COMP/INDR 421/521 HW03: Multiclass Multilayer Perceptron

Asma Hakouz 0063315

November 3, 2017

The purpose of this assignment was to implement a multiclass multilayer perceptron algorithm in R.

The problem consisted of four main parts:

1- **Initialization**
2- **Data generation**
3- **Linear Discrimination Algorithm:** Where the multiclass multilayer perceptron 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**

```
# mean parameters
class_means <- matrix(c(+2.0, +2.0,
                        -4.0, -4.0,
                        -2.0, +2.0,
                        +4.0, -4.0,
                        -2.0, -2.0,
                        +4.0, +4.0,
                        +2.0, -2.0,
                        -4.0, +4.0), 2, 8)
# covariance parameters
class_covariances <- array(c(+0.8, -0.6, -0.6, +0.8,
                             +0.4, +0.0, +0.0, +0.4,
                             +0.8, +0.6, +0.6, +0.8,
                             +0.4, +0.0, +0.0, +0.4,
                             +0.8, -0.6, -0.6, +0.8,
                             +0.4, +0.0, +0.0, +0.4,
                             +0.8, +0.6, +0.6, +0.8,
                             +0.4, +0.0, +0.0, +0.4), c(2, 2, 8))

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

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

$$class\_means[,i] = [\mu_{i1} \quad \mu_{i2}]$$

$$covariance\ matrix\ \mathbf{\Sigma} \equiv class\_covariances[,,i] = \begin{bmatrix} {\sigma_{i1}}^2 & \sigma_{i12} \\ \sigma_{i21} & {\sigma_{i2}}^2 \end{bmatrix}$$

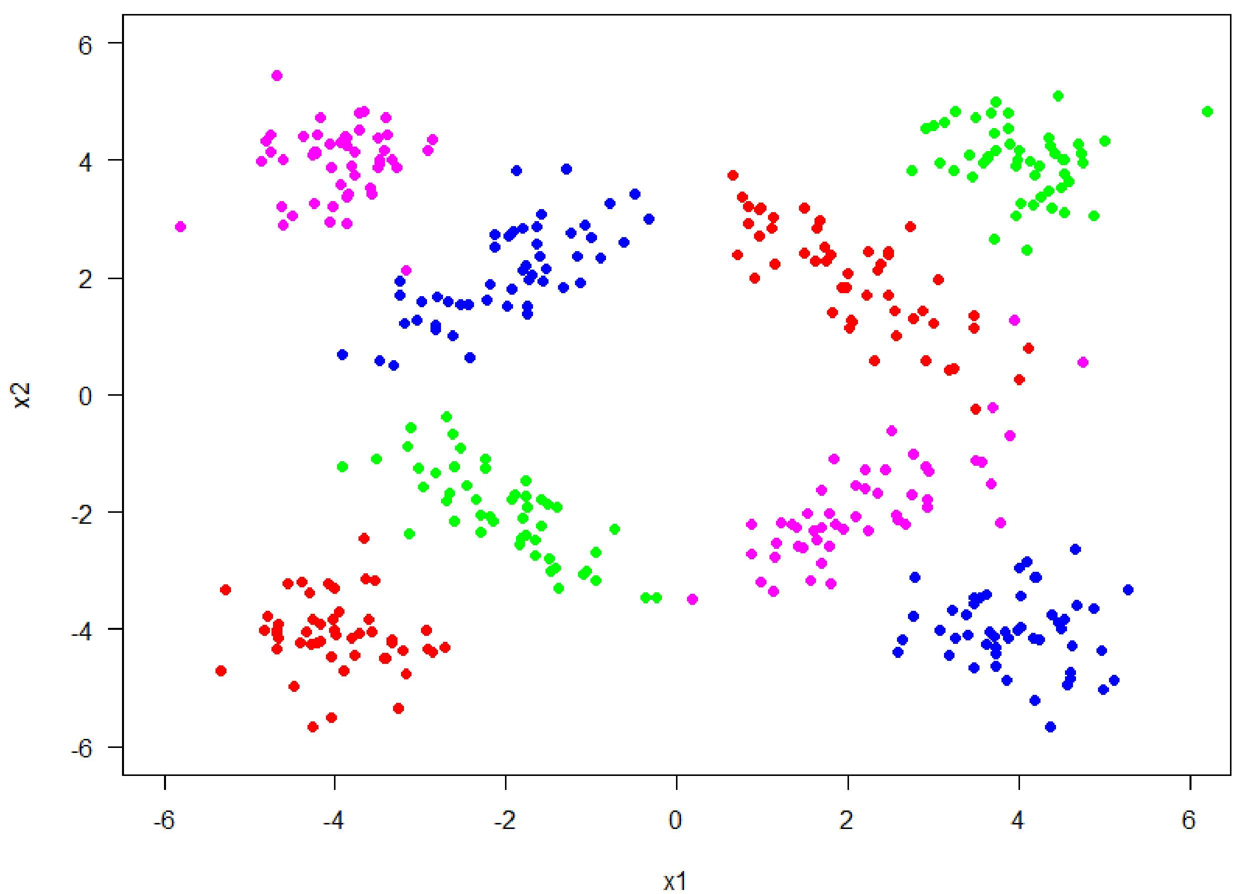
- **Data generation**

```r
# generate random samples
points1 <- mvrnorm(n = class_sizes[1] / 2, mu = class_means[,1], Sigma = class_covariances[,,1])
points2 <- mvrnorm(n = class_sizes[1] / 2, mu = class_means[,2], Sigma = class_covariances[,,2])
points3 <- mvrnorm(n = class_sizes[2] / 2, mu = class_means[,3], Sigma = class_covariances[,,3])
points4 <- mvrnorm(n = class_sizes[2] / 2, mu = class_means[,4], Sigma = class_covariances[,,4])
points5 <- mvrnorm(n = class_sizes[3] / 2, mu = class_means[,5], Sigma = class_covariances[,,5])
points6 <- mvrnorm(n = class_sizes[3] / 2, mu = class_means[,6], Sigma = class_covariances[,,6])
points7 <- mvrnorm(n = class_sizes[4] / 2, mu = class_means[,7], Sigma = class_covariances[,,7])
points8 <- mvrnorm(n = class_sizes[4] / 2, mu = class_means[,8], Sigma = class_covariances[,,8])
X <- rbind(points1, points2, points3, points4, points5, points6, points7, points8)
colnames(X) <- c("x1", "x2")


# generate corresponding labels
y <- c(rep(1, class_sizes[1]), rep(2, class_sizes[2]), rep(3, class_sizes[3]), rep(4, class_sizes[4]))

# write data to a file
write.csv(x = cbind(X, y), file = "HW3_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:



- **Multiclass Multilayer Perceptron 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("HW3_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 samples and number of features and number of classes
N <- length(y_truth)
D <- ncol(X)
K <- max(y_truth)
```

- Defining softmax

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

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

- Classification

  Where the class scoring function is the softmax function.

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

```r
while (1) {
  print(paste0("running iteration#", iteration))
  for (i in sample(N)) { # sample(N) gives a random ordering of numbers 1 to N, we want to access data in a random order
    # calculate hidden nodes
    Z[i,] <- sigmoid(c(1, X[i,]) %*% W)

    # calculate output node
    y_predicted[i,] <- sapply(1:K, function(c){softmax(matrix(c(1, Z[i,]), 1, H + 1), v, K, c)})
    for(e in 1:K){
      if(y_truth[i] == e){
        v[,e] <- v[,e] + eta * (1 - y_predicted[i, y_truth[i]]) * c(1, Z[i,])
      } else {
        v[,e] <- v[,e] + eta * -1 * y_predicted[i, e] * c(1, Z[i,])
      }
    }
    for (h in 1:H) {
      sum_error <- 0
      for(class in 1:K){
        if(y_truth[i] == class){
          sum_error <- sum_error + (1 - y_predicted[i, class]) * v[h, class]
        } else {
          sum_error <- sum_error - y_predicted[i, class] * v[h, class]
        }
      }
      W[,h] <- W[,h] + eta * sum_error * Z[i, h] * (1 - Z[i, h]) * c(1, X[i,])
    }
  }
  error <- 0
  for(i in 1:N){
    error <- error + log(y_predicted[i, y_truth[i]] +  1e-100)
  }
  objective_values <- c(objective_values, -error)
  if(iteration != 1){
    if ((abs(objective_values[iteration] - objective_values[iteration - 1]) < epsilon || iteration >= max_iteration)) {
      break
    }
  }
  iteration <- iteration + 1
}
```

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

```r
y_predicted_class <- c()
for(i in 1:N){
  y_predicted_class <- rbind(y_predicted_class, which.max(y_predicted[i,]))
}
```

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

```
> print(W)
           [,1]      [,2]      [,3]      [,4]      [,5]     [,6]      [,7]      [,8]       [,9]    [,10]     [,11]
[1,]  6.6421114 16.183535 13.568445  9.606574 10.882318 9.457999 10.258945  4.246377  0.9366378 7.515359  7.851477
[2,] -0.3973043 -3.247718 -2.738522 -1.675305  1.872582 2.172038  2.069357 -3.938547 -6.1934437 1.340761 -1.434020
[3,] -1.3745071 -1.888773 -2.414250 -1.643067  2.734930 1.660503 -1.532246  5.863815  0.2611013 1.965851  1.609761
          [,12]      [,13]    [,14]      [,15]     [,16]     [,17]       [,18]    [,19]    [,20]
[1,]  7.549591 6.08786684 8.358916  9.0584142  9.649272  6.542305  3.73750780 8.169214 6.101761
[2,] -1.556821 0.33129887 2.149746 -0.4373296 -2.395928 -3.045919 -0.09817704 2.596878 1.877990
[3,]  1.552035 0.03377539 1.107313  2.8371040  1.684598  2.114441  7.78099965 4.469213 3.155037
> print(v)
              [,1]       [,2]       [,3]        [,4]
 [1,]   1.22578898 -3.2752719 -0.8138384   2.85491376
 [2,]   6.45466756 -0.5562685 -6.0946715   0.21017778
 [3,]   6.48803439  0.2287617 -5.1419805  -1.57672859
 [4,]   4.42965542  2.4213326 -6.8581354   0.01189458
 [5,] -10.46428810  9.3609146 -1.9226784   3.02511974
 [6,]  -6.29895900  1.9403376  2.3865458   1.98564019
 [7,]  -3.88727056  1.7979948  3.2354870  -1.14258910
 [8,]   3.62658762  3.9069454  3.4525445 -10.98362312
 [9,]  -2.29755614  1.9565008  2.0482115  -1.70898779
[10,]  -6.45489239  3.2503260  2.7969166   0.38510835
[11,]   0.05263743 -4.2880708  2.0518933   2.18187222
[12,]   1.22027299 -5.4567859  2.4293527   1.81034826
[13,]  -0.11906993 -4.5325438  2.4379738   2.21387609
[14,]  -3.72530229 -1.6997933  2.2524160   3.18287697
[15,]  -3.43953756 -2.1092468  2.6597211   2.87995519
[16,]   2.86544434 -8.5651483  2.6571417   3.05699048
[17,]   3.35122658 -4.6881655  2.2584355  -0.92444250
[18,]   0.64322297  2.7412882  2.9941344  -6.39329884
[19,]   3.02454686  5.3580628 -3.2904141  -5.07990564
[20,]   0.34256288  3.7710632 -5.5250503   1.41472120
[21,]   2.66228065  2.3268913 -6.3755187   1.37370073
```

- 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.

```
> print(confusion_matrix)
                 y_truth
y_predicted_class   1   2   3   4
                1  97   0   0   1
                2   0 100   0   1
                3   2   0 100   0
                4   1   0   0  98
```