Faculty of Engineering

Department of Systems Design Engineering

UNIVERSITY OF WATERLOO

SYDE 675: Pattern Recognition

Assignment 1

Prepared by

Rahij Imran Gillani

rgillani@uwaterloo.ca

University of Waterloo

200 University Avenue West

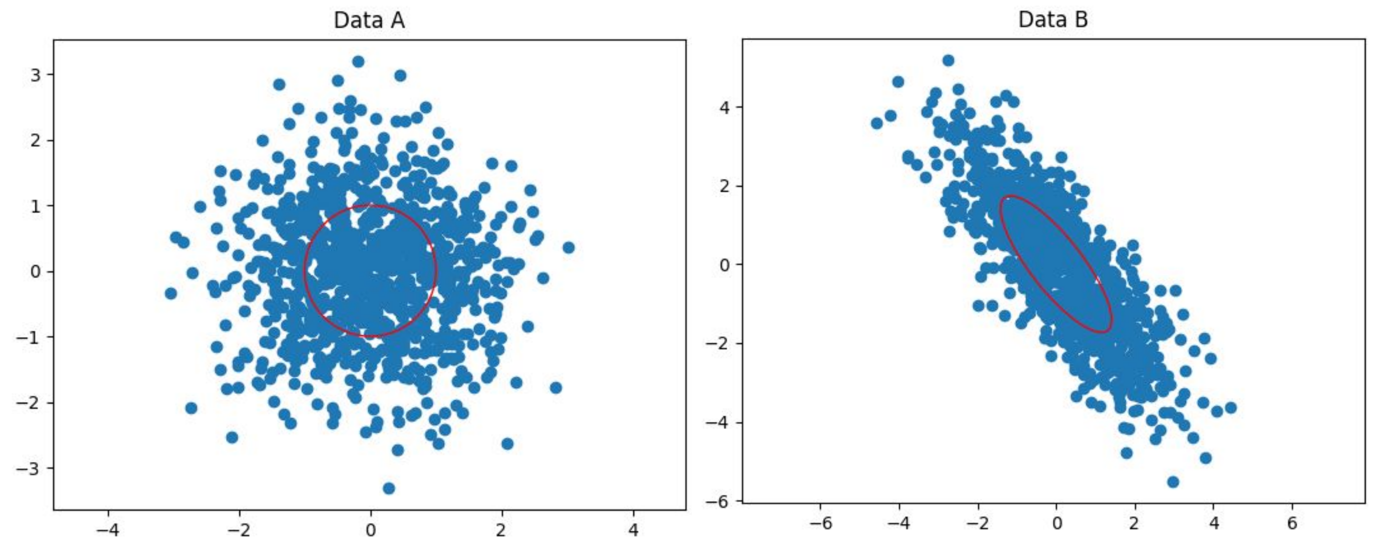
Waterloo, Ontario, Canada

Table Of Contents

Question 1

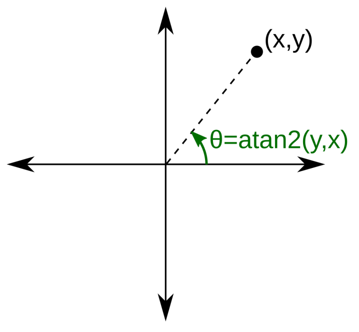
Part (A & B)

First we generated data according to the given covariance matrix using the built-in function `np.random.multivariate_normal`.



After that we needed to plot the contours onto the data which was done using the ellipse function in python. The function requires the mean and the major and minor axis lengths as parameters. The mean for both data A & B is zero. The major and minor axis length is basically the twice the standard deviation (SD) of the data along the two principal axis. The SD of the principal axis is the square root of the eigenvalues of the eigen-vectors of the principal axis'.
For Data A as the covariance matrix is the identity matrix, the eigenvalues are the ones along the diagonal so ve simply take their square root. For Data B we first calculate the Eigenvalues and eigenvectors using the built-in python function: `np.linalg.eig(covariance_matrix),` and then take the square root of the eigenvalues.

To rotate the Ellipse we calculate the angle of rotation by looking at the eigenvector of Data B, and use the following equation:



```
angle = (np.arctan2(y, x)* 180 / np.pi)
```

Part (C & D) Difference between Covariance Matrix

Given Covariance

$$A.\ \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad B.\ \Sigma = \begin{bmatrix} 2 & -2 \\ -2 & 3 \end{bmatrix}$$

Now we calculate the **sample** covariance of the data according to the following formula:

Population Covariance Formula

$$Cov(x,y) = \frac{\Sigma(x_i-\bar{x})(y_i-\bar{y})}{N}$$

Sample Covariance

$$Cov(x,y) = \frac{\Sigma(x_i-\bar{x})(y_i-y)}{N-1}$$

We get the following covariance:

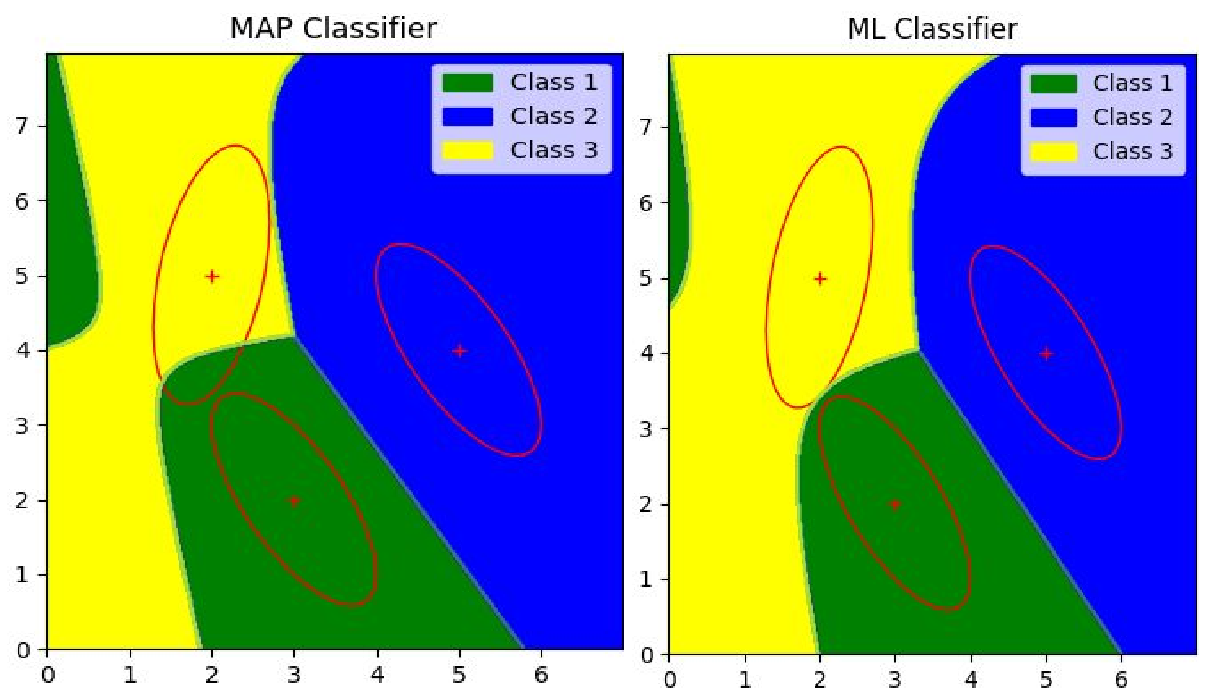
```
          Data A Covariance:
 [[ 1.01953861 -0.01421691]
  [-0.01421691  1.00326314]]

          Data B Covariance:
 [[ 1.8988389  -1.91388559]
  [-1.91388559  2.98506265]]
```

The difference in the Covariances is as a result of the difference in the Global and Sample Dataset. The covariance will be the same when the sample data is equal to the population data (i.e have an infinite data set.)

Question 2

Part (A) Difference between decision boundaries.



MAP Classifier Equation:

$$g_k(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) - \frac{1}{2}\log(|\Sigma_k|) + \log\big(P(C_k)\big)$$

ML Classifier Equation:

$$g_k(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) - \frac{1}{2}\log(|\Sigma_k|)$$

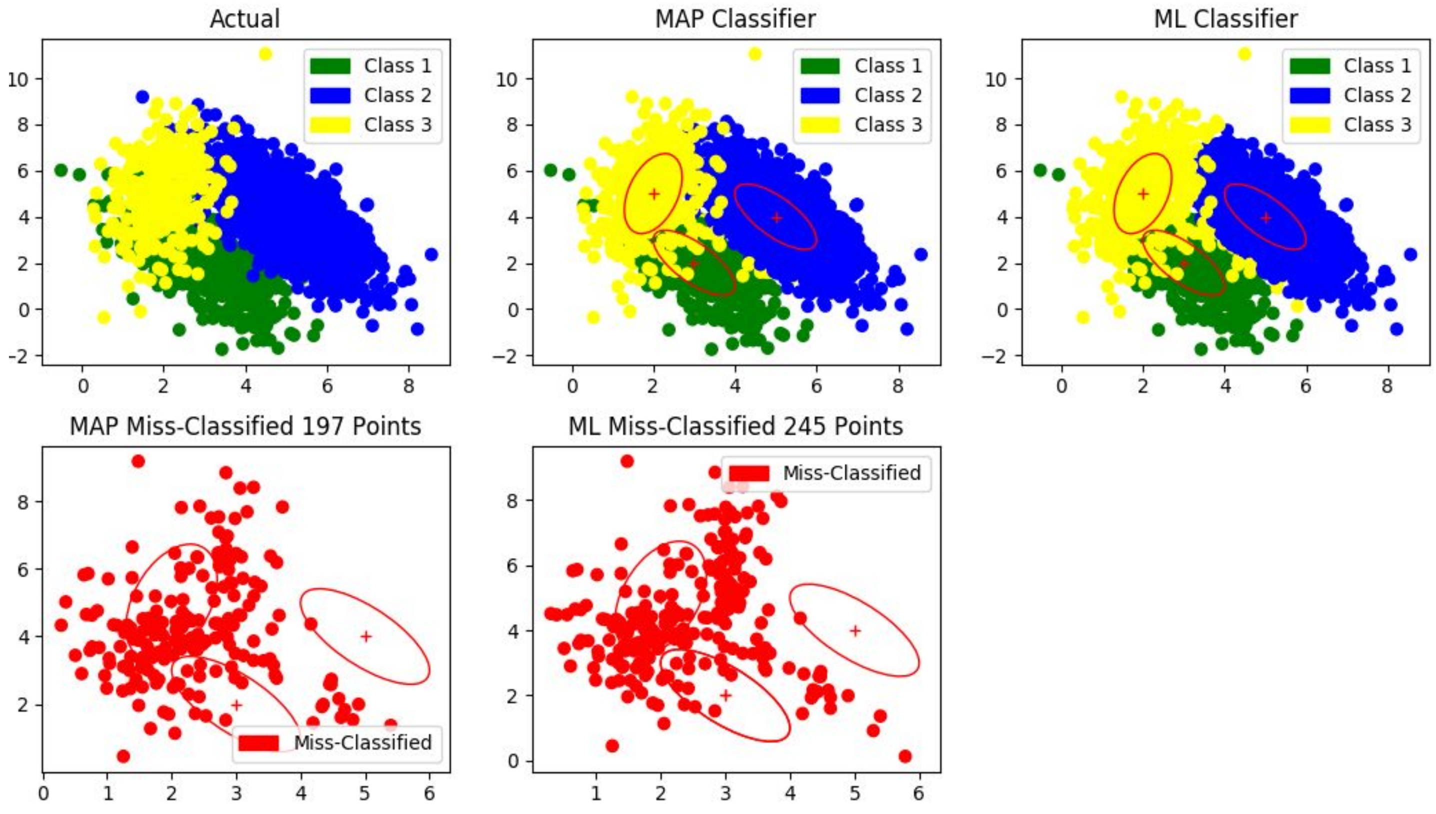
The ML classifier makes a better decision boundary if we consider the first standard deviation contour of class 3, as in that case the decision boundary of class 1 in the MAP classifier, cuts into the contour of class 3. Moreover, there is a greater region misclassified as class 1 in the top left corner by the MAP classifier than by the ML classifier.

Part (B) Confusion Matrix

```
MAP Confusion Matrix
Experimental P(e):
 5.8666666666666645 %
[[ 543   10   47]
 [   9 2063   28]
 [  52   30  218]]
```

```
Ml Confusion Matrix
Experimental P(e):
 7.8999999999999996 %
[[ 510    8   82]
 [  22 1978  100]
 [  23    2  275]]
```

The Experimental error for ML is greater than MAP as ML doesn't take into account the Prior probabilities.



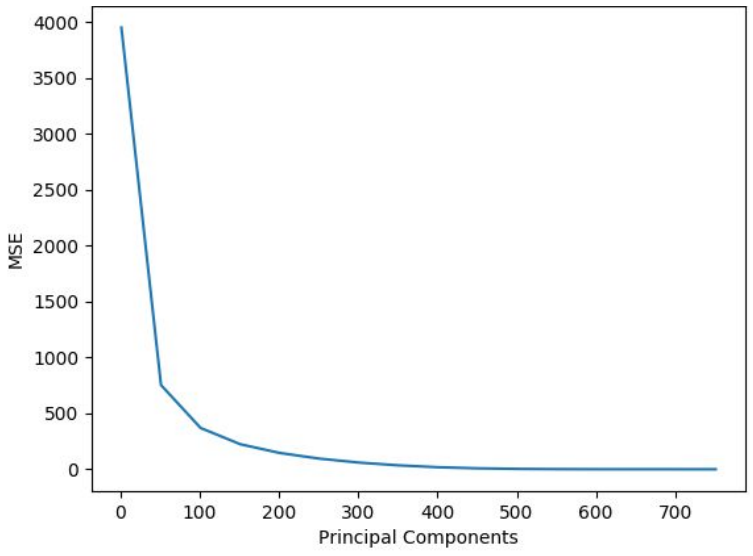
Question 3

Part (A) We 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.

- The first step is to calculate the mean of the data and subtract it from all the data points to center the data.
- Then we generate the covariance matrix and the Eigenvalue & Eigenvectors using the function `np.linalg.eig(covariance matrix).`
- After that we sort the eigenvalues in descending order of indexes (`np.argsort(eig_vals)[::-1]`), and select the first ‘d’ Principal Components.
- Then finally we take the dot product of our centered data with the selected d principal components to transform the data to a lower dimension.

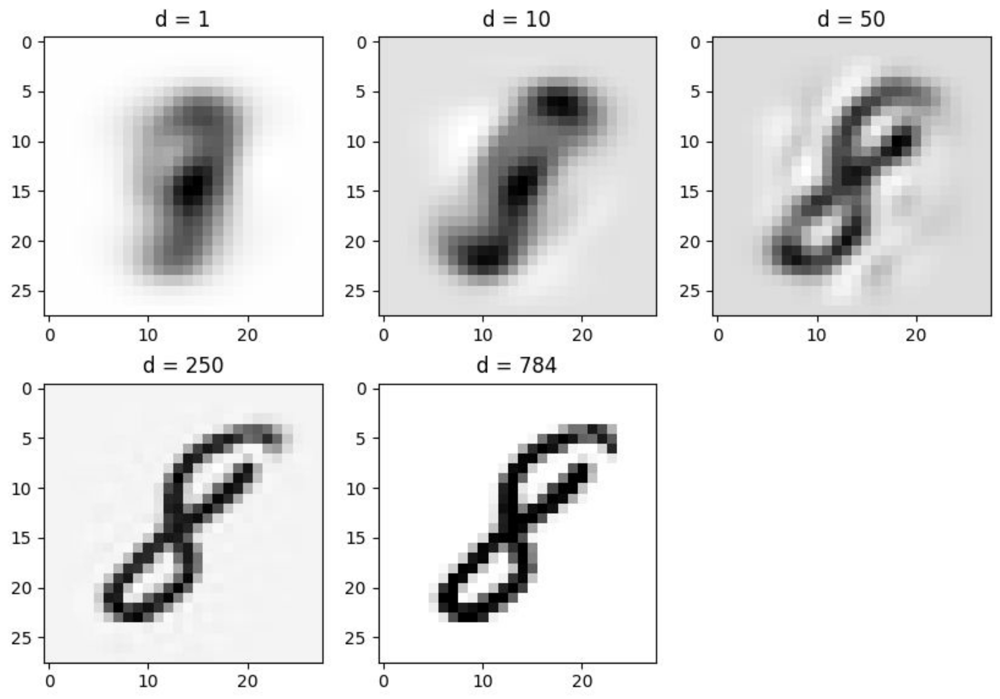
Part (B &C) POV & Reconstruction

We try to find a ‘d’, which is the number of principal components to choose, which gives us a Proportion of Variance >= 0.95. The result comes out to be d=154.
We now reconstruct the image using the compressed data. So we take the dot product of the compressed image with the principal components and then add the mean of the original data to it. We then calculate the Mean Squared Error (MSE) by comparing the original data with the reconstructed data using different number of Principal components and plot the graph below:



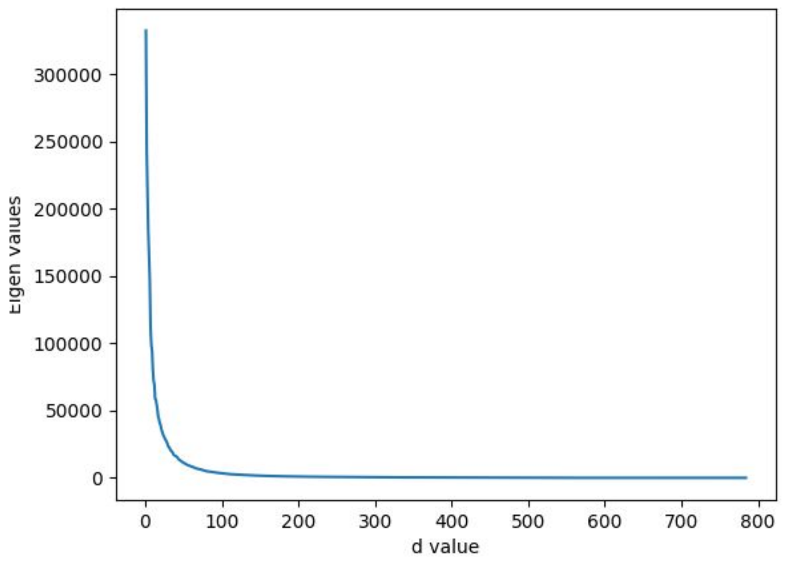
We can clearly see that there is a huge decrease in the MSE in the beginning as just a few of the eigenvalues contribute to most of the variance in the data, and after around 154 principal components the MSE is very close to 0 which was the ‘d’ that we came up with when we were calculating the POV to be equal to 95%.

Part (D) Reconstruct 8



We reconstructed 8 after compressing it using PCA with different values of ‘d’ (the number of principal components chosen); and we can clearly see that there is not much difference in the image that is reconstructed when ‘d’ was set to 250 and when it was set to 784. This is because as we say earlier that 95% of the variance in the data is captured by 154 principal components, so there will not be a drastic difference in the detail of the image reconstructed for a ‘d’ higher than 154.

Part (E) Eigenvalues VS ‘d’ value



We can see from the plot that the magnitude of the eigenvalues decreases exponentially till around d=50. Hence, just a few eigenvectors of the 784 store most of the information about the data.

Question 4

Part (A) Logistic Regression Cost Function

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

Python Implementation:

```
m = x.shape[0]
c = (-1/m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
```

Part (B) Estimate Parameters

```
Parameters
 [[1.72155637]
 [4.00972187]
 [3.74452106]]
```

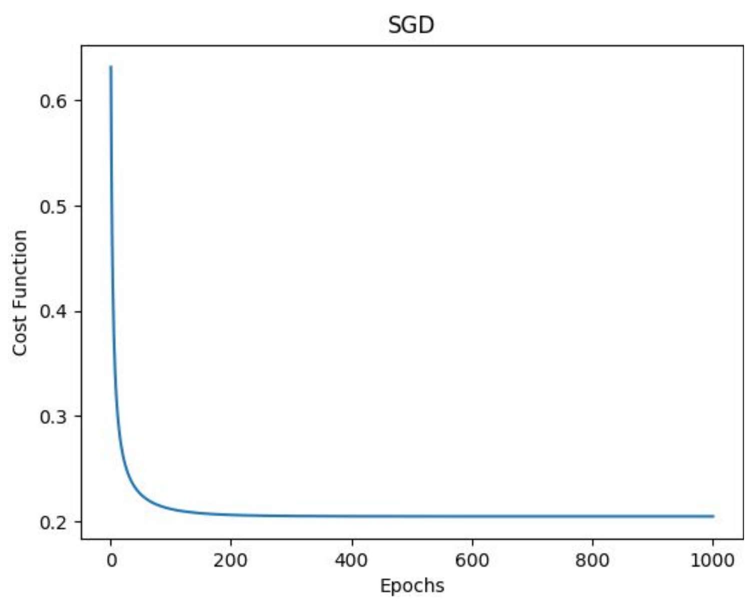
These are the three parameters, θ_0 , θ_1 and θ_2 that were calculated using Stochastic Gradient Descent for Logistic regression. The equation used for the gradient is the following.

$$\frac{\partial}{\partial\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

Python Implementation:

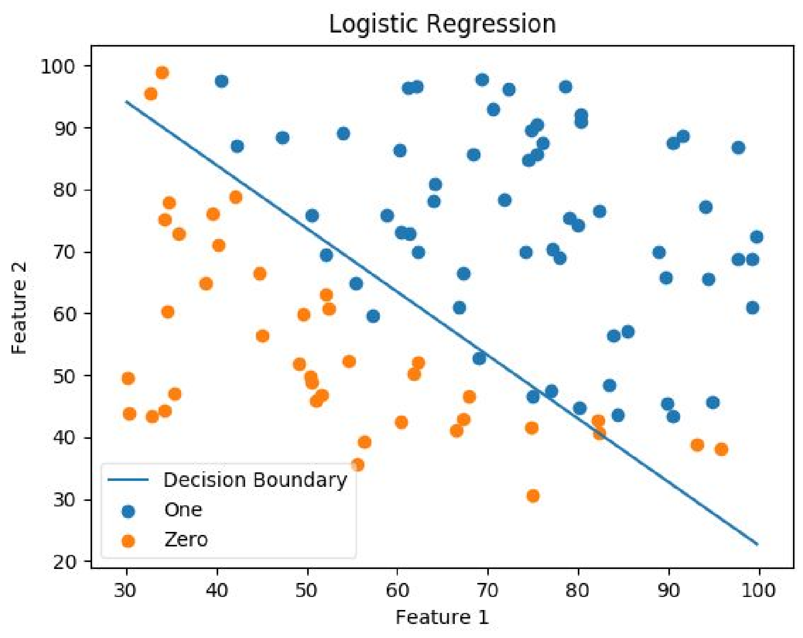
```
 m = x.shape[0]
 g = (1/m) * np.dot((h - y), x)
```

Part (C) Cost Function vs Epochs



As we can observe from the graph that it takes approximately less than 100 epochs for convergence.

Part(D-F) Classification & Accuracy



Accuracy = **89.0%**

The accuracy of the model is 89% as shown.

Question 5

Naïve Bayes Discussions:

Part (A) **What is the difference between Bayes and Naïve Bayes classifiers?**

ANS: In Naive Bayes we assume conditional independence, which means all the features are independent in a given class label C_k.

Part (B) **In which situation, Naïve Bayes is equivalent to Bayes?**

ANS: Naive Bayes is equivalent to Bayes when there is no correlation between the features or you can say when the data has a diagonal covariance matrix.

Part (C) **In practice, Bayes classifier is not tractable in many applications. Explain, when Bayesian classifiers can be practically used?**

ANS: It can be practically used when the number of features are less.

Part(D) **Discuss why Bayes classifier is not tractable while Naïve Bayes is tractable.**

ANS: Bayes classifier is not tractable because it is computationally very expensive with a time complexity of $2(2^n-1)$. Whereas, Naive Bayes has a time complexity of 2n hence it is computationally feasible.