COMP421 HW #2, Prepared by Arda Kırkağaç, 0049799

Our task was to create a multiclass linear discriminator for bivariate normal distributions, and the class number was already set to three with the given assignment. Again, according to the given distribution statistics, using MASS library, relevant points (100 for each class) were generated. A seed of 521 is used for the random generation. Those points were first visualized for a better understanding. For the prior classes of the samples, I used one-of-K encoding method as seen in the lecture. As you can see in my code, “y_BinaryTruth” is a 300*3 matrix that contains 1’s and 0’s at points where the appropriate classes are.

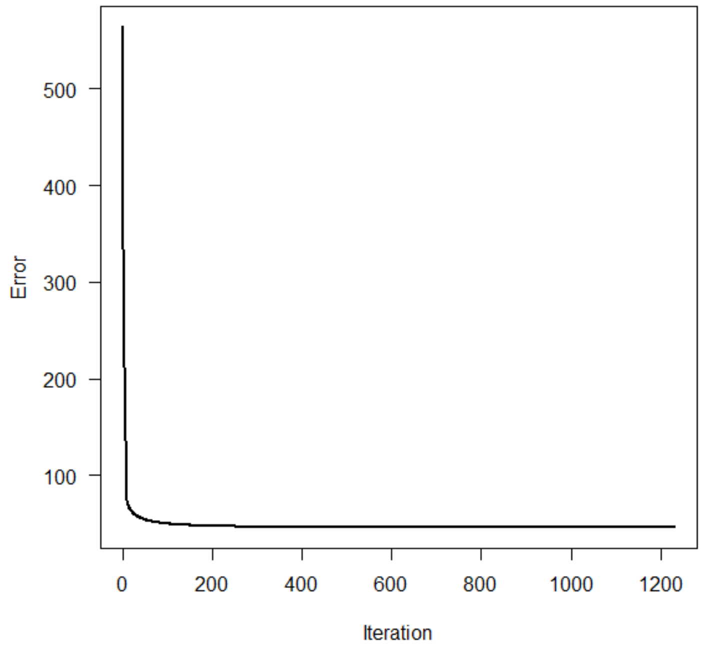
Then for a little bit of simplicity, I defined the softmax denominator and then the softmax function itself as follows:

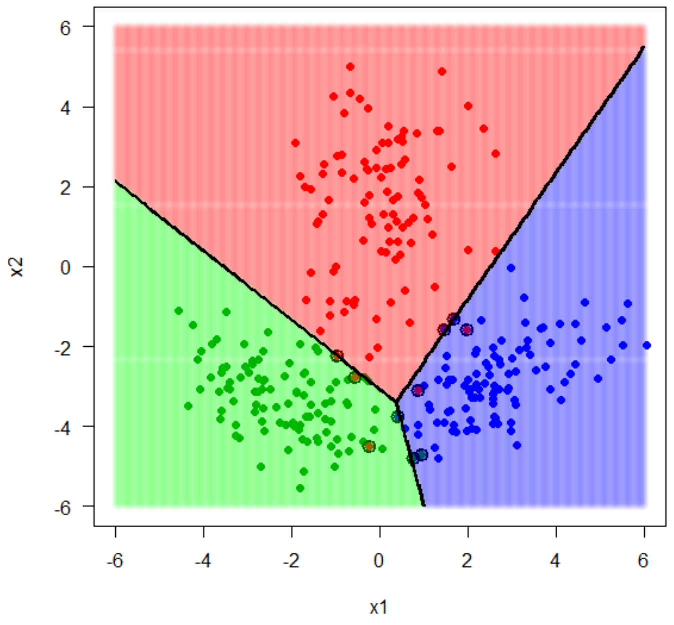
```r
softmaxDen <- function(X,w,w0){
  return (exp(X%*%(w[,1])+w0[1])+exp(X%*%(w[,2])+w0[2])+exp(X%*%(w[,3])+w0[3]))
}
#define softmax function
softmaxFcn <- function(X,w,w0){
  denominator <- softmaxDen(X,w,w0)
  softmaxClass1 <- (exp(X%*%(w[,1])+w0[1]))/(denominator)
  softmaxClass2 <- (exp(X%*%(w[,2])+w0[2]))/(denominator)
  softmaxClass3 <- (exp(X%*%(w[,3])+w0[3]))/(denominator)

  softmaxOutput <- cbind(softmaxClass1,softmaxClass2,softmaxClass3)
  return (softmaxOutput)
}
```

After this, gradient functions for w’s and w0’s were defined just as in the sample lab session, but for multiple classes. I defined gradients for each class separately and then combined them. After that, I initialized the iteration parameters (eta and epsilon) . I also changed the seed for random w points to 421, although it will not affect gradients too much since the system will converge the same way, as long as the learning data is the same. In this iterative algorithm, the function for “objective_values” was kept same since the only thing that changed is the column size, and R can handle that.

One can choose different expressions in order to stop iterations, by checking that value against epsilon at the end of each iteration. Upon trying different expressions and different values for learning parameters, I saw that almost every set of them converged to very similar gradient final values and very similar learning performances. I chose epsilon as 1e-3, eta as 0.001, and break condition as the square root of the summations of gradients (w and w0 separately). After iterations, my learning curve, my grid span and my gradient values are as follows:





My algorithm stopped at 1229th iteration. We can see that the values and graphs exactly look like as they appear in the pdf document.

To achieve grid graph, I defined another softmax function with now fixed weights after iterations. It basically assigned 3 regression values to each point on the grid, whom of max was selected to assign the final class to this particular point. I did not use the denominator of the original softmax in this function, because I realized that it is just a constant for a fixed point on the grid. In order to apply the max function with MARGIN=1, I decompose the grid structure first. After the assignments, grid shape was reconstructed, and grid colors were shown on screen. The last thing to do was again to separate the grid into three parts by using “nlevels=3”, which separates the given grid into three distinct areas, which is quite easy since we have only three different values in grid assignment matrix.

```
> w
     gradient_w1 gradient_w2 gradient_w3
[1,] -0.04076754   -3.192507   3.2402326
[2,]  1.90521841   -1.731307  -0.1738119
> w0
     gradient_w01 gradient_w02 gradient_w03
[1,]     6.462594     -4.69406    -1.756108
>
>
> w
     gradient_w1 gradient_w2 gradient_w3
[1,] -0.04076754   -3.192507   3.2402326
[2,]  1.90521841   -1.731307  -0.1738119
> w0
     gradient_w01 gradient_w02 gradient_w03
[1,]     6.462594     -4.69406    -1.756108
> print(confusion_matrix)
             y_truth
y_predictions  1  2  3
            1 95  0  2
            2  3 98  1
            3  2  2 97
```