UUN: s1661552

Task 3

This task has caused me the most problems. The first one being finding an efficient way of calculating the covariance matrix, which I solved using the tip provided in the lectures about how to vectorize the operation. Second problem that arose was the zero-determinant problem, which I overcame by using the $slogdet()$ function in numpy. Surprisingly, only adding a small number to the diagonal elements of the matrix was not enough to solve the problem. In my effort to reduce the running time as much as possible, I precomputed all the inverse covariance matrices for each class, as well as natural logarithms of the determinants of the covariance matrices. I think that in this case equal prior probabilities of the data could have been assumed as the number of data vectors for each class was equal, but in my implementation a general form with a varying prior was chosen. After some investigation I found that the best epsilon value for the classification was 0.03, which helped to reach an accuracy of 84.60%.

Epsilon	Time taken, s	Accuracy, %	N	N_{errs}
0.005	118.27	83.14	7800	1315
0.01	114.62	83.82	7800	1262
0.02	116.45	84.42	7800	1215
0.03	118.26	84.60	7800	1201
0.04	113.44	84.42	7800	1215
0.05	117.54	84.35	7800	1220
0.1	114.25	83.65	7800	1275

For the second part of the task, I decided to use k-means clustering to cluster each class into smaller classes and run the gaussian classification algorithm on the newly labelled data. This approach did increase the computation time significantly, but also improved the classification accuracy. I randomized initial cluster centres for the k-means algorithm and then used mean squared error to choose the best classification out of three runs of k-means. 6 clusters for each class proved to be the best choice with an accuracy of 89.69%. Then I decided to run an experiment on the best cluster choice for each class. Unfortunately, even choosing only 3 different number of cluster values to consider, the computation gets impossibly long, as the algorithm has to consider 3^{26} different combinations of clusters for each class. Approximating the runtime to be 120s for a one average run gives us a prediction of the total runtime of around $8.5 * 10^{10}$ hours, which is obviously not a reasonable algorithm to even try to run. Instead, I thought about clustering the data into, say, 5 smaller datasets, either by using a visual inspection or the same k-means algorithm, and considering them separately. The idea behind this is that if the data in the smaller dataset is similar among itself and different enough from the other datasets, finding an optimal number of clusters for each dataset and then merging the data labels together should be accurate, since the datasets would be mostly independent. This approach would require 5 times 3^5 classifications, which results in estimated time of around 40.5 hours, which may seem reasonable for someone building a robust classification tool. Also, the computation could be easily parallelised, decreasing the time significantly. For my implementation, I would have used only two different values of clusters for each class, which would have required around 5.33 hours to finish. Due to time constraints I did not

manage to implement this idea. To try and at least somehow optimise the number of clusters, I ran an experiment on clustering only one class at a time with other classes remaining the same and then measuring the performance of each number of clusters for a specific class. This did not produce the expected result with an accuracy of 86.99%, lower than the accuracy of just clustering each class into 6 subclasses. The $experiment()$ function was called in $my\_improved\_gaussian\_system.py$ and is commented out at the moment. It is included in the $my\_improved\_gaussian\_classify.py$ file. Running the $experiment()$ takes 3.65 hours to finish on DICE.

Number of Clusters	Time taken, s	Accuracy, %	N	N_{errs}
1	81.37	84.60	7800	1201
2	100.48	87.54	7800	972
4	124.91	88.83	7800	871
6	135.69	89.69	7800	804
8	140.57	88.79	7800	874
10	146.67	89.15	7800	846
12	150.68	86.94	7800	1019
Custom Clusters	**96.67**	**86.99**	**7800**	**1015**