Project 642

Zijing Gao

Goal

Our goal is to identify the glass types by detecting or analyzing the materials or properties of the broken glass in the crime scene to help the police crack the criminal.

Data Preprocessing

```
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(ISLR)
library(class)
library(MASS)
library(splines)

# load the data
glass = read.csv("glass.csv")
colnames(glass) = c("id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe
", "Type")
```

binary classification

At first, we simplify our problem into binary classification problem, since our first goal is to detect whether the glass is float processed or not.

```
binary = function(x){
  if((x==1 | x==3)){
    return(1)
  }else{
    return(2)
  }
}
glass$Type = sapply(glass$Type, binary)
glass$Type = as.factor(glass$Type)

glass.type = glass$Type
glass.id = glass$id
glass = glass[,-1]

head(glass)
```

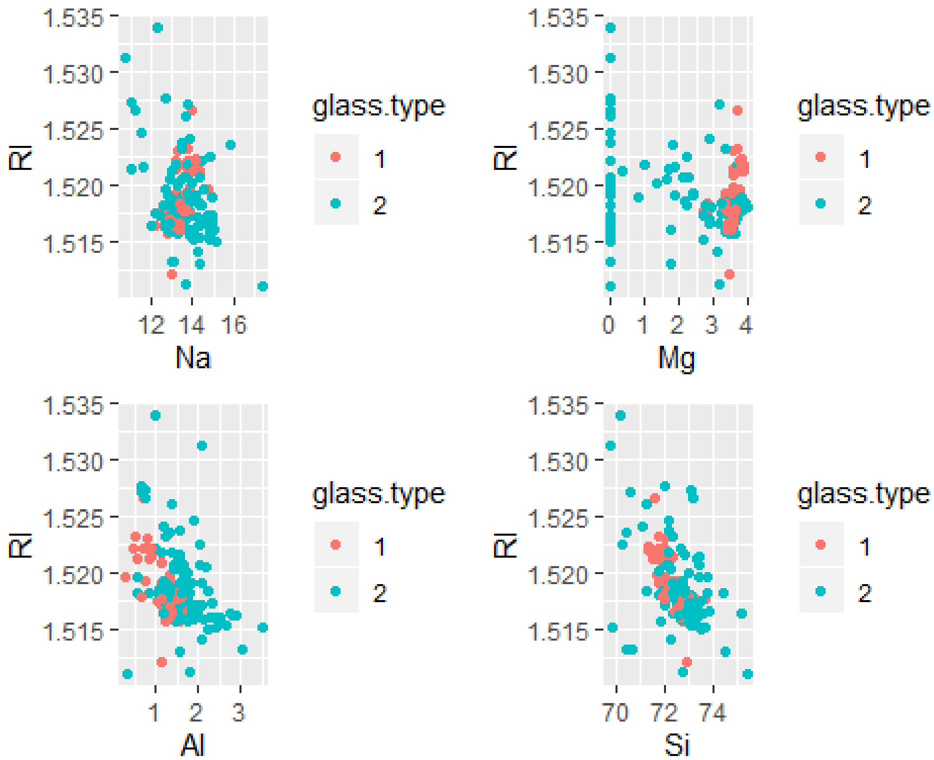
```
##          RI    Na   Mg   Al    Si    K   Ca Ba   Fe Type
## 1 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
## 2 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
## 3 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
## 4 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
## 5 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1
## 6 1.51743 13.30 3.60 1.14 73.09 0.58 8.17  0 0.00    1
```

Train Set Split

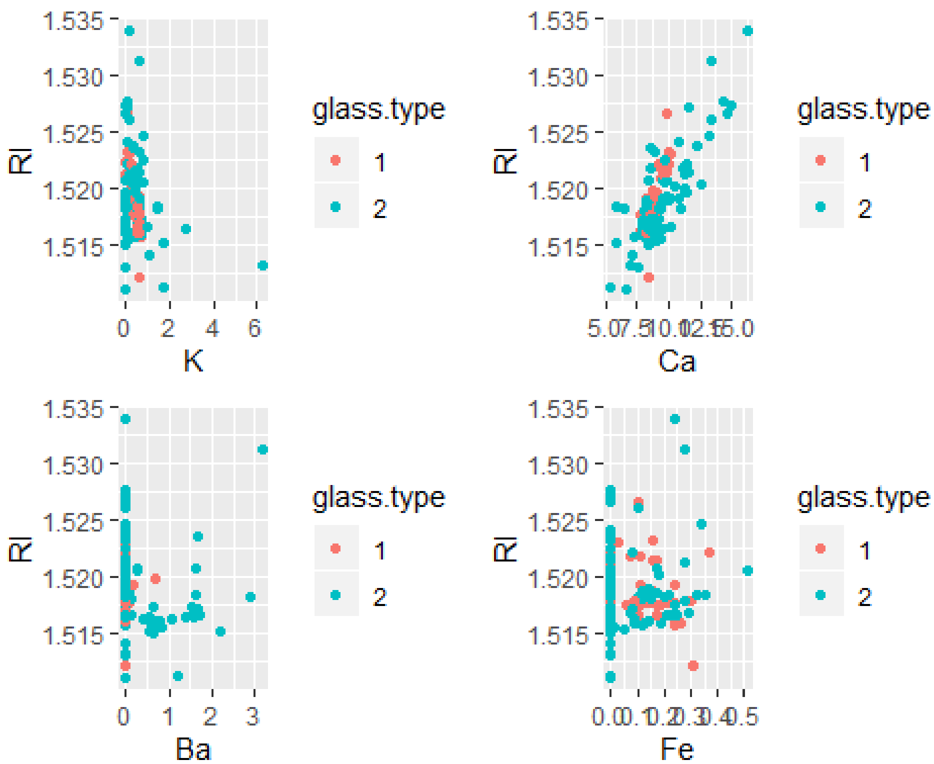
```
# we split the data
set.seed(1)

glass_idx = sample(nrow(glass), size = trunc(0.8 * nrow(glass)))
glass_trn = glass[glass_idx,]
glass_tst = glass[-glass_idx,]

X_train = glass_trn[,1:9]
y_train = glass_trn$Type

X_test = glass_tst[,1:9]
y_test = glass_tst$Type
```

pairplot

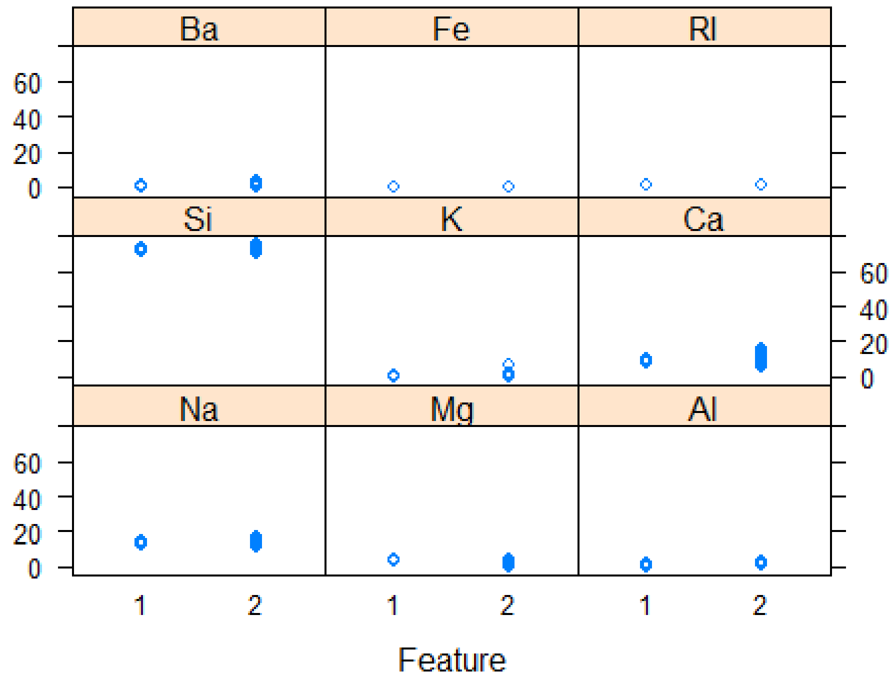
```
par(mfrow = c(4,2))
library(ggplot2)
library(gridExtra)
p1 = qplot(Na,RI,data=glass,colour = glass.type)
p2 = qplot(Mg,RI,data=glass,colour = glass.type)
p3 = qplot(Al,RI,data=glass,colour = glass.type)
p4 = qplot(Si,RI,data=glass,colour = glass.type)
p5 = qplot(K,RI,data=glass,colour = glass.type)
p6 = qplot(Ca,RI,data=glass,colour = glass.type)
p7 = qplot(Ba,RI,data=glass,colour = glass.type)
p8 = qplot(Fe,RI,data=glass,colour = glass.type)
grid.arrange(p1, p2,p3,p4, nrow = 2, ncol=2)
```




```r
grid.arrange(p5, p6,p7,p8, nrow = 2, ncol=2)
```

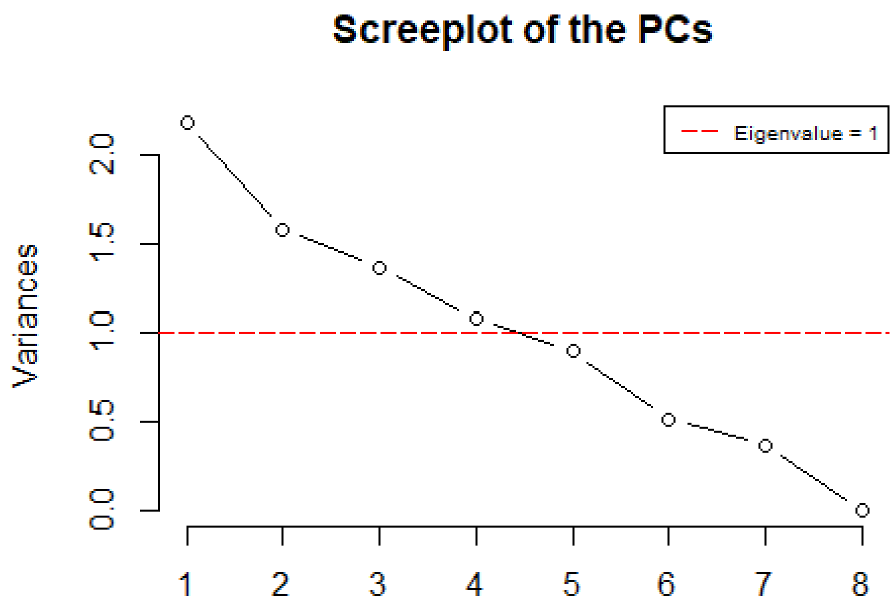


```r
featurePlot(x = glass[,c("Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe","RI
")], y = glass$Type)
```

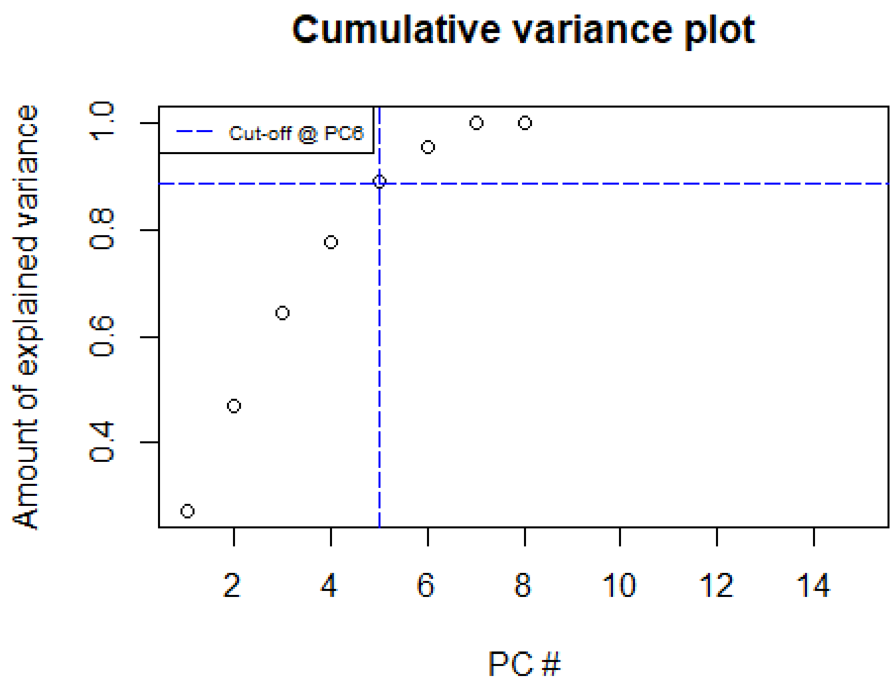


PCA

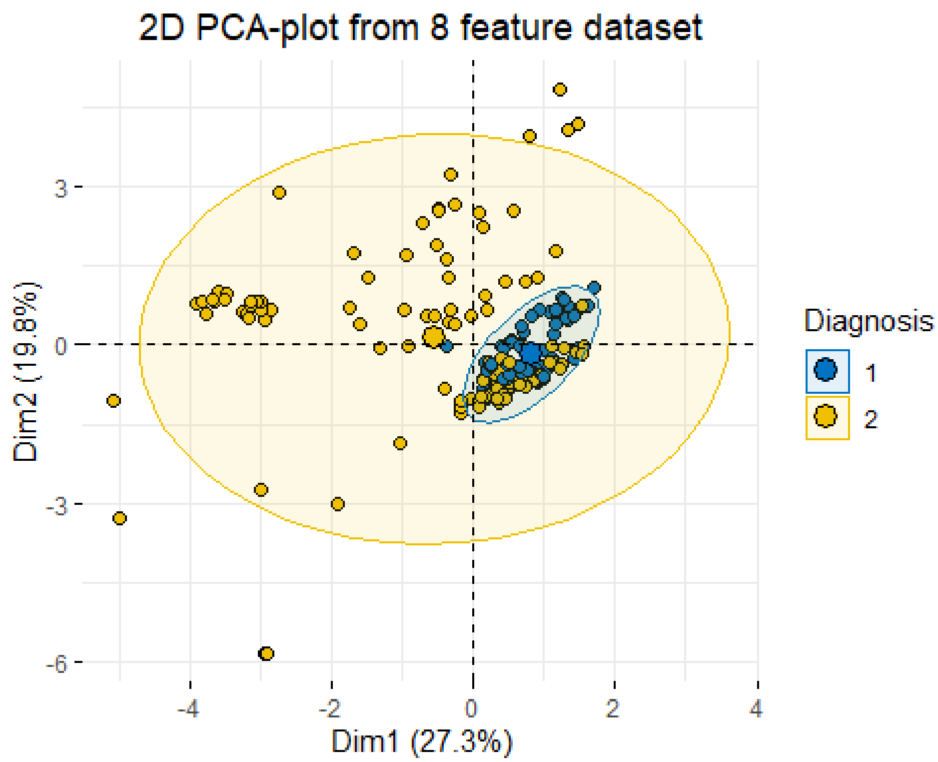
```r
glass.pr = prcomp(glass[,c(2:9)], center = TRUE, scale = TRUE)
summary(glass.pr)

## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5     PC6    PC7
## Standard deviation     1.4789 1.2587 1.1690 1.0394 0.9487 0.71754 0.6040
## Proportion of Variance 0.2734 0.1980 0.1708 0.1351 0.1125 0.06436 0.0456
## Cumulative Proportion  0.2734 0.4714 0.6423 0.7773 0.8898 0.95419 0.9998
##                            PC8
## Standard deviation     0.04105
## Proportion of Variance 0.00021
## Cumulative Proportion  1.00000

screeplot(glass.pr, type = "l", npcs = 8, main = "Screeplot of the PCs")
abline(h = 1, col="red", lty=5)
legend("topright", legend=c("Eigenvalue = 1"),
       col=c("red"), lty=5, cex=0.6)
```



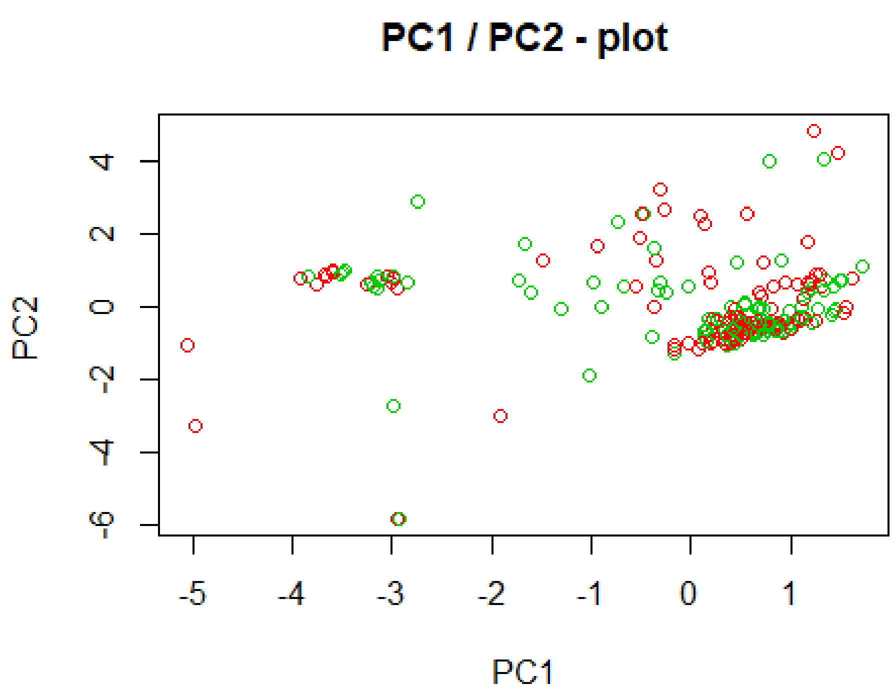

```
cumpro <- cumsum(glass.pr$sdev^2 / sum(glass.pr$sdev^2))
plot(cumpro[0:15], xlab = "PC #", ylab = "Amount of explained variance", main
 = "Cumulative variance plot")
abline(v = 5, col="blue", lty=5)
abline(h = 0.88759, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC6"),
       col=c("blue"), lty=5, cex=0.6)
```




```r
library("factoextra")

## Welcome! Want to learn more? See two factoextra-related books at https://g
oo.gl/ve3WBa

fviz_pca_ind(glass.pr, geom.ind = "point", pointshape = 21,
             pointsize = 2,
             fill.ind = as.factor(glass$Type),
             col.ind = "black",
             palette = "jco",
             addEllipses = TRUE,
             label = "var",
             col.var = "black",
             repel = TRUE,
             legend.title = "Diagnosis") +
  ggtitle("2D PCA-plot from 8 feature dataset") +
  theme(plot.title = element_text(hjust = 0.5))
```

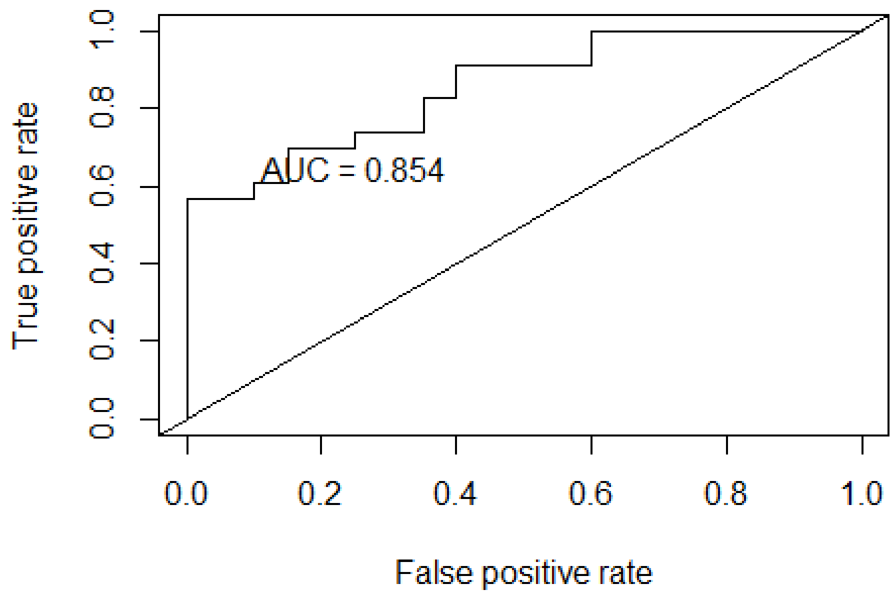



```r
plot(glass.pr$x[,1],glass.pr$x[,2], col = c(2,3), xlab="PC1", ylab = "PC2 ",
main = "PC1 / PC2 - plot")
```



LDA

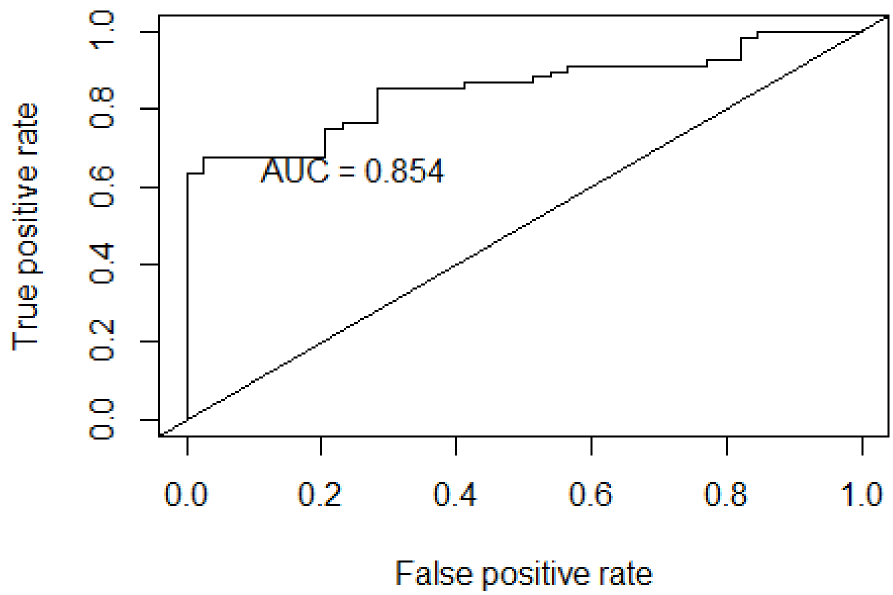
```
library(MASS)
glass.lda = lda(Type ~ ., data = glass_trn)

glass.lda.predict = predict(glass.lda, newdata = glass_tst)

### CONSTRUCTING ROC AUC PLOT:
# Get the posteriors as a dataframe.
library(ROCR)

## Loading required package: gplots

##
## Attaching package: 'gplots'

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

glass.lda.predict.posteriors <- as.data.frame(glass.lda.predict$posterior)
# Evaluate the model
pred <- ROCR::prediction(glass.lda.predict.posteriors[,2], y_test)
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
auc.train <- performance(pred, measure = "auc")
auc.train <- auc.train@y.values
# Plot
plot(roc.perf)
abline(a=0, b= 1)
text(x = .25, y = .65 ,paste("AUC = ", round(auc.train[[1]],3), sep = ""))
```



LDA (pca)

```
glass.pcst = glass.pr$x[,1:4]
glass.pcst <- cbind(glass.pcst, as.numeric(glass.type)-1)
colnames(glass.pcst)[5] <- "type"

set.seed(1996)
num_obs = nrow(glass.pcst)

train_index = sample(num_obs, size = trunc(0.50 * num_obs))
train_data = data.frame(glass.pcst[train_index, ])
test_data = data.frame(glass.pcst[-train_index, ])

library(MASS)
glass.lda = lda(type ~ PC1+PC2+PC3+PC4, data = train_data)

glass.lda.predict = predict(glass.lda, newdata = test_data)

### CONSTRUCTING ROC AUC PLOT:
# Get the posteriors as a dataframe.
library(ROCR)
glass.lda.predict.posteriors <- as.data.frame(glass.lda.predict$posterior)
# Evaluate the model
pred <- ROCR::prediction(glass.lda.predict.posteriors[,2], test_data$type)
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
auc.train <- performance(pred, measure = "auc")
auc.train <- auc.train@y.values
# Plot
```

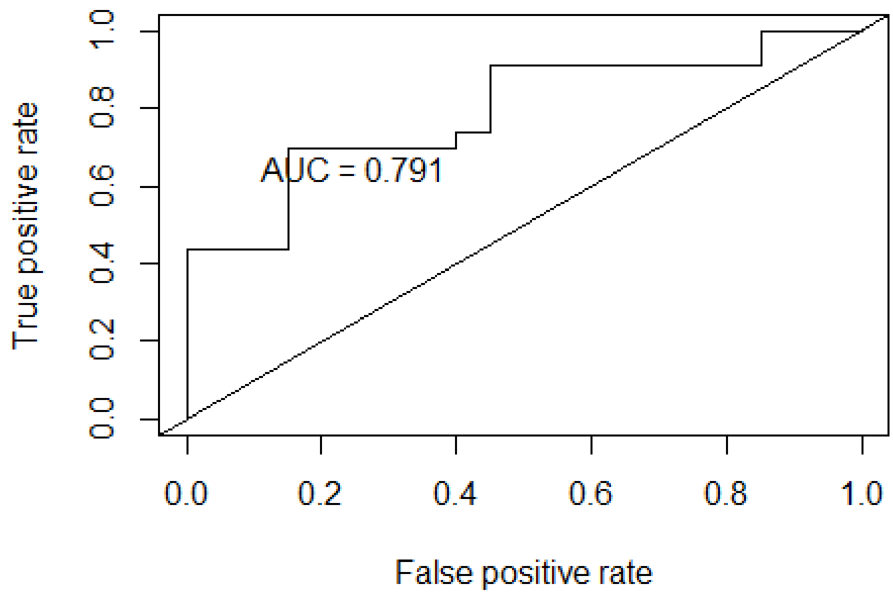
```r
plot(roc.perf)
abline(a=0, b= 1)
text(x = .25, y = .65 ,paste("AUC = ", round(auc.train[[1]],3), sep = ""))
```



So, it is worse than the result from what we get from the original training set. So, we give it up.

Now, let us see the performance of QDA.

```r
glass.qda = qda(Type ~ ., data = glass_trn)

glass.qda.predict = predict(glass.qda, newdata = glass_tst)

### CONSTRUCTING ROC AUC PLOT:
# Get the posteriors as a dataframe.
library(ROCR)
glass.qda.predict.posteriors <- as.data.frame(glass.qda.predict$posterior)
# Evaluate the model
pred <- ROCR::prediction(glass.qda.predict.posteriors[,2], y_test)
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
auc.train <- performance(pred, measure = "auc")
auc.train <- auc.train@y.values
# Plot
plot(roc.perf)
abline(a=0, b= 1)
text(x = .25, y = .65 ,paste("AUC = ", round(auc.train[[1]],3), sep = ""))
```

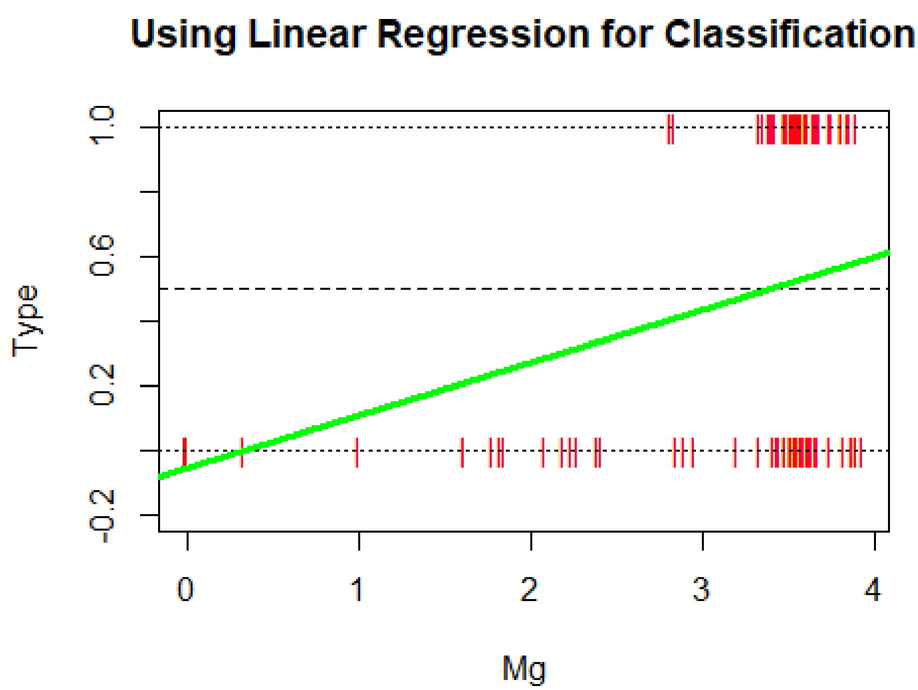


Supervised Learning for binary classification

Linear Regression

```
lm.model = lm(Type~Mg,data = glass_trn)
sqrt(mean((lm.model$residuals)^2))

## [1] NA

for(i in names(glass_trn[1:9])){
  fit = lm(glass_trn$Type ~ glass_trn[,i])
  cat(i,"-->", mean((fit$residuals)^2),"\n")
}



## RI --> 0.2327927
## Na --> 0.2255
## Mg --> 0.1791075
## Al --> 0.1789052
## Si --> 0.2331429
## K --> 0.2347119
## Ca --> 0.2336665
## Ba --> 0.2190514
## Fe --> 0.22811
```

```r
# Mg is chosen for the predictor for linear regression.

plot(Type ~ Mg, data = glass_trn,
     col = "red", pch = "|", ylim = c(-0.2, 1),
     main = "Using Linear Regression for Classification")

## Warning in spineplot.default(x, y, ...): y axis is on a cumulative probabi
lity
## scale, 'ylim' must be in [0,1]

abline(h = 0, lty = 3)
abline(h = 1, lty = 3)
abline(h = 0.5, lty = 2)
abline(lm.model, lwd = 3, col = "green")
```



This model is not good since it provides some negative probabilities.

Generalized Linear Model

```r
# LOOCV
library(boot)

##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
##
##     melanoma
```

```r
cv.error = rep(0,5)
for(i in 1:5){
  glm.fit = glm(Type~poly(Mg,i),data = glass_trn, family = "binomial")
  cv.error[i] = cv.glm(glass_trn,glm.fit)$delta[1]
  cat("polynomial: ", i, "--> cv error: ",cv.error[i], "\n")
}

## polynomial:  1 --> cv error:  0.1754791
## polynomial:  2 --> cv error:  0.1661204
## polynomial:  3 --> cv error:  0.1652688
## polynomial:  4 --> cv error:  0.1668545
## polynomial:  5 --> cv error:  0.1677744

polynomial = max(which.min(cv.error))
cat('the best ploynomial is ', polynomial)

## the best ploynomial is  3

glm.model = glm(Type ~ poly(Mg,polynomial), data = glass_trn, family = "binom
ial")
glm.prob = predict(glm.model, newdata = glass_tst, type = "response")
glm.pred = ifelse(glm.prob>0.5,2,1)

cal_class_err = function(actual, predicted){
  mean(actual!=predicted)
}

cal_class_err(actual = y_test, predicted = glm.pred)

## [1] 0.3255814

CM_log = confusionMatrix(y_test, factor(glm.pred))
CM_log

## Confusion Matrix and Statistics
##
##             Reference
## Prediction  1  2
##          1 15  5
##          2  9 14
##
##                Accuracy : 0.6744
##                  95% CI : (0.5146, 0.8092)
##     No Information Rate : 0.5581
##     P-Value [Acc > NIR] : 0.08227
##
##                   Kappa : 0.3541
##
##  Mcnemar's Test P-Value : 0.42268
##
##             Sensitivity : 0.6250
##             Specificity : 0.7368
```

```
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.6087
##              Prevalence : 0.5581
##          Detection Rate : 0.3488
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.6809
##
##        'Positive' Class : 1
##
```

```
metrics.log = CM_log$byClass
metrics.log
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.6250000            0.7368421            0.7500000
##       Neg Pred Value            Precision               Recall
##            0.6086957            0.7500000            0.6250000
##                   F1           Prevalence       Detection Rate
##            0.6818182            0.5581395            0.3488372
## Detection Prevalence    Balanced Accuracy
##            0.4651163            0.6809211
```

```
# k-fold
set.seed(1)
cv.error.10 = rep(0,10)
for(i in 1:10){
  glm.fit = glm(Type ~ poly(Mg,i),data =glass_trn,family = "binomial")
  cv.error.10[i] = cv.glm(glass_trn, glm.fit, K=10)$delta[1]
  cat("polynomial: ", i, "--> cv error: ",cv.error.10[i], "\n")
}
```

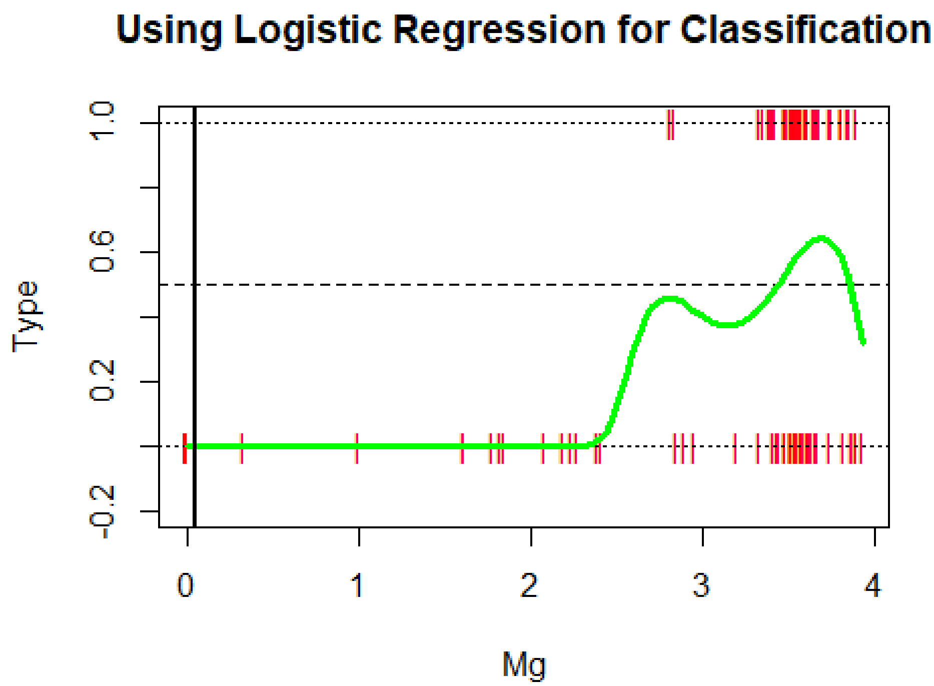
```
## polynomial:  1 --> cv error:  0.1765927
## polynomial:  2 --> cv error:  0.1666792
## polynomial:  3 --> cv error:  0.167374
## polynomial:  4 --> cv error:  0.1687767
## polynomial:  5 --> cv error:  0.1669423
## polynomial:  6 --> cv error:  0.2016099
## polynomial:  7 --> cv error:  0.18601
## polynomial:  8 --> cv error:  0.1891212
## polynomial:  9 --> cv error:  0.3563134
## polynomial:  10 --> cv error:  0.3294118
```

```
polynomial = max(which.min(cv.error.10))
cat('the best ploynomial is ', polynomial)
```

```
## the best ploynomial is  2
```

So, we take the polynomial to be 7 so that we can reduce the varibility and have least chance to overfit. Let’s see what will happen.

```r
glm.model.10fold = glm(Type ~ poly(Mg,polynomial), data = glass_trn, family =
 "binomial")

glm.pred.10fold = ifelse(predict(glm.model.10fold, newdata = glass_tst, type
= "response")>0.5,2,1)

cal_class_err = function(actual, predicted){
  mean(actual!=predicted)
}

cal_class_err(actual = glass_tst$Type, predicted = glm.pred.10fold)

## [1] 0.3488372

plot(Type ~ Mg, data = glass_tst,
     col = "red", pch = "|", ylim = c(-0.2, 1),
     main = "Using Logistic Regression for Classification")

## Warning in spineplot.default(x, y, ...): y axis is on a cumulative probabi
lity
## scale, 'ylim' must be in [0,1]

abline(h = 0, lty = 3)
abline(h = 1, lty = 3)
abline(h = 0.5, lty = 2)
curve(predict(glm.model.10fold, data.frame(Mg = x), type = "response"),
      add = TRUE, lwd = 3, col = "green", )
abline(v = -coef(glm.model.10fold)[1] / coef(glm.model.10fold)[2], lwd = 2)
```



K-Nearest Neighbor Classifier

```
knn.pred = knn(train = scale(X_train), test = scale(X_test), cl = y_train, k
= 3)
knn.pred

##  [1] 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 2 2 2 2 1 1 2 2 2 2 2 2 1 2 1 1 2
 2 2 2
## [39] 2 2 2 2 2
## Levels: 1 2

cal_class_err(actual = y_test, predicted = knn.pred)

## [1] 0.1627907
```

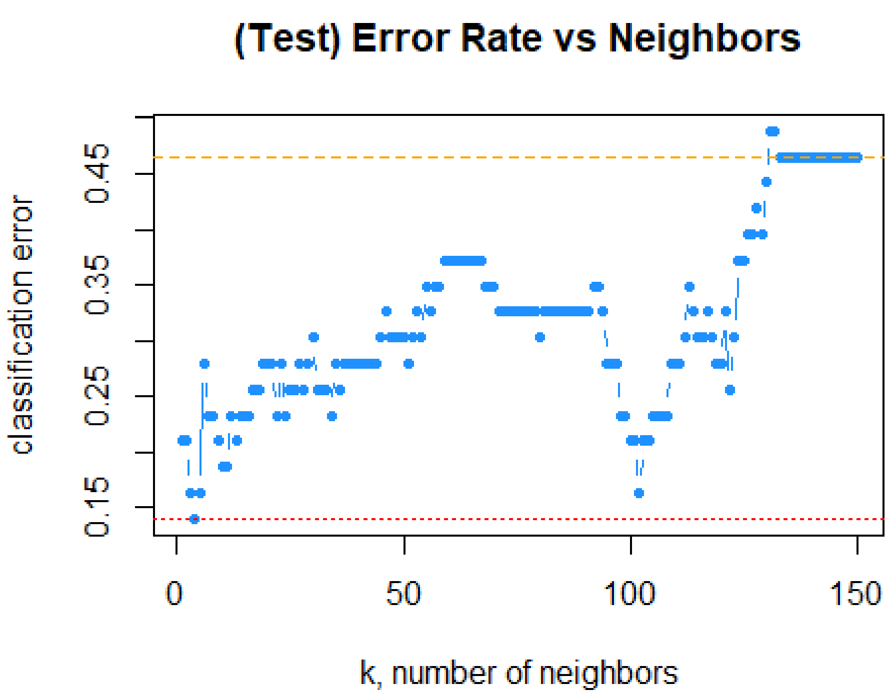
Then, I try to choose k.

```
set.seed(199)
k_to_try = 1:150
err_k = rep(x = 0, times = length(k_to_try))

for (i in seq_along(k_to_try)) {
  pred = knn(train = scale(X_train),
             test  = scale(X_test),
             cl    = y_train,
             k     = k_to_try[i])
  err_k[i] = cal_class_err(actual = y_test, predicted = pred)
  if(i %% 10 == 0)
```

```r
    cat('K:', i, "--> error: ", err_k[i], "\n")
}
```

```
## K: 10 --> error:  0.1860465
## K: 20 --> error:  0.2790698
## K: 30 --> error:  0.3023256
## K: 40 --> error:  0.2790698
## K: 50 --> error:  0.3023256
## K: 60 --> error:  0.372093
## K: 70 --> error:  0.3488372
## K: 80 --> error:  0.3023256
## K: 90 --> error:  0.3255814
## K: 100 --> error:  0.2093023
## K: 110 --> error:  0.2790698
## K: 120 --> error:  0.2790698
## K: 130 --> error:  0.4418605
## K: 140 --> error:  0.4651163
## K: 150 --> error:  0.4651163
```

```r
# plot error vs choice of k
plot(err_k, type = "b", col = "dodgerblue", cex = 1, pch = 20,
     xlab = "k, number of neighbors", ylab = "classification error",
     main = "(Test) Error Rate vs Neighbors")
# add line for min error seen
abline(h = min(err_k), col = "red", lty = 3)
# add line for minority prevalence in test set
abline(h = mean(y_test == 1), col = "orange", lty = 2)
```



```r
which(min(err_k) == err_k)
```

```
## [1] 4
```

```r
k.best = max(which(min(err_k) == err_k))
```

In this case, we choose k = 4 since the largest one is the least variable, and has the least chance of overfitting.

```r
knn.pred.best = knn(train = scale(X_train), test = scale(X_test), cl = y_trai
n, k = k.best)
```

```r
CM_knn = confusionMatrix(factor(y_test), factor(knn.pred.best))
CM_knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 15  5
##          2  6 17
##
##                Accuracy : 0.7442
##                  95% CI : (0.5883, 0.8648)
##     No Information Rate : 0.5116
##     P-Value [Acc > NIR] : 0.001574
##
##                   Kappa : 0.4875
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.7143
##             Specificity : 0.7727
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.7391
##              Prevalence : 0.4884
##          Detection Rate : 0.3488
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.7435
##
##        'Positive' Class : 1
##
```

```r
metrics.knn = CM_knn$byClass
metrics.knn
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.7142857            0.7727273            0.7500000
##       Neg Pred Value            Precision               Recall
##            0.7391304            0.7500000            0.7142857
##                   F1           Prevalence       Detection Rate
```

```
##             0.7317073             0.4883721             0.3488372
## Detection Prevalence    Balanced Accuracy
##             0.4651163             0.7435065
```

SVM

```
my_confusionmatrix = function(pred,truth,lvs = c(1,2,3,5,6,7)){
  lvs = lvs
  truth = factor(truth,levels = lvs)
  prediction = factor(pred,levels = lvs)

  CM = confusionMatrix(truth, prediction)
  return(CM)
}

library(e1071)
mysvm = function(kernel){
  svm.tune=tune(svm ,Type~.,data=glass_trn ,kernel = kernel,
ranges=list(gamma = 2^(-8:1), cost = 2^(0:4)),
tunecontrol = tune.control(sampling = "fix"))

  best_gamma = svm.tune$best.parameters[1]
  best_cost = svm.tune$best.parameters[2]

  x.svm <- svm(Type~., data = glass_trn, cost=best_cost, gamma=best_gamma, ke
rnel = kernel, probability = TRUE)
  x.svm.prob <- predict(x.svm, type="prob", newdata=glass_tst[-10], probabili
ty = TRUE)

  return(list(
    best_model = svm.tune$best.model,
    svm.prob = x.svm.prob
  ))
}

library(pROC)

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

SVM_pred = function(kernel){
  model = mysvm(kernel = kernel)$best_model
  ypred = predict(model,glass_tst[-10])

  CM_svm = my_confusionmatrix(ypred,glass_tst[,10], lvs = c(1,2))
  print(CM_svm)
```

```r
  accuracy = (sum(diag(CM_svm$table)))/sum(CM_svm$table)

  predictions <- as.numeric(predict(model, glass_tst[-10], type = 'response
'))

  roc.multi <- multiclass.roc(glass_tst[,10], predictions, quiet = TRUE)

  cat('kernel: ',kernel, '\n')
  cat('accuracy: ',accuracy, '\n')
  cat('AUC: ',auc(roc.multi), '\n')
  cat('\n')
  return(list(
    predictions = ypred,
    accuracy = accuracy
  ))
}

svm1=SVM_pred('linear')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 14  6
##          2  3 20
##
##                Accuracy : 0.7907
##                  95% CI : (0.6396, 0.8996)
##     No Information Rate : 0.6047
##     P-Value [Acc > NIR] : 0.007818
##
##                   Kappa : 0.5752
##
##  Mcnemar's Test P-Value : 0.504985
##
##             Sensitivity : 0.8235
##             Specificity : 0.7692
##          Pos Pred Value : 0.7000
##          Neg Pred Value : 0.8696
##              Prevalence : 0.3953
##          Detection Rate : 0.3256
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.7964
##
##        'Positive' Class : 1
##
## kernel:  linear
## accuracy:  0.7906977
## AUC:  0.7847826
```

```r
svm2=SVM_pred('polynomial')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 18  2
##          2  7 16
##
##                Accuracy : 0.7907
##                  95% CI : (0.6396, 0.8996)
##     No Information Rate : 0.5814
##     P-Value [Acc > NIR] : 0.003287
##
##                   Kappa : 0.5861
##
##  Mcnemar's Test P-Value : 0.182422
##
##             Sensitivity : 0.7200
##             Specificity : 0.8889
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.6957
##              Prevalence : 0.5814
##          Detection Rate : 0.4186
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.8044
##
##        'Positive' Class : 1
##
## kernel:  polynomial
## accuracy:  0.7906977
## AUC:  0.7978261

svm3=SVM_pred('radial')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 14  6
##          2  2 21
##
##                Accuracy : 0.814
##                  95% CI : (0.666, 0.9161)
##     No Information Rate : 0.6279
##     P-Value [Acc > NIR] : 0.006902
##
##                   Kappa : 0.6211
##
##  Mcnemar's Test P-Value : 0.288844
```

```
##
##             Sensitivity : 0.8750
##             Specificity : 0.7778
##          Pos Pred Value : 0.7000
##          Neg Pred Value : 0.9130
##              Prevalence : 0.3721
##          Detection Rate : 0.3256
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.8264
##
##        'Positive' Class : 1
##
## kernel:  radial
## accuracy:  0.8139535
## AUC:  0.8065217
```

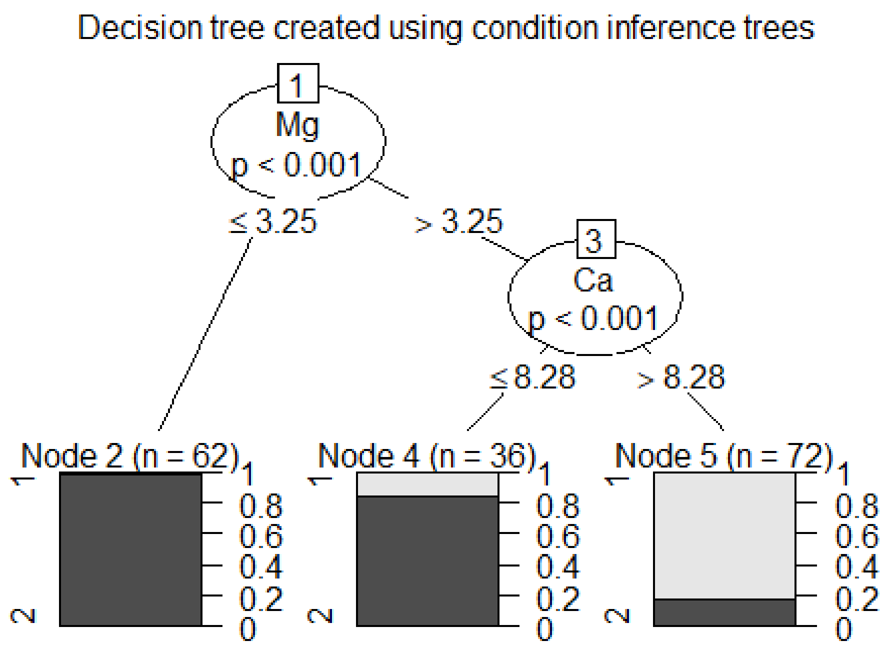
```r
svm4=SVM_pred('sigmoid')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 12  8
##          2  7 16
##
##                Accuracy : 0.6512
##                  95% CI : (0.4907, 0.7899)
##     No Information Rate : 0.5581
##     P-Value [Acc > NIR] : 0.141
##
##                   Kappa : 0.2966
##
##  Mcnemar's Test P-Value : 1.000
##
##             Sensitivity : 0.6316
##             Specificity : 0.6667
##          Pos Pred Value : 0.6000
##          Neg Pred Value : 0.6957
##              Prevalence : 0.4419
##          Detection Rate : 0.2791
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.6491
##
##        'Positive' Class : 1
##
## kernel:  sigmoid
## accuracy:  0.6511628
## AUC:  0.6478261
```

So, I choose the kernel to be “radial”.

Tree

```
library(party)

## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo

##
## Attaching package: 'zoo'

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

## Loading required package: sandwich

x.ct <- ctree(Type ~ ., data=glass_trn)
x.ct.pred <- predict(x.ct, newdata=glass_tst)
x.ct.prob <-  1- unlist(treeresponse(x.ct, glass_tst), use.names=F)[seq(1,nro
w(glass_tst)*2,2)]
# To view the decision tree, uncomment this line.
plot(x.ct, main="Decision tree created using condition inference trees")
```



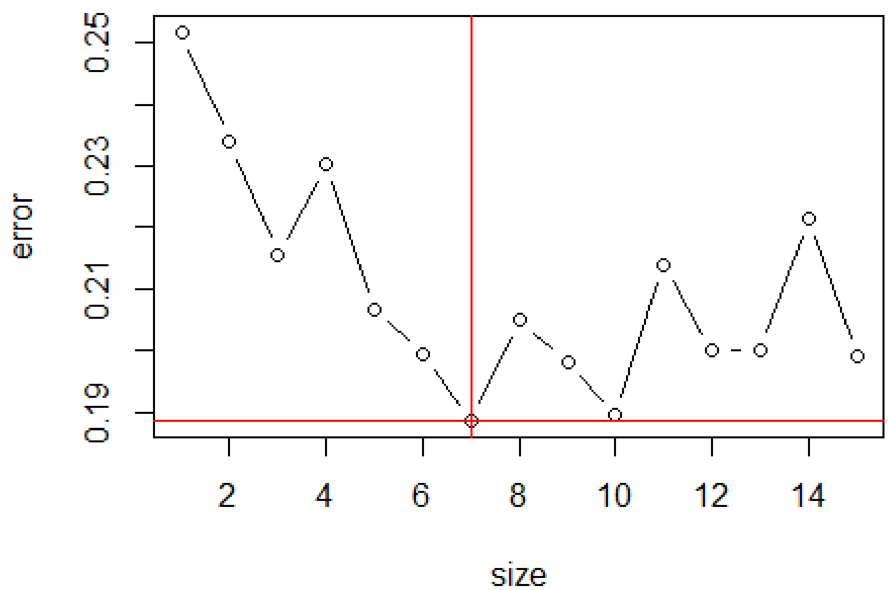
Random Forest

```
x.cf <- cforest(Type ~ ., data=glass_trn, control = cforest_unbiased(mtry = n
col(glass)-2))
x.cf.pred <- predict(x.cf, newdata=glass_tst)
x.cf.prob <-  1- unlist(treeresponse(x.cf, glass_tst), use.names=F)[seq(1,nro
w(glass_tst)*2,2)]
```

Neural Network

```
library(nnet)

# creating training and test set
# fit neural network
set.seed(202)

scaler = function(x){
  return(
    (x - min(x)) / (max(x) - min(x))
  )
}

glass_trn[-10] = apply(glass_trn[-10],2,scaler)
glass_tst[-10] = apply(glass_tst[-10],2,scaler)


my_nnet = function(size){
  NN = nnet(Type~.,data = glass_trn, size = size,maxit = 200, decay = 5e-4)
```

```
  return(NN)
}

# nnet with 10 fold cv
glass[,1:9] = apply(glass[,1:9],2,scaler)
m = tune.nnet(Type~., data = glass, size = 1:15)
nn.cv = summary(m)

plot(nn.cv$performances[,1:2], type = "b")
abline(h = min(nn.cv$performances[,2]), v = nn.cv$performances[,1][which.min
(nn.cv$performances[,2])], col=2)
```




```
nn.pred = predict(nn.cv$best.model, glass_tst[-10], type = "class")
table(nn.pred, glass_tst[,10])

##
## nn.pred  1  2
##       1 12  4
##       2  8 19

NN.prediction = function(size){
  NN = my_nnet(size)
  pred = predict(NN, glass_tst[-10], type = "class")
  tab = table(pred,glass_tst[,10])
  print(tab)
  accuracy = sum(diag(tab))/sum(tab)
  return(accuracy)
}
```

```r
size = seq(2,20,2)
res = c()
for(i in size){
  res = append(res, NN.prediction(i))
}
```

```
## # weights:  23
## initial  value 132.508485
## iter  10 value 69.220885
## iter  20 value 52.324635
## iter  30 value 48.434192
## iter  40 value 45.220883
## iter  50 value 44.394378
## iter  60 value 44.059824
## iter  70 value 43.776213
## iter  80 value 43.253933
## iter  90 value 43.138315
## iter 100 value 42.508439
## iter 110 value 36.202623
## iter 120 value 33.051590
## iter 130 value 32.246727
## iter 140 value 31.914488
## iter 150 value 31.868013
## iter 160 value 31.782893
## iter 170 value 31.753759
## iter 180 value 31.739202
## iter 190 value 31.737310
## iter 200 value 31.737104
## final  value 31.737104
## stopped after 200 iterations
##
## pred  1  2
##    1 14  2
##    2  6 21
## # weights:  45
## initial  value 112.847533
## iter  10 value 71.138702
## iter  20 value 62.371784
## iter  30 value 50.905120
## iter  40 value 44.125952
## iter  50 value 39.910832
## iter  60 value 32.019915
## iter  70 value 25.065594
## iter  80 value 22.238562
## iter  90 value 19.528563
## iter 100 value 18.428845
## iter 110 value 16.901135
## iter 120 value 16.037493
## iter 130 value 15.646787
## iter 140 value 14.980940
```

```
## iter 150 value 14.538360
## iter 160 value 14.265518
## iter 170 value 14.129305
## iter 180 value 14.062468
## iter 190 value 14.042130
## iter 200 value 14.026472
## final  value 14.026472
## stopped after 200 iterations
##
## pred  1  2
##    1 17  4
##    2  3 19
## # weights:  67
## initial  value 143.408617
## iter  10 value 68.851641
## iter  20 value 52.895284
## iter  30 value 42.309064
## iter  40 value 32.880064
## iter  50 value 25.664523
## iter  60 value 22.000378
## iter  70 value 20.573571
## iter  80 value 19.874903
## iter  90 value 19.512827
## iter 100 value 18.090708
## iter 110 value 15.914967
## iter 120 value 14.385483
## iter 130 value 13.863577
## iter 140 value 13.609342
## iter 150 value 13.342196
## iter 160 value 12.501904
## iter 170 value 11.485049
## iter 180 value 10.800676
## iter 190 value 10.284500
## iter 200 value 9.804449
## final  value 9.804449
## stopped after 200 iterations
##
## pred  1  2
##    1  4  0
##    2 16 23
## # weights:  89
## initial  value 129.058449
## iter  10 value 70.834308
## iter  20 value 55.226176
## iter  30 value 44.809660
## iter  40 value 35.815852
## iter  50 value 30.294163
## iter  60 value 25.217558
## iter  70 value 22.148851
## iter  80 value 18.838759
```

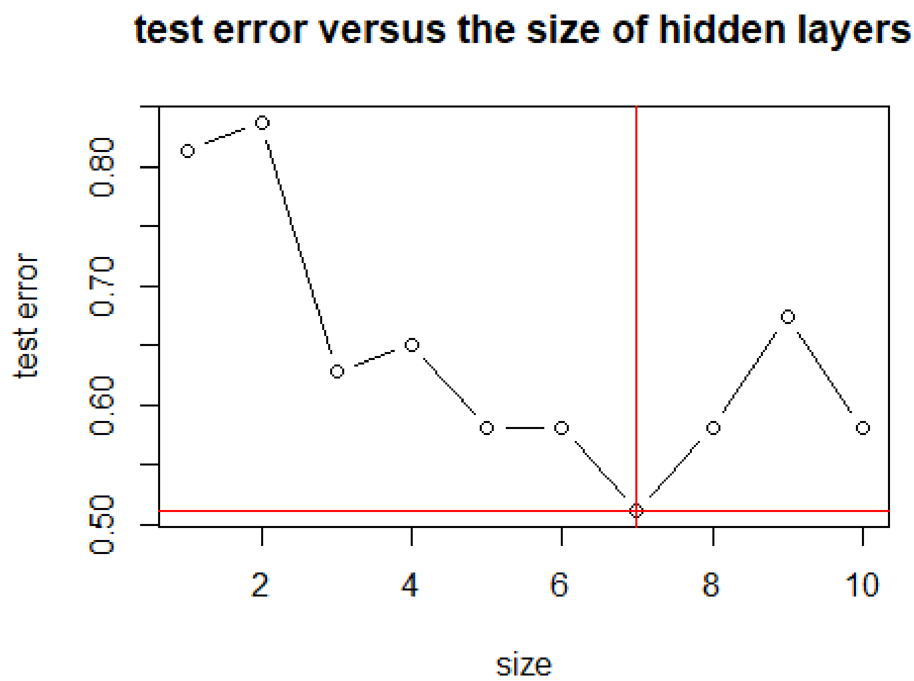
```
## iter  90 value 17.495769
## iter 100 value 15.798384
## iter 110 value 14.551730
## iter 120 value 14.080037
## iter 130 value 13.597459
## iter 140 value 13.363647
## iter 150 value 13.178701
## iter 160 value 12.968993
## iter 170 value 12.636582
## iter 180 value 11.753933
## iter 190 value 11.516281
## iter 200 value 10.747245
## final  value 10.747245
## stopped after 200 iterations
##
## pred  1  2
##    1  6  1
##    2 14 22
## # weights:  111
## initial  value 124.501315
## iter  10 value 71.479015
## iter  20 value 59.463583
## iter  30 value 46.703123
## iter  40 value 35.941064
## iter  50 value 21.804171
## iter  60 value 13.390708
## iter  70 value 10.514113
## iter  80 value 9.436769
## iter  90 value 8.930039
## iter 100 value 8.633947
## iter 110 value 8.453587
## iter 120 value 8.310715
## iter 130 value 8.210514
## iter 140 value 8.111169
## iter 150 value 8.056969
## iter 160 value 8.006119
## iter 170 value 7.981566
## iter 180 value 7.962822
## iter 190 value 7.949626
## iter 200 value 7.935431
## final  value 7.935431
## stopped after 200 iterations
##
## pred  1  2
##    1  9  7
##    2 11 16
## # weights:  133
## initial  value 140.932674
## iter  10 value 71.552846
## iter  20 value 51.598663
```

```
## iter  30 value 45.077147
## iter  40 value 35.788495
## iter  50 value 27.667835
## iter  60 value 21.908164
## iter  70 value 17.828512
## iter  80 value 14.882487
## iter  90 value 13.026209
## iter 100 value 11.536873
## iter 110 value 10.446957
## iter 120 value 9.894924
## iter 130 value 9.536264
## iter 140 value 9.284833
## iter 150 value 9.117504
## iter 160 value 9.009823
## iter 170 value 8.945685
## iter 180 value 8.889246
## iter 190 value 8.816351
## iter 200 value 8.745064
## final  value 8.745064
## stopped after 200 iterations
##
## pred  1  2
##    1  8  6
##    2 12 17
## # weights:  155
## initial  value 144.020480
## iter  10 value 69.317131
## iter  20 value 51.739718
## iter  30 value 37.558289
## iter  40 value 29.994401
## iter  50 value 23.414495
## iter  60 value 17.653970
## iter  70 value 15.117157
## iter  80 value 13.222829
## iter  90 value 12.423081
## iter 100 value 11.786134
## iter 110 value 11.225736
## iter 120 value 10.952711
## iter 130 value 10.642506
## iter 140 value 10.259163
## iter 150 value 9.986078
## iter 160 value 9.574566
## iter 170 value 8.899050
## iter 180 value 8.509894
## iter 190 value 8.321493
## iter 200 value 8.141327
## final  value 8.141327
## stopped after 200 iterations
##
## pred  1  2
```

```
##    1  2  3
##    2 18 20
## # weights:  177
## initial  value 195.012771
## iter  10 value 70.937332
## iter  20 value 59.744348
## iter  30 value 49.609532
## iter  40 value 43.425189
## iter  50 value 35.606437
## iter  60 value 28.544369
## iter  70 value 19.194441
## iter  80 value 13.737595
## iter  90 value 11.022446
## iter 100 value 9.619982
## iter 110 value 8.731473
## iter 120 value 8.182607
## iter 130 value 7.944586
## iter 140 value 7.738746
## iter 150 value 7.614328
## iter 160 value 7.472878
## iter 170 value 7.380981
## iter 180 value 7.305850
## iter 190 value 7.269297
## iter 200 value 7.242066
## final  value 7.242066
## stopped after 200 iterations
##
## pred  1  2
##    1  9  7
##    2 11 16
## # weights:  199
## initial  value 178.751458
## iter  10 value 79.045422
## iter  20 value 62.972859
## iter  30 value 47.449605
## iter  40 value 38.863727
## iter  50 value 28.795081
## iter  60 value 20.909213
## iter  70 value 14.929449
## iter  80 value 11.828881
## iter  90 value 10.356128
## iter 100 value 9.154166
## iter 110 value 8.824533
## iter 120 value 8.453648
## iter 130 value 8.281333
## iter 140 value 8.134035
## iter 150 value 8.043183
## iter 160 value 7.903396
## iter 170 value 7.819644
## iter 180 value 7.699095
```

```
## iter 190 value 7.620053
## iter 200 value 7.560030
## final  value 7.560030
## stopped after 200 iterations
##
## pred  1  2
##    1 15  9
##    2  5 14
## # weights:  221
## initial  value 134.618009
## iter  10 value 69.048306
## iter  20 value 58.958238
## iter  30 value 46.256504
## iter  40 value 39.698253
## iter  50 value 33.377728
## iter  60 value 24.465613
## iter  70 value 18.602863
## iter  80 value 15.800134
## iter  90 value 13.817932
## iter 100 value 12.454091
## iter 110 value 11.522106
## iter 120 value 10.845439
## iter 130 value 10.462885
## iter 140 value 10.143775
## iter 150 value 9.950769
## iter 160 value 9.780655
## iter 170 value 9.669216
## iter 180 value 9.546300
## iter 190 value 9.435715
## iter 200 value 9.317816
## final  value 9.317816
## stopped after 200 iterations
##
## pred  1  2
##    1  2  0
##    2 18 23
```

```
best_size = size[which.min(res)]
plot(res, type = "b", ylab = 'test error', xlab = 'size', main = "test error 
versus the size of hidden layers")
abline(v = which.min(res), h = min(res), col = 2)
```

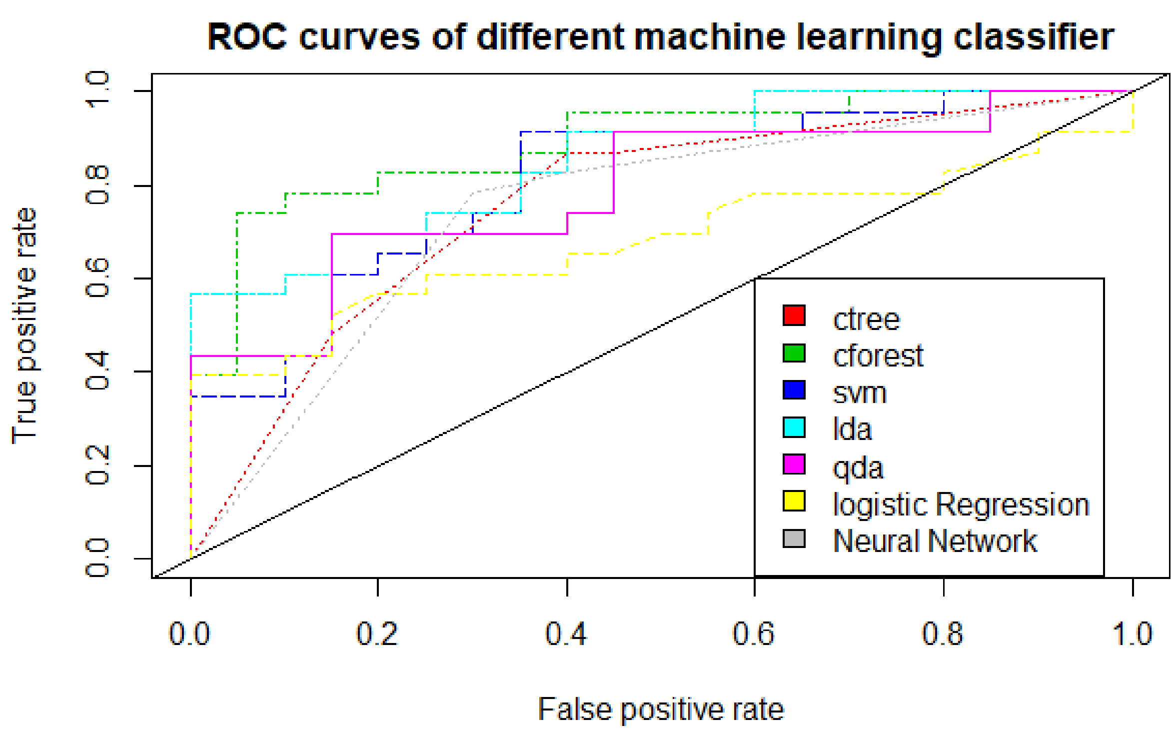


ROC

```
# ctree

x.ct.prob.rocr <- ROCR::prediction(x.ct.prob, y_test)

x.ct.perf <- performance(x.ct.prob.rocr, "tpr","fpr")

# add=TRUE draws on the existing chart

plot(x.ct.perf, lty = 3, col=2, main="ROC curves of different machine learnin
g classifier")



# Draw a legend

legend(0.6, 0.6, c('ctree', 'cforest','svm','lda','qda', 'logistic Regression
', 'Neural Network'), 2:8)



# cforest

x.cf.prob.rocr <- ROCR::prediction(x.cf.prob, y_test)

x.cf.perf <- performance(x.cf.prob.rocr, "tpr","fpr")
```

```
plot(x.cf.perf, col=3, lty = 4,add=TRUE)


# svm

x.svm <- svm(Type~., data = glass_trn,kernel = "linear", probability = TRUE)

x.svm.prob <- predict(x.svm, type="prob", newdata=glass_tst[-10], probability
 = TRUE)

x.svm.prob.rocr <- ROCR::prediction(attr(x.svm.prob, "probabilities")[,2], y_
test)

x.svm.perf <- performance(x.svm.prob.rocr, "tpr","fpr")

plot(x.svm.perf, col=4, lty = 5,add=TRUE)


# lda

glass.lda.predict.posteriors <- as.data.frame(glass.lda.predict$posterior)

# Evaluate the model

pred <- ROCR::prediction(glass.lda.predict.posteriors[,2], y_test)

roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")

auc.train <- performance(pred, measure = "auc")

auc.train <- auc.train@y.values

# Plot

plot(roc.perf, col=5, lty = 6,add=TRUE)


# QDA

glass.qda.predict.posteriors <- as.data.frame(glass.qda.predict$posterior)

# Evaluate the model

pred <- ROCR::prediction(glass.qda.predict.posteriors[,2], y_test)

roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")

auc.train <- performance(pred, measure = "auc")

auc.train <- auc.train@y.values

# Plot
```

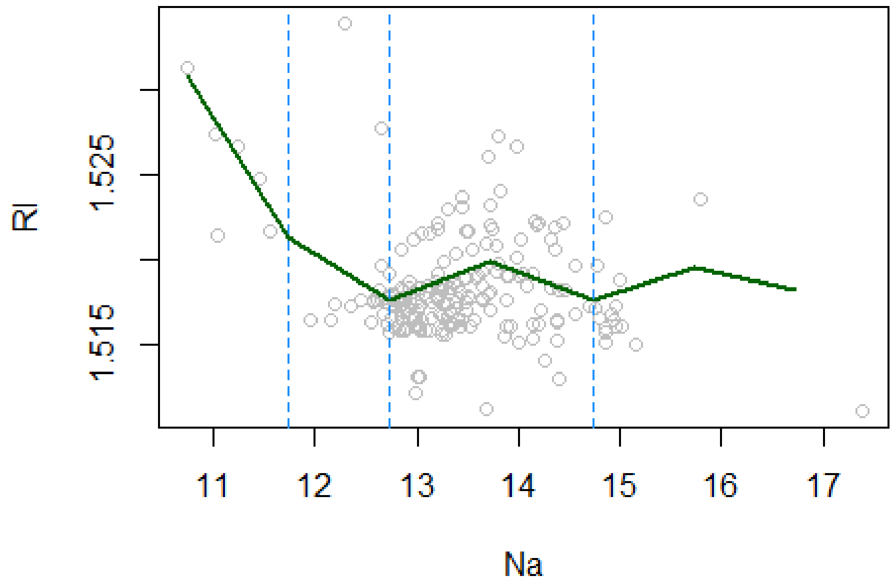
```
plot(roc.perf, col = 6 , lty = 7,add = TRUE)


# Logistic regression

pred = ROCR::prediction(glm.prob, y_test)

roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")

auc.train <- performance(pred, measure = "auc")

auc.train <- auc.train@y.values

plot(roc.perf, col = 7 , lty = 8,add = TRUE)


nn.prob = predict(nn.cv$best.model, glass_tst[-10], type = "raw")

pred = ROCR::prediction(nn.prob, glass_tst[,10])

perf = performance(pred, "tpr", "fpr")

plot(perf, col = 8, lty = 9, add = TRUE)

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



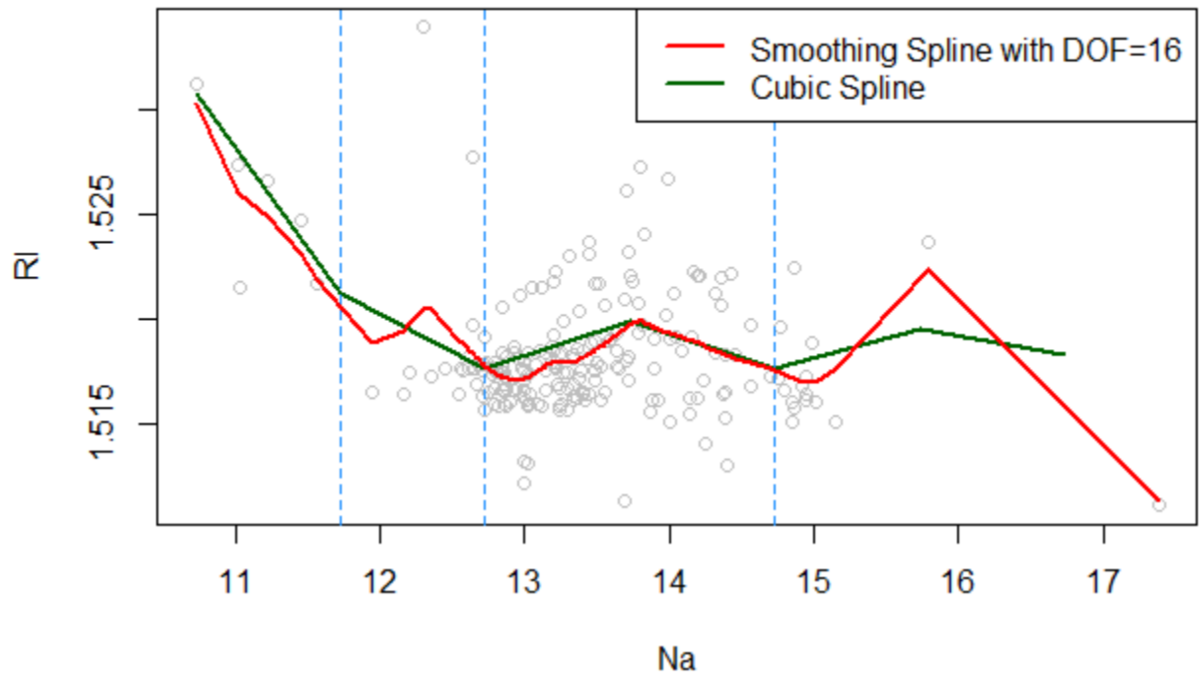
Spline

```r
# Load the data
glass = read.csv("glass.csv")
colnames(glass) = c("id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe
", "Type")

attach(glass)
set.seed(19)
knots = sample(min(Na):max(Na),3)


fit.spline = lm(RI~bs(Na,knots))

summary(fit.spline)

##
## Call:
## lm(formula = RI ~ bs(Na, knots))
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0085651 -0.0014435 -0.0003332  0.0008984  0.0139868
##
## Coefficients: (1 not defined because of singularities)
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.530806   0.002392 639.852  < 2e-16 ***
## bs(Na, knots)1  -0.012716   0.006137  -2.072  0.03956 *
## bs(Na, knots)2  -0.007729   0.003588  -2.154  0.03243 *
## bs(Na, knots)3  -0.013448   0.002683  -5.013 1.18e-06 ***
## bs(Na, knots)4  -0.013318   0.002560  -5.202 4.85e-07 ***
## bs(Na, knots)5  -0.014766   0.002755  -5.360 2.28e-07 ***
## bs(Na, knots)6  -0.011937   0.002858  -4.177 4.41e-05 ***
## bs(Na, knots)7  -0.012913   0.002683  -4.813 2.93e-06 ***
## bs(Na, knots)8  -0.013588   0.002519  -5.394 1.93e-07 ***
## bs(Na, knots)9  -0.009326   0.002569  -3.630  0.00036 ***
## bs(Na, knots)10 -0.019617   0.003740  -5.245 3.97e-07 ***
## bs(Na, knots)11 -0.002695   0.006504  -0.414  0.67909
## bs(Na, knots)12 -0.019322   0.003562  -5.424 1.67e-07 ***
## bs(Na, knots)13        NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002644 on 200 degrees of freedom
## Multiple R-squared:  0.2855, Adjusted R-squared:  0.2426
## F-statistic:  6.66 on 12 and 200 DF,  p-value: 5.228e-10

Nalims = range(Na)
Na.grid = seq(Nalims[1],Nalims[2])
```

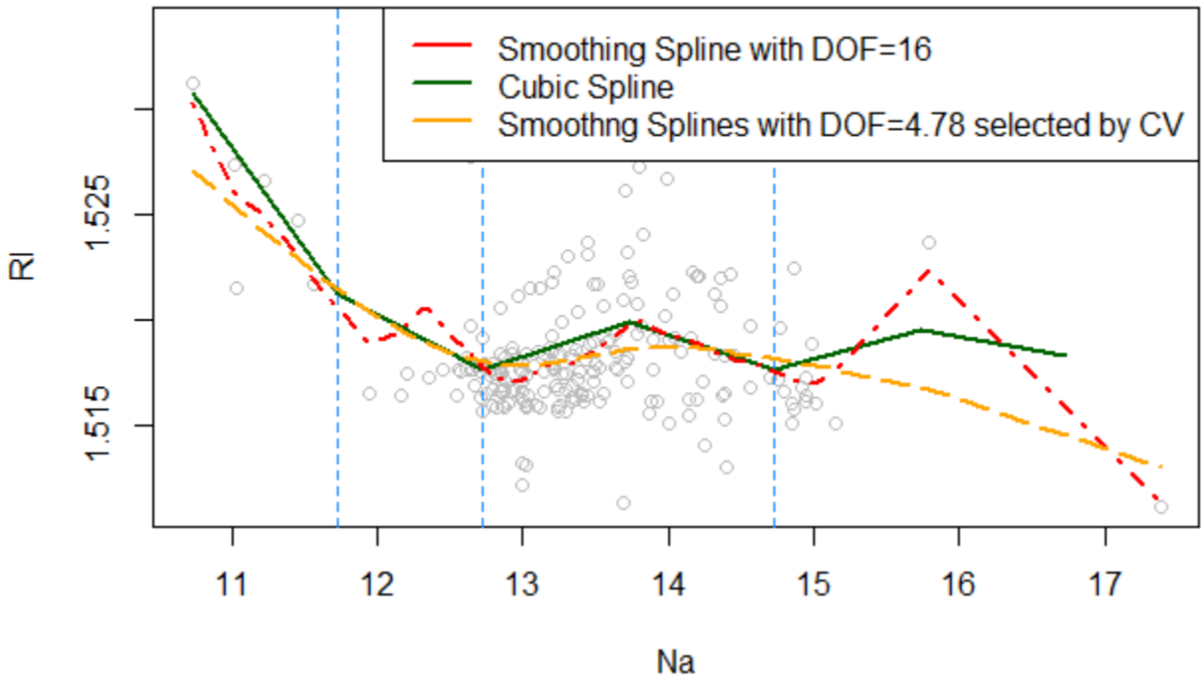
```r
plot(Na,RI,col="grey",xlab='Na',ylab='RI')
points(Na.grid,predict(fit.spline,newdata = list(Na = Na.grid)),col="darkgree
n",lwd=2,type="l")
#adding cutpoints
abline(v=knots,lty=2,col='dodgerblue')
```




```r
# smoothing spline
fit.spline.1 = smooth.spline(Na,RI,df = 16)
plot(Na,RI,col="grey",xlab="Na",ylab="RI")
points(Na.grid,predict(fit.spline,newdata=list(Na=Na.grid)),col="darkgreen",l
wd=2,type="l")
# adding cut points
abline(v = knots,lty = 2,col = "dodgerblue")
lines(fit.spline.1,col="red",lwd=2)
legend("topright",c('Smoothing Spline with DOF=16','Cubic Spline'),col = c('r
ed','darkgreen'),lwd = 2)
```




```
fit.spline.2 = smooth.spline(Na,RI,cv = TRUE)
fit.spline.2

## Call:
## smooth.spline(x = Na, y = RI, cv = TRUE)
##
## Smoothing Parameter  spar= 1.052077  lambda= 0.006681436 (11 iterations)
## Equivalent Degrees of Freedom (Df): 4.764399
## Penalized Criterion (RSS): 0.001204002
## PRESS(l.o.o. CV): 7.84908e-06

#It selects $\lambda=0.006579777 $ and df = 4.781314 as it is a Heuristic and
 can take various values for how rough the function is
plot(Na,RI,col="grey",xlab="Na",ylab="RI")
points(Na.grid,predict(fit.spline,newdata=list(Na=Na.grid)),col="darkgreen",l
wd=2,type="l")
# adding cut points
abline(v = knots,lty = 2,col = "dodgerblue")
lines(fit.spline.1,col="red",lwd=2,lty=4)
lines(fit.spline.2,col="orange",lwd=2,lty=5)
legend("topright",c('Smoothing Spline with DOF=16','Cubic Spline','Smoothng S
plines with DOF=4.78 selected by CV'),col = c('red','darkgreen','orange'),lwd
 = 2)
```


Smoothing Spline with DOF=16
Cubic Spline
Smoothng Splines with DOF=4.78 selected by CV
RI
1.515
1.525
Na
11
12
13
14
15
16
17