COMP/INDR 421/521 HW02: Multiclass Linear Discrimination

Asma Hakouz 0063315

October 27, 2017

The purpose of this assignment was to implement a multiclass linear discrimination algorithm in R.

The problem consisted of four main parts:

1- **Initialization**
2- **Data generation**
3- **Linear Discrimination Algorithm:** Where the multiclass linear discrimination 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.

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

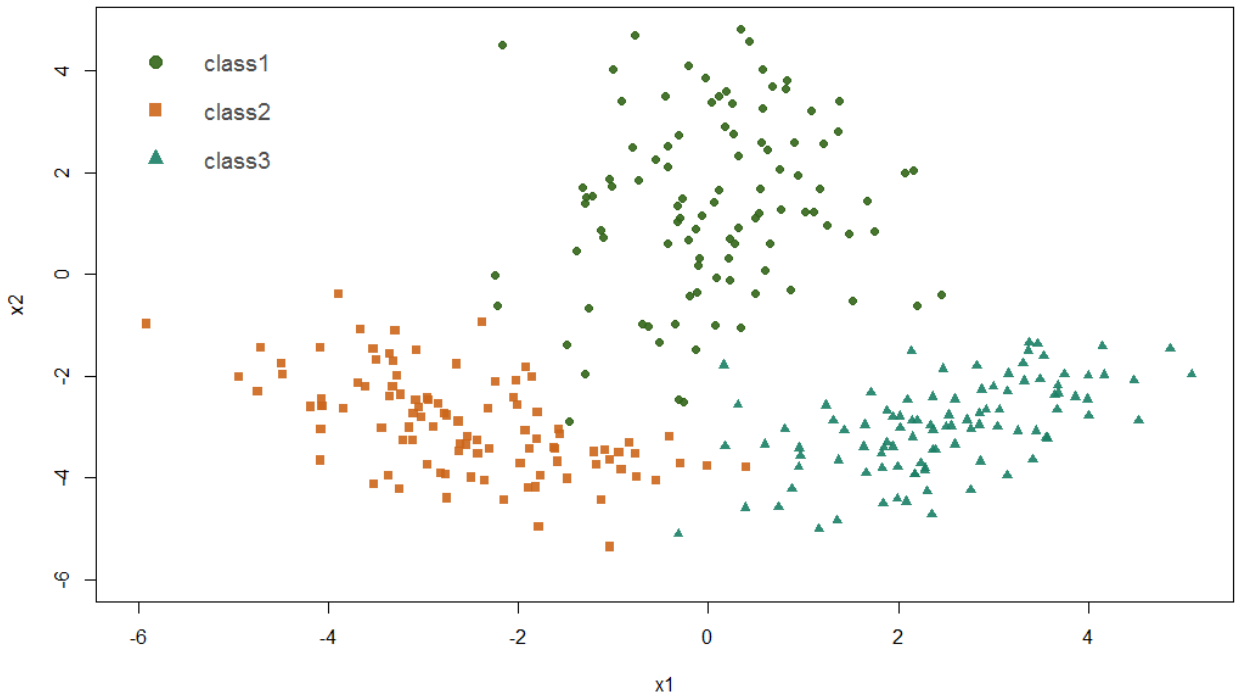
```
library(MASS)
## GENERATING DATA
# choose an arbitrary value for the seed which will be used for random numbers generation
set.seed(401)

# 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)
```

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

- **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 = "HW02_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:



- **Linear Discrimination 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("HW02_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)

# set learning parameters
eta <- 0.02
epsilon <- 1e-3

# randomly initalize w and w0
set.seed(401)
w <- matrix(runif(ncol(X)*K, min = -0.01, max = 0.01), d, K)
w0 <- matrix(runif(K, min = -0.01, max = 0.01), 1, K)

# learn w and w0 using gradient descent
iteration <- 1
objective_values <- c()
```

- Defining softmax and gradient functions

$$\Delta w_j = \eta \sum_{i=1}^{N} (r_i^i - y_i^t) \cdot x^t$$

```
# define the gradient functions
gradient_w <- function(X, y_predicted_val, y_truth) {
  delta_w <- c()
  for(c in 1:3){
    sum <- 0
    for(s in 1:300){
      if(y_truth[s] == c){
        sum <- sum + (1 - y_predicted_val[s, c]) * X[s,]
      }
      else {
        sum <- sum - y_predicted_val[s, c] * X[s,]
      }
    }
    delta_w <- cbind(delta_w, sum)
  }
  return (delta_w)
}
```

$$\Delta w_{j0} = \eta \sum_{i=1}^{N} (r_i^i - y_i^t)$$

```
gradient_w0 <- function(y_predicted_val, y_truth) {
  delta_w0 <- c()
  for(c in 1:3){
    sum <- 0
    for(s in 1:300){
      if(y_truth[s] == c){
        sum <- sum + (1 - y_predicted_val[s, c])
      } else {
        sum <- sum - y_predicted_val[s, c]
      }
    }
    delta_w0 <- cbind(delta_w0, sum)
  }
  return (delta_w0)
}
```

$$softmax \equiv y_i = \hat{p}(C_i|x) = \frac{\exp(w_i^T x + w_{i0})}{\sum_{j=1}^{K} \exp(w_j^T x + w_{j0})}$$

```
# define the softmax function
softmax <- function(X, w, w0, k, c){
  denomSum <- 0
  for(i in 1:k){
    denomSum <- denomSum + exp(X %*% w[,i] + w0[i])
  }
  return (exp(X %*% w[,c] + w0[c])/denomSum)
}
```

- Linear Discrimination
  A class is chosen if it has the maximum value of the soring function.

```
# define class predicting function
predictClass <- function(X, w, w0, k, N) {
  softmaxScore <- c()
  # calculate score functions for all classes and choose the one with the
  # highest score / probability using softmax function
  for(i in 1:k){
    softmaxScore <- cbind(softmaxScore, softmax(X, w, w0, k, i))
  }
  #print(softmaxScore)
  y_predicted <- c()
  y_predicted_probability <- c()
  y_predicted_val <- c()
  for(j in 1:N){
    maxScore <- -1
    mostProbableClass <- -1
    for(i in 1:k){
      if(softmaxScore[j,i] > maxScore){
        maxScore <- softmaxScore[j,i]
        mostProbableClass <- i
      }
    }
    y_predicted <- rbind(y_predicted, mostProbableClass)
    y_predicted_val <- rbind(y_predicted_val, maxScore)
  }
  return(cbind(y_predicted, softmaxScore, y_predicted_val))
}
```

Where the class scoring function is the softmax function.

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

- These steps are then repeated iteratively with randomly initialized w and w_0 values, and an update rule is equal to gradient function multiplied by the learning rate (η)

```
while (1) {
  print(paste0("running iteration#", iteration))
  predictions <- c()
  predictions <- predictClass(X, w, w0, K, N)

  y_predicted <- predictions[,1]
  predicted_prob <- predictions[,2:4]
  predicted_val <- predictions[,5]

  error <- 0
  for(i in 1:N){
    error <- error + log(predicted_prob[i, y_truth[i]] +  1e-100)
  }

  objective_values <- c(objective_values, -error)

  w_old <- w
  w0_old <- w0
  w <- w + eta * gradient_w(X, predicted_prob, y_truth)
  w0 <- w0 + eta * gradient_w0(predicted_prob, y_truth)
  if (sqrt((w0[1,2] - w0_old[1,2])^2 + (w0[1,1] - w0_old[1,1])^2 + (w0[1,3] - w0_old[1,3])^2) < epsilon)
    break
  }

  iteration <- iteration + 1
}
```

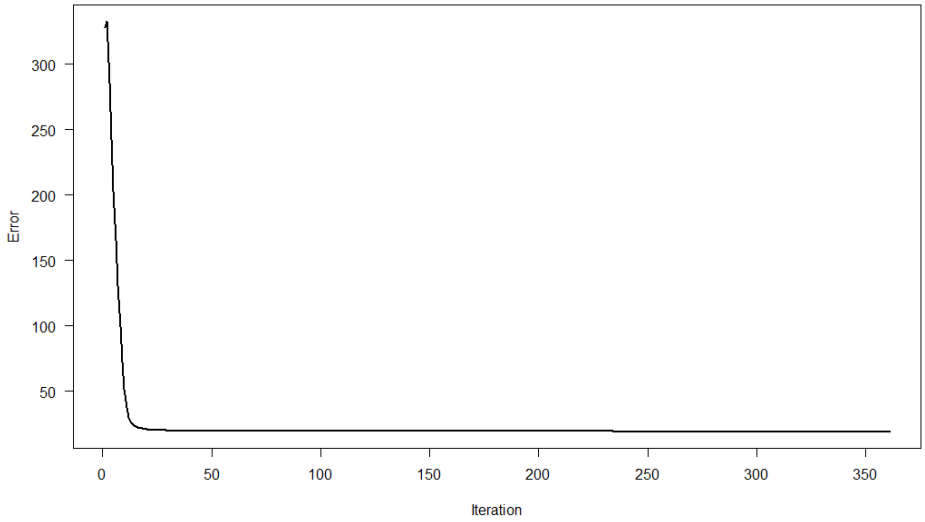
The values of w parameters after the iterations were as follows:

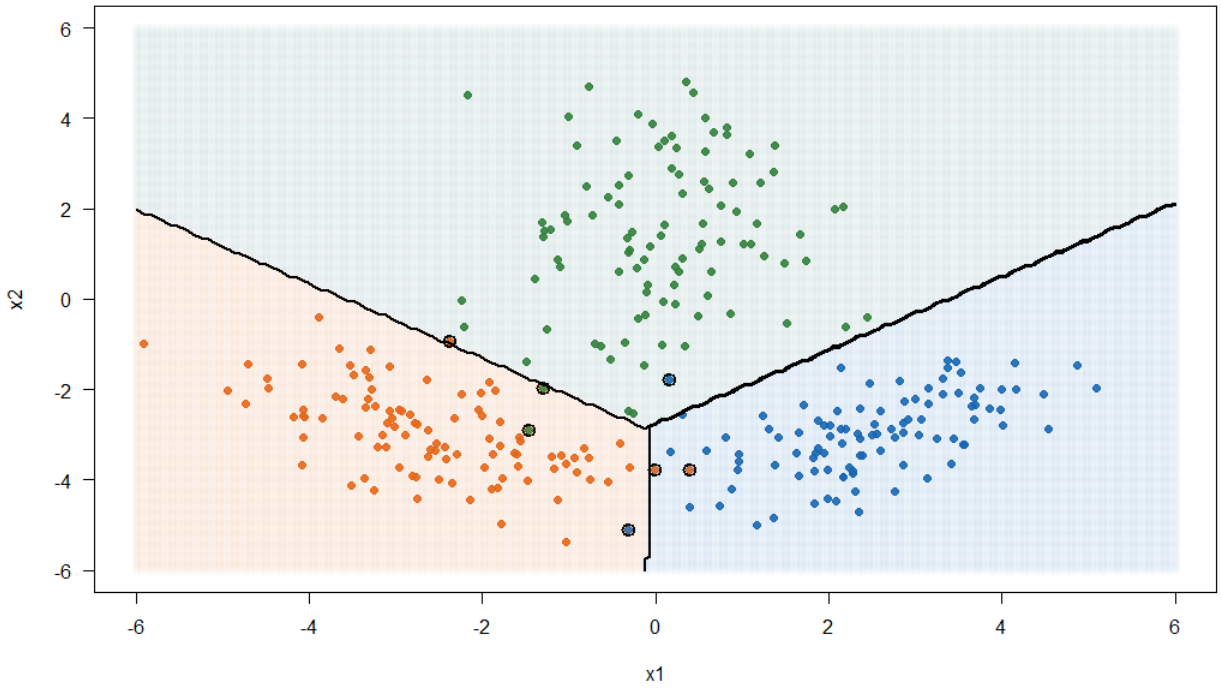
```
> print(w)
           w1         w2         w3
x1 0.01196065 -2.601623  2.604194
x2 2.11974557 -1.050529 -1.061161
> print(w0)
      w10       w20       w30
 6.001567 -3.281965 -2.718327
```

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

```
# plot objective function during iterations
plot(1:(iteration), objective_values,
     type = "l", lwd = 2, las = 1,
     xlab = "Iteration", ylab = "Error")

# calculate confusion matrix
confusion_matrix <- table(y_predicted, y_truth)
print(confusion_matrix)
```

```
> print(confusion_matrix)
           y_truth
y_predicted  1  2  3
          1 98  1  1
          2  2 97  1
          3  0  2 98
```


Error
300
250
200
150
100
50
0
50
100
150
200
250
300
350
Iteration


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