# Income Classification with Census Income Data

Final Report

**DNSC 6279 Machine Learning and Data Mining**

**Authors** Laura Le, Yue Zheng, Lechen Tan

# Summary

## Problem of Interest

We are interested in discovering those factors that contribute to the difference in income of US adults. By figuring out the relationship between income and occupation, age, native country, race, capital gain, capital loss, education, work class, etc., we will predict whether a person can make over $50K a year based on their given profile. The problem is important since the understanding of factors that contribute to the wage difference could help the government make better decisions. A thorough analysis of the census data will reveal those critical factors and the decision makers could focus on policies and regulations that reduce the inequality derived from such attributes.

## Dataset Description

The Census Income Data Set was extracted by Barry Becker from the 1994 US Census Database. The data set consists of 14 attributes with anonymous information such as occupation, age, native country, race, capital gain, capital loss, education, work class and more. Each row is labelled as either having an annual income of ">50K" or "<=50K". This data set is already separated into train and test sets with 32561 rows and 16281 rows each.
The categorical attributes are: work class, education, education number, marital status, occupation, relationship, race, gender, native country. The continuous attributes are: age, final weight, capital gain, capital loss, hours per week. This data set is obtained from the UCI repository, and it can be found at: http://mlr.cs.umass.edu/ml/datasets/Census+Income.

## Machine Learning Methods

We will apply classification models in the order of logistic regression, Naive Bayes, decision tree, random forest, adaptive boosting and gradient boosting. At the beginning, simple models like binary logistic regression will act good as a baseline and be easy to interpret. From the results, we could understand the relative importance of the variables and whether they are relevant or not. We will then move on to tree-based models and fine-tune them with subtle adjustments. Finally, we will apply the tuned models to the test set and compare their performance by accuracy and AUC.

# Data Preprocessing

Our dataset is already very clean since there are no apparent misinput and only three columns with missing values. After reading in the data, we append the column names and row-wise bind the already-splitted train and test data into a single data frame.

We then carry out some simple exploratory data analysis. We first run the table function on all categorical variables. The result shows that **workclass**, **occupation** and **native.country** columns have NAs denoted with "?" which we will deal with later. Besides that, all categorical variables seem solid and sound to feed into models.

Then we also plot histograms of all continuous variables, **age** and **fnlwgt** columns seem right skewed and the **fnlwgt** column has some outliers.

The **capital.gain** and **capital.loss** variables are both continuous, however, with table function we discover that the majority of them is 0. It may be appropriate to transform these columns since they are better represented by dichotomous variables of whether they are 0 or not. However, we decide to discard these two variables since they exhibit almost no variance and seem not to be significant in some preliminary analysis.

We decide not to carry out standardizing or normalising and encoding. All of our models are unaffected by standardizing or normalising. These two techniques preserve the order of original variables. Thus, tree based models won't improve the results. Naive Bayes and logistic regression also don't care about the actual values. For encoding, we choose not to adopt the popular XGBoost model which mandates the use of dummy encoding. Instead, we explore the adaptive boosting and gradient boosting which are tolerant of normal categorical variables, same as our other models. Thus, encoding is unnecessary. Most importantly, tree-based models will perform best on original data in usual cases.

We then carry out imputation of unknown variables. We use the knn method for this task as it is most widely used and reliable. The other choice is mice library with more versatile methods options such as random forest. However, we choose knn since our dataset is medium-size, which will take huge amounts of time for random forest to execute one round of imputation. Knn also has the advantage of imputing all three columns at once. After imputation using the setting of 5 neighbours and all other complete columns, we use pie charts to check the validity of the imputation. The graphs show similar proportions for different levels in all columns we impute. We move on to replace the original column with the imputed column.

Finally, we split the whole dataset back into train and test data and their labels. Now we are fully prepared to enter the model fitting phase.

## Methods and Prediction

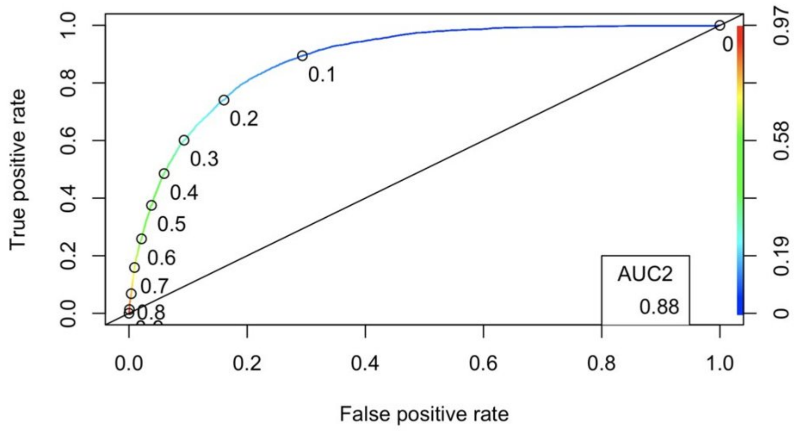
### Binary Logistic Regression

With the purpose of classifying people earning <=50k or >50k based on several explanatory factors affecting the income of a person like Age, Occupation, Education, etc. We believe that Logistic Regression will be the first fundamental and essential method that we should apply on this dataset.

To check how well is our builded-logistic regression model, we need to calculate predicted probabilities. Our calculated probabilities also need to be classified. In order to do that, we need to decide the threshold that best classifies our predicted results.

```
          0     1
<=50K 20333  4387
>50K   1784  6057
```

The accuracy result shows that the logistic regression can predict correctly 81%. Therefore, this is a good model because the sensitivity and specificity percentages of this model are also large.



The confusion matrix for the test set:

```
           0     1
<=50K. 10236  2199
>50K.    896  2950
```

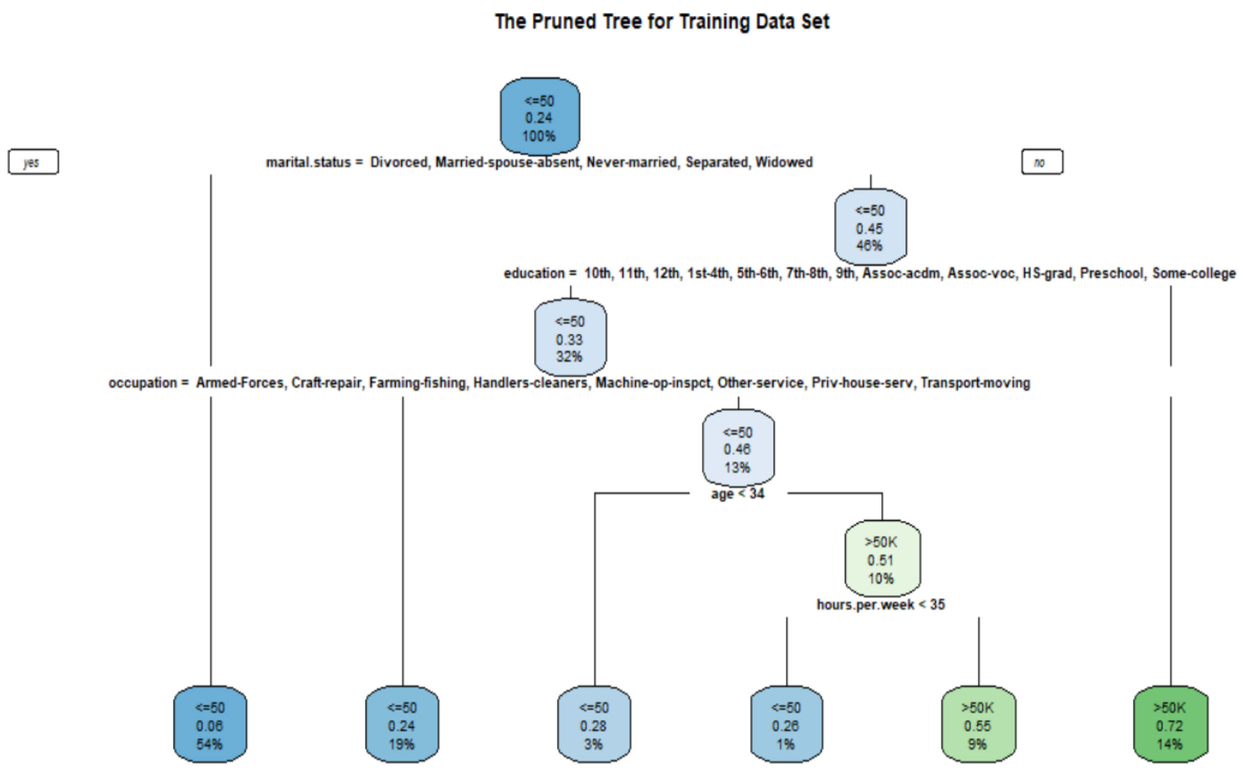
We can conclude that the accuracy of this model is about 82% with 88% of our predicted values lying under the curve. The misclassification rate is 18%. Therefore, this is a good model because the sensitivity and specificity percentages of this model are also large.

## Decision Tree

Decision tree models are easy to understand and interpret, and they have fast algorithms for both learning and prediction. So we think it's a good approach to classify the observations in our dataset. Since the response variable, 'income', has two levels in total, ' <=50K' and ' >50K', we can create a classification tree for it.

Firstly, we created a full tree based on the training set. And then prune the tree to select a subtree that leads to the lowest test error. By finding and applying the optimal complexity value (0.01) associated with the minimum error to the model, we completed tree pruning. The pruned tree is exactly the same as the full tree, which means that the full tree has been optimal enough.

From the output shown below, we can know about how the branches were split in the pruned tree, and how the tree plot looks like.


The Pruned Tree for Training Data Set
<=50 0.24 100%
yes
marital.status = Divorced, Married-spouse-absent, Never-married, Separated, Widowed
no
<=50 0.45 46%
education = 10th, 11th, 12th, 1st-4th, 5th-6th, 7th-8th, 9th, Assoc-acdm, Assoc-voc, HS-grad, Preschool, Some-college
<=50 0.33 32%
occupation = Armed-Forces, Craft-repair, Farming-fishing, Handlers-cleaners, Machine-op-inspct, Other-service, Priv-house-serv, Transport-moving
<=50 0.46 13%
age < 34
>50K 0.51 10%
hours.per.week < 35
<=50 0.06 54%
<=50 0.24 19%
<=50 0.28 3%
<=50 0.26 1%
>50K 0.55 9%
>50K 0.72 14%

The tree plot is very easy to interpret. For example, the top node shows the overall proportion of income. The top node tells us 24% of people's income is greater than $50K. The node on the left tells us there are 54% of people whose marital status are divorced, married spouse absent, never married, separated and widowed. And only 6% of this group has income greater than $50K per year. So in this way, we can check how it divides at each branch from the top to bottom.

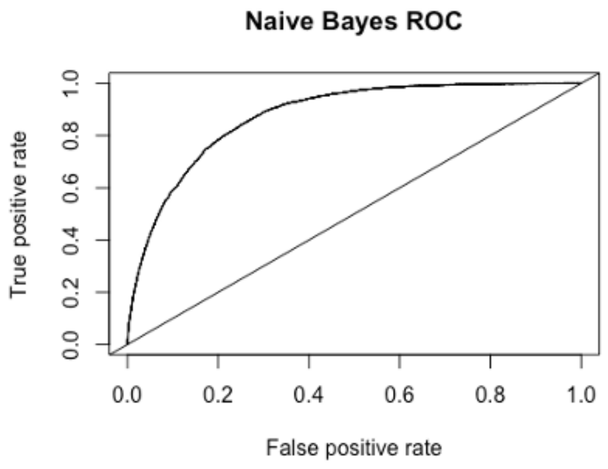
The confusion matrix for test data set:

```
Confusion Matrix and Statistics

          Reference
Prediction  <=50K  >50K
     <=50K  11112  1521
     >50K    1323  2325
```

The accuracy rate for the test set is 83%, and the AUC value is 83%, which means the classification tree model has good performance to predict the test data set in general. Maybe that’s because the response variable, ‘income’, is a binary variable, which is either ‘ <=50K’ or ‘ >50K’, meaning that the decision tree would have the good structure for predicting such a label.

## Naive Bayes

Naive Bayes is a very simple model. It has a naive assumption that all predictors in the data set are independent of each other. Then it will apply Bayes statistical method with overall possibility as prior possibility and possibility within each level of a categorical variable as posterior possibility. Since such assumptions are never met in real life, the performance of the model is hindered. However, usually the result will be acceptable, and it is very easily trained with training time linear in the number of variables. Since our dataset is medium-size, it will act good as a baseline and provide us with reference.



After fitting the model and predicting on test sets, we obtain accuracy of 82.77% and AUC of 87.67% which is not bad and comparable to logistic regression given the shorter training time.

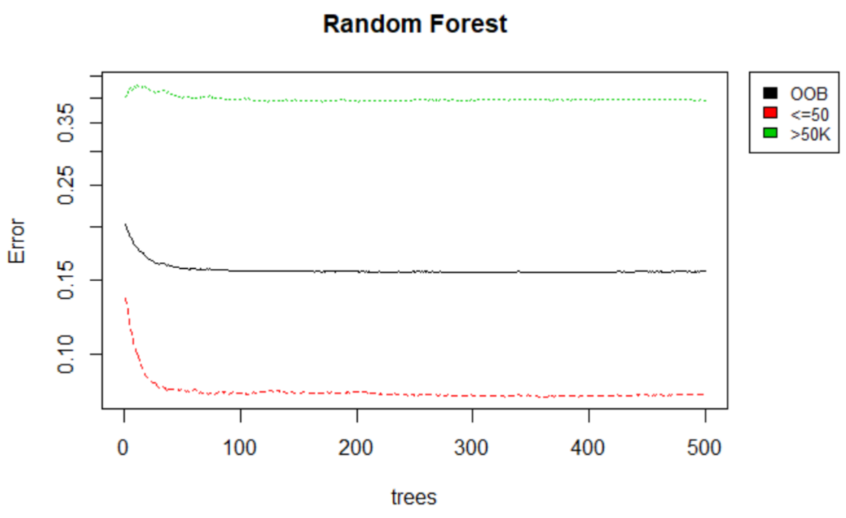
## Random Forest

Based on ensemble learning methods, random forest models are very effective in classification. And since it's robust to overfitting, we choose to build it for our dataset.

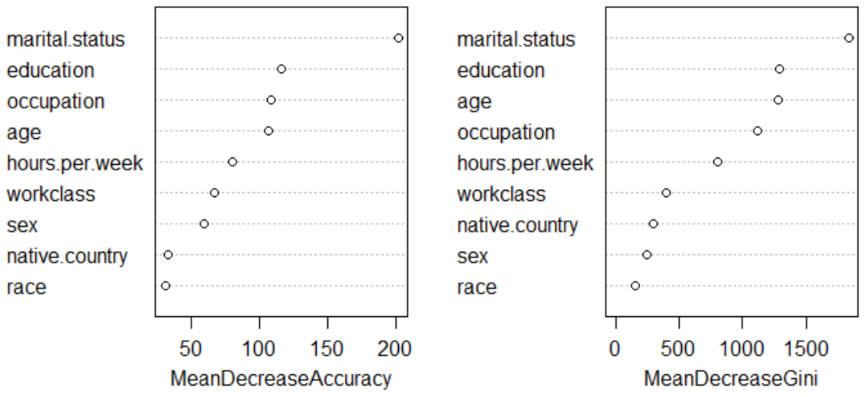
To find the optimal number of variables randomly sampled as candidates at each split ('mtry'), we tried several possible values.

| mtry | OOB Rate |
|---|---|
| 2 | 15.68% |
| 3 | 16.45% |
| 4 | 17.21% |
| 5 | 17.71% |
| 6 | 18.23% |

The table above shows the comparison of out-of-bagging rates for different numbers of variables sampled at each split. When selecting 2 variables randomly sampled at each split, the OOB (out-of-bagging) rate will be the lowest. So we choose it as 2, and the number of trees as the default number, 500.



As we can see from the error plot, as the number of trees increases, the error rate will tend to be stable, and that illustrates that the random forest model will not overfit.



The two scatter plots reflect the importance of each feature in the random forest model. As we can see, the most important variable is marital.status, which means marital status plays the most important role when predicting income. Besides, some other factors such as age, education, and occupation also have high importance in this model.

The confusion matrix for test data set:

```
Confusion Matrix and Statistics

          Reference
Prediction   <=50  >50K
      <=50 12074   361
      >50K  2489  1357
```

The accuracy rate for the test set is 82.49% and the AUC value is 87.71%, which means the random forest model has good performance for prediction in the test set.

## Gradient Boosting

Based on the form of an ensemble of weak prediction models, gradient boosting usually provides good predictive accuracy, so we think it's a good choice to fit our dataset.

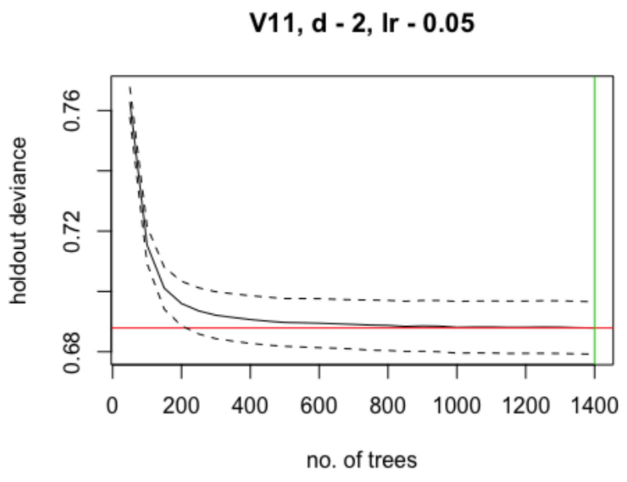
To minimize the errors of our gradient boosting model, we tuned the parameters learning rate, interaction depth, and tree size.

Learning rate and interaction depth tuning:

| Learning Rate | Accuracy Rate |
|---|---|
| 0.5 | 82.27% |
| 0.25 | 82.47% |
| 0.1 | 82.54% |
| 0.05 | 82.70% |
| 0.01 | 82.72% |

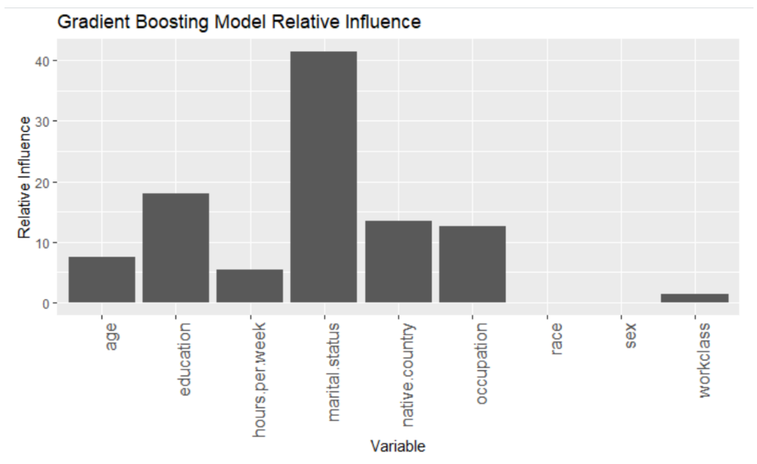
| Interaction Depth | Accuracy Rate |
|---|---|
| 1 | 82.82% |
| 2 | 82.56% |
| 3 | 82.83% |
| 4 | 82.58% |
| 5 | 82.65% |

The tree size tuning:

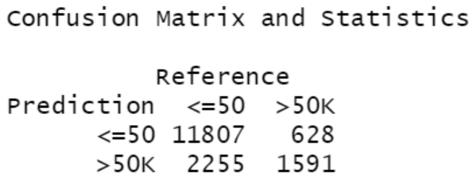


From the plot above, the optimal number of trees should be 1400. Because when it's 1400, the holdout deviance will be the lowest.

After tuning the parameters, we decided to apply learning rate = 0.01, interaction depth = 3, and tree size = 1400 in our final gradient boosting model.



The bar plot above shows the relative influence of each feature in the gradient boosting model. Clearly, marital status has the highest value among all the features, which is the same as the conclusion from other models. And the features race and sex have very low relative influence, which means race and sex are relatively insignificant when prediciting income.



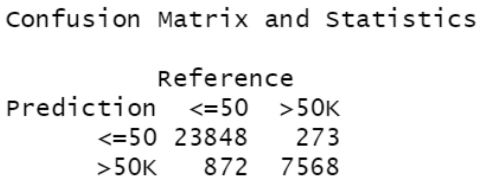
```
Confusion Matrix and Statistics

                Reference
Prediction  <=50  >50K
      <=50 11807   628
      >50K  2255  1591
```

The accuracy of the final gradient boosting model is 82.49%, and the AUC value is 88.95%.

## Adaptive Boosting

Similar to gradient boosting, adaptive boosting is based on building weak learners in a sequential fashion as well. But compared to gradient boosting, adaptive boosting weighs higher on misclassified observations.

We built the adaptive boosting model by applying the package 'fastAdaboost' in R. Besides the basic Adaboost model, we also apply the Real adaboost function to train the dataset. The difference between AdaBoost and Real AdaBoost is: AdaBoost is based on errors in predicted class labels, whereas Real AdaBoost uses the predicted class probabilities. The accuracy of the Real AdaBoost model is 81.16%, which is lower than that of AdaBoost model (81.71%), so our final choice is the AdaBoost model.



```
Confusion Matrix and Statistics

                Reference
Prediction  <=50  >50K
      <=50 23848   273
      >50K   872  7568
```

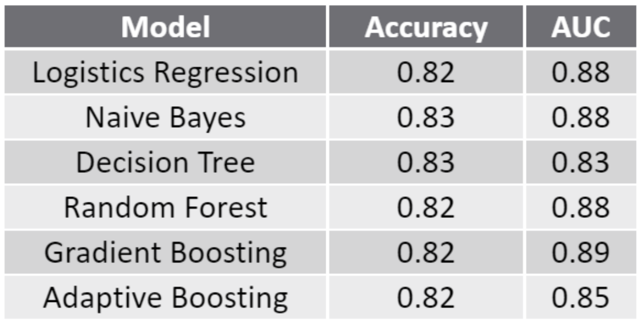
The accuracy of the final adaptive boosting model is 81.71%, and the AUC value is 84.82%.

# Conclusion

Criteria to select the best model to fit the US Adult Income:

- Test accuracy
- Interpret ability

Compare the models' accuracy on the test set:



| Model | Accuracy | AUC |
|---|---|---|
| Logistics Regression | 0.82 | 0.88 |
| Naive Bayes | 0.83 | 0.88 |
| Decision Tree | 0.83 | 0.83 |
| Random Forest | 0.82 | 0.88 |
| Gradient Boosting | 0.82 | 0.89 |
| Adaptive Boosting | 0.82 | 0.85 |

Since there is no big difference between accuracy of these methods, we will choose the one that is the easiest to interpret and understand for both technical and non-technical stakeholders. Therefore, the best model that we selected for this data is the Decision Tree.

In conclusion, based on this Decision Tree model, we can predict annual incomes of US Adults based on 20 parameters with an accuracy of 83%. This model can be applied to data from other Census years, or income evaluation in credit card applications, or tax analysis because it can continuously learn from changing data in the training set to adapt to new parameters, thus improving its accuracy and other metrics. The model will perform better and more meaningful if using predictors that related to income such as: age, hour per week, education, occupation and sex.

On the other hand, for this dataset, we suggest using categorical instead of numerical values for data analysis to make the predictions. For example, we can group 21 and 24 year-olds to a category because a person in this age would not be expected to make more than $50,000 a year in 1994 (according to dollarsigns.com report), that would be the equivalent of about $80,000 today. By grouping them up to the same inference, the noise can be reduced and we can apply other models such as KNN. This can also help speed up the model's training as well.

# Appendix

***Remarks:***

1. *Only shows the selective coding because of space limitation*
2. *The coding output would have very slight differences from the output data in report because of random generation in R*

## Work Distribution

Laura Le: logistic regression, conclusion

Yue Zheng: decision tree, random forest, gradient boosting, adaptive boosting

Lechen Tan: data preprocessing, exploratory graphs, naive bayes

## 1.Data Preprocessing

Load the data

```
# Load 'adult' dataset, replace " ?" values with "NA" values.

training <- read.csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data',
```

```r
header = FALSE, na.strings = c(" ?","NA"))

# Name the columns

colnames(training) <-
c("age","workclass","fnlwgt","education","education.num","marital.status","occupation","relationsh
ip","race","sex","capital.gain","capital.loss","hours.per.week","native.country","income")

# Load 'test' dataset, replace " ?" values with "NA" values, skip the first row of data.

testing <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test",
header = FALSE, skip = 1, na.strings = c(" ?","NA"))

# Name the columns

colnames(testing) <-
c("age","workclass","fnlwgt","education","education.num","marital.status","occupation","relationsh
ip","race","sex","capital.gain","capital.loss","hours.per.week","native.country","income")

# Rename the factors of testing to be consistent with training
levels(training$income) <-c("<=50",">50K")
levels(testing$income) <-c("<=50",">50K")
levels(testing$native.country) <- levels(training$native.country)

nrow(training)

## [1] 32561

nrow(testing)

## [1] 16281

train=1:32561
test=-(train)

#Check the summary of data:
full  <- bind_rows(training, testing)

str(full)
```

KNN imputation

```r
full1 <- kNN(full, variable = c('workclass', 'occupation','native.country'), k = 5)
```

Check validity of the imputation

```r
pie(table(full$workclass), main="workclass: Original")
pie(table(full1$workclass), main="workclass: Imputed")
```

```r
#Train and test set split
x.train <- full1[train,]#put regressors from training set into a matrix
y.train <- full1[train,]$income #label for training set
x.test <- full1[test,]#put regressors from test set into a matrix
y.test <- full1[test,]$income #label for test set
```

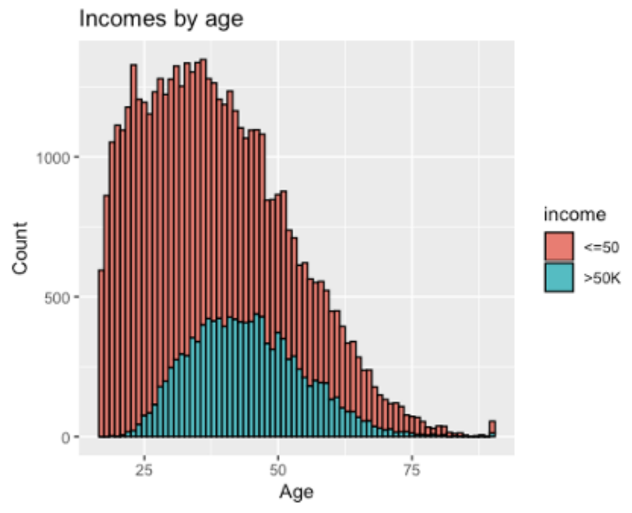
## 2.Exploratory Data Analysis

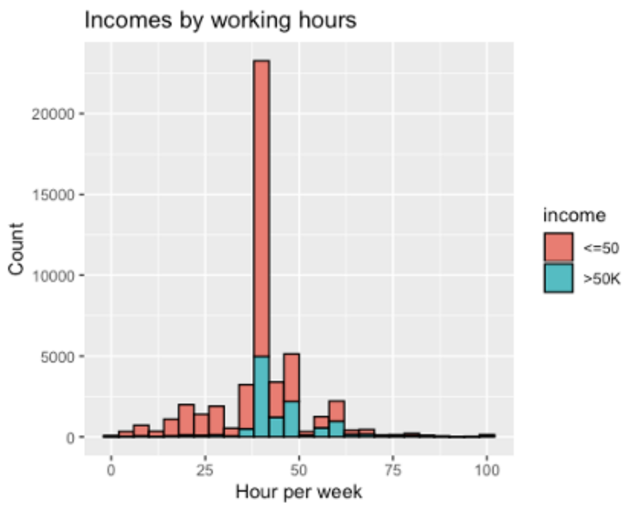
*Check the relationship between income and age*

```r
ggplot(full) + aes(x=as.numeric(age), group = income, fill = income) +
  geom_histogram(binwidth = 1, color = "black") +
  labs(x="Age", y = "Count", title = "Incomes by age")
```

*Check the relationship between income and hours per week*

```r
ggplot(full) + aes(x=as.numeric(hours.per.week), group = income, fill = income) +
  geom_histogram(binwidth = 4, color = "black") +
  labs(x="Hour per week", y = "Count", title = "Incomes by working hours")
```



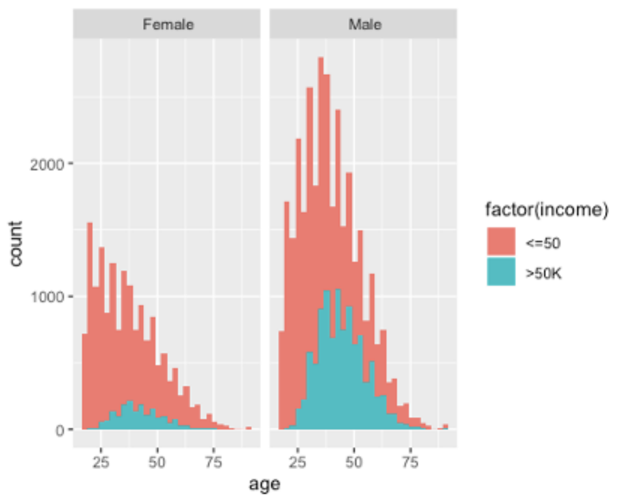


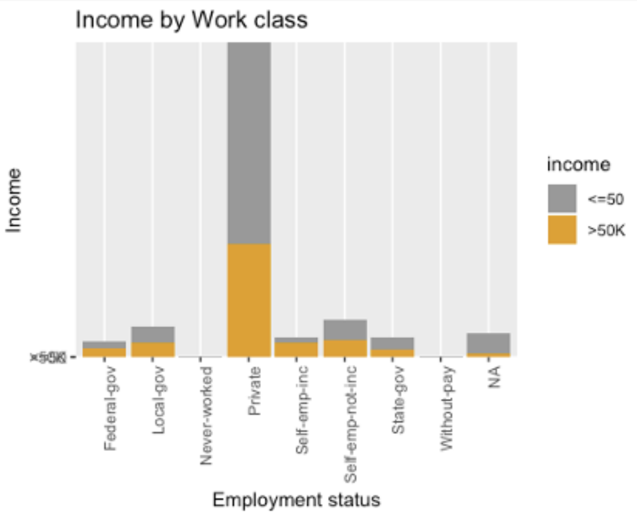
*Check the relationship between income and gender*

```
ggplot(full, aes(age, fill = factor(income))) +
  geom_histogram() +
  facet_grid(.~sex)
```

*Check Income by Workclass*

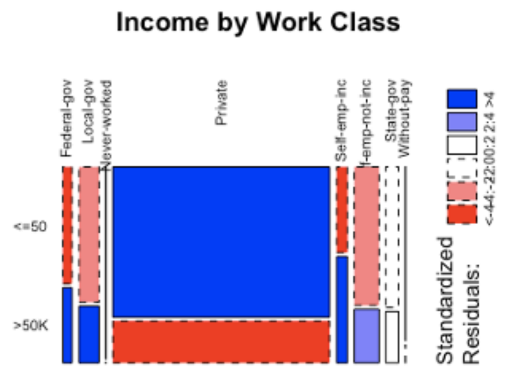
```
ggplot(data= full, aes(x=workclass, y = income, fill=income)) +
  geom_bar(stat="identity") +
  labs(title="Income by Work class",
       x="Employment status", y = "Income") +
  scale_fill_manual(values=c("#999999", "#E69F00")) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

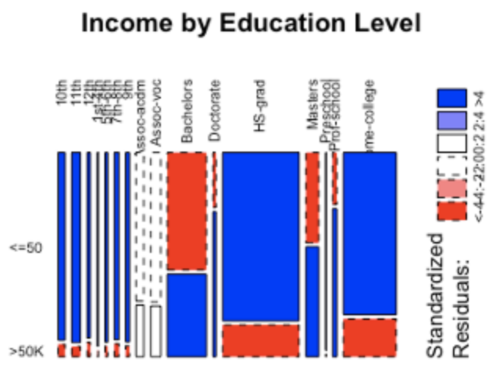


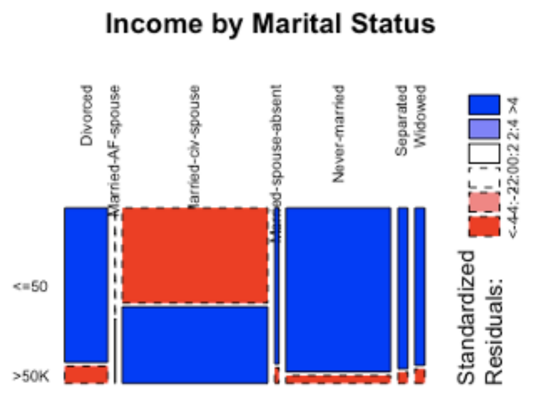


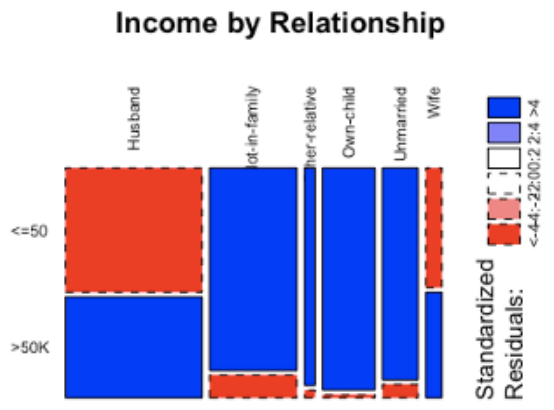
*Mosaic Plots*

```
par(las=2)
mosaicplot(table(full$workclass, full$income), main='Income by Work Class', shade=TRUE)
par(las=2)
Mosaicplot(table(full$education, full$income), main='Income by Education Level', shade=TRUE)
par(las=2)
mosaicplot(table(full$marital.status, full$income), main='Income by Marital Status', shade=TRUE)
par(las=2)
mosaicplot(table(full$relationship, full$income), main='Income by Relationship', shade=TRUE)
```





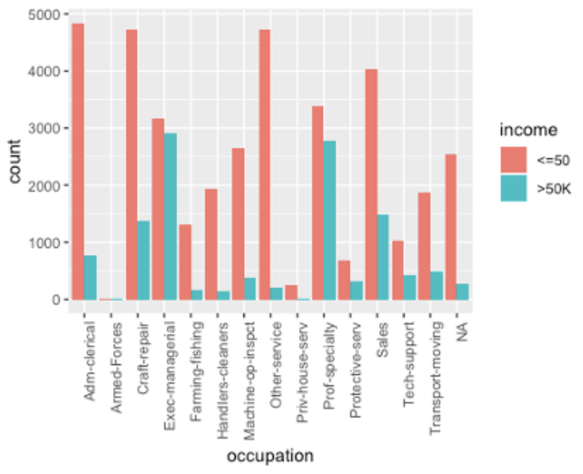


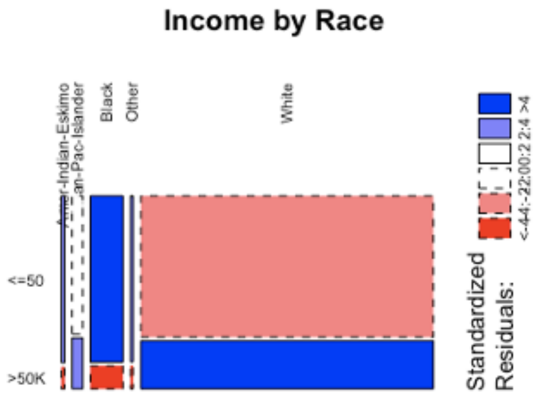


```
#Income by race
par(las=2)
mosaicplot(table(full$race, full$income), main='Income by Race', shade=TRUE)
```

*Check Income by Occupation*

```
ggplot(full, aes(x = occupation, fill = income))+
  geom_bar(stat='count', position='dodge') +
  labs(x = 'occupation')+ theme(axis.text.x=element_text(angle=90,hjust=1))
```





## 3.Logistic Regression

```
# model implementation

adult_lg<-glm(income~.,data = x.train,family = "binomial")

# argument (family = "binomial") is necessary as we are creating a model with dichotomous result
```

*Add column predicted probabilities to the training dataset*

```
lm.pred<- fitted(adult_lg)

#Compare predicted values with the actual values in the training set

prediction <- prediction(lm.pred, y.train)

# Stores the measures with respect to which we want to plot the ROC graph

perf<-performance(prediction,"tpr","fpr")

# We assign that threshold where sensitively and specifically have almost similar values after
observing the ROC graph

lm.pred<-ifelse(lm.pred < 0.5,0,1)

# This 'predict_income' column will classify probabilities we calculated and classify them as 0 or
1 based on our threshold value (0.5) and store in this column
```

Creating confusion matrix and assessing the results:

```
# Creating confusion matrix
train_log <- table(lm.pred,y.train)

# Calculate accuracy
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
accuracy (train_log)

## [1] 83.47102
```

*Check how well this builded model fit the test data*

```
log_predict_prob_income <- predict(adult_lg, x.test, type = "response")

# an extra argument(type = "response") is required while using 'predict' function to generate
response as probabilities
# this argument is not required while using 'fitted'

log_predict_income<-ifelse(log_predict_prob_income < 0.5,0,1)

# we take the same threshold to classify which we considered while classifying probabilities of
training data

#head(test2)
```

Creating confusion matrix and assessing the results:

```
# Creating confusion matrix
test_log <- table(y.test,log_predict_income)


# Plots the ROC curve
```

```
# We assign that threshold where sensitively and specifically have almost similar values after
observing the ROC graph

# Calculate accuracy
accuracy (test_log)

## [1] 81.83158

prediction_test <- prediction(log_predict_prob_income, y.test)

auc<-performance(prediction_test,"auc")

auc@y.values

## [[1]]
## [1] 0.882189
```

## 4.Decision Tree

*Implement the decision tree:*

```
adult_tree <- rpart(income~., data = x.train)

tr_predict_income <- predict (adult_tree,x.test,type = "class")
```

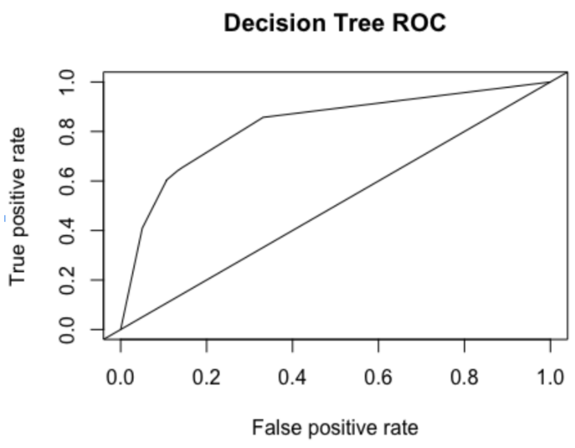
Creating confusion matrix and assessing the results:

```
# Creating confusion matrix
test_tree <- table(y.test,tr_predict_income)

# Calculate accuracy
accuracy (test_tree)

## [1] 82.46422
```

We are getting an model accuracy of 82.46%

```
#This function returns the optimal complexity value associated with the minimum error.
optimal_cp = adult_tree$cptable[which.min(adult_tree$cptable[,"xerror"]),"CP"]
optimal_cp

## [1] 0.01

# Prune the tree by applying the optimal complexity value
adult_tree.prune<- prune(adult_tree, cp= optimal_cp)
adult_tree.prune

## n= 32561
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 32561 7841 <=50 (0.75919044 0.24080956)
##    2) marital.status= Divorced, Married-spouse-absent, Never-married, Separated, Widowed 17562
1139 <=50 (0.93514406 0.06485594) *
##    3) marital.status= Married-AF-spouse, Married-civ-spouse 14999 6702 <=50 (0.55317021
0.44682979)
##      6) education= 10th, 11th, 12th, 1st-4th, 5th-6th, 7th-8th, 9th, Assoc-acdm, Assoc-voc, HS-
grad, Preschool, Some-college 10526 3484 <=50 (0.66901007 0.33098993)
##       12) occupation= Armed-Forces, Craft-repair, Farming-fishing, Handlers-cleaners, Machine-
op-inspct, Other-service, Priv-house-serv, Transport-moving 6215 1510 <=50 (0.75703942 0.24296058)
*
##       13) occupation= Adm-clerical, Exec-managerial, Prof-specialty, Protective-serv, Sales,
Tech-support 4311 1974 <=50 (0.54210160 0.45789840)
##         26) age< 33.5 1033  287 <=50 (0.72216844 0.27783156) *
##         27) age>=33.5 3278 1591 >50K (0.48535692 0.51464308)
##           54) hours.per.week< 34.5 403  105 <=50 (0.73945409 0.26054591) *
##           55) hours.per.week>=34.5 2875 1293 >50K (0.44973913 0.55026087) *
##      7) education= Bachelors, Doctorate, Masters, Prof-school 4473 1255 >50K (0.28057232
0.71942768) *

# Plot the pruned tree
rpart.plot(adult_tree.prune, cex = 0.6, main="The Pruned Tree for Training Data Set")

#making predictions for test set
tree.pred=predict(adult_tree.prune,x.test,type="class")

#Create the confusion matrix for test set
confusionMatrix(tree.pred, y.test)


# The accuracy rate is 0.8246

# Create the roc plot

dt_auc
```

```
## [1] 0.8248358
```



## 5.Naive Bayes

```
set.seed(1)
nb.adult1 <- naiveBayes(income ~ ., data = x.train)
nb.adult1
#Check the test error
nb_pred <- predict(nb.adult1, x.test)
confusionMatrix(nb_pred, y.test)
nb_pred = predict(nb.adult1, x.test, type="raw")
nb_prediction <- prediction(nb_pred[,2], y.test)
nb_roc<-performance(nb_prediction,"tpr","fpr")
plot(nb_roc, main = "Naive Bayes ROC")
nb_auc <- performance(nb_prediction,"auc")
nb_auc <- unlist(slot(nb_auc, "y.values"))
nb_auc
## [1] 0.8767256
abline(a=0,b=1)
```

## 6.Random Forest

```
## Apply bagging on the decision tree. The default number of trees is 500. To improve the model
performance, we can try different numbers of "mtry" and check performance.

set.seed(1)
bag.adult2 = randomForest(income~.,data=x.train, mtry=2, importance=TRUE)
bag.adult2

# The OOB rate is 15.68% when mtry = 2
```

The Out of Bag error (OOB) gives us the miscalssification rate (MCR) of the model. In this case it comes out to be 15.79%, which gives us the accuracy of 84.21%

```
set.seed(1)
bag.adult3 = randomForest(income~.,data=x.train, mtry=3, importance=TRUE)
bag.adult3
bag.adult4 = randomForest(income~.,data=x.train, mtry=4, importance=TRUE)
bag.adult4
bag.adult5 = randomForest(income~.,data=x.train, mtry=5, importance=TRUE)
bag.adult5
bag.adult6 = randomForest(income~.,data=x.train, mtry=6, importance=TRUE)
bag.adult6
```

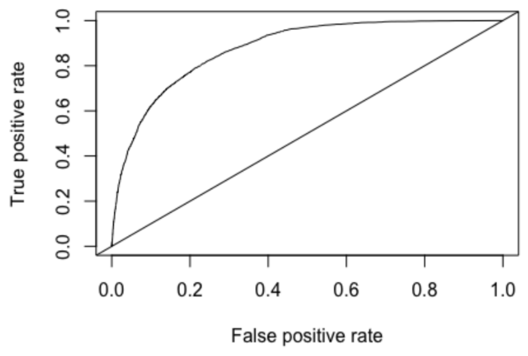
Since the OOB rate will be lowest when mtry = 2 after trying different numbers of mtry, we will select 2 as the mtry value for the final random forest model. Its OOB rate comes out to be 15.68%, which gives us the accuracy of 84.32%

*Check classwise error*

```
plot(bag.adult2)
```

The red line represents MCR of class <= 50k, the green line represents MCR of class >50k and black line represents overall MCR or OOB error. We need to find out the overall MCR and it is considered quite good.

```
#Check the importance of the variables in the dataset

varImpPlot(bag.adult2)

importance(bag.adult2)

##                    <=50        >50K MeanDecreaseAccuracy MeanDecreaseGini
## age             12.06740 109.2186743            106.52284        1285.4233
## workclass       57.10568  26.7889218             66.43960         397.7282
## education       70.83489  80.6843124            115.98325        1291.9854
## marital.status 108.95243 180.5184529            202.18943        1839.0792
## occupation      66.58974  69.1525004            108.15380        1118.1444
## race            22.70577  16.3107299             30.54472         154.8794
## sex             44.12429  10.0022536             59.34527         250.6416
## hours.per.week  25.67321  73.7177952             79.95946         802.8507
## native.country  39.37676  -0.2563874             32.96597         296.1373

# Create the roc plot

rf_auc

## [1] 0.877062
```

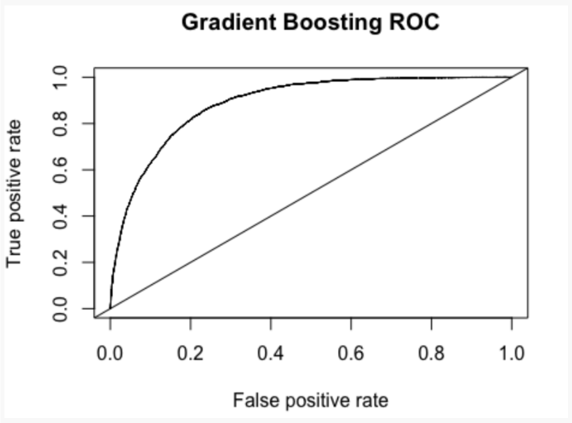


```
# The auc value is 0.8771
```

## 7.Gradient Boosting

```
# Initially, we build the gradient boosting model by selecting n.trees = 2000 and shrinkage = .05. And we will do parameter tuning afterwards.

# The predicted result is numeric data. To evaluable the model performance, we'll get the prediction of 'income' with the highest prediction value.

# The accuracy rate for the test set is 0.8194, which means the gradient boosting model fits the data set well.

# learning rate tuning
learning_rates = c(0.5, 0.25, 0.1, 0.05, 0.01)
for (i in learning_rates)
{
gb_adult2 = gbm(income ~.,
              data = x.train,
              distribution = "multinomial",
              cv.folds = 10,
              shrinkage = i,
              n.minobsinnode = 10,
              n.trees = 200)
pred2 = predict.gbm(object = gb_adult2,
                    newdata = x.test,
                    n.trees = 200,
                    type = "response")

gb_income2 = colnames(pred2)[apply(pred2, 1, which.max)]
gb_accu = accuracy(table(y.test, as.factor(gb_income2)))
print(paste(i, ":",gb_accu))

}

## [1] "0.5 : 82.2676739758"
## [1] "0.25 : 82.4703642282415"
## [1] "0.1 : 82.537927645722"
## [1] "0.05 : 82.7958970579203"
## [1] "0.01 : 82.7160493827161"

# As we can see from the output, when the learning rate is 0.05, the accuracy rate will be lower
than the others. So we will select 0.05 as the learning rate for this model.

# Interaction.depth tuning
interaction_depth = c(1:5)

## [1] "1 : 82.8204655733677"
## [1] "2 : 82.5563540323076"
## [1] "3 : 82.8327498310915"
## [1] "4 : 82.580922547755"
## [1] "5 : 82.6484859652356"

# As we can see from the output, when interaction_depth = 1, the accuracy rate will be lowest. So
we will select 1 as the interaction depth for this model.

# Tree size tuning
# Calculate the optimal number of trees by applying gbm.step function
x.train[,11] <- as.integer(x.train[,10])-1

opt_ntree <- gbm.step(data=x.train, gbm.x = 1:9, gbm.y = 11,
                              family = "bernoulli", tree.complexity = 2,
                              learning.rate = 0.05, bag.fraction = 0.5)

# As we can from the output, the optimal number of trees should be 1400.

# After doing parameter tuning, the final values used for the model were shrinkage = .05,
interaction.depth = 3, and n.trees = 1400
set.seed(1)

gb_adult4 = gbm(income ~.,
              data = x.train,
              distribution = "multinomial",
              cv.folds = 10,
              shrinkage = .01,
              interaction.depth = 3,
              n.minobsinnode = 10,
              n.trees = 1400)
pred4 = predict.gbm(object = gb_adult4,
                    newdata = x.test,
                    n.trees = 1400,
                    type = "response")


gb_income4 = colnames(pred4)[apply(pred4, 1, which.max)]
confusionMatrix(y.test, as.factor(gb_income4))

# So the final gradient boosting model has the accuracy rate of 0.8249.

# Create the roc plot


gb_auc
```

```
## [1] 0.8895496

abline(a=0, b=1)
```

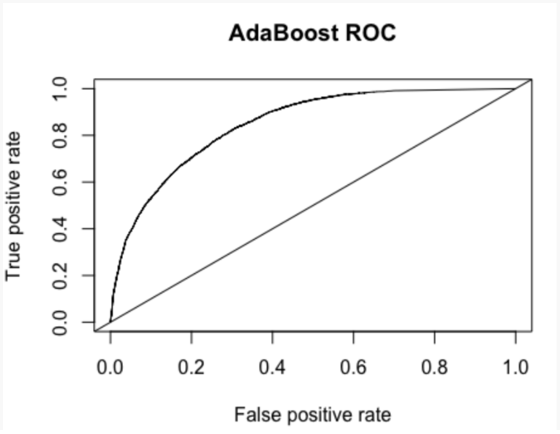


```
# The auc value is 0.8895
```

## 8.Adaptive Boosting

```
# Initially, we choose to use 15 classifiers for adaptive boosting
test_adaboost <- adaboost(income~. , x.train, 15)

# The accurate rate for training set is 0.9596, and for test set is 0.8171.

### (Need long time to run)
# To improve the model performance, we can try different number of iterations and check
performance.
set.seed(1)

n_list = c(10,30,50,70)

for (n_iter in n_list)
{
  test_adaboost2 <- adaboost(income~. , x.train, n_iter)
  bpred_test2 <- predict(test_adaboost2, x.test)
  print(paste(n_iter, ":",bpred_test2$error))
}

## [1] "10 : 0.181929856888398"
## [1] "30 : 0.181499907868067"
## [1] "50 : 0.180947116270499"
## [1] "70 : 0.180148639518457"

# As we can see from the output, when iteration number is 30, it will have the lowest error rate
compared to other iteration numbers, which is 0.1903. So we will choose 30 as the iteration number
in our final adaboost model.

set.seed(1)

test_adaboost3 <- adaboost(income~. , x.train, 30)
bpred_test3 <- predict(test_adaboost, x.test)
confusionMatrix(bpred_test3$class, y.test)

#When iteration number is 30, the accuracy rate is 0.8171 for the test set and 0.9596 for the
training set.

# AdaBoost adapts based on errors in predicted class labels whereas Real AdaBoost uses the
predicted class probabilities. Next, we will try Real AdaBoost to train our data set.
set.seed(1)

test_real_adaboost <- real_adaboost(income~. , x.train, 30)
real_bpred_test <- predict(test_real_adaboost, x.test)
confusionMatrix(real_bpred_test$class, y.test)
```

```
# The accurate rate for the training set is 0.9576, and for the test set is 0.8109.

# Create the roc plot

ab_auc

## [1] 0.8482151

abline(a=0,b=1)
```


AdaBoost ROC
True positive rate
False positive rate