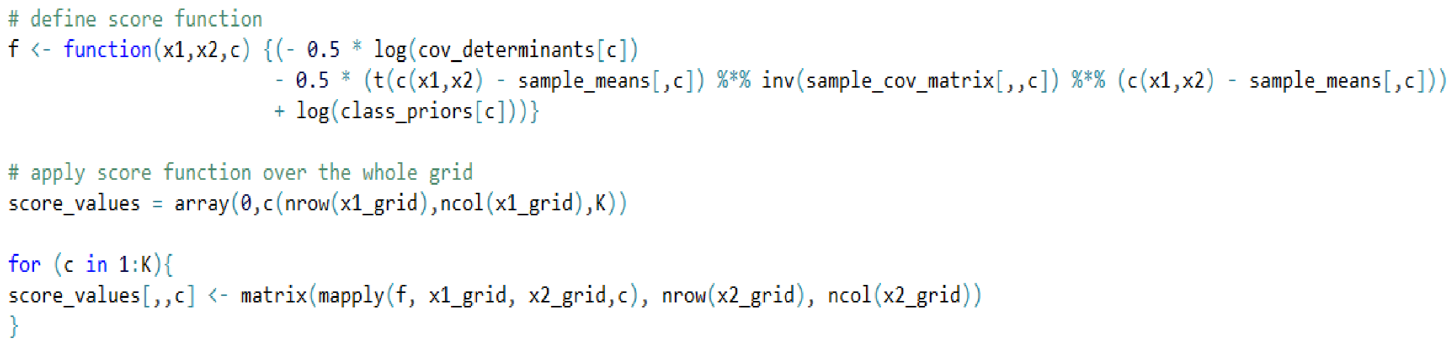
COMP421 HW #1, Prepared by Arda Kırkağaç, 0049799

Our task was to create a multiclass classifier 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 written to an external file, and visualized for a better understanding. Then sample statistics were calculated using our generated points. Class priors were assigned to each class.

Then, I defined the score function, taking the scoring formula in the book as the base. In this scoring formula, inverse function was used to achieve the best result (and the result that looks the same as the proposed answer). This required installing and using another library called “matlib”, and I think this exact inverse operation makes our whole process a lot slower. However; for the exact calculation of score functions, it turns out that using “inv” function is required. My exact score function is as follows:

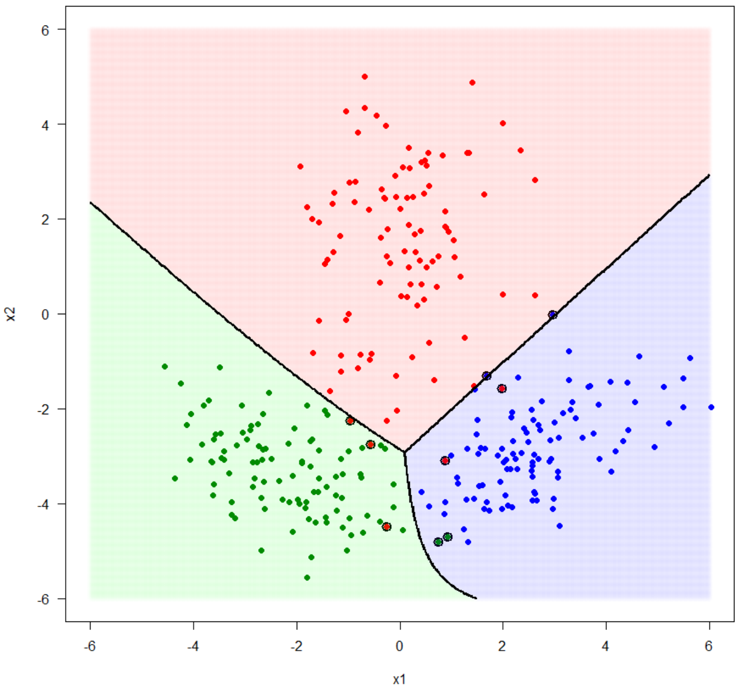


```r
# define score function
f <- function(x1,x2,c) {(- 0.5 * log(cov_determinants[c])
                        - 0.5 * (t(c(x1,x2) - sample_means[,c]) %*% inv(sample_cov_matrix[,,c]) %*% (c(x1,x2) - sample_means[,c]))
                        + log(class_priors[c]))}

# apply score function over the whole grid
score_values = array(0,c(nrow(x1_grid),ncol(x1_grid),K))

for (c in 1:K){
score_values[,,c] <- matrix(mapply(f, x1_grid, x2_grid,c), nrow(x2_grid), ncol(x2_grid))
}
```

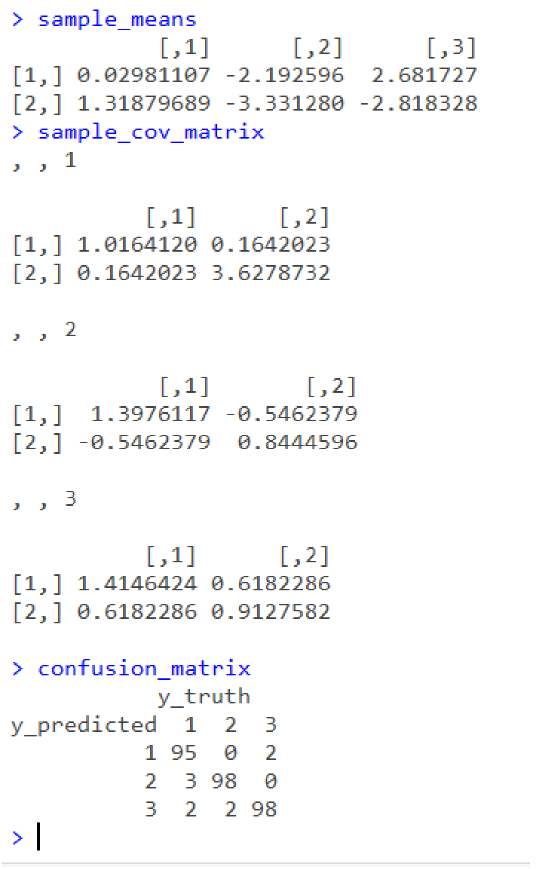
Since we have a grid of 201*201 (when the sequence interval is 0.06) with 3 different scoring layouts for each class, and since our scoring function involves complex matrix operations, the scoring process takes some time, depending on the interval sampling frequency. However; the smaller the gap between two points, the more accurate and smoother the result.

For each point in the grid and for the data points, score values are calculated. Class assignments were to be made according to the maximum of those score values for any point. Furthermore, the class assignments which make the score values maximum for any point on the grid were written into matrices using “which.max” function over the class margin. This operation gave us predicted class assignments, which were further used to create confusion table with truth values.

The last part was to plot the changes and assignments. All the points were plotted. According to the confusion table, the points whose predicted classes do not match were indicated. Then, the whole grid was colored according to the class assignments for the grid points. The last thing to do was to sketch a contour between different areas of assignments, using the critical assignment “nlevels=3”, which separates the assignment grid into 3 distinct areas. Since we had 3 different values through the whole matrix, this argument clearly drew the contours between two different classes. On the right, you can see my final plot with 0.02 interval for grid points. However, my final code sets that value to 0.06, which dramatically increases the operation speed.



On the right, you can see the sample statistics from my generated data points. Since the same seed is used for the whole process, the values are exactly the same as in the assignment document.



```
> sample_means
           [,1]      [,2]      [,3]
[1,] 0.02981107 -2.192596  2.681727
[2,] 1.31879689 -3.331280 -2.818328
> sample_cov_matrix
, , 1

          [,1]      [,2]
[1,] 1.0164120 0.1642023
[2,] 0.1642023 3.6278732

, , 2

           [,1]       [,2]
[1,]  1.3976117 -0.5462379
[2,] -0.5462379  0.8444596

, , 3

          [,1]      [,2]
[1,] 1.4146424 0.6182286
[2,] 0.6182286 0.9127582

> confusion_matrix
           y_truth
y_predicted  1  2  3
          1 95  0  2
          2  3 98  0
          3  2  2 98
> |
```