SYDE 675: Pattern Recognition (Winter 2020)
Assignment 1 Report

Report Submitted by:
Ramanpreet Kaur Brar
SID:- 20801545

Question 1 :- Consider two classes described by the covariance matrices below (assume zero mean)

A. ∑ = [1 0 0 1] B. ∑ = [−22 −32]

a) For each matrix generate 1000 data samples and plot them on separate figures.

Answer:-
Cholesky decomposition is applied to the covariance matrix and random standard normal distribution is generated. 1000 data points are generated for class A and class B. Mean is added to the data points to shift the location for distribution.

The covariance matrix of class A has off-diagonal elements as zero so generated distribution is uncorrelated on the other hand diagonal elements of the class B covariance matrix are negative so distribution is negatively correlated.

Scatter plots are plotted with matplotlib for class A and class B and are shown in figure 1.1. The red plot is for class A and a blue plot for class B.

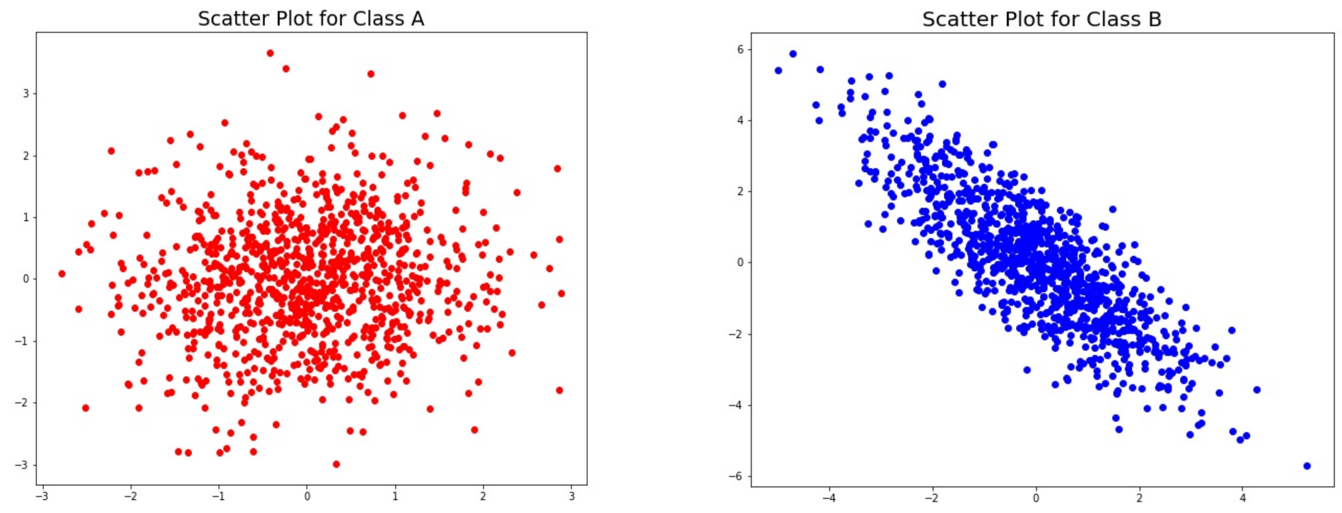

Figure :- 1.1 Scatter plot for class A and class B

b) For each case calculate first standard deviation contour as a function of the mean, eigenvalues, and eigenvectors. Show your calculation (Hint: consider distribution whitening from the tutorial). You may use preexisting functions for Eigen computation. Plot each contour on the respective plots from part (a).

Figure 1.2 shows the scatter plot of data distribution for class A and class B and standard deviation contours are shown as black ellipses.

First Eigenvalues and Eigenvectors of the distribution are calculated. Eigenvectors represent the direction of contour and the square root of Eigenvalues are height and width of the standard deviation contour. Ellipse function from matplotlib is used to plot the contour. The angle of the ellipse is calculated from Eigenvectors. The mean is the center of the ellipse.

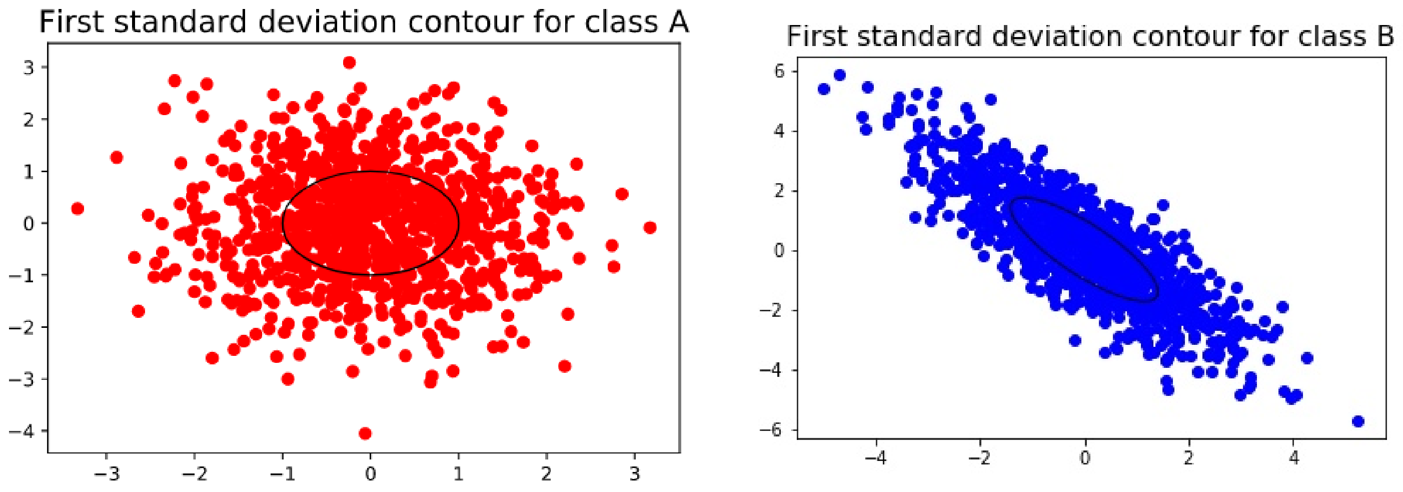

Figure:- 1.2 first standard deviation contour for class A and class B

c) Calculate sample covariance matrices for each class using the data generated in part. Do not use a Python/Matlab function for computing the covariance.

Covariance Matrix for Class A		Covariance Matrix for Class B	
0.98004519	0.02437547	2.0504002	-2.08228778
0.02437547	1.01274745	-2.08228778	3.11281302

Table : 1.1 Sample covariance matrices for class A and class B

$$\text{Cov}(x, y) = \frac{\sum(x - \bar{x})(y - y)}{N - 1}$$

Equation:- 1.1 sample covariance

Sample covariance is calculated as shown in equation 1.1.First, the mean of all the points is calculated and subtracted from the original data matrix. The updated data matrix is multiplied with its transpose. The result of the matrix product is multiplied with 1/(N-1) to compute sample covariance.
Sample covariance is calculated from data points generated in part A. Each covariance is calculated from 1000 data points. Results of the calculated covariance matrix are there in the table 1.1.

d) Explain the difference between the given covariance matrix for each class and the corresponding sample covariance matrix generated in (b). In which condition they can be the same?

Tables 1.2 and 1.3 show the given covariance matrices and the covariance matrices calculated from the generated data points. These are different for two reasons.

First, we are provided with population covariance and we have calculated sample covariance from the generated data points. The population covariance matrix has 1/(N) multiplication factor and the sample covariance matrix has 1/(N-1) multiplication factor. So these results are different.

Second, we have generated only 1000 data samples from the covariance matrix and the mean of each class. If we will generate infinite samples then the actual mean of those samples will be zero and covariance will be equal to the given covariance matrix. Presently they are only 1000 samples for each class so their mean is not exactly zero, it only tends to zero. Even the population covariance matrix of these 1000 samples will not be the same to the given covariance matrix.

Calculated Covariance Matrix for Class A		Given Covariance Matrix for Class A	
0.98004519	0.02437547	1	0
0.02437547	1.01274745	0	1

Table: 1.2

Calculated Covariance Matrix for Class B		Given Covariance Matrix for Class A	
2.0504002	-2.08228778	2	-2
-2.08228778	3.11281302	-2	3

Table: 1.3

Question (2) Consider a 2D problem with 3 classes where each class is described by the following priors, mean vectors, and covariance matrices.

P(C1) = 0.2	**P(C2) = 0.7**	**P(C3) = 0.1**
µ1 = [3 2]*T*	**µ2 = [5 4]*T***	**µ3 = [2 5]*T***
∑1 = [−11 −21]	**∑2 = [−11 −21]**	**∑3 = [0 0..5 5 03.5]**

a) Create a program to plot the decision boundaries for a ML and MAP classifier. Plot the means and first standard deviation contours for each class. Discuss the differences between the decision boundaries.

Figure 2.1 shows the mean, first standard deviation contour and decision boundaries for the MAP and ML classifiers. The red decision boundary is for the ML classifier and the black decision boundary is for MAP classifier. Means and standard deviation contours of the different classes are also shown in the figure. Class 1 is shown in magenta, class 2 is shown in cyan and class 3 is shown in green color.

Decision boundaries are created by calculation the probability of different classes on the entire mash grid and then plotting the contours. I have used the mash grid from 0 to 10 on the x-axis and 0 to 10 on the y-axis with a step size of 0.05. Taking a smaller step size will create smother boundaries.

$$g_k(x) = -\frac{1}{2}(x-\mu_k)^T \Sigma_\mathrm{k}^{-1}(x-\mu_k) - \frac{1}{2}\log(|\Sigma_\mathrm{k}|) + \log(P(C_k))$$

Equation:- 2.1 probability for MAP

Equation 2.1 and Equation 2.2 calculates the probability for ML and MAP classifier. Decision boundary points have equal probabilities for adjacent classes. The contours of MAP and ML have a very similar path but they are not the same because the MAP considers the prior probability but ML doesn’t consider it. All the classes have different prior probability so log(P(c)) term is different for all classes. Class 2 the highest prior probability among all the classes and lowest negative log(P(c)) term so this class has maximum MAP probability.

$$g_k(x) = -\frac{1}{2}(x-\mu_k)^T \Sigma_{\mathrm{k}}^{-1}(x-\mu_k) - \frac{1}{2}\log(|\Sigma_{\mathrm{k}}|)$$

Equation: 2.2 probability for ML

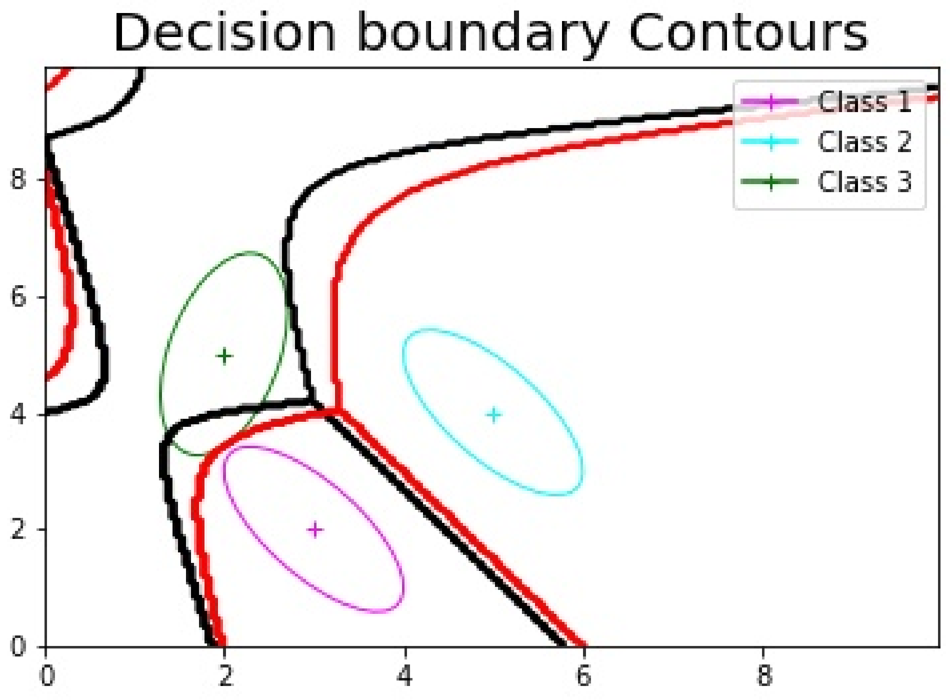

Figure :- 2.1

b) Generate a 3000 sample dataset using the prior probabilities of each class. For both the ML and MAP classifiers: classify the generated dataset, calculate a confusion matrix, and calculate the experimental P(ε). Discuss the results.

Cholesky decomposition is applied to the covariance matrix and random standard normal distribution is generated. 3000 data points are generated for all the classes. A scatter plot for generated data is shown in figure 2.2. Data points in each class are proportional to the prior probability of a class. Class 1 is shown in magenta, class 2 is shown in cyan and class 3 is shown in green color.

The confusion matrix for ML and MAP classifier is shown in table 2.1 and 2.2. MAP has classified most of the classes better than the ML classifier. MAP has 93.6% accuracy on the data on the other hand ML classifier has 91.87% accuracy. MAP is also considering the prior probability so it is better at classifying the data. The accuracy of the models varies with randomly generated data.

The results of experimental P(e) is shown in table 2.3. Class 1 has the highest error for all the classifiers and class 2 has the lowest error for both classifiers. MAP classifier is performing better for class 1 and class 2. ML has better performance for class 3.

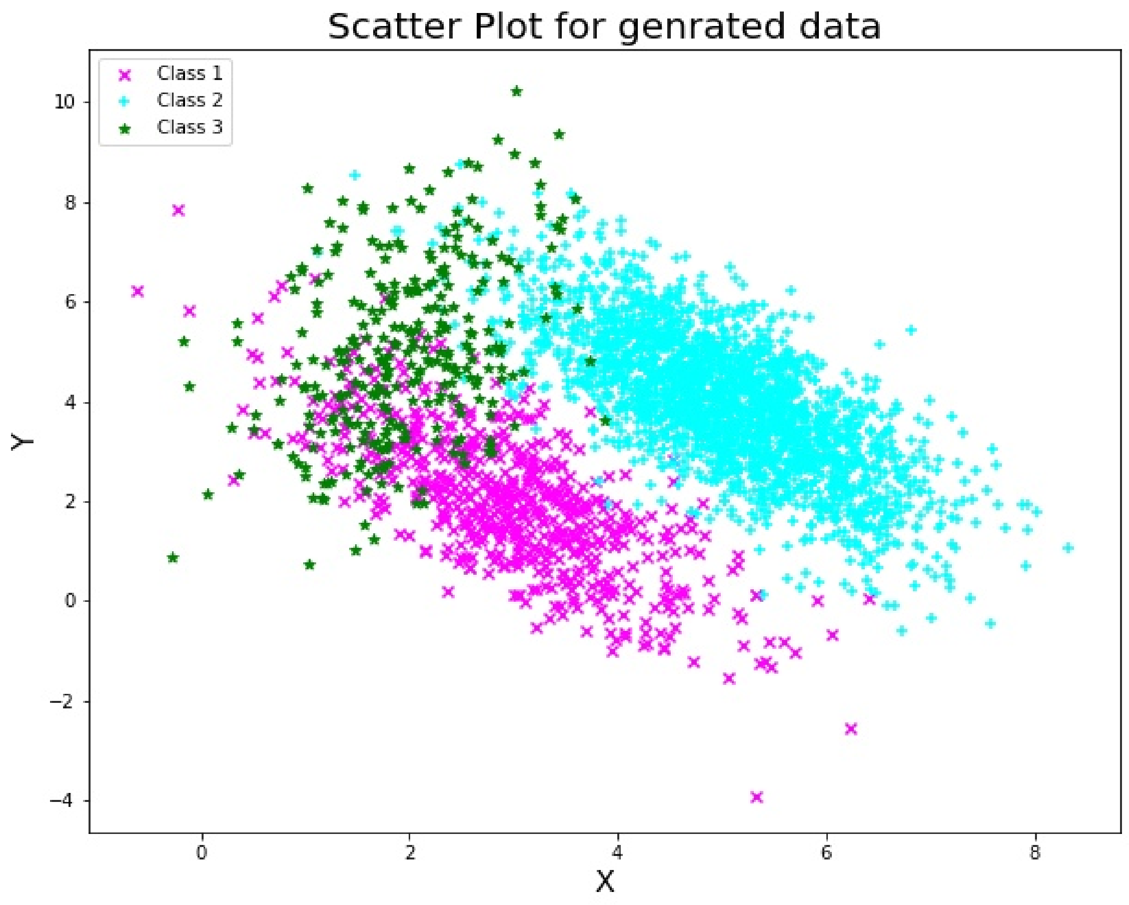

Figure:- 2.2 Scatter plot for class 1 , class 2 and class 3

Confusion Matrix for MAP classifier			
Actual class	Predicted class		
	C1	C2	C3
C1	518	17	65
C2	11	2068	21
C3	49	29	222

Table:- 2.1 Confusion Matrix for MAP

Confusion Matrix for ML classifier			
Actual Class	Predicted class		
	C1	C2	C3
C1	476	11	113
C2	24	2008	68
C3	21	7	272

Table:- 2.2 Confusion Matrix for ML

Experimental P(e)			
Classifier	C1	C2	C3
MAP Classifier	0.125	0.019	0.273
ML Classifier	0.180	0.058	0.100

Table:- 2.3 Experimental P(e)

The MNIST dataset contains a set of images containing the digits 0 to 9. Each image in the data set is a 28x28 image. The data is divided into two sets of images: a training set and a testing set. The MNIST dataset can be downloaded from http://yann.lecun.com/exdb/mnist/. Use only the training set to perform this part.

a) Program PCA that takes X(DxN) and returns Y(dxN) where N is the number of samples, D is the number of input features, and d is the number of features selected by the PCA algorithm. Note that you must compute the PCA computation method by yourself. You may use preexisting functions for Eigen computation.

PCA is used for dimension reduction. The MNIST training set has 6000 data points and each has 784 features. So the input data passed to the function is X(784X6000) and d is number features selected after PCA. Eigenvalues represent the variance of data in the direction of corresponding Eigenvectors. So d Eigenvectors with maximum Eigenvalues are used for dimension reduction.

As we are not using any builtin function for computing PCA, so I have explained the steps used for computing PCA. The first step for PCA is to calculate the mean for each feature and subtract it from the dataset. Then the covariance matrix is calculated. Covariance matrix for MNIST data set is 784X784. Eigenvalues and Eigenvectors of the covariance matrix are calculated. Sort the Eigenvectors in descending order of their Eigenvalues. Return the Eigen matrix which has d the Eigenvectors which has the highest Eigenvalues.

b) Propose a suitable d using proportion of variance (POV) =95%.

A value of d with POV more than 95% is calculated from Eigenvalues of Eigenvectors. A suitable value of d with PoV is above 95% is 152. As shown in figure 3.3 Eigenvalues of the Eigenvectors decrease very rapidly with an increase in d. So only 152 out of 784 Eigenvectors contain 95% of the information.

c) Program PCA reconstruction that takes Y(dxN) and returns $\hat{X}$ (DxN) (i.e., a reconstructed image). For different values of d= {1, 20, 40, 60, 80, …, 760, 784} reconstruct all samples and calculate the average mean square error (MSE). Plot MSE (y-axis) versus d (x-axis). Discuss the results.

Image is reconstructed by using reverse PCA and d represents the number of Eigenvector used to PCA. With the increase in values of d eigenvalue decrease. There are only a few Eigenvectors that contribute to the majority of the variance. For example, 152 vectors contain 95% of the information. If we go with very few Eigenvectors there is a larger error. As the number of vector increases, MSE decreases. After certain value of d, there is no benefit of adding more dimensions.

Mean Squared error vs dimensions plot is shown in figure 3.1. Mean squared error decrease with an increase in d initially but after a certain value of d, there is not much improvement. As MSE decreases with an increase in d, this will provide a better reconstruction of the image.

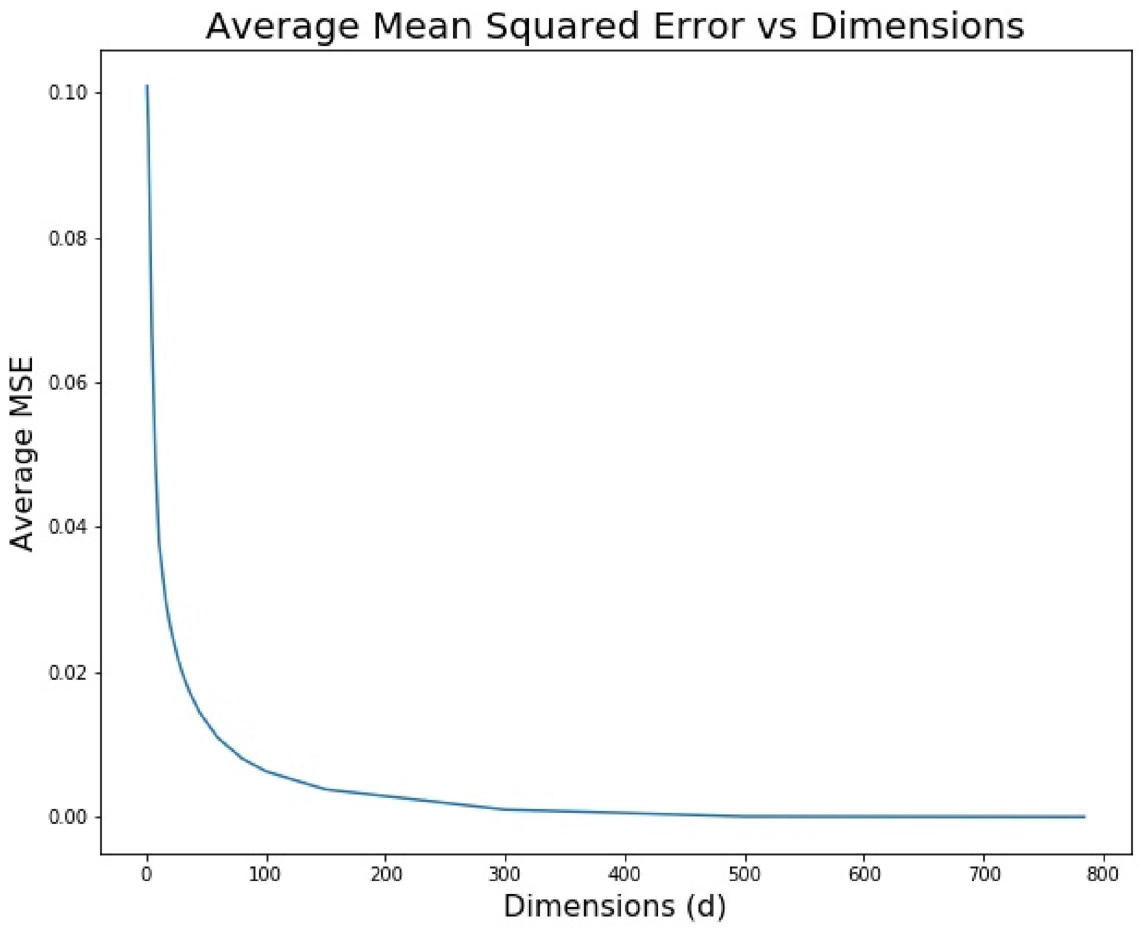

Figure :- 3.1

d) Reconstruct a sample from the class of number ‘8’ and show it as a ‘png’ image for d= {1, 10, 50, 250, 784}. Discuss the results.

The image is reconstructed using reverse PCA. The number of dimensions d is the main deciding factor for the quality of the reconstructed image. With very small d like 1 or 10 image is not clear. As we increase the value of d the image clarity improves. As discussed in part b of this question 152 Eigenvectors contain around 95% of the information. So the image with d=250 contains most of the information and there is no visible improvement from d=250 to d=784.

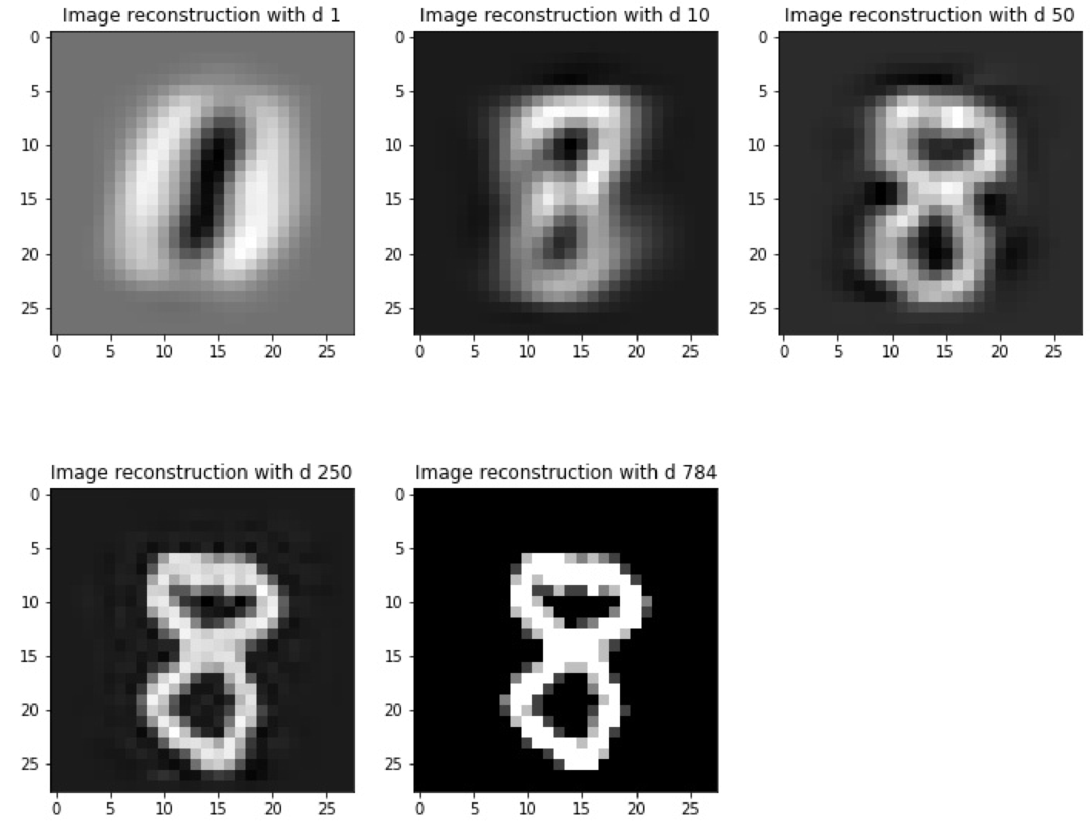
Image reconstruction with d 1
Image reconstruction with d 10
Image reconstruction with d 50
Image reconstruction with d 250
Image reconstruction with d 784

Figure :- 3.2 Image reconstruction of 8 with different values of d

e) For the values of d= {1, 2, 3, 4, …, 784} plot eigenvalues (y-axis) versus d (x-axis). Discuss the results.

As shown in the figure there are very few eigenvectors that have considerable variance in their direction. Variance in the direction of the rest of the Eigenvectors is close to zero. As we have already seen that 152 vectors out of 784 contain 95% of the information, so vectors from 153 to 784 have only 5% information. This is because only the center portion of images have digits and some variance. Corners of most of the images don’t have any variance. That’s why there are only a few Eigenvectors with non zero Eigenvalues.

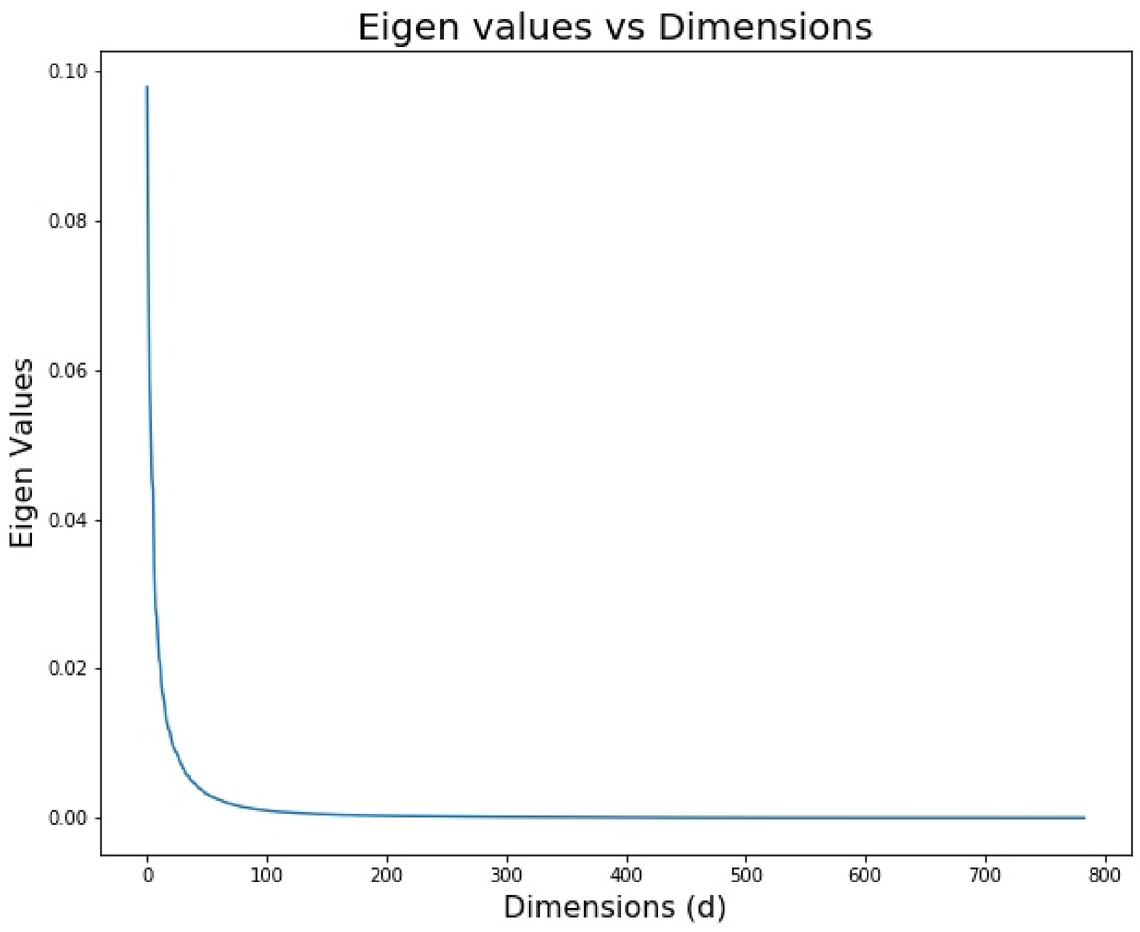

Figure :- 3.3 Eigenvalues vs dimensions

Question 4 :- Consider the attached file dataset3.txt. The first two columns of the data file show the feature of each sample and the last column illustrates its corresponding binary level.
a) What is the cost function in logistic regression?

Equation 4.1 is the cost function for logistic regression. This is a combinations of two logarithmic functions and a convex function. Here h(x) is a sigmoid function as represented in equation 4.2 which is used to calculate the probability of class. Here 'm' represents the number of samples in the dataset.

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

Equation:- 4.1

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

Equation: - 4.2

b) Estimate the parameters using stochastic gradient descent (SGD) method. You need to implement the SGD function for the optimization.

Gradient descent is used to minimize some function by iteratively moving in direction of steepest descent. The cost function of logistic regression is concave as shown in equation 2.1 so we try to calculate the minima for this function. For this logistic regression, we have used a sigmoid function and that has an equation as the power of e.

g(x) = theta0 + theta1*f1 + theta2*f2

As we are not using in build function for gradient descent so stochastic gradient descent has been implemented. Stochastic gradient updates the theta on every sample. The flow diagram for the algorithm of stochastic gradient descent is shown in the figure below.

step size = gradient * learning rate.

Here we need to calculate three random parameters theta0, theta1, and theta2. First, start with the random values of all the theta. Calculate the gradient for each theta. Calculate the step size to update theta. The step size at which we will change the parameters depends on the learning rate. Calculate the step size and update all the parameters together. Calculate the gradient again by plugging in the updated parameters. Wait until the function converges. After some point there is no further decrease in the cost then we can stop and assume that function has reached very close to its lowest value.

I have applied scaling on the input feature. My gradient descent has converged better after scaling. Value of parameters is theta0 = [0.57788545] , theta1= [4.09797773], theta2 = [3.69592299]] for scaled data.

```
Repeat until convergence
for $i = 1 : m$
        $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_0} J^{(i)}(\theta_j)$
        (for every $\theta_j$)
end
```

Algorithm flow diagram for SGD

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

Cost function for logistic regression

c) Plot the cost function along the epochs of the SGD.

Figure 4.1 shows the graph of SGD cost function vs number of epoches. As the number of epoches increase the cost is decreasing. This in the situation when learning rate in good and gradient decent is converging if learning rate is high then gradient descent oscillates.

I am calculating the cost at the end of every epoch but theta is updated on every sample in SGD.

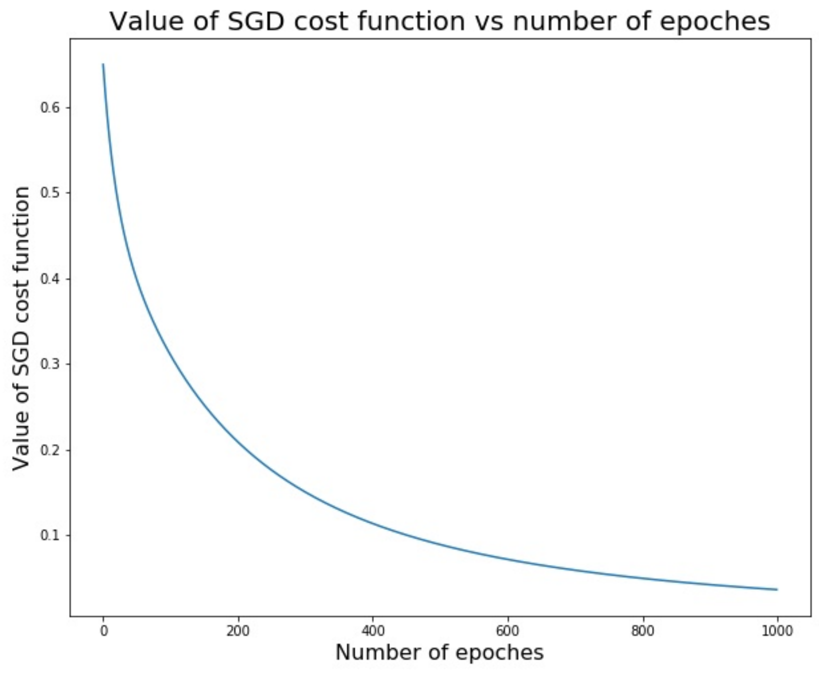

Figure:- 4.1

d) Use the learned model to classify all training samples and report the accuracy.

I have trained the logistic regression model on all the input and they that model is used to calculate the accuracy. My model is giving me 91% accuracy. I have calculated accuracy as the percentage of correctly identified samples out of total samples.

e) Plot the data and show the class of the sample using different colors.

Figure 4.2 shows the scatter plot of 2 classes with two features. Class 0 is represented with blue and class 1 represented by orange color. Both of these classes are linearly separable. g(x) is an equation of decision boundary for this distribution where f1 and f2 are feature1 and feature2 respectively. Theta0, theta1, and theat2 are the parameters computed from SGD.

g(x) = theta0 + theta1*f1 + theta2*f2

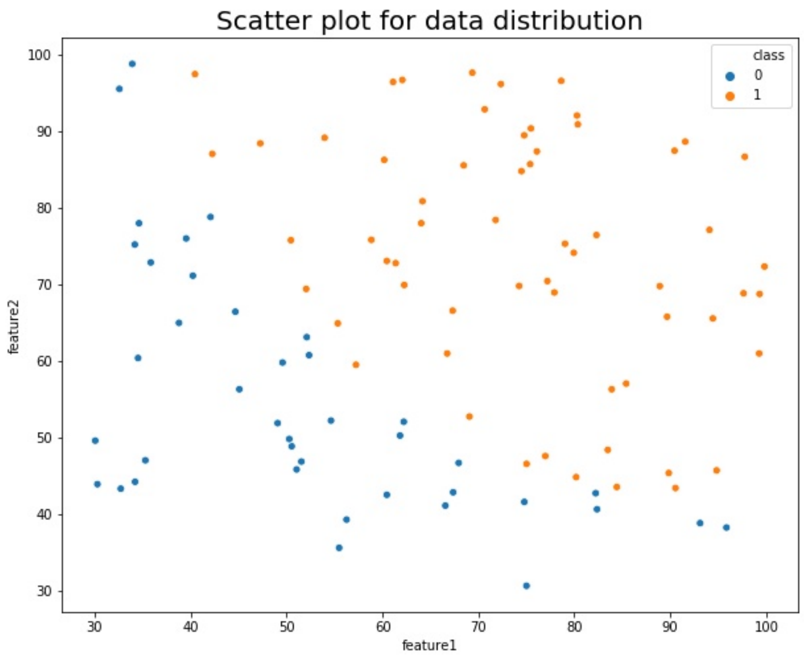

Figure:- 4.2

f) Plot the Decision boundary of the classifier.
Decision boundary for the classifier is shown in figure 4.3. This is a straight line with equation.

g(x) = theta0 + theta1*f1 + theta2*f2

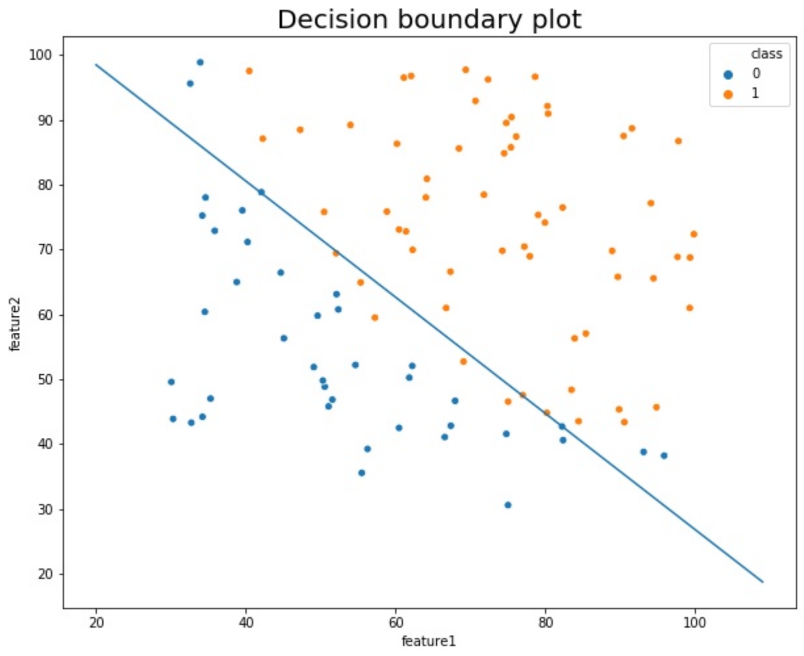

Figure :- 4.3

5) Naïve Bayes Discussions:

a) What is the difference between Bayes and Naïve Bayes classifiers?
Naive Bayes classifier is approximation of Bayes classifier. In naive Bayes classifier we assume feature are independent for given class. So in naive Bayes we only model for independent model. On the other hand Bayes classifier has no such assumption.

b) In which situation, Naïve Bayes is equivalent to Bayes?

If all the features used for the class label are independent then both Naive Bayes and Bayes classifiers will give the same results. In Naive Bayes, we assume that features are independent. If that assumption is true then there is no difference in the results of both classifiers.

c) In practice, Bayes classifier is not tractable in many applications. Explain, when the Bayesian classifier can be practically used?
Bayes classifier is optimal but this has issues. With a large number of features, Bayes classifier is computationally expensive to compute. For 2 class labels and n feature, Bayes classifier has 2(2^n -1) parameters for modeling. The Bayesian classifier can only be used when there is a small number of features.

d) Discuss why Bayes classifier is not tractable while Naïve Bayes is tractable.

Naive Bayes makes the assumption of conditional independence which significantly reduces the computational complexity. Which makes Naive Bayes more tractable than the Bayes classifier.

For example for 2 class labels and n feature, Bayes classifier has 2*(2^n-1) parameters for modeling, on the other hand, Naive Bayes has only 2*n parameters for modeling. Bayes classifier has an exponential increase in computational terms with an increase in the number of features. That makes Bayes classifier very difficult to use for more number of features. On the other hand, the computational complexity for Naive Bayes increases linearly with the increase in the number of features so Naive Bayes is tractable.