SLLD - Module 1

Classification

S.Tonini, F.Chiaromonte

Sant'Anna School of Advanced Study - Pisa

14/2/2024

Libraries

```r
library(dplyr)
library(caret)
library(MASS)
library(klaR)
library(mvtnorm)
```

Data

We simulate data as follows

```r
set.seed(123)
n <- 500
p <- 5
X <- matrix(rnorm(n*p), nrow = n, ncol = p)
X <- data.frame(X)
colnames(X) <- paste("X", 1:p, sep="")
eta<-0.5-1*X[,1]-2*X[,2]+X[,3]+3*X[,4]+0.5*X[,5]
piy<-(exp(eta))/(1+exp(eta))
mean(piy)
```

```
## [1] 0.5653752
```

```r
Y<-rbinom(n, 1, piy)
summary(Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00    1.00    0.56    1.00    1.00
```

```r
df<-cbind(Y,X)

set.seed(123)
training_samples <- df$Y %>%
  caret::createDataPartition(p = 0.8, list = FALSE)
# p indicates the percentage of training data
train <- df[training_samples, ]
test  <- df[-training_samples, ]
```

Logistic Regression

Simple Logistic Regression

The **glm()** function can be used to fit many types of generalized linear models, including logistic regression.

It is similar to **lm()**, except that we must linear model pass in the argument **family = binomial** in order to tell R to run a logistic regression rather than some other type of generalized linear model.

Using the training set, we build a simple logistic regression model using X_1 as the only explanatory variable of the response Y

```r
simple_glm <- glm(Y ~ X1,
                  data = train, family = 'binomial')
summary(simple_glm)
```

```
##
## Call:
## glm(formula = Y ~ X1, family = "binomial", data = train)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.2409     0.1028   2.343   0.0191 *
## X1           -0.4338     0.1109  -3.910 9.22e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 549.22  on 399  degrees of freedom
## Residual deviance: 533.02  on 398  degrees of freedom
## AIC: 537.02
##
## Number of Fisher Scoring iterations: 4
```

Let's see predictive power of our model in terms of **accuracy** -- i.e. the proportion of correct predictions, both true positives and true negatives, among the total number of cases examined

```
# Test for accuracy: predict test data
predict_1 <- predict(simple_glm, newdata = test,type = 'response')

# round up the predictions
predict_1 <- ifelse(predict_1>0.5, 1, 0)

# calculate accuracy
accuracySim <- mean(predict_1==test$Y)
accuracySim
```

```
## [1] 0.71
```

```
ConfMat_SR = confusionMatrix(as.factor(predict_1),
                             as.factor(test$Y))
ConfMat_SR$table
```

```
##           Reference
## Prediction  0  1
##          0 22  8
##          1 21 49
```

Note: the "event" is 0!

Multiple Logistic Regression

```r
glm_complete <- glm(Y ~ ., data=train, family = 'binomial')
summary(glm_complete)
```

```
##
## Call:
## glm(formula = Y ~ ., family = "binomial", data = train)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.5152     0.1759   2.929  0.00340 **
## X1           -1.2380     0.2220  -5.577 2.44e-08 ***
## X2           -2.4090     0.2807  -8.582  < 2e-16 ***
## X3            1.0528     0.1935   5.441 5.29e-08 ***
## X4            3.0775     0.3355   9.173  < 2e-16 ***
## X5            0.5673     0.1755   3.232  0.00123 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 549.22  on 399  degrees of freedom
## Residual deviance: 231.04  on 394  degrees of freedom
## AIC: 243.04
##
## Number of Fisher Scoring iterations: 6
```

Let's see its predictive accuracy

```r
# Test for accuracy: predict test data
predict_1 <- predict(glm_complete, newdata =
                       test,type = 'response')

# round up the predictions
predict_1 <- ifelse(predict_1>0.5, 1, 0)

# calculate accuracy
accuracySat <- mean(predict_1==test$Y)
accuracySat
```

```
## [1] 0.92
```

```r
ConfMat_LR = confusionMatrix(as.factor(predict_1),
                             as.factor(test$Y))
ConfMat_LR$table
```

```
##           Reference
## Prediction  0  1
##          0 38  3
##          1  5 54
```

LDA and QDA

We use the function **lda** to perform linear discriminant analysis

```
lda.Y <- lda(Y~ ., data=train)
lda.Y
```

```
## Call:
## lda(Y ~ ., data = train)
##
## Prior probabilities of groups:
##      0      1
## 0.4425 0.5575
##
## Group means:
##           X1         X2         X3         X4          X5
## 0  0.2168210  0.4320500 -0.1111468 -0.4935911 -0.20301669
## 1 -0.1689612 -0.3024828  0.1179516  0.5367388  0.04018593
##
## Coefficients of linear discriminants:
##           LD1
## X1 -0.4173204
## X2 -0.8728278
## X3  0.4242211
## X4  1.1153619
## X5  0.2349530
```

Here is the model accuracy on training data.

```
predmodel.train.lda = predict(lda.Y, data=train_transformed)
ConfMat_LDAtrain = confusionMatrix(as.factor(
  predmodel.train.lda$class),
                              as.factor(train$Y))
ConfMat_LDAtrain$table
```

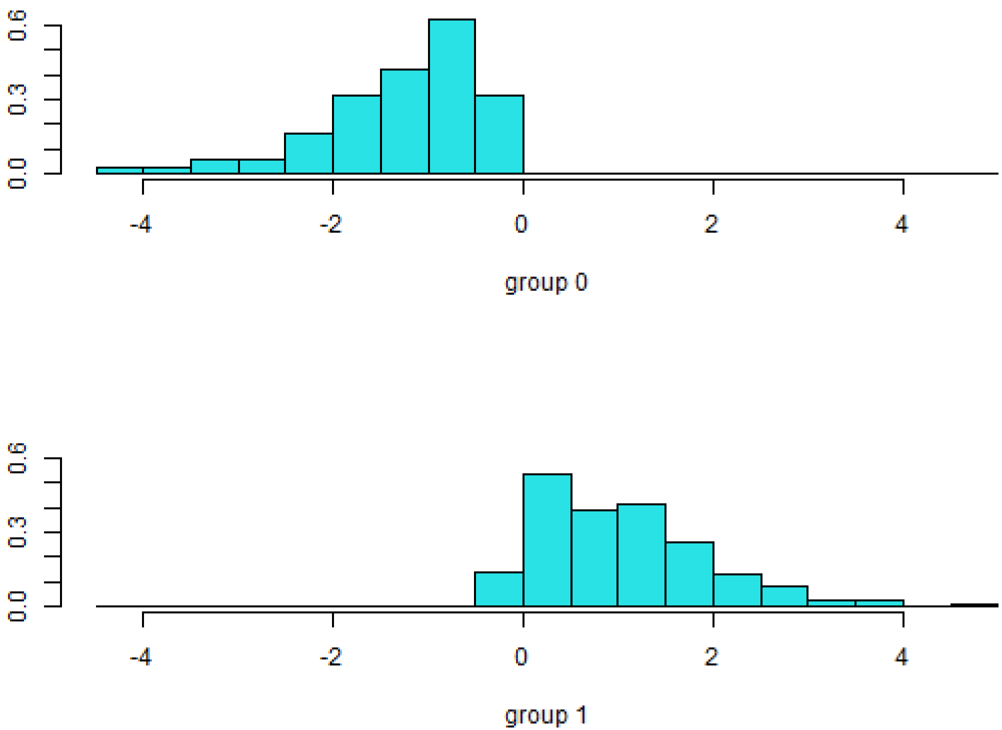
```
##           Reference
## Prediction   0   1
##          0 150  32
##          1  27 191
```

```
ConfMat_LDAtrain$byClass[1:2]
```

```
## Sensitivity Specificity
##   0.8474576   0.8565022
```

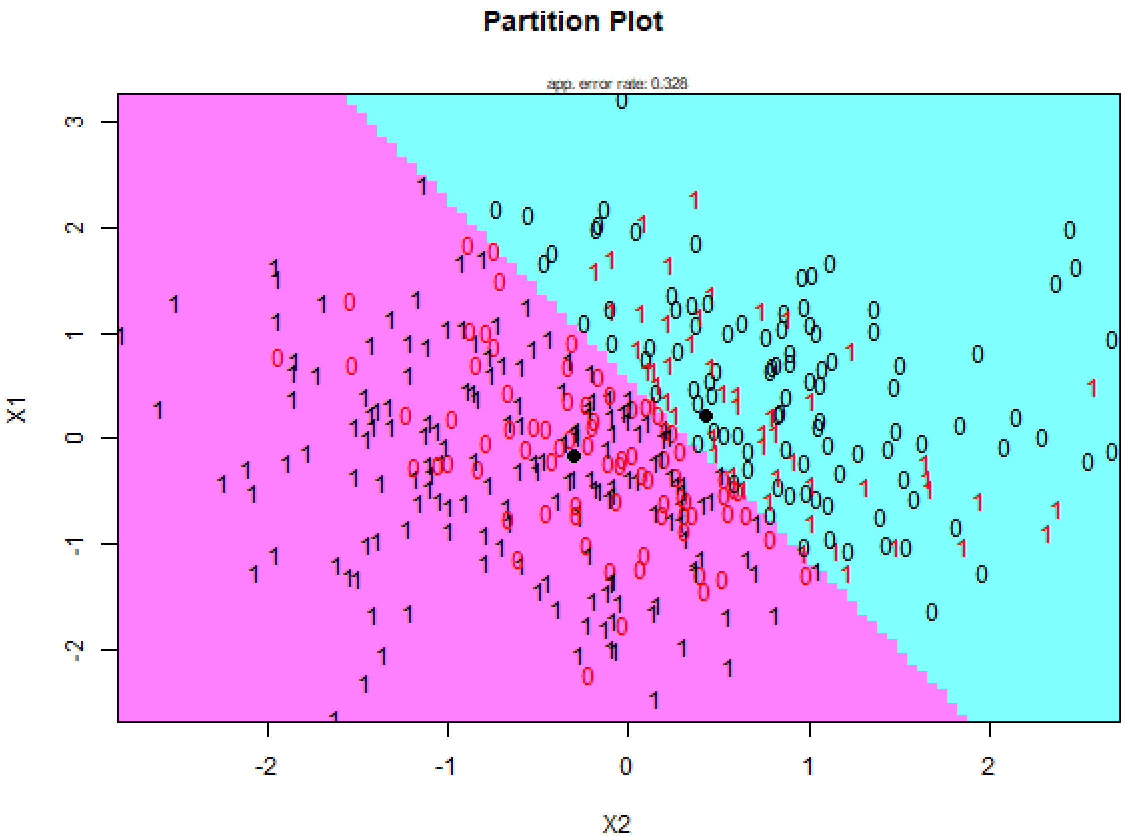
The plot below shows how the response class has been classified by the LDA classifier. The x-axis shows the value of the line defined by the coefficient of linear discriminant for LDA. Groups are the ones in the response classes.

```
ldahist(predmodel.train.lda$x[,1], g= predmodel.train.lda$class)
```



We use the function **partimat** to see geometric division

```
partimat(factor(Y)~ X1+X2,
          data=train, method = "lda")
```



Now we check the model accuracy on test data.

```
predmodel.test.lda = predict(lda.Y, newdata=test)
ConfMat_LDAtest = confusionMatrix(as.factor(
  predmodel.test.lda$class),
                  as.factor(test$Y))
ConfMat_LDAtest$table
```

```
##           Reference
## Prediction  0  1
##          0 38  4
##          1  5 53
```

```
mean(predmodel.test.lda$class==test$Y)
```

```
## [1] 0.91
```

```
ConfMat_LDAtest$byClass[1:2]
```

```
## Sensitivity Specificity
##   0.8837209   0.9298246
```

Next we will fit the model trough QDA. The command is similar to LDA and it outputs the prior probabilities and Group means. Note that "Prior Probabilities" and "Group Means" values are same as of LDA.

```
qda.Y <- qda(Y~ ., data=train)
qda.Y
```

```
## Call:
## qda(Y ~ ., data = train)
##
## Prior probabilities of groups:
##      0      1
## 0.4425 0.5575
##
## Group means:
##           X1         X2         X3         X4          X5
## 0  0.2168210  0.4320500 -0.1111468 -0.4935911 -0.20301669
## 1 -0.1689612 -0.3024828  0.1179516  0.5367388  0.04018593
```

Now we check the model accuracy on test data.

```
predmodel.test.qda = predict(qda.Y, newdata=test)
ConfMat_QDAtest = confusionMatrix(as.factor(
  predmodel.test.qda$class),
                  as.factor(test$Y))
ConfMat_QDAtest$table
```

```
##           Reference
## Prediction  0  1
##          0 37  4
##          1  6 53
```

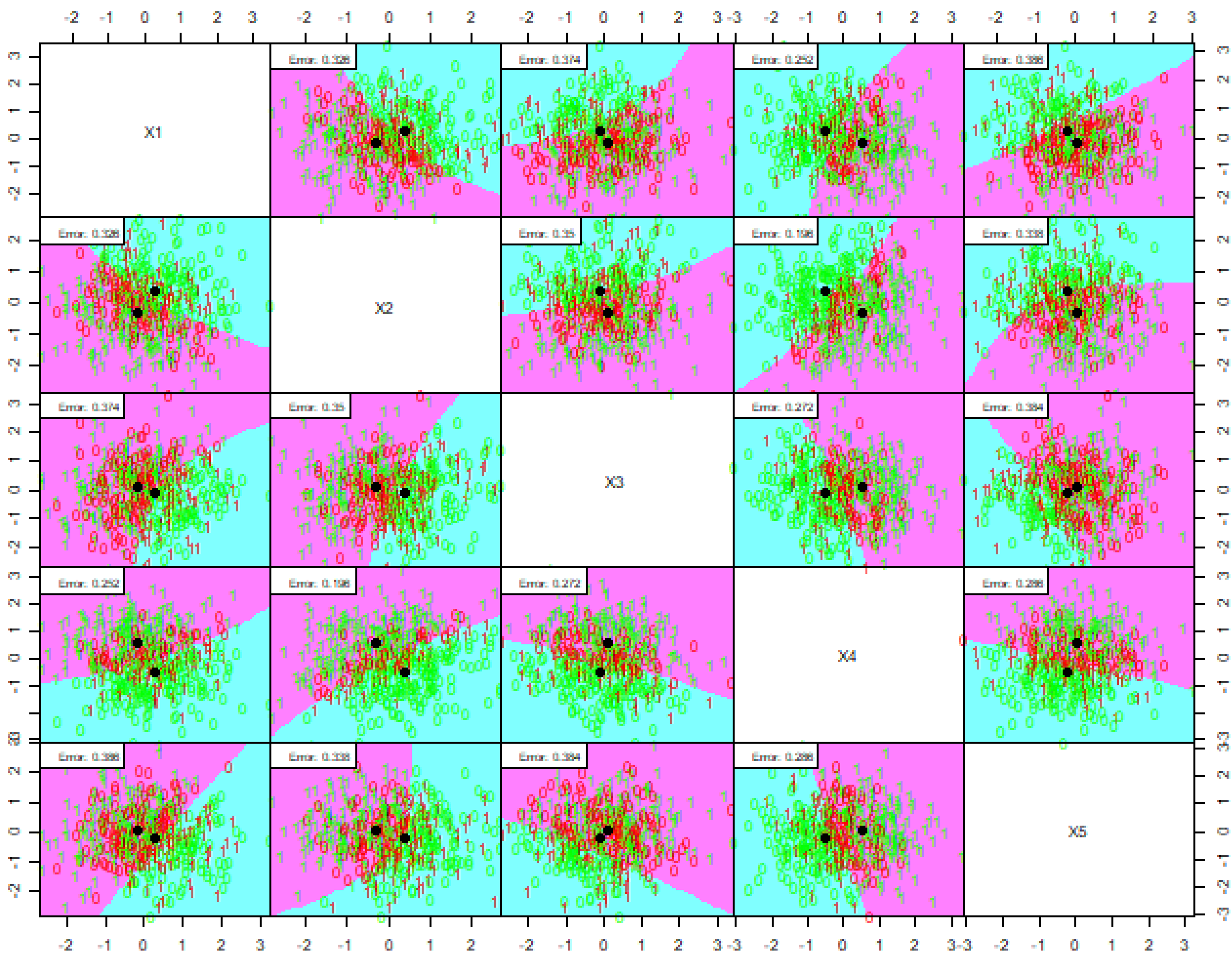
```
mean(predmodel.test.qda$class==test$Y)
```

```
## [1] 0.9
```

```
ConfMat_QDAtest$byClass[1:2]
```

```
## Sensitivity Specificity
##   0.8604651   0.9298246
```

```
partimat(factor(Y) ~ .,
data=df, method = "qda", plot.matrix=TRUE,
col.correct='green', col.wrong='red')
```



kNN

We are going to use the **knn3** function within caret package. Let's train the knn with k = 1, 10

```
knn_Y1 <- knn3(factor(Y) ~ .,data=train, k = 1)
knn_Y10 <- knn3(factor(Y) ~ .,data=train, k = 10)
predict_knn1 <- predict(knn_Y1, test, type='class')
ConfMat_knn1 = confusionMatrix(predict_knn1, as.factor(test$Y))
ConfMat_knn1$overall[1] # the accuracy
```

```
## Accuracy
##     0.83
```

```
predict_knn10 <- predict(knn_Y10, test, type='class')
ConfMat_knn10 = confusionMatrix(predict_knn10, as.factor(test$Y))
ConfMat_knn10$overall[1] # the accuracy
```

```
## Accuracy
##     0.86
```

Now it's your turn!!!