stat154_FinalReport

Chase Enzweiler

12/4/2017

Stat 154 Final Report

Introduction

In this report, we are tasked with classifying observations in the Census Income Data Set from the UCI machine learning repository. Observations in the Census Income Data Set have 15 variables. Our goal for this project is to predict which observations have an income of greater than 50,000 dollars per year. To predict the income of observations we fit a classification tree, a bagged tree, and a random forest and use the best of these classifiers to predict which individuals make more than 50,000 dollars a year. This report will be organized into an preprocessing phase, a model building phase for each model, a model validation phase, and lastly a conclusion.

Preprocessing

The data initially given had already been split into a training data set to fit the models on and a test set to validate the models. Both the training data set and test data set had missing values. Because both data sets had a considerable amount of observations (over 32000 for training set and over 16000 for test set), the missing values were then removed from the data set. We removed the missing values anticipating that they might cause us problems when fitting our random forest and bagged tree with function randomForest(). After removing the missing values we still had a good amount of data to work with (30162 observations for training set and 15060 for test set). We next noticed that variables “education” and “education.num” were essentially the same variable so we dropped the variable “education”. We did not remove any outliers because decision trees are not sensitive to outliers and splitting in our trees happen based on proportions. Decision trees need no assumptions of linearity and they will keep the same structure regardless of transformation. To find out a bit more about our data we did take a look at a few other things including a summary of the training data set.

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

Something interesting we noticed about the summary of the training data is that the data is imbalanced. In our training data set more than 75% of the observations make less than or equal to 50,000 dollars a year. This is worrisome because fitting a random forest on imbalanced data may cause our random forest to be bias towards the class with majority observations and decrease its training accuracy. So, just incase our random forest becomes too biased, we can create another seperate training data set that is balanced. We create this new data set by keeping all the observations with the minority class from the original training set and combining it with an equal amount of observations randomly sampled from majority class. This new training set will then have equal observations from each class of income and we will call it the undersampled training set.

Model Building: Classification Tree

The first model we will fit to the training data is a classification tree. Classification trees are grown using recursive binary splitting and in our tree the Gini index is used to determine the quality of a particular split. We first grow our classification tree on the training data set using the function rpart() from the rpart package. We want to grow our tree large and then prune the tree back to avoid overfitting. Therefore, we use parameters in minsplit = 20 and cp = 0 in rpart.control to make splits until cost complexity parameter(cp) was at zero with a minimum of 20 observations in each node. The table below is calculted from the rpart function and we can use the table to find at which split the tree has the lowest cross-validation error. We find the cp associated with the smallest cross-validation error and then prune the tree with the function prune() from rpart using our just found cp as a parameter in the prune function.

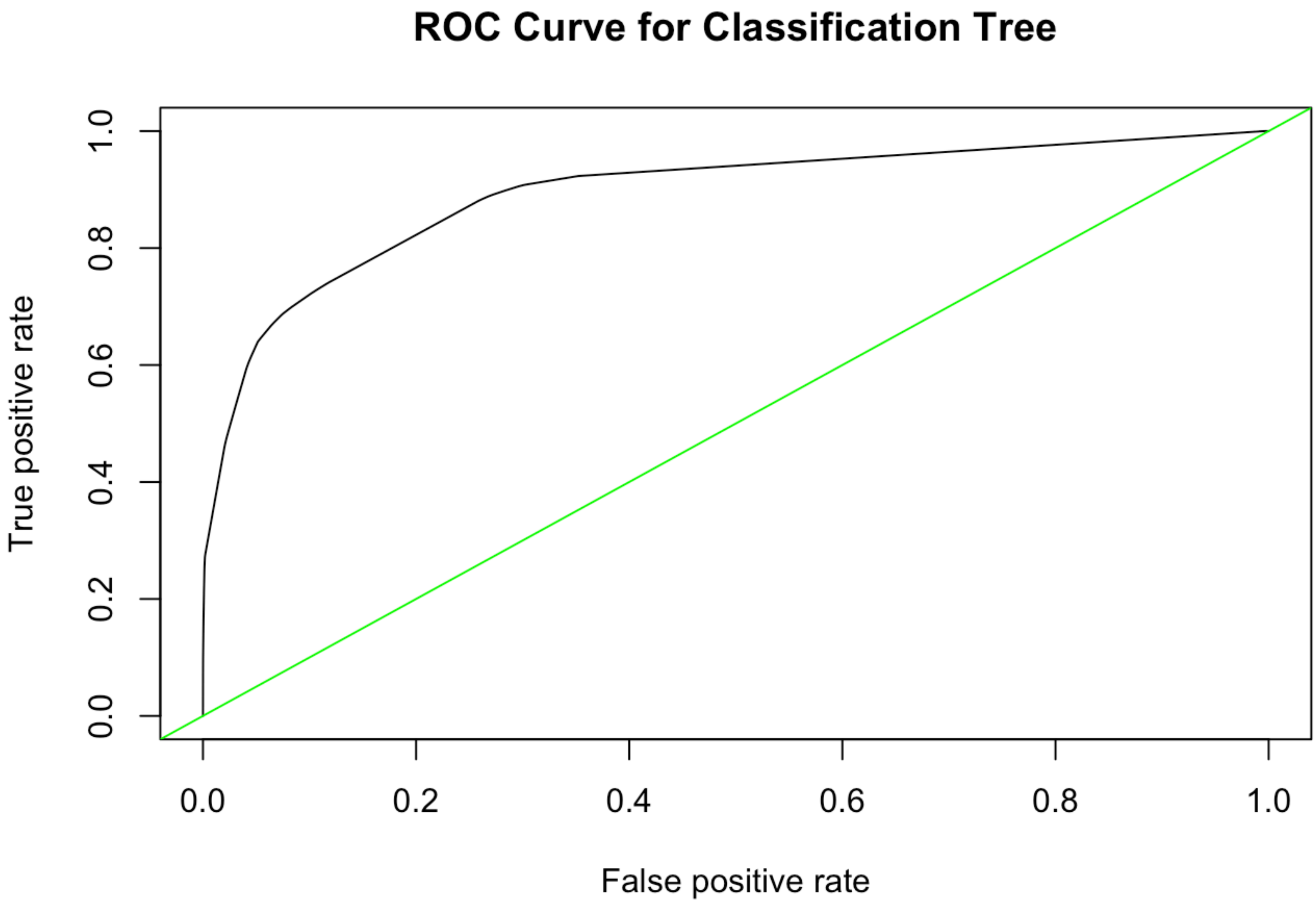
```
##
## Classification tree:
## rpart(formula = income ~ ., data = train, method = "class", control = rpart.contro
l(minsplit = 20,
##     cp = 0))
##
## Variables actually used in tree construction:
##  [1] age            capital.gain   capital.loss   education.num
##  [5] fnlwgt         hours.per.week 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
```

From this table we see that the lowest cross-validation error occurs at 78 splits with associated cp of 4.6617e-04. We will then prune our tree back to 78 splits using the function prune. 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. The output below shows the decrease in Gini for each variable.

```
##   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
```

We can see that our most important variables are relationship, marital.status, education.num, capital.gain, and occupation in that order. We can then calculate how well our model performs on the training data by calculating the training accuracy rate. We can get estimated probabilities of observations for each class using the function predict() and parameter newdata = training data. Using the cutoff >.5 we can classify observations using the estimated probabilities of the class greater than 50,000 dollars a year. The calculated training accuracy rate for our classification tree is 0.8716597 which means about 87% of the observations are classified correctly. We can then view the overall performance of the model over all cutoffs/thresholds by looking at its ROC curve on the training data and the area under the ROC curve. We create a ROC by running the function prediction() to create a prediction object and then calling the function performance() on our prediction object. We use parameters measure = “tpr” and x.measure = “fpr” which means true positive rate will be on the y axis when plotted and x.measure will be on the x axis. The performance function will evaluate the performance measures at unspecified thresholds and we can plot it to form a ROC curve. we can also set the paramter “measure” in performance equal to “auc” to report the area under the roc curve. All the functions needed to create the ROC curve are in the package ROCR.



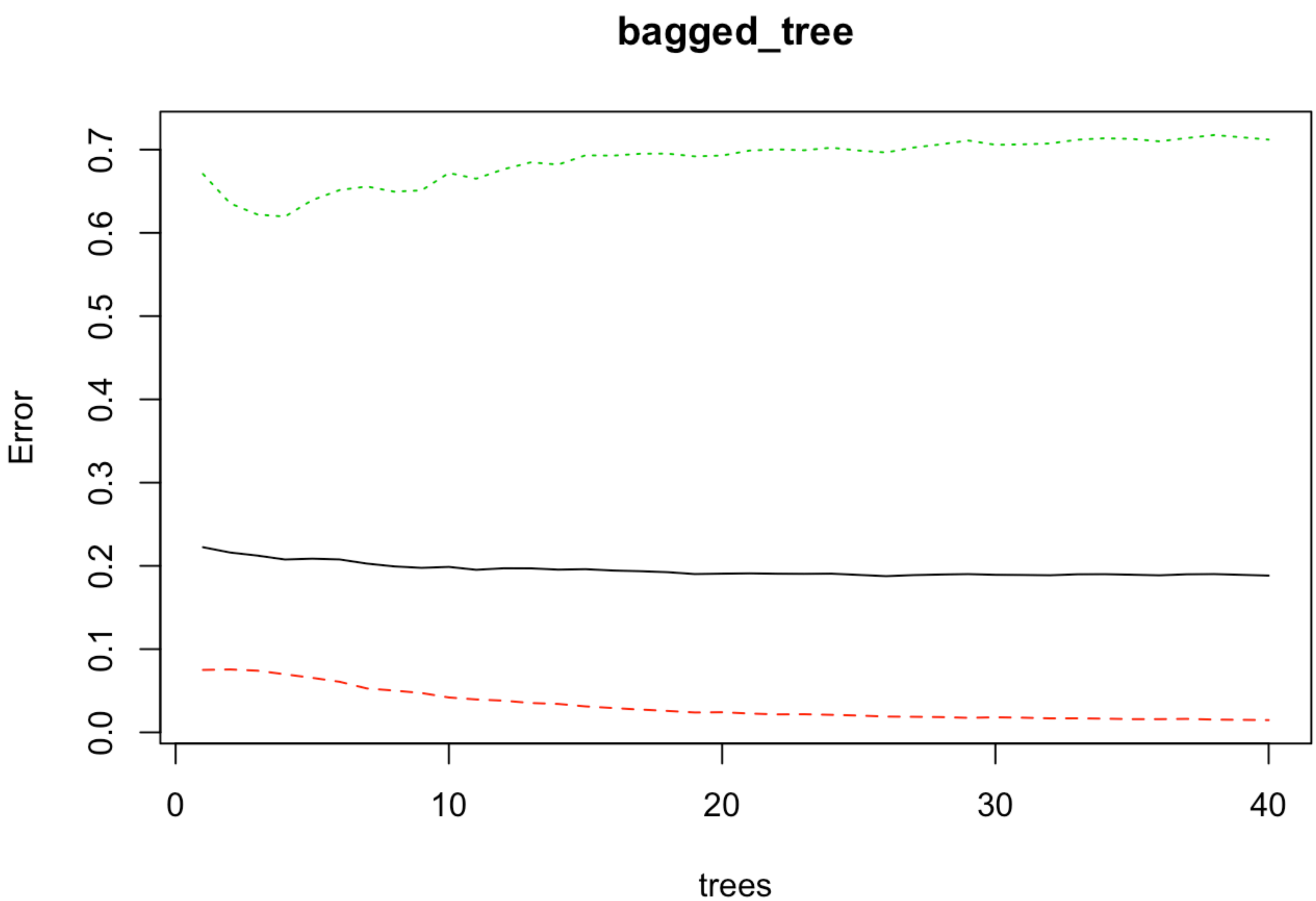
The area under the curve(AUC) is 0.8929411 which means that our classification tree is a good classifier over all thresholds and performs much better than the no information classifier represented by the green line with AUC = .5. Lastly, we can look at the confusion matrix of our classifier predicted on training data to see how well it does for each class.

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

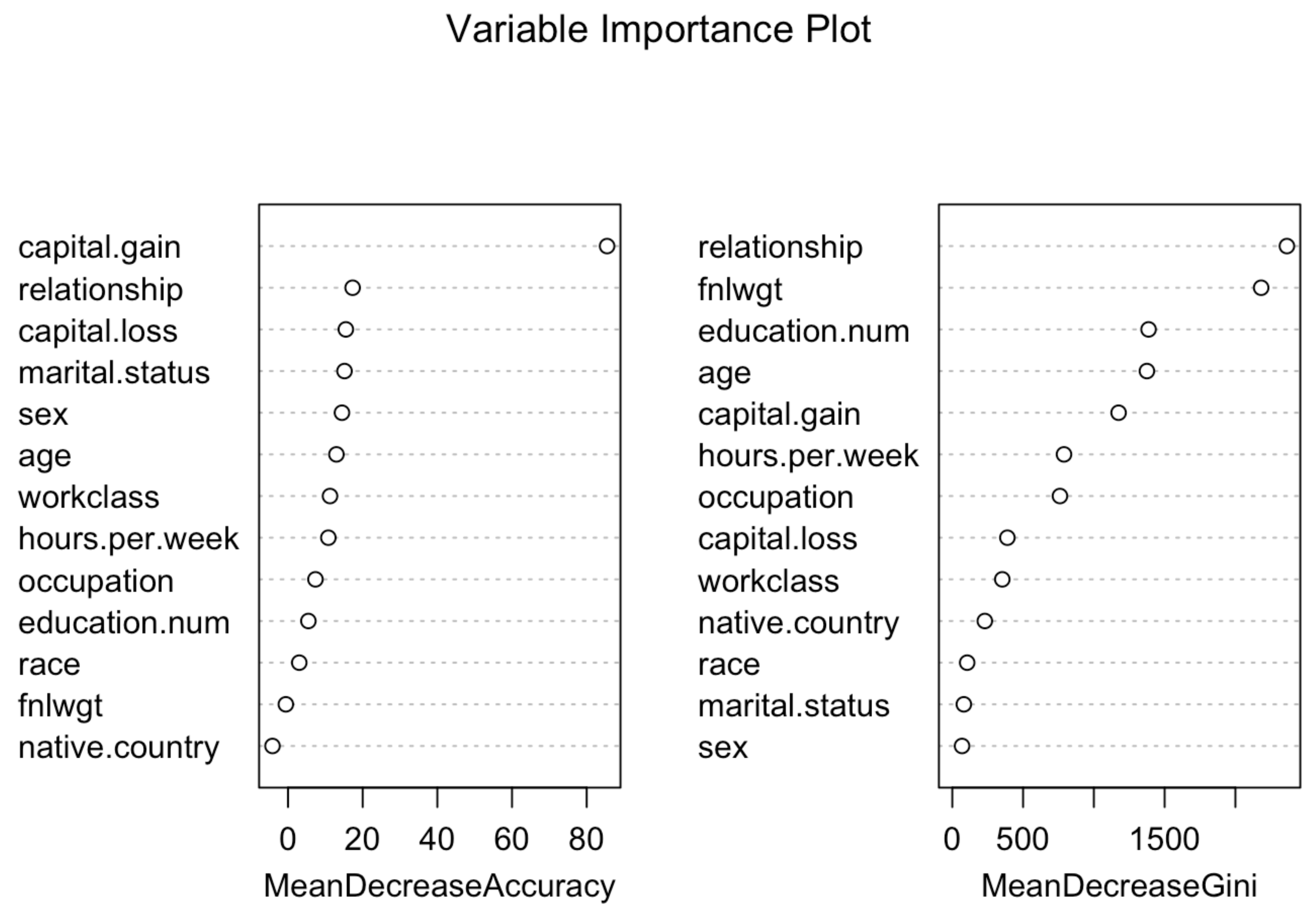
We see that our classification tree has a higher error rate for class >50K, but we are satisfied since we are mainly concerned with training accuracy rate.

Model Building: Bagged Tree

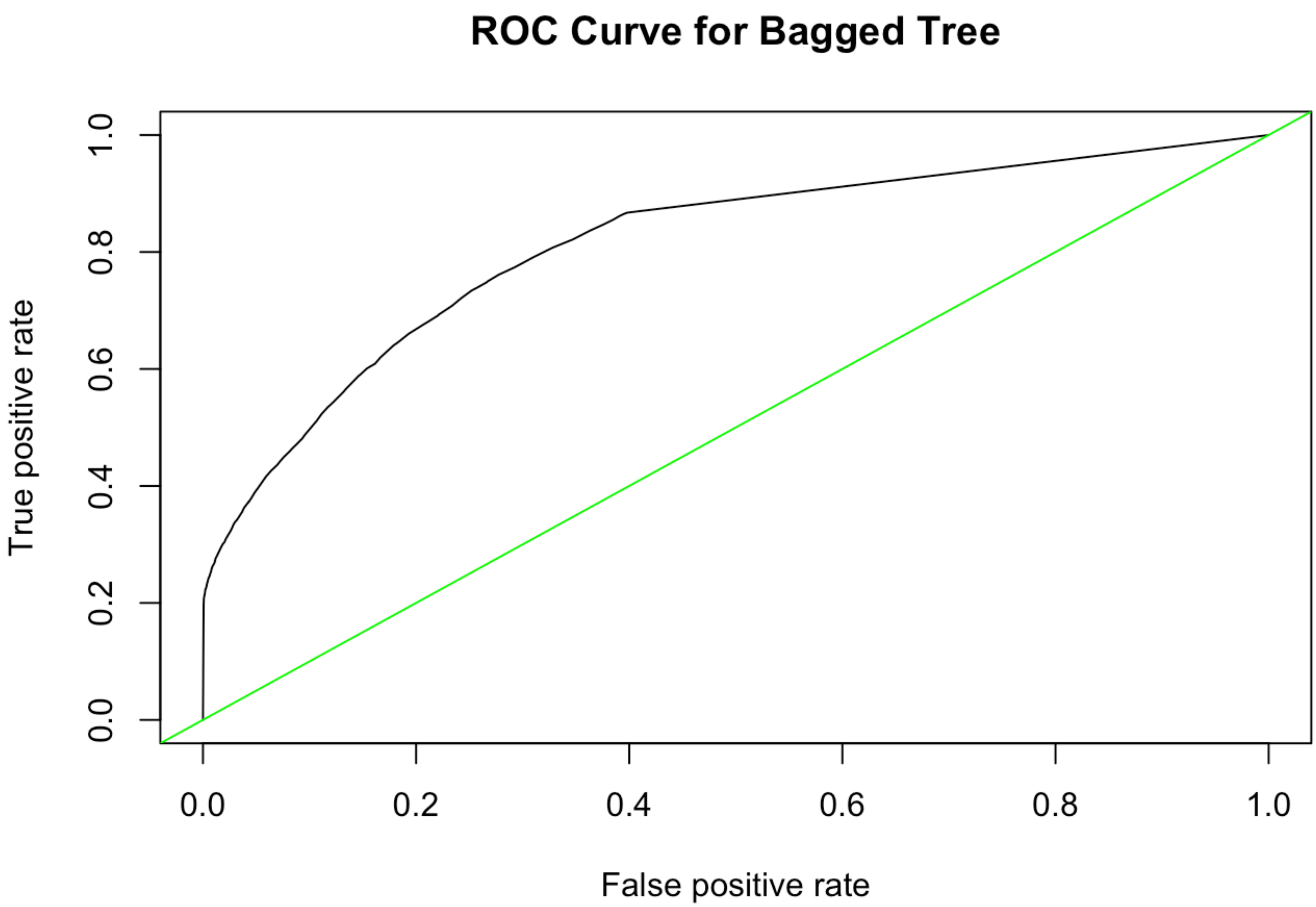
Now we will fit a bagged tree to the training data. Bagging is used to reduce the variance of classification trees by bootstrapping the training data and fitting new classification trees to the resampled data. Predictions are then made using majority vote over all the created trees. So we then start by fitting a bagged tree on the training set using the function randomForest() from the package randomForest. We fit our bagged tree starting with 40 trees. Because increasing how many trees are used in a bagged tree won't lead to overfitting, any large number of trees will work, but we start with 40 because it is not too computationally expensive. We will verify how many trees we need by seeing at what number of trees the out-of-bag error levels out where increasing the trees won't have much more of a decrease in out-of-bag error. We can do this by visualizing the plot below.



We see that the out-of-bag error represented by the solid black line levels out between 30 and 40 trees. Therefore 40 trees will be sufficient for our bagged tree. The dashed lines are 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. We determine that our most important variables for this model are those who have the highest mean decrease of Gini index for that variables splits. We can plot a variable importance to compare the mean decrease in Gini of all our variables.



Looking at the variable importance plot the 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. We can then determine how accurate our model is on the training data by calculating the training accuracy rate. The training accuracy rate for our bagged tree is 0.8630064, so when classifying observations on the training set our bagged tree is correct for about 86 percent of the observations. Our bagged tree performs slightly worse than our single classification tree that has a training accuracy rate of 0.8716597. We can then view the overall performance of the model over all cutoffs/thresholds by looking at its ROC curve on the training data and the area under the ROC curve.



The area under the curve(AUC) for this model is 0.8126158. Unlike the ROC curve for our classification tree, the ROC curve for our bagged tree was determined not by probabilities but by proportions of votes by all the trees for each class. Our bagged model performs much better than the no information classifier that has AUC = .5. Lastly, to get a better idea of how the bagged tree performed for each class on the training set we can compute the confusion matrix of the bagged tree predicted on the training set.

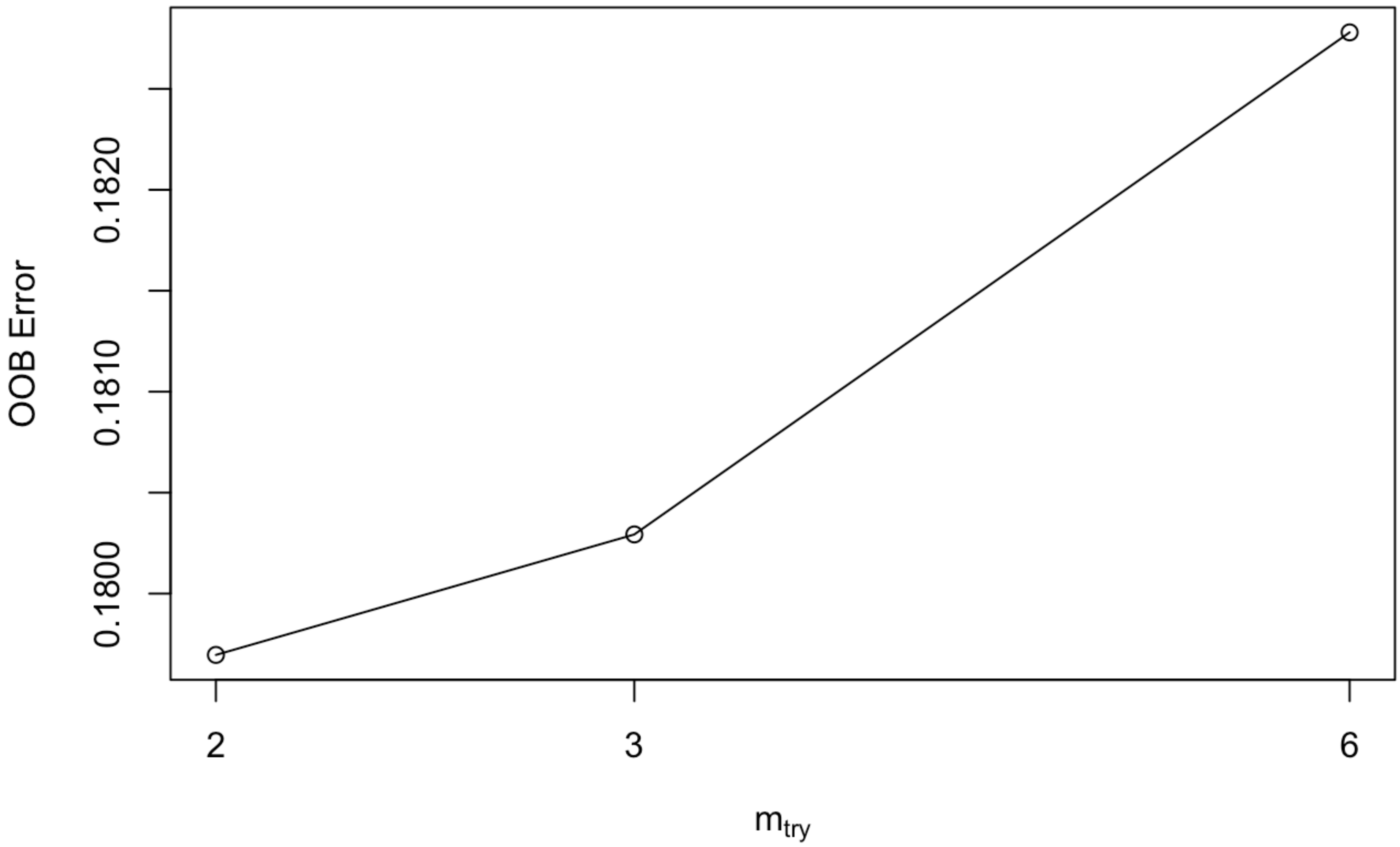
```
## 
## bagged_tree_predictions  <=50K  >50K
##                   <=50K  22639  4117
##                   >50K      15  3391
```

Looking at the confusion matrix we see that our bagged tree performs very well classifying observations with true class <=50K correctly, it only misclassified 15 times. However, our bagged tree performed poorly when trying to correctly classify observations that true class were >50K. This could reveal that our bagged tree could be biased towards the class <=50K which could be a consequence of training the bagged tree on a imbalanced training set. Regardless, we still get a good training accuracy rate of 0.8630064 which is what we are more concerned with so we won’t retrain the bagged tree on the undersampled training set.

Model Building: Random Forest

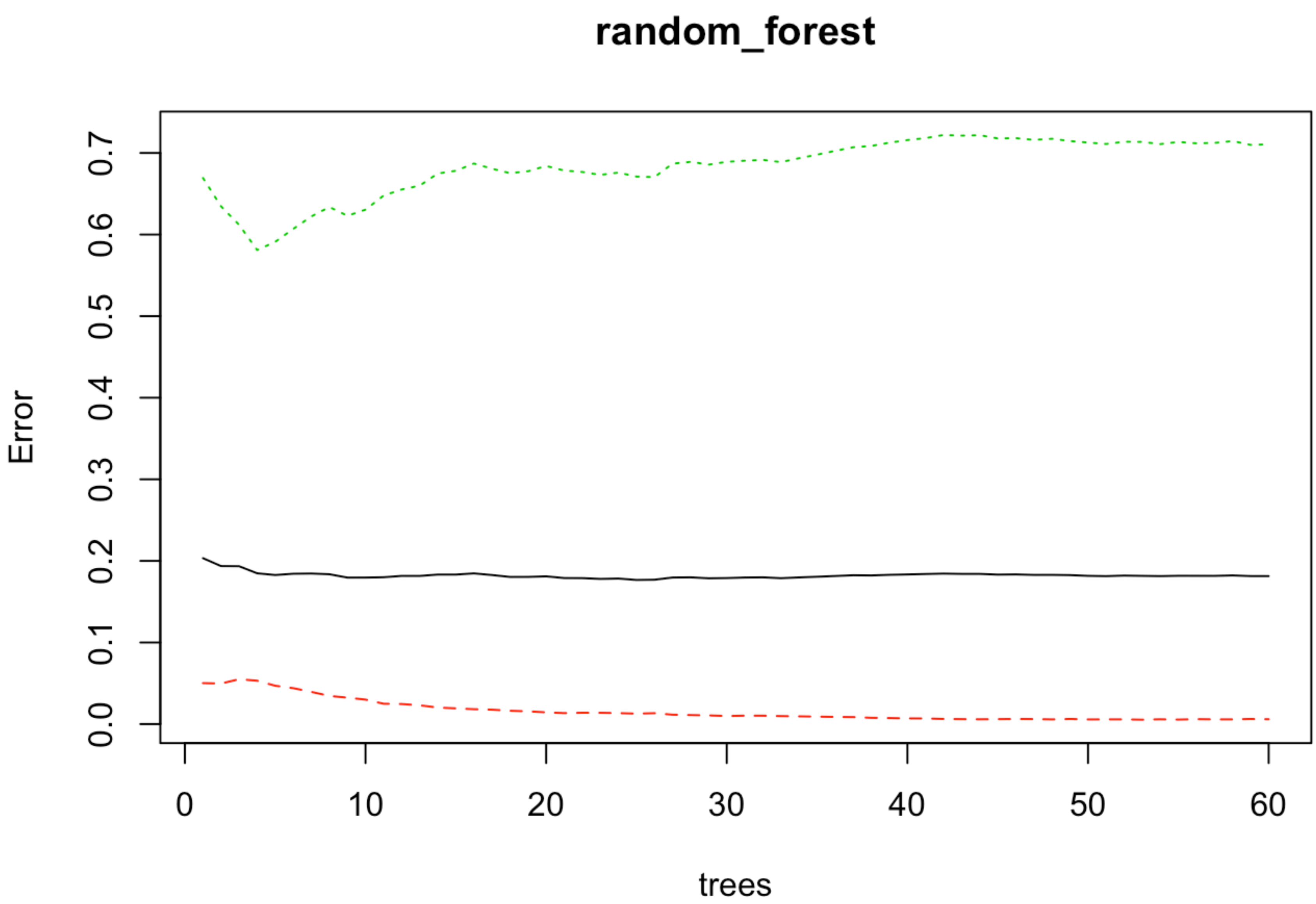
Lastly we will fit a random forest to the training data using the function randomForest() from the package RandomForest. Random forests are similiar to bagged trees where they fit many trees over bootstrapped training data and predict using majority votes. However, when grwoing the trees random forests restrict how many variables may be chosen from when selecting a variable to perform each split. For each split a specified amount of variables are chosen randomly from all the variables and then the best variable for the split of the chosen variables is used. This is done to decorrelate the trees that make up the random forest and make predictions less variable. To fit a random forest on our training data we first must find an optimal amount of variables to be selected randomly to be chosen from for each split, this is the paramter mtry in the function randomForests(). The parameter mtry is usually chosen as the square root of the total amount of variables in the data, but here we will use the function tuneRF() from the package randomForest to find the ideal number for the parameter mtry. The function tuneRF() fits random forests with 50 trees to the training data with multiple values for the paramter mtry and then calculates the out-of-bag error for each of those random forests. The value for mtry that produces the random forest with the lowest out-of-bag error is the value of mtry that we want to use for our model. Below is the results from running the function tuneRF().

```
## mtry = 3  OOB error = 18.03%
## Searching left ...
## mtry = 2     OOB error = 17.97%
## 0.00331004 0.05
## Searching right ...
## mtry = 6     OOB error = 18.28%
## -0.01379184 0.05
```

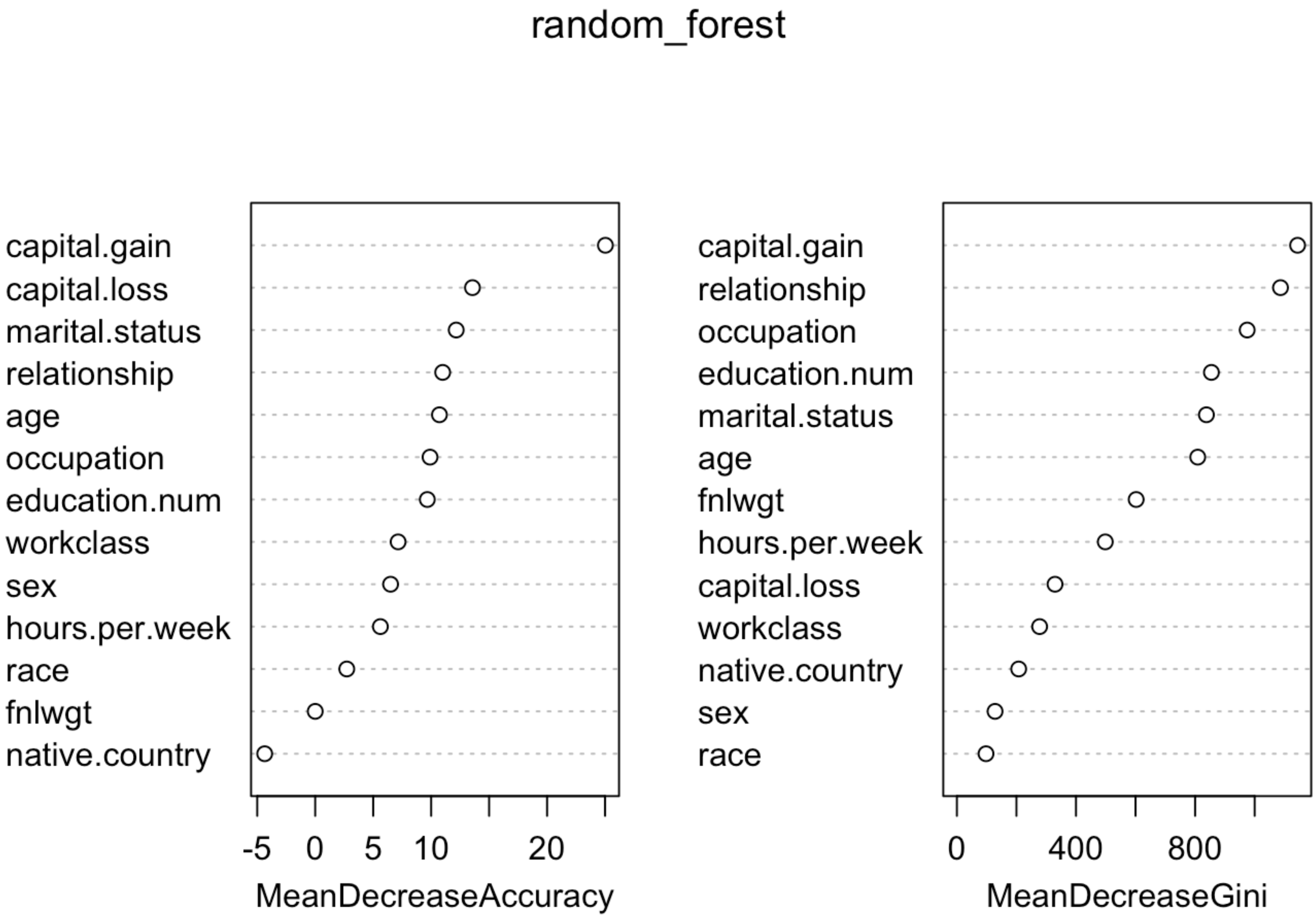



```
##        mtry  OOBError
## 2.OOB    2 0.1796963
## 3.OOB    3 0.1802931
## 6.OOB    6 0.1827797
```

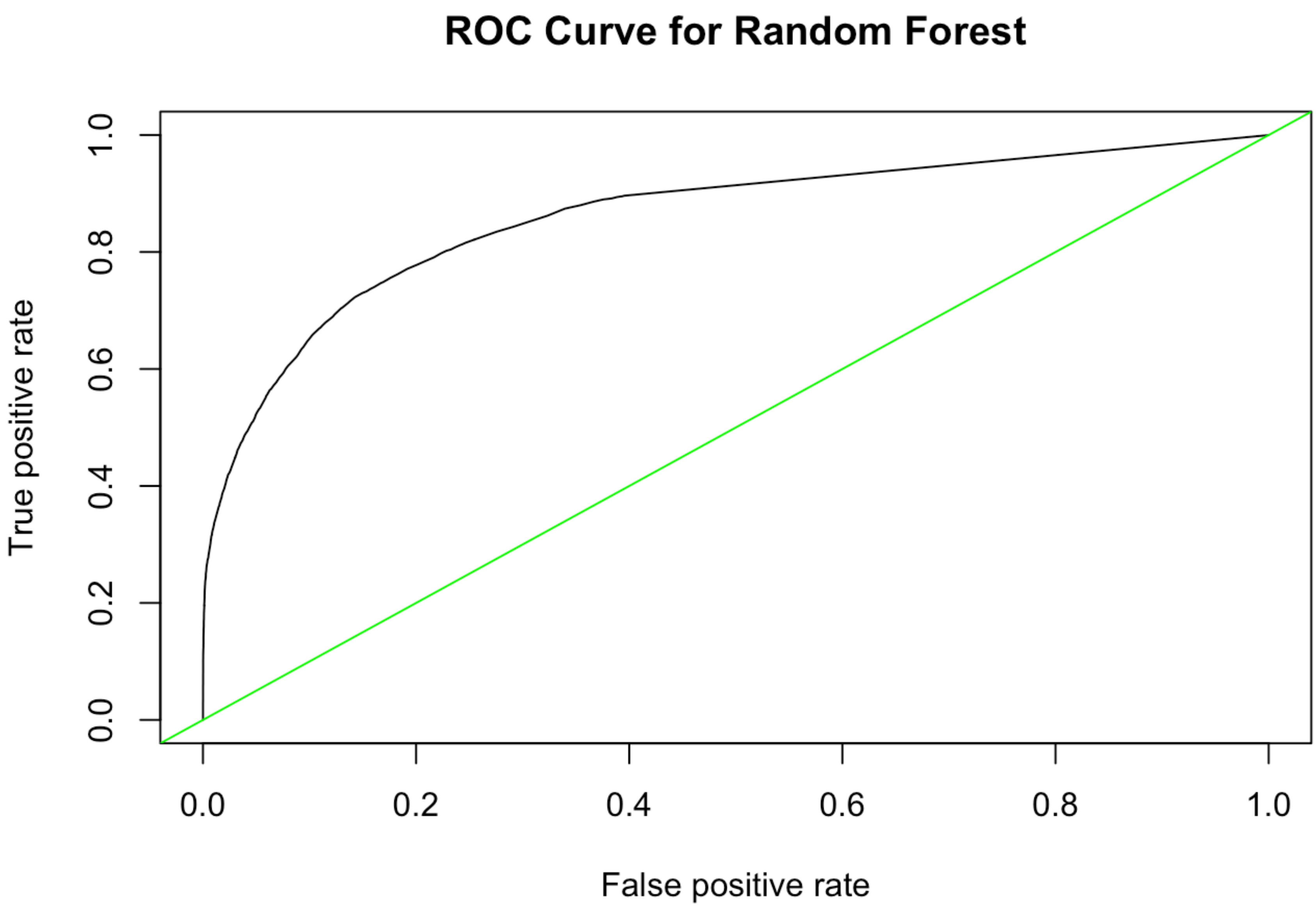
From the above output we get the result that the optimal value of mtry with the lowest out-of-bag error of 0.1756515 is 2. Now we can build our random forest with the function randomForest() using parameters mtry = 2 and initially choosing 60 trees to make up our random forest(ntree = 60). To make sure that 60 trees is enough trees for our random forest we can plot the out-of-bag error against the number of trees and see where the error doesn't change much with respect to how many trees there are.



Looking at the plot of the out-of-bag error against the number trees we see that the out-of-bag error remains roughly level after about 40 trees. Therefore since adding more trees won't cause overfitting, 60 trees is sufficient for our random forest. We determine that our most important variables for this model are those who have the highest mean decrease of Gini index for that variables splits. As before we can plot a variable importance to compare the mean decrease in Gini of all our variables.



From the above plot we can see that 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. We then calculate the the training accuracy of our random forest to see how well our random forest performs on the training data. The training accuracy rate of our random forest is 0.8275976, so when classifying observations on the training set our bagged tree is correct for about 82.8 percent of the observations. Our random forest has the lowest training accuracy of all our previous classifiers. We can then view the overall performance of the random forest over all cutoffs/thresholds by looking at its ROC curve on the training data and the area under the ROC curve.

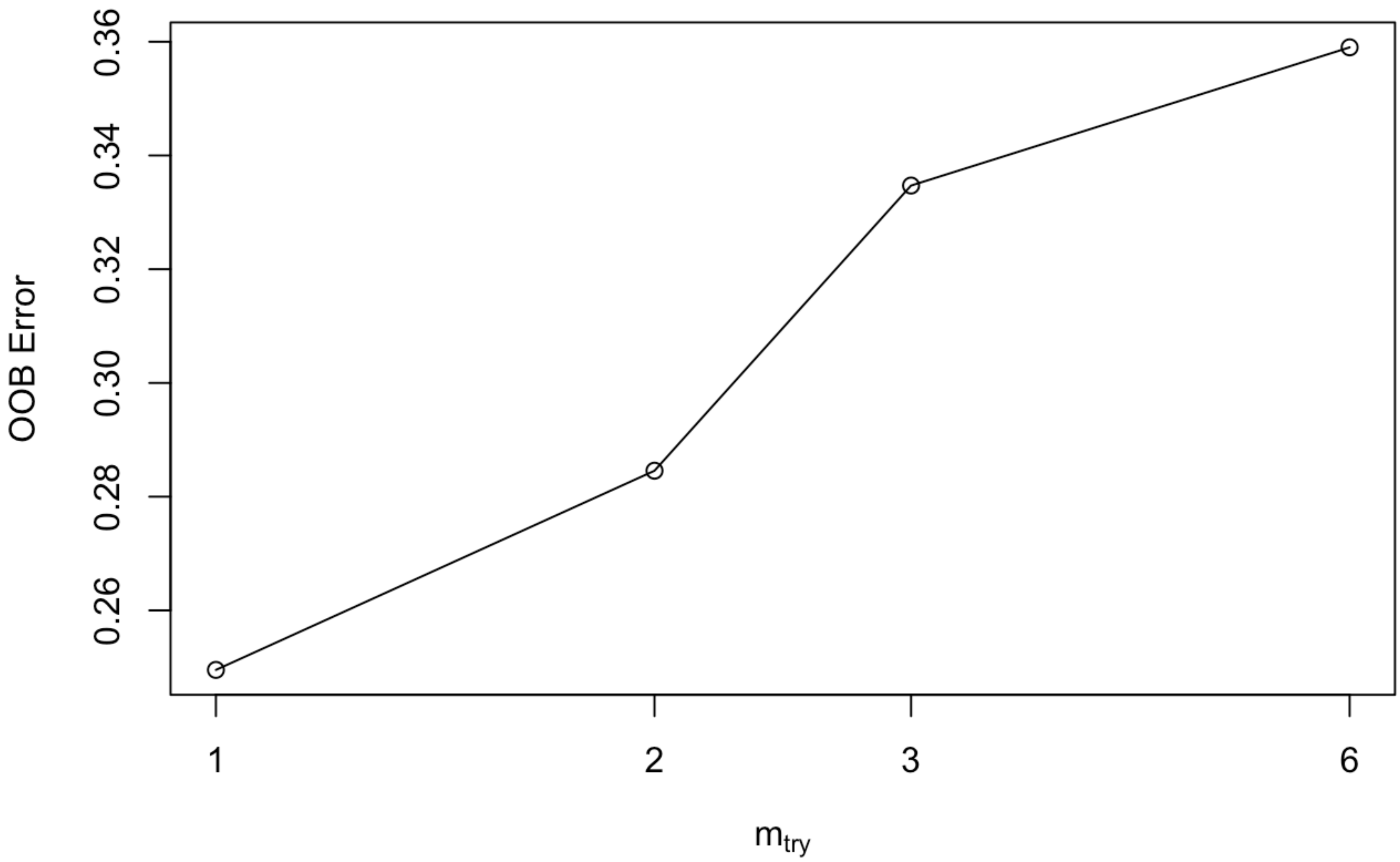


The area under the curve for our random forest is 0.8606042. By looking at the ROC curve we see that our random forest performs much better than the no information classifier which has an area under the curve of .5. To get a better idea of how our random forest performs for each class we can view the confusion matrix of the random forests predictions on the training data.

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

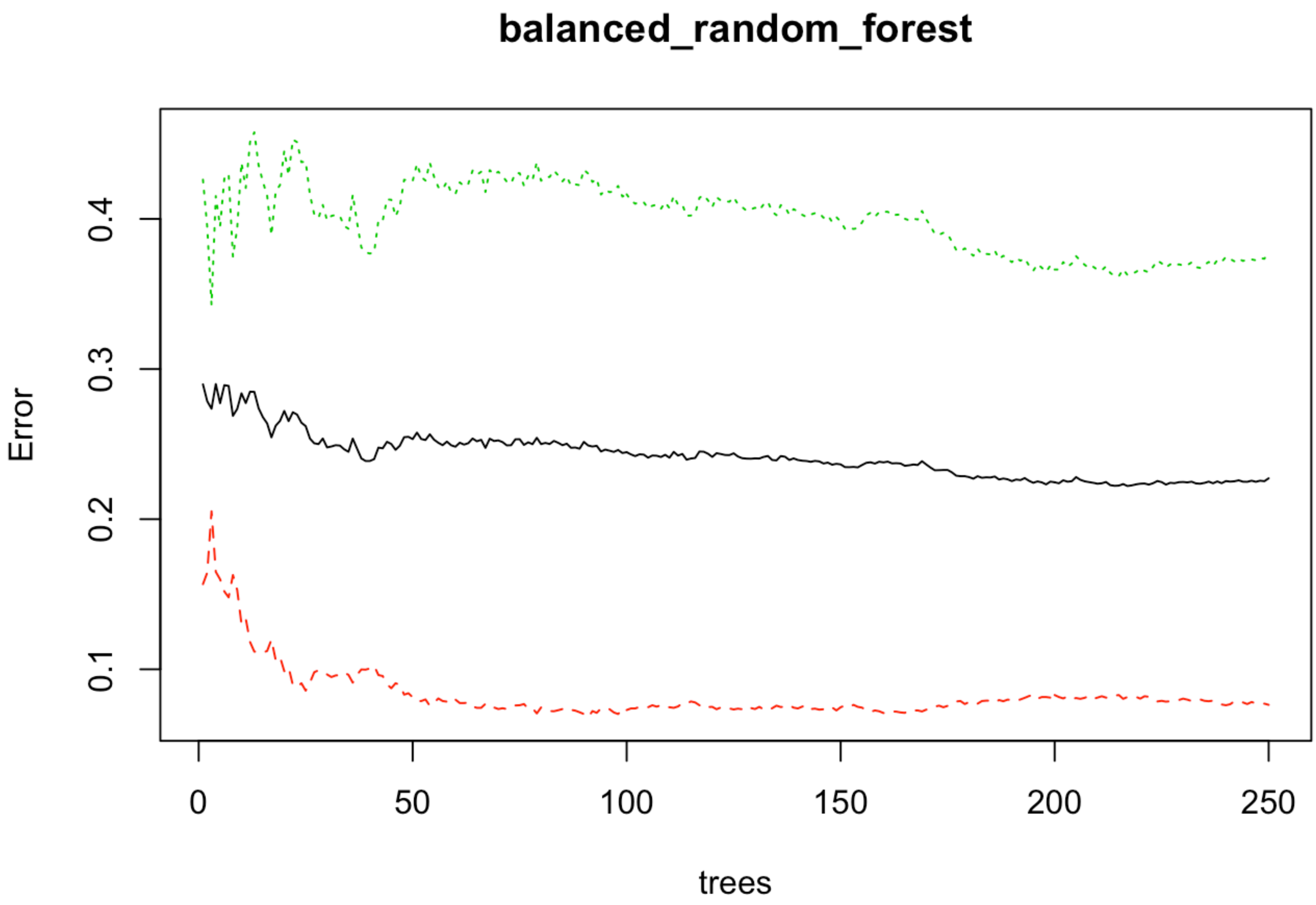
Looking at the confusion matrix for our random forest we notice that our sensitivity (true positive rate) is very low (with >50K being regarded as positive event) at 2327 / (2327 + 5181) = .3099361 and our specificity (1 - false positive rate) is extremely high at 1 - (19 / (19 + 22635)) = 0.9991613. This shows that training our random forest on an imbalanced data set may have made our random forest very biased towards the class <=50K. We are worrying about this because compared to our other classifiers the training accuracy rate of our random forest is much lower at 0.8275976 where the others are 0.8630064 and 0.8716597. Therefore we want to try to improve our training accuracy rate of our random forest.   Mentioned earlier in the data preprocessing section, we created a seperate balanced training set that we refered to as undersampled training set. We will then fit a new random forest on the undersampled training set to see if fitting the random forest on this data will decrease the random forests bias and increase its training accuracy rate on the original training data. We will refer to this new random forest as our balanced random forest. We will fit the new balanced random forest with exactly the same steps as we fit the original random forest (except on the undersampled training data). We will then explain this very shortly to avoid redundancy. We use the function tuneRF() as we did above to find the optimal value for the paramter mtry.

```
## mtry = 3  OOB error = 33.47%
## Searching left ...
## mtry = 2     OOB error = 28.46%
## 0.1498209 0.05
## mtry = 1     OOB error = 24.95%
## 0.1230985 0.05
## Searching right ...
## mtry = 6     OOB error = 35.9%
## -0.438751 0.05
```

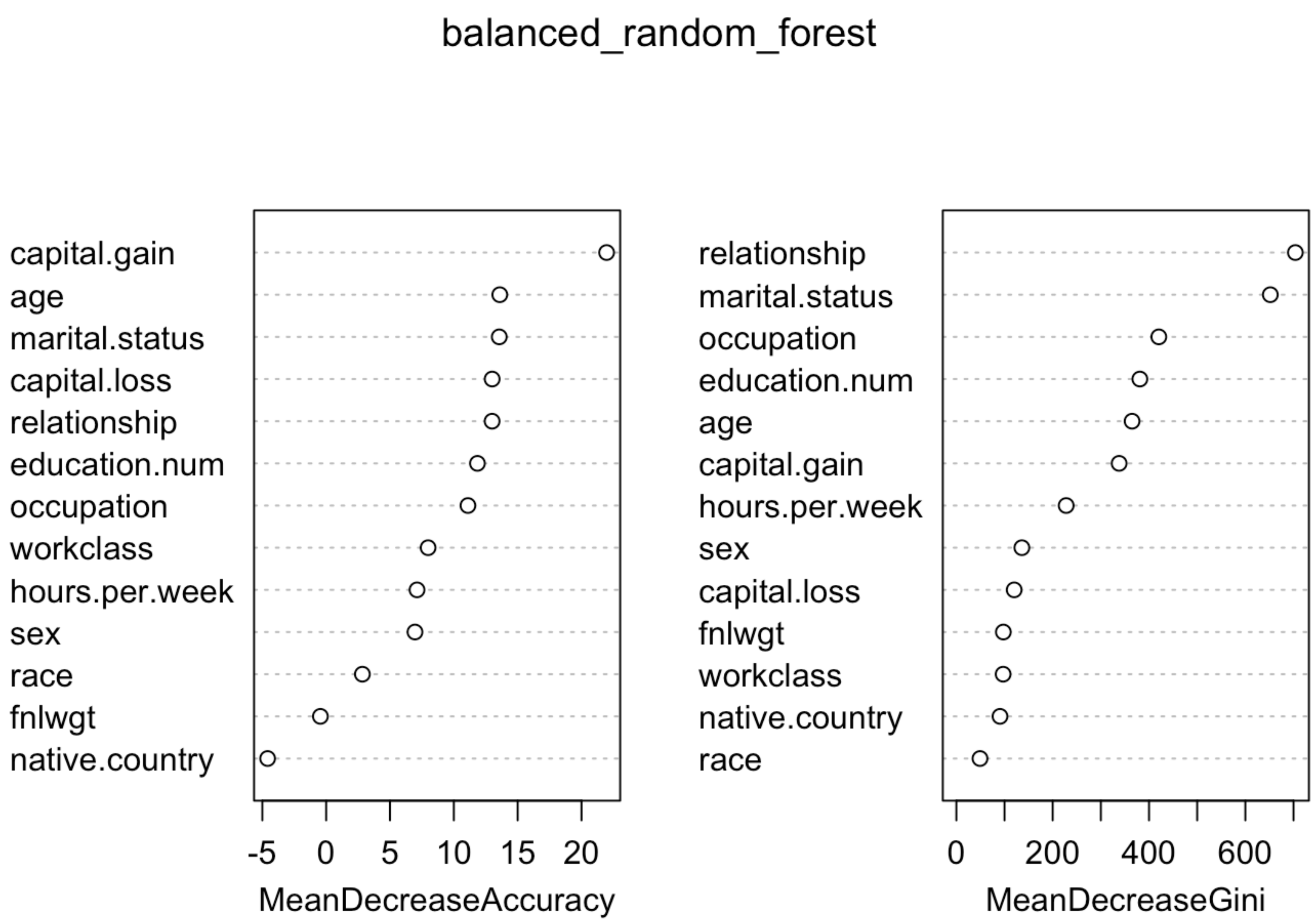



```
##       mtry  OOBError
## 1.OOB    1 0.2495338
## 2.OOB    2 0.2845631
## 3.OOB    3 0.3347096
## 6.OOB    6 0.3590170
```

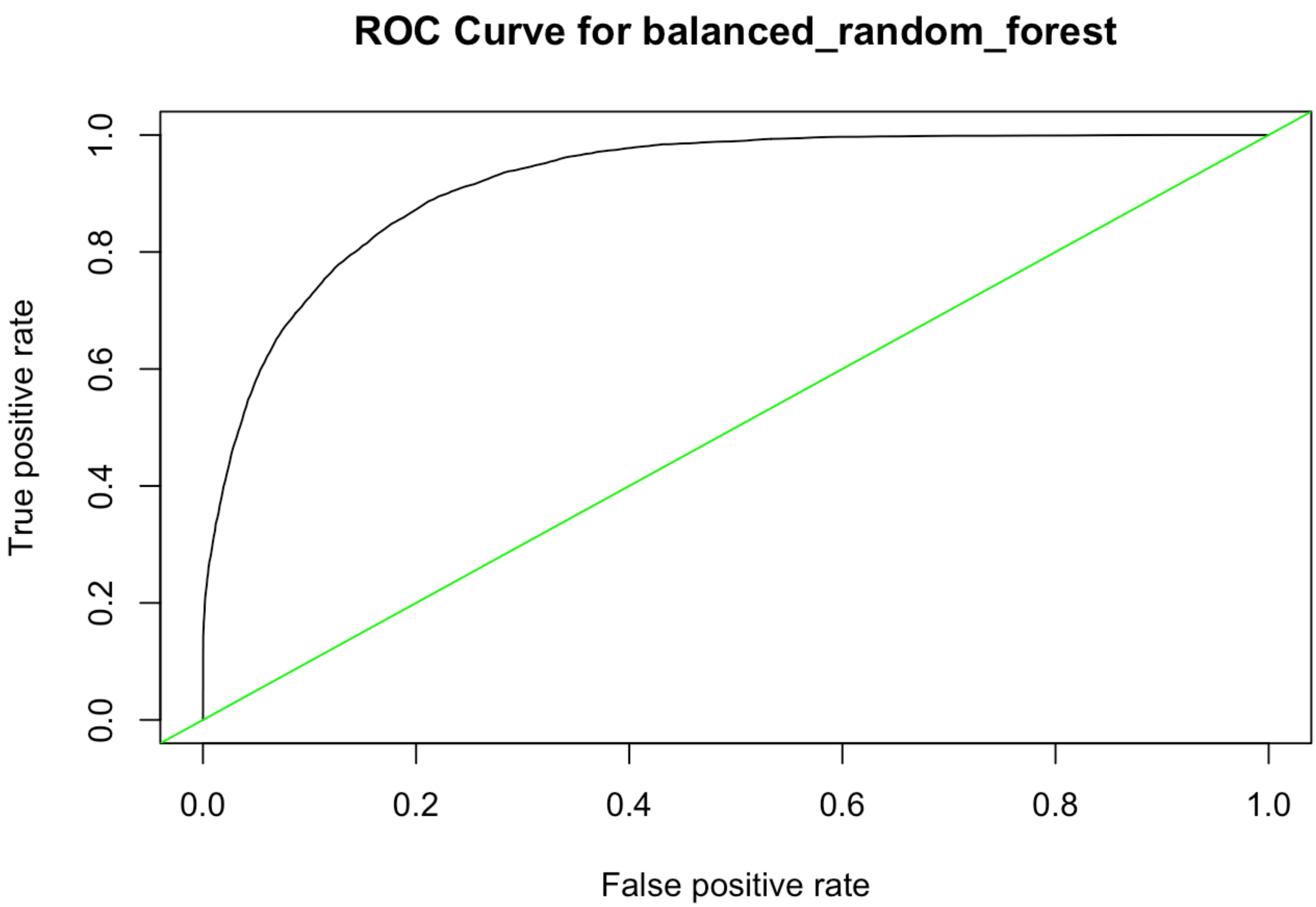
We see that the optimal value for mtry is 1. Then we fit the new balanced random forest using the function randomForest() with mtry = 1 and initially 250 trees(since computational costs are a bit lower). We then look at the plot of out-of-bag error against trees to verify if 250 trees is sufficient.



We see that the out-of-bag error seems to be roughly level after 180 trees, therefore 250 trees is sufficient. Then we look at plots of the most important variables.



The most important variables in our new balanced random forest are relationship, marital.status, ocupation, education.num, age, and capital.gain with mean decrease in Gini 703.82267, 651.84948, 420.33837, 380.99798, 364.83827, and 337.92745 respectively. We then plot the ROC curve and calculate the area under the curve.



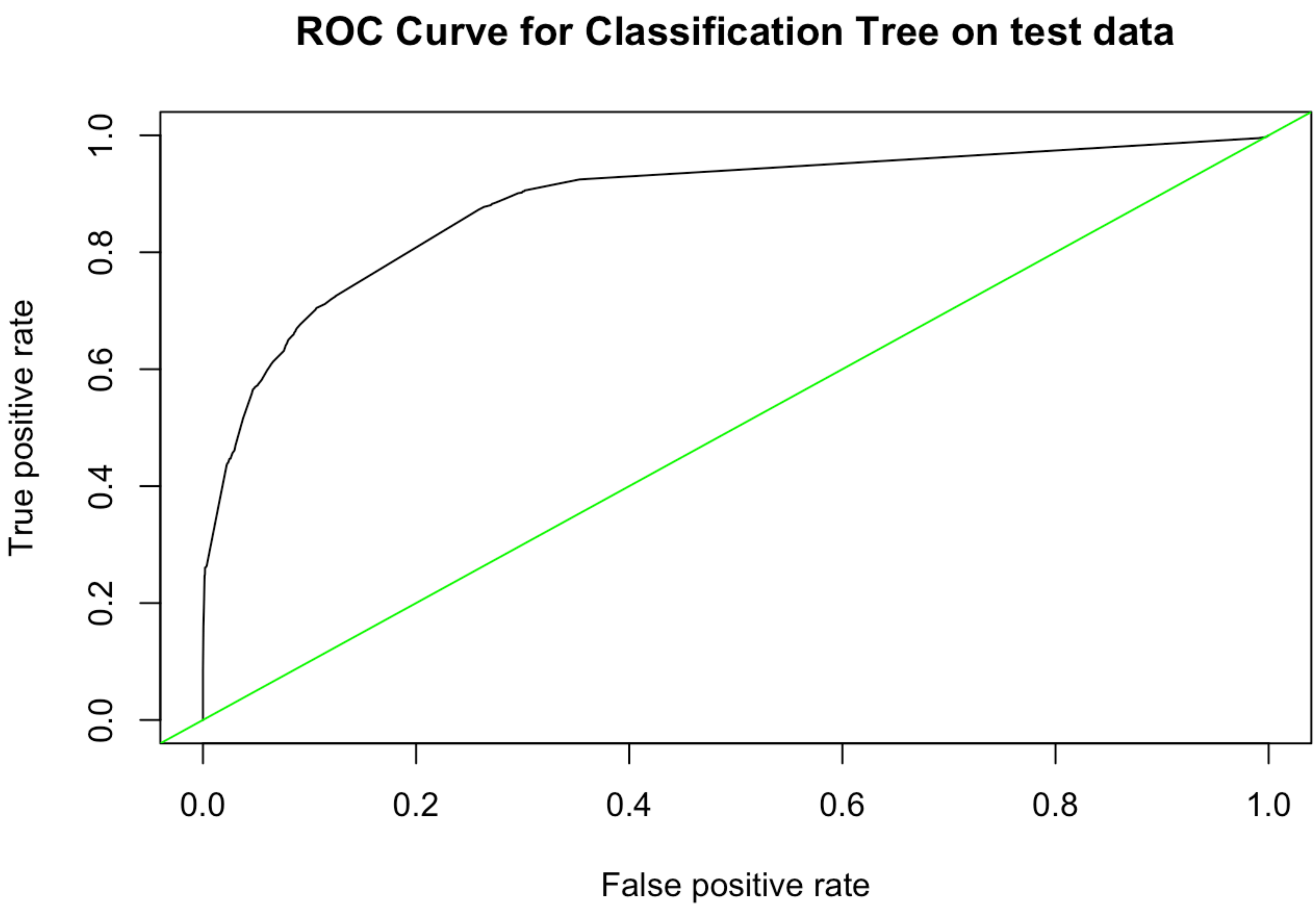
We see that our balanced random forest is much better than the no information classifier and the AUC of the balanced random forest is 0.919952. Lastly we want to calculate the training accuracy rate to finally determine if our balanced random forest trained on the undersampled training data has a higher training accuracy than our original random forest. We then make predictions on the original data set using the new balanced random forest and calculate that the balanced random forests training accuracy rate is 0.8612161 which is better than our original random forest who's rate was 0.8275976. We then consider our balanced random forest to be more favorable than our original random forest.

Model Validation

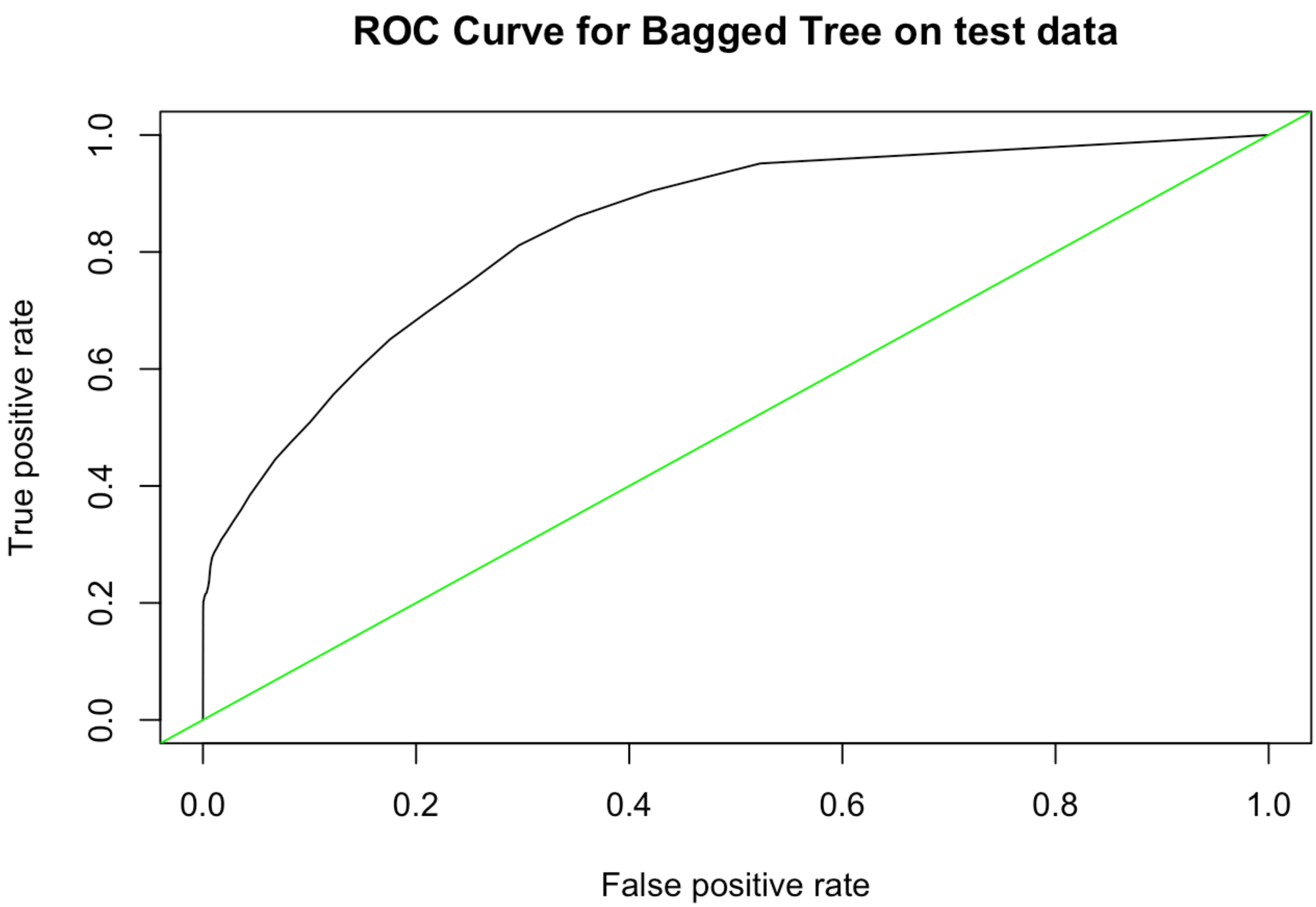
We will now take our best classifier from the previous sections and see how well it performs predicting on the test data. Since our classification tree has the highest training accuracy rate of 87.16597 percent. Therefore, we validate our classification tree on the test set using the function predict and classifying probabilities greater than .5 as belonging to class >50K. We get the test accuracy rate of 85.55777 percent for our classification tree. We can now compute the confusion matrix to see how the classfication tree performed on both classes on the test set.

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

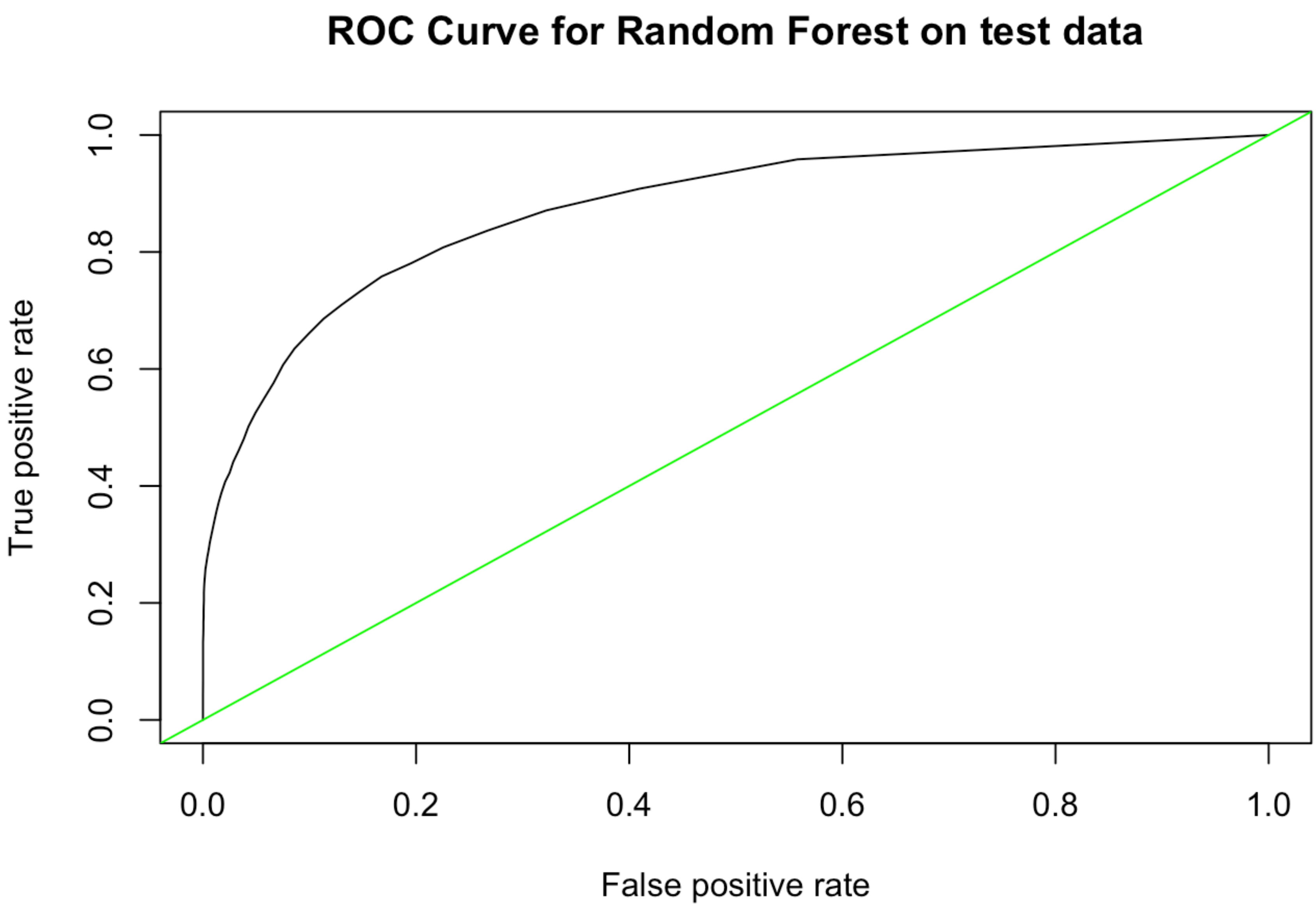
From a glance we see that our classification tree does well classifying observations with income <=50K correctly and classifies a majority of observations with income >50K correctly. However, we can get more precise and calculate the sensitivity(proportion of correctly identified positives aka true positive rate) and the specificity (proportion of correctly identified negatives aka 1 - false positive rate) from this confusion matrix. We calculate that the sensitivity is (2239 / (2239 + 1461)) = 0.6051351 and the specifity is 1 - (714 / (714 + 10646)) = 0.9371479. Lastly we can plot the ROC curves of all our classifiers predicted on the test set to see how all our classifiers performance compare over all thresholds. First we can plot the ROC curve for our classification tree predicted on the test set.



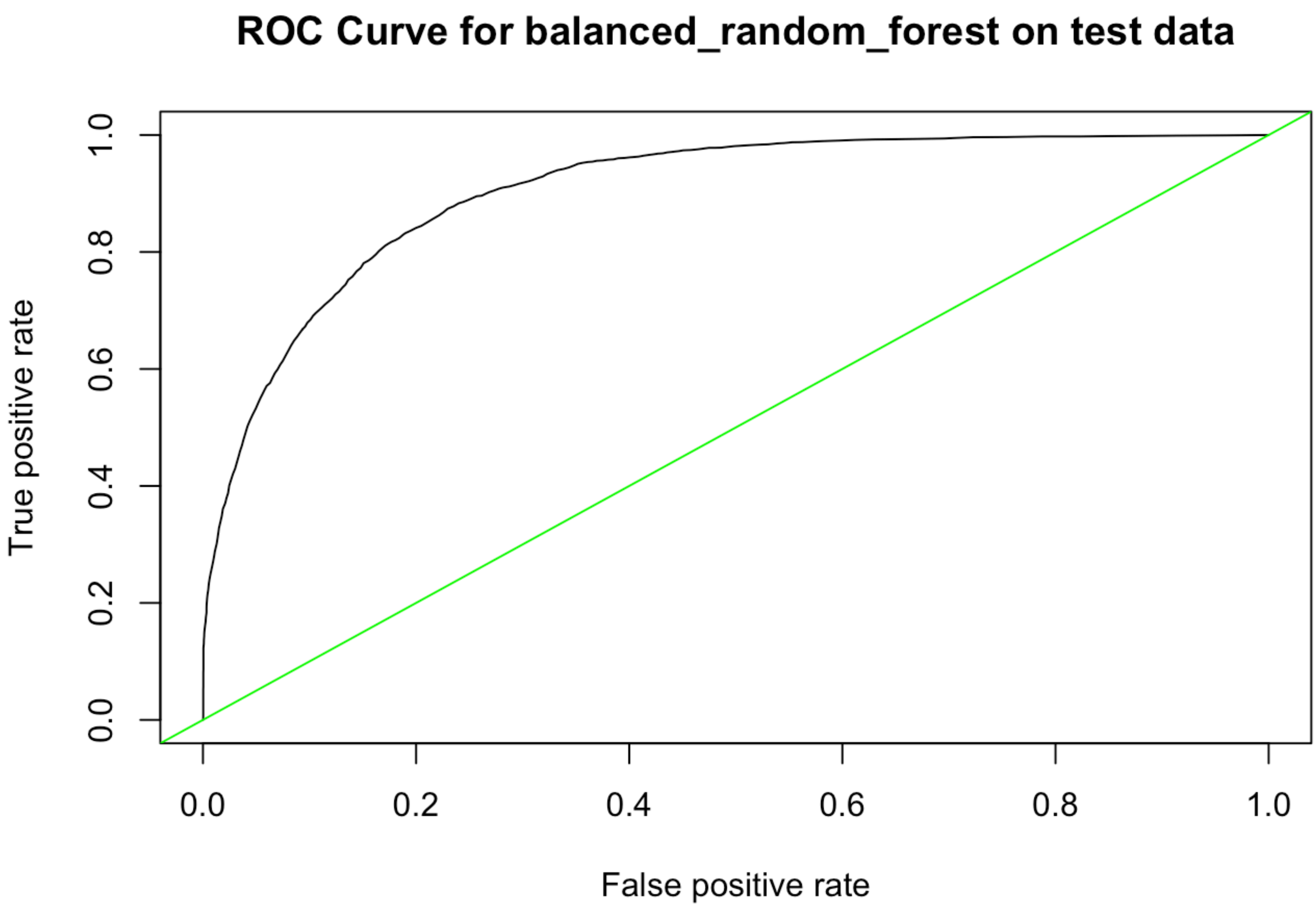
The area under the curve of the above ROC curve is 0.8842563. Next we can plot the ROC curve for our bagged tree predicited on the test data.



the area under the curve for the bagged tree predicted on the test data is 0.8387625. Next we will plot the ROC curve for our original random forest predicted on the test data.



The area under the curve for the above ROC curve is 0.8743767. Lastly we will plot the ROC curve for the balanced random forest (the random forest fit with the undersampled training set) predicted on the test set.



The area under the curve for the balanced random forest predicted on the test set is 0.9045263.

Conclusion

In conclusion, the best of the classifiers to predict whether an individual makes more than 50,000 dollars a year is our classification tree. The classification tree achieves our goal of determining which inviduals make over 50,000 dollars a year with a test accuracy of 85.55777 percent. One thing to note from our analysis is the improvement of our random forest when trained on a seperate balanced training set. In the future, it would be worthwhile to explore more options in tuning our random forest using different ways to treat our original imbalanced data set to see if we could get the random forest to surpass the test accuracy rate of the classification tree on this data. Further calculation shows the balanced random forest fit on the undersampled training data has a test accuracy rate of 0.8487384. Therefore the test accuracy of the balanced random forest is just less than the classification tree and it would be interesting to explore different options to improve our random forest in this way. It would also be interesting to see how much the training the bagged tree on a balanced data set would affect the bagged tree results.