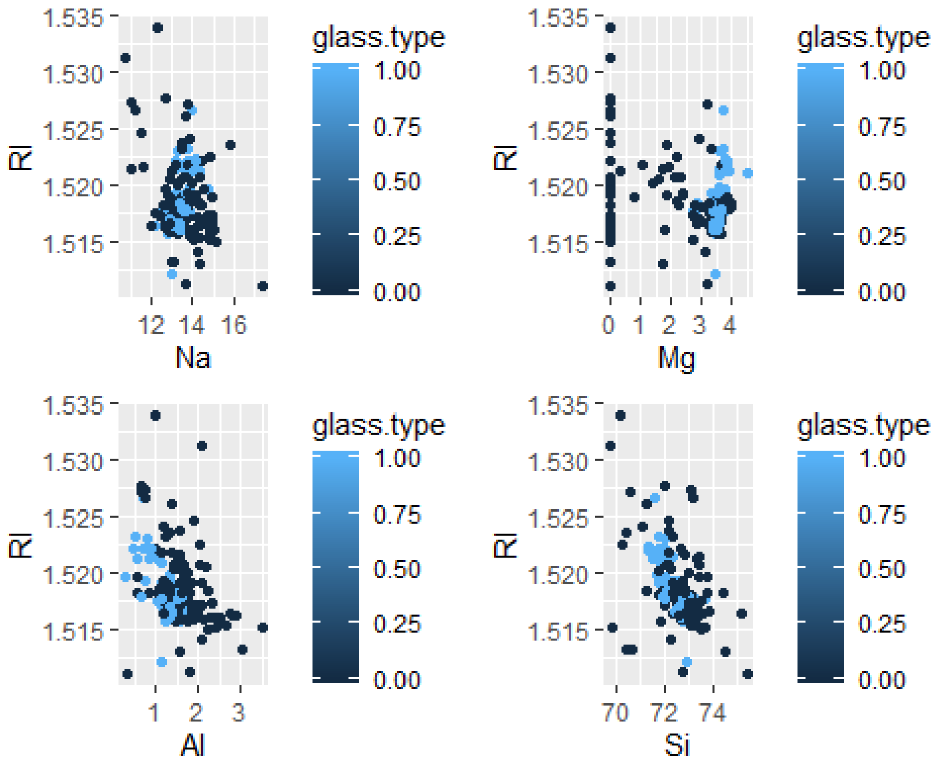
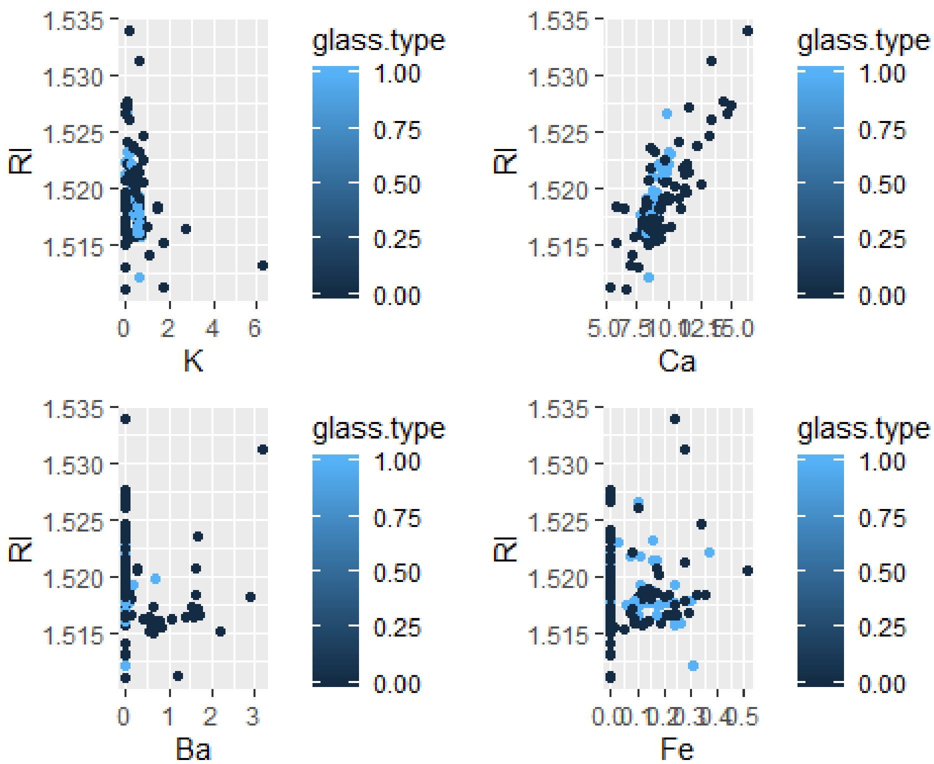
Appendix

Zijing Gao

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

glass$Type[glass$Type == 3] = 1
glass$Type[glass$Type == 5] = 0
glass$Type[glass$Type == 2] = 0
glass$Type[glass$Type == 6] = 0
glass$Type[glass$Type == 7] = 0
glass.type = glass$Type
glass.id = glass$id
glass = glass[,c("RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type")]

head(glass)

##        RI    Na   Mg   Al    Si    K   Ca Ba   Fe Type
## 1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00    1
## 2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
## 3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
## 4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
## 5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
## 6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1

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)
```

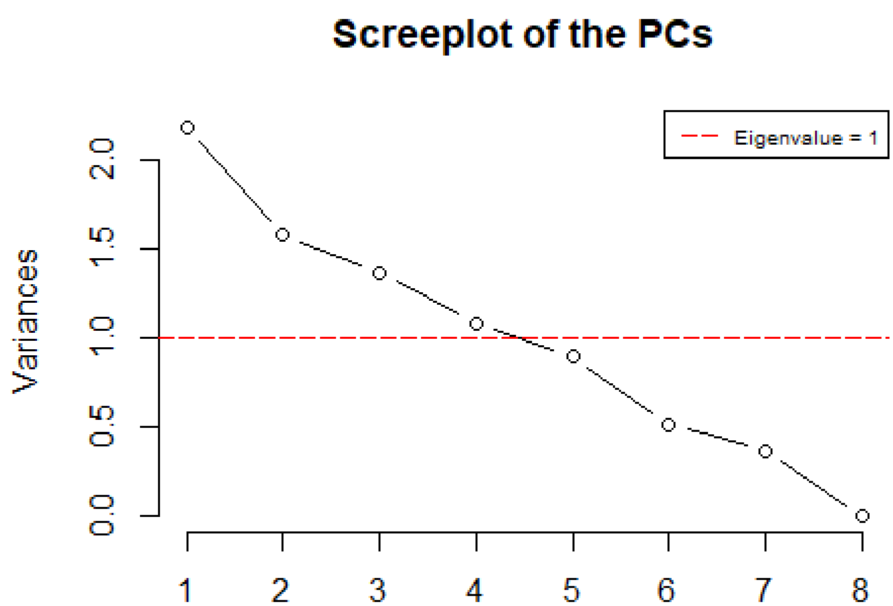



```r
grid.arrange(p5, p6,p7,p8, nrow = 2, ncol=2)
```

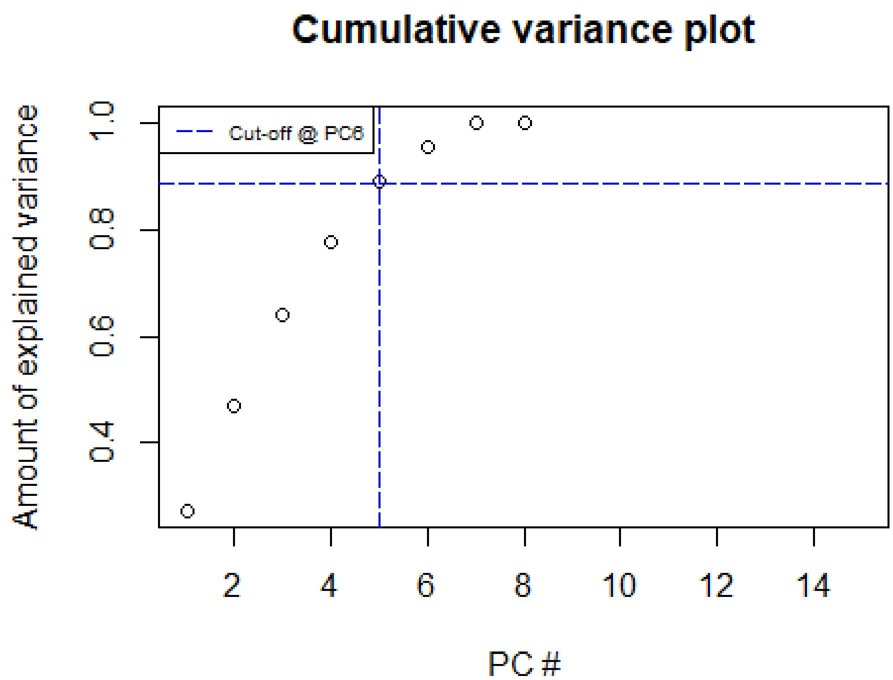


PCA

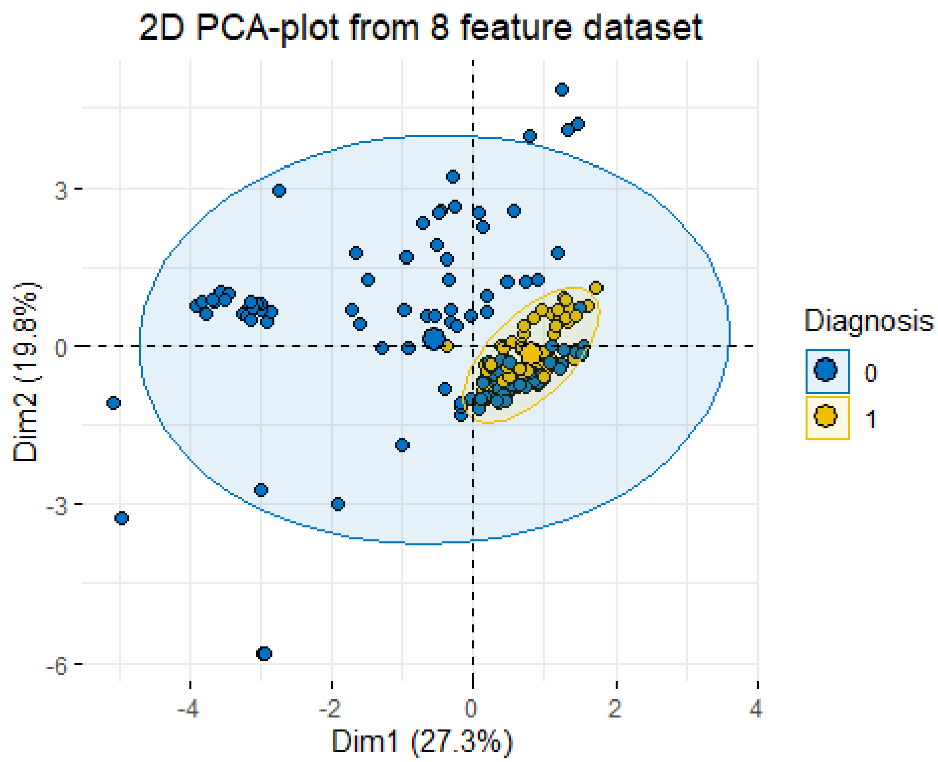
```
glass.pr = prcomp(glass[,c(2:9)], center = TRUE, scale = TRUE)
summary(glass.pr)

## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5     PC6     PC7
## Standard deviation     1.4785 1.2575 1.1688 1.0421 0.9484 0.71757 0.60389
## Proportion of Variance 0.2732 0.1976 0.1708 0.1357 0.1124 0.06436 0.04559
## Cumulative Proportion  0.2732 0.4709 0.6417 0.7774 0.8898 0.95420 0.99979
##                            PC8
## Standard deviation     0.04098
## 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)
```



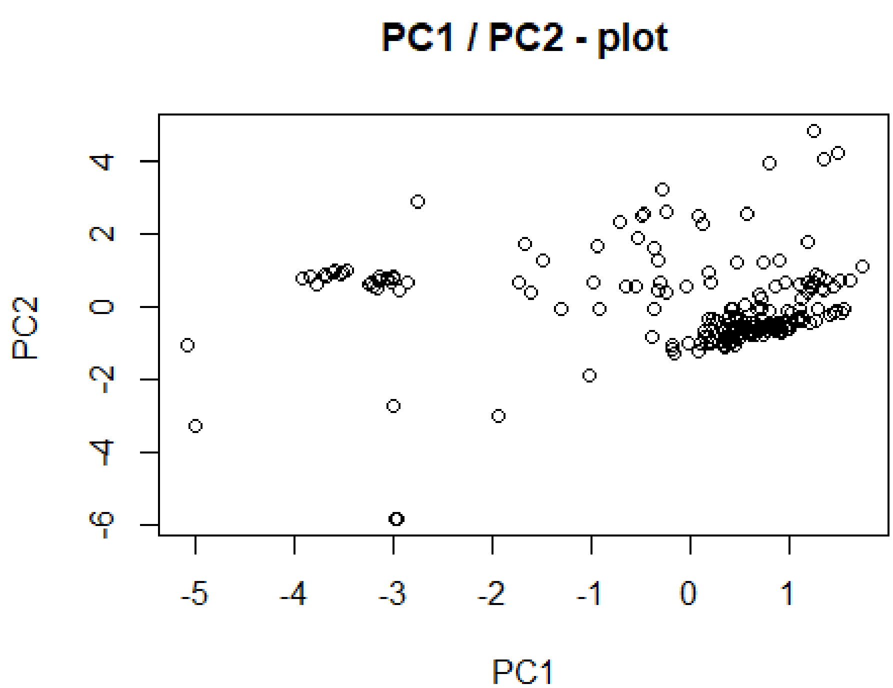

```
cumpro <- cumsum(glass.pr$sdev^2 / sum(glass.pr$sdev^2))
plot(cumpro[0:15], xlab = "PC #", ylab = "Amount of explained variance", main
 = "Cumulative variance plot")
abline(v = 5, col="blue", lty=5)
abline(h = 0.88759, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC6"),
       col=c("blue"),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




```
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))
```

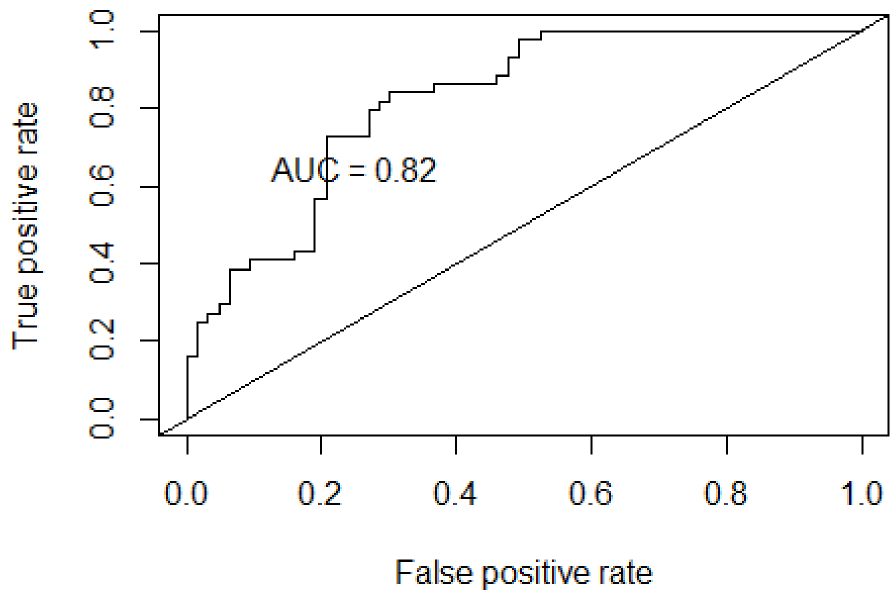



```
plot(glass.pr$x[,1],glass.pr$x[,2], xlab="PC1", ylab = "PC2 ", main = "PC1 / PC2 - plot")
```

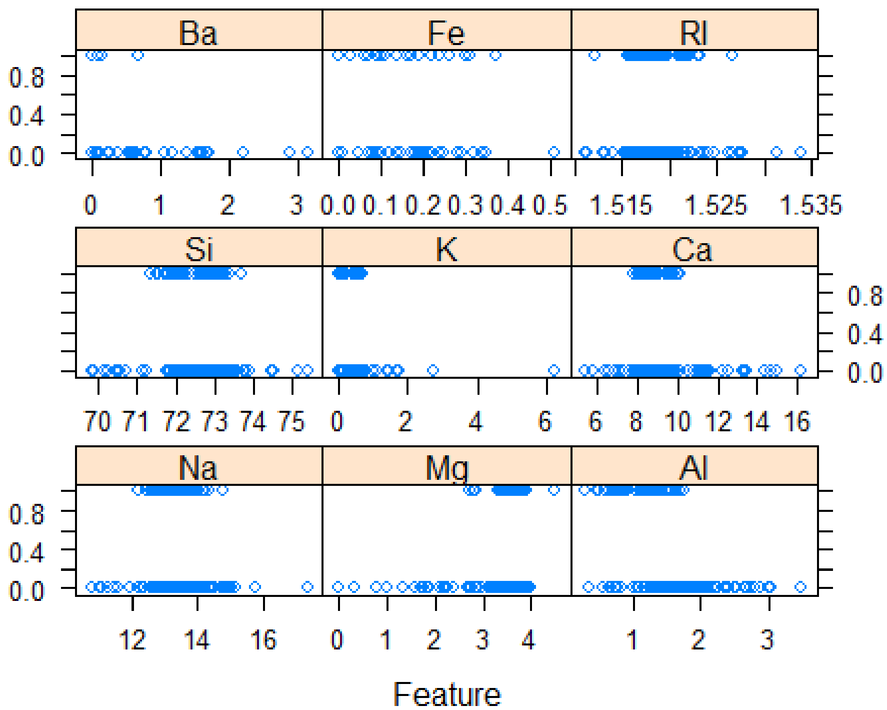


LDA

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

set.seed(9)
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)

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



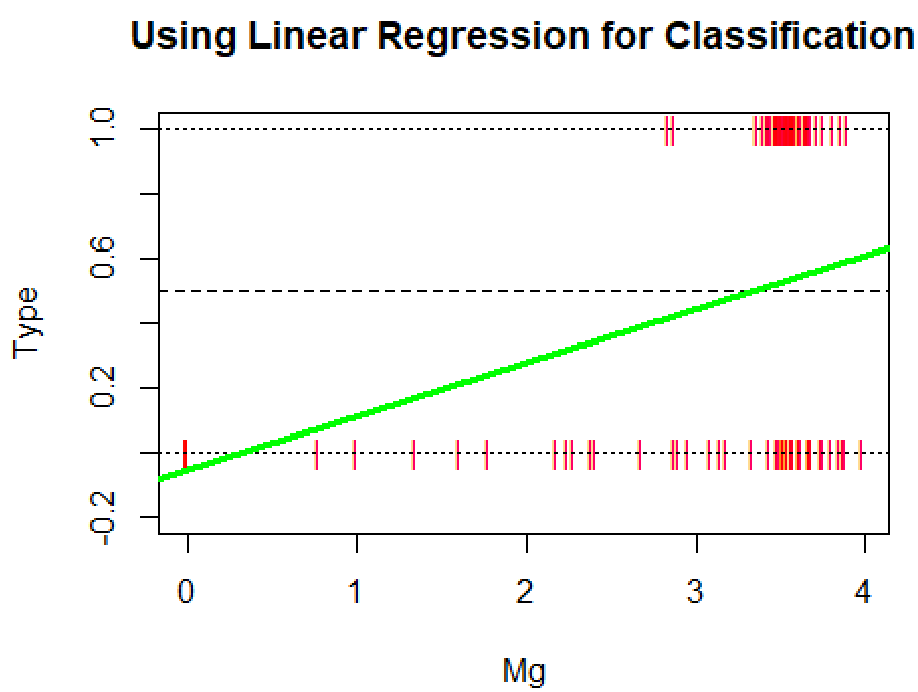

```r
library(caret)

## Loading required package: lattice

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



LR for classification

```r
# Again, we split the data
set.seed(5)

glass_idx = sample(nrow(glass), size = trunc(0.50 * nrow(glass)))
glass_trn = glass[glass_idx,]
glass_tst = glass[-glass_idx,]

model_lm = lm(Type~Mg,data = glass_trn)
mean((model_lm$residuals)^2)

## [1] 0.1841587

#+Na+Mg+Al+Si+K+Ca+Ba+Fe

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

## RI --> 0.2380162
## Na --> 0.2251071
## Mg --> 0.1841587
## Al --> 0.1947701
## Si --> 0.2379773
## K --> 0.2380082
```

```
## Ca --> 0.2357313
## Ba --> 0.2235814
## Fe --> 0.2383037

# 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")
abline(h = 0, lty = 3)
abline(h = 1, lty = 3)
abline(h = 0.5, lty = 2)
abline(model_lm, lwd = 3, col = "green")
```



This model is not bad since the distribution of glass type is balanced.

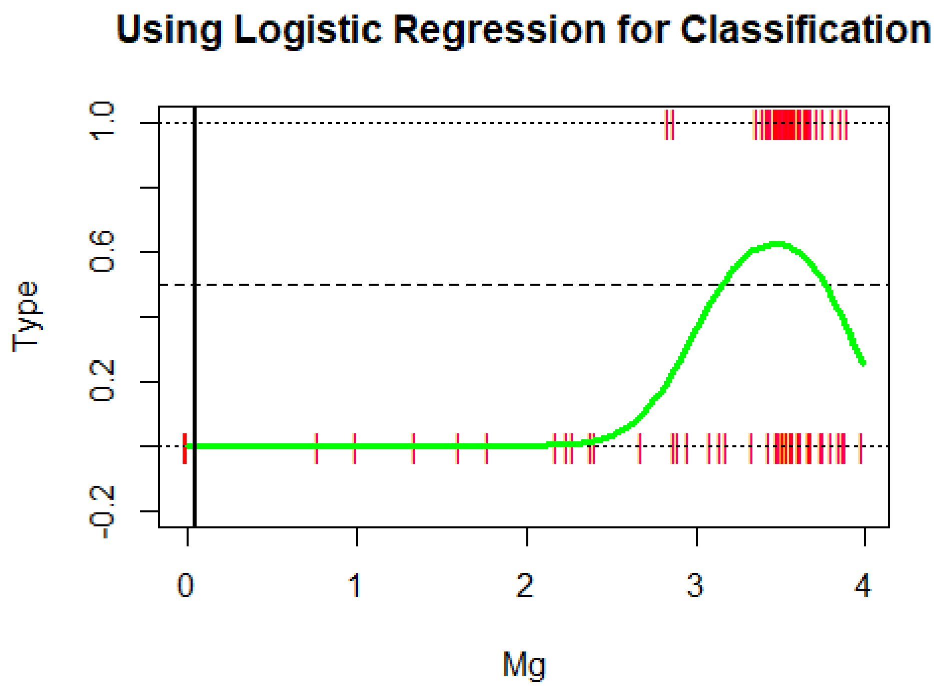
glm

```
# LOOCV
library(boot)

##
## Attaching package: 'boot'

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

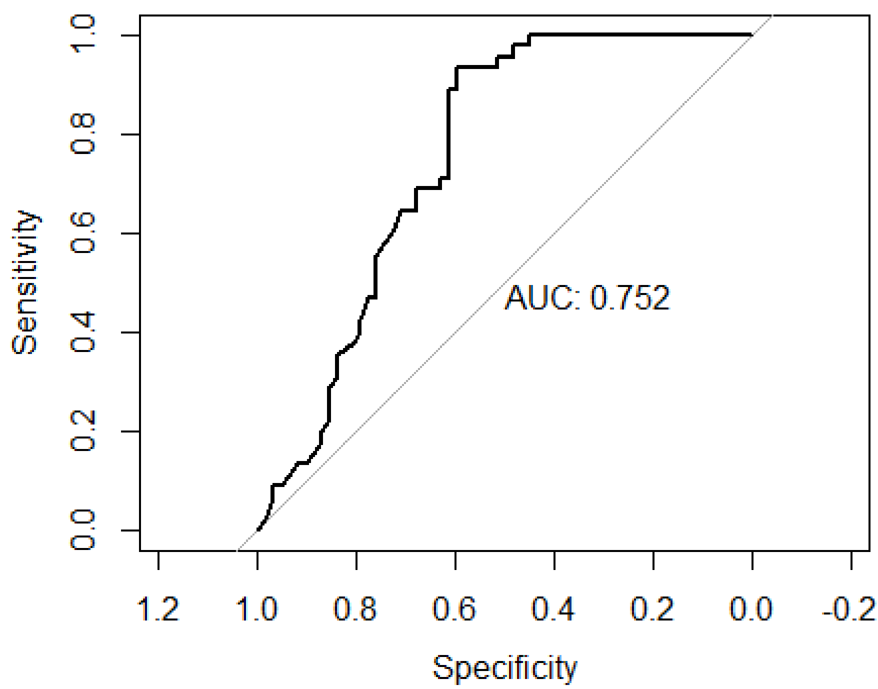
```
cv.error = rep(0,5)
for(i in 1:5){
  glm.fit = glm(Type~poly(Mg,i),data = glass_trn, family = "binomial")
  cv.error[i] = cv.glm(glass_trn,glm.fit)$delta[1]
}

result = data.frame(polynomial = seq(1,5),
                    cv.error = cv.error)

result

##   polynomial  cv.error
## 1          1 0.1868940
## 2          2 0.1836243
## 3          3 0.1824162
## 4          4 0.1866311
## 5          5 0.2001998

print(result[,1][min(cv.error) == cv.error])

## [1] 3

polynomial = max(result[,1][min(cv.error) == cv.error])
polynomial

## [1] 3

model_glm = glm(Type ~ poly(Mg,polynomial), data = glass_trn, family = "binom
ial")

model_glm_pred = ifelse(predict(model_glm, type = "response")>0.5,1,0)
```

We can calculate metrics such as te error rate.

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

cal_class_err(actual = glass_trn$Type, predicted = model_glm_pred)

## [1] 0.271028

plot(Type ~ Mg, data = glass_trn,
     col = "red", pch = "|", ylim = c(-0.2, 1),
     main = "Using Logistic Regression for Classification")
abline(h = 0, lty = 3)
abline(h = 1, lty = 3)
abline(h = 0.5, lty = 2)
curve(predict(model_glm, data.frame(Mg = x), type = "response"),
      add = TRUE, lwd = 3, col = "green")
abline(v = -coef(model_glm)[1] / coef(model_glm)[2], lwd = 2)
```




```
library(pROC)

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

##
## Attaching package: 'pROC'

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

test_prob = predict(model_glm, newdata = glass_tst, type = "response")
test_roc = roc(glass_tst$Type ~ test_prob, plot = TRUE, print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
as.numeric(test_roc$auc)

## [1] 0.7519713

# k-fold
cv.error.10 = rep(0,10)
for(i in 1:10){
  glm.fit = glm(Type ~ poly(Mg,i),data =glass_trn)
  cv.error.10[i] = cv.glm(glass_trn, glm.fit, K=10)$delta[1]
}

result = data.frame(polynomial = seq(1,10),
                    cv.error = cv.error)
result

##    polynomial  cv.error
## 1           1 0.1868940
## 2           2 0.1836243
## 3           3 0.1824162
## 4           4 0.1866311
## 5           5 0.2001998
## 6           6 0.1868940
## 7           7 0.1836243
## 8           8 0.1824162
## 9           9 0.1866311
## 10         10 0.2001998
```

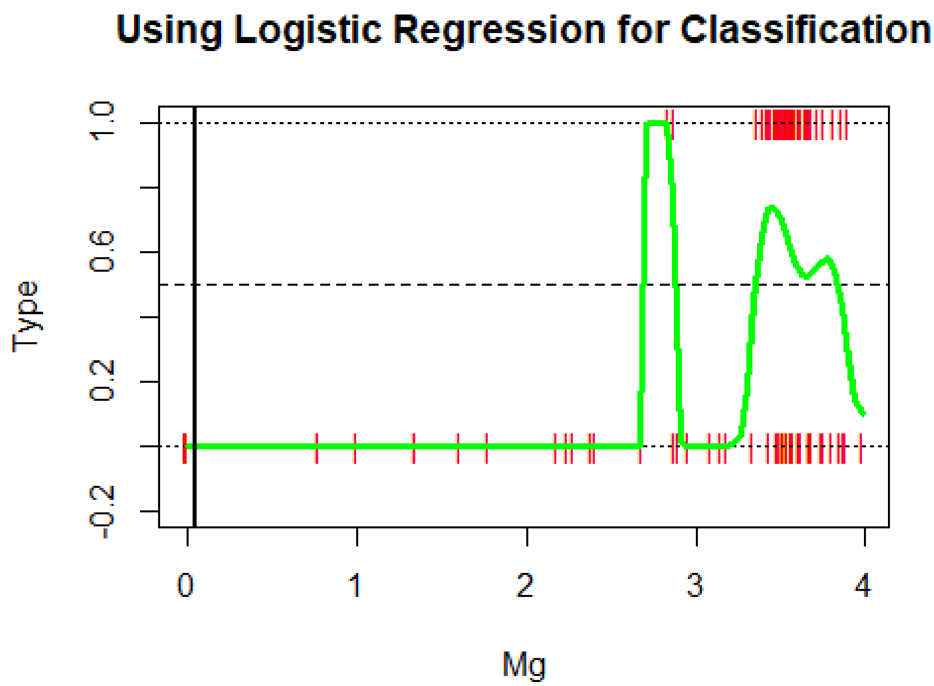
```
print(result[,1][min(cv.error) == cv.error])

## [1] 3 8

polynomial = max(result[,1][min(cv.error) == cv.error])
polynomial

## [1] 8
```

So, we take the polynomial to be 6 so that we can reduce the varibility and have least chance to overfit. Let's see what will happen.

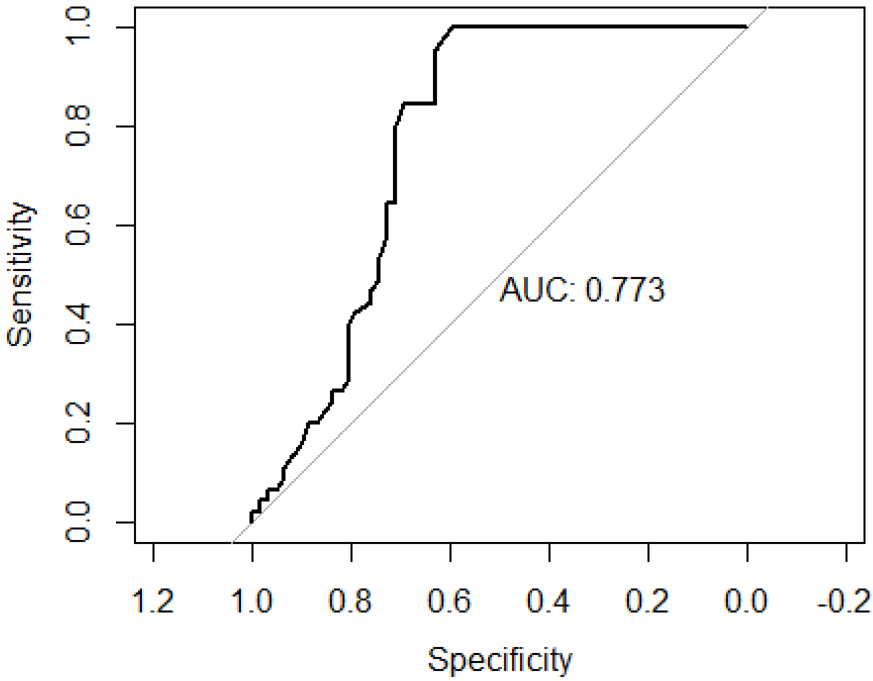
```
model_glm = glm(Type ~ poly(Mg,polynomial), data = glass_trn, family = "binom
ial")

model_glm_pred = ifelse(predict(model_glm, type = "response")>0.5,1,0)
```

We can calculate metrics such as te error rate.

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

cal_class_err(actual = glass_trn$Type, predicted = model_glm_pred)

## [1] 0.2336449

plot(Type ~ Mg, data = glass_trn,
     col = "red", pch = "|", ylim = c(-0.2, 1),
     main = "Using Logistic Regression for Classification")
abline(h = 0, lty = 3)
abline(h = 1, lty = 3)
abline(h = 0.5, lty = 2)
curve(predict(model_glm, data.frame(Mg = x), type = "response"),
      add = TRUE, lwd = 3, col = "green", )
abline(v = -coef(model_glm)[1] / coef(model_glm)[2], lwd = 2)
```




```
library(pROC)
test_prob = predict(model_glm, newdata = glass_tst, type = "response")
test_roc = roc(glass_tst$Type ~ test_prob, plot = TRUE, print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
as.numeric(test_roc$auc)

## [1] 0.7734767
```

Since the AUC is not very high, I assume the model needs improved to get a higher sensitivity and specificity.

```
# rmse = function(actual, predicted) {
#   sqrt(mean((actual - predicted) ^ 2))
# }
#
#
# get_rmse = function(model, data, response) {
#   rmse(actual = subset(data, select = response, drop = TRUE),
#        predicted = predict(model, data))
# }
#
# get_complexity = function(model) {
#   length(coef(model)) - 1
# }
```

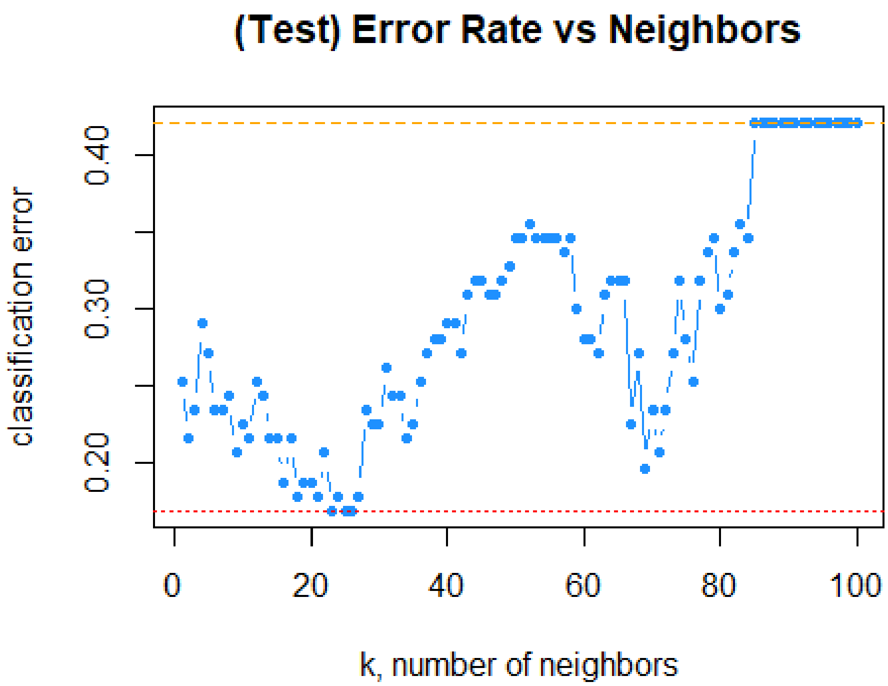
KNN

```
library(ISLR)
library(class)

# training data
X_glass_trn = glass_trn[,1:9]
```

```
y_glass_trn = glass_trn$Type


# testing data
X_glass_tst = glass_tst[,1:9]
y_glass_tst = glass_tst$Type

model_knn = knn(train = scale(X_glass_trn), test = scale(X_glass_tst), cl = y
_glass_trn, k = 3)
model_knn

##   [1] 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 1
1 1 1
##  [38] 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0
0 1 1
##  [75] 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1

cal_class_err(actual = y_glass_tst, predicted = model_knn)

## [1] 0.2336449
```

Then, I try to choose k.

```
set.seed(42)
k_to_try = 1:100
err_k = rep(x = 0, times = length(k_to_try))

for (i in seq_along(k_to_try)) {
  pred = knn(train = scale(X_glass_trn),
             test  = scale(X_glass_tst),
             cl    = y_glass_trn,
             k     = k_to_try[i])
  err_k[i] = cal_class_err(actual = y_glass_tst, predicted = pred)
}

# 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_glass_tst == 1), col = "orange", lty = 2)
```




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

## [1] 23 25 26

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

In this case, we choose k = 26 since the largest one is the least variable, and has the least chance of overfitting.

```
table(y_glass_tst)

## y_glass_tst
##  0  1
## 62 45

knn.result = knn(train = scale(X_glass_trn), test = scale(X_glass_tst), cl =
y_glass_trn, k = k.best)
knn.result

##   [1] 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1
0 1 0
##  [75] 0 1 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1

table(knn.result)
```

```
## knn.result
##  0  1
## 54 53
```