# COMP/INDR 421/521 HW01: Multivariate Parametric Classification

*Asma Hakouz 0063315*

October 20, 2017

The purpose of this assignment was to implement a multivariate parametric classification algorithm in R.

The problem consisted of four main parts:

1- **Initialization**
2- **Data generation**
3- **Classification Algorithm:** Where the multivariate parametric classification algorithm is applied on the data set generated in the first part.
4- **Algorithm performance evaluation**

Following, more details will be discussed about each part accompanied by snippets from my source code.

- ## Initialization

As can be seen in the previous code snippet, actual class parameters are initialized for all three classes.

$$class_i\_means = [\mu_{i1} \quad \mu_{i2}]$$

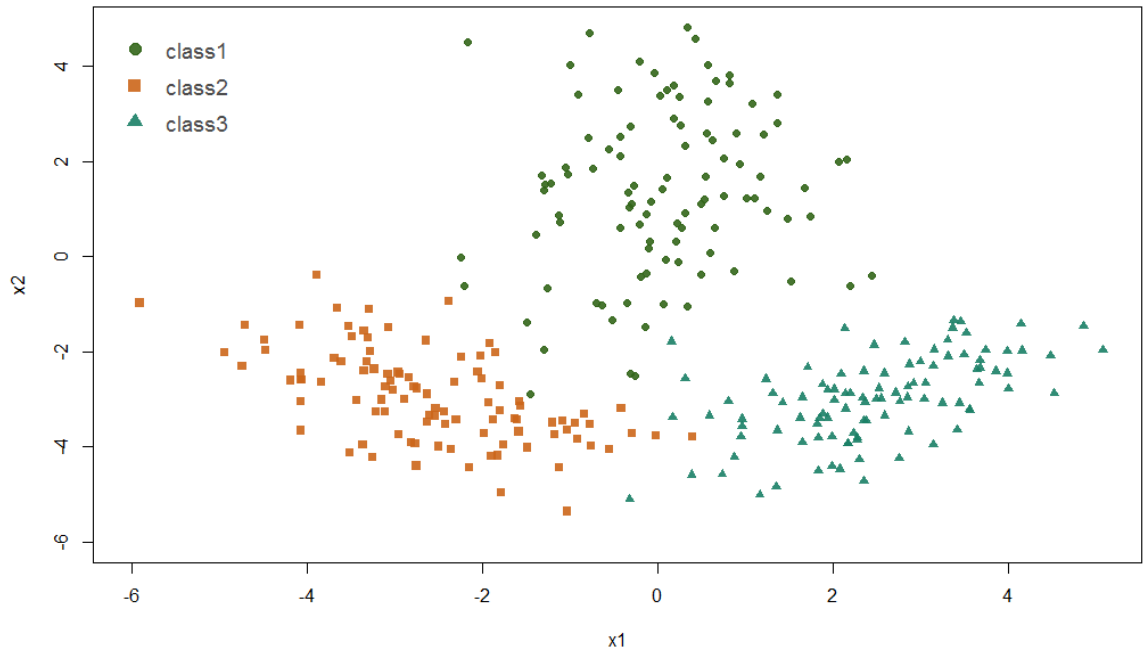
```
library(MASS)
## GENERATING DATA
# choose an arbitrary value for the seed which will be used for random numbers generation
set.seed(401)

# define classes parameters, each with 2 inputs(features); x1 & x2
# mean parameters
class1_means <- c(0.0, 1.5)
class2_means <- c(-2.5, -3.0)
class3_means <- c(2.5, -3.0)

# Coveriance matrices
class1_sigma <- matrix(c(1, 0.2, 0.2, 3.2), 2, 2)
class2_sigma <- matrix(c(1.6, -0.8, -0.8, 1.0), 2, 2)
class3_sigma <- matrix(c(1.6, 0.8, 0.8, 1.0), 2, 2)

# sample sizes
class_sizes <- c(100, 100, 100)
```

$$covariance\ matrix\ \boldsymbol{\Sigma} \equiv class_i\_sigma = \begin{bmatrix} {\sigma_{i1}}^2 & \sigma_{i12} \\ \sigma_{i21} & {\sigma_{i2}}^2 \end{bmatrix}$$

- **Data generation**

```r
# generate random samples from multivariate (in our case it's bivariate) normal distributions
points1 <- MASS::mvrnorm(n = class_sizes[1], class1_means, class1_sigma)
points2 <- MASS::mvrnorm(n = class_sizes[2], class2_means, class2_sigma)
points3 <- MASS::mvrnorm(n = class_sizes[3], class3_means, class3_sigma)

# plot the generated data points from all classes.
plot(points1[,1], points1[,2], type = "p", col = rgb(0.2,0.4,0.1,0.9), lwd = 0.5,
     xlab = "x1", ylab = "x2", ylim = c(-6, max(points1[,2], points2[,2], points3[,2])),
     xlim = c(-6, max(points1[,1], points2[,1], points3[,1])), pch = 19)
points(points2[,1], points2[,2], type = "p", col = rgb(0.8,0.4,0.1,0.9), lwd = 0.5, pch = 15)
points(points3[,1], points3[,2], type = "p", col = rgb(0.1,0.5,0.4,0.9), lwd = 0.5, pch = 17)

legend("topleft",
       legend = c("class1", "class2", "class3"),
       col = c(rgb(0.2,0.4,0.1,0.9),
               rgb(0.8,0.4,0.1,0.9),
               rgb(0.1,0.5,0.4,0.9)),
       pch = c(19, 15, 17), bty = "n", pt.cex = 1.5, cex = 1.2,
       text.col = rgb(0.3,0.3,0.3,1), horiz = F, inset = c(0.01, 0.01))


x1 <- c(points1[,1], points2[,1], points3[,1])
x2 <- c(points1[,2], points2[,2], points3[,2])

# generate corresponding labels
y <- c(rep(1, class_sizes[1]), rep(2, class_sizes[2]), rep(3, class_sizes[3]))

# write data to a file
write.csv(x = cbind(x1, x2, y), file = "HW01_data_set.csv", row.names = FALSE)
```

This part consists of generating the data set (including input/feature values and their corresponding outputs/labels) from bivariate Gaussian densities. Then, the generated data set was plotted as the following graph:



- **Classification Algorithm**

This part contains multiple steps as follows:

- Importing data and initializing algorithm parameters

```
## DATA PROCESSING AND ALGORITHM
# read data into memory
data_set <- read.csv("HW01_data_set.csv")

# get x1, x2 and y values
x1 <- data_set$x1
x2 <- data_set$x2
x <- cbind(x1, x2)
y_truth <- data_set$y

# get number of classes(K) number of samples(N),
# and number of inputs/features(d)
K <- max(y_truth)
N <- length(y_truth)
d <- ncol(x)
```

- Parameters estimation, including means, covariance matrices and classes' priors.

$$\boldsymbol{m}_c^{\star} = \frac{\sum_{i=1}^{N} \boldsymbol{x}_i \, \mathbf{1}(y_i = c)}{\sum_{i=1}^{N} \mathbf{1}\,(y_i = c)}, for\ c\ \epsilon(1,2,3)$$

```
# estimate classes means by calculating sample means
sample_means <- rbind(sapply(X = 1:K, FUN = function(c) {mean(x1[y_truth == c])}),
                      sapply(X = 1:K, FUN = function(c) {mean(x2[y_truth == c])}))
print(sample_means)
```

```
> print(sample_means)
           [,1]      [,2]      [,3]
[1,] 0.05887385 -2.580098  2.463599
[2,] 1.36023102 -2.954950 -3.005643
```

where the value in the $i^{th}$ column, $j^{th}$ row represents the estimated means of the $j^{th}$ input feature for the $i^{th}$ class.

$$\mathbf{S} = \begin{bmatrix} {s_{i1}}^2 & s_{i12} \\ s_{i21} & {s_{i2}}^2 \end{bmatrix}$$

$$where \quad s_c^{2\star} = \frac{\sum_{i=1}^{N}(x_c^i - m_c^{\star})^2}{N} \quad , \quad s_{ck} = \frac{\sum_{i=1}^{N}(x_c^i - m_c^{\star}) \cdot (x_k^i - m_k^{\star})}{N}$$

```
# estimate classes variances
sample_covariance <- array(sapply(X = 1:K, FUN = function(c) {
   matrix(c(mean((x1[y_truth == c] - sample_means[1, c])^2),
    mean((x1[y_truth == c] - sample_means[1, c])*(x2[y_truth == c] - sample_means[2, c])),
    mean((x1[y_truth == c] - sample_means[1, c])*(x2[y_truth == c] - sample_means[2, c])),
    mean((x2[y_truth == c] - sample_means[2, c])^2))}), dim=c(2,2,3))

print(sample_covariance)
# these estimations gets better when the size of the sample increases
# we can also notice that estimation of the mean is closer to the actual values
# than covariance estimation since covarience estimator is a biased estimator
# but as the sample size increases the bias will become negligable.
```

$$\hat{P}(y_i = c) = \frac{\sum_{i=1}^{N} \mathbf{1}(y_i = c)}{N}$$

```
# calculate prior probabilities
class_priors <- sapply(X = 1:K, FUN = function(c) {mean(y_truth == c)})
```

- Parametric classification

```
y_predicted <-c()
data_interval <- matrix(rbind(seq(from = -7, to = +7, by = 0.01),
                              seq(from = -7, to = +7, by = 0.01)), nrow =2)

for (i in 1:N) {
  # evaluate score functions
  max_c = 1
  max_score = -0.5 * d * log(2 * pi) - 0.5 * log(det(sample_covariance[,,1]))
          - 0.5 * (t(x[i,]) - sample_means[,1]) %*% chol2inv(chol(sample_covariance[,,1]))
          %*% t(t(x[i,]) - sample_means[,1]) + log(class_priors[1])
  for(c in 2:K){
    temp <- -0.5 * d * log(2 * pi) - 0.5 * log(det(sample_covariance[,,c]))
        - 0.5 * (t(x[i,]) - sample_means[,c]) %*% chol2inv(chol(sample_covariance[,,c]))
        %*% t(t(x[i,]) - sample_means[,c]) + log(class_priors[c])
    if (temp > max_score){
      max_score <- j
      max_c <- c
    }
  }
  y_predicted <- rbind(y_predicted, max_c)
}
```

Where the class scoring function is as follows:

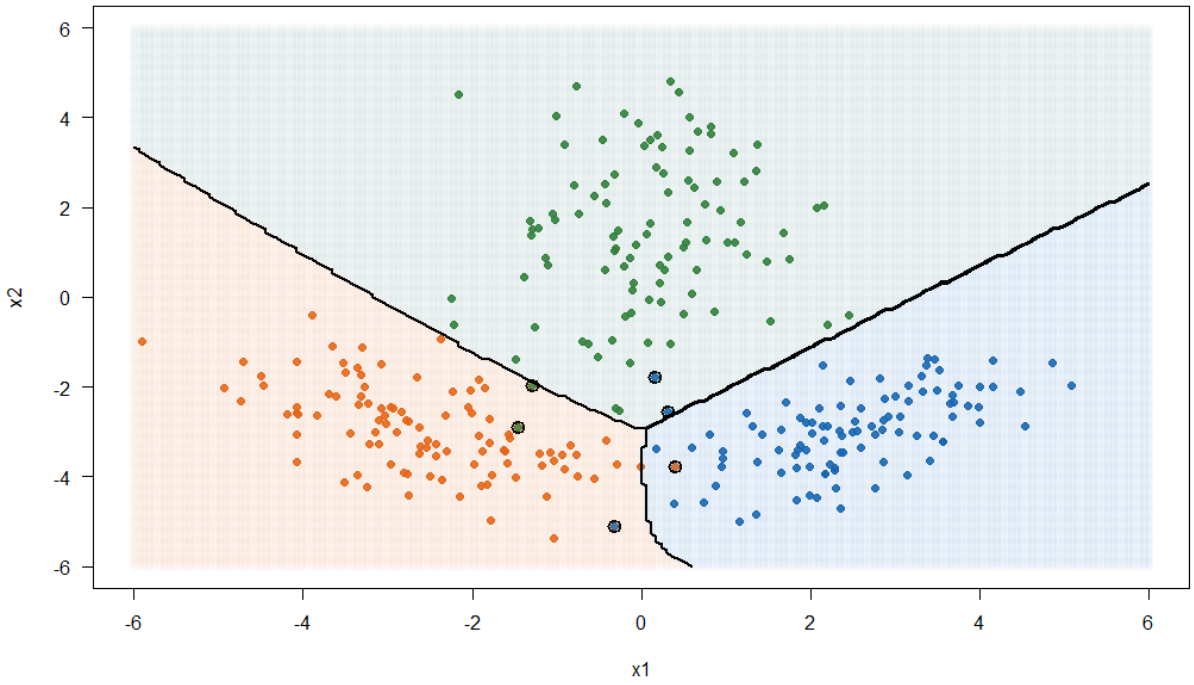
$$g_c(\boldsymbol{x}) = -\frac{1}{2}\log|\boldsymbol{\Sigma}_c| - \frac{1}{2}(\boldsymbol{x}-)$$

Then, class classification is determined as same as the best scoring class.

- The last step is to evaluate the performance of the algorithm by finding the confusion matrix and visualizing the classification by plotting the points based on their predicted labels, with the decision boundaries.

```
confusion_matrix <- table(y_predicted, y_truth)
print(confusion_matrix)
> print(confusion_matrix)
            y_truth
y_predicted  1  2  3
          1 98  0  2
          2  2 99  1
          3  0  1 97
```

```
plot(x[y_truth == 1, 1], x[y_truth == 1, 2], type = "p", pch = 19,
     col = rgb(0.2,0.5,0.2,0.9), xlim = c(-6, +6),
     ylim = c(-6, +6), xlab = "x1", ylab = "x2", las = 1)

points(x[y_truth == 2, 1], x[y_truth == 2, 2], type = "p", pch = 19,
       col = rgb(0.9,0.4,0.1,0.9))
points(x[y_truth == 3, 1], x[y_truth == 3, 2], type = "p", pch = 19,
       col = rgb(0.1,0.4,0.7,0.9))
points(x[y_predicted != y_truth, 1], x[y_predicted != y_truth, 2],
       cex = 1.5, lwd = 2)

x1_interval <- seq(from = -6, to = +6, by = 0.06)
x2_interval <- seq(from = -6, to = +6, by = 0.06)
x1_grid <- matrix(x1_interval, nrow = length(x1_interval),
                  ncol = length(x1_interval), byrow = FALSE)
x2_grid <- matrix(x2_interval, nrow = length(x2_interval),
                  ncol = length(x2_interval), byrow = TRUE)

calcG <- function(x1, x2, c){
  -0.5 * log(det(sample_covariance[,,c]))
  - 0.5 * (cbind(x1, x2) - sample_means[,c])
  %*% chol2inv(chol(sample_covariance[,,c]))
  %*% t(cbind(x1, x2) - sample_means[,c]) + log(class_priors[c])
}

f <- function(x1, x2) {
  if(max(calcG(x1, x2, 1), calcG(x1, x2, 2), calcG(x1, x2, 3)) == calcG(x1, x2, 1))
    return(1)
  if(max(calcG(x1, x2, 1), calcG(x1, x2, 2), calcG(x1, x2, 3)) == calcG(x1, x2, 2))
    return(2)
  else
    return(3)
}
discriminant_values <- matrix(mapply(f, x1_grid, x2_grid), nrow(x2_grid),
                              ncol(x2_grid))
points(x1_grid[discriminant_values == 1], x2_grid[discriminant_values == 1],
       col = rgb(0.2,0.5,0.5,0.03), pch = 16)
points(x1_grid[discriminant_values == 2], x2_grid[discriminant_values == 2],
       col = rgb(0.9,0.4,0.1,0.03), pch = 16)
points(x1_grid[discriminant_values == 3], x2_grid[discriminant_values == 3],
       col = rgb(0.1,0.4,0.7,0.03), pch = 16)
contour(x1_interval, x2_interval, discriminant_values, levels = c(3),
        add = TRUE, lwd = 2, drawlabels = FALSE)
contour(x1_interval, x2_interval, discriminant_values, levels = c(2),
        add = TRUE, lwd = 2, drawlabels = FALSE)
```


6
4
2
0
-2
-4
-6
x2
-6
-4
-2
0
2
4
6
x1