UNIVERSITY OF WATERLOO

Department of Electrical & Computer Engineering

# Pattern Recognition
# SYDE-675
# Assignment – 1

**Prepared By:**

**Name:** Kunal Taneja

**Student ID:** 20790967

**Winter 2019**

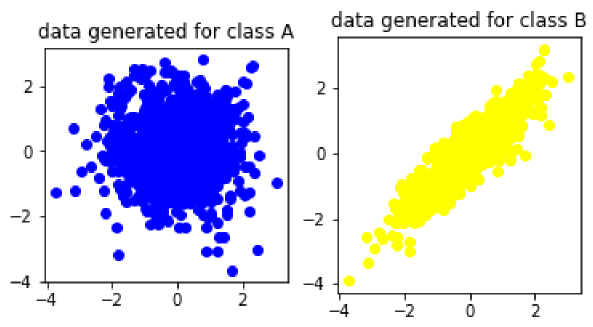
## Problem 1.

$$a.\ \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad b.\ \Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

Given co-variance matrix of 2 classes. For the first problem we needed to generate the data samples corresponding to each co-variance matrix.

**1(a).** We generate the required data points using concept of Cholesky decomposition. It is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose, which is useful for efficient numerical solutions. We perform Cholesky decomposition on provided covariance matrices of class A and class B. After calculating the value of L (for both classes) after decomposition, we perform a dot product of L with randomly uniformed generated data using **numpy.random.standard_normal()** function. At last we scatter plot the last produced data after all mathematical calculations.

```
In [74]: runfile('C:/Users/Kunal Taneja/Q1.py', wdir='C:/Users/Kunal Taneja')
```



**1(b).** For calculating the first standard deviation contour as a function of the mean, eigenvalues, and eigenvector we follow the mathematics behind Ellipse models. the equation of an axis-aligned ellipse with a major axis of length $2a$ and a minor axis of length $2b$, centered at the origin, is defined by the following equation:

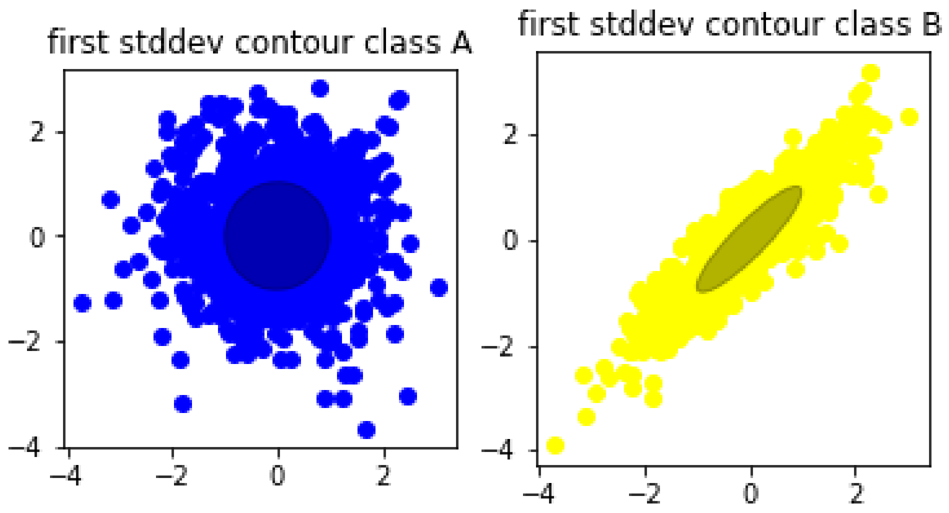
$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

But in case of standard deviation contours, the length of the axes are defined by the standard deviations $\sigma_x$ and $\sigma_y$ of the data such that the equation of the error ellipse becomes:

$$\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2 = s$$

where s defines the scale of the ellipse and could be any arbitrary number (e.g. s=1).

So first we calculate the eigenvalue and eigenvectors from the given covariance matrices. Now we can use Ellipse function present in **matplotlib.patches** library of python. As the mean is assumed to be zero, so center of ellipse is defined as (0,0) the width and height of the ellipse that is the principal axis of the ellipse are the square root of the eigenvalues of respective classes. Finally the principal axis are aligned in the direction of eigenvectors so in the angle attribute we pass eigenvector of respective class. Then we plot the first standard deviation contour over the scatter plot we plotted in 1(a).



**1(c).** Covariance indicates how two variables are related. A positive covariance means the variables are positively related, while a negative covariance means the variables are inversely related. The formula for calculating covariance of sample data is shown below.

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

The factor of n-1 is due to the bias we encounter while calculating sample covariance. So, in order to eliminate the bias we divide the formula by n/n-1. Usually the n in numerator gets cancelled with the n in denominator while the n-1 factor stays in the denominator.

So the same formula was applied over the data in our problem to get sample covariance matrix of each class. Diagonal elements of covariance matrix are simply variance of respective classes whereas off diagonal elements correspond to the covariance between the classes.

```
Calculated sample covariance of dataset A is :  [array([ 0.99153544,  0.00888345]), array([ 0.00888345,
1.04625032])]
Expected covariance matrix of class A is: [[1, 0], [0, 1]]
Calculated sample covariance of dataset B is :  [array([ 0.99153544,  0.8962541 ]), array([ 0.8962541 ,
1.00890124])]
Expected covariance matrix of class B is: [[1, 0.9], [0.9, 1]]
```

**1(d).** As we can see from the results that the expected and observed covariances matrices of given classes are quite close to each other. That is the computed covariance matrix is nearly equal to the given covariance matrix. This justifies that our method of generating data samples using Cholesky method was successful and the generated dataset is nearly similar to the dataset whose covariance matrix was provided.

## Problem 2.

| | | |
|---|---|---|
| $P(C_1) = 0.2$ | $P(C_2) = 0.3$ | $P(C_3) = 0.5$ |
| $\mu_1 = [3 \quad 2]^T$ | $\mu_2 = [5 \quad 4]^T$ | $\mu_3 = [2 \quad 5]^T$ |
| $\Sigma_1 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$ | $\Sigma_2 = \begin{bmatrix} 1 & -1 \\ -1 & 7 \end{bmatrix}$ | $\Sigma_3 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 3 \end{bmatrix}$ |

Given 3 classes where each class is described by the following priors, mean vectors, and covariance matrices we need to make decision boundaries corresponding to MAP and ML classification rules.

**2(a)**. General discriminant function for a MAP rule can be given as:

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

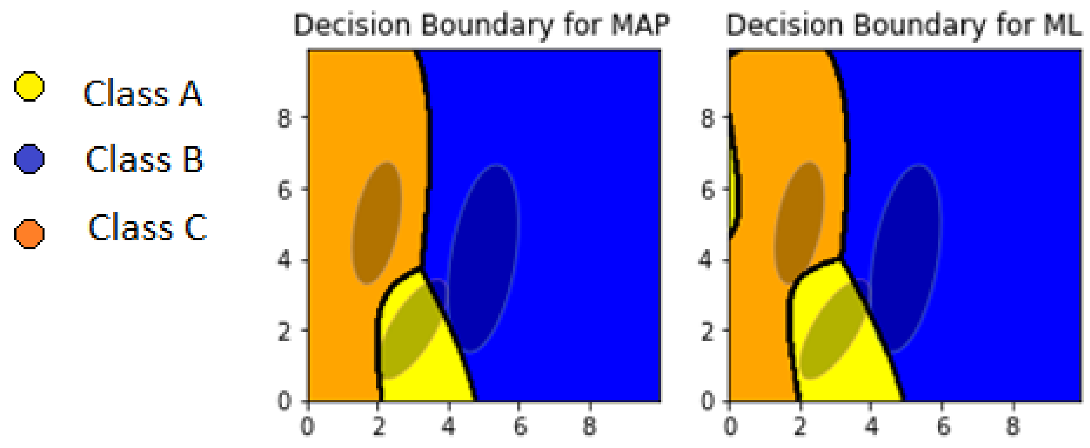
And general discriminant function for a ML rule can be given as:

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

**x** – data sample.
**u$_k$** - mean of class k.
**$\Sigma_k$** - covariance matrix of class k.

Using these classification rules we try to build decision boundary. Our main approach is to use a uniformly distributed sampling grid to draw the decision boundaries. We use a 10x10 grid with a step size of 0.05 in grid mesh. Every point in the grid is fed into the above equation of all 3 classes and whichever equation gives the best value, the point is set to class corresponding to the equation. We use a colormap such that each class has unique colour so, after classifying the point it is coloured with the colour of respective class. Similarly, all points in the grid are classified and coloured resulting into partitioning the grid into 3 different colours with clear and distinct decision boundaries. In addition to this, we plot first standard deviation contours for each class over the grid in a similar way which we did in Question 1.

The difference between the decision boundaries is quite minimal, but there is one key difference between MAP decision boundary and the ML decision boundary. In ML decision boundary we see some part of class A lies in the upper left corner of the grid. This is defined as the error region. As we do not conider class priories in ML decision rule as compared to MAP decsion rule there might be some cases where the points therefore the regions are misclassified. In the next section we see the P(err) for each decision boundary corresponding to each class.



**2(b).** In this section we generate a 3000 sample dataset based on the class priories given in the question. And once the dataset is generated we concatenate the dataset into one vector and pass it to the MAP and ML classifiers (functions defined in the code). The results of the classification are stored and the corresponding confusion matrix is calculated. For confusion matrix we use built in function of **sklearn library.** Also based on the confusion matrix we calculate the P(Err) for both ML and MAP classifiers corresponding to all 3 classes. The result are as follows:

```
Confusion Matrix MAP:
[[ 404   53  143]
 [  49  815   36]
 [  48   42 1410]]
Confusion Matrix of ML Classifier:
[[ 462   41   97]
 [  58  817   25]
 [ 109   54 1337]]
P(Err) MAP Classifier For:
 Class A- 0.06533333333333333,
 Class B- 0.028333333333333332,
 Class C- 0.03
P(Err) ML Classifier For:
 Class A- 0.046,
 Class B- 0.027666666666666666,
 Class C- 0.05433333333333333
```

As we can see a lot of Class A points are misclassified as Class B and Class C. Therefore, P(Err) for class A is higher than P(Err) of class B and class C in MAP decision rule.

Class B has the minimal P(Err) so most of the datapoints are correctly classified in both MAP and ML classifiers.

ML classifier tends to overfit the model as compared to MAP classifier. Therefore, the error region is the result of overfitting and misclassifications of a tend to lie in region of class C which can be observed from the values of P(err) in ML rule.

## Problem 3.

The Problem revolves around the MNIST dataset containing 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.

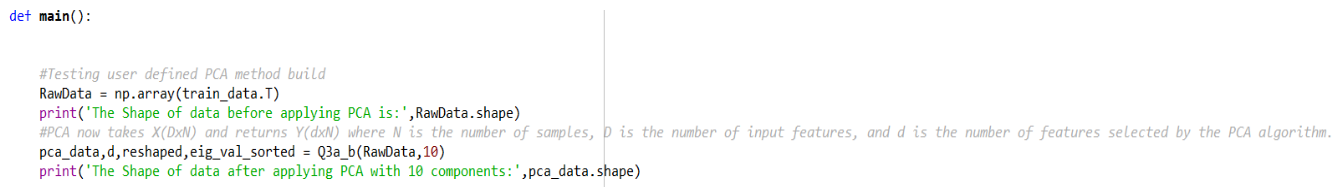
**3(a).** The problem require us to design PCA algorithm from scratch. So what basically PCA does is dimensionality reduction by reducing the high dimensional data into low dimensional data. PCA takes X(DxN) and returns Y(dxN) where N is the number of samples, D is the number of input features, and d is the number of features selected by the PCA algorithm.

Designing PCA algorithm is easy and efficient. First, we center the dataset around the mean value by subtracting the mean of the corresponding features from the data itself. Then we find the covariance matrix of the centered data. This covariance matrix is then fed into **numpy.linalg.eig()** method of numpy library which returns eigenvalues and eigenvectors from the covariance matrix. Further we sort eigenvalues in the decreasing order in order to get principal components that is components with highest eigenvalues. As higher is the eigenvalue more of the variance is retained in the lower dimension when transformed from higher dimension.

Now an appropriate value of d (lower dimension) is chosen to transform X(DxN) where D is the higher dimension into Y(dxN). Further topmost d eigenvalues are chosen out of all the eigenvalues and are stored into a vector.

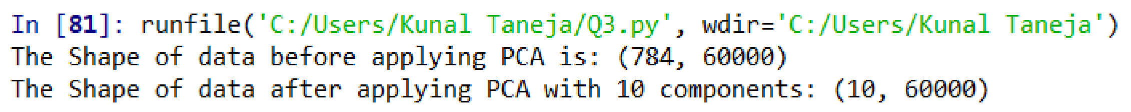
Following this dot product of these topmost d eigenvalues are data centred around mean is taken which results into pca_data. Now pca__data is the required transformed data from the original data.

We tested our code after choosing value of d as 10 and following were the results obtained.



```
def main():


    #Testing user defined PCA method build
    RawData = np.array(train_data.T)
    print('The Shape of data before applying PCA is:',RawData.shape)
    #PCA now 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.
    pca_data,d,reshaped,eig_val_sorted = Q3a_b(RawData,10)
    print('The Shape of data after applying PCA with 10 components:',pca_data.shape)
```



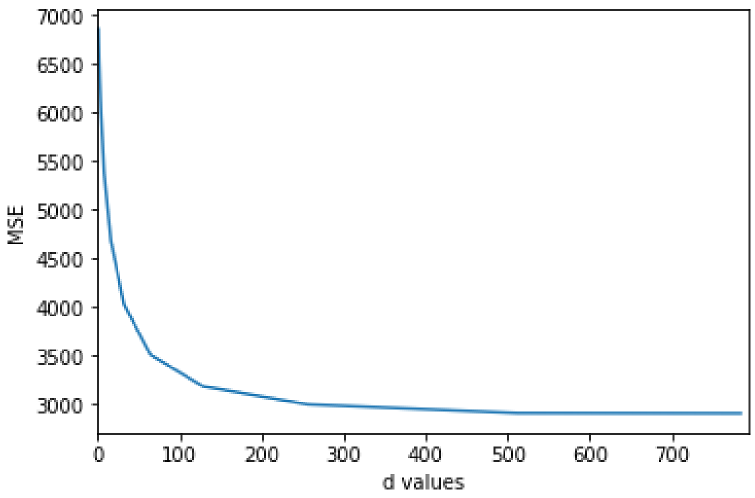
```
In [81]: runfile('C:/Users/Kunal Taneja/Q3.py', wdir='C:/Users/Kunal Taneja')
The Shape of data before applying PCA is: (784, 60000)
The Shape of data after applying PCA with 10 components: (10, 60000)
```

**3(b).** Suitable value of d for POV = 95%  came out to be 154 when tested over running the PCA function.

**3(c).** In order to reconstruct the data, we reverse the transformation applied the PCA. We take the dot product of pca_data (reduced data) with the vector of eigenvalues in order to get reconstructed data.

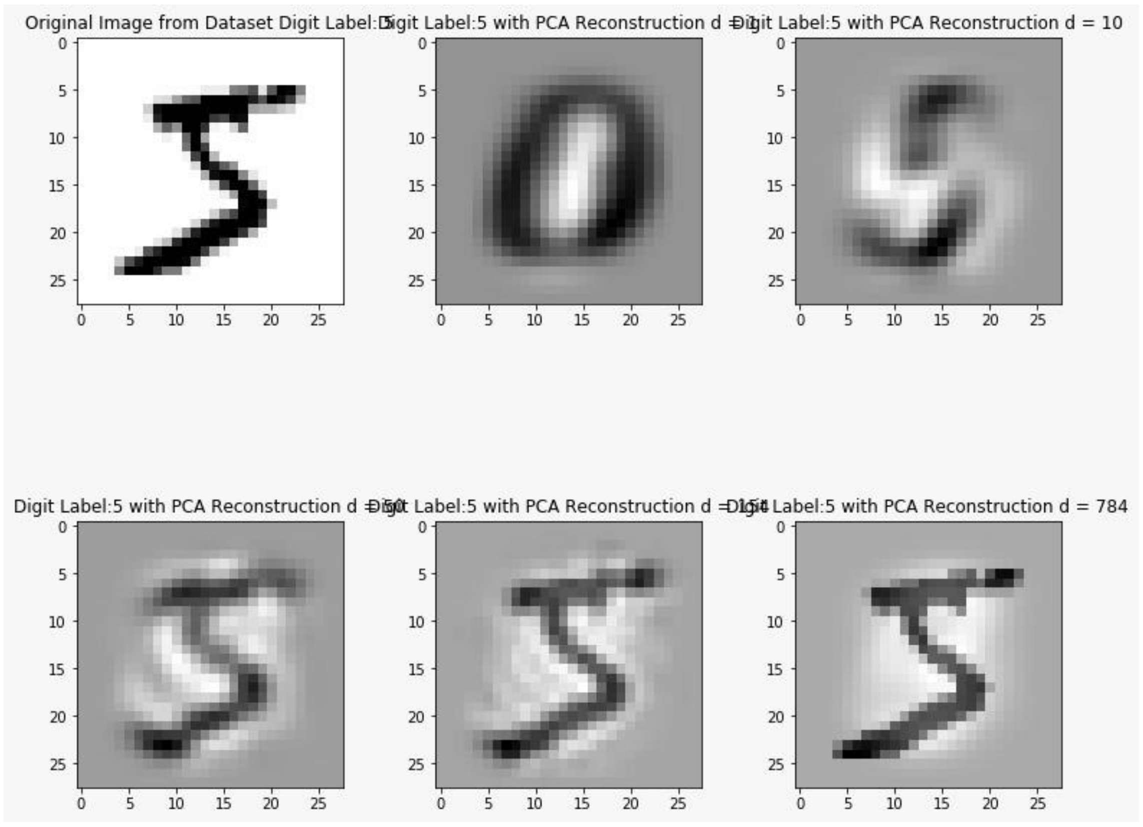
We then find mean square error between the original data and the reconstructed data using **mean_squared_error** function from **scikit.metrics library** for different values d and hence for different sets of PCA reduced data and the original data.

The plot we get is:



As we can see from the plot that if we try to reconstruct the original data using lower value of d hence lower number of eigenvalues we get higher values of MSE. As we try increasing the number of dimensions the value of MSE drops exponentially. So if we try reconstructing the original data with larger value of d we get better results.

**3(d).** We take data sample as png image of '5' and try reconstructing it after applying PCA using different values of d. Following are the results we get for different values of d:

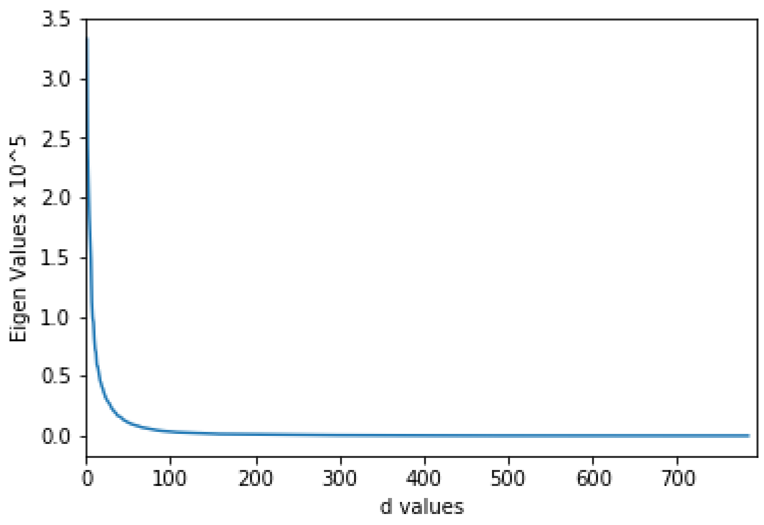

Original Image from Dataset Digit Label:5
Digit Label:5 with PCA Reconstruction d = 1
Digit Label:5 with PCA Reconstruction d = 10
Digit Label:5 with PCA Reconstruction d = 50
Digit Label:5 with PCA Reconstruction d = 154
Digit Label:5 with PCA Reconstruction d = 784

From the above figures it is clear that most of the information is lost if we try to reconstruct it using just 1 feature. For d = 10 also we do not get good results as the reproduced image is quite different from the original image.

For d = 50 we get a quite blur and smooth image of the original image and hence only some of the information or the variance of the original image is captured. However, increasing value of d to 250 we get much finer and better results as most of the information is retained and reconstructed image matches with the original image.

For d = 784 that is considering all the features we get nearly the reconstructed image same as the original one. Majorly all of the information is retained and one cannot differentiate between two images.

**3(e).** Plot of eigenvalues versus d values is given as:

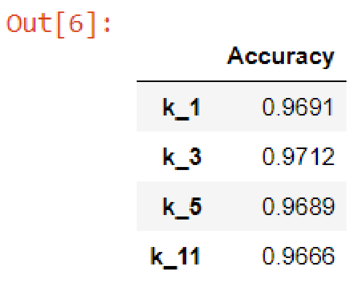


From the plot we can visualise that only few d values are responsible for pertaining the overall variance of the dataset. Not all features are needed to represent the data. This was also seen when d = 154 was found for POV = 95%. Hence few of the d values have quite high eigenvalues as these are the principal components of the data which are responsible for pertaining the variance. Apart from these most of the d values correspond to nearly zero eigen values as these features have quite less information regarding the spread and variance of the data. Hence we can identify proper value of d uptill which most of the information is retained and then reduce the data.

## Problem 4.

**4(a).** This problem requires us to solve the KNearestNeighbours Classifier  from the scratch. The working of algorithm is based on the fundamental logic behind KNN. We need to first evaluate the distance metric between all the training and testing data. For this we use Euclidean method in the form of function **distance.cdist(test_data, train_data, 'euclidean')** present in **from scipy.spatial library.** After calculating the distance metric k nearest neighbours are selected and the training data is fitted using the fit_method in the code. After fitting the data, the predictions made are compared with the original labels of the test data and the accuracy is calculated for different values of k.

The accuracies found for k = {1,3,5,11} are:

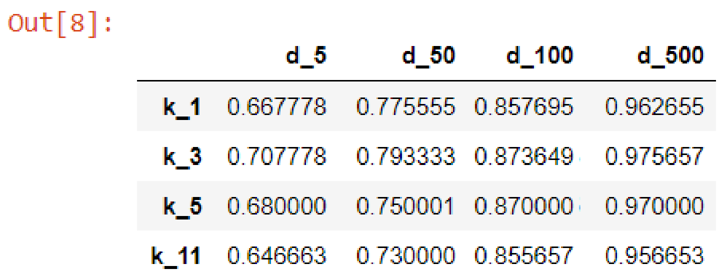


Out[6]:

| | Accuracy |
|---|---|
| k_1 | 0.9691 |
| k_3 | 0.9712 |
| k_5 | 0.9689 |
| k_11 | 0.9666 |

**4(b).** Now rather than giving the training data with all feature values, PCA is applied for different values of d using the function defined in problem 3. And then the accuracies of different d values are compared with the different k values.

The accuracies found were:



Out[8]:

| | d_5 | d_50 | d_100 | d_500 |
|---|---|---|---|---|
| k_1 | 0.667778 | 0.775555 | 0.857695 | 0.962655 |
| k_3 | 0.707778 | 0.793333 | 0.873649 | 0.975657 |
| k_5 | 0.680000 | 0.750001 | 0.870000 | 0.970000 |
| k_11 | 0.646663 | 0.730000 | 0.855657 | 0.956653 |

**4(c).** As we can see from the above results that smaller values of d no matter how much is the value of k gives poor accuracy. Because most of the information is lost when data is reduced to 5 dimensions. As the value of d is increased the accuracy increases which is correct as more of the information is retained and better predictions are made based on fitting on training data.

Whereas in part (a) for k = 3 we had the maximum accuracy but that was when all features are included. However in part (b) accuracy depends on the number of features selected.