Data Mining Final-Part2

Ang Li, ANL125@pitt.edu

2/6/2017

Task: analyze dataset D8 audit.csv the objective is to predict the binary (TARGET_Adjusted) target variables. Apply different classification techniques (incl. logistic regression, kNN, Naive Bayesian, decision tree, SVM, and Ensemble methods) on this dataset. Use all available predictors in your models.

Following are data pre-processing part to get the data ready for classification

```
suppressMessages(library(textir))
suppressMessages(library(MASS))
library(class)
library(ROCR)
```

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

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

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

```
library(e1071) #Naive Bayes
library(rpart) # for decision tree
library(ada) # for adaboost
library('ggplot2')
```

```
setwd("/Users/angli/ANG/GoogleDrive/GoogleDrive_Pitt_PhD/UPitt_PhD_0/2017_Spring/Data-Mining-Spring17/da
#setwd("/Users/ANG/GoogleDrive/GoogleDrive_Pitt_PhD/UPitt_PhD_0/2017_Spring/Data-Mining-Spring17/data/f

data = read.csv("students_grade_final.csv")
#data = grading_data
```

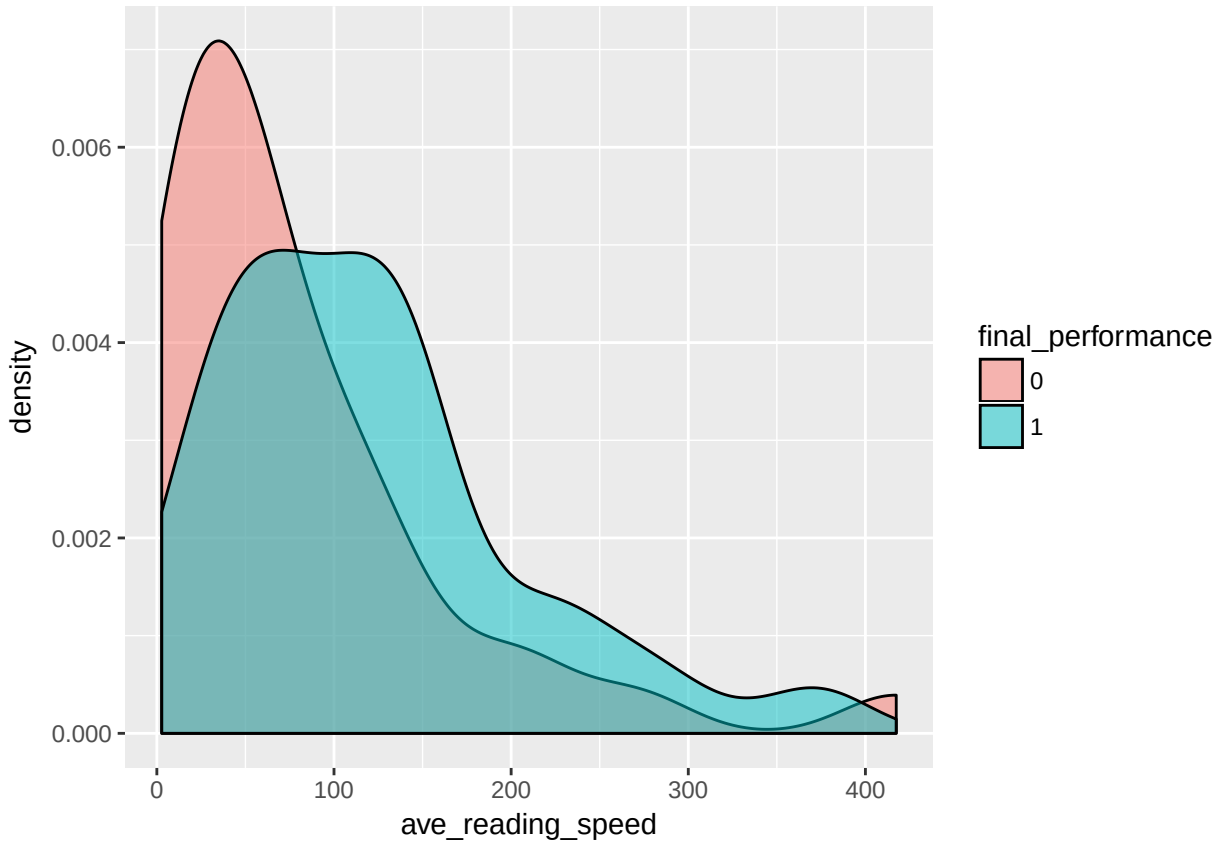
```
#logsitic regression
summary(data$final_comp_grade)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.7756  0.8469  0.8754  0.8785  0.9090  0.9841
```

```
data$final_performance = 0
data$final_performance[which(data$final_comp_grade >= 0.88)]=1
data$final_performance = as.factor(data$final_performance)
summary(data$final_performance)
```

```
##  0  1
## 34 29
```

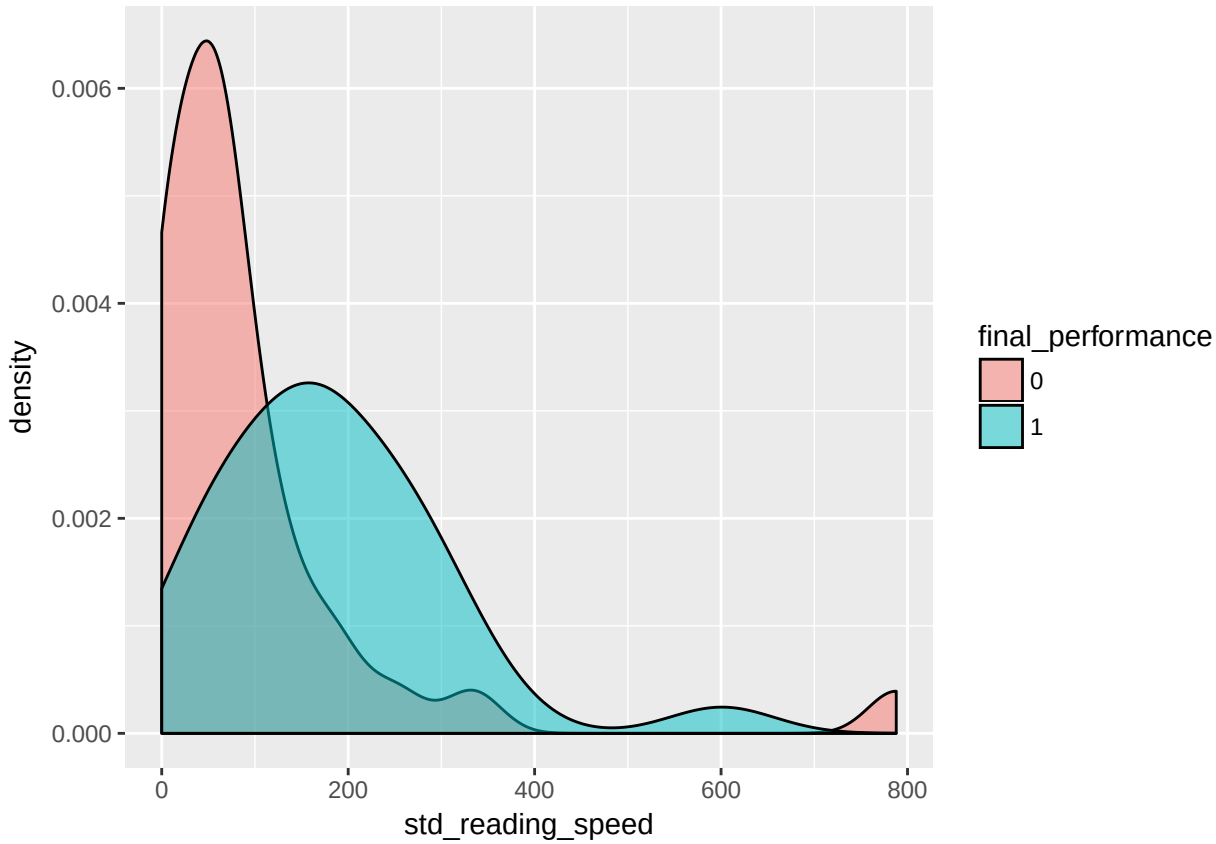
```
#students comparison
ggplot(data, aes(x = ave_reading_speed, fill = final_performance)) +
  geom_density(alpha = 0.5) #+ facet_grid(final_performance ~ .)
```




```
#class_HCI: Anova test to compare means
t.test(ave_reading_speed~final_performance, data=data)
```

```
##
##  Welch Two Sample t-test
##
## data:  ave_reading_speed by final_performance
## t = -1.792, df = 59.846, p-value = 0.07818
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -82.99065   4.55957
## sample estimates:
## mean in group 0 mean in group 1
##        79.50703       118.72257
```

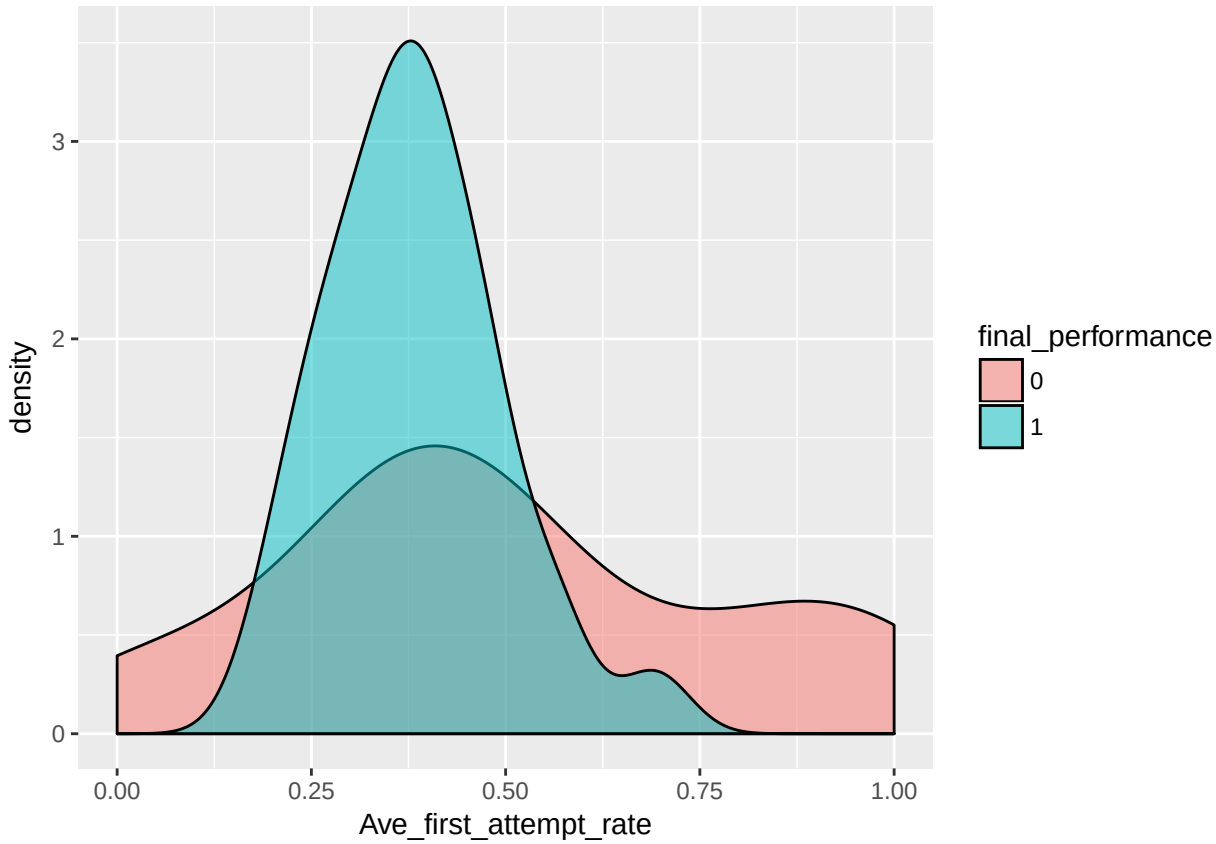
```
ggplot(data, aes(x = std_reading_speed, fill = final_performance)) +
  geom_density(alpha = 0.5) #+ facet_grid(final_performance ~ .)
```




```
#class_HCI: Anova test to compare means
t.test(std_reading_speed~final_performance, data=data)
```

```
##
##  Welch Two Sample t-test
##
## data:  std_reading_speed by final_performance
## t = -2.5193, df = 60.993, p-value = 0.0144
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -152.16886  -17.49778
## sample estimates:
## mean in group 0 mean in group 1
##        95.28073       180.11405
```

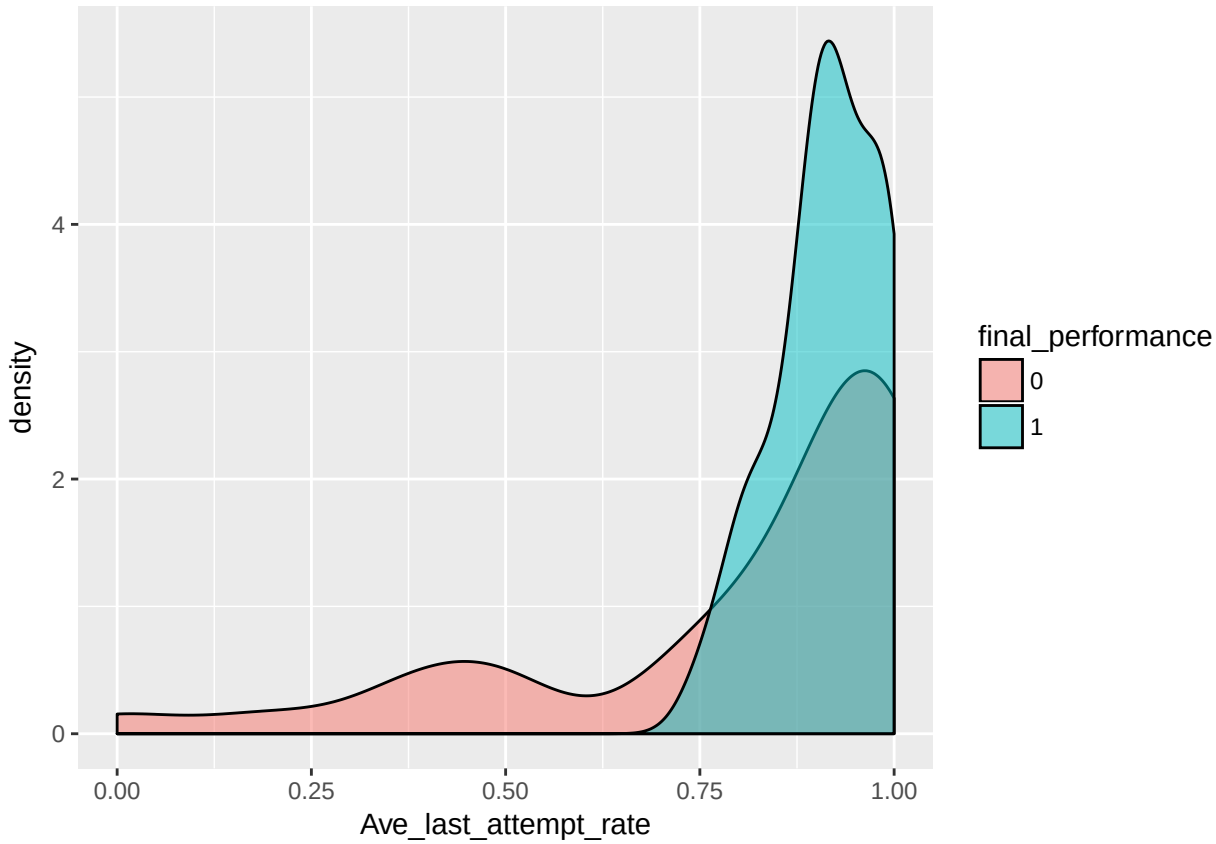
```
ggplot(data, aes(x = Ave_first_attempt_rate, fill = final_performance)) +
  geom_density(alpha = 0.5) #+ facet_grid(final_performance ~ .)
```




```
#class_HCI: Anova test to compare means
t.test(Ave_first_attempt_rate~final_performance, data=data)
```

```
##
##  Welch Two Sample t-test
##
## data:  Ave_first_attempt_rate by final_performance
## t = 2.0831, df = 44.4, p-value = 0.04303
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.003660582 0.219770908
## sample estimates:
## mean in group 0 mean in group 1
##       0.4955954       0.3838796
```

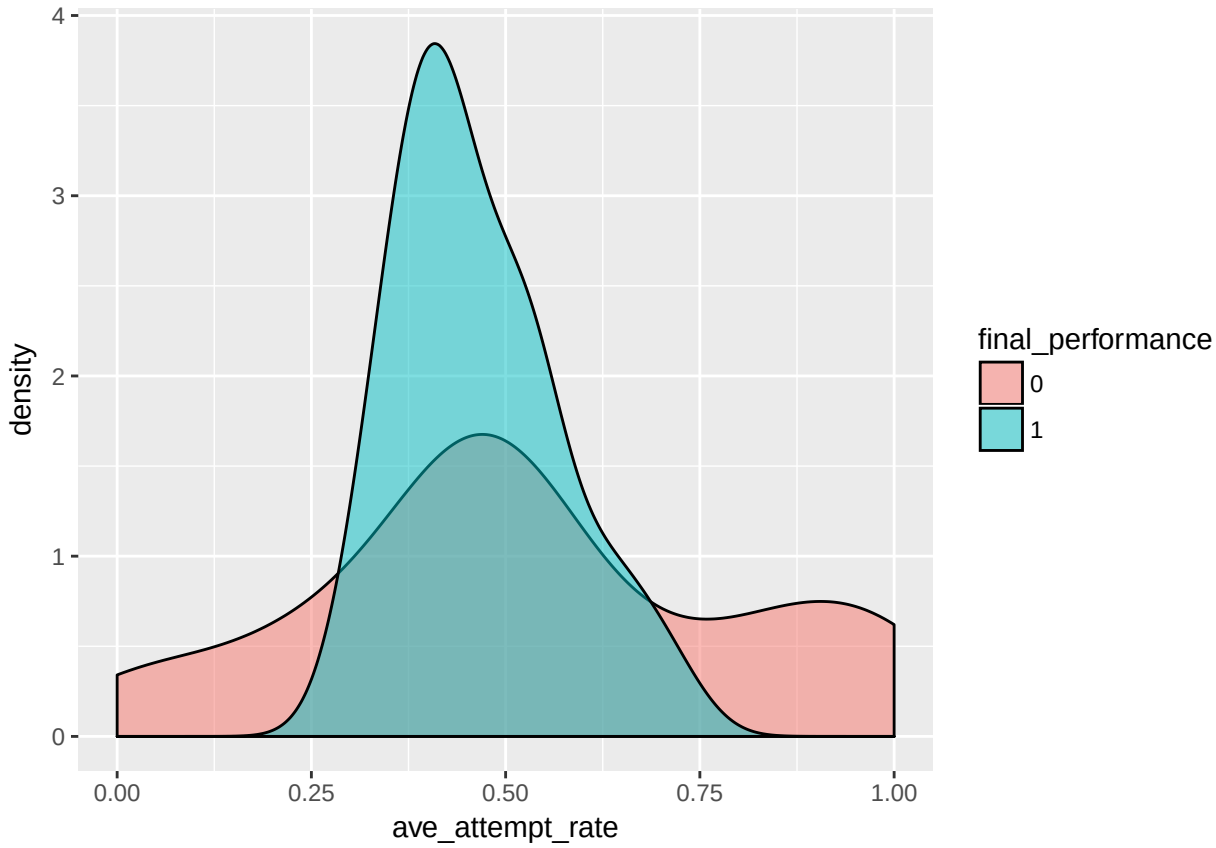
```
ggplot(data, aes(x = Ave_last_attempt_rate, fill = final_performance)) +
  geom_density(alpha = 0.5) #+ facet_grid(final_performance ~ .)
```




```
#class_HCI: Anova test to compare means
t.test(Ave_last_attempt_rate~final_performance, data=data)
```

```
##
##  Welch Two Sample t-test
##
## data:  Ave_last_attempt_rate by final_performance
## t = -2.381, df = 38.023, p-value = 0.02238
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.20694800 -0.01675379
## sample estimates:
## mean in group 0 mean in group 1
##       0.8035942       0.9154451
```

```
ggplot(data, aes(x = ave_attempt_rate, fill = final_performance)) +
  geom_density(alpha = 0.5) #+ facet_grid(final_performance ~ .)
```




```
#class_HCI: Anova test to compare means
t.test(ave_attempt_rate~final_performance, data=data)
```

```
##
##  Welch Two Sample t-test
##
## data:  ave_attempt_rate by final_performance
## t = 1.2528, df = 43.371, p-value = 0.217
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.03897607  0.16690034
## sample estimates:
## mean in group 0 mean in group 1
##       0.5283419       0.4643798
```

```
data = data[c("ave_reading_speed","std_reading_speed","Average_nword","prop_SkipReading",
                      "prop_fullreading","num_question_answered","average_attempt_times","Std_average_a
                      "Ave_first_attempt_rate","Std_first_attempt_rate","Ave_last_attempt_rate","Std_las
                      "ave_attempt_rate","Std_ave_attempt.rate","class_HCI","final_performance")]
data$num_question_answered = log(data$num_question_answered)
data$average_attempt_times = log(data$average_attempt_times)
data$class_HCI =as.factor(data$class_HCI)
colnames(data)[16] = "Y"
```

```
#logsitic regression
m1=glm(Y ~ ave_reading_speed+std_reading_speed
          #+ prop_SkipReading + prop_fullreading
          + average_attempt_times+Std_average_attempts
```

```
          + Ave_first_attempt_rate + Std_first_attempt_rate
          +Ave_last_attempt_rate + Std_last_attempt_rate
          +ave_attempt_rate + Std_ave_attempt.rate
          + class_HCI,  family=binomial(link='logit'),data=data)

summary(m1)
```

```
##
## Call:
## glm(formula = Y ~ ave_reading_speed + std_reading_speed + average_attempt_times +
##     Std_average_attempts + Ave_first_attempt_rate + Std_first_attempt_rate +
##     Ave_last_attempt_rate + Std_last_attempt_rate + ave_attempt_rate +
##     Std_ave_attempt.rate + class_HCI, family = binomial(link = "logit"),
##     data = data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.46246  -0.55207  -0.00126   0.56990   2.26895
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -16.961424  11.091876  -1.529  0.12622
## ave_reading_speed       -0.019397   0.010648  -1.822  0.06851 .
## std_reading_speed        0.013132   0.006556   2.003  0.04518 *
## average_attempt_times   -6.770597   2.835347  -2.388  0.01694 *
## Std_average_attempts     0.252717   0.465023   0.543  0.58682
## Ave_first_attempt_rate  -8.310631   7.644366  -1.087  0.27697
## Std_first_attempt_rate -11.238639   8.519611  -1.319  0.18712
## Ave_last_attempt_rate   35.164074  13.508249   2.603  0.00924 **
## Std_last_attempt_rate   12.837498   7.292996   1.760  0.07837 .
## ave_attempt_rate       -13.638940   9.921830  -1.375  0.16924
## Std_ave_attempt.rate    13.819029  11.446461   1.207  0.22733
## class_HCI1              -0.337055   0.965327  -0.349  0.72697
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 86.939  on 62  degrees of freedom
## Residual deviance: 48.329  on 51  degrees of freedom
## AIC: 72.329
##
## Number of Fisher Scoring iterations: 7
```

```
#---leave one out Cross validation-----
n=nrow(data)
#m1 model is to use all predictors
ptest_all1 = data.frame() #store prediction result data frame
for(i in 1:nrow(data)){
    #Segement your data by fold using the which() function
    train1 = c(1:n)
    train = train1[train1!=i] ## pick elements that are different from k
    #Use the training set to train a logistic regression model to predict the response variable
    m1=glm(Y ~ ave_reading_speed+std_reading_speed
```

```
          #+ prop_SkipReading + prop_fullreading
          + average_attempt_times+Std_average_attempts
          + Ave_first_attempt_rate + Std_first_attempt_rate
          +Ave_last_attempt_rate + Std_last_attempt_rate
          +ave_attempt_rate + Std_ave_attempt.rate
          + class_HCI, family=binomial(link='logit'),data=data[train ,])
    ## prediction: predicted default probabilities for cases in test set
    ptest = predict(m1,newdata=data[-train ,], type="response")
    ptest_data=data.frame(predictions=ptest,labels=data$Y[-train])
    ptest_all1 = rbind(ptest_all1,ptest_data)
}


btest=floor(ptest_all1$predictions+0.5)  ## use floor function to clamp the value to 0 or 1
conf.matrix = table(ptest_all1$labels,btest)
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall =TP/(TP+FN) #TPR 0.9219258
precision = TP/(TP+FP) #0.8683
TNR = TN/(TN+FP)
FPR = 1- TNR
f1 = 2*precision*recall/(precision+recall) #0.8942884
error=(FN+FP)/sum(conf.matrix) #0.3015873
accuracy_lg = 1-error

## ROC for hold-out period
roc_data1=ptest_all1
pred1 <- prediction(roc_data1$predictions,roc_data1$labels)
auc_lg = performance(pred1, "auc")@y.values[[1]] #0.7444219

#model summary
summary(m1)
```

```
##
## Call:
## glm(formula = Y ~ ave_reading_speed + std_reading_speed + average_attempt_times +
##     Std_average_attempts + Ave_first_attempt_rate + Std_first_attempt_rate +
##     Ave_last_attempt_rate + Std_last_attempt_rate + ave_attempt_rate +
##     Std_ave_attempt.rate + class_HCI, family = binomial(link = "logit"),
##     data = data[train, ])
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.45759  -0.55313  -0.00072   0.60076   2.18582
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -18.148942  11.522048  -1.575  0.11522
## ave_reading_speed       -0.018860   0.010491  -1.798  0.07221 .
## std_reading_speed        0.012713   0.006462   1.967  0.04913 *
## average_attempt_times   -6.509714   2.810650  -2.316  0.02055 *
## Std_average_attempts     0.272894   0.462174   0.590  0.55488
```

```
## Ave_first_attempt_rate  -7.581489   7.702844  -0.984  0.32499
## Std_first_attempt_rate -10.376925   8.482784  -1.223  0.22122
## Ave_last_attempt_rate   35.625801  13.779935   2.585  0.00973 **
## Std_last_attempt_rate   13.366088   7.451324   1.794  0.07285 .
## ave_attempt_rate       -13.608951   9.944064  -1.369  0.17114
## Std_ave_attempt.rate    12.831485  11.456928   1.120  0.26272
## class_HCI1              -0.310952   0.959149  -0.324  0.74579
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 85.692  on 61  degrees of freedom
## Residual deviance: 47.724  on 50  degrees of freedom
## AIC: 71.724
##
## Number of Fisher Scoring iterations: 7
```

K Nearest Neighbour

```
#KNN
NN_CV <- function(data, k=3){
  #Split the data into 10 equally size folds
  folds <- cut(seq(1,nrow(data)),breaks=nrow(data),labels=FALSE)
  #m1 model is to use all predictors
  #Perform 10 fold cross validation
  ptest_all2 = data.frame() #store prediction result data frame
  for(i in 1:nrow(data)){
      #Segement your data by fold using the which() function
      testIndexes <- which(folds==i,arr.ind=TRUE)
      test <- data[testIndexes, ]
      n.test = nrow(test)
      train <- data[-testIndexes, ]
      #Use the training set to train a logistic regression model to predict the response variable
      x_train = train[ , !(names(train) %in% c("Y"))]
      x_test = test[, !(names(train) %in% c("Y"))]
      y_train = train$Y
      y_test = test$Y
      m2=knn(train=x_train, test=x_test,cl=y_train,k=k,prob = TRUE)
      ptest = attr(m2, "prob") #get the probability results
      ## prediction: predicted default probabilities for cases in test set
      ptest_data=data.frame(predictions=m2, probability=ptest, labels=y_test)#predictions=ptest,labels=
      ptest_all2 = rbind(ptest_all2,ptest_data)
  }
  return(ptest_all2)
}

data_NN = data
#data_NN$Y[which(data_NN$Y == 0)] <- -1 #recode "0" into "-1"

#change the prediction results into the probability of being 1
ptest_all2 = NN_CV(data_NN)
ptest_all2$probability2=1
for (i in 1:nrow(ptest_all2)){
```

```
 if (ptest_all2[i,]$predictions == 0) {ptest_all2[i,]$probability2 = 1-ptest_all2[i,]$probability}
  else {ptest_all2[i,]$probability2 = ptest_all2[i,]$probability}
}

#using the probability of being 1
btest=floor(ptest_all2$probability2+0.5)  ## use floor function to clamp the value to 0 or 1
#ptest_all2$predLabel = btest
test_labels = ptest_all2$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_NN =TP/(TP+FN) #0.4966443
precision_NN = TP/(TP+FP) #0.6132597
f1_NN = 2*precision_NN*recall_NN/(precision_NN+recall_NN) #0.5488257
error=(FN+FP)/sum(conf.matrix) #0.3174603
accuracy_NN=1-error

#AUC
## ROC for hold-out period
roc_data2=ptest_all2
pred_NN3 <- prediction(roc_data2$probability2,roc_data2$labels)
auc2_NN = performance(pred_NN3, "auc")@y.values[[1]] #0.6673428

##--------------------------
#KNN K=5
ptest_all5_NN = NN_CV(data_NN, k=5)
ptest_all5_NN$probability2=1
for (i in 1:nrow(ptest_all5_NN)){
 if (ptest_all5_NN[i,]$predictions == 0) {ptest_all5_NN[i,]$probability2 = 1-ptest_all5_NN[i,]$probabili
  else {ptest_all5_NN[i,]$probability2 = ptest_all5_NN[i,]$probability}
}

#using the probability of being 1
btest=floor(ptest_all5_NN$probability2+0.5)  ## use floor function to clamp the value to 0 or 1
#ptest_all2$predLabel = btest
test_labels = ptest_all5_NN$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_NN5 =TP/(TP+FN)
precision_NN5 = TP/(TP+FP)
f1_NN5 = 2*precision_NN5*recall_NN5/(precision_NN5+recall_NN5)
error=(FN+FP)/sum(conf.matrix)
accuracy_NN5=1-error #0.6825397

#AUC
## ROC for hold-out period
roc_data_NN5=ptest_all5_NN
pred_NN5 <- prediction(roc_data_NN5$probability2,roc_data_NN5$labels)
```

```
auc_NN5 = performance(pred_NN5, "auc")@y.values[[1]] #0.6957404
#--------------------------------------

#KNN K=10
ptest_all10_NN = NN_CV(data_NN, k=10)
ptest_all10_NN$probability2=1
for (i in 1:nrow(ptest_all10_NN)){
 if (ptest_all10_NN[i,]$predictions == 0) {ptest_all10_NN[i,]$probability2 = 1-ptest_all10_NN[i,]$probal
  else {ptest_all10_NN[i,]$probability2 = ptest_all10_NN[i,]$probability}
}

#using the probability of being 1
btest=floor(ptest_all10_NN$probability2+0.5)  ## use floor function to clamp the value to 0 or 1
#ptest_all2$predLabel = btest
test_labels = ptest_all10_NN$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_NN10 =TP/(TP+FN)
precision_NN10 = TP/(TP+FP)
f1_NN10 = 2*precision_NN10*recall_NN10/(precision_NN10+recall_NN10)
error=(FN+FP)/sum(conf.matrix)
accuracy_NN10=1-error #0.5714286

#AUC
## ROC for hold-out period
roc_data_NN10=ptest_all10_NN
pred_NN10 <- prediction(roc_data_NN10$probability2,roc_data_NN10$labels)
auc_NN10 = performance(pred_NN10, "auc")@y.values[[1]] #0.5638945
```

Decision Tree

- The function DT_CV will take the two additional parameters besides data: prune_tree and split_method. When prune_tree set to FALSE, the function will only use the rpart to build the tree without pruning. When prune_tree set to TRUE, the function will prune the tree. It will return the cross validation results.

```
#Decision Tree
#---10 Fold Cross validation-----

DT_CV <- function(data, split_method="information", prune_tree=FALSE){
  #Split the data into 10 equally size folds
  folds <- cut(seq(1,nrow(data)),breaks=nrow(data),labels=FALSE)

  #m1 model is to use all predictors
  #Perform 10 fold cross validation

  ptest_all4 = data.frame() #store prediction result data frame
  for(i in 1:nrow(data)){
      #Segement your data by fold using the which() function
      testIndexes <- which(folds==i,arr.ind=TRUE)
```

```
      test <- data[testIndexes, ]
      n.test = nrow(test)
      train <- data[-testIndexes, ]
      #Use the training set to train a DT model to predict the response variable
      m4=rpart(Y~., data=train, parms = list(split = split_method))
      ## prediction: predicted default probabilities for cases in test set
      if (prune_tree == TRUE){
        pfit<- prune(m4, cp=m4$cptable[which.min(m4$cptable[,"xerror"]),"CP"])
        ptest = predict(pfit, newdata=test)
      } else {
        ptest = predict(m4,newdata=test)
        #ptest = ptest[,2]/rowSums(ptest) # renormalize the prob.
      }
      ptest_data=data.frame(predictions=ptest,labels=test$Y)
      ptest_all4 = rbind(ptest_all4,ptest_data)
  }
  return(ptest_all4)
}

#DT
data_DT = data
#data_NN$Y[which(data_NN$Y == 0)] <- -1 #recode "0" into "-1"
ptest_all4 = DT_CV(data_DT,split_method="information", prune_tree=TRUE)
btest=floor(ptest_all4$predictions.1+0.5)  ## use floor function to clamp the value to 0 or 1
test_labels = ptest_all4$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_DT =TP/(TP+FN) #0.4765101
precision_DT = TP/(TP+FP) #0.6474164
#TNR = TN/(TN+FP) #0.9201102
#FPR = 1- TNR #0.07988981
f1_DT = 2*precision_DT*recall_DT/(precision_DT+recall_DT)
error=(FN+FP)/sum(conf.matrix)
accuracy_DT=1-error #0.6984127

#AUC
## ROC for hold-out period
roc_data4=ptest_all4
pred4_DT <- prediction(roc_data4$predictions.1,roc_data4$labels)
auc4_DT = performance(pred4_DT, "auc")@y.values[[1]] #0.693712


####---DT: gini
ptest_tuned_gini = DT_CV(data_DT, split_method="gini", prune_tree=TRUE)
btest=floor(ptest_tuned_gini$predictions.1+0.5)  ## use floor function to clamp the value to 0 or 1
test_labels = ptest_tuned_gini$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
```

```
FN = conf.matrix[1,2]
recall_tuned_gini =TP/(TP+FN)
precision_tuned_gini = TP/(TP+FP)
f1_tuned_gini = 2*precision_tuned_gini*recall_tuned_gini/(precision_tuned_gini+recall_tuned_gini)
error=(FN+FP)/sum(conf.matrix)
accuracy_tuned_gini=1-error #0.6031746

#AUC
## ROC for hold-out period
roc_data_tuned_gini=ptest_tuned_gini
pred_tuned_gini <- prediction(roc_data_tuned_gini$predictions.1,roc_data_tuned_gini$labels)
auc_tuned_gini = performance(pred_tuned_gini, "auc")@y.values[[1]] #0.5649087
```

SVM

- The function SVM_CV takes the 4 additional parameters besides data: tuned_model, K, gamma, and cost. If the tuned_model set as FALSE, it will only use svm function with all default settings. If the tuned_model set as TRUE, it will take the "Kernel" as K, and user defined gamma and cost.

```
#SVM
SVM_CV <- function(data, tuned_model=FALSE, K="radial", gamma=gamma, cost=cost){
  #---10 Fold Cross validation-----
  #Split the data into 10 equally size folds
  folds <- cut(seq(1,nrow(data)),breaks=nrow(data),labels=FALSE)

  #m1 model is to use all predictors
  #Perform 10 fold cross validation
  ptest_all5 = data.frame() #store prediction result data frame
  for(i in 1:nrow(data)){
      #Segement your data by fold using the which() function
      testIndexes <- which(folds==i,arr.ind=TRUE)
      test <- data[testIndexes, ]
      n.test = nrow(test)
      train <- data[-testIndexes, ]
      if (tuned_model==FALSE){
      #Use the training set to train a DT model to predict the response variable
      m5=svm(Y~., data=train, probability = TRUE) #probability=T
      ## prediction: predicted default probabilities for cases in test set
      ptest = predict(m5,newdata=test,probability = TRUE)
      ptest = attr(ptest,"probabilities")
      ptest = ptest[,which(colnames(ptest)==1)]/rowSums(ptest)
      }
      if (tuned_model==TRUE) {
      model = svm(Y~., data = train, probability=T, kernel=K, gamma=gamma, cost=cost)
      ptest = predict(model, newdata=test, probability=T)
      ptest = attr(ptest,"probabilities")
      ptest = ptest[,which(colnames(ptest)==1)]/rowSums(ptest)
      }

      ptest_data=data.frame(predictions=ptest,labels=test$Y)
      ptest_all5 = rbind(ptest_all5,ptest_data)
  }
  return(ptest_all5)
```

```
}

data_SVM = data
#data_SVM$Y[which(data_SVM$Y == 0)] <- -1 #recode "0" into "-1"
data_SVM$Y = as.factor(data_SVM$Y)

ptest_all5 = SVM_CV(data_SVM,tuned_model=FALSE)
btest=as.numeric(ptest_all5$predictions > 0.5)
test_labels = ptest_all5$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_SVM =TP/(TP+FN)
precision_SVM = TP/(TP+FP)

f1_SVM = 2*precision_SVM*recall_SVM/(precision_SVM+recall_SVM)
error=(FN+FP)/sum(conf.matrix)
accuracy_SVM=1-error #0.7619048

#AUC
## ROC for hold-out period
roc_data5=ptest_all5
pred5_SVM <- prediction(roc_data5$predictions,roc_data5$labels)
auc5_SVM = performance(pred5_SVM, "auc")@y.values[[1]] #0.8073022


#########SVM tuned the model #########
n.obs <- nrow(data_SVM) # no. of observations in dataset
n.train = floor(n.obs*0.7)
train.idx = sample(1:n.obs,n.train)

train.set = data_SVM[train.idx,]
test.set = data_SVM[-train.idx,]

#########radial kernal  #########
tuned_radial <- tune.svm(Y~., data = train.set, kernel="radial", gamma = 10^(-6:-1), cost = 10^(-1:1))
#print(summary(tuned))
gamma = tuned_radial[['best.parameters']]$gamma
cost = tuned_radial[['best.parameters']]$cost

ptest_all_SVM_radial = SVM_CV(data_SVM, tuned_model=TRUE, K="radial", gamma=gamma, cost=cost)
btest=as.numeric(ptest_all_SVM_radial$predictions > 0.5)
test_labels = ptest_all_SVM_radial$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_SVM_radial =TP/(TP+FN)
precision_SVM_radial = TP/(TP+FP)
```

```
f1_SVM_radial = 2*precision_SVM_radial*recall_SVM_radial/(precision_SVM_radial+recall_SVM_radial) #0.57
error=(FN+FP)/sum(conf.matrix)
accuracy_SVM_radial=1-error #0.7936508

#AUC
## ROC for hold-out period
roc_data_SVM_radial=ptest_all_SVM_radial
pred_SVM_radial <- prediction(roc_data_SVM_radial$predictions,roc_data_SVM_radial$labels)
auc_SVM_radial = performance(pred_SVM_radial, "auc")@y.values[[1]] #0.8225152

######Poly kernal############
tuned_poly <- tune.svm(Y~., data = train.set, kernel="polynomial", gamma = 10^(-6:-1), cost = 10^(-1:1))
#print(summary(tuned))
gamma = tuned_poly[['best.parameters']]$gamma
cost = tuned_poly[['best.parameters']]$cost

ptest_all_SVM_poly = SVM_CV(data_SVM, tuned_model=TRUE, K="polynomial", gamma=gamma, cost=cost)
btest=as.numeric(ptest_all_SVM_poly$predictions > 0.5)
test_labels = ptest_all_SVM_poly$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_SVM_poly =TP/(TP+FN) #0.4765101
precision_SVM_poly = TP/(TP+FP) #0.7171717

f1_SVM_poly = 2*precision_SVM_poly*recall_SVM_poly/(precision_SVM_poly+recall_SVM_poly) #0.5725806
error=(FN+FP)/sum(conf.matrix)
accuracy_SVM_poly=1-error

#AUC
## ROC for hold-out period
roc_data_SVM_poly=ptest_all_SVM_poly
pred_SVM_poly <- prediction(roc_data_SVM_poly$predictions,roc_data_SVM_poly$labels)
auc_SVM_poly = performance(pred_SVM_poly, "auc")@y.values[[1]] #0.7707911


##################
######sigmoid kernal
tuned_sig <- tune.svm(Y~., data = train.set, kernel="sigmoid", gamma = 10^(-6:-1), cost = 10^(-1:1))
#print(summary(tuned))
gamma = tuned_sig[['best.parameters']]$gamma
cost = tuned_sig[['best.parameters']]$cost

ptest_all_SVM_sig = SVM_CV(data_SVM, tuned_model=TRUE, K="sigmoid", gamma=gamma, cost=cost)
btest=as.numeric(ptest_all_SVM_sig$predictions > 0.5)
test_labels = ptest_all_SVM_sig$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
```

```
recall_SVM_sig =TP/(TP+FN)
precision_SVM_sig = TP/(TP+FP)

f1_SVM_sig = 2*precision_SVM_sig*recall_SVM_sig/(precision_SVM_sig+recall_SVM_sig) #0.5725806
error=(FN+FP)/sum(conf.matrix)
accuracy_SVM_sig=1-error

#AUC
## ROC for hold-out period
roc_data_SVM_sig=ptest_all_SVM_sig
pred_SVM_sig <- prediction(roc_data_SVM_sig$predictions,roc_data_SVM_sig$labels)
auc_SVM_sig = performance(pred_SVM_sig, "auc")@y.values[[1]] # 0.7332657
```

ensemble

```
# ensemble

Ada_CV <- function(data){
  #---10 Fold Cross validation-----
  #Split the data into 10 equally size folds
  folds <- cut(seq(1,nrow(data)),breaks=nrow(data),labels=FALSE)

  #m1 model is to use all predictors
  #Perform 10 fold cross validation
  ptest_all6 = data.frame() #store prediction result data frame
  for(i in 1:nrow(data)){
      #Segement your data by fold using the which() function
      testIndexes <- which(folds==i,arr.ind=TRUE)
      test <- data[testIndexes, ]
      n.test = nrow(test)
      train <- data[-testIndexes, ]
      #Use the training set to train a DT model to predict the response variable
      m6=ada(Y~., data=train) #probability=T
      ## prediction: predicted default probabilities for cases in test set
      ptest = predict(m6, newdata=test, type='probs')
      ptest = ptest[,2]/rowSums(ptest)
      ptest_data=data.frame(predictions=ptest,labels=test$Y)
      ptest_all6 = rbind(ptest_all6,ptest_data)
  }
  return(ptest_all6)
}

data_ADA = data

ptest_all6 = Ada_CV(data_ADA)
btest=as.numeric(ptest_all6$predictions > 0.5)
test_labels = ptest_all6$labels
conf.matrix = table(factor(test_labels,levels = c(1,0)),factor(btest,levels = c(1,0)))#change factor le
TP = conf.matrix[1,1]
TN = conf.matrix[2,2]
FP = conf.matrix[2,1]
FN = conf.matrix[1,2]
recall_ADA =TP/(TP+FN) #0.5592841
```

```
precision_ADA = TP/(TP+FP) #0.7122507

f1_ADA = 2*precision_ADA*recall_ADA/(precision_ADA+recall_ADA)
error=(FN+FP)/sum(conf.matrix)
accuracy_ADA=1-error

#AUC
## ROC for hold-out period
roc_data6=ptest_all6
pred_ADA <- prediction(roc_data6$predictions,roc_data6$labels)
auc6_ADA = performance(pred_ADA, "auc")@y.values[[1]] #0.7555781
```

Summarize the model performance based on the table and the ROC plot in one or two paragraphs.

```
All_accuracy_list = c(accuracy_lg, accuracy_NN, accuracy_NN5, accuracy_NN10, accuracy_DT, accuracy_tuned

All_AUC_list = c(auc_lg, auc2_NN, auc_NN5, auc_NN10, auc4_DT, auc_tuned_gini, auc_SVM_radial, auc_SVM_po

All_Result_DF = data.frame(All_accuracy_list, All_AUC_list, row.names=c("Logistic Regression","K Neares

All_Result_DF
```

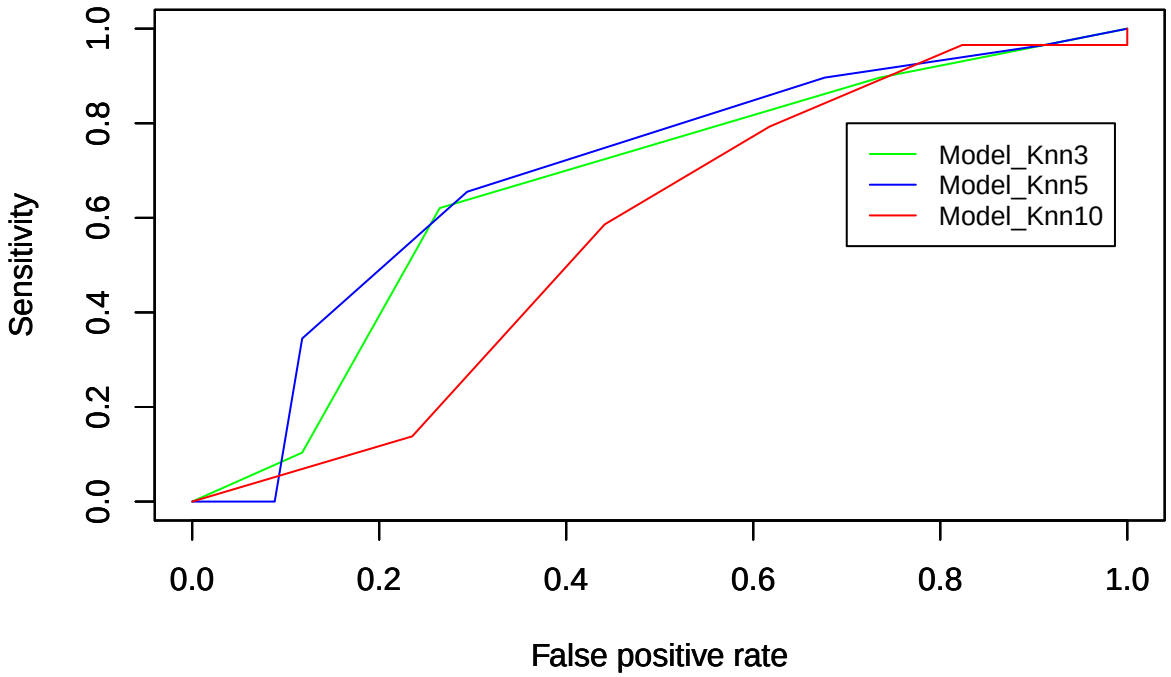
```
##                                       All_accuracy_list All_AUC_list
## Logistic Regression                           0.6984127    0.7444219
## K Nearest Neighbour K=3                       0.6825397    0.6673428
## K Nearest Neighbour K=5                       0.6825397    0.6957404
## K Nearest Neighbour K=10                      0.5714286    0.5638945
## Decision Tree Pruned informaion Split         0.7142857    0.6389452
## Decision Tree Pruned Gini Split               0.6190476    0.6054767
## SVM_radial                                    0.7936508    0.8265720
## SVM_poly                                      0.6666667    0.7363083
## SVM_sigmoid                                   0.8095238    0.7373225
## Ada Boosting                                  0.6984127    0.7444219
```

```
#write.csv(All_Result_DF, "student_perform_classify.csv")
```

- As shown in the table above that SVM perform the best within all the algorithms.
- When comparing the different K in the KNN algorithm, I tested using k=3,5,10, and found that k=5 provided the best formance in terms of both high F1 and AUC score.

```
#ROC curve for Nearest Neighbour
perf_NN3 <- performance(pred_NN3, "sens", "fpr")
perf_NN5 <- performance(pred_NN5, "sens", "fpr")
perf_NN10 <- performance(pred_NN10, "sens", "fpr") #best
plot(perf_NN3, col="green")
par(new=TRUE)
plot(perf_NN5, col="blue")
par(new=TRUE)
plot(perf_NN10, col="red")

legend(0.7, 0.8, legend=c("Model_Knn3", "Model_Knn5","Model_Knn10"),
       col=c("green","blue","red"), lty=1, cex=0.8)
```

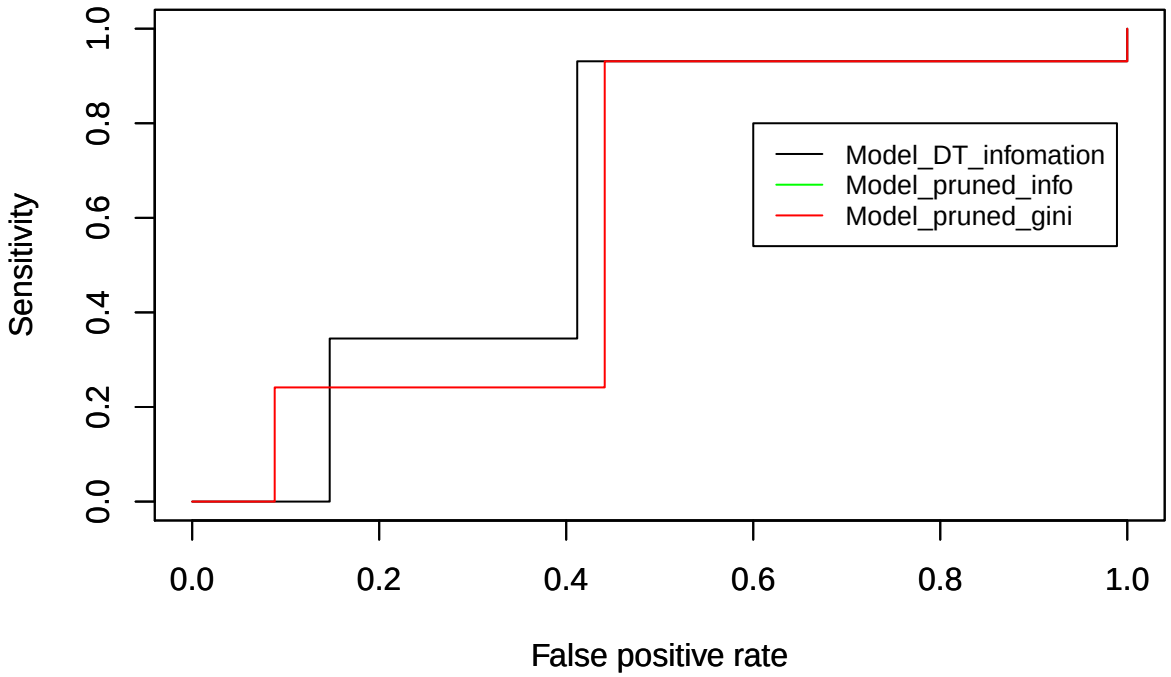


- When comparing different methods in spliting tree (information and gini), I did not observe much difference among these methods, as showed in the graph below.

```
#ROC curve for Decision Tress
perf_DT1 <- performance(pred4_DT, "sens", "fpr") #best
perf_DT3 <- performance(pred_tuned_gini, "sens", "fpr")

plot(perf_DT1)
par(new=TRUE)
plot(perf_DT3, col="red")

legend(0.6, 0.8, legend=c("Model_DT_infomation", "Model_pruned_info", "Model_pruned_gini"),
       col=c("black","green","red"), lty=1, cex=0.8)
```

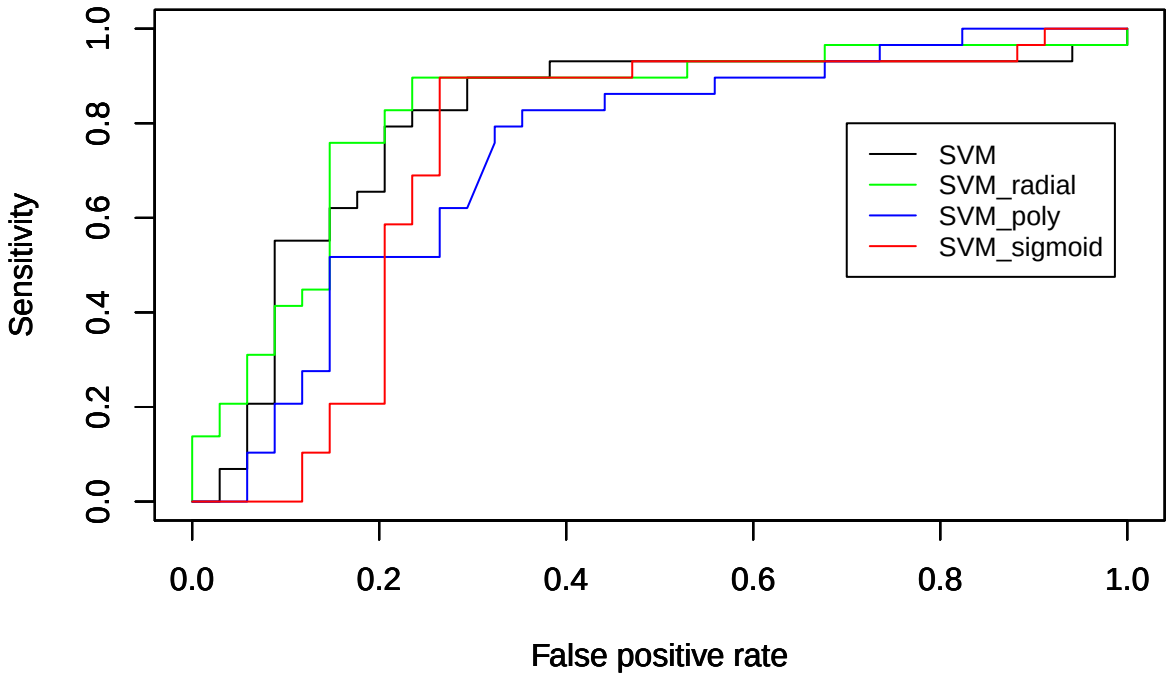


- When comparing different kernals using under SVM, I observed that the radio kernal performed the best (red line), followed by the radial kernal in terms of AUC curve.

```
#ROC curve for SVM
perf_SVM <- performance(pred5_SVM, "sens", "fpr")
perf_SVM_radial <- performance(pred_SVM_radial, "sens", "fpr")
perf_SVM_poly <- performance(pred_SVM_poly, "sens", "fpr")
perf_SVM_sig <- performance(pred_SVM_sig, "sens", "fpr") #best

plot(perf_SVM)
par(new=TRUE)
plot(perf_SVM_radial, col="green")
par(new=TRUE)
plot(perf_SVM_poly, col="blue")
par(new=TRUE)
plot(perf_SVM_sig, col="red")

legend(0.7, 0.8, legend=c("SVM", "SVM_radial", "SVM_poly","SVM_sigmoid"),
       col=c("black","green","blue","red"), lty=1, cex=0.8)
```



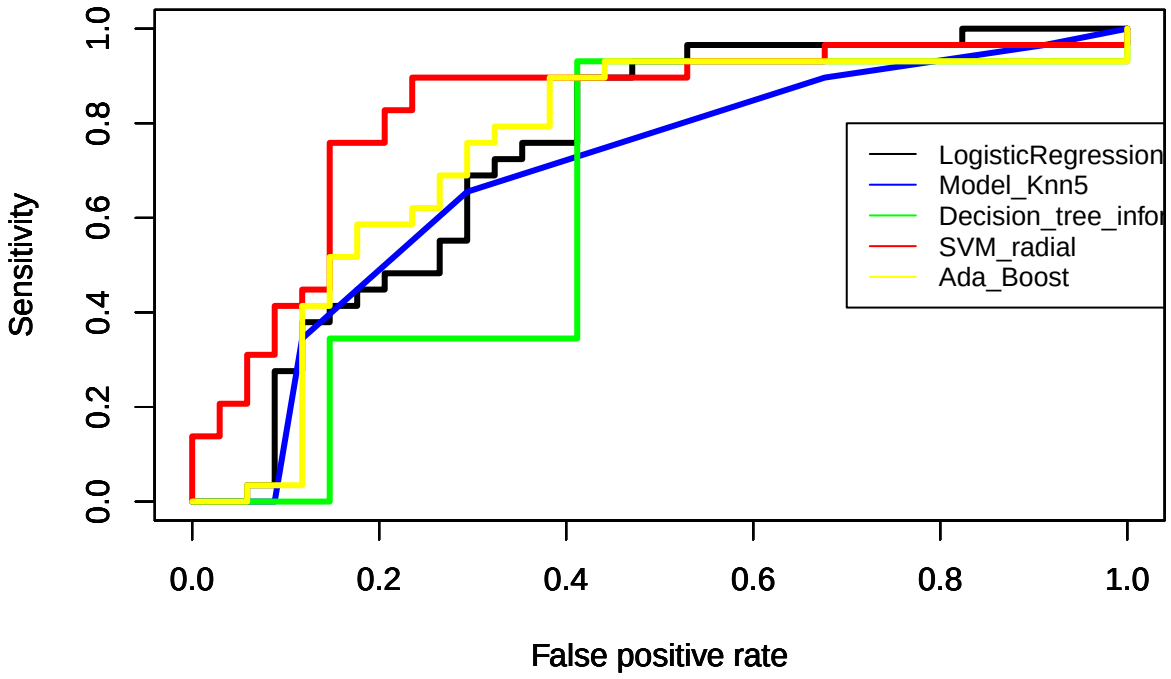

```
All_Result_DF
```

```
##                                        All_accuracy_list All_AUC_list
## Logistic Regression                            0.6984127    0.7444219
## K Nearest Neighbour K=3                        0.6825397    0.6673428
## K Nearest Neighbour K=5                        0.6825397    0.6957404
## K Nearest Neighbour K=10                       0.5714286    0.5638945
## Decision Tree Pruned informaion Split          0.7142857    0.6389452
## Decision Tree Pruned Gini Split                0.6190476    0.6054767
## SVM_radial                                     0.7936508    0.8265720
## SVM_poly                                       0.6666667    0.7363083
## SVM_sigmoid                                    0.8095238    0.7373225
## Ada Boosting                                   0.6984127    0.7444219
```

```
#ROC Curve for all best model
perf_lg <- performance(pred1, "sens", "fpr") #best
perf_ADA <- performance(pred_ADA, "sens", "fpr")

plot(perf_lg,lwd=3)
par(new=TRUE)
plot(perf_NN5, col="blue",lwd=3)
par(new=TRUE)
plot(perf_DT1, col="green",lwd=3)
par(new=TRUE)
plot(perf_SVM_radial, col="red",lwd=3)
par(new=TRUE)
plot(perf_ADA, col="yellow",lwd=3)

legend(0.7, 0.8, legend=c("LogisticRegression", "Model_Knn5","Decision_tree_information","SVM_radial",
       col=c("black","blue","green","red","yellow"), lty=1, cex=0.8)
```


Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
LogisticRegression
Model_Knn5
Decision_tree_infor
SVM_radial
Ada_Boost