Project-1 Report Mnist digit classification

Karthik Satya Raghuram Bylapudi (1225577730)

Introduction:

The problem statement can be briefly summarized as performing mnist digit classification using Bayesian decision rules over multivariate normal distribution on a reduced dimension feature space.

The dimension reduction has to be performed using PCA over the training samples. We also need to plot and analyze the findings and observations of the 2-d projected data.

The dataset provided is a subset of mnist digit recognition dataset of digits 5 and 6. It comprises of 28x28 sized images. The statistics are - Number of samples in the training set Digit 5 - 5421, Digit 6 - 5918, Number of samples in the testing set digit 5 – 892, digit 6 – 958. Also note that the priors of both classes are given to be same (0.5).

Method:

The coding language chosen is python and the packages used are seaborn, numpy, pandas and matplotlib.

Step1 (preprocess the data to get into the required format):

The data presented to us are of .mat format images of digits 5 and 6 each of dimension 28x28 pixels.

The data is loaded as a dictionary. Then we extract the values of the key pertaining to train and test data of digits 5 and 6. The data is then formatted by traversing columnwise and doing list.extend essentially converting the (x,28,28) array to (x,784) list. This list is converted to array and labels 0 and 1 are added as a new column for digits 5 and 6 respectively and all the data is merged/concatenated making the array of size (11339,785) for train and (1850,785) for test.

Step2 (normalize the data):

This step involves normalizing the data points by subtracting the mean and dividing with standard deviation for each feature. (mean and std_dev are calculated for each feature individually). Note that we are checking if the std_dev is non zero before we divide as if it is 0 that essentially means data point and mean are same over all points so the normalized value can be taken as 0 (data – mean).

Step3 (apply PCA):

This is the most important step although python makes it short. Essentially for the normalized data we obtained, we calculate the covariance matrix and with respect to that covariance matrix we calculate the eigen values and eigen vectors. The computation of covariance matrix is simply ((x-mean)x(transpose(x-mean)))/n which is done by using the inbuilt python function np.cov. similarly the eigen values are obtained by solving cov – (lambda x Identity) = 0. The corresponding eigen vectors are obtained by solving the same equation but with unknown eigen vector e, and its known corresponding eigen value lambda. All this is performed by the inbuilt function np.linalg.eig. Once the eigen values and vectors are obtained, sort the eigen vectors in ascending order of their eigen values. For our case we obtain the top 2 eigen values as 51.41078512078789 and 42.16026337982371. We choose the corresponding eigen vectors onto which we project our data (perform a dot product over data and eigen vectors).

Step4 (Density estimation)

We know that for a given dataset/sample D, the closest normal distribution that can be fit over those data points must have mean as sample mean and variance as sample variance (which can be computed by performing MLE over normal distribution with unknown mean and variance).

- Normal density with unknown mean μ and Σ
  - First consider only 1-D case, $\theta = (\theta_1, \theta_2) = (\mu, \sigma^2)$

$$\log\ p(x_k \mid \boldsymbol{\theta}) = -\frac{1}{2}\ln 2\pi\theta_2 - \frac{1}{2\theta_2}(x_k - \theta_1)^2$$

Set $\nabla_\theta l(\boldsymbol{\theta}) = 0$, we have

$$\begin{cases} \sum_{k=1}^{n} \frac{1}{\theta_2}(x_k - \theta_1) = 0 \\ -\sum_{k=1}^{n} \frac{1}{\theta_2} + \sum_{k=1}^{n} \frac{(x_k - \theta_1)^2}{\theta_2^2} = 0 \end{cases}$$

$$\rightarrow \quad \hat{\mu} = \frac{1}{n}\sum_{k=1}^{n} x_k$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{n}(x_k - \hat{\mu})^2$$

14

So now we compute the sample mean and sample variance matrices which define the respective normal distributions for each class. We also define our multivariate normal equation

$$f_{\mathbf{X}}(x_1, \ldots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}}$$

in the function multivariate_normal_pdf. The function takes in the mean, variance and data as its input and returns the value f(X).

Step5 (Bayesian Decision Theory for optimal classification)

We finally defined the classifier function which classifies the data points into respective classes based on the class conditionals obtained. If cl_cond_1 (= multivariate_normal_pdf(mean_of_digit_5,cov_of_digit_5,data) * prior_P_5) > cl_cond_2 (= multivariate_normal_pdf(mean_of_digit_6,cov_of_digit_6,data) * prior_P_6) then classify as class 0 (digit 5) else classify as class 1 (digit 6)

Results and Observation:

Task 1 gives us vectorized and normalized train and test set.

```
In [380]: x[0]

Out[380]: array([ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00, -9.39143104e-03, -9.39143104e-03,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00, -9.39143104e-03, -9.39143104e-03,
                 -9.39143104e-03, -1.82705863e-02, -1.58695090e-02, -9.39143104e-03,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00, -9.39143104e-03, -1.46680171e-02,
```

Task 2 gives you the following eigen values:

```
In [381]: print(eig_pairs[0][0],eig_pairs[1][0])
          51.41078512078789 42.16026337982371
```

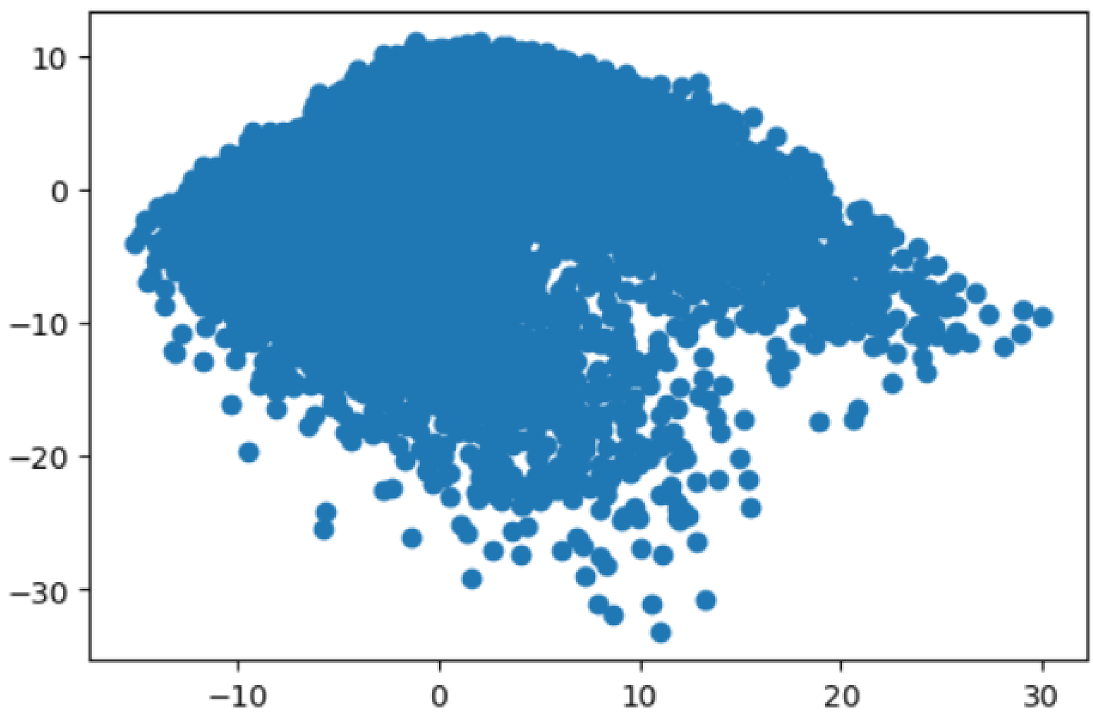
And corresponding eigen vectors.

Task 4

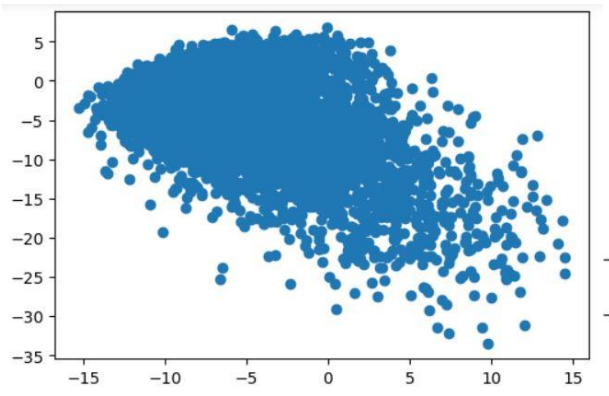
We then get the projected data: train shape (11339,2), test_shape (1850,2) .

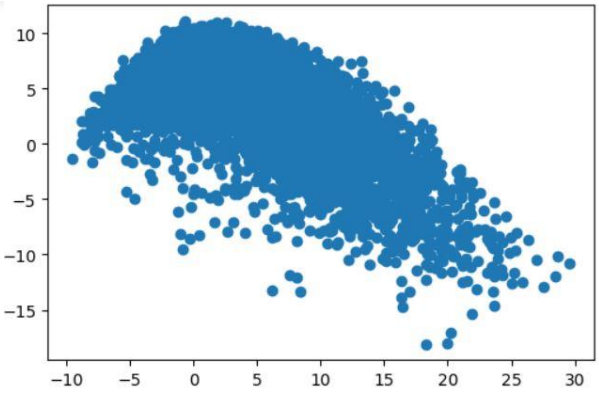
Then, plotting the projected data enables us to observe the following –

Scatter plot of the projected training data with respect to principal components for digits 5 and 6. as we observe the data is concentrated onto narrow range of (-10,0) for both pc1 and pc2 for digit 5 and between (0,10) for PC1 and (-5,10) for PC2 for digit 6. digit 6 plot kind of resembles a normal distribution but I am unable to say the same for digit 5.



Train dataset

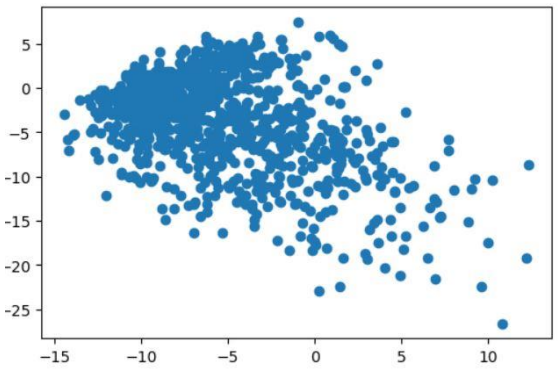


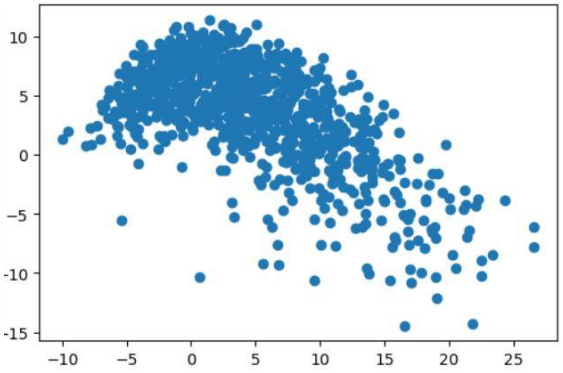


Digit 5

Digit 6

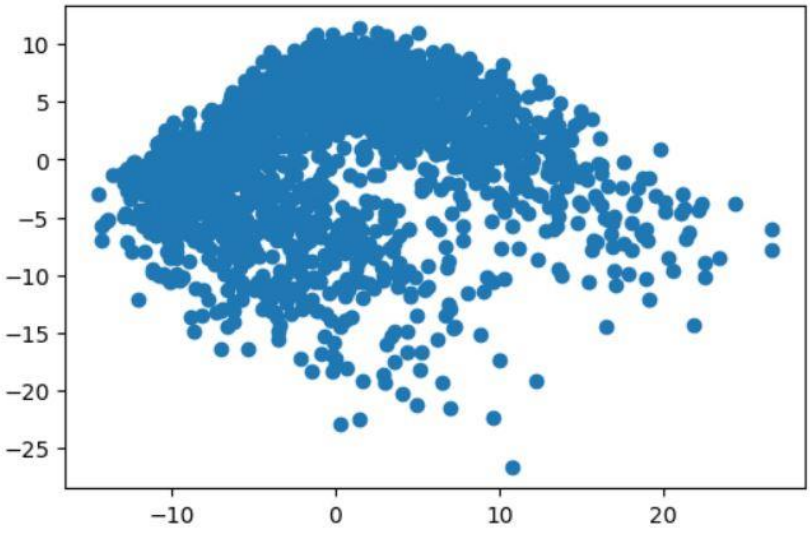
For the test dataset as well we observe the data points to resemble normal distribution after PCA projection. the data is also resembling the distribution of train which Is to be expected since our assumption is our train and test set come from same true distribution.





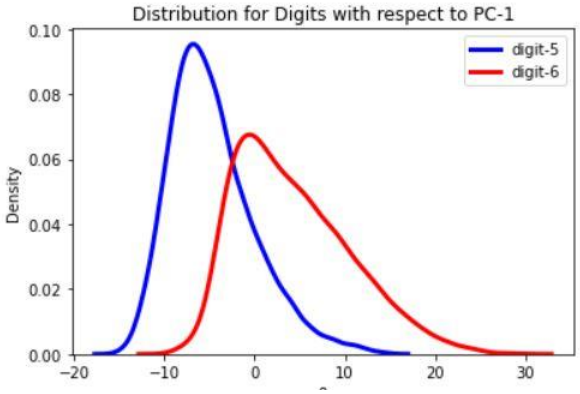
Digit 5 test

Digit 6 test

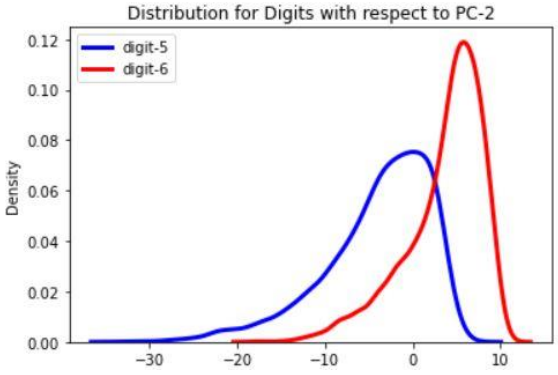


Test set

We further analyse the plots - The below 4 plots are for the distribution of components PC1 and PC2 with respect train and test. Essentially they show the density of value of Principal component for digit 5 and 6 for train and test. This plot would kind of give us a good idea as to how the values of features/PC are distributed with respect to the class.

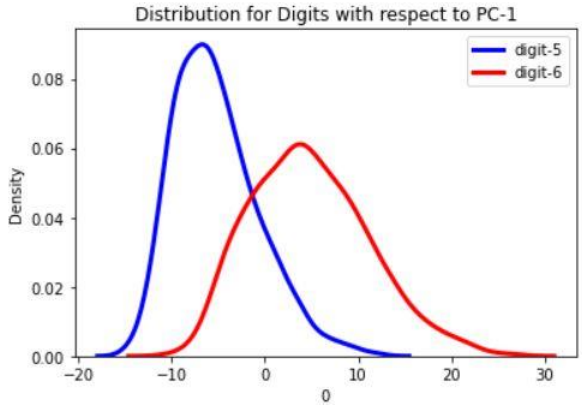


Train PC1

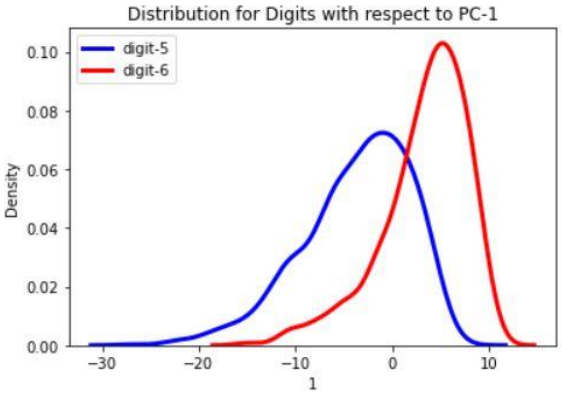


Train PC2

We observe that the value of PC1 for train falls in a narrow range of around (-15,0) and has a shar peak indicating for digit 5 most values of PC1 are concentrated around that narrow range. Similarly with respect to PC2, we observe an even sharper peak for digit 6 indicating the PC2 values of digit 6 lie in the narrow range of (0,10).



Test PC1



Test PC2

We observe pretty much similar observations even in the test data set. This indicates that our test set has been taken from a distribution that is generated by our training samples. Which isn't surprising since our train and test data must belong to the same true distribution.

Task 4:

The density estimates are as follows:

Digit 5 mean and covariance :

```
In [37]: print(mean_of_digit_5,'\n',cov_of_digit_5)

[-4.4531976 +0.j -4.06950581+0.j]
 [[ 23.39752387+0.j -15.13656603+0.j]
 [-15.13656603+0.j  36.44251424+0.j]]
```

Digit 6 mean and covariance :

In [40]:
```
print(mean_of_digit_6,'\n',cov_of_digit_6)
```

```
[4.07921328+0.j 3.72774434+0.j]
 [[ 42.26834766+0.j -17.94685568+0.j]
 [-17.94685568+0.j  18.33380651+0.j]]
```

Task 5:

Running the classifier and calculating the train and test accuracy give us the following results.

In [89]:
```
"""Here we define the classifier which essentially calculates the class conditionals. Here the data points are substituted in
the multivariate normal pdf we calculated with mean and variance as sample mean and variance for each class. This value is
calculated for each class and multiplied with the prior. As per our knowledge of bayesian decision theory, we pick the class
having the greater class conditional out of the given distribution"""
def classifier(data, prior_P_5, prior_P_6):

    cl_cond_1 = multivariate_normal_pdf(mean_of_digit_5,cov_of_digit_5,data) * prior_P_5
    cl_cond_2 = multivariate_normal_pdf(mean_of_digit_6,cov_of_digit_6,data) * prior_P_6

    if cl_cond_1 >= cl_cond_2:
        return 0
    return 1
```

In [90]:
```
pred_train = [classifier(i, prior_P_5, prior_P_6) for i in p_comp]
pred_test = [classifier(i, prior_P_5, prior_P_6) for i in p_comp_test]
```

In [91]:
```
train_df['pred_train'] = pred_train
test_df['pred_test'] = pred_test
```

In [92]:
```
"""We have obtained a training accuracy of 94.277 and testing accuracy of 94.11"""
training_acc = np.mean(train_df['label'] == train_df['pred_train'])
print(training_acc)
```

```
0.9427639121615663
```

In [93]:
```
testing_acc = np.mean(test_df['label'] == test_df['pred_test'])
print(testing_acc)
```

```
0.9394594594594594
```

Train accuracy: 94.28%, Test accuracy: 93.946%

Conclusion:

The following project is an excellent showcase of the power of PCA. The dimensions were drastically reduced from 784 to 2, which for the given dataset of 11339 train images and 1850 test images, reduces our computation by ~99.75%. We also observe that the projected data after PCA resembled slightly as normal distributions (digit 6 especially) since they were concentrated over particular ranges of values. Thus, enabling our simple multivariate normal distribution classifier to work relatively well (93.946% accuracy) with minimal computation requirement. We also saw that for any given dataset D, the best fit normal distribution would be having the mean as sample mean and variance as sample variance.