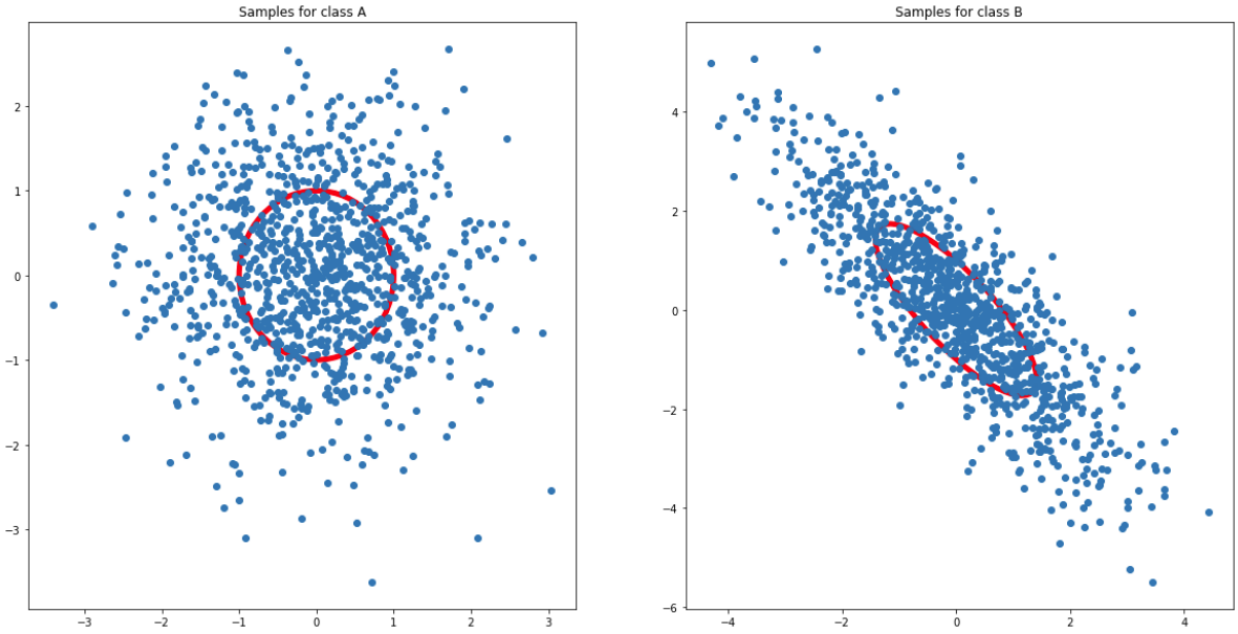
SYDE 675
Assignment 1
Sharhad Bashar
ID: 20474328

1) a) and b)

The 1000 data samples and the first standard deviation contours are below:



The equation for the contour is as follows:

$$\frac{\cos(eigenvector[0])\,(x - mean[0]) + \sin(eigenvector[0])(y - mean[1])^2}{2 * \ eigenvalue[0]^2} + \frac{\sin(eigenvector[0])\,(x - mean[0]) + \cos(eigenvector[0])(y - mean[1])^2}{2 * eigenvalue[1]^2} = 1$$

c) The covariance matrices are as follows:

$$\Sigma_A = \begin{bmatrix} 1.02807 & -0.0268 \\ -0.0268 & 0.9730 \end{bmatrix} \quad \Sigma_B = \begin{bmatrix} 2.1606 & -2.1060 \\ -2.1060 & 3.0377 \end{bmatrix}$$

d) The generated covariance matrices are slightly different compared to the given covariance matrices. This is because the 1000 data points is an approximation of the distribution of the continuous features. As the number of data samples increase, the closer they will be to the given covariance matrices. They will be the same if there are infinite data samples

2) a) The decisions boundaries[1], first standard deviation contours and the data points for ML and MAP classifiers are shown below (next page):
The difference between ML and MAP classifier is the use of the log(P(C)) which is used to calculate $g_x(k)$ for MAP, but not for ML. MAP considers the prior probabilities P(C) in its calculation, where as ML only compares $P(x|C_i)$ while calculating the decision boundaries. Using

the extra log term, we see a shift in the boundaries between the different classes in MAP vs ML. This extra term also makes the MAP classifier more accurate

b) The confusion matrices are shown below:

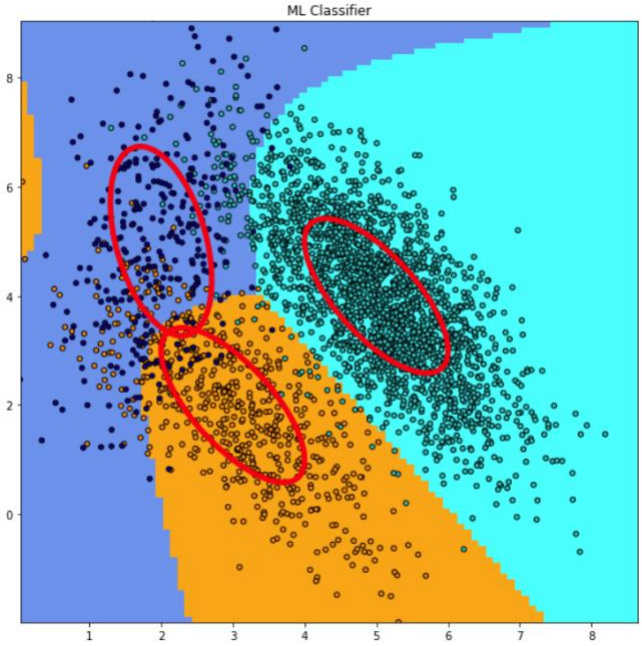
$$ML = \begin{bmatrix} 492 & 5 & 103 \\ 24 & 1991 & 85 \\ 29 & 3 & 268 \end{bmatrix} \; MAP = \begin{bmatrix} 521 & 11 & 65 \\ 13 & 2060 & 27 \\ 57 & 20 & 223 \end{bmatrix}$$

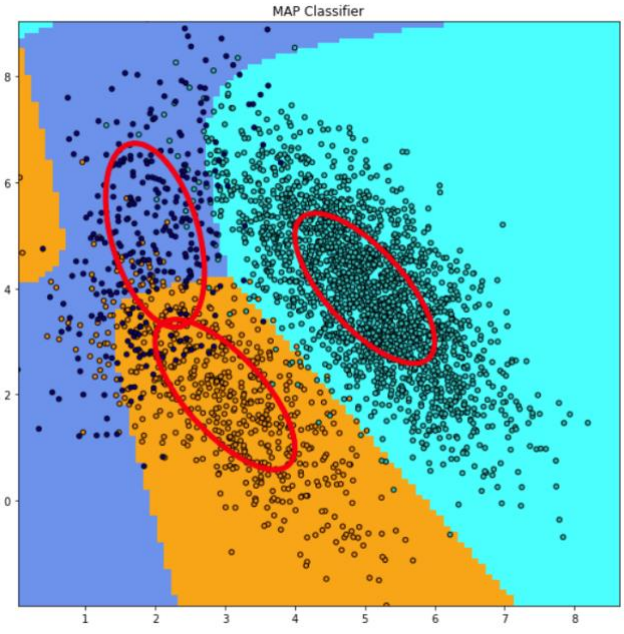
$$P(\varepsilon)_{ML} = 0.083 \; P(\varepsilon)_{MAP} = 0.064$$

The probability error is calculated using the confusion matrices, using the following equation:

$$P(\varepsilon) = \frac{total\ data - sum(diagonal(confusion\ matrix))}{total\ data}$$

As expected, the probability error of MAP is smaller than that of ML, which means more points are correctly classified using MAP than ML.

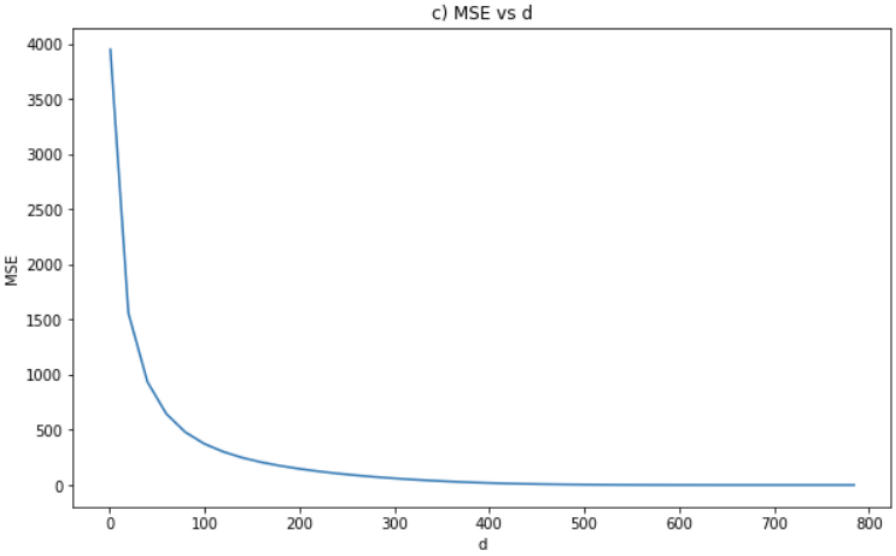




3) a) See def PCA(self) in 3.py

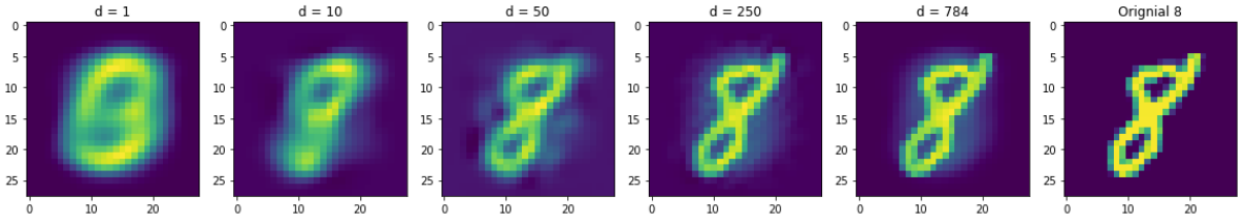
b) A suitable d for POV = 95% is 154 (see def POV(self) in 3.py)

c) See the graph below



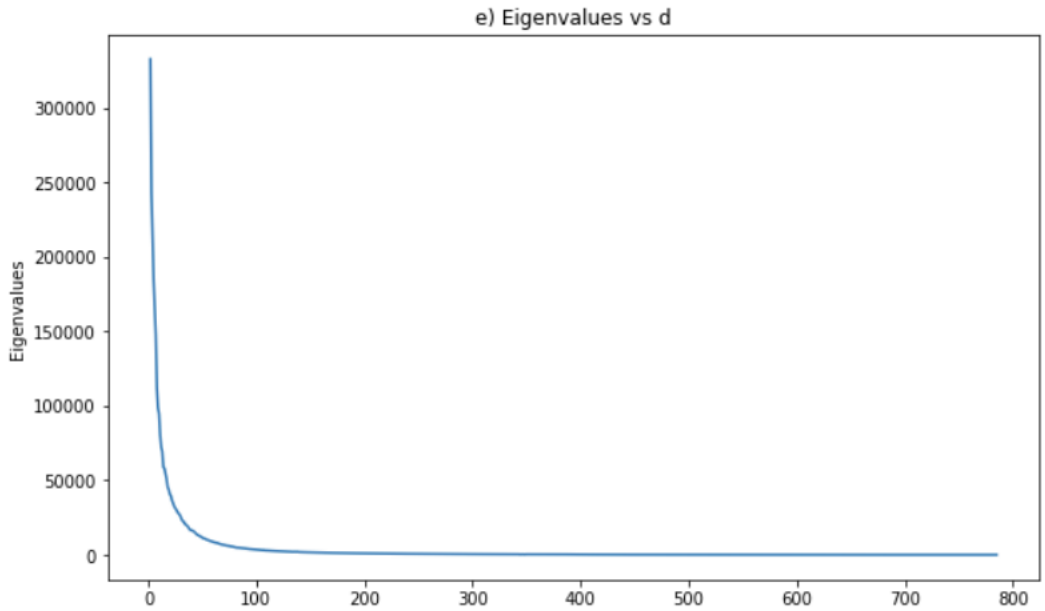
As d increases, our MSE decreases. This means the more eigenvectors we use, the more accurate our results are. If we use all the eigenvectors, our MSE is 0.

d) See def reconstructTarget (self) in q3.py



This confirms our answer from part c. As d increases, the reconstructed image becomes clearer. For d = 784, where we are using all the eigenvectors, we get the clearest image. Also showing the original image beside it

e) See def plotE (self) in q3.py



Since the covariance matrices are semi-positive definite, all the eigenvalues are greater equal to zero. The graph shows, as d increases, eigenvalues get smaller. Eigenvalues get closer to 0 as d goes over 154, which is the value we got in part b).

Data Processing for q3 done using this GitHub repo[2]

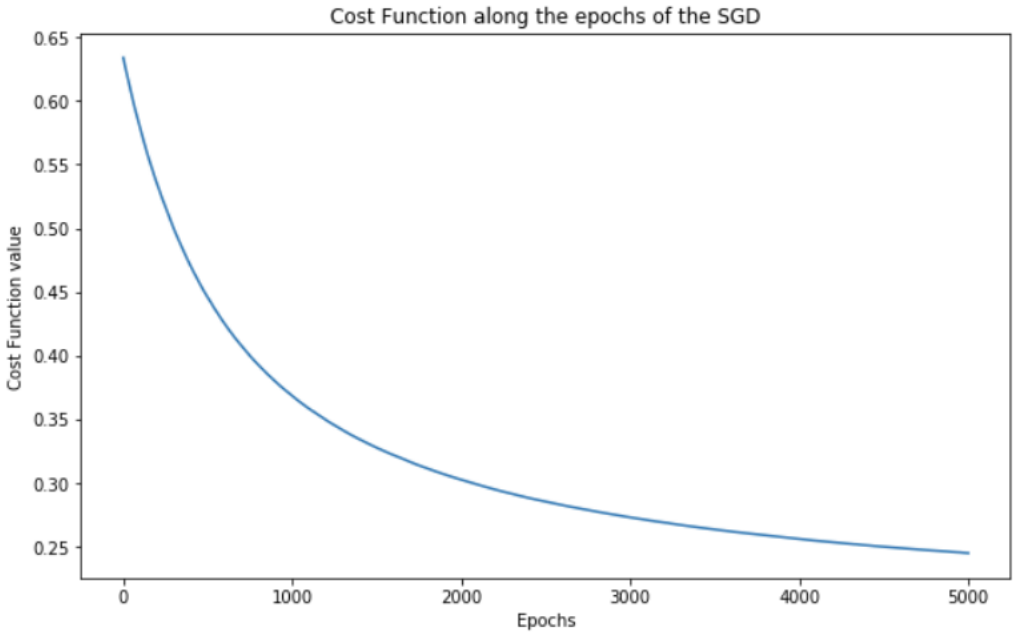
4) a) Cost function:

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)]$$

$$h_\theta\left(x^{(i)}\right) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$

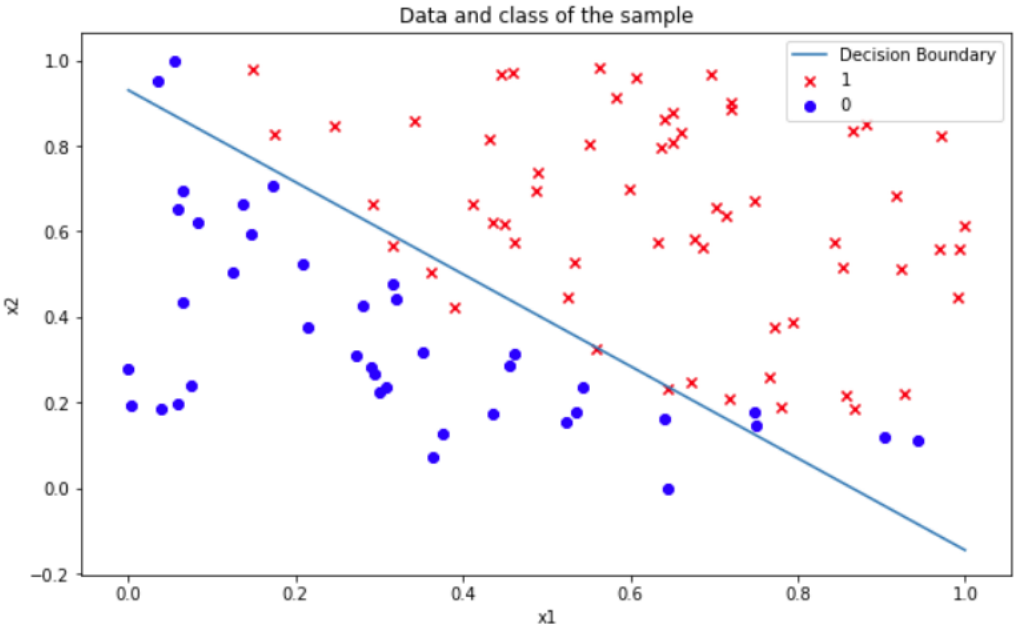
b) See q4.py

c) See the plot below:



d) Accuracy is 89%. See q4.py

e) and f) See the plot below:



5) a) The Naive Bayes classifier is an approximation to the Bayes classifier.
Bayes classifier is seen as a decision rule. Suppose we want to classify a data point given a set of features. Bayes classifier first finds all data points with the same features. It then determines

what class of those data points have in common. Then it assigns the most common class to that data:

$$argmax\ P(C = c\ |F_1 =\ f_1, F_2 =\ f_2, \dots, F_k =\ f_k)$$

However, if there are many features, it is unlikely to find data that match exactly the same features. This is where Naive Bayes comes in. Naive Bayes assumes conditional independence. In Naive Bayes, we compute the probabilities of each feature occurring in a class independently:

$$argmax\ P(C = c) \prod_{i=1}^{k} P(F_i =\ f_i\ |C = c)$$

b) According to part a, Naive Bayes and Bayes are equivalent when all the features in the dataset are independent of one another.

c) Bayes classifier can be practically used (as we did in q2) when we know the conditional probabilities of all combinations. This requires the data set (number of features k) to be small

d) For Bayes:

$$argmax\ P(C = c)P(F_1 =\ f_1, F_2 =\ f_2, \dots, F_k =\ f_k\ |C = c)$$

This is not tractable because a lot of observations are required to calcualte these conditional distributions since they are depended on each other, and it gets even harder as the number of features, k increases.

For Naive Bayes, we compute the probabilities of each feature occurring in a class independently:

$$\prod_{i=1}^{k} P(F_i =\ f_i\ |C = c)$$

thus it is tractable.

References

[1] Motivation for creating the decision boundaries taken from here: https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html

[2] Preprocessing the mnist data set taken from here: https://github.com/pjreddie/mnist-csv-png