Report for Assignment 1

I coded everything in Jupyter Notebook; Therefore, in my submission, I am also adding Jupyter Notebook with all output cells, along with R script file which includes the whole code.

Task 1: I have implemented the Gradient Descent Algorithm using NLL function. Instead of using the gradient algorithm provided in Appendix C. I have derived it myself and have obtained a different result. The results I have obtained for the derivative are computationally efficient.

I have not considered y =1 and y=0 separately as used in the theorem; I have made a combined equation and differentiated the same. I used a single NLL equation. -yi*log(p(xi))-(1-yi)*log(1-(pi)) under the summation, if y is 0, the first term reduces to 0 and if y is 1, the second term in that equation reduces to zero. I have also used the same to compare with inbuilt functions and the results are highly similar. I have attached the derivation in my submission.

Task 2: I applied the method over the whole set, and It provided an MSE error of 0.2988 and 35 misclassifications. When I tried the inbuilt R glm functions, I found equal number of misclassifications.

Task 3: The whole set divided into 2 equal parts, choosing the samples using 2810 as the seed value.

Task 4: I tried different values of learning rate and Epoch combinations. Below are some examples:

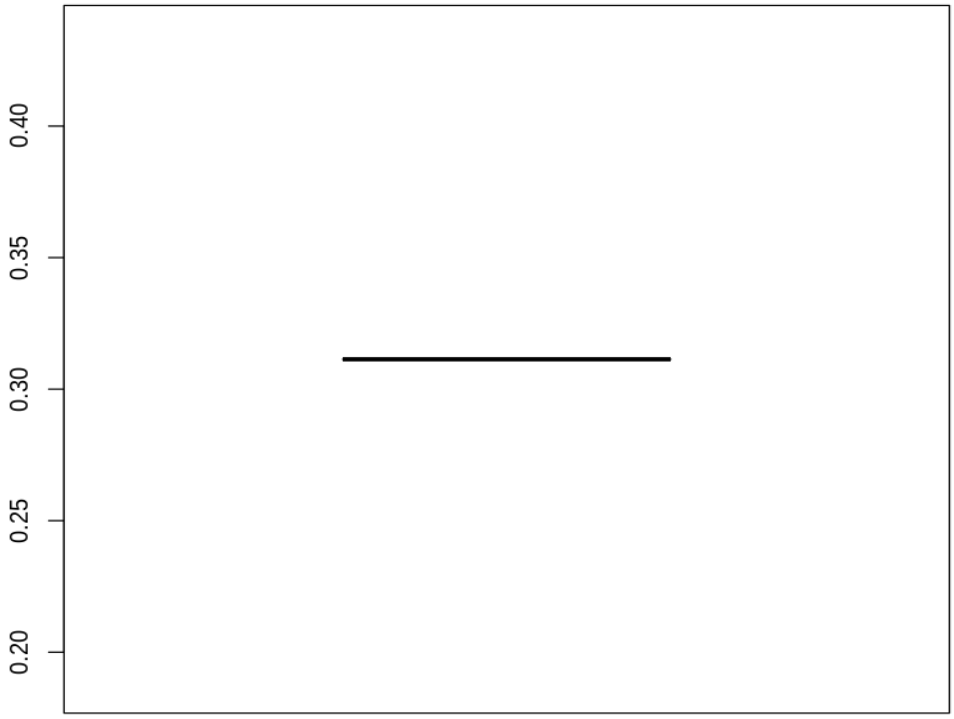
(Learning rate, Epoch)	Training MSE	Test MSE
(10000,0.01)	0.3350	0.3350
(1000,0.01)	0.3642	0.4103
(1000,0.05)	0.3350	0.3499
(2000,0.05)	0.3350	0.3350
(2000,0.01)	0.3350	0.3113
(10000,0.01)	0.3273	0.3113
(10000,0.2)	0.3273	0.3113

(5000,0.2)	0.3273	0.3113
(2000,0.3)	0.3273	0.3113

Interestingly for this test set, Test MSE is lower than Training MSE, I tried it with different seed value, the same chance doesn't occur again. After, this table I chose Epoch = 2000, Learning Rate = 0.3 as optimum values.

Task 5: (Optional) I created a Cost function which computed the NLL or Briar Loss for every step and saved it in a variable J_history. In the code, I used only one stopping method, that was to check if the reduction in NLL in the next step was less than 0.00001, our terminate our loop. To check at what iteration it stopped, I printed the value of loop variable after loop exit. It turned out to be 1094, which is approximately half of our epoch value (2000). The J_history variable and Cost function can be used to implement other stopping methods, and even to check the convergence of our function and debugging.

Task 6: I implemented it with 100 different initialisations of weights, and calculated the MSE and plot it using boxplot function in R. Here, I found very interesting results, my MSE was the same in all the cases. I studied a bit more in books and found that for Gradient Descent cost function, it is expected if our Cost function converges to the minima, and practically there is only one minima in the cost function. To confirm the theory, I chose sub-optimal values of learning parameter and epoch, and the function doesn't converge to minima by this and the box plot shows variability in the MSE. Below is the graph for optimal parameters as asked in the question.



Task 7: I ran Gradient Descent 4 times with different initialisation and chose the weight with minimum MSE. Since the MSE was same for all initialisation, the step had little to no importance. I checked all the corresponding weights and they had same valued up to 2 decimal places.