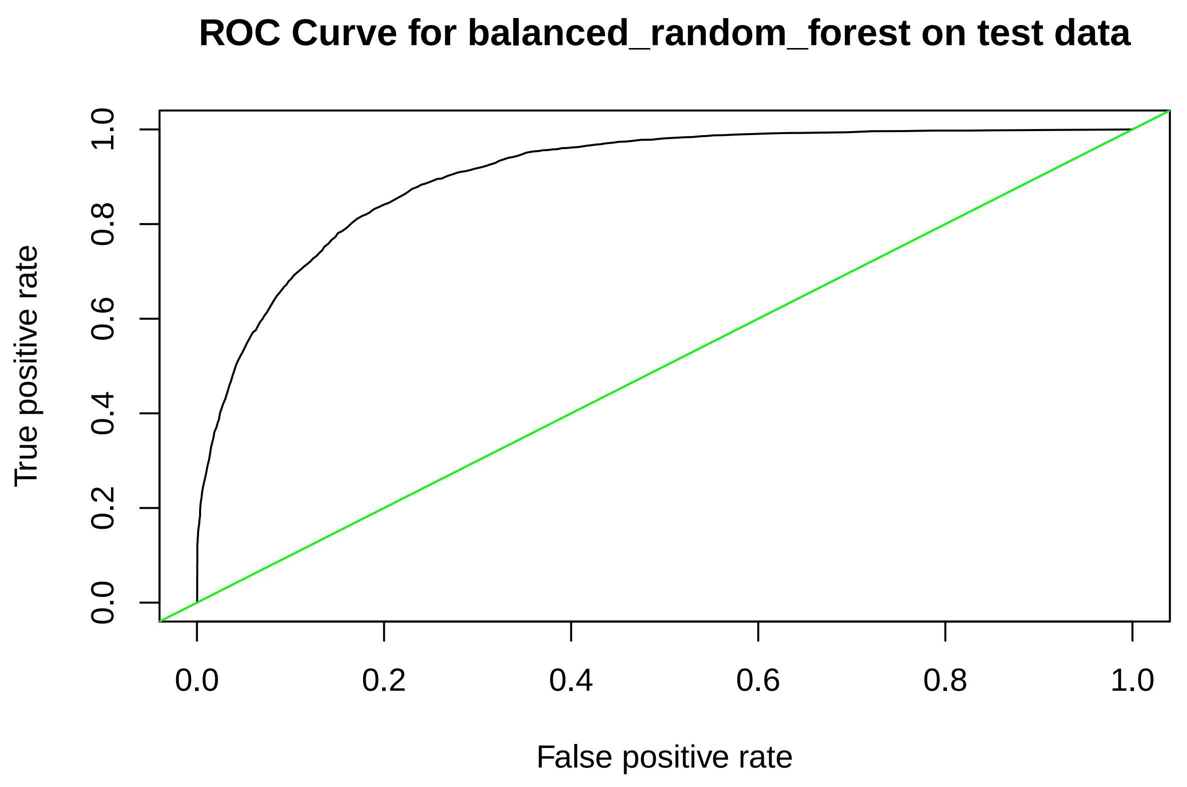


```r
# report the area under curve

rf_test_auc <- performance(rf_test_prediction, measure = "auc")

rf_test_auc@y.values
```

```
## [[1]]
## [1] 0.8743767
```

ROC curve for the random forest fit on the undersampled balanced training set.

```r
# roc curve for balanced random forest
# prediction object
set.seed(1)
balanced_test_prediction <- prediction(predict(balanced_random_forest, type = "vote", newdata = adjuste

# plot roc curve

bal_test_roc <- performance(balanced_test_prediction, measure = "tpr", x.measure = "fpr")

plot(bal_test_roc, main = "ROC Curve for balanced_random_forest on test data")
abline(a = 0, b = 1, col = "green")
```



```
# report the area under curve

bal_test_auc <- performance(balanced_test_prediction, measure = "auc")

bal_test_auc@y.values
```

```
## [[1]]
## [1] 0.9045263
```