ISEN 613 final.R

Pulkit Jain, UIN 625006181

```
# I
# Logistic Regression, LDA, QDA, KNN, Trees, Pruning of Trees, Bagging, Rando
m Forests, Boosting, SVM (linear, polynomial and radial)

# II
# The four measure to compare the performance are: Accuracy=(TN+TP)/(N+P); #E
rror Rate= 1-Accuracy; Sensitivity = TP/P & Specificity = TN/N.

# III

library(tree)
library(class)
library(boot)
library(e1071)
library(randomForest)
library(gbm)
library(MASS)

# read and split data
data1 <- read.csv("C:/Users/ISEN__/613 EnggDataAnalysis/Final/Dataset1.csv")
data2 <- read.csv("C:/Users/ISEN__/613 EnggDataAnalysis/Final/Dataset2.csv")
data1$quality <- factor(data1$quality)
summary(data1$quality)

##   0   1
## 686 171

set.seed(1004)
train <- sort(sample(nrow(data1), 600, replace = F))

data1_train <- data1[train,]
data1_test <- data1[-train,]

# III.1 logistic regression

logistic.fit1 <- glm(quality~., data=data1_train, family=binomial)
summary(logistic.fit1)

##
## Call:
## glm(formula = quality ~ ., family = binomial, data = data1_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.2662  -0.5969  -0.3401  -0.1136   2.5494
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.673e+02  2.592e+02   2.188  0.02864 *
## fix          4.329e-01  2.494e-01   1.736  0.08256 .
## vol         -4.694e+00  1.351e+00  -3.475  0.00051 ***
## cit          6.148e-01  1.187e+00   0.518  0.60439
## res          1.932e-01  1.005e-01   1.923  0.05446 .
## chl         -1.142e+01  1.011e+01  -1.130  0.25859
## fre          2.440e-02  1.102e-02   2.215  0.02677 *
## tot         -8.115e-03  4.711e-03  -1.723  0.08497 .
## den         -5.892e+02  2.620e+02  -2.249  0.02452 *
## pH           3.586e+00  1.227e+00   2.923  0.00347 **
## sul          3.503e+00  1.187e+00   2.951  0.00317 **
## alc          2.008e-01  3.078e-01   0.653  0.51401
## colwhite    -5.941e-01  8.684e-01  -0.684  0.49387
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 624.61  on 599  degrees of freedom
## Residual deviance: 462.14  on 587  degrees of freedom
## AIC: 488.14
##
## Number of Fisher Scoring iterations: 6
```

```
# citric acid, alcohol and color don't appear to be affecting quality
# fixed acidity, residual sugar & total sulfur dioxide are mild influencers
# sulphates, pH and volatile acidity do appear to influencing the model

logistic.probs1 <- predict(logistic.fit1, data1_test, type = "response")
logistic.pred1 <- rep(0, 257)
logistic.pred1[logistic.probs1>0.5] <- 1
table(Predicted = logistic.pred1, Actual = data1_test$quality)
```

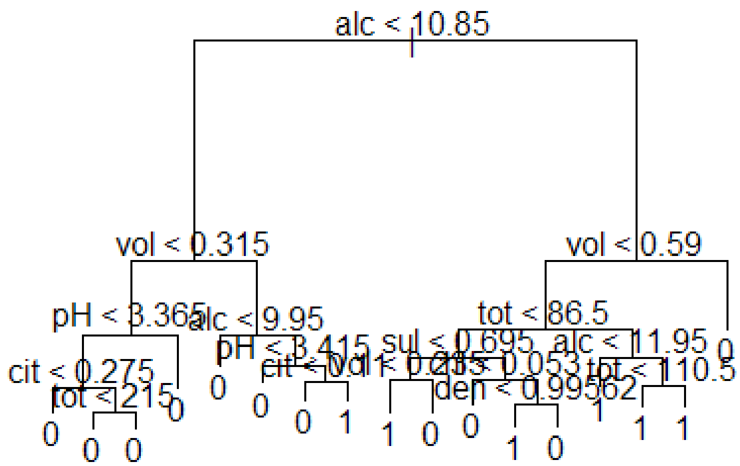
```
##          Actual
## Predicted   0   1
##         0 202  25
##         1  13  17
```

```
mean(logistic.pred1 == data1_test$quality)
```

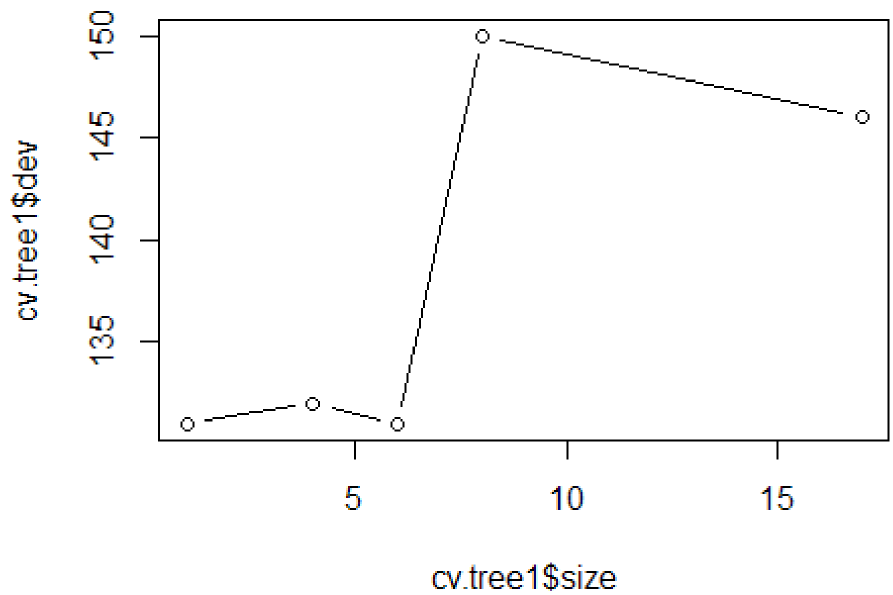
```
## [1] 0.8521401
```

```
# accuracy = 0.8521, error = .1479, sen = .4048, spec = .9395

# III.2 tree

tree1 <- tree(quality~., data1_train)
```

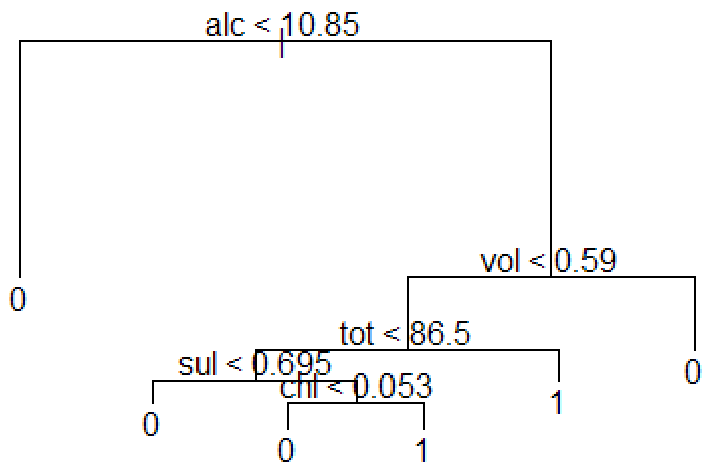
```r
plot(tree1)
text(tree1, pretty=0)
```




```r
tree.pred1 <- predict(tree1, data1_test, type="class")
table(Predicted = tree.pred1, Actual = data1_test$quality)

##           Actual
## Predicted   0   1
##         0 192  24
##         1  23  18

mean(tree.pred1 == data1_test$quality)

## [1] 0.8171206

# accuracy = 0.8171, error = 0.1829, sen = 0.4285, spec = 0.8930

# tree pruning

set.seed(1004)
cv.tree1 <- cv.tree(tree1, FUN= prune.misclass)
plot(cv.tree1$size, cv.tree1$dev, type= "b")
```

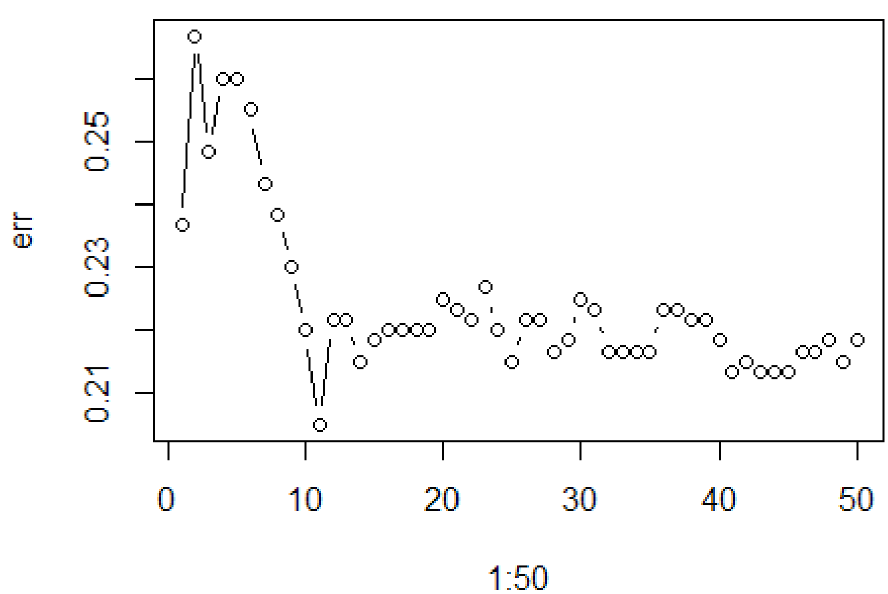



```
cv.tree1$dev

## [1] 146 150 131 132 131

cv.tree1$size

## [1] 17  8  6  4  1

min(cv.tree1$dev)

## [1] 131

best_tree_size <- cv.tree1$size[which.min(cv.tree1$dev)]
best_tree_size

## [1] 6

prune1 <- prune.misclass(tree1, best = best_tree_size)
plot(prune1)
text(prune1, pretty= 0)
```




```
# the first split is made on alcohol (shows its importance),
# the other variables are volatile acidity, total sulfur dioxide, sulphates a
nd chlorides

# The pruned tree has much lower nodes in it

prune.pred1 <- predict(prune1, data1_test, type="class")
table(Predicted = prune.pred1, Actual = data1_test$quality)

##            Actual
## Predicted   0   1
##         0 186  21
##         1  29  21

mean(prune.pred1 == data1_test$quality)

## [1] 0.8054475

# accuracy = 0.8054, error = 0.1946, sen = 0.50, spec = 0.8651

# III.3

# KNN


# convert white to 0, red to 1 in the variable “col”
```

```r
# train set independent variables
train_var <- data1_train[,-13]
train_var$col <- as.character(train_var$col)
train_var$col[train_var$col == "white"] <- 0
train_var$col[train_var$col == "red"] <- 1

# test set independent variables
test_res <- data1_test[,-13]
test_res$col <- as.character(test_res$col)
test_res$col[test_res$col == "white"] <- 0
test_res$col[test_res$col == "red"] <- 0

# train set dependent variables
train_res <- data1_train$quality


# Run cross validation to find best K
err <- matrix(0,1,50)
for (i in 1:50){
set.seed(1004)
  results = knn.cv(train_var, k=i, cl = train_res )
  err[i] = 1-mean(results == train_res)
    }

plot(1:50, err, type = "b")
```



```r
best_knn = which.min(err)
best_knn

## [1] 11

# KNN works best when K( no. of nearest neighbours) is 11

knn.pred1 <- knn(train_var, test_res, train_res, k = best_knn)
table(Predicted = knn.pred1, Actual = data1_test$quality)

##          Actual
## Predicted   0   1
##         0 203  37
##         1  12   5

# KNN accuracy equals: (other measures in table below)

mean(knn.pred1 == data1_test$quality)

## [1] 0.8093385

# SVM (for linear)

aa = data.frame(train_var, train_res)
colnames(aa)[13] = "quality"

# Run cross validation to find value of cost with best model

set.seed(1004)
tune.out <- tune(svm, quality~., data = aa, kernel = "linear",
                 ranges = list(cost=c(.001, .01, .1, 1, 5, 10, 100) ))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.2083333
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2150000 0.03722637
## 2 1e-02 0.2150000 0.03722637
## 3 1e-01 0.2133333 0.04700932
## 4 1e+00 0.2116667 0.05273390
## 5 5e+00 0.2116667 0.05446145
```

```
## 6 1e+01 0.2116667 0.05331597
## 7 1e+02 0.2083333 0.05679180
```

```
tune.out$best.parameters
```

```
##   cost
## 7  100
```

```
bestmod = tune.out$best.model

bb = data1_test
bb$col <- as.character(bb$col)
bb$col[bb$col == "white"] <- 0
bb$col[bb$col == "red"] <- 1


ypred = predict(bestmod, bb)

table(Predicted = ypred, bb$quality)
```

```
##
## Predicted   0   1
##         0 204  25
##         1  11  17
```

```
# SVM Linear Accuracy equals: (other measures in table below)

mean(ypred == bb$quality)
```

```
## [1] 0.8599222
```

```
# SVM Radial
# Run cross validation to find values of cost and gamma

set.seed(1004)
tune.out.radial <- tune(svm, quality~., data=aa, kernal = "radial",
                        ranges = list(cost = 10^(seq(-1,3)),
                                      gamma = 0.5*(seq(1,5)) ) )

# A total of 25 combination will be test using above ranges

tune.out.radial$best.parameters
```

```
##    cost gamma
## 17    1     2
```

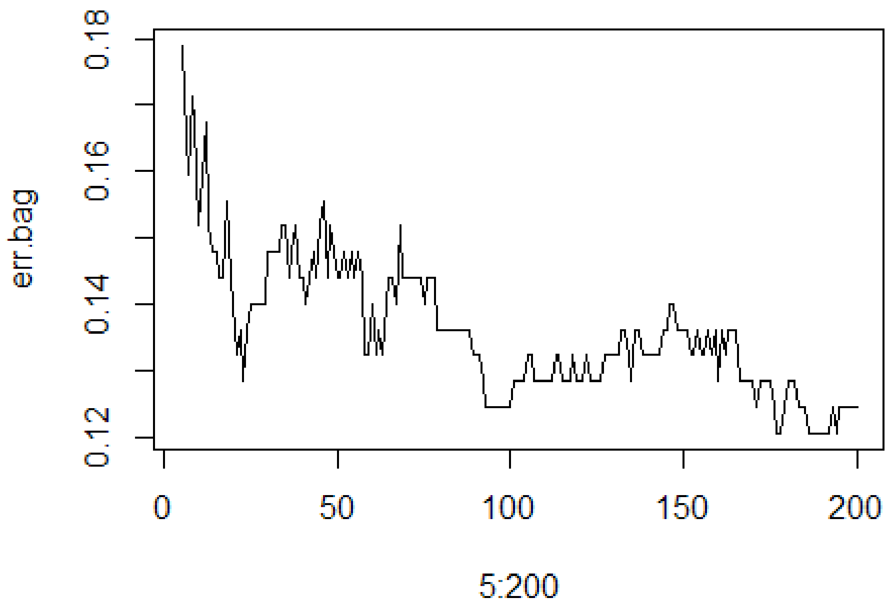
```
tune.out.radial$best.performance
```

```
## [1] 0.1633333
```

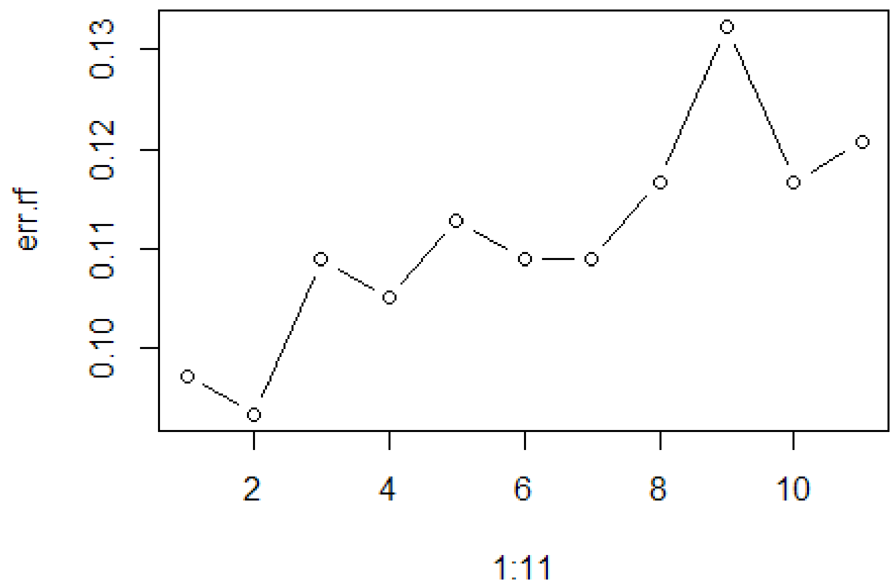
```
bestmod_rad = tune.out.radial$best.model
```

```r
ypred_rad = predict(bestmod_rad, bb)

table(Predicted = ypred_rad,  Actual = bb$quality)

##            Actual
## Predicted   0   1
##         0 214  29
##         1   1  13

# SVM Radial Accuracy equals: (other measures in table below)

mean(ypred_rad == bb$quality)

## [1] 0.8832685

# SVM polynomial

# Run cross validation to find value of cost and degree

set.seed(1004)
tune.out.poly <- tune(svm, quality~., data=aa, kernal = "polynomial",
                      ranges = list(cost = 10^(seq(-1,3)),
                                    degree = c(2,3,4,5,10)) )
# The above ranges will result in a total of 25 combinations

tune.out.poly$best.parameters

##   cost degree
## 3   10      2

tune.out.poly$best.performance

## [1] 0.1883333

bestmod_poly <- tune.out.poly$best.model

ypred_poly = predict(bestmod_poly, bb)

table(Predicted = ypred_poly,  Actual = bb$quality)

##            Actual
## Predicted   0   1
##         0 197  22
##         1  18  20

# SVM Polynomial Accuracy Equals: (other measures in table below)

mean(ypred_rad == bb$quality)

## [1] 0.8832685

# Bagging
```

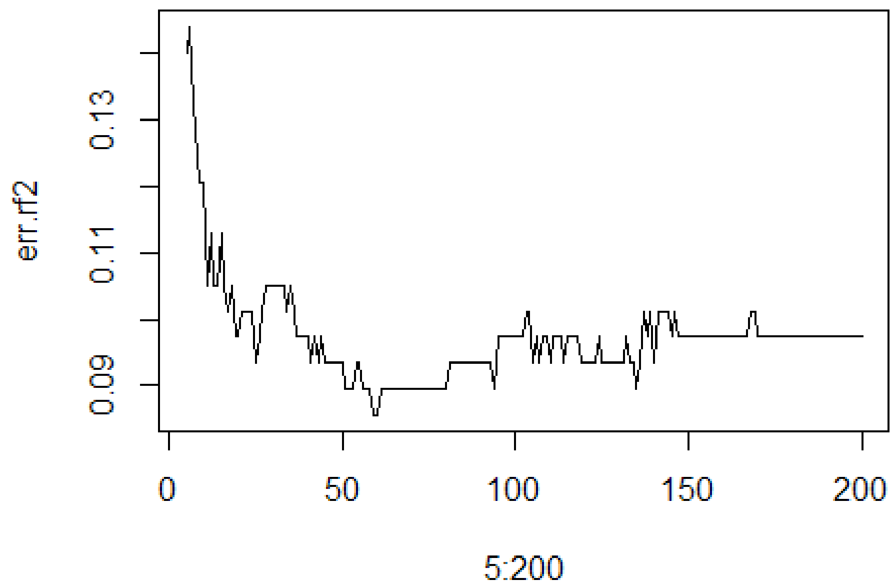
```r
set.seed(1004)
bag.wine <- randomForest(quality~., data=data1_train, mtry = 12, ntree =100,
                         importance=T)
bag.wine

##
## Call:
##  randomForest(formula = quality ~ ., data = data1_train, mtry = 12,      n
tree = 100, importance = T)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 15.17%
## Confusion matrix:
##     0  1 class.error
## 0 438 33  0.07006369
## 1  58 71  0.44961240

yhat.bag <- predict(bag.wine, newdata = data1_test )
table(Predicted = yhat.bag,  Actual = data1_test$quality)

##          Actual
## Predicted   0   1
##         0 200  17
##         1  15  25

# Bagging Accuracy Equals : (other measures in table below)

mean(yhat.bag == data1_test$quality)

## [1] 0.8754864

# Check if the number of trees equals 100 is a viable option

err.bag = matrix(0,1,196)

for(i in 1:196){


  set.seed(1004)
  bag.wine <- randomForest(quality~., data=data1_train, mtry = 12, ntree  =i+
4,  importance=T)

  yhat.bag <- predict(bag.wine, newdata = data1_test )
  table(yhat.bag, data1_test$quality)
  err.bag[i]= 1- mean(yhat.bag == data1_test$quality)
}

plot(5:200, err.bag, type="l")
```




```
# ntree 100 is good enough


# Random Forest


# Find number of predictors that should be used
err.rf = matrix(0,1,11)

for(i in 1:11){

set.seed(1004)
rf.wine <- randomForest(quality~., data=data1_train, mtry = i, ntree =100,
                        importance=T)
yhat.rf <- predict(rf.wine, newdata = data1_test )
table(Predicted = yhat.rf, Actual =  data1_test$quality)
err.rf[i] = 1- mean(yhat.rf == data1_test$quality)
}
plot(1:11, err.rf, type="b")
```




```
which.min(err.rf)

## [1] 2

# a value of 2 is close to default value of sqrt(12)~3

no_pred = which.min(err.rf)

# Check for number of trees
err.rf2 = matrix(0,1,196)

for(i in 1:196){


  set.seed(1004)
  rf.wine <- randomForest(quality~., data=data1_train, mtry = no_pred, ntree
=i+4,  importance=T)

  yhat.rf <- predict(rf.wine, newdata = data1_test )

  err.rf2[i]= 1- mean(yhat.rf == data1_test$quality)
}

plot(5:200, err.rf2, type="l")
```

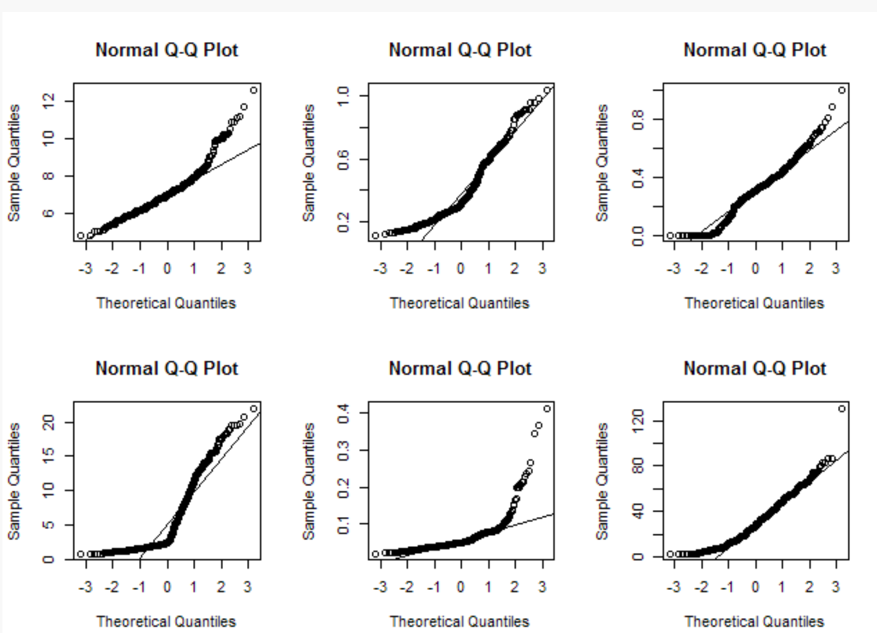



```
# Again ntree 100 is a viable option


# Train using 2 predictors and 100 trees
rf.wine <- randomForest(quality~., data=data1_train, mtry = no_pred,
                        ntree =100,  importance=T)
```

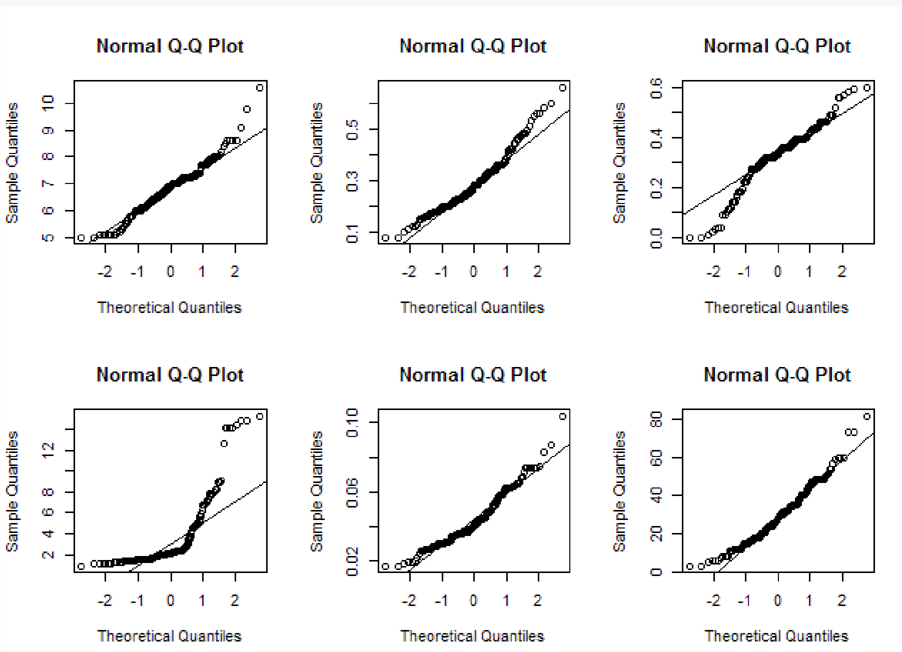
```
yhat.rf <- predict(rf.wine, newdata = data1_test )
# Accuracy of random forests is : (other measures in table below)

table(Predicted = yhat.rf,  Actual = data1_test$quality)

##            Actual
## Predicted   0   1
##          0 205  17
##          1  10  25

mean(yhat.rf == data1_test$quality)

## [1] 0.8949416

# Boosting

set.seed(1004)
boost.wine <- gbm(quality~., data = data1_train, distribution = "multinomial"
                  ,n.trees = 5000, interaction.depth = 4)
```

```r
# multinomial distribution has been used, which is generalized form of binomi
al,due to some error in binomial syntax

yhat.boost <- predict(boost.wine, newdata = data1_test, n.trees = 5000)

yyhat.boost <- apply(yhat.boost, 1, which.max)

yyhat.boost[yyhat.boost==1] = 0
yyhat.boost[yyhat.boost==2] = 1
table(Predicted = yyhat.boost, Actual =  data1_test$quality)

##          Actual
## Predicted   0   1
##         0 200  21
##         1  15  21

# Accuracy after boosting is : (other measures in table below)

mean(yyhat.boost == data1_test$quality)

## [1] 0.8599222

# LDA
lda.fit <- lda(quality~., data=data1_train)
lda.pred <- predict(lda.fit, data1_test)
names(lda.pred)

## [1] "class"     "posterior" "x"

table(Predicted = lda.pred$class,  Actual = data1_test$quality)

##          Actual
## Predicted   0   1
##         0 199  23
##         1  16  19

# Accuracy of LDA is : (other measures in table below)

mean(lda.pred$class == data1_test$quality)

## [1] 0.848249

# qda

qda.fit <- qda(quality~., data=data1_train)
qda.pred <- predict(qda.fit, data1_test)
names(qda.pred)

## [1] "class"     "posterior"

table(Predicted = qda.pred$class,  Actual = data1_test$quality)
```

```
##           Actual
## Predicted   0   1
##         0 161   7
##         1  54  35

# Accuracy of QDA is : (other measures in table below)

mean(qda.pred$class == data1_test$quality)

## [1] 0.7626459

# check normality of independent variables for different classes of response
# when quality is 0, (first 6 variables)
par(mfrow = c(2,3) )
qqnorm(data1$fix[data1$quality==0])
qqline(data1$fix[data1$quality==0])

qqnorm(data1$vol[data1$quality==0])
qqline(data1$vol[data1$quality==0])

qqnorm(data1$cit[data1$quality==0])
qqline(data1$cit[data1$quality==0])

qqnorm(data1$res[data1$quality==0])
qqline(data1$res[data1$quality==0])

qqnorm(data1$chl[data1$quality==0])
qqline(data1$chl[data1$quality==0])

qqnorm(data1$fre[data1$quality==0])
qqline(data1$fre[data1$quality==0])
```



```r
# when quality is 1, (first 6 variables)

par(mfrow = c(2,3) )
qqnorm(data1$fix[data1$quality==1])
qqline(data1$fix[data1$quality==1])

qqnorm(data1$vol[data1$quality==1])
qqline(data1$vol[data1$quality==1])

qqnorm(data1$cit[data1$quality==1])
qqline(data1$cit[data1$quality==1])

qqnorm(data1$res[data1$quality==1])
qqline(data1$res[data1$quality==1])

qqnorm(data1$chl[data1$quality==1])
qqline(data1$chl[data1$quality==1])

qqnorm(data1$fre[data1$quality==1])
qqline(data1$fre[data1$quality==1])
```

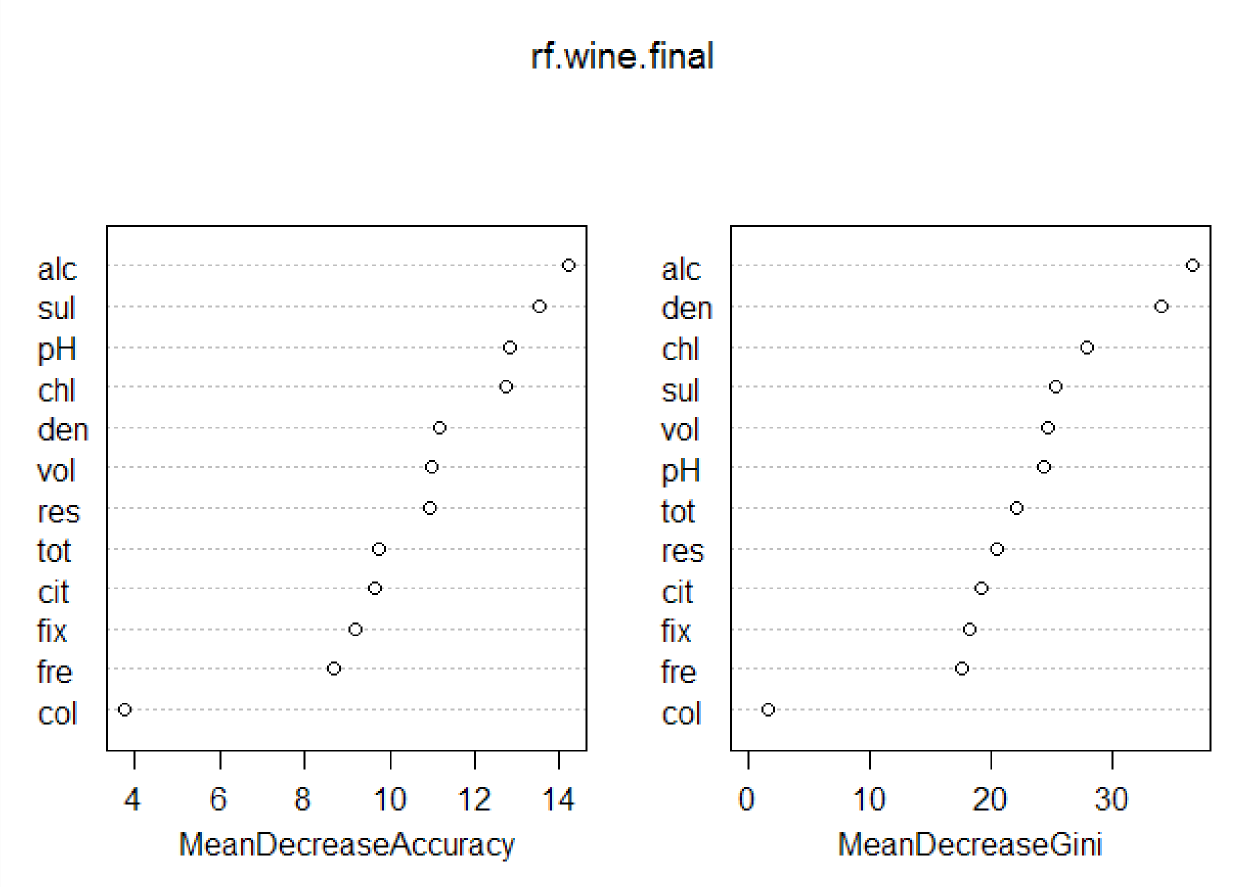



```r
# The normality in data is low, therefore it is not a good idea to use LDA/QD
A

# III.4

# Summary of performance


#
```

S.No	Method	Accuracy	Error	Sensitivity	Specificity
1	Logistic Regression	85.21%	14.79%	40.48%	93.95%
2	Tree	81.71%	18.29%	42.85%	89.30%
3	Prune (Tree size 6)	80.54%	19.46%	50.00%	86.51%
4	KNN (K =11)	80.93%	19.07%	11.90%	94.42%
5	SVM linear	85.99%	14.01%	40.47%	94.88%
6	SVM Radial (Cost 1, gamma 2)	88.32%	11.68%	30.95%	99.53%
7	SVM Poly (Cost 10, Deg 2)	88.32%	11.68%	47.61%	91.62%
8	Bag	87.54%	12.46%	57.14%	93.48%
9	**RF**	**90.27%**	**9.73%**	**64.28%**	**95.34%**
10	Boost	85.99%	14.01%	50.00%	93.02%
11	LDA	84.82%	15.18%	45.23%	92.55%
12	QDA	76.26%	23.74%	83.33%	74.88%

```
# Modified trees, LDA and Support vector classifiers/machines have comparable
accuracy and error. QDA has very high sensitivity but with low accuracy. Spec
ificity if nearly 100% in radial SVM, but it has low sensitivity. LDA perform
s reasonably good, but normality assumption MAY NOT hold. In Random Forests,
for a slight compromise in specificity (than SVM radial) we see high increase
in sensitivity, it has a high accuracy of 90%. Therefore, we choose Random Fo
rests (with 2 predictors and 100 trees) as our final model.

# Same seed has been used throughout so that data can be compared

# III.5 train model on whole data

rf.wine.final <- randomForest(quality~., data=data1, mtry = no_pred
                              , ntree =100, importance=T)
importance(rf.wine.final)

##            0         1 MeanDecreaseAccuracy MeanDecreaseGini
## fix 5.741194  6.271370             8.834741        18.216826
## vol 6.593639 10.653059            10.421294        28.814095
## cit 5.283544  7.140271             8.345603        19.849885
## res 7.675493  9.795416            11.691190        21.889809
## chl 6.470673 10.357985            11.177341        26.047403
## fre 6.637876  7.675269            10.178153        19.024498
## tot 4.862552  8.630136             9.774753        23.395567
## den 7.357501 10.407623            11.899524        29.349598
## pH  9.374407 12.376561            13.291285        25.641016
## sul 6.624613 11.102144            12.544707        23.188373
## alc 7.511942 12.075964            13.424674        37.850148
## col 1.924580  2.780026             3.305905         1.057568
```

```r
varImpPlot(rf.wine.final)
```




```r
# Alcohol is the most important variable in Random Forests

# IV Predict on Dataset2
yhat.rf.final <- predict(rf.wine.final, newdata = data2)
yhat.rf.final

##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## Levels: 0 1

# Observations 3 and 5 have quality 1. Rest are 0.
```