Assignment 3

Zahidur Talukder
Student ID- 1001771748

USED LATE DAY QUOTA

Problem 1

The main requirement of this problem is to write a function which can find the Kernel Density Estimation on a data and find the estimated density both in 1D and 2D.
For finding the Kernel density, I have used Gaussian distribution of data. Since the formula for Gaussian function for 1D and 2D is different. So, I have written a Class named 'mykde' which can estimate kernel density estimation both in 1D and 2D.

```python
## Code begins here
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

class mykde:

    def __init__(self):
        pass

    def gaus1d(self,x,x_i, bandwidth):
        x_bar  = x_i
        pdf_1d = (np.sqrt(2*np.pi*bandwidth**2)**-1) * np.exp(-((x - x_bar)**2)/(2*bandwidth**2))
        return(pdf_1d)

    def gaus2d(self,x=0, y=0, mx=0, my=0, sx=1, sy=1):
        pdf_2d = 1. / (2. * np.pi * sx * sy) * np.exp(-((x - mx)**2. / (2. * sx**2.) + (y - my)**2. / (2. * sy**2.)))
        return(pdf_2d)

    def kde_1d(self,x_i,bandwidth):
        mu=x_i
        variance = bandwidth
        sigma= variance
        x = np.linspace(min(mu) - 3*sigma, max(mu) + 3*sigma, 10000)
        kde_final= np.zeros(10000)
        for i in range(len(mu)):
            a=self.gaus1d(x, mu[i], sigma)
            kde_final=kde_final+a
        kde_final /= sp.integrate.simps(kde_final, x)
        plt.plot(x,kde_final)
        plt.show()

    def kde_2d(self,x_i,y_i,bandwidth):
        X = x_i
        Y = y_i
        x = np.linspace(-5, 5)
        y = np.linspace(-5, 5)
        width= bandwidth
        x, y = np.meshgrid(x, y)
        kde_final= np.zeros((50,50))
        for i in range(len(X)):
            a=self.gaus2d(x,y,mx=X[i], my=Y[i],sx=width, sy=width)
            kde_final=kde_final+a
        plt.contourf(x, y, kde_final/len(X), cmap='Blues')
        plt.colorbar()
        plt.show()
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.plot_surface(x, y, kde_final/len(X), rstride=1, cstride=1,
                        cmap='viridis', edgecolor='none')

        ax.set_title('surface')
        ax.view_init(30, 30)
        fig

```

As you can see from the Class mykde, I have defined 2 function namely 'gaus1d' and 'gaus2d' which returns the Gaussian Probability Distribution function (PDF) of 1D and 2D respectively. The formula for generating Gaussian probability are

In 1D:

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

In 2D:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

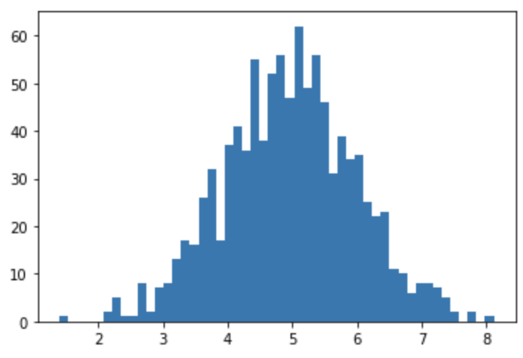
Figure 1: Gaussian PDF for 1D and 2D

I have used exactly the same function to generate my 1D density and 2D density.
I have also defined line-space for 1D based on the data and both X and Y co-ordinate line space for 2D and created a mesh-grid. Thus adding the density of each point and finally normalizing the data eventually gives the kernel density estimation of the data.

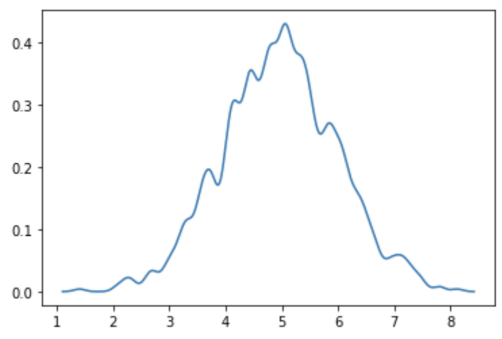
Histogram is an another form of visualizing data but kernel Density gives a smooth view of data in the data space.

Problem 1.1a

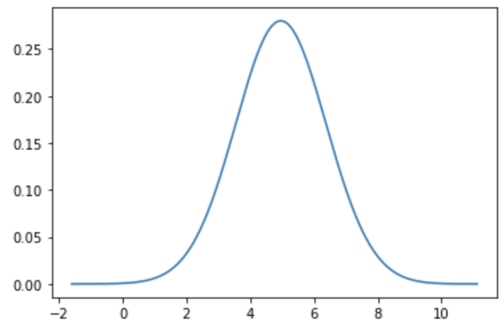
Here I have generated random Gaussian data of 1000 points. Then based on different width of my Kernel density estimation function I have calculated the KDE of the data. The Histogram of the data as well as the KDE based on different bandwidth is given below.



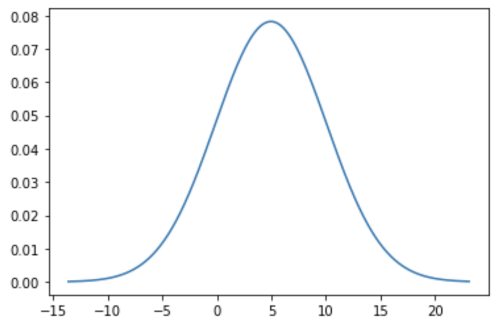
(a) Histogram of the data



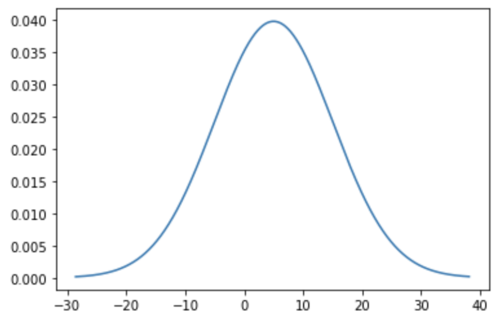
(b) KDE with bandwidth 0.1



(c) KDE with bandwidth 1



(d) KDE with bandwidth 5



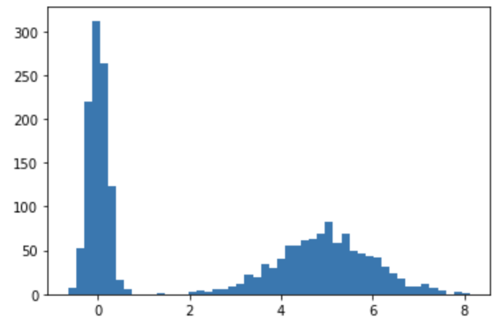
(e) KDE with bandwidth 10

Figure 2: Kernel Density Estimation with different Bandwidth

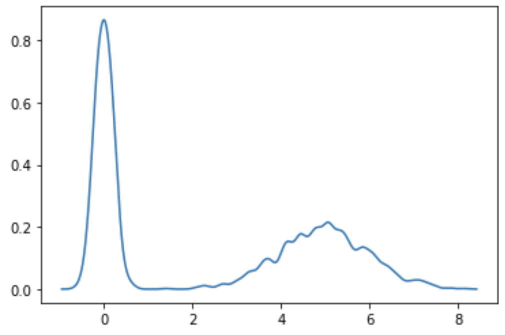
As we can see from the figure when the bandwidth is 0.1, which is too low, the density estimation is not smooth. But when we increase the bandwidth the data become more smoother but density is sparse over larger area.

Problem 1.1b

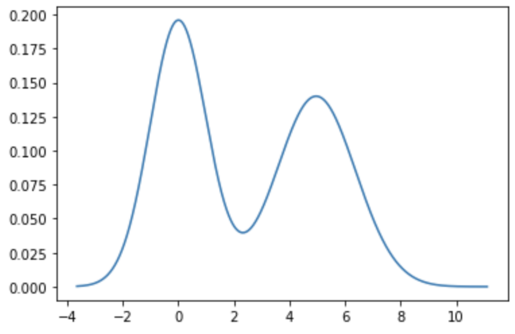
Here I have generated two sets of random Gaussian data of 1000 points each. Then I added the data together. Then based on different width of my Kernel density estimation function I have calculated the KDE of the data. The Histogram of the data as well as the KDE based on different bandwidth is given below.



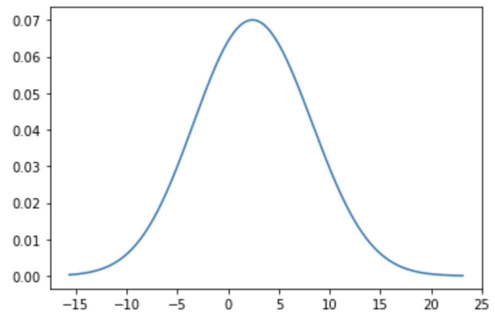
(a) Histogram of the data



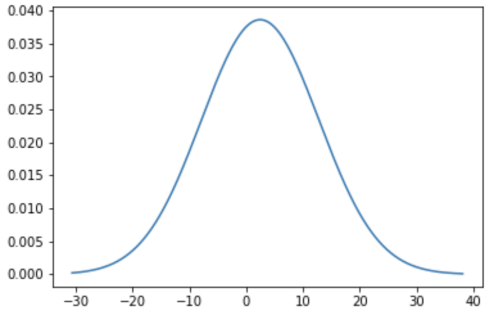
(b) KDE with bandwidth 0.1



(c) KDE with bandwidth 1



(d) KDE with bandwidth 5



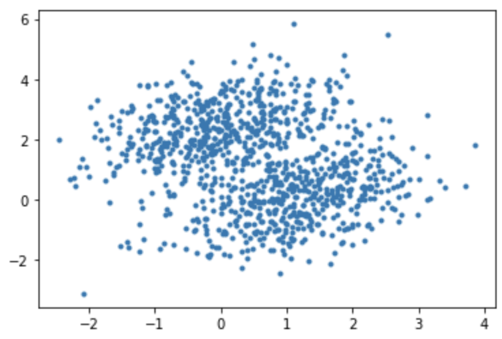
(e) KDE with bandwidth 10

Figure 3: Kernel Density Estimation with different Bandwidth

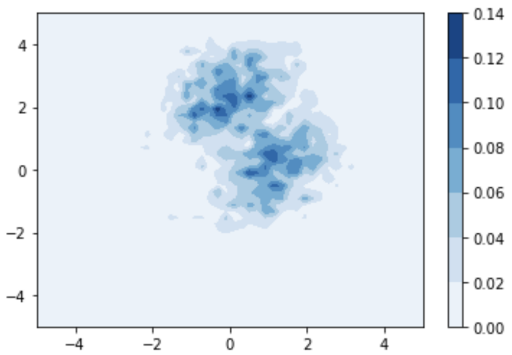
As we can see from the figure, here two clusters of data in the region. The data set have mean of 0 and 5. So when the bandwidth of my kernel Density Function is 0.1, It is able to find the density estimation correctly but not smoothly. As we increase the bandwidth, the density becomes smoother, but when width is above 5, we no longer can find different cluster as if crosses the distance between means of two clusters. Thus For above bandwidth, we only see one cluster of data.

Problem 1.2

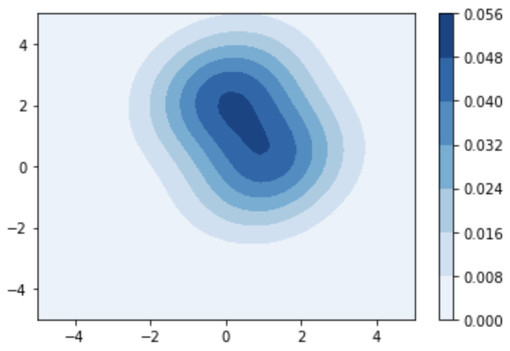
Here I have generated two sets 2D random Gaussian data of 500 points each. Then based on different width of my Kernel density estimation function I have calculated the KDE of the data. The discrete plot of the data as well as the KDE based on different bandwidth is given below. In the 1st page I have plotted the data in 2D and color density represent another dimension. In the following page I have plotted the data in 3D for clear visualization.



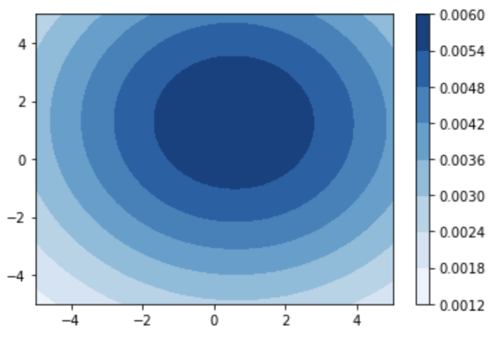
(a) Scatter plot of the data



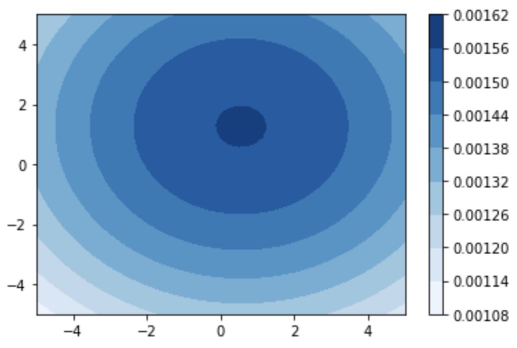
(b) KDE with bandwidth 0.1



(c) KDE with bandwidth 1



(d) KDE with bandwidth 5

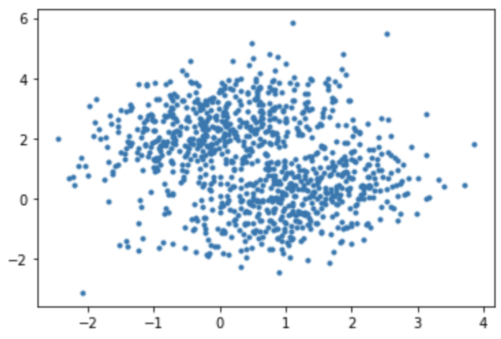


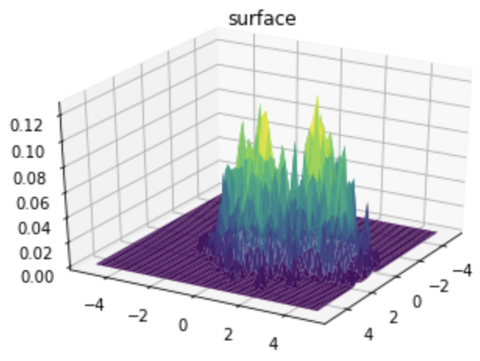
(e) KDE with bandwidth 10

Figure 4: Kernel Density Estimation in 2D with different Bandwidth

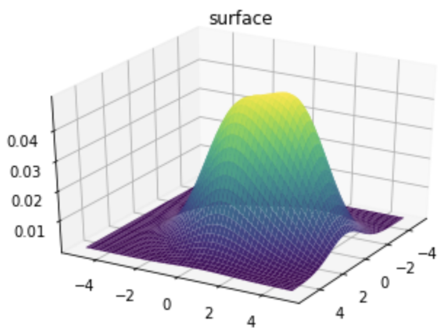
As we can see from the figure when the bandwidth is 0.1, we can distinguish two sets of data. Then by increasing the bandwidth we no longer able to find two cluster rather data seemed to be clustered in a single area.

For clear visualization of the density, I have plotted the 3d plot as well. as We can see , when the bandwidth is too low, The density function has many peaks. As we increase the bandwidth data is merged on a single pick, which is logical as the cluster mean is less than the bandwidth of KDE.

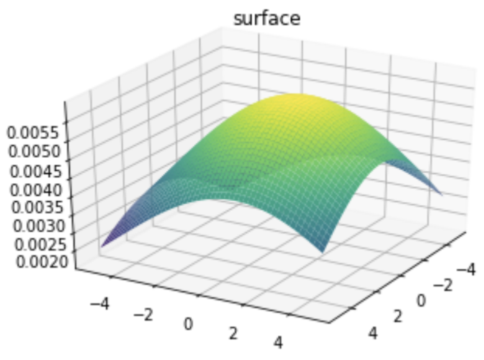

(a) Scatter plot of the data



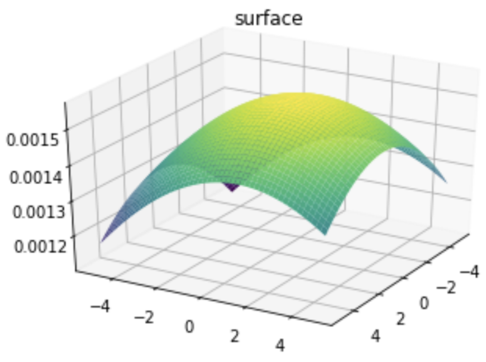
(b) KDE with bandwidth 0.1



(c) KDE with bandwidth 1



(d) KDE with bandwidth 5



(e) KDE with bandwidth 10

Figure 5: Kernel Density Estimation in 3D with different Bandwidth

Problem 2.1

The main challenge in this problem was to find the Principle Component analysis of a given Dataset.
PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensional reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The i-th principal component can be taken as a direction orthogonal to the first i-1 principal components that maximizes the variance of the projected data.

```python
## Code begins here

from tensorflow.keras.datasets import mnist
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np

def myPCA(X , num_components):

    X_meaned = X - np.mean(X , axis = 0)

    # finding Covariance matrix
    cov_mat = np.cov(X_meaned , rowvar = False)

    # Decomposition to eigen value and eigen vectors
    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

    # Sorting eigen value based on highest variance
    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    eigenvector_subset = sorted_eigenvectors[:,0:num_components]

    # PCA
    X_reduced = np.dot(eigenvector_subset.transpose() , X_meaned.
   transpose() ).transpose()

    return X_reduced, eigenvector_subset

```

Problem 2.2

In this problem I have used the PCA to MNIST data set. As we know MNIST dataset has 60000 hand written digits of 0 to 9 for train and 10000 data for test. Each data has 28x28 matrix of total 784 features. So here our main goal was to reduce the feature to a lower value using PCA.

```python
## Code begins here

from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train_reshape = x_train.reshape((len(x_train), np.prod(x_train.shape
    [1:])))
x_test_reshape = x_test.reshape((len(x_test), np.prod(x_test.shape[1:]))
    )
a,b=myPCA(x_train_reshape,2)

```

Here I have reshaped the image to data value having 784 features. Then I call myPCA function for generating 1st 2 principle components. I didn't have any issue with time for doing the PCA. It was pretty fast.

Problem 2.3

The main challenge here was to get the first 5 class of data from MNIST data set. There are 60000 training data and there are 10000 testing data in the dataset for 10 classes. So when I take only 5 classes the number of dataset was about half as we can see in the figure. Again the MNIST dataset is 28*28 pixesls imaage. I convert them to a linear array of 28*28= 784. Again I have done hot encoding to my target data as you can see the shape of the data.

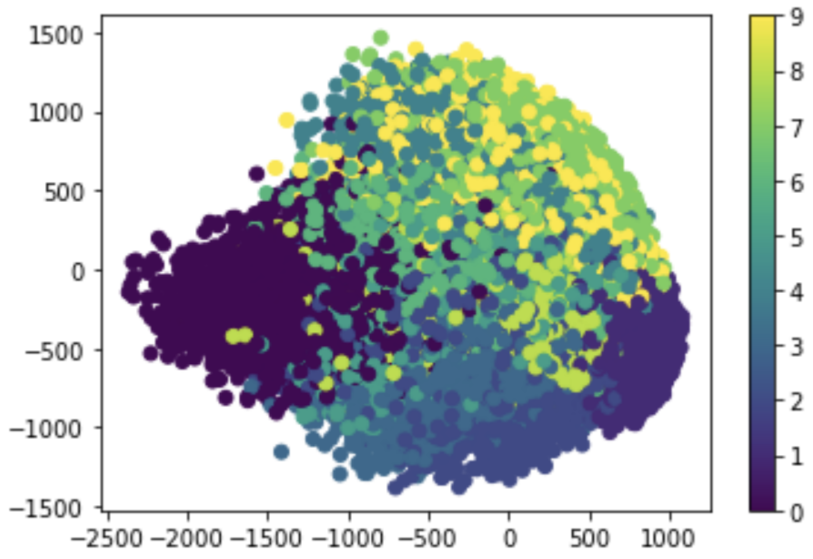


Figure 6: Scatter Plot of 2PC from MNIST Dataset

As we can see from the page, data plot is based on the value of the value of the data i.e. 0 to 9. As we can see from the graph 0 is in the left side where 1 i in the right side, so there is a clear separation of data between 0 and 1. so by 2PC we can separate 0 and 1. But other data points are merged with other points. So it is not possible to separate data using 2PC only for other values. Which is also reasonable because we had 784 features earlier now we have moved to only 2 principle features, it is tough to separate them. But PCA is a strong tool to wipe out unnecessary features.

Problem 2.4

Here 1st 10 PC is reshaped to image to see whether there is any interesting shape.So, I have reshaped the 1st 10 sorted Eigen vector(784x10) as 28x28 images each. The images are displayed below.

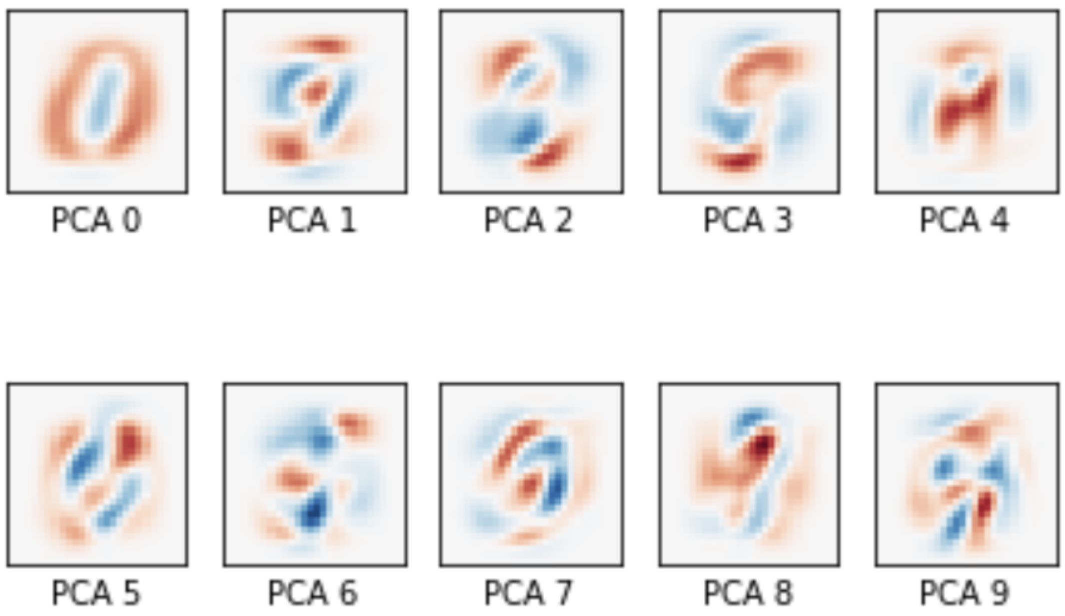
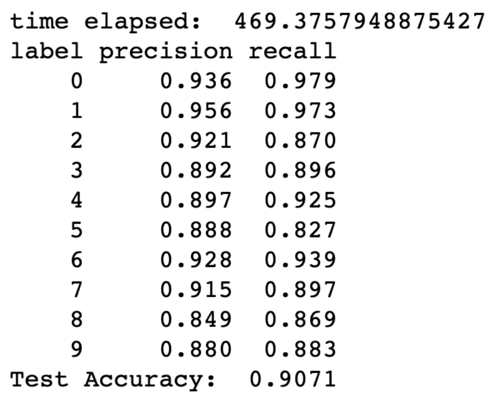


Figure 7: Visualization of 10PC as image

I was expecting some interesting shape in the figure. And while I looked at PCA 0, It feels like there is shape of zero here. But other than this, I could not find any interesting shapes here.

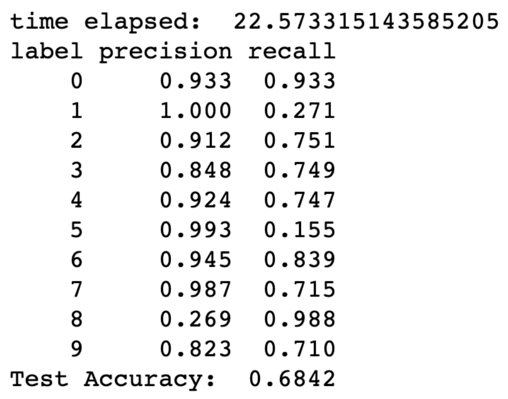
Problem 2.5

For this problem, I did not use any built in function to train my Logistic regression, rather I have used my logistic regression Class from Assignment 2. In this particular Problem, I have trained and tested the model with both normal data and 30 PCA. while using the PCA data i have reduced my test data to 30 PCA as well with the same Eigen vector for training data which increases my accuracy in test data. The result of my Testing in both kind of data set is given below:



```
time elapsed:  469.3757948875427
label precision recall
    0     0.936  0.979
    1     0.956  0.973
    2     0.921  0.870
    3     0.892  0.896
    4     0.897  0.925
    5     0.888  0.827
    6     0.928  0.939
    7     0.915  0.897
    8     0.849  0.869
    9     0.880  0.883
Test Accuracy:  0.9071
```

(a) Training and testing logistic regression with raw data



```
time elapsed:  22.573315143585205
label precision recall
    0     0.933  0.933
    1     1.000  0.271
    2     0.912  0.751
    3     0.848  0.749
    4     0.924  0.747
    5     0.993  0.155
    6     0.945  0.839
    7     0.987  0.715
    8     0.269  0.988
    9     0.823  0.710
Test Accuracy:  0.6842
```

(b) Training and testing logistic regression with 30 PCA

Figure 8: Training and testing logistic regression time and accuracy

Since I have used my own logistic regression, The time taken may be high. But since both the test is done in the same environment, we can compare their result. As we can see, I have done 1000 iterations for both the cases. The time taken for Raw data is about 460s where for the 30 PCA, the time is only about 23s. So this is a huge improvement in time. But for the accuracy i have found the raw data has accuracy of 0.9 whereas for 30 PCA, the accuracy is about 0.69. I am not sure why accuracy is low for PCA. There might be some ways to improve the result for PCA, But unfortunately, Raw datas accuracy was more for my case.

Problem 2.6

Neural networks generally perform supervised learning tasks, building knowledge from data sets where the right answer is provided in advance. The networks then learn by tuning themselves to find the right answer on their own, increasing the accuracy of their predictions.
Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. Neural networks help us cluster and classify.

After training the logistic regression, we know there 784 weight associated with each output node (0 to 9). Here I have reshaped 784 weights to 28x28 shape images. The images that I got based on each class level is given below.

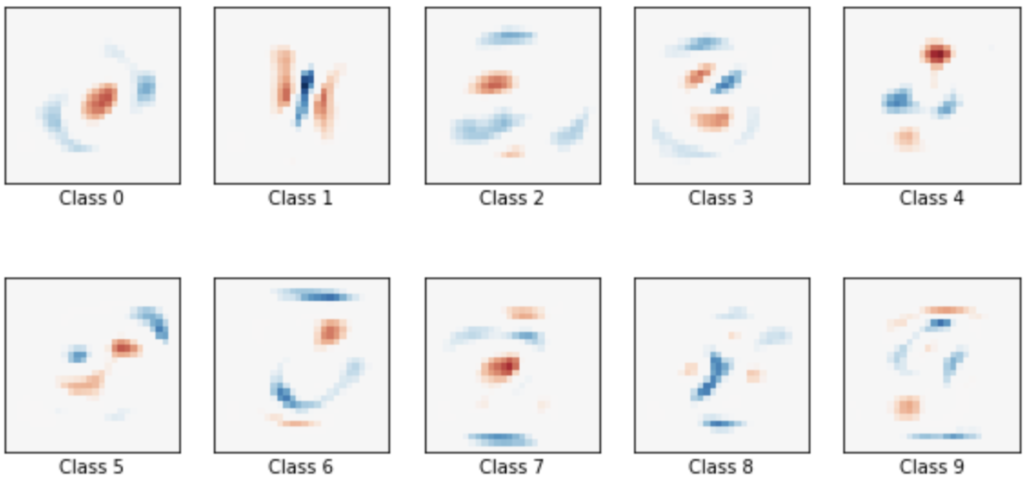


Figure 9: Visualization of 10PC as image

This is much more interesting to see. As I can visualize some of the numbers from the picture. Since this output node corresponding to digits from 0 to 9, I can relate the picture of each class to the digits. As we can see, class 0 is related to 0, 1 can be visualized from class 1, 3 and 4 are clearly visible, 6, 7 and 9 are visible to. This is so much interesting compared to PCA. As PCA projects the data based on highest variance, In the 1st 10 Projection we could not find any shape, but for logistic regression weigh, The weight is given based on output value itself, we see some resembles with the output.