Foundations of Machine Learning Assignment

Brijee Rana

br1e21@soton.ac.uk

1 Labs 1-5

I have completed all the 5 labs and uploaded the reports. I have also re-visited and completed any tasks or feedback given.

2 Task 2: K-Means Clustering

2.1 Implementing K-means clustering algorithm

I had to sample data from a mixture Gaussian density and implement the K means clustering algorithm. Using code from the appendix I wrote code for the K-means iterations. Results of the implementation are shown in figure 2.

2.2 Contours

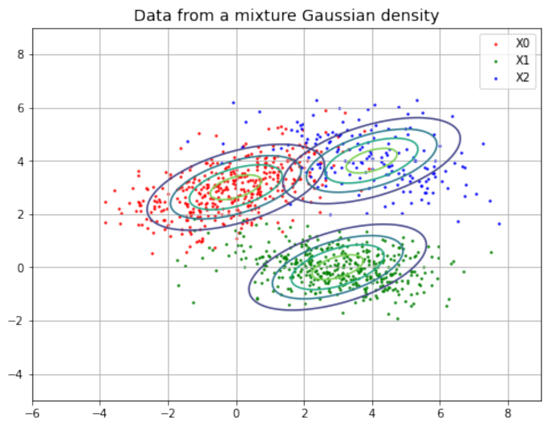


Figure 1: Distribution of the sample data

Figure 1 shows the contours of the probability densities I have used. The sample data is from 3 Gaussian densities thus we use K=3 for the number of cluster centers. As you can see each region are close to each other and only slightly overlap each other. The X2 gaussian density is more sparsely spread than the others All 3 contours are skewed diagonally but does not seem to reflect the data in some regions.

2.3 Sklearn K-means clustering algorithm comparison

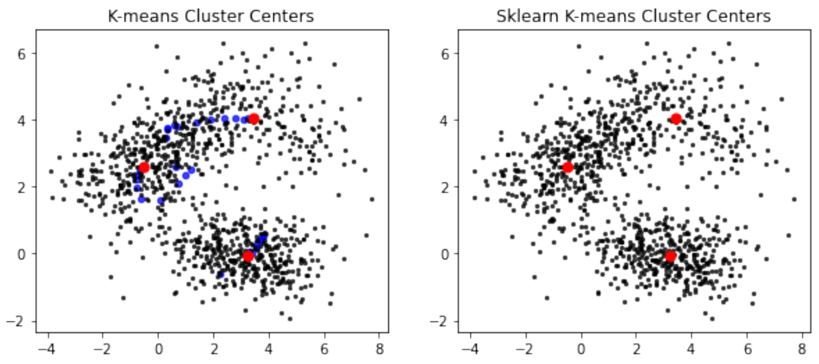


Figure 2: Comparison with sklearn algorithm

As you can see by figure 2, both algorithms have similar in results in fact, the position of the cluster centers are the same to 9 significant figures. The plot on the left has the iterations of the cluster center estimations marked in blue.

2.4 Initial cluster centers and parameter K

Initial cluster centers:

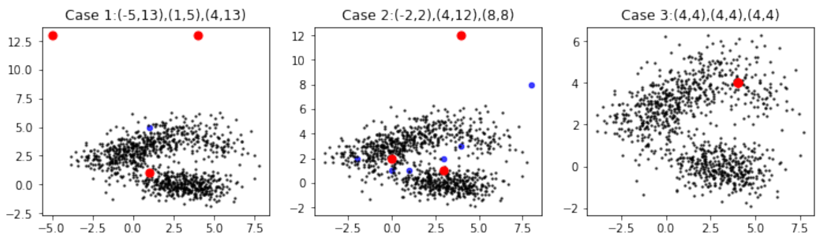


Figure 3: The algorithm with different initial cluster centers

In figure 3, all 3 cases make the algorithms fail. The algorithm is sensitive to the initial guess of the clusters. When the initial cluster center locations are too far away from any data, the algorithm cannot re-average the centroid and move it to maximise the likelihood. To solve this issue, we can choose the locations of the cluster centers to be instance in the dataset. The algorithm is also sensitive to the number of clusters K as shown in figure 5. When K is too small, it cannot cluster all the regions properly and when K is too big, it overfits and makes it so that the location of cluster centers are wrong as the algorithm has to fit more cluster centers than needed. A common solution is the Elbow curve method.

Number of clusters K:

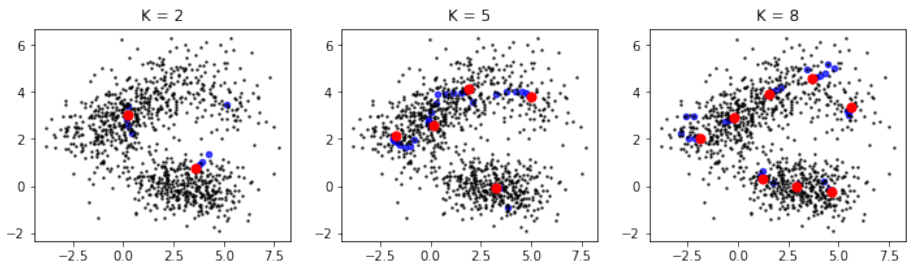


Figure 4: The algorithm with different K values

2.5 Applying the algorithm on a UCI classification dataset

I chose to use the iris dataset from the UCI repository. The dataset contains 4 features and 3 classes(setosa, virginica, versicolour). One class(setosa) is linearly separable from the other two(virginica, versicolour) yet the other two are not linearly separable from each other. Each class has 50 instances. Two features are about the dimensions of the petals and the other 2 are about the dimensions of the sepal. I decided to investigate this classification problem in terms of petals and sepals. Figure 5 and 7 show the distributions w.r.t to petals and sepals and figure 6 and 8 show the respective results when applying the K-means clustering algorithm. Using the petals dimension to solve this classification problem is better than using sepals as the classes are less overlapped with each other. This is shown by figure 6 as the cluster centers are located more accurately than in figure 8. K=3 for this dataset as there are 3 classes. Figure 5 shows that it is more easier to linearly separate the data using petals than sepals.

Petal set:

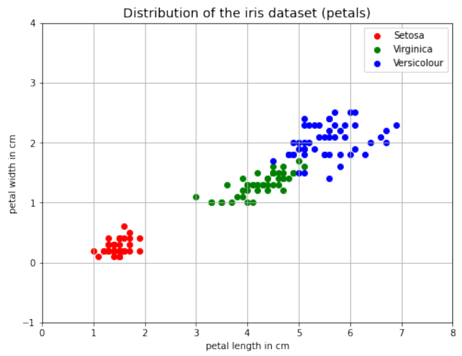


Figure 5: Distribution of dataset

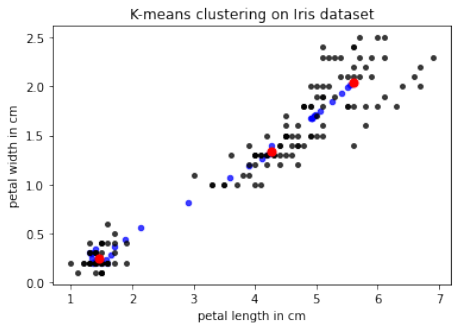


Figure 6: K-means clusters

Sepal set:

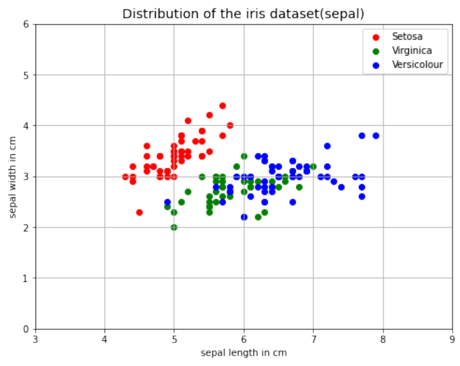


Figure 7: Distribution of dataset

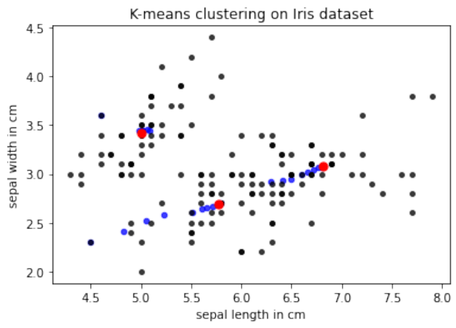
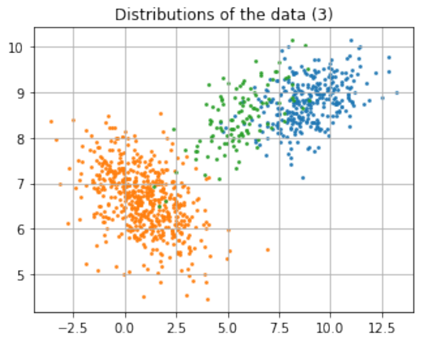


Figure 8: K-means clusters

3 Task 3: Multi-Layer Perceptron

3.1 Setting up the classification problems

I begin with setting up the 2 classification problems using a Mixture of Gaussians (3 Classes). Figure 9 shows the distribution of the data and the training targets. We will apply our classifiers upon this dataset.



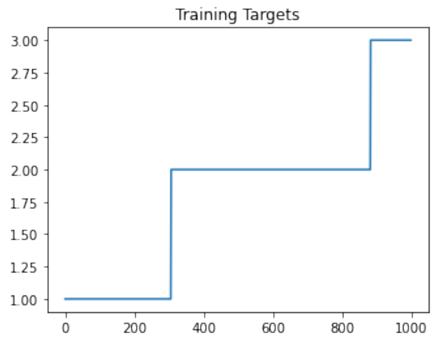


Figure 9: The data distribution and training targets of the classification problem

3.2 Bayesian classifier and MLP classifier

I then implement a Bayesian and MLP classifier using scikit.learn upon the data that has been split into training and testing sets. I alse make box-plots from ten-fold cross validation to compare the performance of each classifier. Figure 10 shows the box-plots of the classifiers. As you can see, both classifiers solve the problem up very well (100% accuracy in some iterations).

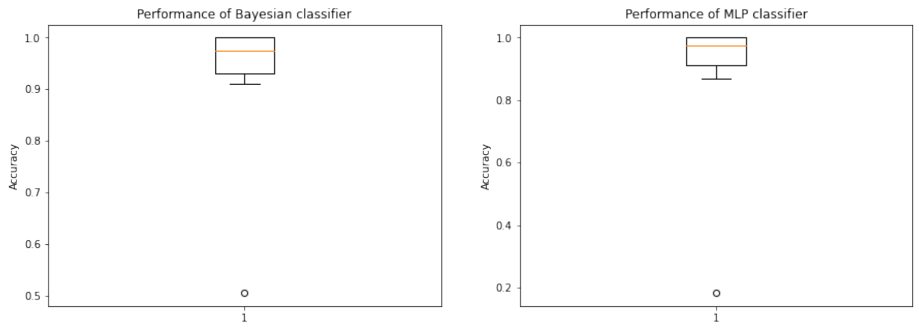


Figure 10: Box-plots of the classifiers

3.3 Class boundaries of the classifiers

Figure 11 and 12 show the class boundaries of their respective classifiers. Both class boundaries look very similar. All class boundaries are taken at the 8th fold.

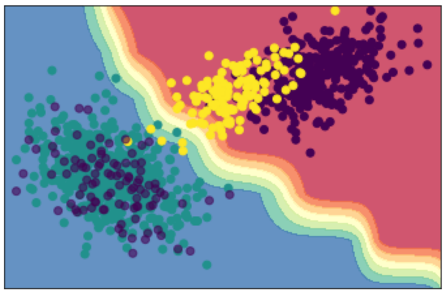


Figure 11: Bayesian class boundaries

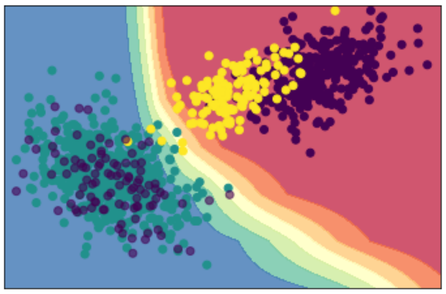


Figure 12: MLP class boundaries

I then compared the class boundaries of a simple MLP (10 hidden nodes) to a complex MLP(600 hidden nodes). As expected, the simple MLP had poorer accuracy than than the complex MLP. Figure 13 shows how the class boundaries have not been separated properly whereas figure 14 shows a clear separation in classes.

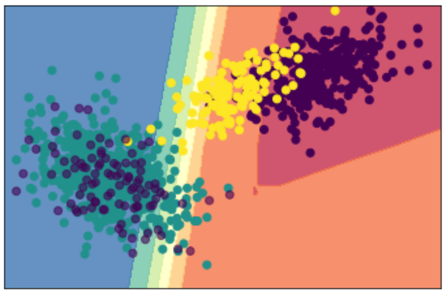


Figure 13: Simple MLP

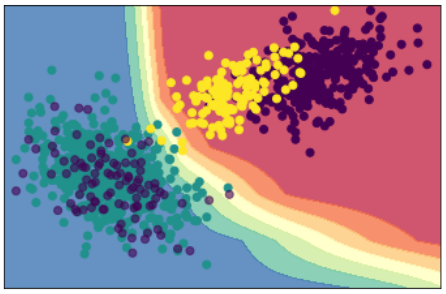


Figure 14: Complex MLP

3.4 MLP classifier parameters

The parameters I experimented with MLP are: shuffle and early_stopping. From the documentation, default setting has shuffle = True (shuffles samples in each iteration) and early_stopping = False (stops training when validation score does not improve). As you can see by figures 15, 16 and 17 there is no significant change. The changes in parameters are similar and different to the default setting. Changing shuffle did not affect classification accuracy but changing early_stopping made the classification slightly worse. Both parameters are only effective when the parameter solver = 'sgd' or 'adam' which are both stochastic gradient based weight optimization.

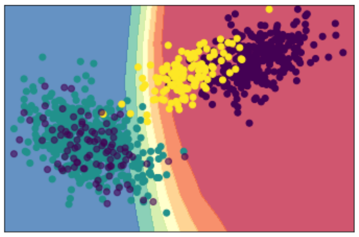


Figure 15: default settings

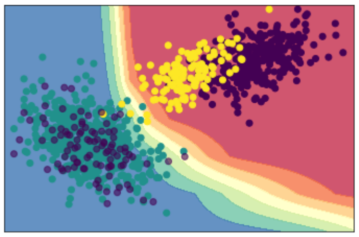


Figure 16: shuffle = False

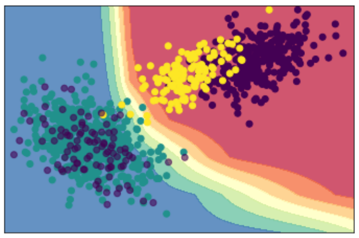


Figure 17: early_stopping = True