Setup:

In this problem set you will need, at minimum, the following R packages.

```
# Load standard libraries
library(tidyverse)
library(gridExtra)
library(MASS)
library(pROC)
library(arm)
library(randomForest)
library(xgboost)
```

Data: In this problem set we will use the `titanic` dataset used previously in class. The Titanic text file contains data about the survival of passengers aboard the Titanic. Table 1 contains a description of this data.

```
# Load data
titanic_data <- read.csv('titanic.csv')
# explore data structure
str(titanic_data)
```

```
## 'data.frame':    1309 obs. of  14 variables:
##  $ pclass   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ survived : int  1 1 0 0 0 1 1 0 1 0 ...
##  $ name     : Factor w/ 1307 levels "Abbing, Mr. Anthony",..: 22 24 25 26 27 31 46 47 51 55 ...
```

```
##  $ sex      : Factor w/ 2 levels "female","male": 1 2 1 2 1 2 1 2 1 2 ...
##  $ age      : num  29 0.917 2 30 25 ...
##  $ sibsp    : int  0 1 1 1 1 0 1 0 2 0 ...
##  $ parch    : int  0 2 2 2 2 0 0 0 0 0 ...
##  $ ticket   : Factor w/ 929 levels "110152","110413",..: 188 50 50 50 50 125 93 16 77 826 ...
##  $ fare     : num  211 152 152 152 152 ...
##  $ cabin    : Factor w/ 187 levels "","A10","A11",..: 45 81 81 81 81 151 147 17 63 1 ...
##  $ embarked : Factor w/ 4 levels "","C","Q","S": 4 4 4 4 4 4 4 4 4 2 ...
##  $ boat     : Factor w/ 28 levels "","1","10","11",..: 13 4 1 1 1 14 3 1 28 1 ...
##  $ body     : int  NA NA NA 135 NA NA NA NA NA 22 ...
##  $ home.dest: Factor w/ 370 levels "","?Havana, Cuba",..: 310 232 232 232 232 238 163 25 23 230 ...
```

Variable	**Description**
pclass	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
survived	Survival (0 = No; 1 = Yes)
name	Name
sex	Sex
age	Age
sibsp	Number of Siblings/Spouses Aboard
parch	Number of Parents/Children Aboard
ticket	Ticket Number
fare	Passenger Fare
cabin	Cabin
embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
boat	Lifeboat
body	Body Identification Number
home.dest	Home/Destination

Table 1: Description of variables in the Titanic Dataset

1. As part of this assignment we will evaluate the performance of a few different statistical learning methods. We will fit a particular statistical learning method on a set of *training* observations and measure its performance on a set of *test* observations.

   (a) Discuss the advantages of using a training/test split when evaluating statistical models. In order for the model to more accurately predict results we need to make sure the model is well trained in predicting outcomes. There are chances where the model could over fit and predict far from accurate results. In order to narrow the size of errors we split the data set into training and test so that the training data can be used to improve the predictions of the model and the test data could be used to compare the prediction results with the test results.

   (b) Split your data into a *training* and *test* set based on an 80-20 split, in other words, 80% of the observations will be in the training set.

```
#Replace 0 and 1 values in survived variable to make the data better represented
titanic_data$survived[titanic_data$survived == 0] <- "Died"
titanic_data$survived[titanic_data$survived == 1] <- "Survived"

#Convert pclass and survived variable into a factor variable
titanic_data$pclass <- as.factor(titanic_data$pclass)
titanic_data$survived <- as.factor(titanic_data$survived)
titanic_data$fare  <- as.factor(titanic_data$fare)

#Set seed as 1 so that values are reproducible
set.seed(1)

#Obtain the training index which is 80% of the dataset index
trainindex <- sample(1309,1309*0.8)

#Obtain the trainin data set using the training index
trainset <- titanic_data[trainindex, ]

#Obtain the test data set by exclusing the training data set from the main dataset
testset <- titanic_data[-trainindex, ]
```

2. In this problem set our goal is to predict the survival of passengers. First consider training a logistic regression model for survival that controls for the socioeconomic status of the passenger.

   (a) Fit the model described above using the `glm` function in R.

```
#Attach the titanic datase
attach(titanic_data)

#Fit bayesglm model for pclass and survival
pclassModel <- glm(survived ~ pclass, family = binomial, data = trainset )

summary(pclassModel)
```

```
##
## Call:
## glm(formula = survived ~ pclass, family = binomial, data = trainset)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.363  -0.771  -0.771   1.003   1.648
##
## Coefficients:
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.4253     0.1286   3.307 0.000943 ***
## pclass2      -0.6876     0.1850  -3.717 0.000202 ***
## pclass3      -1.4864     0.1607  -9.251  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1390.7  on 1046  degrees of freedom
## Residual deviance: 1297.4  on 1044  degrees of freedom
## AIC: 1303.4
##
## Number of Fisher Scoring iterations: 4
```

Note: I suggested `bayesglm` as well in case the model was unstable (you can see this with extremely large s.e. estimates for the coefficients). Be sure you included `pclass` as a `factor` because it is a categorical variable!

(b) What might you conclude based on this model about the probability of survival for lower class passengers?

The slopes for the pclass model are not close to zero. The p value is significantly higher than 0.05. THe Std Error does not steer the slope to zero. These are indicators that the pclass is a significant predictor of survival

If we look at the p values we can see the pclass 2 reduces a bit. But the pclass3 reduces significantly. This is an indicator that the survival of 3rd class passengers is very low.

```
#Create a dataset to hold the predicted values
pclass <- c(1,2,3)

#Add the dataset to a dataframe
class_df <- data_frame(pclass)

#Convert the classData
class_df$pclass <- as.factor(class_df$pclass)

#Predict the results based on the each observation in the dataset
predict_testSet <- predict(pclassModel, class_df, type = "response")

predict_testSet
```

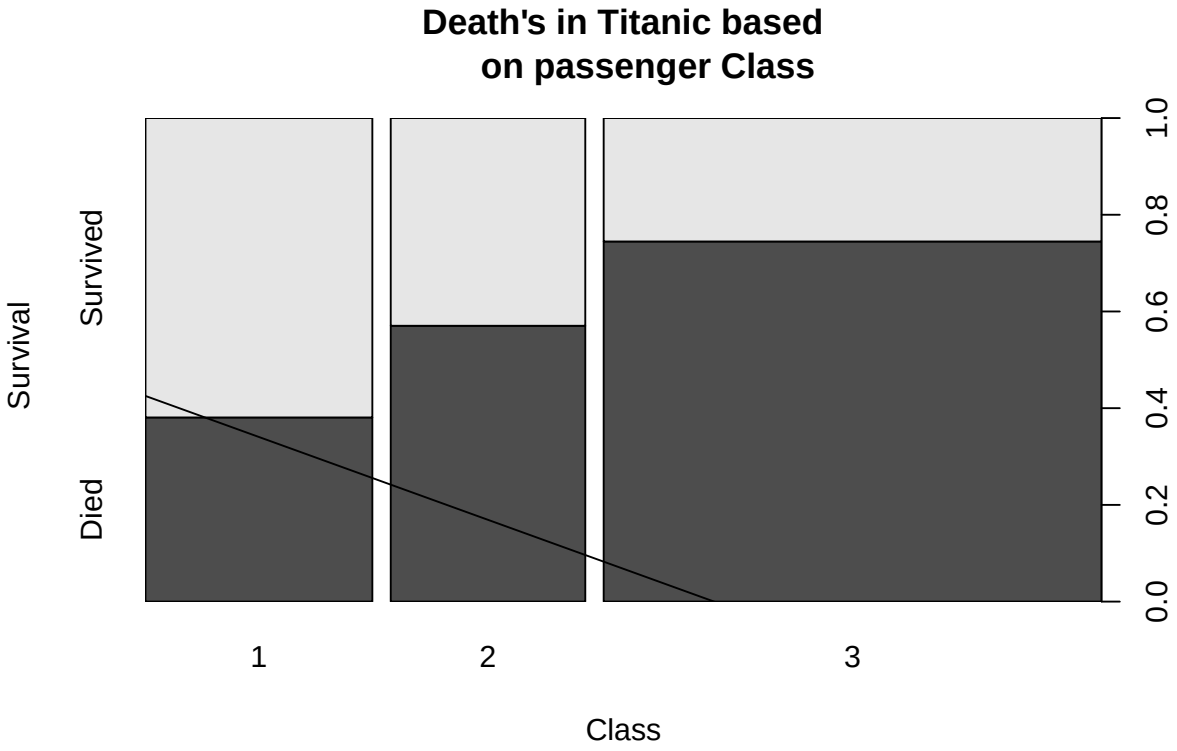
```
##         1         2         3
## 0.6047431 0.4347826 0.2570922
```

According to the model the survival of passengers for 2nd Class is 43.5% 3rd Class is 25.7%

Let us plot the actual data to verify our prediction results.

```
plot(titanic_data$survived ~ titanic_data$pclass, main = "Death's in Titanic based
     on passenger Class", xlab="Class" , ylab=" Survival")
abline(pclassModel)
```

```
## Warning in abline(pclassModel): only using the first two of 3 regression
## coefficients
```



Throught the plot we can observe that the deaths are highest for 3rd class, then for 2nd class and then for 1st class. The abline proves the prediction matches the actual observed data.

3. Next, let's consider the performance of this model.

   (a) Predict the survival of passengers for each observation in your test set using the model fit in Problem 2. Save these predictions as `yhat`.

   Predicting the survival of passengers for each observation in test data set using the model we created above.

```
# yhat will store the predicted value for our test data based on our logistic model
yhat <- predict(pclassModel, testset, type = "response")
```

   (b) Use a threshold of 0.5 to classify predictions. What is the number of false positives on the test data? Interpret this in your own words.

   Using the threshold of 0.5 means we will consider all passengers with predicted values of survival of 0.5 or above as survived and those below 0.5 as dead.

```
#Lets create a vector to store value of survival and set initial value as dead
predictionTest = rep("Died", 262)

#Set passerers with predicted values of 0.5 as survived
predictionTest[yhat > .5]="Survived"
```

   Lets test if our model worked

```
# Creating confusion matrix to check how many test observations were correctly classified
table(predictionTest, testset$survived)
```

```
##
```

```
## predictionTest Died Survived
##       Died      137       55
##       Survived   23       47
```

Let us check the overall accuracy of our model

```
# Finding the percentage of correctly classified observations
mean(predictionTest == testset$survived) * 100
```

```
## [1] 70.22901
```

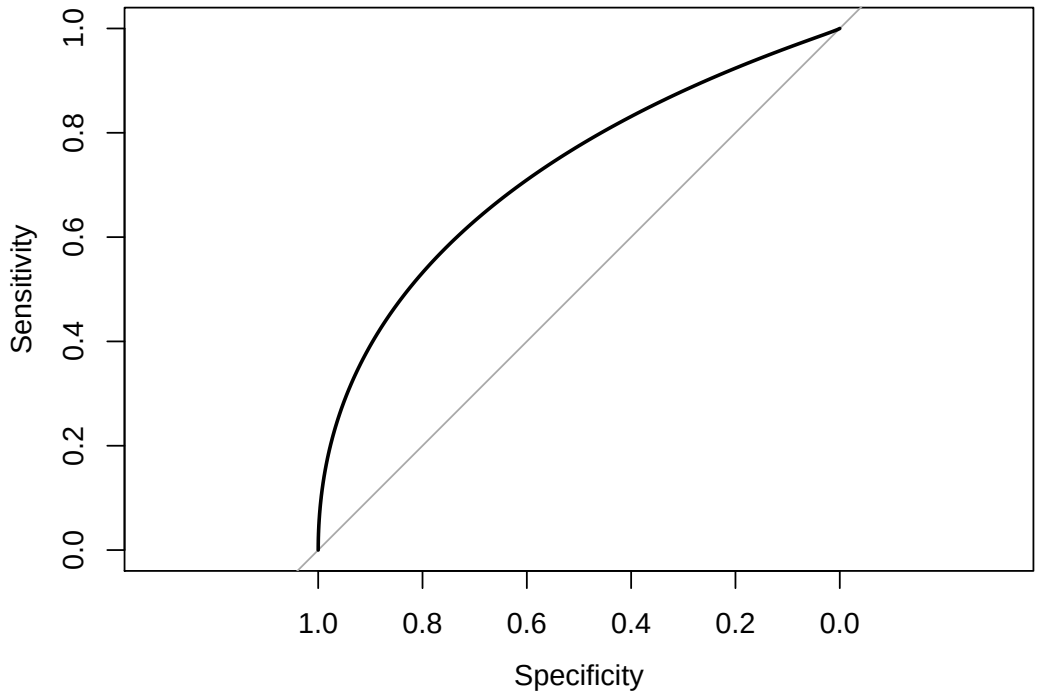
The accuracy of the model is 70.22%.

From the confusion matrix we can see that the number of false positives is 23 which means the model predicts 23 people as surviours when they were actually dead. THe total number of dead people can be found by adding the first row values.

(c) Using the `roc` function, plot the ROC curve for this model. Discuss what you find.

Plotting the ROC curve for our logistic model to show accuracy.

```
#Modify variable to be ordered to work in roc curve
predictionTest <- as.ordered(predictionTest)
testset$survived <- as.ordered(testset$survived)

#Plot ROC curve to match predicted values of survival with actual survival
roc_logistic <- roc(testset$survived, yhat,smooth = TRUE)

plot(roc_logistic)
```




```
##
## Call:
## roc.default(response = testset$survived, predictor = yhat, smooth = TRUE)
```

```
##
## Data: yhat in 160 controls (testset$survived Died) < 102 cases (testset$survived Survived).
## Smoothing: binormal
## Area under the curve: 0.7222
```

From observing the curve we know that there is a significant area between the curve and the 45 degree line. Accouring to the ROC graph, bigger the area between the curve and the 45 degree line more accurate is the model. The farther the distance of the curve from the 45 degree line the better is the accuracy of the model.

4. Suppose we use the data to construct a new predictor variable based on a passenger's listed title (i.e. Mr., Mrs., Miss., Master).

   (a) Why might this be an interesting variable to help predict passenger survival?

   The Variable MR. Mrs. Miss and Master indicates whether a passenger is Male, Female, Adult or Child. Knonwing these values we can predict or know if the survival rates for male vs female and adult vs child.

   (b) Write a function to add this predictor to your dataset.

```
#Function to return text Mr, Miss, Mrs or Master if they are part of the name

generateTitle <- function(name){
  for(title in c("Master", "Miss", "Mrs.", "Mr.")){
    if(grepl(title,name)){
      return( title)
    }
  }
  return("nothing")
}

#Populate  title in the main datatset
for(i in 1:nrow(titanic_data)){
  titanic_data$title[i] = generateTitle(titanic_data$name[i])
}

#Summarise the data
titanic_data %>%
  group_by(title) %>%
  summarise(count=n())
```

```
## # A tibble: 5 × 2
##     title count
##     <chr> <int>
## 1  Master    61
## 2    Miss   260
## 3     Mr.   758
## 4    Mrs.   199
## 5 nothing    31
```

   (c) Fit a second logistic regression model including this new feature. Use the `summary` function to look at the model. Did this new feature improve the model?

   Fit a second logistic regression model including new feature

```
trainset_title <- titanic_data[trainindex,]
testset_title <- titanic_data[-trainindex,]
```

```r
#Update the data type as factor for all variables being used

trainset_title$pclass = as.factor(trainset_title$pclass)
trainset_title$survived = as.factor(trainset_title$survived)
trainset_title$title = as.factor(trainset_title$title)

testset_title$pclass = as.factor(testset_title$pclass)
testset_title$survived = as.factor(testset_title$survived)
testset_title$title = as.factor(testset_title$title)

#Fit glm regression model

secondModel <- bayesglm(survived ~ pclass + title, family = binomial, data = trainset_title)

summary(secondModel)
```

```
##
## Call:
## bayesglm(formula = survived ~ pclass + title, family = binomial,
##     data = trainset_title)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1751  -0.6452  -0.4198   0.6721   2.2239
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    1.6179     0.3395   4.765 1.89e-06 ***
## pclass2       -0.8941     0.2240  -3.992 6.54e-05 ***
## pclass3       -1.8149     0.2027  -8.954  < 2e-16 ***
## titleMiss      0.4378     0.3245   1.349   0.1772
## titleMr.      -2.1876     0.3179  -6.881 5.94e-12 ***
## titleMrs.      0.6491     0.3519   1.845   0.0651 .
## titlenothing  -1.7519     0.5552  -3.156   0.0016 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1390.69  on 1046  degrees of freedom
## Residual deviance:  978.85  on 1040  degrees of freedom
## AIC: 992.85
##
## Number of Fisher Scoring iterations: 6
```

We will compare both models using AIC. AIC is used to determine which model is more accurate (Akaike Information Criteria). THis compares the relative quality of models for a given set of data. Lesser the AIC, better is the model.

```r
#AIC for first model with only pclass as predictor
pclassModel$aic
```

```
## [1] 1303.423
```

```
#AIC for second model with pclass and title as predictor
secondModel$aic
```

```
## [1] 992.8529
```

We oberve that the second model has a lesser AIC value compared to the first model. Hence we can conclude that the second model is more accurate than the first model.

(d) Comment on the overall fit of this model. For example, you might consider exploring when misclassification occurs.

We will fit the model on the entire titanic dataset.

```
model.fit <- predict(secondModel, titanic_data, type="response" )

# Creating vector to store predicted value with treshold of 0.5
predictionTestFit = rep("Died" ,1309)
predictionTestFit[model.fit >.5]="Survived"

# Modifying the variables to be used in the roc
predictionTestFit <- as.ordered(predictionTestFit)
titanic_data$survived <- as.ordered(titanic_data$survived)

# Creating confusion matrix to check how many test observations were correct
table(predictionTestFit, titanic_data$survived)
```

```
##
## predictionTestFit Died Survived
##          Died      683      151
##          Survived  126      349
```

```
# Finding the percentage of correctly classified observations
(mean(predictionTestFit == titanic_data$survived))*100
```

```
## [1] 78.83881
```

```
# Finding the percentage of misclassified observations
(1- mean(predictionTestFit == titanic_data$survived))*100
```

```
## [1] 21.16119
```

This indicates the model correcly predicts the outcome on the dataset and only 21% of it is misclassified.

(e) Predict the survival of passengers for each observation in your test data using the new model. Save these predictions as `yhat2`.

Predicting the survival of passengers for each test observation and saving these predictions as yhat2

```
yhat2 <- predict(secondModel , testset_title, type ="response")

predictionTest_title = rep("Died" ,262)
predictionTest_title[yhat2 >.5]="Survived"

predictionTest_title <- as.ordered(predictionTest_title)

# Creating a confusion matrix to check how many test observations were correctly
# classified
table(predictionTest_title, testset$survived)
```

```
##
## predictionTest_title Died Survived
##              Died      138       32
##              Survived   22       70
```

```
# Finding the percentage of correctly classified observations
mean(predictionTest_title == testset$survived)*100
```

```
## [1] 79.38931
```

After predicting for the test dataset nearly 80% of the predictions are accurate.

5. Another very popular classifier used in data science is called a *random forest*[1].

   (a) Use the `randomForest` function to fit a random forest model with passenger class and title as predictors. Make predictions for the test set using the random forest model. Save these predictions as `yhat3`.

   Fitting random forest model

```
randForestTitanicModel <- randomForest(survived ~ pclass + title, mtry = 1, data = trainset_ti
```

```
randForestTitanicModel
```

```
##
## Call:
##  randomForest(formula = survived ~ pclass + title, data = trainset_title,      mtry = 1, im
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 20.53%
## Confusion matrix:
##          Died Survived class.error
## Died      631       18  0.02773498
## Survived  197      201  0.49497487
```

```
#Predict survival using the random Forest

yhat3 <- predict(randForestTitanicModel, testset_title)


# Finding percentage of correctly classified observation
mean(yhat3 == testset_title$survived)*100
```

```
## [1] 78.24427
```

We can see that the OOB estimate of error rate is 20.63% and the % correctly predicted results is 78.244%

   (b) Develop your own random forest model, attempting to improve the model performance. Make predictions for the test set using your new random forest model. Save these predictions as `yhat4`.

   We will make use of other variables like Sex and embark to see if they make a difference to the model

[1] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

```r
#Set embarked and sex as factors in the training and testing data sets

trainset_title$sex <- as.factor(trainset_title$sex)
trainset_title$embarked <- as.factor(trainset_title$embarked)

testset_title$sex <- as.factor(testset_title$sex)
testset_title$embarked <- as.factor(testset_title$embarked)

#Fittin randomForest model with predictors as pclass, title, embark and sex

randForestTitanicModel2 <- randomForest(survived ~ pclass + title + embarked + sex, mtry = 3, c

#Display the model
randForestTitanicModel2
```

```
##
## Call:
##  randomForest(formula = survived ~ pclass + title + embarked +      sex, data = trainset_tit
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.43%
## Confusion matrix:
##          Died Survived class.error
## Died      607       42  0.06471495
## Survived  151      247  0.37939698
```

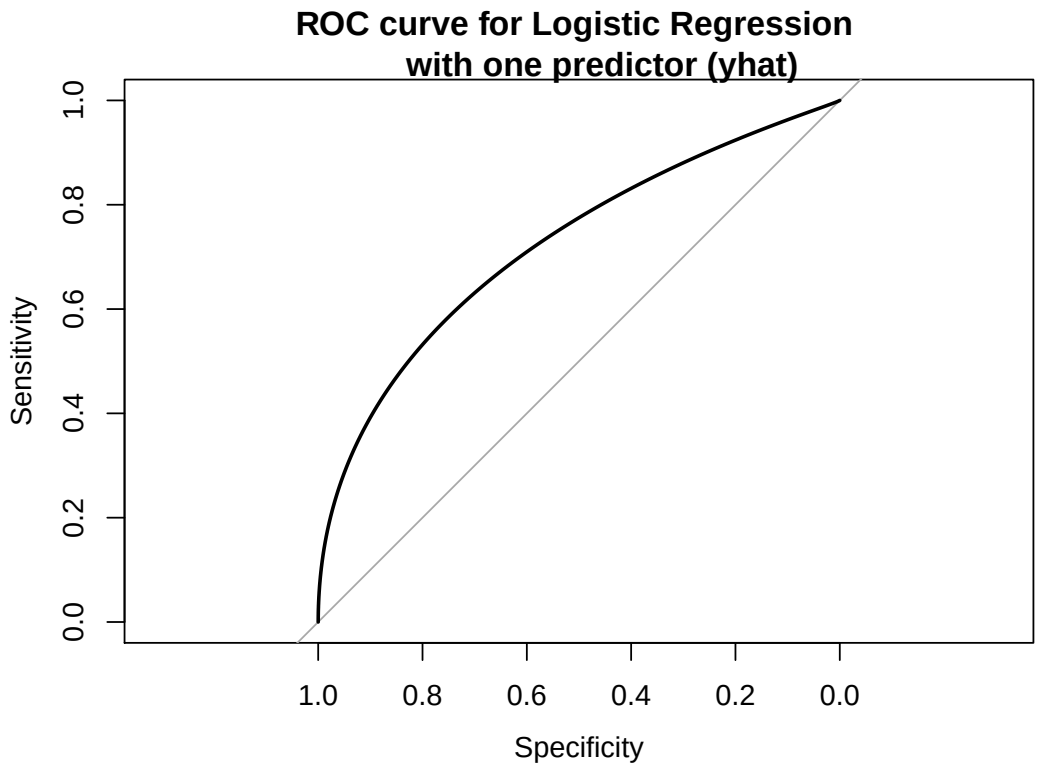
```r
#Predicting survival using random forest model
yhat4 <- predict(randForestTitanicModel2, testset_title)

# Finding percentage of correctly classified observation
mean(yhat4 == testset_title$survived)
```

```
## [1] 0.7938931
```

The Out of Bag (OOB) error estimation is 18.43% and the precentage of correctly classified observation is 79.38%. This is more than the accuracy of the previous model. Hence it suggests that knowing the sex of the passesnger and where the passeger embarked from helps in better predicting if the passenger will survive.

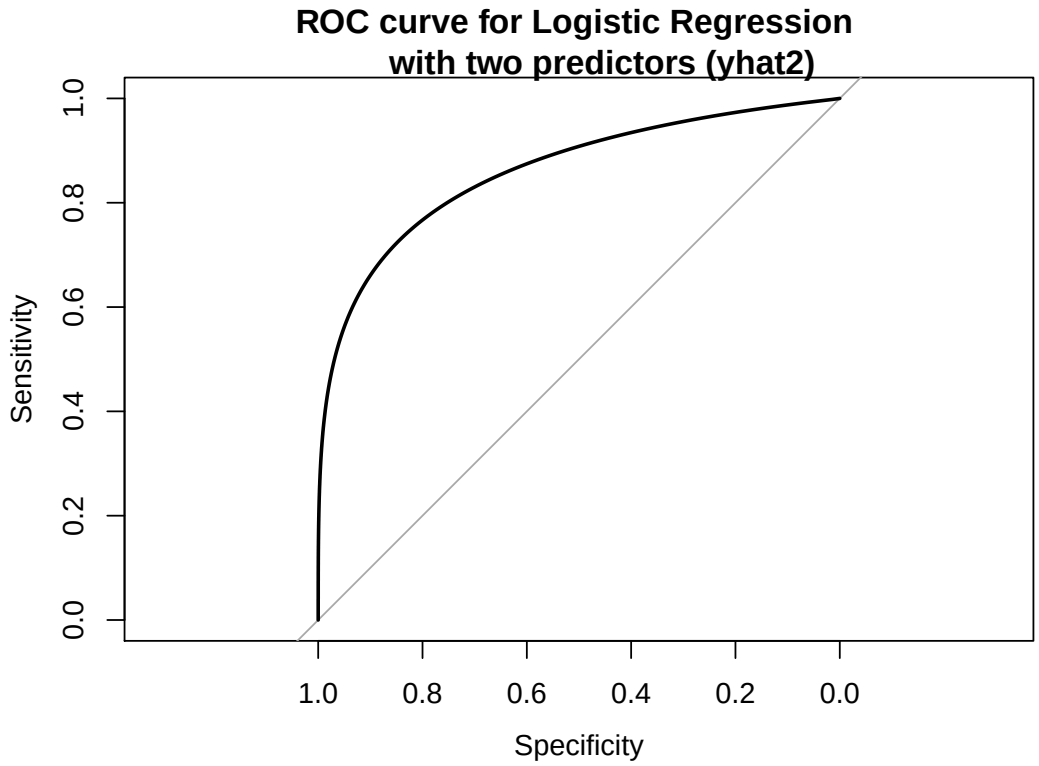
(c) Compare the accuracy of each of the models from this problem set using ROC curves. Comment on which statistical learning method works best for predicting survival of the Titanic passengers.

```r
# Plotting the logistic regression roc curve for single predictor - pclass
plot(roc_logistic, main="ROC curve for Logistic Regression
     with one predictor (yhat)")
```



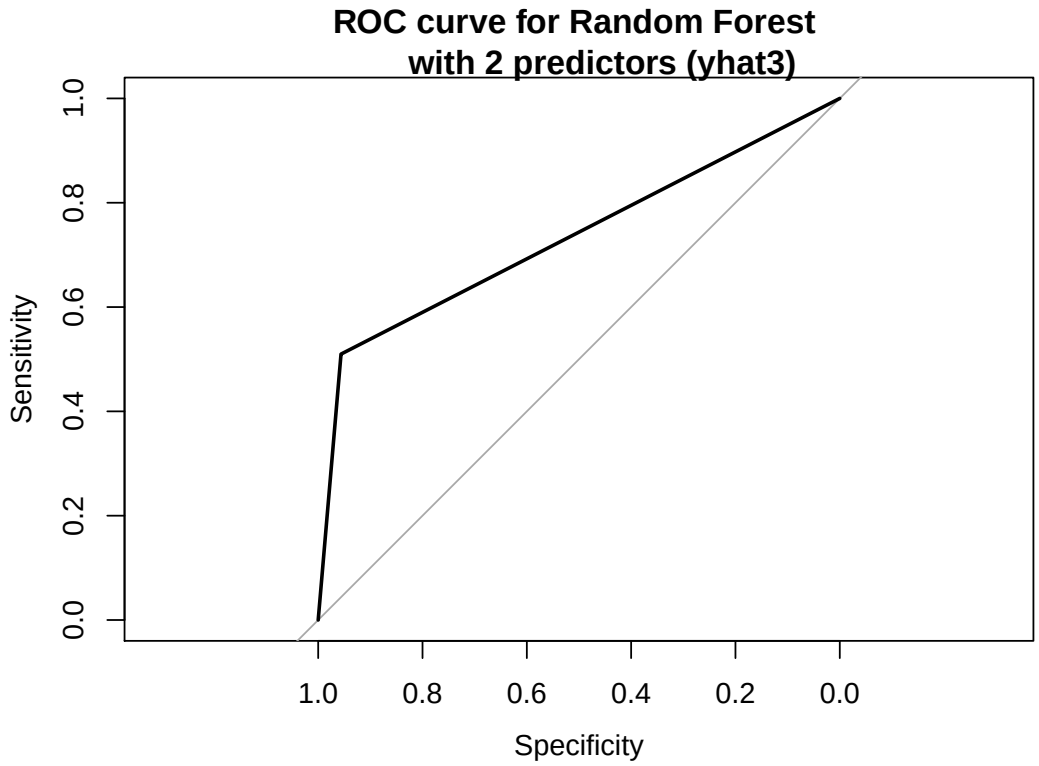

```
##
## Call:
## roc.default(response = testset$survived, predictor = yhat, smooth = TRUE)
##
## Data: yhat in 160 controls (testset$survived Died) < 102 cases (testset$survived Survived).
## Smoothing: binormal
## Area under the curve: 0.7222
```

```
# Plotting the logistic regression roc curve for multiple predictor - pclass and title
roc.logistic.title <- roc(testset_title$survived, yhat2, smooth = TRUE)
plot(roc.logistic.title, main="ROC curve for Logistic Regression
     with two predictors (yhat2)")
```



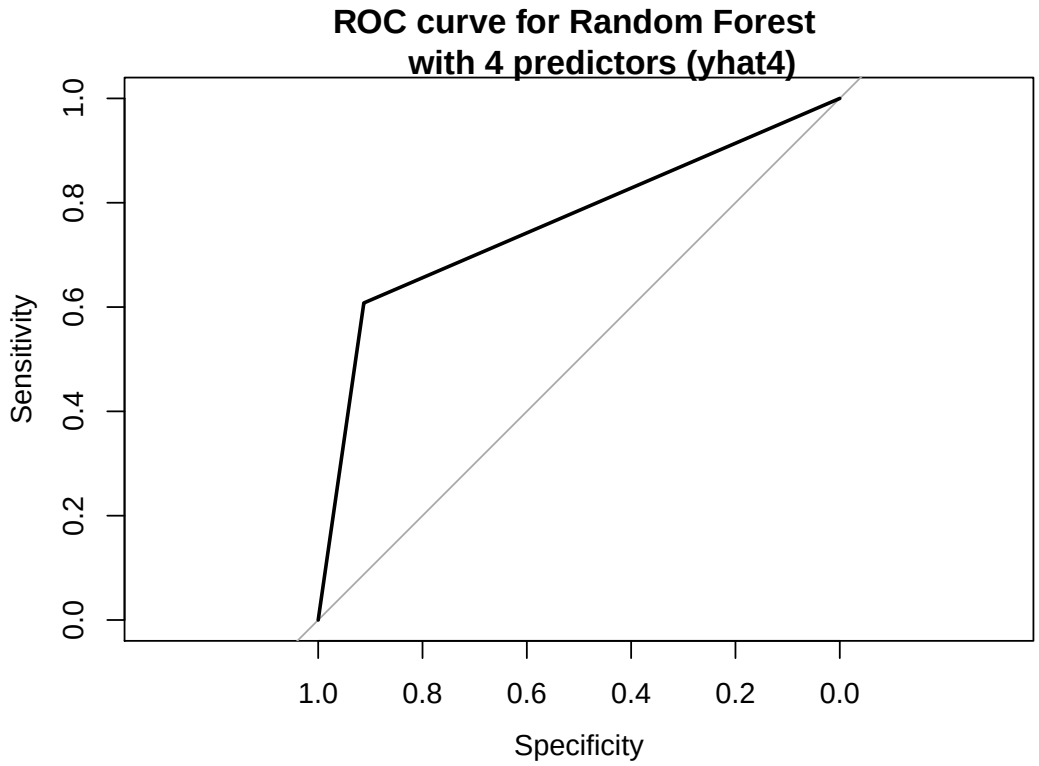

```
##
## Call:
## roc.default(response = testset_title$survived, predictor = yhat2,     smooth = TRUE)
##
## Data: yhat2 in 160 controls (testset_title$survived Died) < 102 cases (testset_title$survive
## Smoothing: binormal
## Area under the curve: 0.8604
```

```
# Plotting the random forest regression roc curve for multiple predictors -
# pclass and title
rocRandomForest <- roc(testset_title$survived, as.ordered(yhat3))
plot(rocRandomForest, main="ROC curve for Random Forest
     with 2 predictors (yhat3)")
```




```
##
## Call:
## roc.default(response = testset_title$survived, predictor = as.ordered(yhat3))
##
## Data: as.ordered(yhat3) in 160 controls (testset_title$survived Died) < 102 cases (testset_t
## Area under the curve: 0.733
```

```
# Plotting the random forest regression roc curve for multiple predictor -
# pclass, title, embark and sex
rocRandomForest2 <- roc(testset_title$survived, as.ordered(yhat4))
plot(rocRandomForest2, main="ROC curve for Random Forest
     with 4 predictors (yhat4)")
```




```
##
## Call:
## roc.default(response = testset_title$survived, predictor = as.ordered(yhat4))
##
## Data: as.ordered(yhat4) in 160 controls (testset_title$survived Died) < 102 cases (testset_t
## Area under the curve: 0.7602
```

- Finally, we will explore a gradient boosted tree model, using the `xgboost` package written by your fellow UW student Tianqi Chen. `xgboost` stands for "Extreme Gradient Boosting", which is state-of-the-art software known for its fast training time and predictive accuracy.

  (a) The XGB algorithm can only handle numeric data, so first we need to convert all categorical variables to a different representation, such as a sparse matrix.

```
library(Matrix)

titanic_data <- read.csv('titanic.csv')

#Function to return text Mr, Miss, Mrs or Master if they are part of the name

generateTitle <- function(name){
  for(title in c("Master", "Miss", "Mrs.", "Mr.")){
    if(grepl(title,name)){
      return( title)
    }
  }
  return("nothing")
}

#Populate  title in the main datatset
```

```
for(i in 1:nrow(titanic_data)){
  titanic_data$title[i] = generateTitle(titanic_data$name[i])
}

xgbTrainSet <- titanic_data[trainindex, ]
xgbTestSet <- titanic_data[-trainindex, ]

# xgbTrainSet$survived <- as.factor(xgbTrainSet$survived)
xgbTrainSet$sex <- as.factor(xgbTrainSet$sex)
xgbTrainSet$pclass <- as.factor(xgbTrainSet$pclass)
xgbTrainSet$title <- as.factor(xgbTrainSet$title)

# xgbTestSet$survived <- as.factor(xgbTestSet$survived)
xgbTestSet$sex <- as.factor(xgbTestSet$sex)
xgbTestSet$pclass <- as.factor(xgbTestSet$pclass)
xgbTestSet$title <- as.factor(xgbTestSet$title)


sparse.matrix.train <- sparse.model.matrix(survived ~ pclass + sex + title -1, data = train

sparse.matrix.test <-  sparse.model.matrix(survived ~ pclass + sex + title -1, data = tests



output_vector = xgbTrainSet$survived #output vector to be predicted
```

(b) The following code fits a boosted tree model and produces a plot. Run the code and provide an explanation of the resulting plot.

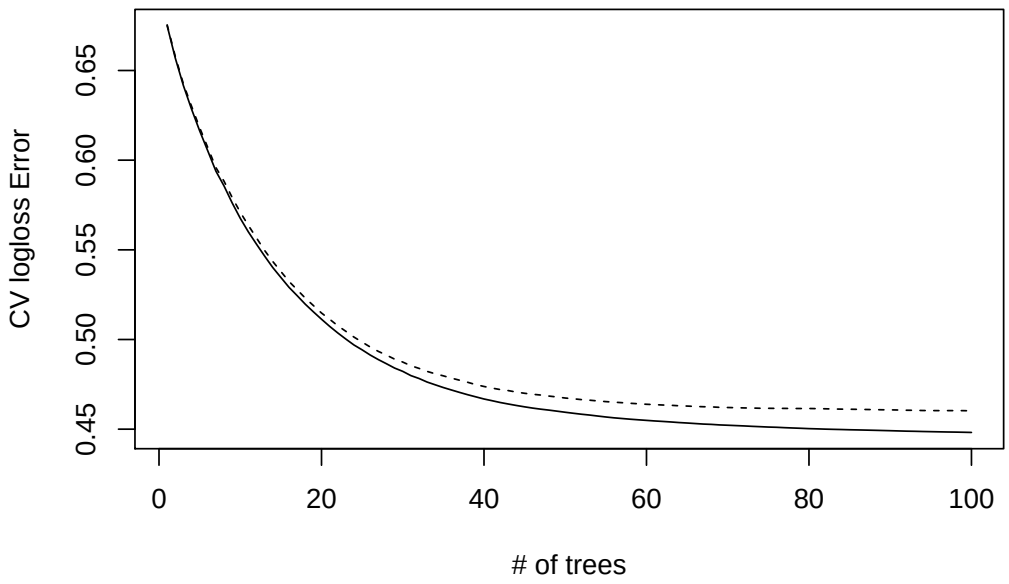
```
xgb.model.one <- xgb.cv(data= sparse.matrix.train,     #train sparse matrix
                        label= output_vector,    #output vector to be predicted
                        eval.metric = 'logloss',     #model minimizes Root Mean Squared Err
                        objective = "reg:logistic", #regression
                        nfold = 10,
                        #tuning parameters
                        max.depth = 3,             #Vary btwn 3-15
                        eta = 0.05,                 #Vary btwn 0.1-0.3
                        nthread = 5,              #Increase this to improve speed
                        subsample= 1,             #Vary btwn 0.8-1
                        colsample_bytree = 0.5,    #Vary btwn 0.3-0.8
                        lambda = 0.5,              #Vary between 0-3
                        alpha = 0.5,               #Vary between 0-3
                        min_child_weight = 3,      #Vary btwn 1-10
                        nround = 100              #Vary btwn 100-3000 based on max.depth, et
                        )
```

```
## [0]  train-logloss:0.674987+0.001244 test-logloss:0.675451+0.003417
## [1]  train-logloss:0.656972+0.001643 test-logloss:0.657881+0.005688
## [2]  train-logloss:0.641609+0.001582 test-logloss:0.642540+0.007902
## [3]  train-logloss:0.628530+0.002638 test-logloss:0.630149+0.010810
## [4]  train-logloss:0.616373+0.003454 test-logloss:0.618153+0.011905
## [5]  train-logloss:0.605193+0.003198 test-logloss:0.607039+0.014335
## [6]  train-logloss:0.593898+0.003352 test-logloss:0.595892+0.015716
## [7]  train-logloss:0.585657+0.005863 test-logloss:0.588271+0.015595
```

```
## [8]  train-logloss:0.576217+0.005758 test-logloss:0.579240+0.017119
## [9]  train-logloss:0.567595+0.005888 test-logloss:0.570970+0.018594
## [10] train-logloss:0.559989+0.006473 test-logloss:0.563605+0.019535
## [11] train-logloss:0.553141+0.006199 test-logloss:0.556612+0.020888
## [12] train-logloss:0.546521+0.006193 test-logloss:0.549669+0.022421
## [13] train-logloss:0.540143+0.006082 test-logloss:0.543509+0.024254
## [14] train-logloss:0.534600+0.006398 test-logloss:0.537913+0.025959
## [15] train-logloss:0.529094+0.006205 test-logloss:0.532371+0.027039
## [16] train-logloss:0.524463+0.005869 test-logloss:0.527776+0.028713
## [17] train-logloss:0.519697+0.005629 test-logloss:0.523097+0.030154
## [18] train-logloss:0.515438+0.005289 test-logloss:0.518939+0.031986
## [19] train-logloss:0.511308+0.005389 test-logloss:0.514755+0.033077
## [20] train-logloss:0.507339+0.004998 test-logloss:0.511311+0.034821
## [21] train-logloss:0.503672+0.004856 test-logloss:0.507693+0.035593
## [22] train-logloss:0.500232+0.005198 test-logloss:0.504556+0.036257
## [23] train-logloss:0.496930+0.004808 test-logloss:0.501339+0.037093
## [24] train-logloss:0.494235+0.004754 test-logloss:0.498540+0.037558
## [25] train-logloss:0.491328+0.004685 test-logloss:0.495804+0.038650
## [26] train-logloss:0.488781+0.004915 test-logloss:0.493367+0.039507
## [27] train-logloss:0.486469+0.004883 test-logloss:0.491295+0.040170
## [28] train-logloss:0.484064+0.005016 test-logloss:0.489044+0.040829
## [29] train-logloss:0.482241+0.005218 test-logloss:0.487416+0.041465
## [30] train-logloss:0.479872+0.005344 test-logloss:0.485380+0.042426
## [31] train-logloss:0.478275+0.005309 test-logloss:0.483903+0.043527
## [32] train-logloss:0.476273+0.005503 test-logloss:0.482210+0.044232
## [33] train-logloss:0.474725+0.005508 test-logloss:0.480884+0.044784
## [34] train-logloss:0.473208+0.005709 test-logloss:0.479734+0.045175
## [35] train-logloss:0.471835+0.005780 test-logloss:0.478438+0.045588
## [36] train-logloss:0.470490+0.005794 test-logloss:0.477303+0.046135
## [37] train-logloss:0.469201+0.005762 test-logloss:0.476130+0.046819
## [38] train-logloss:0.468019+0.005702 test-logloss:0.474820+0.047506
## [39] train-logloss:0.466828+0.005723 test-logloss:0.473810+0.048102
## [40] train-logloss:0.465865+0.005732 test-logloss:0.472894+0.048696
## [41] train-logloss:0.464871+0.005951 test-logloss:0.472080+0.049192
## [42] train-logloss:0.464026+0.006062 test-logloss:0.471500+0.049649
## [43] train-logloss:0.463204+0.006069 test-logloss:0.470665+0.050113
## [44] train-logloss:0.462450+0.006111 test-logloss:0.470007+0.050467
## [45] train-logloss:0.461740+0.006089 test-logloss:0.469454+0.050755
## [46] train-logloss:0.461139+0.006124 test-logloss:0.469024+0.050877
## [47] train-logloss:0.460617+0.006121 test-logloss:0.468444+0.051175
## [48] train-logloss:0.460037+0.006038 test-logloss:0.467943+0.051635
## [49] train-logloss:0.459430+0.006012 test-logloss:0.467364+0.051966
## [50] train-logloss:0.458863+0.005941 test-logloss:0.466951+0.052429
## [51] train-logloss:0.458336+0.005967 test-logloss:0.466472+0.052607
## [52] train-logloss:0.457872+0.005937 test-logloss:0.466160+0.052849
## [53] train-logloss:0.457324+0.005969 test-logloss:0.465734+0.052974
## [54] train-logloss:0.456795+0.005984 test-logloss:0.465362+0.053377
## [55] train-logloss:0.456329+0.005983 test-logloss:0.465059+0.053885
## [56] train-logloss:0.455933+0.006028 test-logloss:0.464765+0.054370
## [57] train-logloss:0.455561+0.006010 test-logloss:0.464433+0.054823
## [58] train-logloss:0.455218+0.006036 test-logloss:0.464167+0.055120
## [59] train-logloss:0.454899+0.006024 test-logloss:0.463888+0.055500
## [60] train-logloss:0.454587+0.006043 test-logloss:0.463676+0.055883
## [61] train-logloss:0.454319+0.006046 test-logloss:0.463477+0.056142
```

```
## [62] train-logloss:0.454021+0.006062 test-logloss:0.463315+0.056241
## [63] train-logloss:0.453692+0.006071 test-logloss:0.463037+0.056512
## [64] train-logloss:0.453431+0.006013 test-logloss:0.462887+0.056797
## [65] train-logloss:0.453141+0.006027 test-logloss:0.462670+0.056958
## [66] train-logloss:0.452843+0.006040 test-logloss:0.462516+0.057147
## [67] train-logloss:0.452664+0.006059 test-logloss:0.462428+0.057363
## [68] train-logloss:0.452373+0.006024 test-logloss:0.462162+0.057805
## [69] train-logloss:0.452188+0.006002 test-logloss:0.462074+0.058107
## [70] train-logloss:0.452005+0.006004 test-logloss:0.461977+0.058135
## [71] train-logloss:0.451802+0.005988 test-logloss:0.461846+0.058295
## [72] train-logloss:0.451572+0.006074 test-logloss:0.461787+0.058423
## [73] train-logloss:0.451385+0.006041 test-logloss:0.461661+0.058641
## [74] train-logloss:0.451179+0.006022 test-logloss:0.461633+0.058756
## [75] train-logloss:0.451050+0.006004 test-logloss:0.461594+0.058856
## [76] train-logloss:0.450838+0.005995 test-logloss:0.461578+0.058997
## [77] train-logloss:0.450651+0.006043 test-logloss:0.461534+0.059072
## [78] train-logloss:0.450493+0.006037 test-logloss:0.461518+0.059351
## [79] train-logloss:0.450297+0.006094 test-logloss:0.461533+0.059515
## [80] train-logloss:0.450172+0.006120 test-logloss:0.461458+0.059576
## [81] train-logloss:0.450022+0.006118 test-logloss:0.461363+0.059726
## [82] train-logloss:0.449902+0.006134 test-logloss:0.461248+0.059708
## [83] train-logloss:0.449803+0.006136 test-logloss:0.461168+0.059897
## [84] train-logloss:0.449675+0.006101 test-logloss:0.461120+0.060113
## [85] train-logloss:0.449580+0.006110 test-logloss:0.461054+0.060203
## [86] train-logloss:0.449474+0.006118 test-logloss:0.460976+0.060220
## [87] train-logloss:0.449372+0.006134 test-logloss:0.460901+0.060373
## [88] train-logloss:0.449257+0.006155 test-logloss:0.460811+0.060407
## [89] train-logloss:0.449149+0.006160 test-logloss:0.460768+0.060464
## [90] train-logloss:0.449019+0.006193 test-logloss:0.460689+0.060534
## [91] train-logloss:0.448897+0.006203 test-logloss:0.460556+0.060608
## [92] train-logloss:0.448810+0.006218 test-logloss:0.460521+0.060689
## [93] train-logloss:0.448696+0.006230 test-logloss:0.460397+0.060761
## [94] train-logloss:0.448610+0.006234 test-logloss:0.460361+0.060793
## [95] train-logloss:0.448520+0.006253 test-logloss:0.460342+0.060869
## [96] train-logloss:0.448432+0.006242 test-logloss:0.460355+0.060971
## [97] train-logloss:0.448369+0.006233 test-logloss:0.460346+0.061065
## [98] train-logloss:0.448279+0.006266 test-logloss:0.460320+0.061233
## [99] train-logloss:0.448180+0.006261 test-logloss:0.460207+0.061375
```

```
plot(data.frame(xgb.model.one)[,1], type='l', col='black', ylab='CV logloss Error', xlab='#
lines(data.frame(xgb.model.one)[,3], type='l', lty=2, col='black')
```



The result plot shows the number of trees vs the log loss error. We can see the for the lesser number of trees the the error is around 0.65 and as the number of trees increases within the model the log loss error reduces. THis signifies that the predictive accuracy improves with the number of trees within the model.

(c)] Modify the code to fit a boosted tree model that allows for 8 levels in each tree and uses a learning rate $\eta = .1$. Produce a visualization comparing the two models and explain what you can conclude about the new model. Which model do you prefer and why?

```
sparse.matrix.train2 <- sparse.model.matrix(survived ~ pclass + sex + fare + embarked + age

sparse.matrix.test2 <-  sparse.model.matrix(survived ~ pclass + sex + fare + embarked + age



output_vector = xgbTrainSet$survived #output vector to be predicted
```

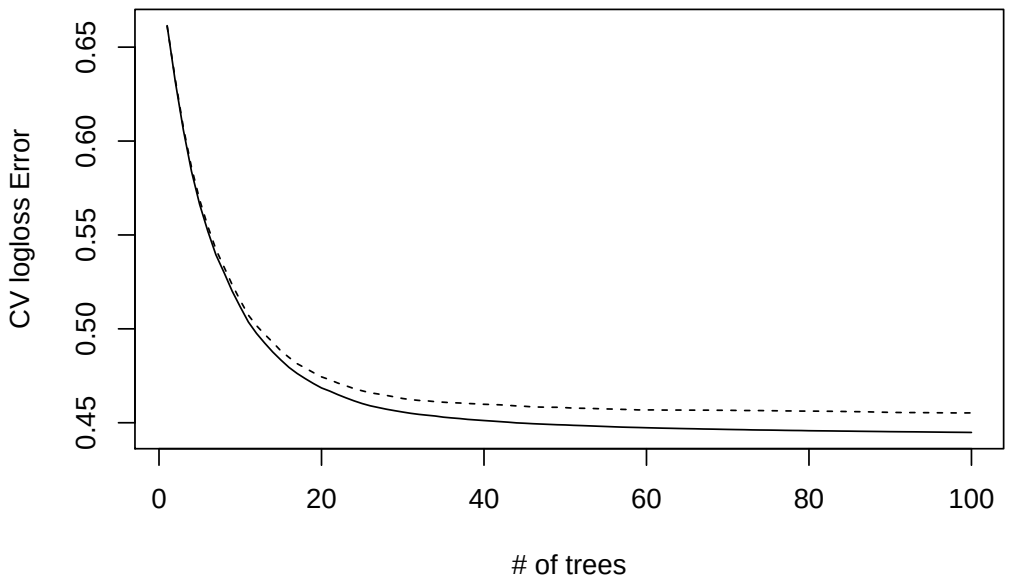
```
xgb.model.one <- xgb.cv(data= sparse.matrix.train,     #train sparse matrix
                        label= output_vector,    #output vector to be predicted
                        eval.metric = 'logloss',     #model minimizes Root Mean Squared Err
                        objective = "reg:logistic", #regression
                        nfold = 10,
                        #tuning parameters
                        max.depth = 3,            #Vary btwn 3-15
                        eta = 0.1,                #Vary btwn 0.1-0.3
                        nthread = 5,             #Increase this to improve speed
                        subsample= 1,            #Vary btwn 0.8-1
                        colsample_bytree = 0.5,   #Vary btwn 0.3-0.8
                        lambda = 0.5,             #Vary between 0-3
                        alpha = 0.5,              #Vary between 0-3
                        min_child_weight = 3,     #Vary btwn 1-10
```

```
                        nround = 100             #Vary btwn 100-3000 based on max.depth, et
                        )
```

```
## [0]  train-logloss:0.661429+0.005317 test-logloss:0.661190+0.007487
## [1]  train-logloss:0.631759+0.004532 test-logloss:0.632980+0.010492
## [2]  train-logloss:0.605827+0.003527 test-logloss:0.607169+0.011397
## [3]  train-logloss:0.583840+0.003850 test-logloss:0.586002+0.012167
## [4]  train-logloss:0.566280+0.004158 test-logloss:0.568721+0.014552
## [5]  train-logloss:0.552104+0.005385 test-logloss:0.554707+0.013925
## [6]  train-logloss:0.539462+0.004699 test-logloss:0.542364+0.016614
## [7]  train-logloss:0.530044+0.004722 test-logloss:0.532966+0.017380
## [8]  train-logloss:0.520150+0.003761 test-logloss:0.523574+0.019950
## [9]  train-logloss:0.511700+0.003794 test-logloss:0.515121+0.021117
## [10] train-logloss:0.503629+0.004415 test-logloss:0.507282+0.021471
## [11] train-logloss:0.497582+0.004941 test-logloss:0.501703+0.021569
## [12] train-logloss:0.492470+0.004614 test-logloss:0.497105+0.023179
## [13] train-logloss:0.487669+0.005458 test-logloss:0.492855+0.023599
## [14] train-logloss:0.483413+0.005184 test-logloss:0.488266+0.023834
## [15] train-logloss:0.479461+0.004849 test-logloss:0.484782+0.025698
## [16] train-logloss:0.476245+0.005057 test-logloss:0.481459+0.026408
## [17] train-logloss:0.473491+0.004995 test-logloss:0.479131+0.027471
## [18] train-logloss:0.470898+0.004872 test-logloss:0.476720+0.027998
## [19] train-logloss:0.468536+0.004842 test-logloss:0.474426+0.029059
## [20] train-logloss:0.466878+0.004734 test-logloss:0.472931+0.029451
## [21] train-logloss:0.464990+0.004636 test-logloss:0.471151+0.029955
## [22] train-logloss:0.463285+0.004635 test-logloss:0.469704+0.030852
## [23] train-logloss:0.461670+0.004469 test-logloss:0.468194+0.031390
## [24] train-logloss:0.460221+0.004494 test-logloss:0.467013+0.032481
## [25] train-logloss:0.459011+0.004398 test-logloss:0.465949+0.033416
## [26] train-logloss:0.458106+0.004365 test-logloss:0.465300+0.034247
## [27] train-logloss:0.457203+0.004611 test-logloss:0.464554+0.034394
## [28] train-logloss:0.456432+0.004753 test-logloss:0.463741+0.034613
## [29] train-logloss:0.455694+0.004708 test-logloss:0.462899+0.034847
## [30] train-logloss:0.455007+0.004812 test-logloss:0.462305+0.035146
## [31] train-logloss:0.454451+0.005037 test-logloss:0.461879+0.035442
## [32] train-logloss:0.454015+0.005033 test-logloss:0.461652+0.035861
## [33] train-logloss:0.453580+0.004985 test-logloss:0.461229+0.036274
## [34] train-logloss:0.452976+0.005043 test-logloss:0.460933+0.036962
## [35] train-logloss:0.452576+0.005020 test-logloss:0.460660+0.037506
## [36] train-logloss:0.452237+0.005083 test-logloss:0.460523+0.038264
## [37] train-logloss:0.451783+0.005161 test-logloss:0.460260+0.038828
## [38] train-logloss:0.451480+0.005114 test-logloss:0.460072+0.038882
## [39] train-logloss:0.451176+0.005126 test-logloss:0.459821+0.039240
## [40] train-logloss:0.450894+0.005109 test-logloss:0.459697+0.039286
## [41] train-logloss:0.450591+0.005110 test-logloss:0.459537+0.039378
## [42] train-logloss:0.450296+0.005139 test-logloss:0.459256+0.039403
## [43] train-logloss:0.449930+0.004987 test-logloss:0.458951+0.039528
## [44] train-logloss:0.449736+0.005078 test-logloss:0.458757+0.039603
## [45] train-logloss:0.449487+0.005030 test-logloss:0.458448+0.039891
## [46] train-logloss:0.449305+0.005014 test-logloss:0.458338+0.040073
## [47] train-logloss:0.449121+0.005039 test-logloss:0.458255+0.040331
## [48] train-logloss:0.448942+0.005121 test-logloss:0.458238+0.040546
## [49] train-logloss:0.448829+0.005113 test-logloss:0.458044+0.040697
## [50] train-logloss:0.448628+0.005112 test-logloss:0.457796+0.040794
```

```
## [51] train-logloss:0.448484+0.005130 test-logloss:0.457711+0.040849
## [52] train-logloss:0.448331+0.005106 test-logloss:0.457654+0.041011
## [53] train-logloss:0.448153+0.005123 test-logloss:0.457490+0.041032
## [54] train-logloss:0.448006+0.005132 test-logloss:0.457350+0.041072
## [55] train-logloss:0.447832+0.005118 test-logloss:0.457274+0.041322
## [56] train-logloss:0.447675+0.005099 test-logloss:0.457129+0.041368
## [57] train-logloss:0.447558+0.005058 test-logloss:0.457012+0.041440
## [58] train-logloss:0.447466+0.005092 test-logloss:0.456828+0.041425
## [59] train-logloss:0.447346+0.005092 test-logloss:0.456820+0.041595
## [60] train-logloss:0.447231+0.005089 test-logloss:0.456842+0.041762
## [61] train-logloss:0.447123+0.005050 test-logloss:0.456755+0.041991
## [62] train-logloss:0.447032+0.005028 test-logloss:0.456736+0.042114
## [63] train-logloss:0.446933+0.005023 test-logloss:0.456763+0.042350
## [64] train-logloss:0.446857+0.005029 test-logloss:0.456709+0.042442
## [65] train-logloss:0.446778+0.005074 test-logloss:0.456709+0.042656
## [66] train-logloss:0.446663+0.005170 test-logloss:0.456693+0.043005
## [67] train-logloss:0.446606+0.005180 test-logloss:0.456707+0.043139
## [68] train-logloss:0.446527+0.005180 test-logloss:0.456637+0.043225
## [69] train-logloss:0.446436+0.005175 test-logloss:0.456645+0.043609
## [70] train-logloss:0.446358+0.005176 test-logloss:0.456560+0.043640
## [71] train-logloss:0.446304+0.005176 test-logloss:0.456517+0.043746
## [72] train-logloss:0.446194+0.005185 test-logloss:0.456503+0.043834
## [73] train-logloss:0.446132+0.005198 test-logloss:0.456488+0.043760
## [74] train-logloss:0.446091+0.005193 test-logloss:0.456425+0.043856
## [75] train-logloss:0.446034+0.005187 test-logloss:0.456363+0.043951
## [76] train-logloss:0.445948+0.005181 test-logloss:0.456294+0.043987
## [77] train-logloss:0.445890+0.005177 test-logloss:0.456285+0.044076
## [78] train-logloss:0.445850+0.005176 test-logloss:0.456202+0.044147
## [79] train-logloss:0.445759+0.005188 test-logloss:0.456196+0.044153
## [80] train-logloss:0.445704+0.005184 test-logloss:0.456169+0.044220
## [81] train-logloss:0.445669+0.005179 test-logloss:0.456139+0.044248
## [82] train-logloss:0.445616+0.005172 test-logloss:0.456090+0.044303
## [83] train-logloss:0.445560+0.005143 test-logloss:0.456026+0.044353
## [84] train-logloss:0.445515+0.005125 test-logloss:0.455944+0.044414
## [85] train-logloss:0.445464+0.005112 test-logloss:0.455885+0.044440
## [86] train-logloss:0.445416+0.005105 test-logloss:0.455837+0.044515
## [87] train-logloss:0.445360+0.005089 test-logloss:0.455725+0.044594
## [88] train-logloss:0.445317+0.005083 test-logloss:0.455623+0.044680
## [89] train-logloss:0.445250+0.005087 test-logloss:0.455520+0.044878
## [90] train-logloss:0.445217+0.005102 test-logloss:0.455466+0.044879
## [91] train-logloss:0.445174+0.005090 test-logloss:0.455419+0.044995
## [92] train-logloss:0.445149+0.005104 test-logloss:0.455424+0.045025
## [93] train-logloss:0.445101+0.005099 test-logloss:0.455374+0.045098
## [94] train-logloss:0.445056+0.005110 test-logloss:0.455352+0.045236
## [95] train-logloss:0.445010+0.005104 test-logloss:0.455306+0.045382
## [96] train-logloss:0.444960+0.005122 test-logloss:0.455262+0.045387
## [97] train-logloss:0.444928+0.005122 test-logloss:0.455226+0.045391
## [98] train-logloss:0.444881+0.005132 test-logloss:0.455218+0.045501
## [99] train-logloss:0.444841+0.005129 test-logloss:0.455255+0.045490
```

```
plot(data.frame(xgb.model.one)[,1], type='l', col='black', ylab='CV logloss Error', xlab='#
lines(data.frame(xgb.model.one)[,3], type='l', lty=2, col='black')
```



After adding additional variables i.e increaing the levels to 8 and changeing the learning rate to .1 the output shows a different result. THe log loss error reduces quickly as the number of trees in creases. Reduction in log loss is better since it would provide better prediction results. This model is more suitable over the previous model due to the smaller log loss with the varying number of trees.