Neural Networks
Group 04
Andrea Jemmett (2573223)
Bas van der Drift (1707507)

Assignment 2 - Statistical Learning

Section A

The main Matlab script for this section can be found in ‘ass2a.m’.

To create a dataset to represent the width/height ratio of characters, we wrote the function contained in ‘gen_dataset.m’. This function accepts two parameters: the number of samples to be generated and a range of values for the ratios to be generated. We used two Normal random number generators and calculated the two means so that their generated numbers will overlap a bit. The generated dataset is put in a tabular data structure like the one seen during the lecture.

In order to calculate the desired probabilities, we wrote the function ‘calc_probabilities.m’. This function takes the dataset as input and returns prior, class conditionals and posterior probabilities.

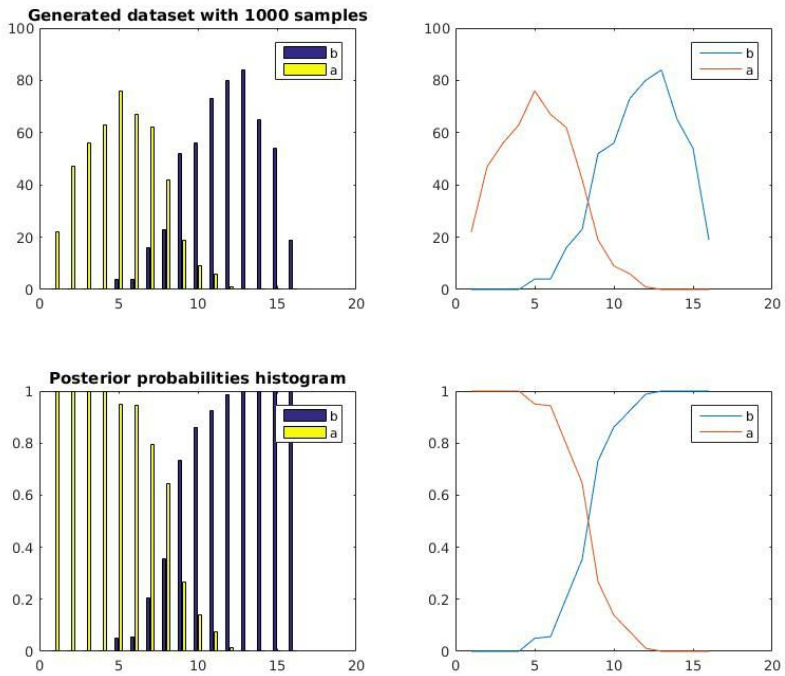

Figure 1. Top: histogram and plot of frequencies per class.
Bottom: histogram and plot of posterior probabilities per class.

After calculating the posteriors, we're able to plot the two required histograms, as shown in Figure 1.
The function 'calc_boundaries_errors.m' calculates for all the possible splits (all possible decision boundaries) the misclassified cases for both classes. It accepts as parameters the dataset and the costs for the misclassifications only and outputs a matrix where each row represents a class and each column represent a possible split. In cell i, j there is the number of misclassified cases for class i and split j. The total number of splits is equal to the number of the possible ratio values minus one; this way split i is in between of i and i+1 ratio value. Later we calculate the total number of misclassified cases per each split and select the one that minimizes the total number of misclassified cases.To do this we used the Matlab min function that returns the minimum value and the index of that value in the vector passed as argument. In this way, with the generated dataset already described, we found an optimal decision boundary between 8 and 9 with 94 misclassified examples.

We then repeated the experiment with different misclassification costs. As asked, we doubled the cost of misclassifying a 'b' as an 'a' and found an optimal decision boundary between 7 and 8. We can conclude that the decision boundary is shifted to the left to minimize the higher misclassification error given by 'b'. This shift is observable in Figure 2.

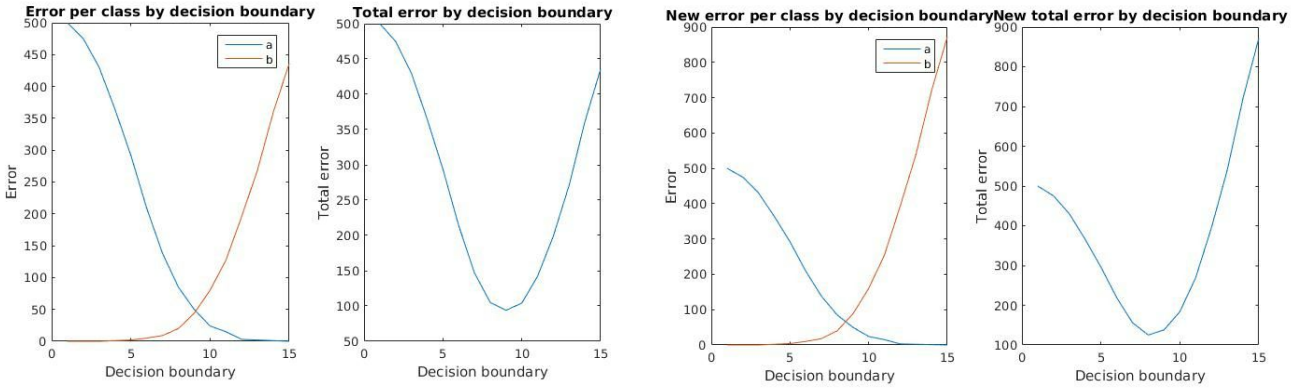

Figure 2-3. Left: error with equal costs. Right: error with cost of 2 vs. 1.

Section B

All the code pertaining this section is contained in the script 'ass2b.m'.
We applied both the perceptron and the LMS algorithms to both datasets and used the function 'testweights.m' from the previous assignment to count the number of misclassified cases. The results can be seen in Figure 4. As shown, for the linearly separable case both the perceptron and the LMS were capable of classifying all cases correctly. For the non-linearly separable case instead, we noticed that the perceptron (if given a maximum number of iterations greater than 10) gives consistently better separation boundaries than the LMS algorithm, with an accuracy (number of correct examples divided by number of all examples) between 0.588 (with 10 iterations) and 0.708 (with 1000 iterations), as opposed to the constant 0.592 for the LMS.

By looking at Figure 4, is easy to notice that the LMS algorithm outputs a better separation boundary than the perceptron for the linearly separable dataset (the distance from the separation boundary to the data points is higher than in the perceptron, which separation boundary is more toward the red dots); this is due to the continuous nature of the error function in the LMS implementation, as opposed to the expression used in the perceptron implementation, that makes use of a signum function.

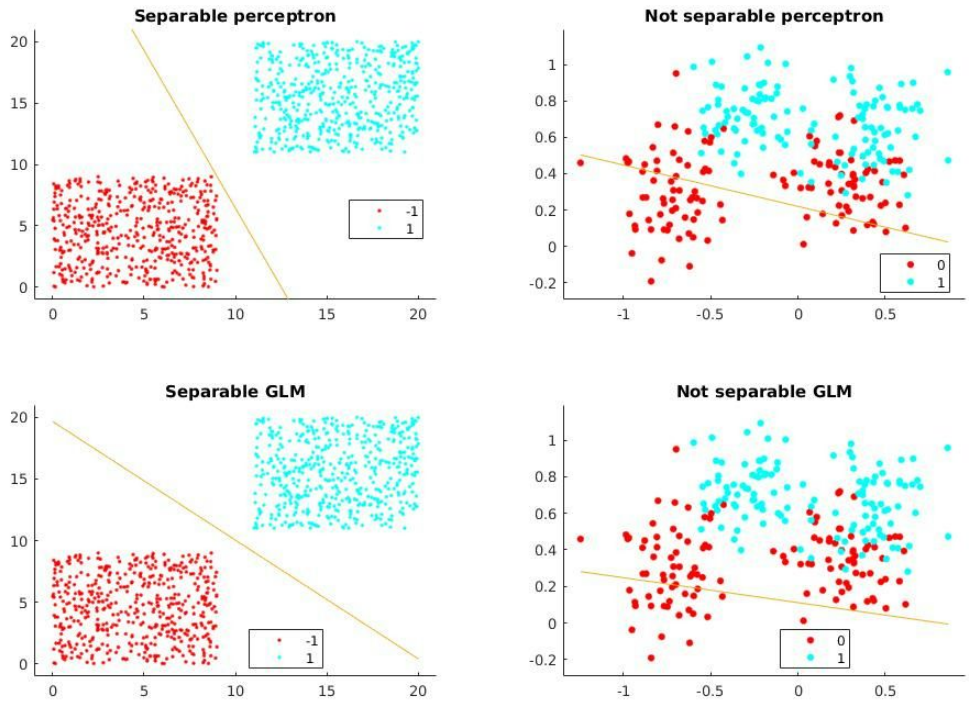

Figure 4. Top half results from perceptron, bottom results from LMS.

Regarding the parameter settings for the Least Mean Square algorithm, we noticed that training a linear GLM with different learning rates and maximum iterations (by means of the PRIOR and option(14) glmtrain parameters), doesn't have any effect on the resulting decision boundary. We also noticed that changing the maximum iterations didn't change the running time of the LMS algorithm and investigating the source code of 'glmtrain.m' we noted that for a linear activation function, the implementation doesn't iterate more than once over the data. If the glmtrain function would have been implemented with a stochastic gradient descent, we would have observed the accuracy incrementing for bigger values of the maximum iterations (for the non-linearly separable dataset). Not even the learning rate has effects on the results of the algorithm; with a stochastic gradient descent implementation, the learning rate would have had effects on the speed of convergence in a linearly separable case or on the time to reach a local (or global) optimum for a non-linearly separable dataset.