Classification Models for Internet Ads

XLin

November 14, 2017

1. Introduction

Many Internet sites draw income from third-party advertisements, usually in the form of images. Usually users are not interested in those ads. And images tend to dominate a page's total download time and make it slower for us to open the webpage. In order to filter, this project aimes develop classification approaches for predicting whether an image on a webpage is an advertisement ("ad") or not ("non-ad").

10 classification methods were used to train models and make predictions. Evaluation of the models were make to see which models perform better.

```r
#Loading packages
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```r
library(ggplot2)
library(plyr)
library(caret)
```

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

```r
library(ROCR)
```

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

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

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

```r
library(crossval)
```

```
##
## Attaching package: 'crossval'
```

```
## The following object is masked from 'package:caret':
##
##     confusionMatrix
```

```r
library(MASS)
library(e1071)
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR

## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```r
library(nnet)
library(ggrepel)
library(randomForest)
```

```
## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

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

```r
library(klaR)
library(gbm)
```

```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

2. Data Description

The data set is open-source data downloaded from Kaggle. It represents a set of possible ads collected from multiple Internet pages.

Predicted variables: geometry of images, phrases occuring in the URL, the image's URL, the anchor text, words occuring near the anchor text and so on.

Target variable : binary advertisement ("ad") and not advertisement ("non-ad")

Size: 3279 observations (2821 nonads, 458 ads) 1558 variables (3 continuous, the rest binary)

Read data into R

```r
setwd('D:/statistics/3rd semester/stat517/final_proj/Internet Ad/ad-dataset')

inad<-read.table(file = 'ad.data', sep = ',')
dim(inad)
```

```
## [1] 3279 1559
```

```r
inad[1:10,1:10]
```

```
##      V1   V2     V3 V4 V5 V6 V7 V8 V9 V10
## 1   125  125    1.0  1  0  0  0  0  0   0
## 2    57  468 8.2105  1  0  0  0  0  0   0
## 3    33  230 6.9696  1  0  0  0  0  0   0
## 4    60  468    7.8  1  0  0  0  0  0   0
## 5    60  468    7.8  1  0  0  0  0  0   0
## 6    60  468    7.8  1  0  0  0  0  0   0
## 7    59  460 7.7966  1  0  0  0  0  0   0
## 8    60  234    3.9  1  0  0  0  0  0   0
## 9    60  468    7.8  1  0  0  0  0  0   0
## 10   60  468    7.8  1  0  0  0  0  0   0
```

```r
inad[,c(1,2,3)]<-apply(inad[,c(1,2,3)],2,as.numeric)
```

```
## Warning in apply(inad[, c(1, 2, 3)], 2, as.numeric): NAs introduced by
## coercion

## Warning in apply(inad[, c(1, 2, 3)], 2, as.numeric): NAs introduced by
## coercion

## Warning in apply(inad[, c(1, 2, 3)], 2, as.numeric): NAs introduced by
## coercion
```

```r
inad = na.omit(inad)

colnames(inad)[1559] <- "Ad"
#inad$Ad<-as.factor(ifelse(inad$Ad ==inad$Ad[1],1,0))
inad$V4<-as.integer(inad$V4)
#remove columns whose variance is equal to zero
inad2=inad[,-1559]
inad2<-inad2[,apply(inad2, 2, var, na.rm=TRUE) != 0]
inad2=cbind(inad2,inad$Ad)
colnames(inad2)[1431] <- "Ad"

inad3<-inad2
# standardize all continous variable
inad2[,-1431] <- apply(inad2[,-1431],2,scale)
```

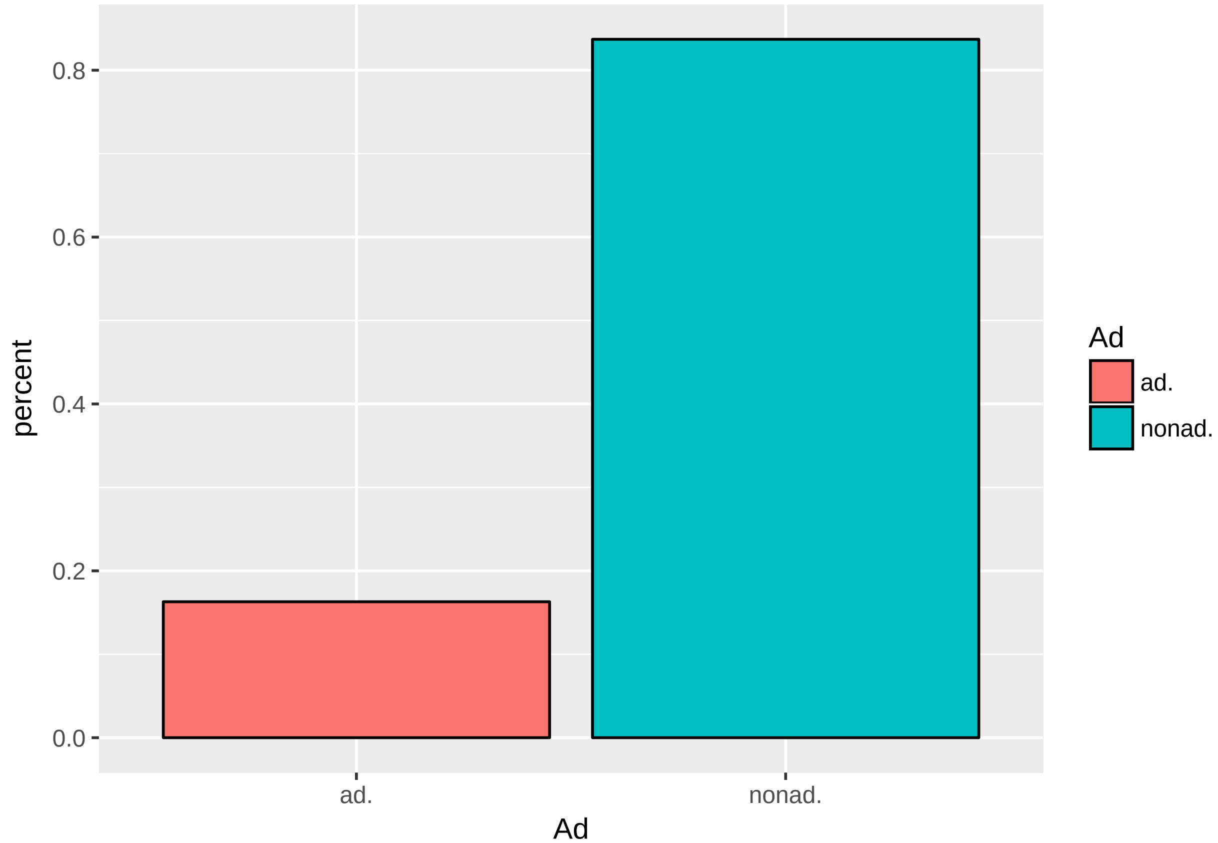
Check The Response

```r
Sys.time()
```

```
## [1] "2018-03-04 22:17:14 PST"
```

```r
#check thr response
Non_ad  <- sum(inad2$Ad == 'nonad.')/2369
ad <- sum(inad2$Ad == 'ad.')/2369

dat <- data.frame(
  Ad = factor(c("nonad.","ad.")),
  percent = c(Non_ad , ad)
```

```
)
ggplot(data=dat, aes(x=Ad, y=percent, fill=Ad)) +
  geom_bar(colour="black", stat="identity")
```



From the plot above, we cna see that less than 80% of the images are advertisements

Principle component analysis

```
t= proc.time()
ad.pca=prcomp(inad2[,-1431],scale=FALSE)
names(ad.pca)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
#The rotation measure provides the principal component loading.
#Each column of rotation matrix contains the principal component loading vector
ad.pca$rotation[1:5,1:4] #first 4 principal components and first 5 rows
```

```
##             PC1          PC2           PC3          PC4
## V1 -0.007630349  0.008873936 -0.0028437416  0.013036151
## V2 -0.038255701  0.045374140 -0.0310723981  0.007252952
## V3 -0.006596172  0.007151080 -0.0063199939 -0.006205335
## V4  0.001539056 -0.029978646 -0.0123431427  0.002280624
## V5  0.001102382 -0.001315656  0.0004387153  0.001103987
```

```
dim(ad.pca$x)
```

```
## [1] 2369 1430
```

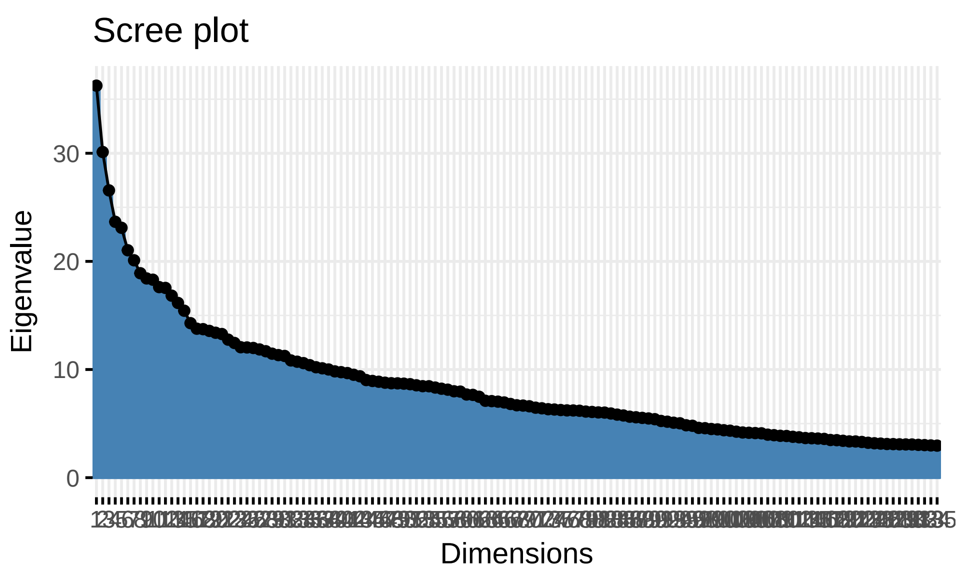
```r
# standard deviation of each principal component
ad.sd = ad.pca$sdev
ad.var=ad.pca$sdev^2  ##compute variance
ad.var[1:10]
```

```
##  [1] 36.25291 30.11637 26.56775 23.66269 23.10616 21.03108 20.10239
##  [8] 18.90437 18.42445 18.30627
```

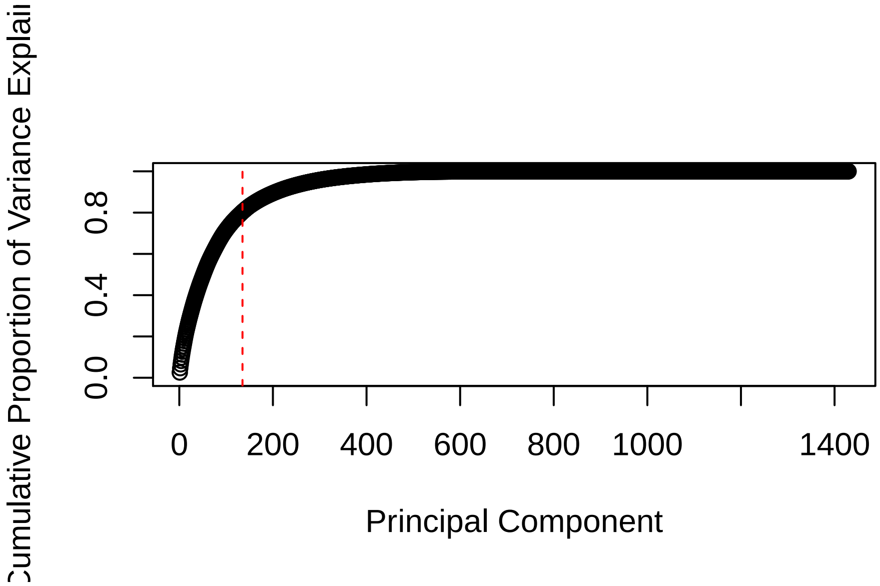
```r
#proportion of variance explained
pve=ad.var/sum(ad.var)

# number of components to achieve account for 80% of the total variance
which.max(cumsum(pve)[cumsum(pve)<=0.804])
```

```
## [1] 135
```

```r
#plot the principal components
#Proportion of Variance Explained
# the first 30 components explain about 95% of the total variability
fviz_screeplot(ad.pca, ncp=135, choice="eigenvalue") #library(factoextra)
```

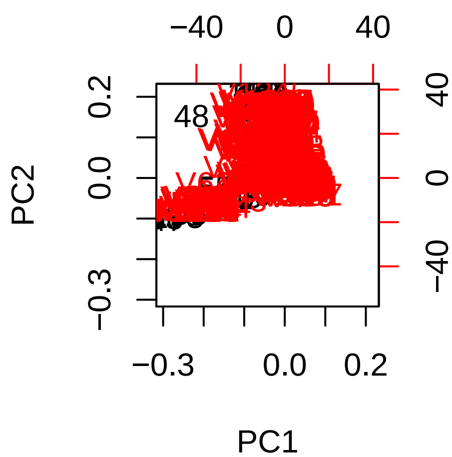



```r
plot(cumsum(pve), xlab="Principal Component",
     ylab="Cumulative Proportion of Variance Explained", ylim=c(0,1),type='b')
abline(v=135, col='red', lty=2)
```

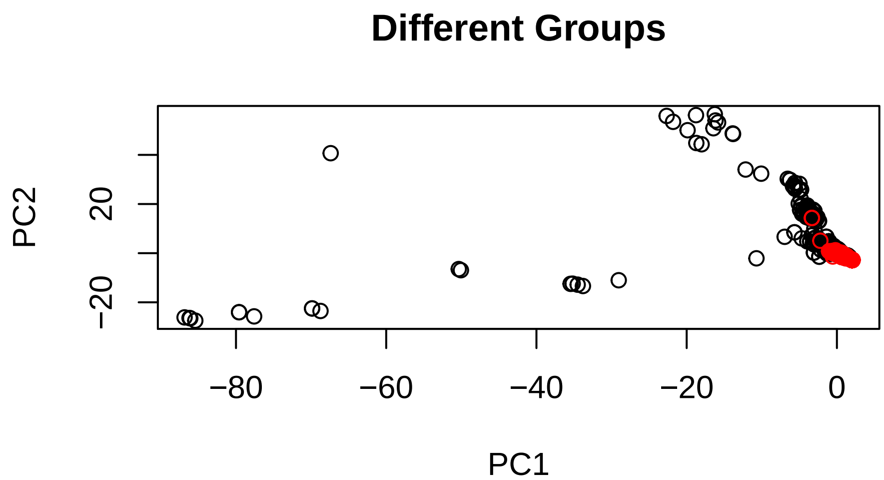



```r
#plot the resultant principal components
biplot(ad.pca)
```

```
## Warning in arrows(0, 0, y[, 1L] * 0.8, y[, 2L] * 0.8, col = col[2L], length
## = arrow.len): zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(0, 0, y[, 1L] * 0.8, y[, 2L] * 0.8, col = col[2L], length
## = arrow.len): zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(0, 0, y[, 1L] * 0.8, y[, 2L] * 0.8, col = col[2L], length
## = arrow.len): zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(0, 0, y[, 1L] * 0.8, y[, 2L] * 0.8, col = col[2L], length
## = arrow.len): zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(0, 0, y[, 1L] * 0.8, y[, 2L] * 0.8, col = col[2L], length
## = arrow.len): zero-length arrow is of indeterminate angle and so skipped
```



```r
AdClasses <- factor(inad2$Ad)
plot(main="Different Groups",ad.pca$x[,1:135], col = AdClasses)
```




```r
#choose the 135 principle components as new variables
adnew=as.data.frame(ad.pca$x[,1:135])

#levels(adnew$Ad) <- make.names(levels(factor(adnew$Ad)))
adnew$Ad<-inad2$Ad

proc.time()-t
```

```
##    user  system elapsed
##   23.23    0.06   23.64
```

Since the data has about 1500 variables, principle component analysis were uesd to reduced the dimension. After conducting the principle component analysis, 135 principle components account for about 80% of the total variability, therefore, the 135 PCs were used to train the classificatioon models

Randomly divide the data set into two sets of labels 1 and 2

```r
ptm<-proc.time()
#Use labels 1 as the training data set and labels 2 as the test data set
set.seed(123)
idx1=sample(1:2,dim(adnew)[1],repl=TRUE)

ad_train<-adnew[idx1==1,]  #training set
X_train<-ad_train[,-136]
Y_train<-ad_train[,136]

ad_test<-adnew[idx1==2,]   #testing set
X_test<-ad_test[,-136]
Y_test<-ad_test[,136]

proc.time() - ptm
```

```
##    user  system elapsed
```

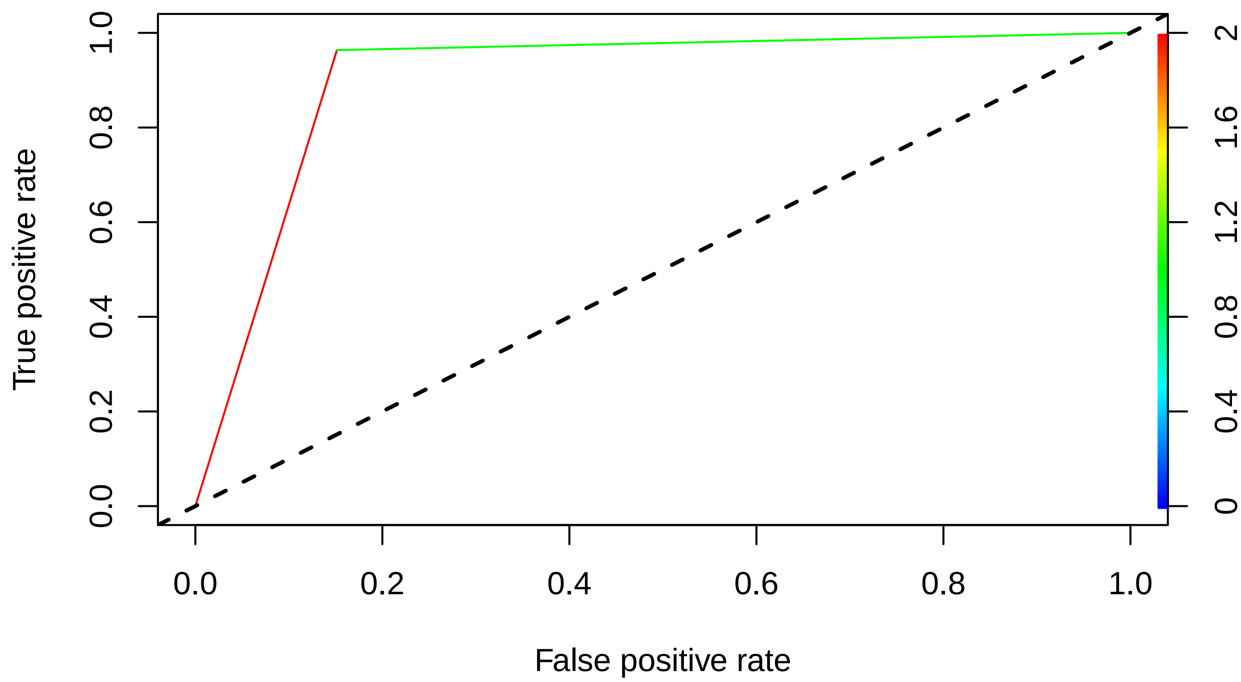
```
##     0.03     0.00     0.03
```

Classification Methods

Logistic Regression Analysis

```
ptm<-proc.time()
set.seed(123)
ctrl <- trainControl(method = "cv", number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

###  tune logistic regression
glm.fit <- train(Ad ~ ., data = ad_train,
                        method = "glm", family = binomial,
                        metric = "Sens",
                        trControl = ctrl)
time.LR<-proc.time()-ptm
time.LR
```

```
ptm<-proc.time()
# using the test date to obtain predicted probabilities of Ad
glm.pred <- predict(glm.fit, ad_test, type = "raw")

table(glm.pred,ad_test$Ad)
mean(glm.pred==ad_test$Ad)
# Plot ROC and AUC for LR
glm.prob<- predict(glm.fit, ad_test, type = "prob")

LRPred <- prediction(glm.prob[,2], ad_test$Ad)
LRPerf <- performance(LRPred, "tpr", "fpr")
plot(LRPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=2, col="black")
```




```
#AUC
AUC.LR<-performance(LRPred, "auc")

#Corresponding Performance Measures
LR.pred <- factor(as.factor(glm.pred), c('nonad.', 'ad.'), labels = c("No-Ad", "Ad"))
LR.Actual <- factor(as.factor(ad_test$Ad), c('nonad.', 'ad.'), labels = c("No-Ad", "Ad"))

CMLR <- confusionMatrix(LR.Actual, LR.pred , negative = "No-Ad" )
DE.LR<-diagnosticErrors(CMLR)
DE.LR
proc.time()-ptm
```

From the results and ROC curve, we can see that Logistic Regression Analysis has about 94% accuracy rate, 85% sensitivity and 96% specitivity, the ROC seemed not bad.

Linear Discriminant Analysis

```
ptm= proc.time()
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

###  tune logistic regression
```

```r
lda.fit <- train(Ad ~ ., data = ad_train, method = "lda",
                         metric = "Sens",
                         trControl = ctrl)
time.LDA<-proc.time() - ptm
time.LDA
```

```r
ptm= proc.time()
lda.fit
```

```
## Linear Discriminant Analysis
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9741107  0.7064286  0.9909596
```

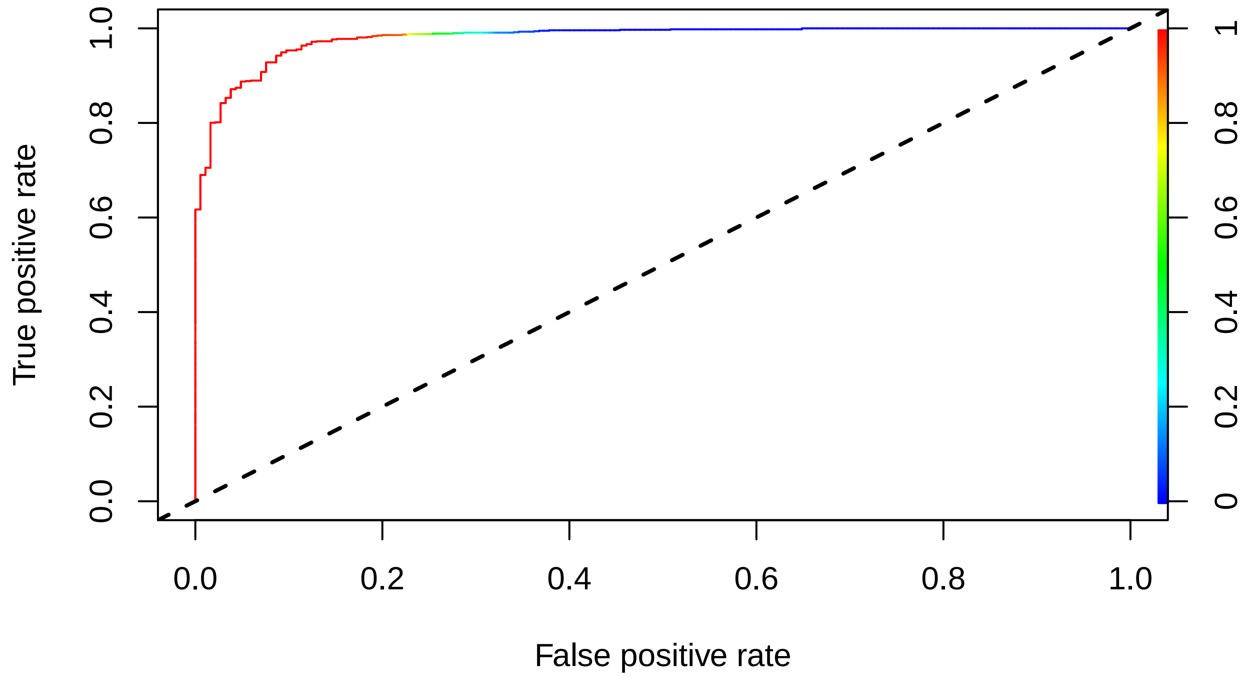
```r
# using the test date to obtain predicted probabilities of Ad
#Predict using the model
lda.pred=predict(lda.fit, ad_test)

#Accuracy of the model
table(lda.pred,ad_test$Ad)
```

```
##
## lda.pred ad. nonad.
##   ad.    138     12
##   nonad.  47    975
```

```r
mean(lda.pred==ad_test$Ad)
```

```
## [1] 0.9496587
```

```r
lda.prob=predict(lda.fit, ad_test, type='prob')
LDA_Pred <- prediction(lda.prob[,2], ad_test$Ad)
LDA_Perf <- performance(LDA_Pred, "tpr", "fpr")
plot(LDA_Perf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=2, col="black")
```




```r
#AUC
AUC.LDA<-performance(LDA_Pred, "auc")

#Corresponding Performance Measures
LDA.class <- factor(as.factor(lda.pred), c('nonad.', 'ad.'), labels = c("No-Ad", "Ad"))
LDA.Actual <- factor(as.factor(ad_test$Ad), c('nonad.', 'ad.'), labels = c("No-Ad", "Ad"))

CMLDA<- confusionMatrix(LDA.Actual, LDA.class , negative = "No-Ad" )
DE.LDA<-diagnosticErrors(CMLDA)
DE.LDA
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9496587 0.7459459 0.9878419 0.9200000 0.9540117 5.4746369
## attr(,"negative")
## [1] "No-Ad"
```

```r
proc.time() - ptm
```

```
##    user  system elapsed
##    0.42    0.00    0.43
```

From the results and ROC curve, we can see that Linear Discriminant Analysis has about 95% accuracy rate, 75% sensitivity and 98.8% specitivity, the ROC looks much better than the ROC of logistic regression.

Quadratic Discriminant Analysis

```r
ptm<-proc.time()
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
#Classify Using Quadratic Discriminant Analysis

qda.fit <- train(Ad ~ ., data = ad_train, method = "qda",
                        metric = "Sens",
                        trControl = ctrl)
time.QDA<-proc.time() - ptm
time.QDA
```

```r
ptm<-proc.time()
qda.fit
```

```
## Quadratic Discriminant Analysis
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8050227  0.7161905  0.8754646
```

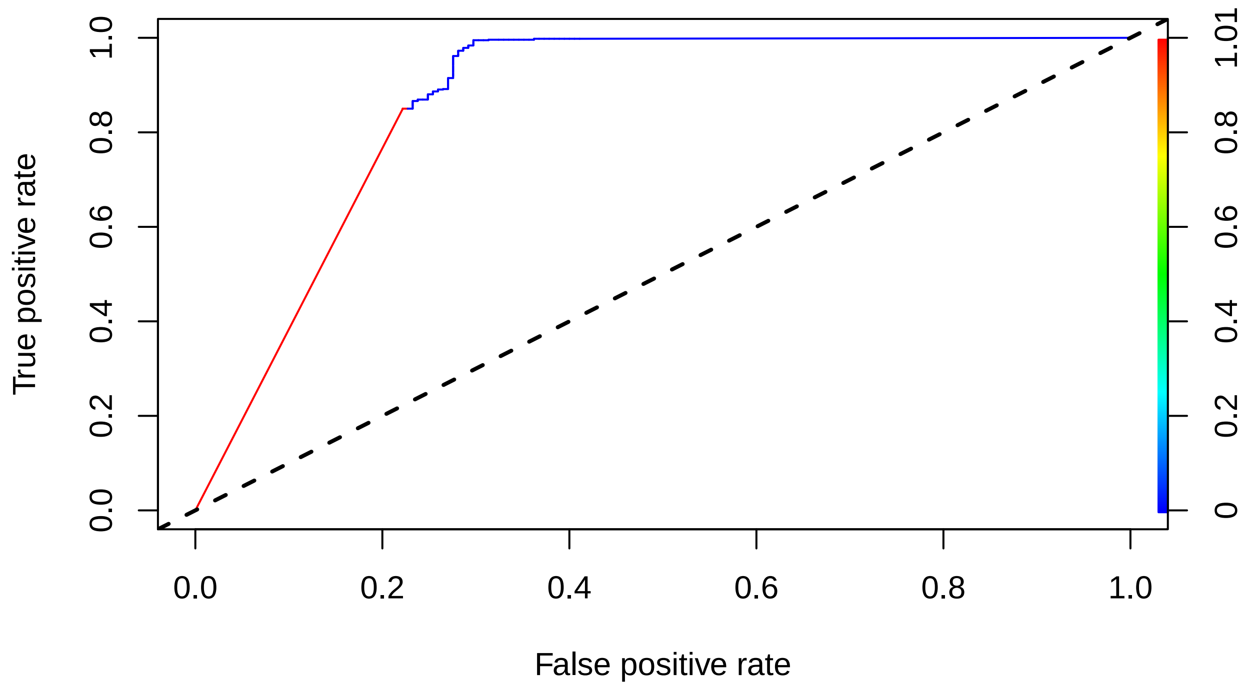
```r
# using the test date to obtain predicted probabilities of Ad
#Predict using the model
qda.pred=predict(qda.fit, ad_test)
#Accuracy of the model
table(qda.pred,ad_test$Ad)
```

```
##
## qda.pred ad. nonad.
##   ad.    144    148
##   nonad.  41    839
```

```r
mean(qda.pred==ad_test$Ad)
```

```
## [1] 0.8387372
```

```r
qda.prob=predict(qda.fit, ad_test,type='prob')
QDA_Pred <- prediction(qda.prob[,2], ad_test$Ad)
QDA_Perf <- performance(QDA_Pred, "tpr", "fpr")
plot(QDA_Perf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=2, col="black")
```




```r
#AUC
AUC.QDA<-performance(QDA_Pred, "auc")

#Corresponding Performance Measures
QDA.class <- factor(as.factor(qda.pred), c('nonad.', 'ad.'), labels = c("No-Ad", "Ad"))
QDA.Actual <- factor(as.factor(ad_test$Ad), c('nonad.', 'ad.'), labels = c("No-Ad", "Ad"))

CMQDA<- confusionMatrix(QDA.Actual, QDA.class , negative = "No-Ad" )
DE.QDA<-diagnosticErrors(CMQDA)
DE.QDA
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.8387372 0.7783784 0.8500507 0.4931507 0.9534091 2.9912397
## attr(,"negative")
## [1] "No-Ad"
```

```r
proc.time() - ptm
```

```
##    user  system elapsed
##    0.18    0.00    0.19
```

From the results and ROC curve, we can see that Quadratic Discriminant Analysis has about 84% accuracy rate, 78% sensitivity and 85% specitivity, the ROC looks worse than the ROC curves of both linear Discriminant Analysis anf logistic regression.

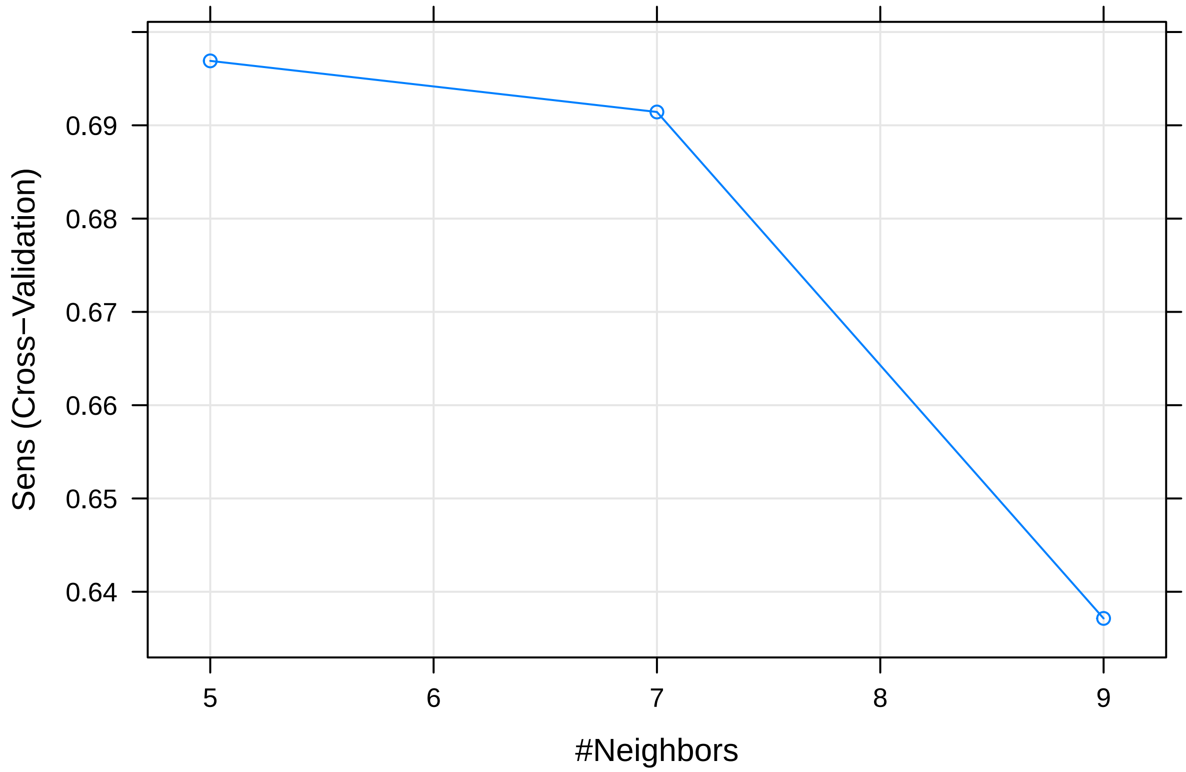
** K Neareat Neighbor **

```r
ptm<-proc.time()
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
#Classify Using KNN
knn.fit <- train(Ad ~ ., data = ad_train, method = "knn",
                 preProcess = c("center","scale"),
                 metric = "Sens",trControl = ctrl)
time.KNN<-proc.time() - ptm
time.KNN
```

```r
ptm<-proc.time()
knn.fit
```

```
## k-Nearest Neighbors
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## Pre-processing: centered (135), scaled (135)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   5  0.9402491  0.6969048  0.9798990
##   7  0.9556771  0.6914286  0.9859192
##   9  0.9481949  0.6371429  0.9879293
##
## Sens was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```r
plot(knn.fit)
```



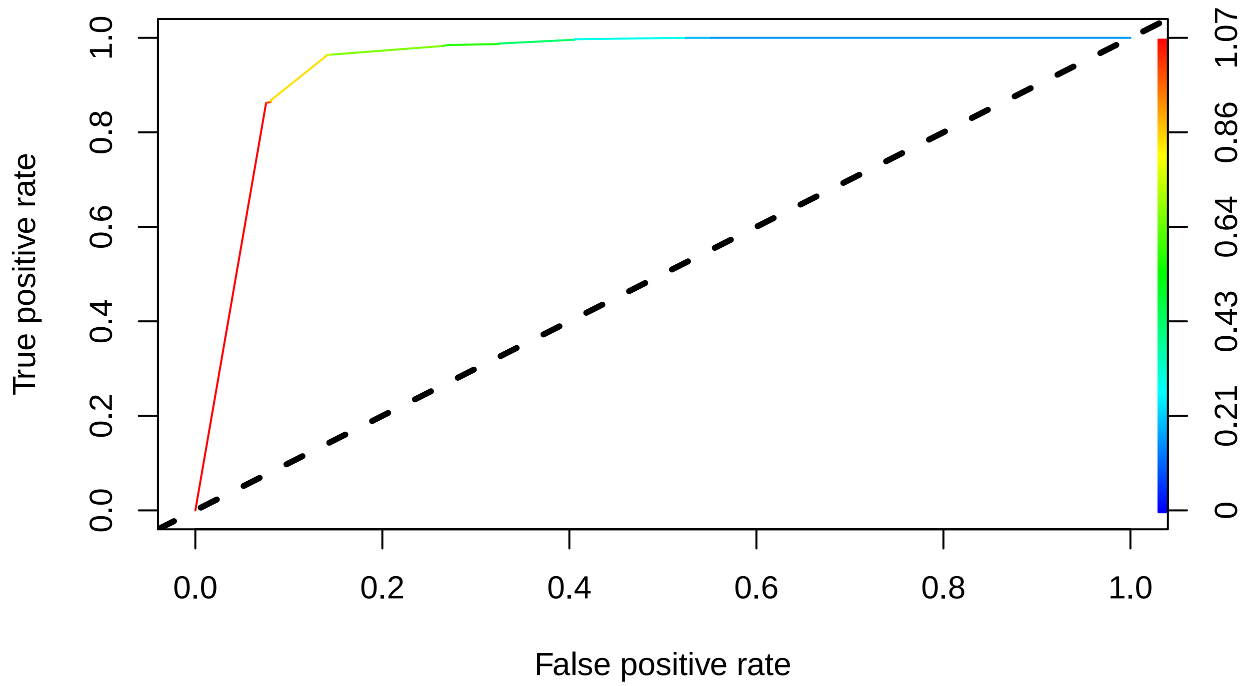

```r
#prediction
knn.pred2 <- predict(knn.fit, X_test)
table(knn.pred2,Y_test)
```

```
##           Y_test
## knn.pred2 ad. nonad.
##    ad.     131     14
##    nonad.   54    973
```

```r
mean(knn.pred2==Y_test)
```

```
## [1] 0.9419795
```

```r
# Plot ROC and AUC for KNN
knn.prob <- predict(knn.fit, X_test, type = 'prob')
KNNPred <- prediction(knn.prob[,2], Y_test)
KNNPerf <- performance(KNNPred, "tpr", "fpr")
plot(KNNPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```
#AUC
AUC.KNN<-performance(KNNPred, "auc")
#Corresponding Performance Measures
KNNPrediction <- factor(as.factor(knn.pred2),  c('nonad.', 'ad.'),labels = c("Not-Ad", "Ad"))
KNNActual <- factor(as.factor(Y_test),  c('nonad.', 'ad.'),labels = c("Not-Ad", "Ad"))

CMKNN <- confusionMatrix(KNNActual,KNNPrediction, negative = "Not-Ad" )
DE.KNN<-diagnosticErrors(CMKNN)
DE.KNN
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9419795 0.7081081 0.9858156 0.9034483 0.9474197 5.1275400
## attr(,"negative")
## [1] "Not-Ad"
```

```
proc.time() - ptm
```

```
##    user  system elapsed
##    0.81    0.00    0.81
```

From the results above, the KNN model performance the best when k=5, with high accuracy and high sensitivity. After using k=5 to make predictions, we can see that it has about 94% accuracy rate, 70% sensitivity and 98.6% specitivity, the ROC looks reasonably good.

Naive Bayes

```r
ptm<-proc.time()
#Classification Using Naive Bayes
set.seed(123)
NB.fit <- train(X_train,Y_train, method = "nb",
                trControl =trainControl(method='cv',number=10))
#prediction
NB.pred <- predict(NB.fit, X_test)
NB.probs <- predict(NB.fit, X_test, type="prob")
time.NB<-proc.time() - ptm
time.NB
```

```r
ptm<-proc.time()
NB.fit
```

```
## Naive Bayes
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.7008361  0.3562630
##    TRUE      0.9081150  0.6812415
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

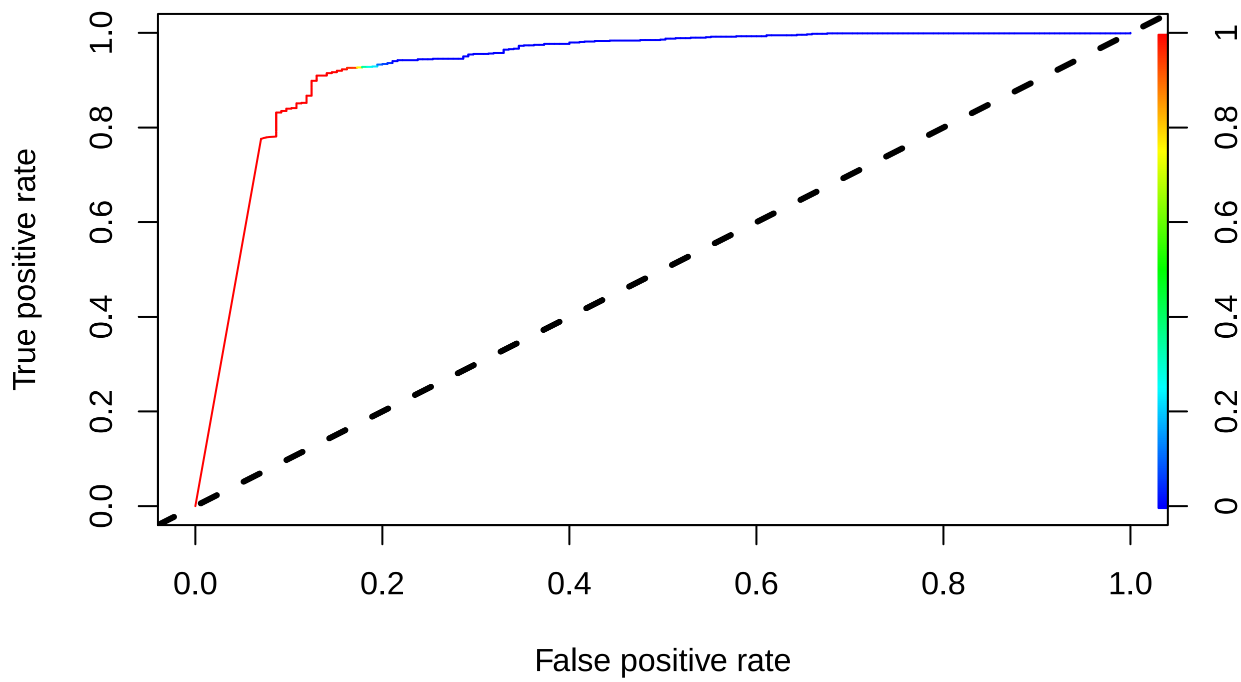
```r
table(NB.pred, Y_test)
```

```
##         Y_test
## NB.pred  ad. nonad.
##   ad.    153     72
##   nonad.  32    915
```

```r
mean(NB.pred==Y_test)
```

```
## [1] 0.9112628
```

```r
# Plot ROC and AUC for NB
NBPred <- prediction(NB.probs[,2], Y_test)
NBPerf <- performance(NBPred, "tpr", "fpr")
plot(NBPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```r
#AUC
AUC.NB<-performance(NBPred, "auc")

#Corresponding Performance Measures
NBPrediction <- factor(as.factor(NB.pred),c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
NBActual <- factor(as.factor(Y_test), c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))

CMNB <- confusionMatrix(NBActual, NBPrediction, negative = "Not-Ad" )
DE.NB<-diagnosticErrors(CMNB)
DE.NB
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9112628 0.8270270 0.9270517 0.6800000 0.9662091 4.1069600
## attr(,"negative")
## [1] "Not-Ad"
```

```r
proc.time() - ptm
```

```
##    user  system elapsed
##    0.27    0.00    0.36
```

From the results and ROC curve, we can see that the Naive Bayes model has about 91% accuracy rate, 83% sensitivity and 93% specitivity, the ROC looks OK.

** Bagging **

```
ptm<-proc.time()
# Bagging
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
#expand.grid(.mtry=135, .ntree=c(1500, 2000, 2500))
bag.fit <- train(Ad~., data=ad_train, method="treebag", trControl=ctrl)

#bag.fit <- bag(X_train, Y_train, B = 10,
#                bagControl = bagControl(fit = ctreeBag$fit,
#                predict = ctreeBag$pred,
#                aggregate = ctreeBag$aggregate))
time.BAG<-proc.time()- ptm
time.BAG
```

```
ptm<-proc.time()
bag.fit
```

```
## Bagged CART
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.9753759  0.8407143  0.975899
```

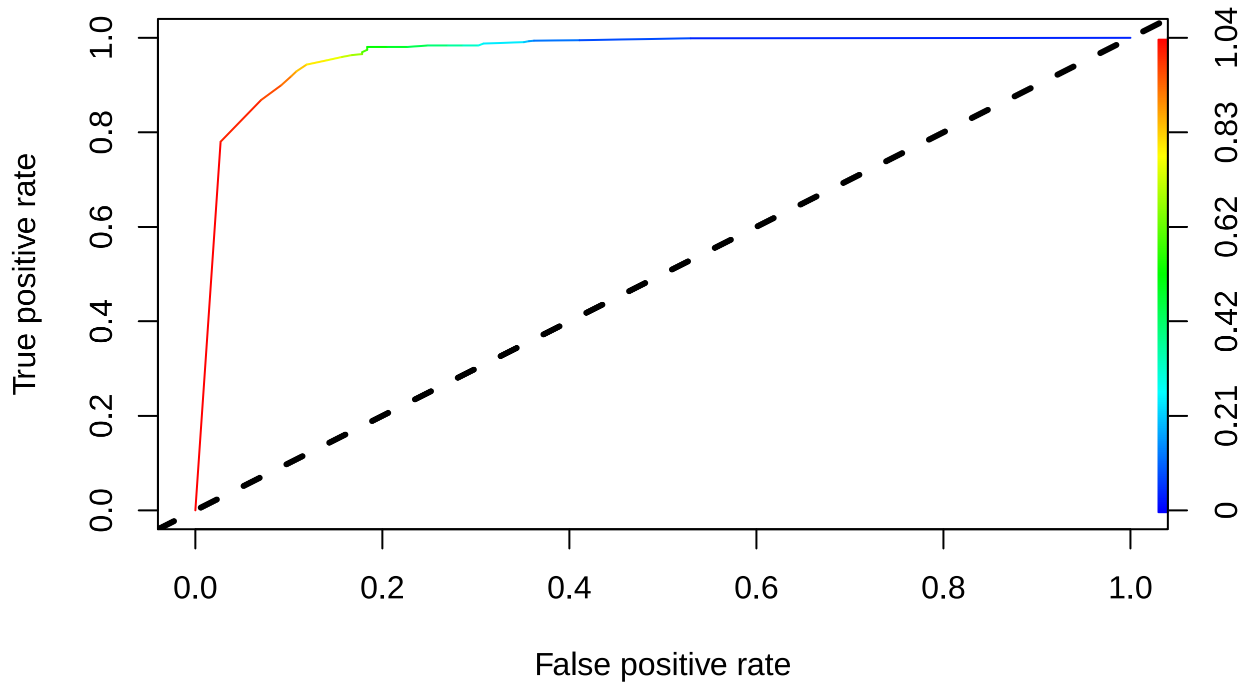
```
#Bagging Prediction
bag.pred = predict(bag.fit,newdata=X_test)
table(bag.pred, Y_test)
```

```
##         Y_test
## bag.pred ad. nonad.
##   ad.    151     19
##   nonad.  34    968
```

```
mean(bag.pred==Y_test)
```

```
## [1] 0.9547782
```

```
# Plot ROC and AUC for Bagging
bag.prob<- predict(bag.fit,newdata=X_test,type="prob")
BAGPred <- prediction(bag.prob[,2], Y_test)
BAGPerf <- performance(BAGPred, "tpr", "fpr")
plot(BAGPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```
#AUC
AUC.BAG<-performance(BAGPred, "auc")
#Corresponding Performance Measures
BAGPrediction <- factor(as.factor(bag.pred), c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
BAGActual <- factor(as.factor(Y_test),c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))

CMBAG <- confusionMatrix(BAGActual, BAGPrediction, negative = "Not-Ad" )
DE.BAG<-diagnosticErrors(CMBAG)
DE.BAG
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9547782 0.8162162 0.9807497 0.8882353 0.9660679 5.4217124
## attr(,"negative")
## [1] "Not-Ad"
```

```
proc.time()-ptm
```

```
##    user  system elapsed
##    0.28    0.00    0.28
```

From the results and ROC curve, we can see that the bagging model has about 95% accuracy rate, 82% sensitivity and 97% specitivity, the ROC looks pretty good.

Random Forests

```r
ptm<-proc.time()
# Random Search
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
#expand.grid(.mtry=c(10:15), .ntree=c(1500, 2000, 2500))
#seq(4,16,4)
set.seed(123)
rf.fit <- train(Ad~., data=ad_train, method="rf",
                tunegrid=expand.grid(.mtry=seq(0,20,5), .ntree=c(1500, 2000, 2500)),
                trControl=ctrl)

time.RF<-proc.time() - ptm
time.RF
```

```r
ptm<-proc.time()
rf.fit
```

```
## Random Forest
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##     2   0.9789861  0.7559524  0.9969798
##    68   0.9782630  0.8654762  0.9819192
##   135   0.9747307  0.8557143  0.9768889
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```r
##Random Forests Prediction
rf.pred = predict(rf.fit,newdata=ad_test)
table(rf.pred, Y_test)
```

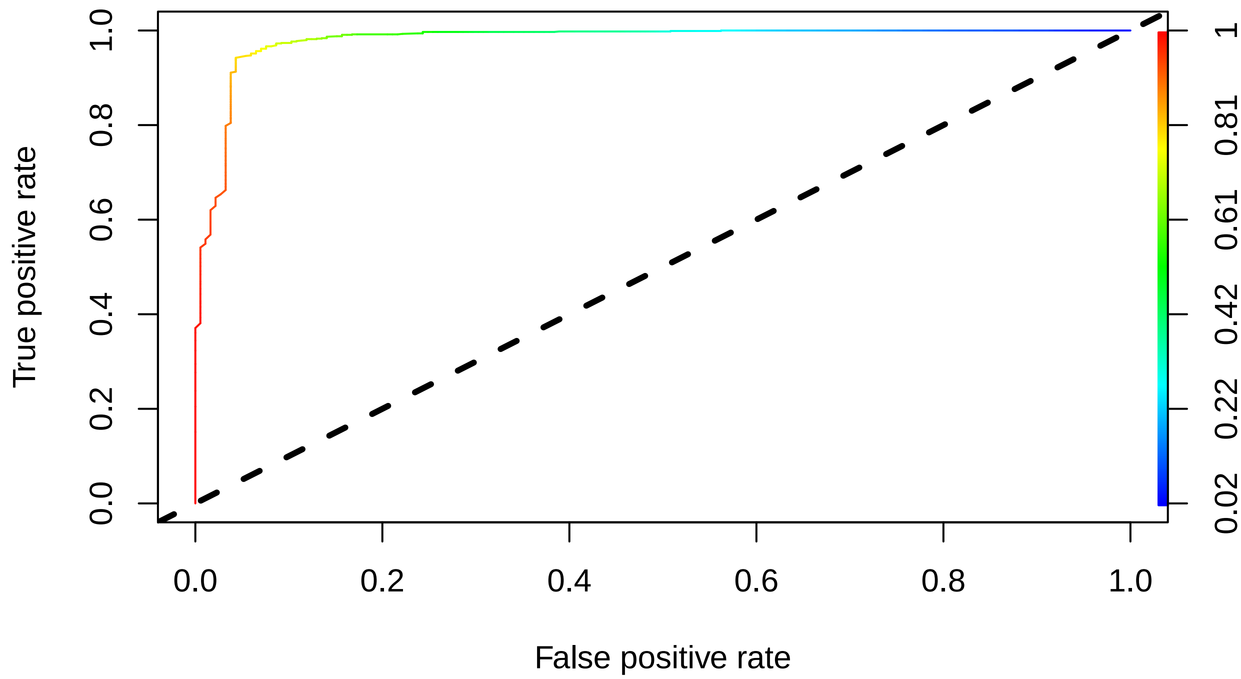
```
##          Y_test
## rf.pred   ad. nonad.
##   ad.     137      3
##   nonad.   48    984
```

```r
mean(rf.pred==Y_test)
```

```
## [1] 0.9564846
```

```r
# Plot ROC and AUC for KNN
#prob got from the predicted model
rf.prob = predict(rf.fit,newdata=ad_test,type="prob")
```

```r
RFPred <- prediction(rf.prob[,2], Y_test)
RFPerf <- performance(RFPred, "tpr", "fpr")
plot(RFPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```r
#AUC
AUC.RF<-performance(RFPred, "auc")

#Corresponding Performance Measures
RFPrediction <- factor(as.factor(rf.pred),
                       c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
RFActual <- factor(as.factor(Y_test),
                   c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))

CMRF <- confusionMatrix(RFActual, RFPrediction, negative = "Not-Ad" )
DE.RF<-diagnosticErrors(CMRF)
DE.RF
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9564846 0.7405405 0.9969605 0.9785714 0.9534884 6.8417935
## attr(,"negative")
## [1] "Not-Ad"
```

```r
proc.time()-ptm
```

```
##    user  system elapsed
##    0.32    0.00    0.31
```

From the results above, the Random Forests model performance better when mtry = 2. After using the model to make predictions, we can see that it has about 95% accuracy rate, 72% sensitivity and 99.6% specitivity, the ROC looks pretty good.

Boosting

```
ptm<-proc.time()
# Boosting fit
set.seed(123)
fitControl = trainControl(method="cv", number=10,  summaryFunction=defaultSummary)

#Using the caret package the get the model preformance in the best iteration.
boost.model = train(Ad~., data=ad_train, method="gbm",distribution="bernoulli",
                    trControl=fitControl, verbose=F,
                    tuneGrid=data.frame(.n.trees=seq(50,1000,50), .shrinkage=0.01,
                                        .interaction.depth=1, .n.minobsinnode=1))

time.BOOST<-proc.time() -ptm
time.BOOST
```

```
ptm<-proc.time()
boost.model
```

```
## Stochastic Gradient Boosting
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   n.trees  Accuracy   Kappa
##     50     0.8320844  0.0000000
##    100     0.9373380  0.7502205
##    150     0.9406853  0.7665643
##    200     0.9415257  0.7716869
##    250     0.9423659  0.7772504
##    300     0.9440257  0.7850705
##    350     0.9498870  0.8107779
##    400     0.9507274  0.8140990
##    450     0.9523801  0.8203238
##    500     0.9548734  0.8307358
##    550     0.9565540  0.8376482
##    600     0.9565540  0.8376482
##    650     0.9565540  0.8376482
##    700     0.9565540  0.8376482
##    750     0.9557137  0.8349832
##    800     0.9557137  0.8349832
##    850     0.9557137  0.8349832
##    900     0.9557137  0.8349832
```

```
##    950      0.9573804  0.8423591
##   1000      0.9573804  0.8423591
##
## Tuning parameter 'interaction.depth' was held constant at a value of
##  1
## Tuning parameter 'shrinkage' was held constant at a value of
##  0.01
## Tuning parameter 'n.minobsinnode' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 950,
##  interaction.depth = 1, shrinkage = 0.01 and n.minobsinnode = 1.
```

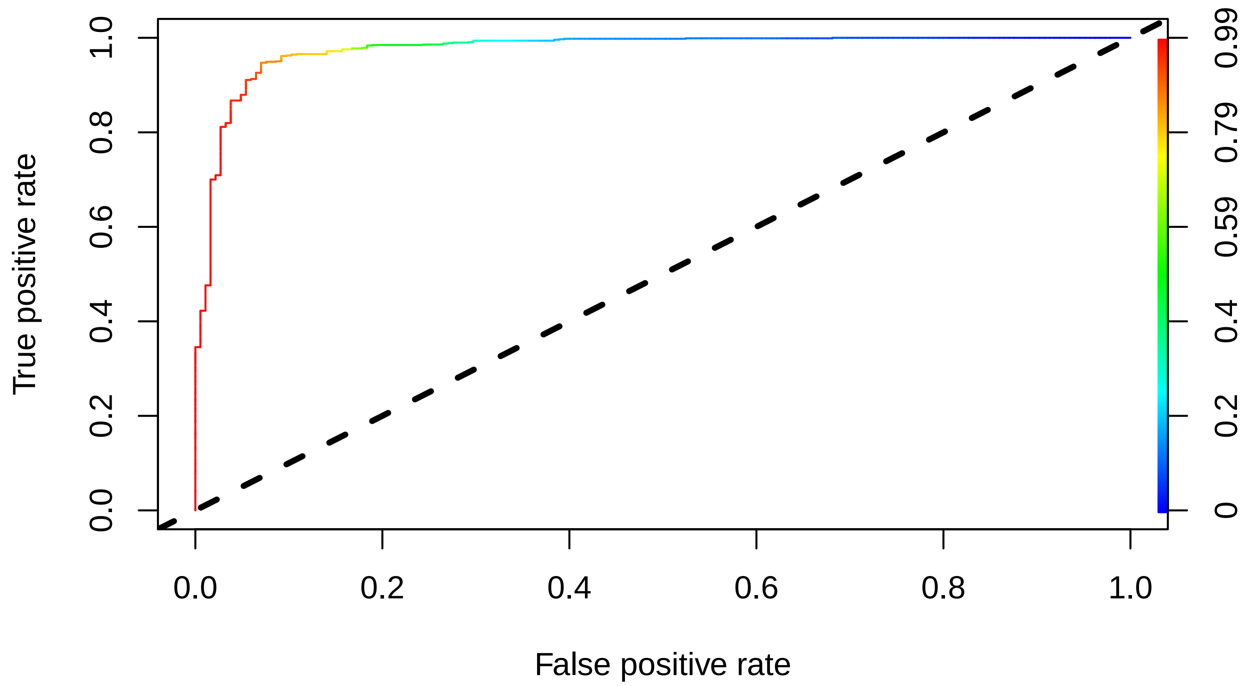
```
#Boosting to predict on test dataset
boost.pred <- predict(boost.model, newdata =ad_test)
table(boost.pred, ad_test$Ad)
```

```
##
## boost.pred ad. nonad.
##     ad.    149     15
##     nonad.  36    972
```

```
mean(boost.pred==ad_test$Ad)
```

```
## [1] 0.9564846
```

```
# Plot ROC and AUC for KNN
#prob got from the predicted model
boost.prob =predict(boost.model, newdata =ad_test, type='prob')

BOOSTPred <- prediction(boost.prob[,2], ad_test$Ad)
BOOSTPerf <- performance(BOOSTPred, "tpr", "fpr")
plot(BOOSTPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```r
#AUC
AUC.BOOST<-performance(BOOSTPred, "auc")

#Corresponding Performance Measures
BOOSTPrediction <- factor(as.factor(boost.pred),
                          c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
BOOSTActual <- factor(as.factor(ad_test$Ad), c('nonad.', 'ad.'),
                      labels = c("Not-Ad", "Ad"))

CMBOOST <- confusionMatrix(BOOSTActual, BOOSTPrediction, negative = "Not-Ad" )
DE.BOOST<-diagnosticErrors(CMBOOST)
DE.BOOST
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9564846 0.8054054 0.9848024 0.9085366 0.9642857 5.5917330
## attr(,"negative")
## [1] "Not-Ad"
```

```r
proc.time()-ptm
```

```
##    user  system elapsed
##    0.46    0.00    0.50
```

From the results above, the Boosting model performance better when n.trees = 900. After using the model to make predictions, we can see that it has about 96% accuracy rate, 81% sensitivity and 98.6% specitivity, the ROC also looks pretty good.

Support Vector Machines:linear kernel

```r
ptm<-proc.time()
#SVM classifier:linear kernel
#Linear Kernel
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
svm.Linear <- train(Ad~.,data=ad_train, method = "svmLinear",
                 trControl=ctrl,
                 preProcess = c("center", "scale"),tuneGrid = expand.grid(
                    C = c(0.01,1,10,100,1000)))
# perform cross-validation using tune() to select the best choice of ?? and cost for an SVM
#set.seed(1)
#linear.tune.out=tune(svm,Ad~.,data=ad_train,kernel="linear",
#              ranges=list(cost=c(0.01,1,10,100,1000)))
#summary(linear.tune.out)
#linear.bestmod=linear.tune.out$best.model

time.SVM.L<-proc.time()-ptm
time.SVM.L
```

```r
ptm<-proc.time()
svm.Linear
```

```
## Support Vector Machines with Linear Kernel
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## Pre-processing: centered (135), scaled (135)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   C      ROC        Sens       Spec
##   1e-02  0.9629201  0.7161905  0.9879394
##   1e+00  0.9483720  0.6464286  0.9899495
##   1e+01  0.9479832  0.5276190  0.9939798
##   1e+02  0.9492149  0.4173810  0.9959899
##   1e+03  0.9341419  0.2335714  0.9970000
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

```r
#make predictions using this best model
svm.linear.pred <- predict(svm.Linear, newdata = ad_test)

table(svm.linear.pred, Y_test)
```

```
##                  Y_test
## svm.linear.pred ad. nonad.
```

```
##            ad.    146     11
##            nonad.  39    976
```

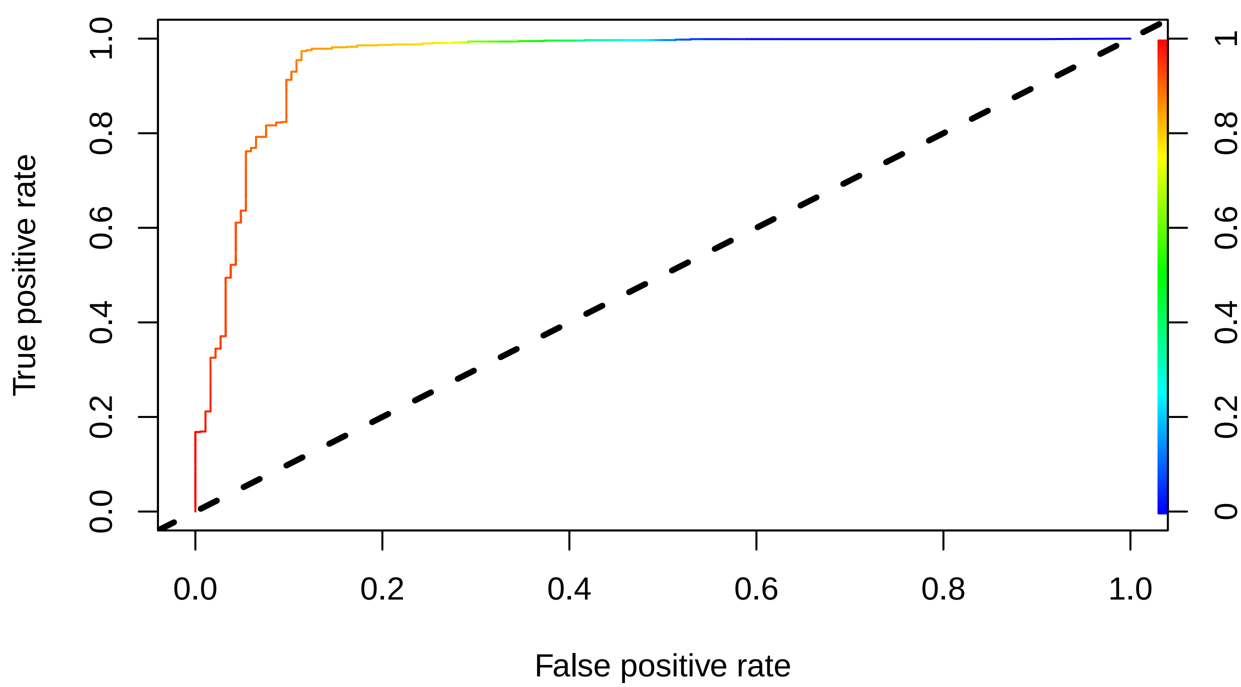
```r
mean(svm.linear.pred==Y_test)
```

```
## [1] 0.9573379
```

```r
# Plot ROC and AUC for SVM
svm.linear.fit=svm(Ad~., data=ad_train, kernel="linear", cost=10, probability=TRUE)

svm.linear.prob <- predict(svm.linear.fit, newdata = ad_test, probability=TRUE)
head(attr(svm.linear.prob, "probabilities"))
```

```
##          ad.       nonad.
## 2 0.9921798 0.0078201966
## 4 0.9993858 0.0006142223
## 5 0.9985081 0.0014919137
## 7 0.9996074 0.0003925931
## 8 0.9997591 0.0002408568
## 9 0.9995652 0.0004348010
```

```r
SVMPred <- prediction(attr(svm.linear.prob, "probabilities")[,2], Y_test)
SVMPerf <- performance(SVMPred, "tpr", "fpr")
plot(SVMPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```r
#AUC
AUC.SVM.L<-performance(SVMPred, "auc")
```

```r
#Corresponding Performance Measures
SVMPrediction <- factor(as.factor(svm.linear.pred),
                        c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
SVMActual <- factor(as.factor(Y_test), c('nonad.', 'ad.'),
                    labels = c("Not-Ad", "Ad"))

CMSVM.L<- confusionMatrix(SVMActual, SVMPrediction, negative = "Not-Ad" )
DE.SVM.L<-diagnosticErrors(CMSVM.L)
DE.SVM.L
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9573379 0.7891892 0.9888551 0.9299363 0.9615764 5.8056123
## attr(,"negative")
## [1] "Not-Ad"
```

```r
proc.time()-ptm
```

```
##    user  system elapsed
##    3.96    0.04    4.04
```

From the results above, the Support Vector Machines with linear kernel perform better when C = 0.01. After using the model to make predictions, we can see that it has about 96% accuracy rate, 82% sensitivity and 98.9% specitivity, the ROC also looks pretty good.

Support Vector Machines:radial kernel

```r
ptm<-proc.time()
#SVM classifier:gausssian kernel
set.seed(123)
ctrl <- trainControl(method = "cv",  number=10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)
svm.Gaussian <- train(Ad~.,data=ad_train, method = "svmRadial",
                 trControl=ctrl,preProcess = c("center", "scale"),
                 tuneGrid = expand.grid(sigma=c(0.5,1,2,3,4),
                    C = c(0.1,1,10,100,1000)))
## perform cross-validation using tune() to select the best choice of ?? and cost for an SVM
#set.seed(1)
#gaussian.tune.out=tune(svm, Ad~., data=ad_train, kernel="radial",
#               ranges=list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))
#summary(gaussian.tune.out)
#gaussian.bestmod=gaussian.tune.out$best.model
time.SVM.G<-proc.time()-ptm
time.SVM.G
```

```r
ptm<-proc.time()
svm.Gaussian
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
```

```
##
## Pre-processing: centered (135), scaled (135)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   sigma  C      ROC        Sens       Spec
##   0.5    1e-01  0.9387875  0.4376190  0.9899192
##   0.5    1e+00  0.9381558  0.4376190  0.9899192
##   0.5    1e+01  0.9353611  0.3978571  0.9869091
##   0.5    1e+02  0.9273689  0.3678571  0.9889192
##   0.5    1e+03  0.9251511  0.3428571  0.9889293
##   1.0    1e-01  0.9345572  0.3776190  0.9899192
##   1.0    1e+00  0.9355176  0.3526190  0.9899192
##   1.0    1e+01  0.9299012  0.3528571  0.9869293
##   1.0    1e+02  0.9277395  0.3078571  0.9919495
##   1.0    1e+03  0.9194677  0.3030952  0.9909394
##   2.0    1e-01  0.9247958  0.3078571  0.9939495
##   2.0    1e+00  0.9253276  0.3128571  0.9949596
##   2.0    1e+01  0.9218854  0.2930952  0.9919596
##   2.0    1e+02  0.9203259  0.2830952  0.9929495
##   2.0    1e+03  0.9164244  0.2780952  0.9929495
##   3.0    1e-01  0.9158031  0.2730952  0.9959697
##   3.0    1e+00  0.9180513  0.2730952  0.9959697
##   3.0    1e+01  0.9150402  0.2780952  0.9929596
##   3.0    1e+02  0.9151563  0.2580952  0.9949596
##   3.0    1e+03  0.9131769  0.2730952  0.9939495
##   4.0    1e-01  0.9101443  0.2630952  0.9959697
##   4.0    1e+00  0.9104137  0.2580952  0.9959697
##   4.0    1e+01  0.9079314  0.2630952  0.9929596
##   4.0    1e+02  0.9080534  0.2580952  0.9949596
##   4.0    1e+03  0.9081278  0.2580952  0.9949596
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.5 and C = 0.1.
```

```r
#make predictions using this best model
svm.gaussian.pred <- predict(svm.Gaussian, newdata = ad_test)

table(svm.gaussian.pred, Y_test)
```

```
##                  Y_test
## svm.gaussian.pred ad. nonad.
##            ad.     83     12
##            nonad. 102    975
```

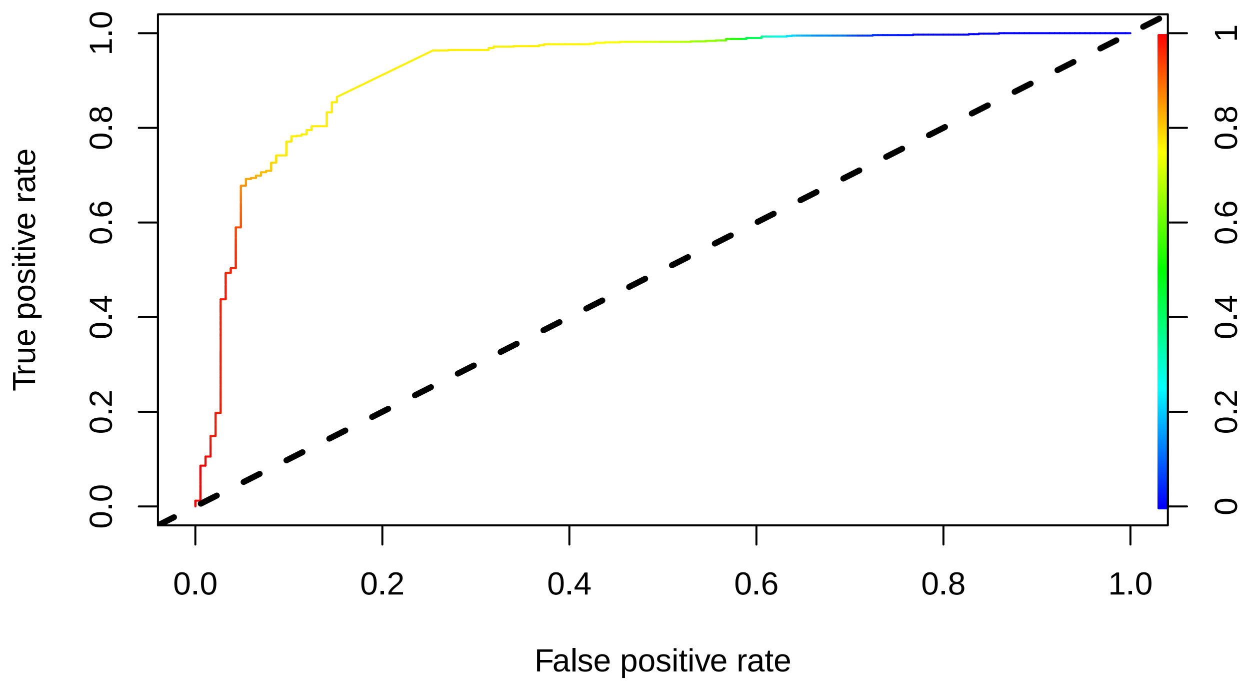
```r
mean(svm.gaussian.pred==Y_test)
```

```
## [1] 0.9027304
```

```r
# Plot ROC and AUC for SVM
svm.gaussian.fit=svm(Ad~., data=ad_train, kernel="radial", cost=10,gamma=0.5, probability=TRUE)

svm.gaussian.prob <- predict(svm.gaussian.fit, newdata = ad_test, probability=TRUE)
head(attr(svm.linear.prob, "probabilities"))
```

```
##           ad.        nonad.
## 2 0.9921798 0.0078201966
## 4 0.9993858 0.0006142223
## 5 0.9985081 0.0014919137
## 7 0.9996074 0.0003925931
## 8 0.9997591 0.0002408568
## 9 0.9995652 0.0004348010
```

```r
SVMPred.g <- prediction(attr(svm.gaussian.prob, "probabilities")[,2], Y_test)
SVMPerf.g <- performance(SVMPred.g, "tpr", "fpr")
plot(SVMPerf.g, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```




```r
#AUC
AUC.SVM.G<-performance(SVMPred.g, "auc")

#Corresponding Performance Measures
SVMPrediction.g <- factor(as.factor(svm.gaussian.pred),
                          c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
SVMActual.g <- factor(as.factor(Y_test), c('nonad.', 'ad.'),
                      labels = c("Not-Ad", "Ad"))

CMSVM.G <- confusionMatrix(SVMActual.g, SVMPrediction.g, negative = "Not-Ad" )
DE.SVM.G<-diagnosticErrors(CMSVM.G)
DE.SVM.G
```

```
##        acc       sens       spec        ppv        npv        lor
```

```
## 0.9027304 0.4486486 0.9878419 0.8736842 0.9052925 4.1913986
## attr(,"negative")
## [1] "Not-Ad"
```

```
proc.time()-ptm
```

```
##    user  system elapsed
##    3.72    0.00    3.72
```

From the results above, the Support Vector Machines with gaussian kernel perform better when sigma = 0.5 and C = 1. After using the model to make predictions, we can see that it has about 90% accuracy rate, 44% sensitivity and 98.9% specitivity, the ROC looks OK.

Neural Network

```
ptm<-proc.time()

set.seed(123)
nnctrl <- trainControl(method = 'cv', number = 10,savePredictions = TRUE,
                       classProbs = TRUE,  summaryFunction = twoClassSummary)
#Neural Network

NN.fit <- train(Ad ~., data = ad_train, method = 'nnet',
                preProcess = c('center', 'scale'), trControl = nnctrl,
                paramGrid=expand.grid(decay = c(0.5, 0.1), size = c(5, 6, 7)))

time.NN<-proc.time()-ptm
time.NN
```

```
ptm<-proc.time()
NN.fit
```

```
## Neural Network
##
## 1197 samples
##  135 predictor
##    2 classes: 'ad.', 'nonad.'
##
## Pre-processing: centered (135), scaled (135)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1077, 1077, 1077, 1077, 1077, 1078, ...
## Resampling results across tuning parameters:
##
##   size  decay  ROC        Sens       Spec
##   1     0e+00  0.9084171  0.8754762  0.8774949
##   1     1e-04  0.9091419  0.8702381  0.8915354
##   1     1e-01  0.9647815  0.8459524  0.9628081
##   3     0e+00  0.9377421  0.8059524  0.9547879
##   3     1e-04  0.9406747  0.8359524  0.9447273
##   3     1e-01  0.9714283  0.8357143  0.9738485
##   5     0e+00  0.9389428  0.8164286  0.9598182
##   5     1e-04  0.9435291  0.7957143  0.9598081
##   5     1e-01  0.9737821  0.8359524  0.9718485
##
## ROC was used to select the optimal model using the largest value.
```

```
## The final values used for the model were size = 5 and decay = 0.1.
```

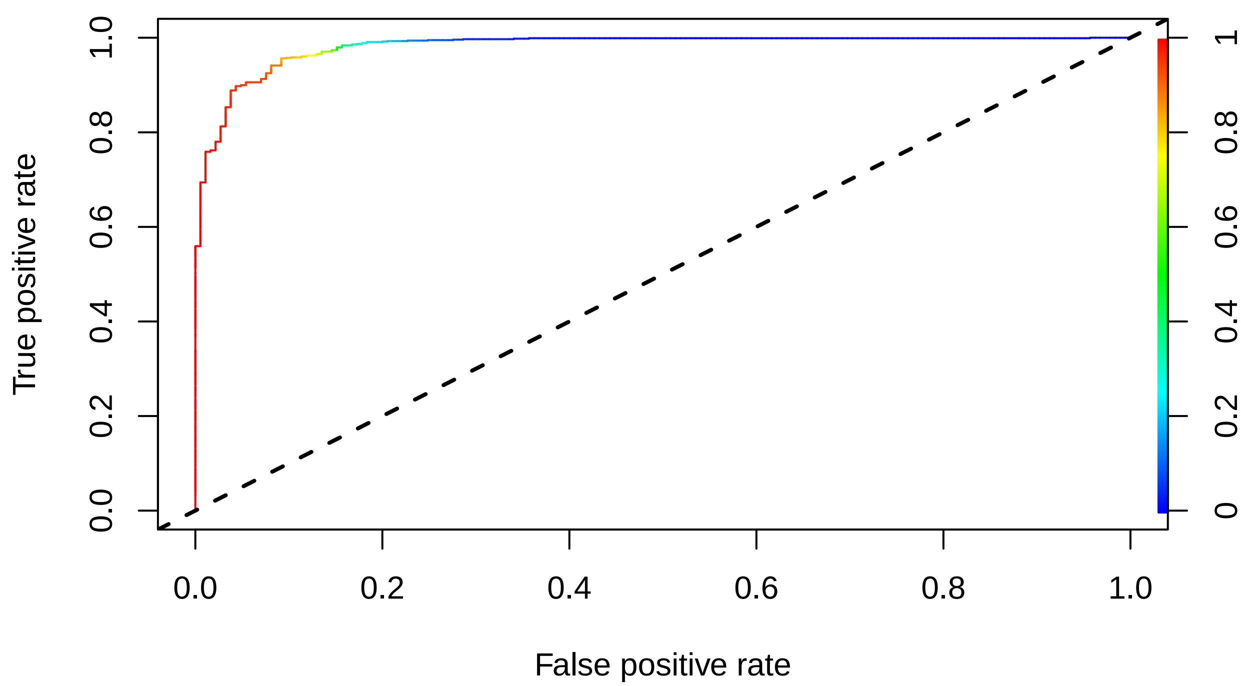
```r
#make prediction
NN.pred <- predict(NN.fit, newdata=ad_test)

table(NN.pred, Y_test)
```

```
##          Y_test
## NN.pred   ad. nonad.
##   ad.     157     22
##   nonad.   28    965
```

```r
mean(NN.pred==Y_test)
```

```
## [1] 0.9573379
```

```r
# Plot ROC and AUC for KNN
#prob got from the predicted model
NN.probs <- predict(NN.fit, newdata=ad_test, type='prob')

NNPred <- prediction(NN.probs[,2], ad_test$Ad)
NNPerf <- performance(NNPred, "tpr", "fpr")
plot(NNPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=2, col="black")
```




```r
#AUC
AUC.NN<-performance(NNPred, "auc")

#Corresponding Performance Measures
```

```r
NNPrediction <- factor(as.factor(NN.pred), c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))
NNActual <- factor(as.factor(ad_test$Ad), c('nonad.', 'ad.'), labels = c("Not-Ad", "Ad"))

CMNN <- confusionMatrix(NNActual, NNPrediction, negative = "Not-Ad" )
DE.NN<-diagnosticErrors(CMNN)
DE.NN
```

```
##       acc      sens      spec       ppv       npv       lor
## 0.9573379 0.8486486 0.9777102 0.8770950 0.9718026 5.5051269
## attr(,"negative")
## [1] "Not-Ad"
```

```r
proc.time()-ptm
```

```
##    user  system elapsed
##    0.48    0.00    0.48
```

From the results above, the Neural Network model perform better when size = 3 and decay = 0.1. After using the model to make predictions, we can see that it has about 96% accuracy rate, 84% sensitivity and 98.3% specitivity, the ROC looks very good.

Summary of Performance Measures For All Models

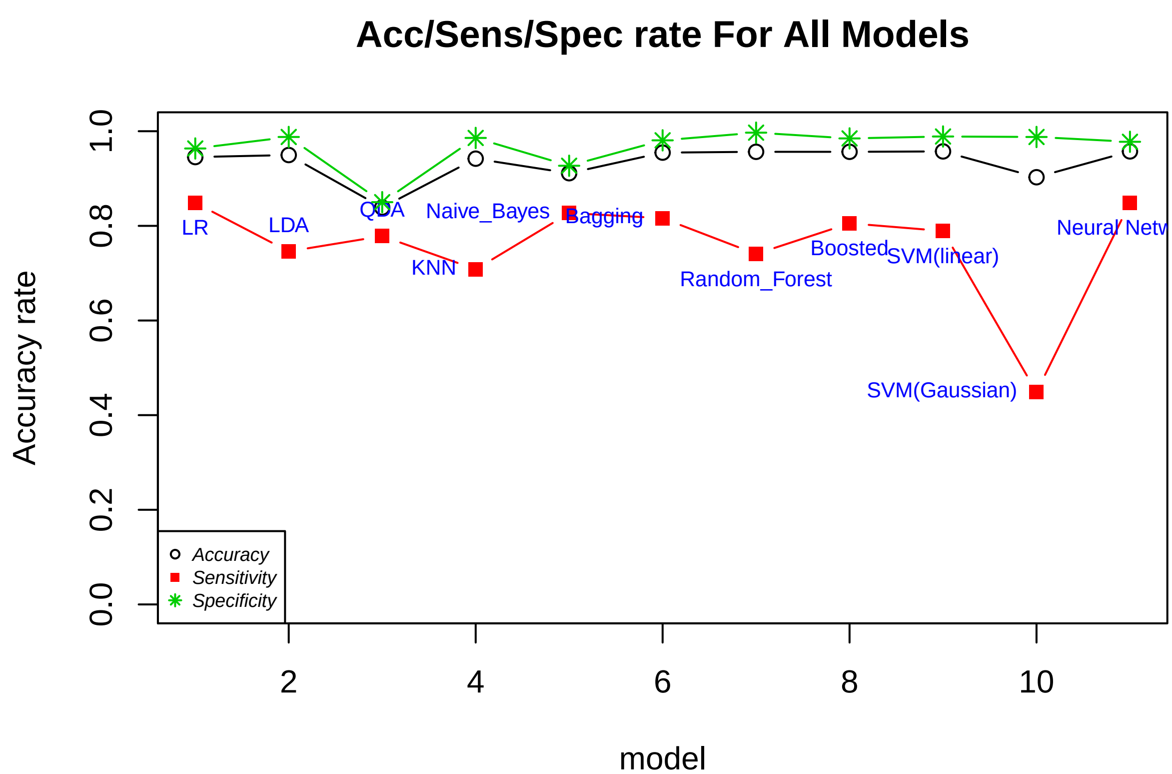
```r
ptm<-proc.time()
# prediction accuracy
DiagnosticErrors <- rbind(DE.LR,DE.LDA,DE.QDA,DE.KNN,DE.NB,
                  DE.BAG,DE.RF,DE.BOOST,DE.SVM.L,DE.SVM.G,DE.NN)
rownames(DiagnosticErrors) <- (c("LR" ,     "LDA" , "QDA" ,
                                 "KNN" ,    "Naive_Bayes" ,     "Bagging" ,
                                 "Random_Forest" ,  "Boosted" ,     "SVM(linear)" ,
                                 "SVM(Gaussian)",'Neural Network'))
colnames(DiagnosticErrors)<-c('Accuracy','Sensitivity','Specificity',
                              'PPV','NPV','Log-odds Ratio')
DiagnosticErrors<-DiagnosticErrors[,-6]
round(DiagnosticErrors, 4)
```

```
##                Accuracy Sensitivity Specificity    PPV    NPV
## LR               0.9454      0.8486      0.9635 0.8135 0.9714
## LDA              0.9497      0.7459      0.9878 0.9200 0.9540
## QDA              0.8387      0.7784      0.8501 0.4932 0.9534
## KNN              0.9420      0.7081      0.9858 0.9034 0.9474
## Naive_Bayes      0.9113      0.8270      0.9271 0.6800 0.9662
## Bagging          0.9548      0.8162      0.9807 0.8882 0.9661
## Random_Forest    0.9565      0.7405      0.9970 0.9786 0.9535
## Boosted          0.9565      0.8054      0.9848 0.9085 0.9643
## SVM(linear)      0.9573      0.7892      0.9889 0.9299 0.9616
## SVM(Gaussian)    0.9027      0.4486      0.9878 0.8737 0.9053
## Neural Network   0.9573      0.8486      0.9777 0.8771 0.9718
```

```r
proc.time() - ptm
```

```
##    user  system elapsed
##    0.01    0.00    0.02
```

```r
#Accuracy
plot(DiagnosticErrors[,1], type='b',col= 1,xlab= 'model',ylab='Accuracy rate',
     ylim=c(0,1),main='Acc/Sens/Spec rate For All Models')
#text(DiagnosticErrors[,1],pos=c(1,3,3,2,2,2,1,1,1,2,2), row.names(DiagnosticErrors),cex=0.7,col=4)

lines(DiagnosticErrors[,2], pch=15,type='b',col= 2)
text(DiagnosticErrors[,2],pos=c(1,3,3,2,2,2,1,1,1,2,1), row.names(DiagnosticErrors),cex=0.7,col=4)
lines(DiagnosticErrors[,3], pch=8,type='b',col= 3)

legend("bottomleft", legend = c("Accuracy" ,    "Sensitivity" , "Specificity" ),
       text.font =3,cex=0.6,col =c(1,2,3),pch = c(1,15,8), xjust = 1, yjust = 1)
```




```r
proc.time()-ptm
```

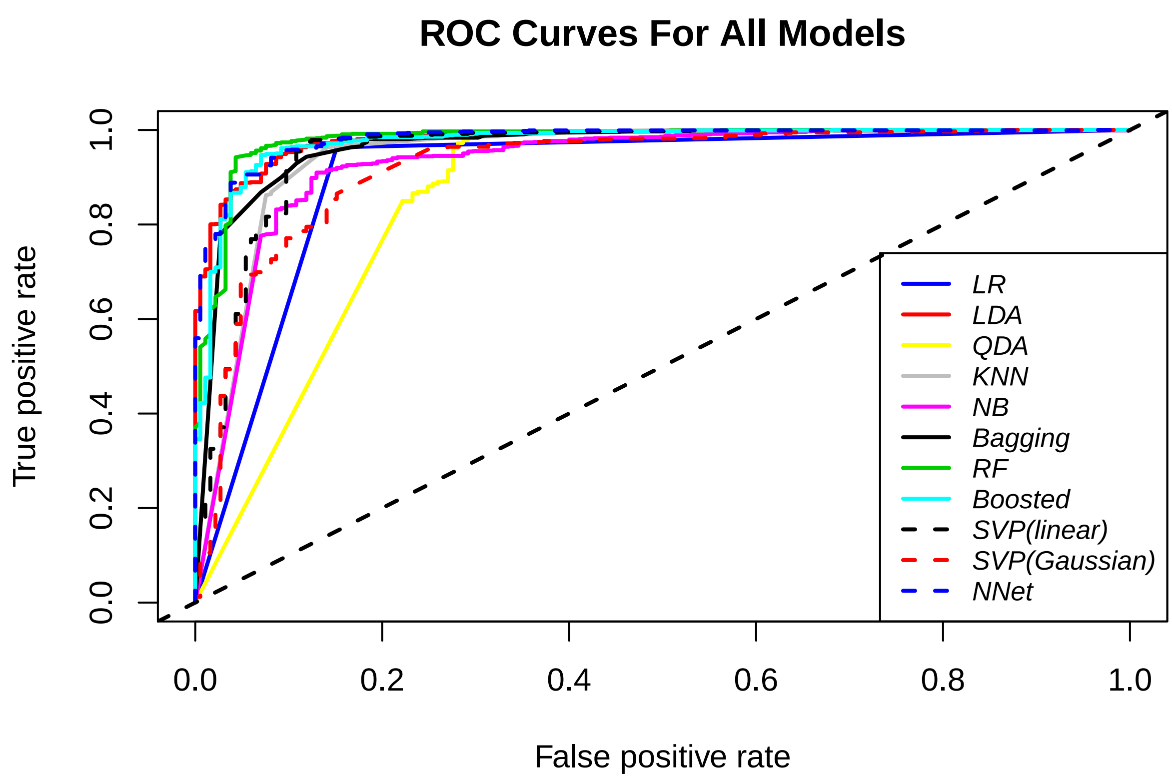
```
##    user  system elapsed
##    0.03    0.00    0.03
```

From the plot above, we can see that all models have pretty high accuracy rate and specificity rate, while the accuracy rate odf the QDA model is the lowest. The sencitivity rate of the Support Vector Machine using Gaussianl kernel is the lowest, which means the model is not good at identify true potisitve (Ad).

ROC Curves For All Models

```r
ptm<-proc.time()
plot(LRPerf, col=4,lwd=2,main='ROC Curves For All Models')
```

```
plot(LDA_Perf,add=TRUE, col=2, lwd=2)
plot(QDA_Perf,add=TRUE, col=7, lwd=2)
plot(KNNPerf,add=TRUE, col=8, lwd=2)
plot(NBPerf,add=TRUE, col=6, lwd=2)
plot(BAGPerf,add=TRUE, col=1, lwd=2)
plot(RFPerf,add=TRUE, col=3, lwd=2)
plot(BOOSTPerf,add=TRUE, col=5, lwd=2)
plot(SVMPerf,add=TRUE, col=1, lty=2, lwd=2)
plot(SVMPerf.g,add=TRUE, col=2,lty=2, lwd=2)
plot(NNPerf,add=TRUE, col=4,lty=2, lwd=2)
abline(a=0, b=1, lty=2, lwd=2, col="black")
legend("bottomright", legend = c("LR" ,     "LDA" , "QDA" , "KNN" , "NB" ,
                                 "Bagging" ,    "RF" ,  "Boosted" ,     "SVP(linear)",
                                 "SVP(Gaussian)",'NNet'), lwd=2,text.font =3,
       cex=0.8,col =c(4,2,7,8,6,1,3,5,1,2,4),
       lty = c(1,1,1,1,1,1,1,1,2,2,2), xjust = 1, yjust = 1)
```




```
proc.time()-ptm
```

```
##    user  system elapsed
##    0.04    0.00    0.04
```

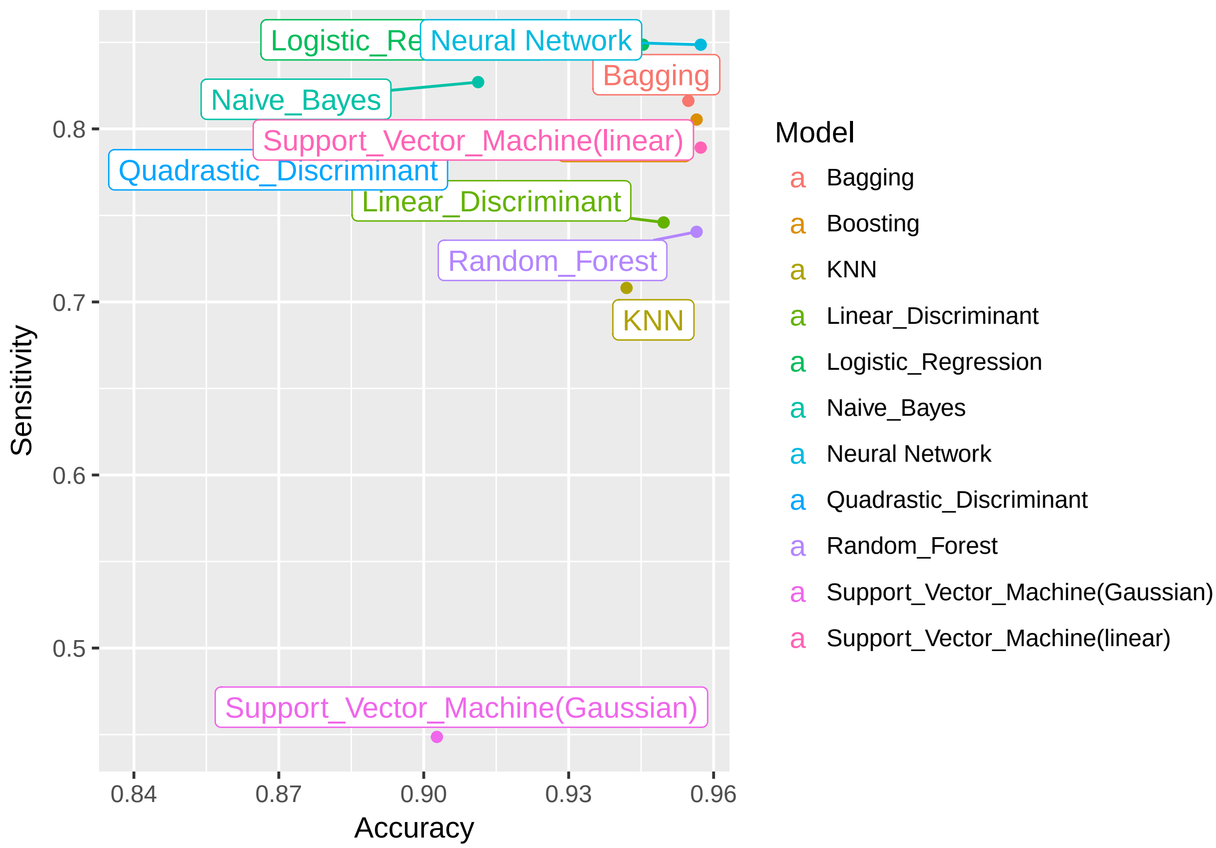
ROC curves of the models show that Neural network,Random Forest, LDA and Bagging outperform other models.

```
#Accuracy
Accuracy <- c(DE.LR[1], DE.LDA[1],  DE.QDA[1],  DE.KNN[1],
```

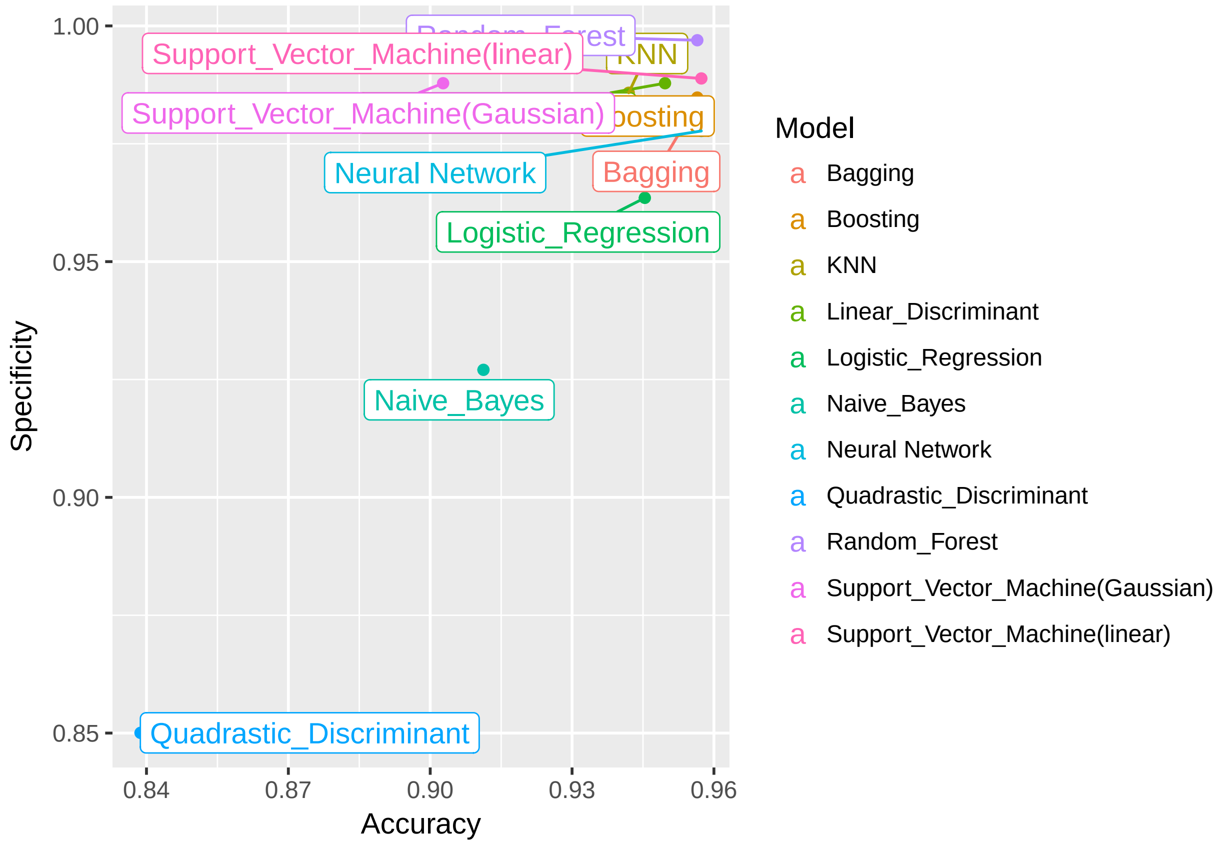
```r
              DE.NB[1], DE.BAG[1], DE.RF[1],    DE.BOOST[1],
              DE.SVM.L[1],  DE.SVM.G[1],DE.NN[1])
#Sensitivity
Sensitivity <- c(DE.LR[2],  DE.LDA[2],  DE.QDA[2],  DE.KNN[2],
                 DE.NB[2],  DE.BAG[2], DE.RF[2],    DE.BOOST[2],
                 DE.SVM.L[2],   DE.SVM.G[2],DE.NN[2])
#Specificity
Specificity <- c(DE.LR[3],  DE.LDA[3],  DE.QDA[3],  DE.KNN[3],
                 DE.NB[3],  DE.BAG[3], DE.RF[3],    DE.BOOST[3],
                 DE.SVM.L[3],   DE.SVM.G[3],DE.NN[3])

#Positive predicted values
PPV <- c(DE.LR[4],  DE.LDA[4],  DE.QDA[4],  DE.KNN[4],
         DE.NB[4],  DE.BAG[4], DE.RF[4],    DE.BOOST[4],
         DE.SVM.L[4],   DE.SVM.G[4],DE.NN[4])

Model<- c("Logistic_Regression" ,   "Linear_Discriminant" ,
          "Quadrastic_Discriminant" ,   "KNN" , "Naive_Bayes" ,
          "Bagging" ,   "Random_Forest" ,   "Boosting" ,
          "Support_Vector_Machine(linear)" ,
          "Support_Vector_Machine(Gaussian)",'Neural Network')

## PLoting Accuracy, Sensitivity and,Specificity  from all models
df1 <- data.frame(col1=Accuracy, col2= Sensitivity,col3= Model)
df2 <- data.frame(col1=Accuracy, col2= Specificity , col3= Model)

Sys.time()
```

```
## [1] "2018-03-04 22:36:15 PST"
```

```r
#Accuracy vs Sensitivity
ggplot(df1, aes(x=Accuracy, y=Sensitivity, color = Model , label = Model )) +
  ##geom_point(aes(size=17.5))+
  geom_point() +geom_label_repel(aes(label=Model))
```




```
#Accuracy vs Specificity
ggplot(df2, aes(x=Accuracy, y=Specificity, color = Model , label = Model )) +
  ##geom_point(aes(size=17.5))+
  geom_point() +geom_label_repel(aes(label=Model))
```



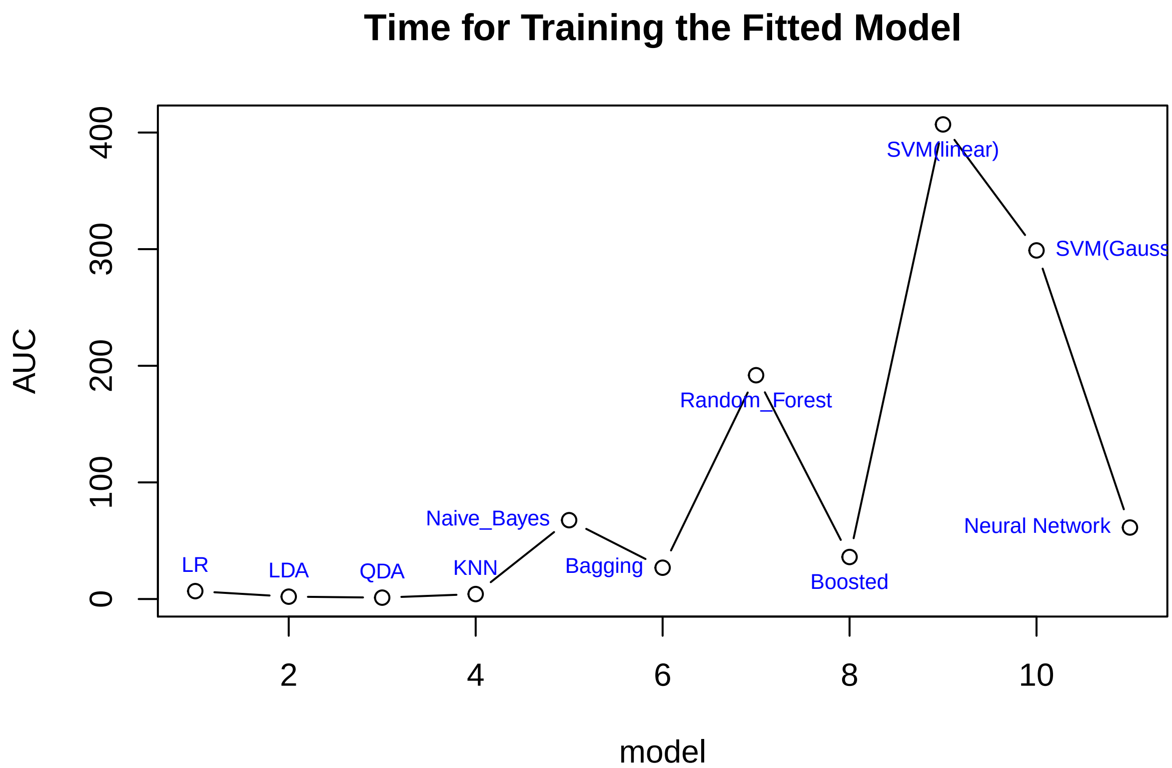
Both the 'Accuracy vs Sensitivity' and 'Accuracy vs Specificity' show that most of the models perform pretty well at predicting the internet ads, especially Neural Network, Support Vector Machine with linear kernel, and Boosting.

Time for Training the Fitted Model

```
ptm<-proc.time()
# prediction accuracy
train.time <- rbind(time.LR,time.LDA,time.QDA,time.KNN,time.NB,
                    time.BAG,time.RF,time.BOOST,time.SVM.L,time.SVM.G,time.NN)
rownames(train.time) <- (c("LR" ,   "LDA" , "QDA" , "KNN" ,
                           "Naive_Bayes" ,  "Bagging" ,     "Random_Forest" ,
                           "Boosted" ,  "SVM(linear)" , "SVM(Gaussian)",
                           'Neural Network'))
proc.time() - ptm
```

```
##    user  system elapsed
##    0.01    0.00    0.02
```

```
train.time <- train.time[,c(1:3)]
#Accuracy
plot(train.time[,3], type='b',col= 1,xlab= 'model',ylab='AUC',
     main='Time for Training the Fitted Model')
text(train.time[,3],pos=c(3,3,3,3,2,2,1,1,1,4,2),row.names(train.time),cex=0.7,col=4)
```




```
proc.time()-ptm
```

```
##    user  system elapsed
##    0.01    0.00    0.02
```

The plot above shows the time used to train each model, SVM(linear) is the most computational expensive, models like SVM(gaussian), random forest and neural network are also more computational expensive than other models.