4.2

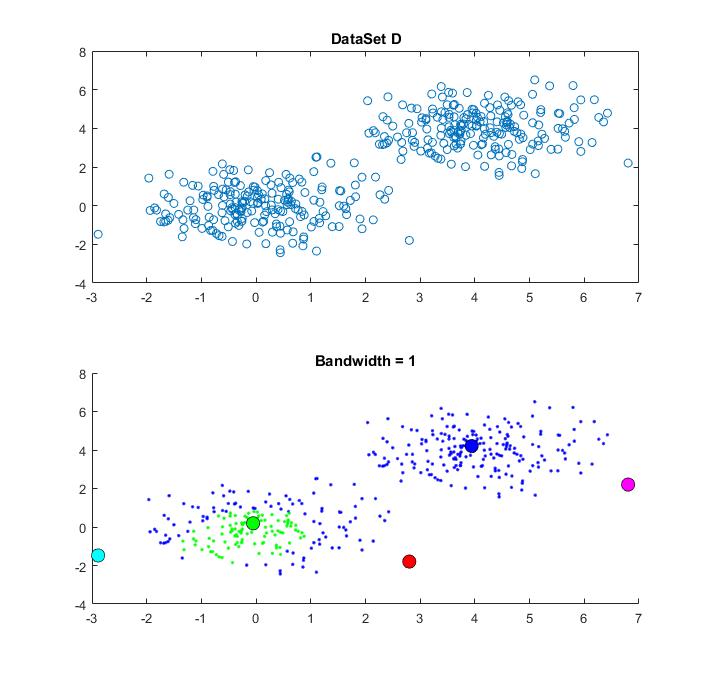
**Presetting the radius of flat kernel function to be radius(bandwidth) = 2**

Matlab Code:

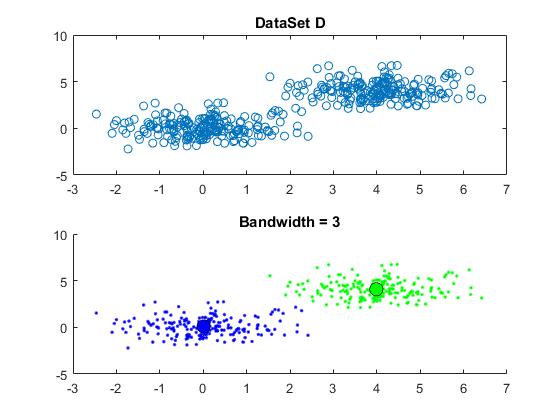




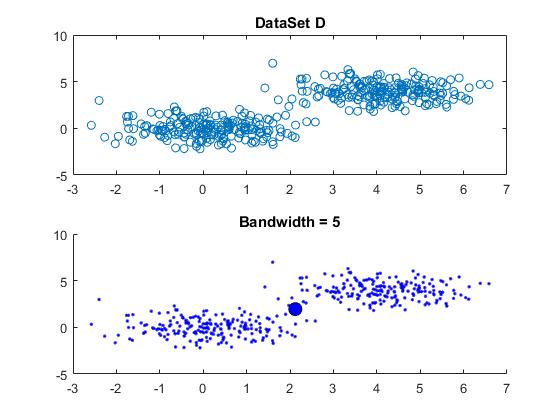
4.3

(1). Dataset D, choose radius(bandwidth) value r = 1, 3, 5, we get the result as

When bandwidth = 1, it can be seen that the number of clusters generated is 5, which is not we expected. The issue is that the dataset is consist of two multivariate Gaussian models with different mean value, which tends to be two clusters intuitively, however, the radius of kernel function is too small so that it cannot cover all the data points on the figure, which results in the extra points (especially edging point) will form a new cluster with very few points.

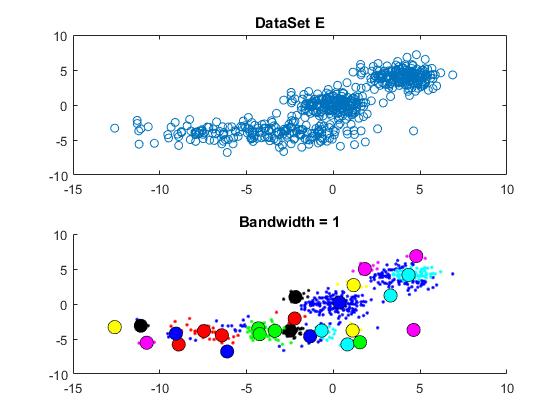


When bandwidth = 3, it can be seen that the number of clusters generated is 2. This model performs well under this bandwidth.

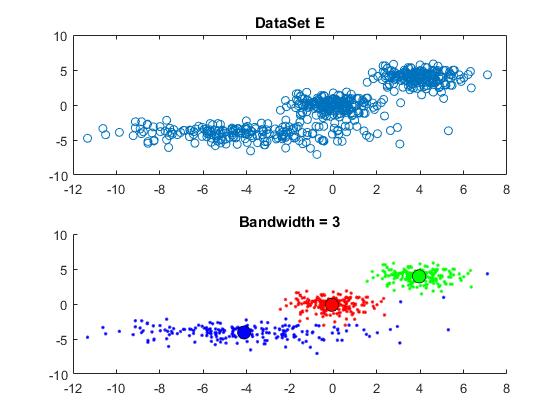


When bandwidth = 5, it can be seen that the number of clusters generated is 1, which is too few in this case. That’s because the bandwidth of the kernel function is too large, which makes the initial cluster cover all the data points so there are no points out of range of first cluster and the loop end.

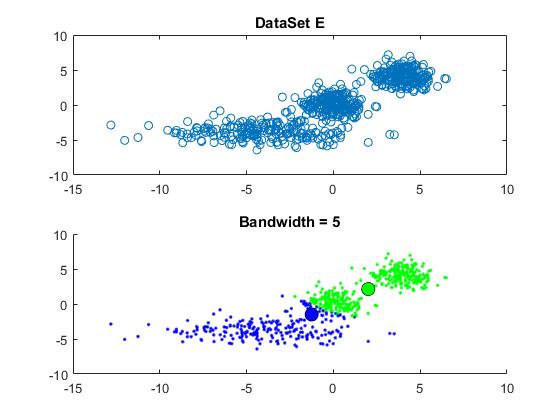
(2). Same as dataset d, now we try to modelling dataset e by the same radius (bandwidth) r = 1, 3, 5



Firstly, we can see the dataset E is consisting mainly from three regions, which expecting to see the number of clusters to be 3. When bandwidth = 1, it can be seen that the model generating a lots of clusters, and each one of them only representing few data points, which is due to the smaller value of bandwidth and it’s not we expected.



When bandwidth is equal to three, it can be seen that three clusters have been generated, and each of them have a clear centre which represent the most density part in that cluster. It is a good model to representing this dataset since we expected the clusters numbers to be three intuitively.



When bandwidth is equal to five, it can be seen there are only two clusters in this case, which is not we expected since it does not successfully distinguish the top two clusters, and the reason for that is the bandwidth of kernel function are too wide, same as dataset d.

Attached: mean shift algorithm test code



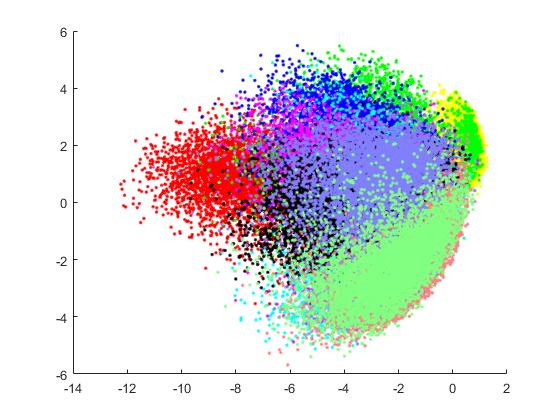
5.1

Principal Component Analysis with reduced dimension d = 2:

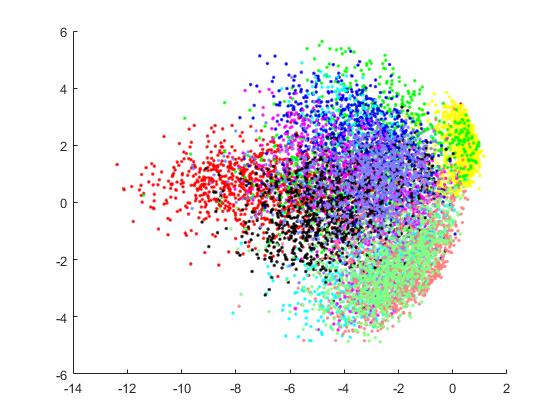


5.2

(1). The reduced data plotted in a two-dimension as shows (mnist\_train)



Testing the PCA with mnist\_test, we can get reduced data plotted as below.

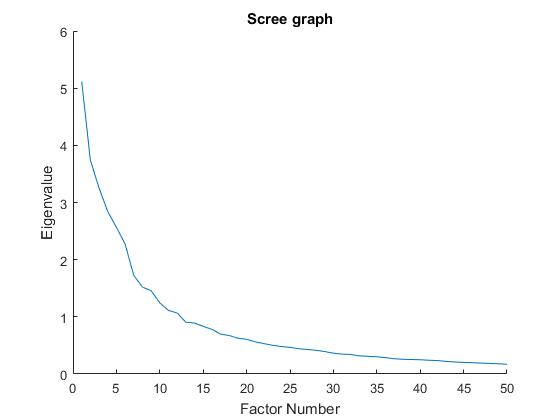


(2). Based on the output for the percentage of eigenvalue, the proportion of variance is calculated as 0.1680 (mnist\_train).

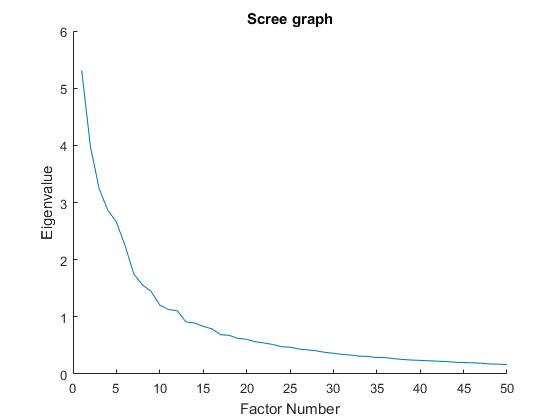
Testing under mnist\_test, the proportion of variance is 0.1759.

(3). Scree graph: Since after 50 factors, the eigenvalue will significantly tends to be 0, so in here it only shows the first maximum eigenvalue to the fiftieth maximum one.

Mnist\_train scree graph:



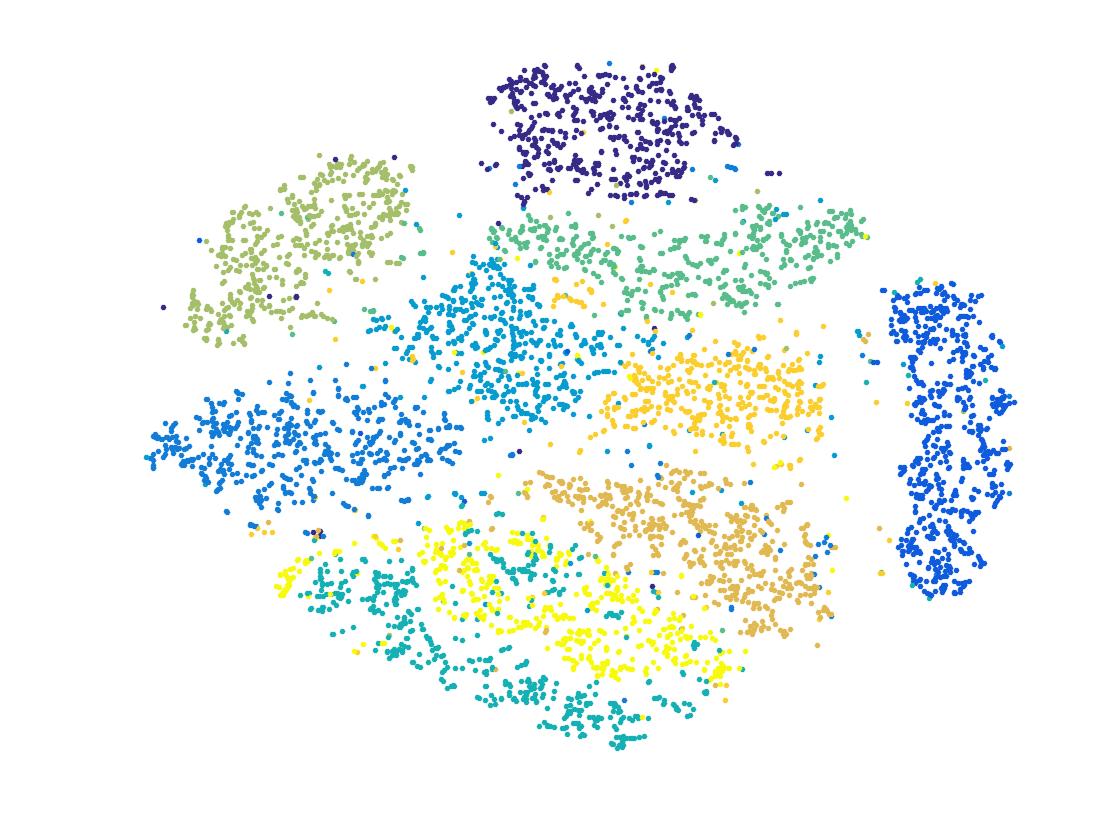
Mnist\_test scree graph:



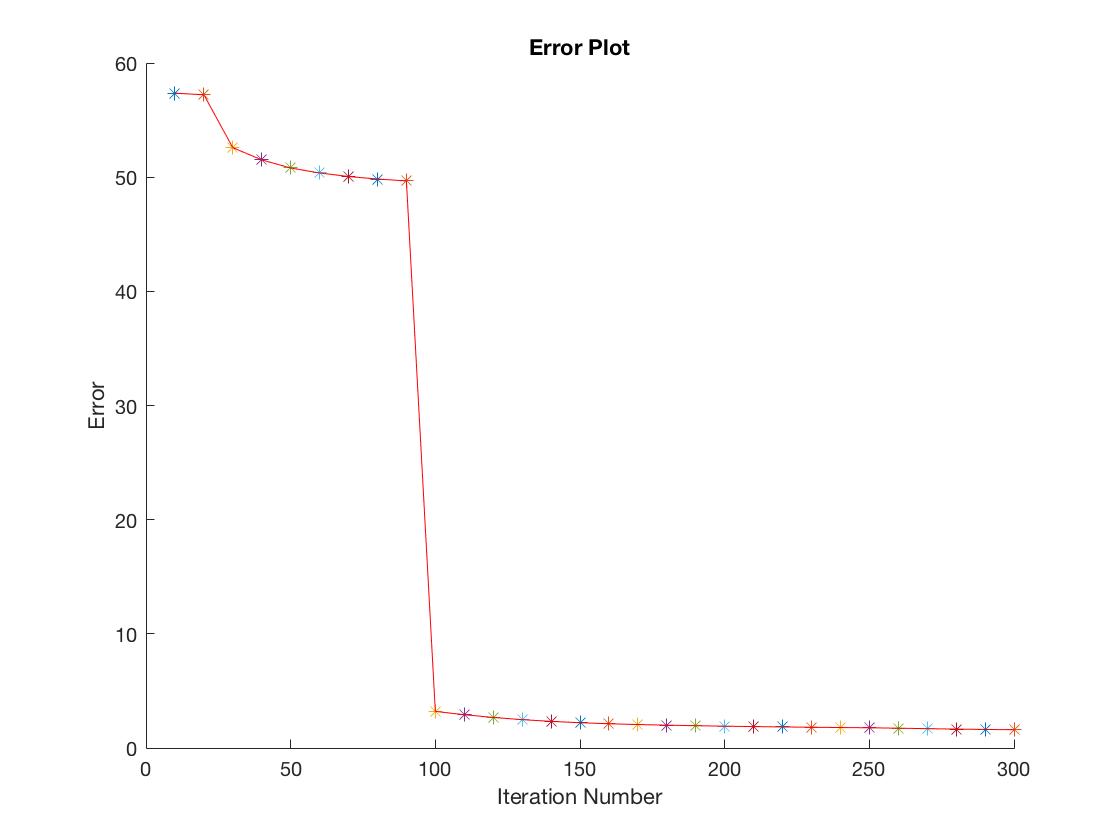
5.6

Running t-SNE algorithm with provided testing code, we can get the plot like below.

The following diagram shows that the data of mnist\_train set be training using t-SNE algorithm when iteration i = 300.

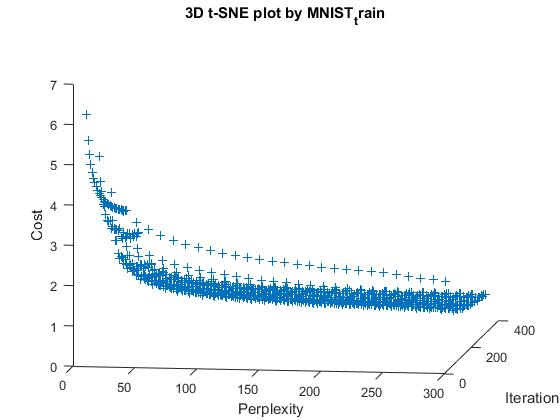


The below plot shows the error at each iteration up to 300 iterations in steps of 10.

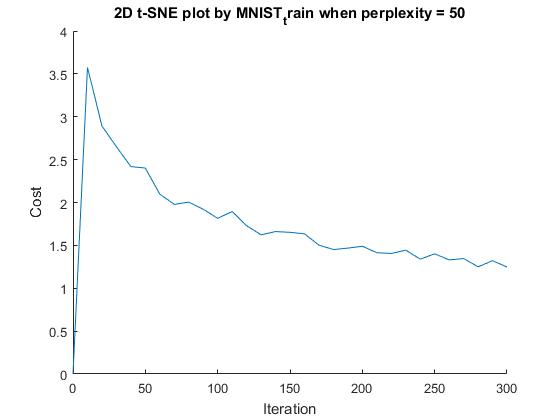


5.8

The 3D plot generated by iteration, perplexity and cost as follow shows.



From above diagram, it can be seen that cost will decrease as the perplexity increasing, however, higher perplexity will cause significant increasing of computation complexity. To balance that, choosing the point where the gradient of cost tends to approaching zero, where the perplexity is 50 as diagram shows. Therefore, we choose 50 as the perplexity value and the following diagram shows the cost vs iteration under perplexity = 50.



6.3

Neutral Network Training under Matlab NN toolbox. (Dataset: MNIST\_train)

(1). Choose Input, Hidden and Output layer.

Input nodes: 784 (Fixed by data)

Output nodes: Since the problem is a classification problem with 10 different classes, so we set the number to be 10 represent 10 different output types.

Hidden layers: According from Marsland(2014), two hidden layers is the most that you every need for normal MLP learning, in fact one hidden layer with lots of hidden nodes is sufficient. In here, we choose the number of hidden layers to be one for simplify complexity and time saving.

Hidden nodes: Since we only choose one hidden layers, so we need to set more hidden nodes to making the training accurate. Therefore, we set the hidden nodes to be 12.

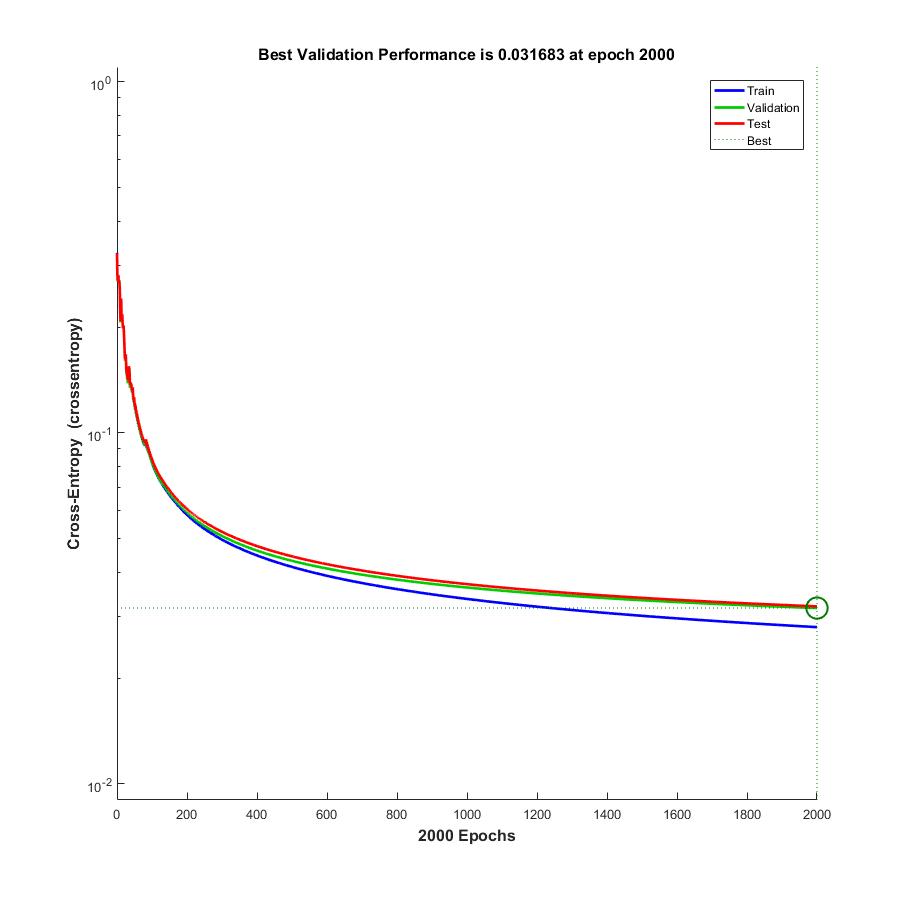
(2). To ensuring the algorithm reliability, we set the maximum iteration of training to be 2000.

(3). Data splitting

To be algorithm reliable, we are using 70% of data for data training, 15% for data validation and 15% for data testing. Using a large proportion of data for training is for ensuring the algorithm accuracy and reliable.

(4). Performance.

To represent data error between data output and target output, we are using cross-entropy to represent the sum-of-square error function. Based on the following diagram, the minimum error of this algorithm, which represents by the best validation performance is 3.168% and still slowly decreasing. If we continue running for some time, this error will be increase again due to overfitting and that’s the threshold time when the algorithm stops running. The performance of training set data won’t be able to affect too much of the validation and testing data after the error goes below 4%, which is causing by the amount of training data (70%) cannot fully cover the rest of data(30%).



MLP Testing Code

