Mohita Chaudhary
20830560

ASSIGNMENT 1: SYDE 675

Solution 1:

A. The 1000 randomly generated samples for the given covariance matrices.

A_sigma=[[1,0],[0,1]], B_sigma=[[2,-2],[-2,3]]

Data generated for A_sigma=[[1,0],[0,1]]

The data points for the unit covariance matrix were generated by using the python numpy function np.random.standard_normal(mean.size), where the mean was assumed to be zero as given in the question.

Data generated for B_sigma=[[2,-2],[-2,3]]

The data points for the B covariance matrix were generated using the eigenvalues and eigenvectors method. In this method the samples generated from the np.random.standard_normal(mean.size) function are transformed using the eigenvalue and eigenvector for the B covariance matrix using the following formula:

mean + Eigenvector * np.sqrt(Eigenvalue) @ np.random.standard_normal(mean.size)

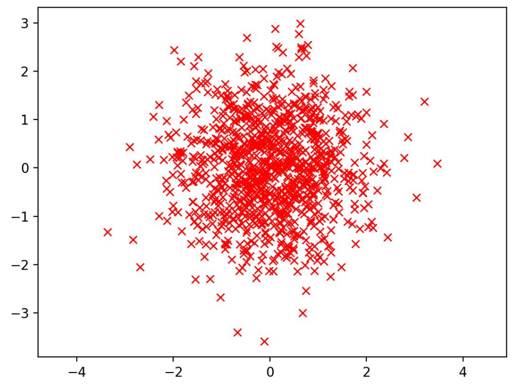


Fig1: Data generated for A_sigma

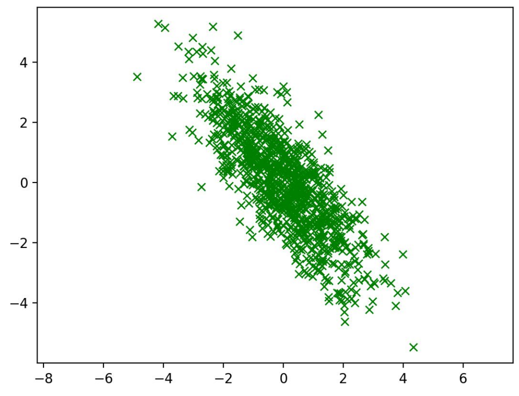


Fig2: Data generated for B_sigma

B. The first standard deviation contour as a function of mean, eigenvalues and eigenvectors for the given covariance matrices.

The first standard deviation contour for the unit covariance matrix would be a simple circle with unit radius with equation $x_1^2 + x_2^2 = 1$, where x_1 and x_2 represent the spread of the coordinates of the randomly generated points. The contour was plotted using the following python function:

```
ax.contour(x1, x2, vector_function(x1, x2),[0],colors='black')
```

Mohita Chaudhary
20830560

Here, the vector_function returns the vectorized equation of a circle which can operate on the set of multiple values.

The first standard deviation contour for the covariance matrix B is generated using distribution whitening method with the help of the following equation:

```
val_2*((a11*x1+a12*x2)**2) + val_1*((a21*x1+ a22*x2)**2) - 1
```

Here, val_2 and val_1 are the eigenvalues for the given covariance matrix and a11,a12, a21 and a22 are the values for the eigenvector. Here, since the mean is assumed to be zero, hence the center of the contour is (0,0). The spread of the contour is defined by the eigenvalues and the angle of the contour is defined with the values of eigenvectors.

If we say that x' and y' are the elements of covariance obtained after the transformation using the distribution whitening method, then we can obtain x' and y' using the below calculation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} a_1^1 & a_1^2 \\ a_1^2 & a_2^2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{\text{val\_1}}} & 0 \\ 0 & \frac{1}{\sqrt{\text{val\_2}}} \end{pmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} a_1^1 x + a_1^2 y \\ a_1^2 x + a_2^2 y \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{\text{val\_1}}} & 0 \\ 0 & \frac{1}{\sqrt{\text{val\_2}}} \end{pmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} \dfrac{a_1^1 x + v_1^2 y}{\sqrt{\text{val\_1}}} \\ \dfrac{a^2{}_1 x + v_2^2 y}{\sqrt{\text{val\_2}}} \end{pmatrix}$$

Here val_1 and val_2 are the eigenvalues and a11,a12, a21 and a22 are the values for the eigenvector. We finally obtain the equation of circle as follows:

$$\left(\frac{a_1^1 x + a_1^2 y}{\sqrt{\text{val\_1}}}\right)^2 + \left(\frac{a_1^2 x + a_2^2 y}{\sqrt{\text{val\_2}}}\right)^2 = 1$$

We use the above equation to obtain the contour for sigma_B. Also, since we follow a normal distribution therefore, the contour is centered at (0,0). Following are contours which were obtained.

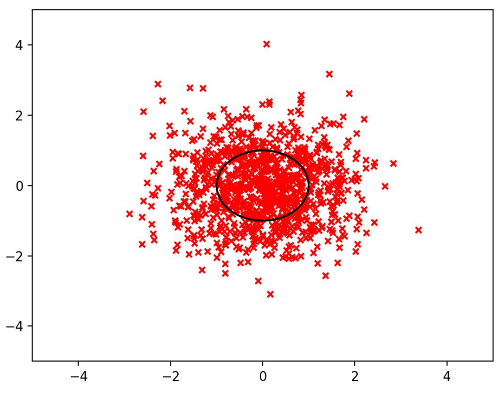


Fig3: First Std dev Contour for A_sigma

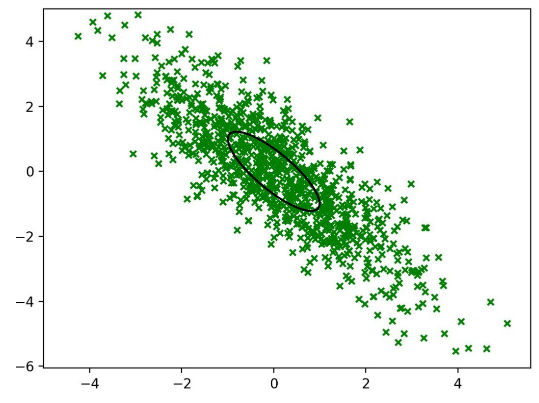


Fig2: First Std dev Contour for B_sigma

C. Sample Covariance matrices for the generated data.

To calculate the sample covariance matrix for the generated data the following equation was used:

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{n-1}$$

The covariance matrix is an indication of how two variables are related. If the value of covariance is greater than 0.5 and positive then it indicates that the variables directly correlated, on the other hand if the value is negative then they are negatively correlated. The diagonal elements of the covariance matrix represent the variance of the two variables, whereas the other elements are the representative of the covariance between the two variables.

The value n-1 in the above formula is because of the bias which we get on calculating the sample covariance. To avoid the bias we multiply the covariance by n/n-1. The n in numerator is cancelled by n in denominator and we are left with n-1.

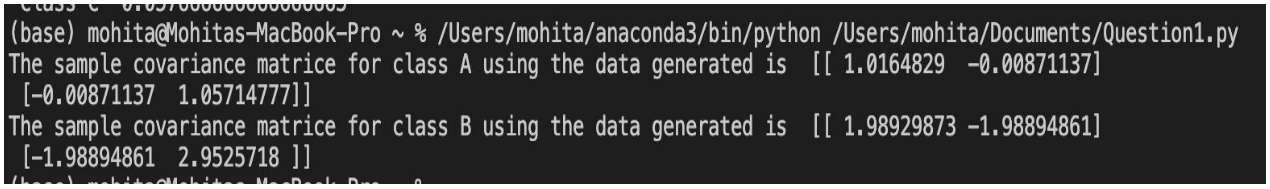
Below, is the screenshot of the value of covariance values obtained for the generated data.

A = [[1.0164829 , -0.00871137] , [-0.00871137 , 1.05714777]]

B = [[1.98929873 , -1.98894861] , [-1.98894861 , 2.9525718]]

We observe that the values of these obtained covariance matrices are quite similar to the given covariance matrix.

Mohita Chaudhary
20830560


```
(base) mohita@Mohitas-MacBook-Pro ~ % /Users/mohita/anaconda3/bin/python /Users/mohita/Documents/Question1.py
The sample covariance matrice for class A using the data generated is  [[ 1.0164829  -0.00871137]
 [-0.00871137  1.05714777]]
The sample covariance matrice for class B using the data generated is  [[ 1.98929873 -1.98894861]
 [-1.98894861  2.9525718 ]]
```

D. The difference between the given and the calculated covariance matrix is that the given covariance matrix is the covariance for the actual data, which we tried reproducing using the eigenvalues and eigenvectors. The computed covariance matrix is the covariance for that generated data. We observe only a very minute difference between the given covariance matrix and the covariance matrix for the generated data. The reason for this observation is that the data which we generated using the Eigenvalues and Eigenvectors were nearly equal to the original data whose covariance matrix was provided to us. This shows that the method which we used for generating the values is correct.
The covariance matrices can be the same if we generate the data for an infinite number of points. In this case we generated only for 1000 and hence we spotted a minute difference between the given and calculated covariance.

Solution 2:
A.
We use the MAP and ML to generate the decision boundaries:
The general discriminant function for Maximum a Posteriori(MAP) is:

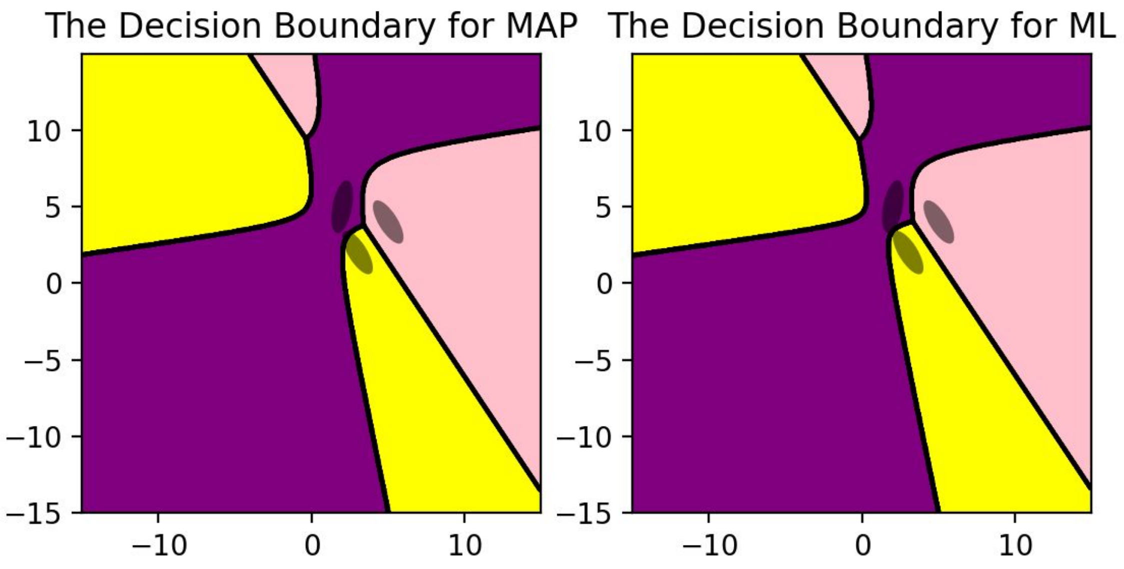
$$g_k(x) \ = \ -\tfrac{1}{2}(x-u_k)^T\Sigma_k^{-1}(x-u_k)-\tfrac{1}{2}log(|\Sigma_k|) \ +log(P(C_k))$$

Here, $log(P(C_k)$ is the prior probability.
The general discriminant function for theMaximum Likelihood(ML) is:

$$g_k(x) \ = \ -\tfrac{1}{2}(x-u_k)^T\Sigma_k^{-1}(x-u_k)-\tfrac{1}{2}log(|\Sigma_k|)$$

where , x is the data sample, u_k is the mean of some class k and Σ_k is the covariance matrix.
To obtain the decision boundaries we use a sampling grid which is uniformly distributed. We put all the points in the grid into the above equation for all the three classes and whichever equation gives the best result the corresponding class is assigned to that point. The only difference between MAP and ML classifiers is that the ML classifier maximizes the likelihood without taking the class probabilities into consideration, whereas the MAP classifier tries to maximise the posterior probability using the Bayes formula. Following are the decision boundaries for MAP and ML classifiers:

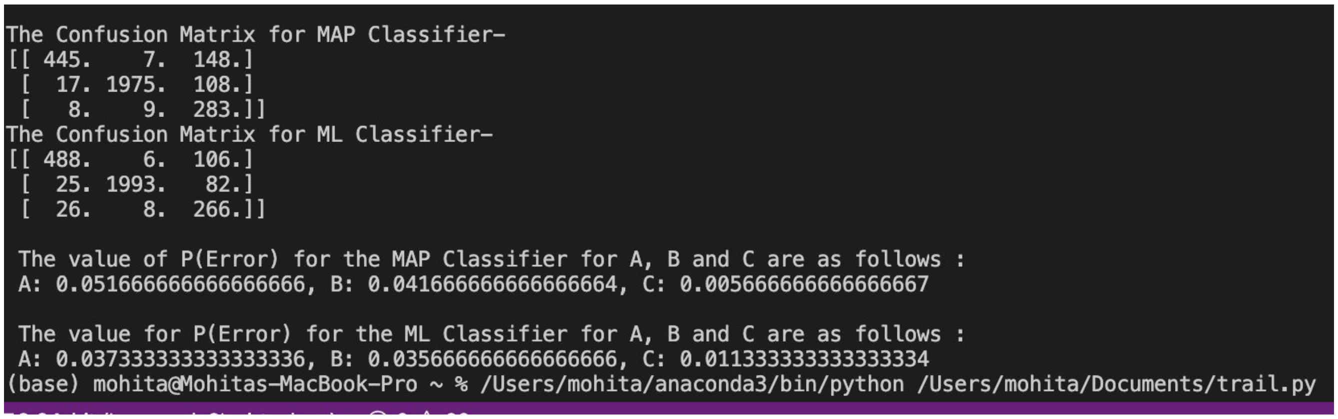
Mohita Chaudhary
20830560



B.

Confusion Matrix and P(E)

Following are the values of confusion matrix for both ML and MAP classifiers. Also, the value of the experimental P(error) for both the classifiers for A, B and C classes is in the below screenshot.



```
The Confusion Matrix for MAP Classifier-
[[ 445.    7.  148.]
 [  17. 1975.  108.]
 [   8.    9.  283.]]
The Confusion Matrix for ML Classifier-
[[ 488.    6.  106.]
 [  25. 1993.   82.]
 [  26.    8.  266.]]

 The value of P(Error) for the MAP Classifier for A, B and C are as follows :
 A: 0.05166666666666666, B: 0.04166666666666664, C: 0.005666666666666667

 The value for P(Error) for the ML Classifier for A, B and C are as follows :
 A: 0.037333333333333336, B: 0.035666666666666666, C: 0.011333333333333334
(base) mohita@Mohitas-MacBook-Pro ~ % /Users/mohita/anaconda3/bin/python /Users/mohita/Documents/trail.py
```

Mohita Chaudhary
20830560

The Confusion matrix was calculated from scratch by checking the predicted labels and comparing them with the original labels. The true positives, false positives, true negatives and false negatives were calculated and outputted. Also, the error for each classifier was calculated.

```
The Confusion Matrix for MAP Classifier-
[[ 445.    7.  148.]
 [  17. 1975.  108.]
 [   8.    9.  283.]]
The Confusion Matrix for ML Classifier-
[[ 488.    6.  106.]
 [  25. 1993.   82.]
 [  26.    8.  266.]]
```

```
The value of P(Error) for the MAP Classifier for A, B and C are as follows :
 A: 0.051666666666666666, B: 0.041666666666666664, C: 0.005666666666666667

 The value for P(Error) for the ML Classifier for A, B and C are as follows :
 A: 0.037333333333333336, B: 0.035666666666666666, C: 0.011333333333333334
```

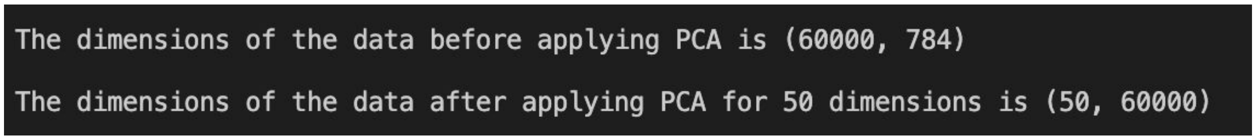
Solution 3:

A.

The Principal Component Analysis is used for the purpose of dimensionality reduction. The PCA takes in the data with high dimension and reduces the dimension of the data to reduce the space as well as computational complexity. Following are the steps in a PCA algorithm, we first feed it with a high dimensional data of dimensions (DxN) where D is dimensions and N are the number of samples. The dataset is first centered around the mean and then the covariance of this centered data is calculated. We then calculate the eigenvalues and eigenvectors of the covariance matrix and sort them in the descending order with the absolute values. The higher the eigenvalue, the more the variance retained in when transformed from a higher to a lower dimension. An appropriate value of smaller dimension d is chosen and then the first d sorted eigenvalues are stored in a vector. We finally take the dot product of these sorted d values with the mean centered original data to get a reduced dimension data (dxN).

A function for PCA was written and was fed MNIST data with original dimensions(60000x784). I tested my function to reduce the dimension from 784 to 50 and the shape of the outputted data (50 x 60000) was obtained. Following is the screenshot of the obtained output.

Mohita Chaudhary
20830560



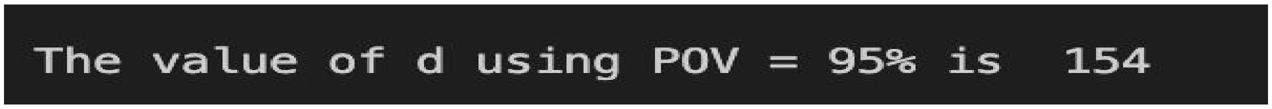
```
The dimensions of the data before applying PCA is (60000, 784)

The dimensions of the data after applying PCA for 50 dimensions is (50, 60000)
```

B.
In order to calculate the POV the following equation was used:

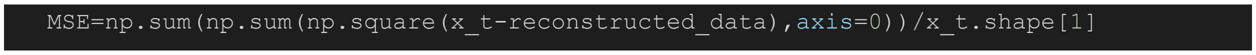
$$POV = \frac{Sum\ of\ first\ d\ sorted\ Eigenvalues(Sorted\ in\ decreasing\ order)}{Sum\ of\ all\ the\ Eigenvalues}$$

The value of d for which we got POV as 95% was 154.
Therefore, the suitable d for obtaining the Proportion of Variance as 95% was obtained as 154.



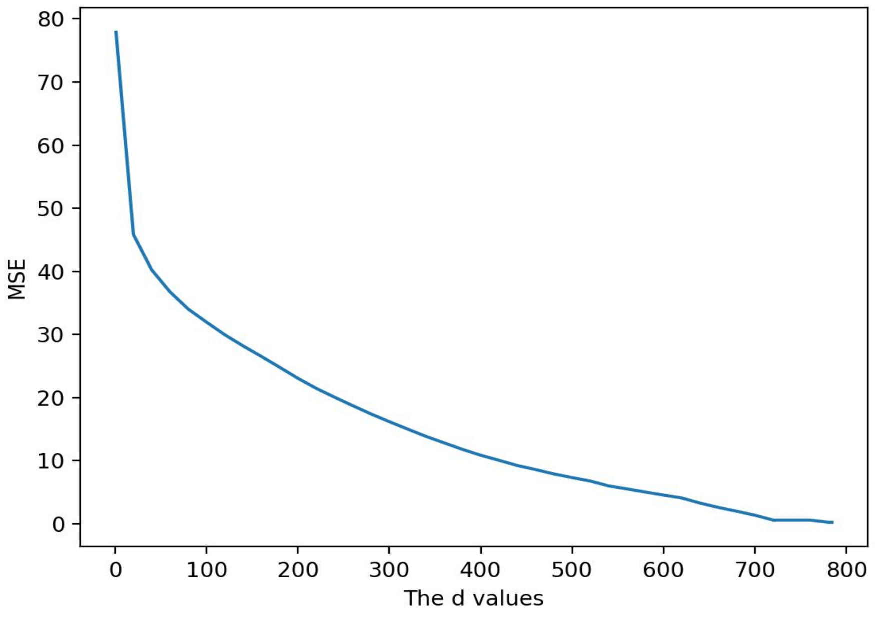
```
The value of d using POV = 95% is  154
```

C.
For reconstructing the data we take the input as the reduced dimension from the PCA function which we created in part(A). We then find the dot product of this reduced data with the first reduced dimension d Eigenvectors sorted in the decreasing order. We obtain the reconstructed image of dimension **(DxN)** by multiplying the eigenvectors of the dimension **(Dxd)** with the compressed image of dimensions **(dxN)**. Then, the Mean Squared Error is calculated between this reconstructed data(for the given values of d) and the original data using the following formula:



```
MSE=np.sum(np.sum(np.square(x_t-reconstructed_data),axis=0))/x_t.shape[1]
```

Where x_t is the original data and reconstructed_data is the data which we reconstructed using the compressed data.
Following is the graph we obtained for MSE for different values of d from 1 to 784.



D.

We tried to reconstruct the image of the digit 8 for the various dimensions d={1, 10, 50, 250, 784}. We observe that on increasing the number of dimensions we start obtaining a clearer picture of 8. The image is almost blurred for the lower dimensions, for d=784 which is the original dimension we observe that the reconstructed image is almost similar to the original image:

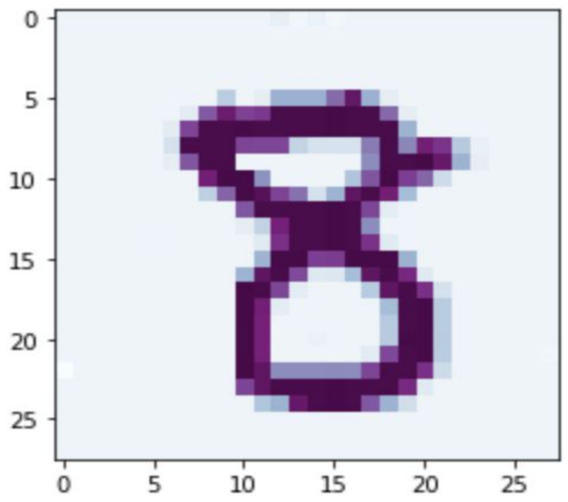


Fig 1: Original Image for 8

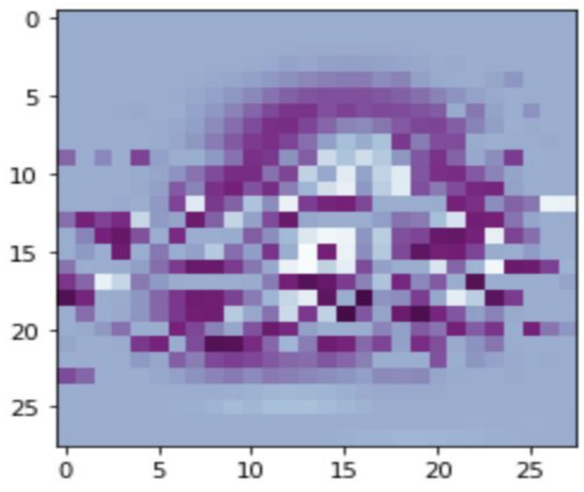


Fig 2: Image of 8 for d=1

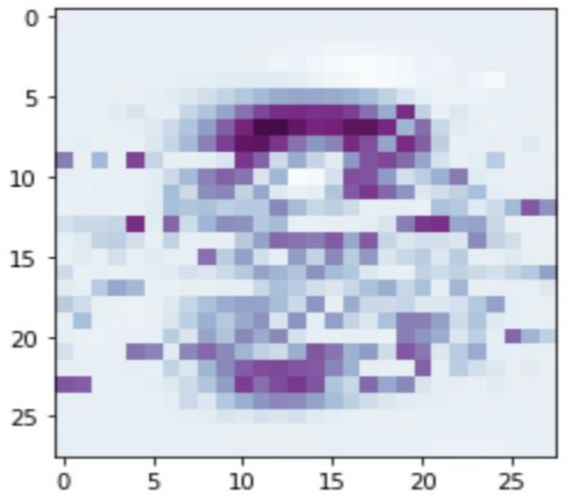


Fig 3: Image of 8 for d=10

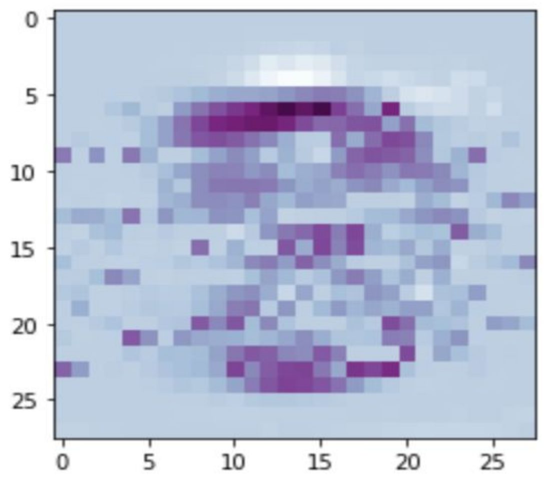


Fig 4: Image of 8 for d=50

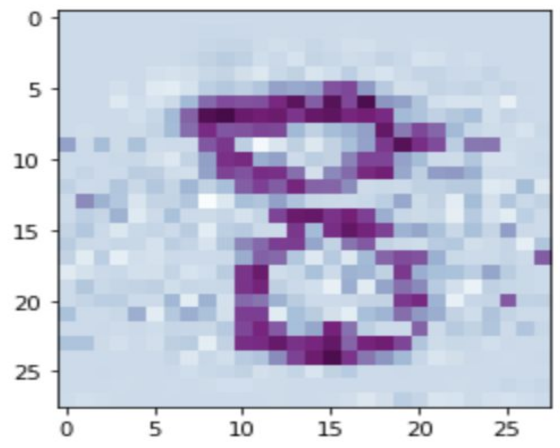


Fig 5: Image of 8 for d=250

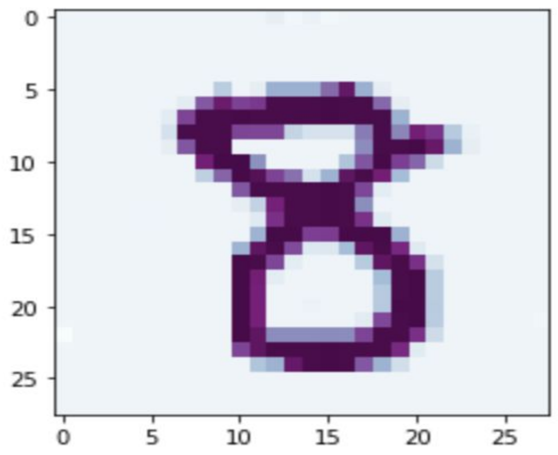
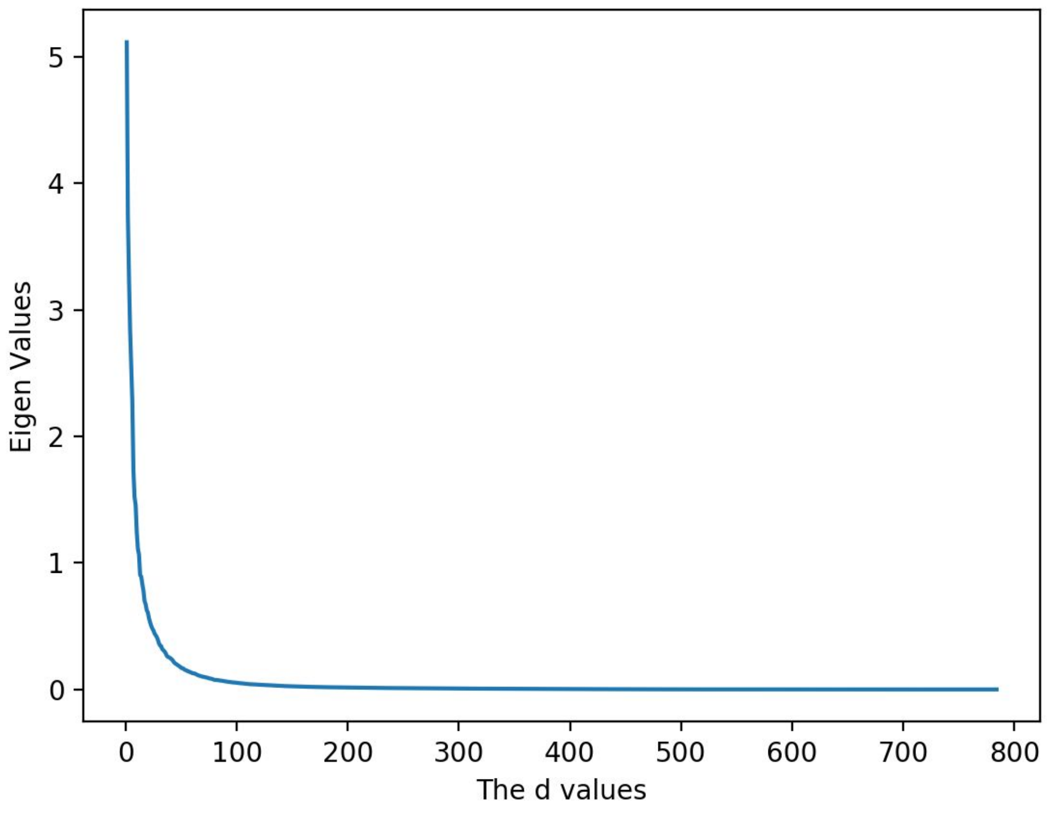


Fig 6: Image of 8 for d=784

E.

The following graph of the EigenValues versus the d dimensions signifies that just a few Eigenvalues and Eigenvectors are responsible for the total variance of any given data. Thus, it is not necessary to retain all the dimensions of the data. The data can b e reduced to some d dimensions which contribute to the maximum variance of the data. For example, here in our experiment we found that only 154 of the first d values (sorted in decreasing order) were

responsible for the 95% of POV. This shows that only a few dimensions are the most significant in any high dimensional data.
Following is the obtained graph for Eigenvalues vs d values.



Solution 4:
A.Cost Function in Logistic Regression
The cost function in logistic regression is the following:

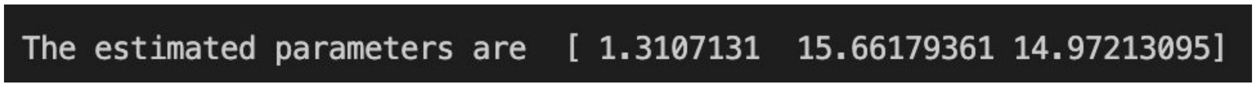
$$Cost(h_\theta(x), y) =$$
$$\{-\,log(h_\theta(x)) \quad if\; y = 1,\; -\,log(1 - h_\theta(x)) \quad if\; y = 0\}$$

This is the cost which has to be paid if our algorithm predicts $h_\theta(x)$ when the actual label is y. If the actual value of the label is 1, then the cost(penalty) approaches to zero as $h_\theta(x)$ approaches to 1. But if the value of $h_\theta(x)$ approaches to 0 then the cost to pay grows to infinity. This is the idea behind the cost function that if the prediction goes far away from the actual value then the

penalty increases. The above function grants convexity to the function which the gradient descent algorithm has to process.
We can write the cost function in a compact form as follows:

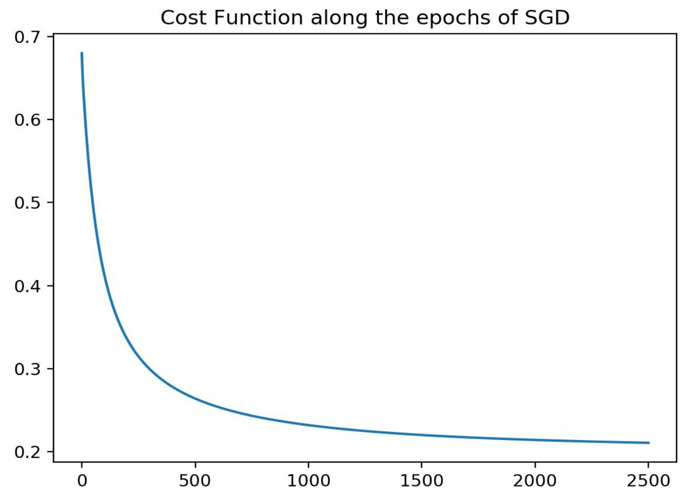
$$J(\theta) =- \frac{1}{m}[\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x))^{i}]$$

B.
The Stochastic Gradient Descent method is used for the purpose of estimating the weights.
For estimating the parameters we begin by initializing the parameters with a default value, then dividing the training data into batches and setting some learning rate(alpha). The gradient for every batch is calculated using $\frac{\delta}{\delta\theta_j} =- \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$, followed by the parameter updation using $\theta = \theta - \alpha X \quad \nabla J(\theta)$. The above process is repeated for all the epochs. We took 2000 epochs for our training and obtained the following parameter values for bias and theta 1 and theta 2 respectively. Following is the screenshot of the result obtained after running the code.



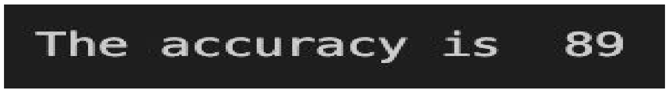
```
The estimated parameters are  [ 1.3107131  15.66179361 14.97213095]
```

C.
Following is the plot for the Cost function versus epochs for Stochastic Gradient Descent. We observe with the increasing number of epochs the value of cost function becomes less, which tells that the accuracy increases on increasing the number of Epochs.



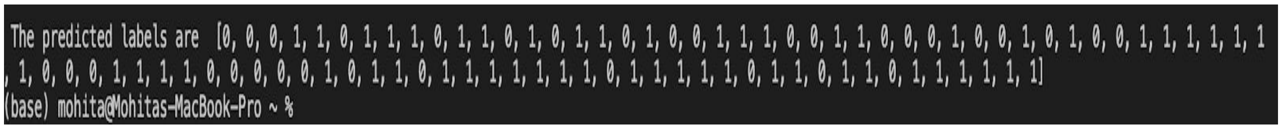
D.

After training the model for 2000 epochs the accuracy I obtained is 89%. Following is screenshot of the output I obtained for accuracy.



```
The accuracy is  89
```

Following are the labels obtained after the prediction. These labels give an accuracy of 89%

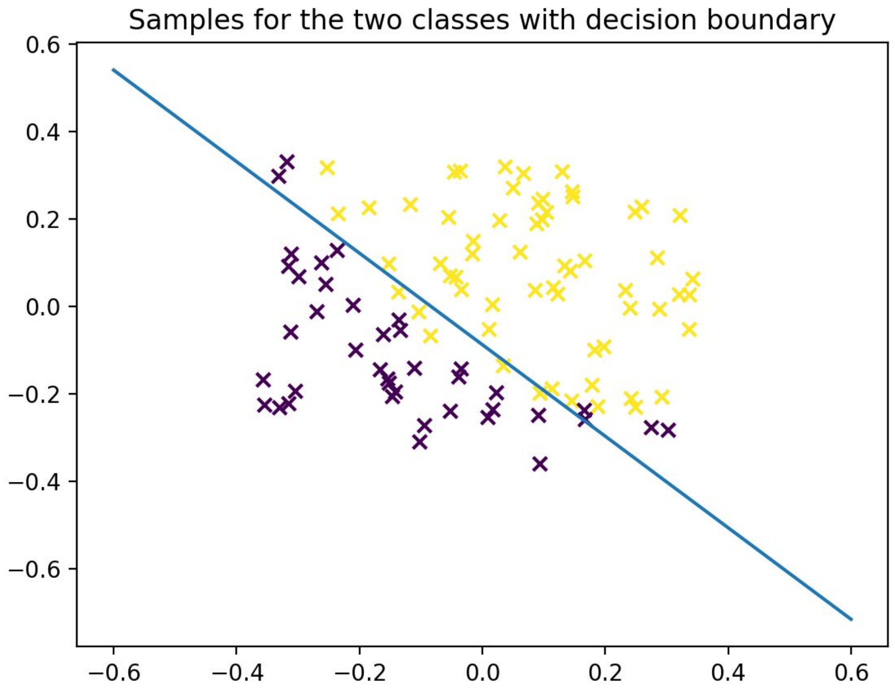


```
The predicted labels are  [0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1
, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1]
(base) mohita@Mohitas-MacBook-Pro ~ %
```

E.

Following are the data points obtained. The two different colors represent the data points from two different classes. There is also a decision boundary plotted on the data separating the two classes.

Mohita Chaudhary
20830560

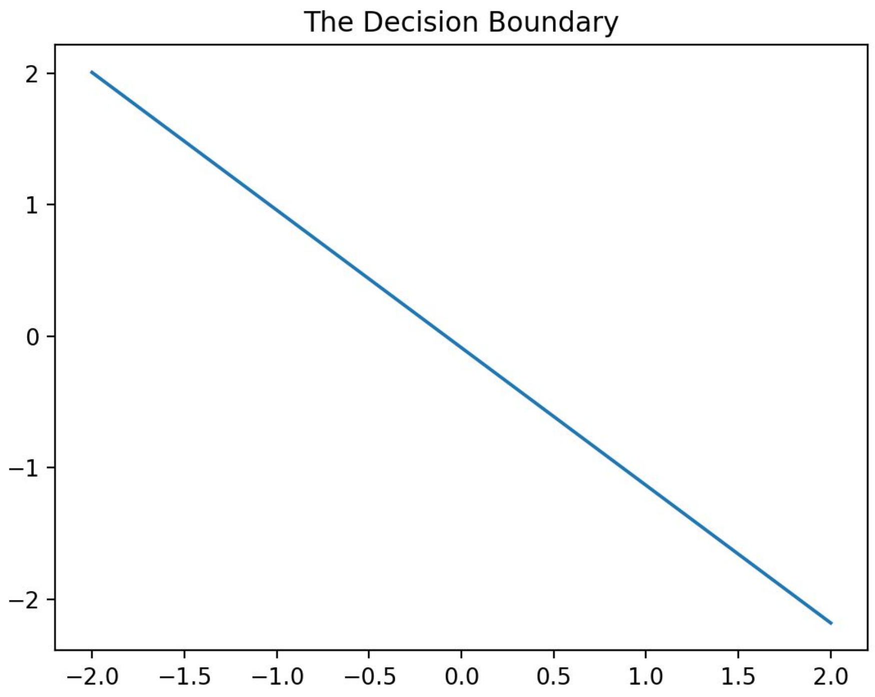


F.

The decision boundary between two classes is obtained by finding the points whose probability to be in both the classes is equally likely. This can only happen when $-w^T x = 0$, where w are the parameters and x is the sample point, this is because for the probability of a point to be equally likely in both the classes the value of $\frac{1}{1+e^{-w^T x}}$ =0.5. This will lead to the result that, bias + $w_1 * x_1$ + $w_2 * x_2$ =0. Finally, we can obtain the points where x_2 is equal to $\frac{-1}{w_2}$ ($w_o + w_1 * x_1)$, for any value of x_1 .

Following, is the decision boundary between the two classes.

Mohita Chaudhary
20830560



Solution 5:

A.Difference between Bayes and Naive Bayes Classifier.

Ans: We can call the Naive Bayes Classifier the approximation for the Bayes Classifier.

The Bayes Approach is computationally expensive, since it assumes that each variable is dependent on the other variables. It is expensive to compute $P(x/c_k)$ since it has many attributes. The number of parameters required for modelling are $2(2^n - 1)$, if there are two labels and each feature stores binary value. It is necessary to have a large dataset which is a good representative of all possible values for the Bayes Classifier.

On the other hand the Naive Bayes Classifier follows the Conditional Independence Assumption, that is all the features are considered independent given the class label c_k.

$P(x_1, x_2, x_3, \ldots\ldots, x_n | c_k) = P(x_1 | c_k) X P(x_2 | c_k) X P(x_3 | c_k), \ldots\ldots\ldots P(x_n | c_k)$ (Cond. Independence)

The number of parameters required for the Naive Bayes approach are 2^n, which are much less than what is required for the Bayes Classifier. Also, the Naive Bayes Classifier has a low tendency to overfit and therefore, it achieves better results even for a small size of training dataset ulike the Bayes Classifier.

B.Situation when Naive Bayes is equivalent to Bayes ?

Ans:

The Naive Bayes classifier would be equivalent to the Bayes classifier when the features which are being considered by the Bayes Classifier are in actual conditionally independent that is they satisfy the Naive Bayes Classification assumption of conditional independence.
The Bayes Classifier is the most optimal option for classification, it should ideally give the best result, but due to its computational expensiveness it is not that used. So if we consider the Bayes Classifier as the best classifier for any classification problem then the Naive Bayes would be equivalent to the Bayes Classifier only if no other classifier can give a better accuracy than the Naive Bayes Classifier.
Also, If the Bayes Classifier just like the Naive Bayes classifier assumes the conditional Independence for it's features then it would give the same result as the Naive Bayes Classifier.

C.In practice, Bayes classifier is not tractable in many applications. Explain, when Bayesian classifiers can be practically used?
Ans: In practical situations it is a challenging task to use the Bayes Theorem for classification. This is because, the data should be sufficiently large(i.e. A representative of a broader problem) to estimate the priors for a class. The probability $P(\, x|\, c_k)$ is not feasible to be calculated unless the data is extraordinarily large enough to find the probability distribution for all possible values. This is very difficult to achieve, therefore, Bayes Classifier remains intractable.
The Bayes Classifier is computationally very expensive and when the number of features increase it becomes intractable. The Bayesian Classifiers can practically be used only when the data is sufficiently large, i.e. a good representative of all the possible input samples.
For the above reasons, In practical scenarios, the Bayes Classifier can be used as a benchmark to evaluate the performance of the other classifiers.

D.Discuss why Bayes classifier is not tractable while Naive Bayes is tractable.
Ans: The Bayes Classifier is computationally expensive since, it assumes that each variable is dependent on the other variables and hence computing $P(x/c_k)$ with a greater amount of attributes make it intractable. On the other hand, the Naive Bayes because of its Conditional Independence Assumption makes it computationally much easier and hence it is tractable unlike the Bayes Classifier. Also, the Naive Bayes gives nice results even for a small amount of training data unlike the Bayes Classifier which requires a large set of training data which is a fair representative of all possible input samples.