Amarthya Kathula ASU Id: 1229583071 akathula@asu.edu

# CSE 569: Fundamentals of Statistical Learning and Pattern Recognition

# Project Part 1

## Introduction:

This phase of the project includes performing a minimum error rate classification for the provided data set of images of numbers '5' and '6'. [1] The given data files are in '.mat' format, which can be viewed in either MATLAB or using the SciPy library, and there are a total of 11339 training images out of which 5421 are images of digit '5' and 5918 belong to digit '6', and a total of 1950 images in the testing set, of which 892 belong to the first class (digits '5') and 958 belong to the second class (digits6').

According to the given data description, the original '.mat' files stored the images in 28*28 array format, and this raw data is pre-processed. A few data preprocessing steps include vectorization, feature normalization, dimensionality reduction, etc.; specifically, there are five tasks in this project, from feature normalization to performing minimum error rate classification and reporting accuracy.

## Methodology and Implementation:

The successful application of BDT for image classification requires careful data preprocessing. The implementation steps are as follows:

[1] Data Loading
[2] Vectorizing images
[3] Feature Normalization
[4] Dimensionality Reduction using *PCA*
[5] Density Estimation
[6] Applying Bayesian Decision Theory to classify the images.

In almost all cases of Machine Learning applications, the data is always pre-processed to improve the data quality, avoid the curse of dimensionality, reduce noise, and have many other benefits. Here the images are vectorized as it is difficult to identify the features and apply BDT.

***Vectorization*** [2]: First, all training samples are vectorized by concatenating their columns to form a 784-dimensional vector. The images in the data sets are loaded into NumPy arrays by using the SciPy library. These arrays are reshaped into 2-D arrays using the *reshape* method, where each image is flattened into a 1-D vector with a size of 784. (28*28 = 784). This results in a 2-D array with one dimension indicating the number of images and a second dimension indicating the size of each image in a 1-D vector.

***Normalization*** [3] After vectorization, the data is normalized, particularly the features of the images in the training dataset are normalized to facilitate effective BDT application. This step ensures that the data is consistent and reduces the difficulties associated with applying BDT in high-dimensional space. In this case, the Normalization is done by using Z-Score Standardization, this process ensures that all the features are on a common scale and that the data is centred around the mean. Using the NumPy library the mean and standard deviation for all the features are calculated, then the mean is subtracted from every feature value and the resulting value is divided by SD. By applying Z-Score standardization to each feature in the dataset, we ensure that the data is appropriately pre-processed for the subsequent steps in the classification process.

```
e = 1e-8 # Small constant to avoid division by zero
normalized_images = (vectorized_images - mean) / (std + e)
```

the above code is used to normalize the features of the images. There are some cases where the standard deviation is 0, then for that feature when we try to normalize, it results in 'division by zero', hence to avoid this we add a small constant 'epsilon'/'e'.

***Dimensionality Reduction using PCA*** [4]: Principal Component Analysis (PCA) is a dimensionality reduction technique used in the ML field for data analysis. Dimensionality reduction is a process where the dimensions of the data or features of the data are reduced because using high-dimensional datasets with ML algorithms may result in overfitting and be computationally costly.

In this project, PCA is done using all the training samples. As the data is centred using the mean previously, the next step is to compute the Covariance Matrix, this can be done by using the 'cov' method in the NumPy library as described below. This matrix describes the relationships between all pairs of the features in the dataset.

```
covariance_matrix = np.cov(normalized_images, rowvar=False)
```

The next step is to calculate the eigenvalues and eigenvectors of the covariance matrix. The direction of maximum variance is represented by the eigenvectors, and the amount of variance explained by each primary component is indicated by the eigenvalues. After finding the eigenvectors and eigenvalues we sort them in descending order, to find the biggest eigenvalue, as it is correlated with the first principal component, the second largest with the second component, and so on. The eigenvectors and values are computed as below

```
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
```

Based on the given problem statement, only 2-D projections of the samples on the first and second principal components are considered, i.e., only the projections of two eigenvalues are considered for further computation. Now, by using these eigenvectors we reduce the data as below

```
reduced_train_images = np.dot(normalized_images, selected_eigenvectors)
```

'np.dot()' is a NumPy function for matrix multiplication, the above code snippet projects the data onto the principal components that are represented by the eigenvectors.

The datasets can be visualized using the library 'matlplotlib'

***Density Estimation*** [5]: It's a fundamental step in BDT since it is used to model the probability distributions of the data from different classes and it is useful to classify new images. To estimate the probability densities for all the classes '5' and '6' we need to calculate the mean and covariance matrices for both the classes

```
mean_train_5 = np.mean(projections_train_5,axis=0)
cov_matrix_train_5 = np.cov(projections_train_5, rowvar=False)
mean_train_6 = np.mean(projections_train_6,axis=0)
cov_matrix_train_6 = np.cov(projections_train_6, rowvar=False)
```

The reduced-dimensional data projections for the training images of the number '5' are found in this case in the projections_train_5 folder. The centre of the distribution is represented by the mean_train_5, which is the mean of these projections. The covariance matrix, or cov_matrix_train_5,

represents the distribution and correlation of the data points. When combined, these characteristics offer insightful information about the '5' class distribution. It is similar for class '6'.

***Applying Bayesian Decision Theory to classify the images*** [6]: In this final phase, the estimated probability distributions are used to perform minimum-error-rate classification. There are many performance metrics to find the performance of the model, here in this project accuracy of the classification model is reported as the key performance metric. By using the estimated parameters (mean and covariance matrix) we compute the likelihood of the images/data points for each class, and the image is classified into a class with the highest likelihood.

```
class_likelihood_5=multivariate_normal.pdf(data_point,mean_train_5,cov_matrix_train_5)
```

```
class_likelihood_6=multivariate_normal.pdf(data_point,mean_train_6,cov_matrix_train_6)
```

from the SciPy library, the 'multivariate_normal' module is imported and it is used for the computation of the likelihood. Now as the data points are classified, we can calculate the accuracy of the model.

```
accuracy_training_data=correct_classified_train_data/len(reduced_train_images)*100
```

Similarly, the accuracy is calculated for the training dataset as well, while doing this we use the parameters estimated using the training set.

## Results and Observations:

**Task 1: Feature normalization (*Data Conditioning*)**

- *Mean and Standard Deviation Calculation*: Using the entire dataset the mean and standard deviation are calculated for each of the 784 features that represent the images after they are vectorized. Now we have two arrays one for the mean and one for SD, the dimensions of these arrays are 784*1, as they are calculated for each feature and there are 784 features.
- Then as mentioned above these values are used to make sure that the data is on one scale and it is centred around the mean.

**Task 2: PCA Using the Training Samples**

- *Covariance Matrix Computation*: Principal Component Analysis is performed using the training samples and the computed eigenvalues and vectors are arranged in descending order to find the topmost eigenvalues.

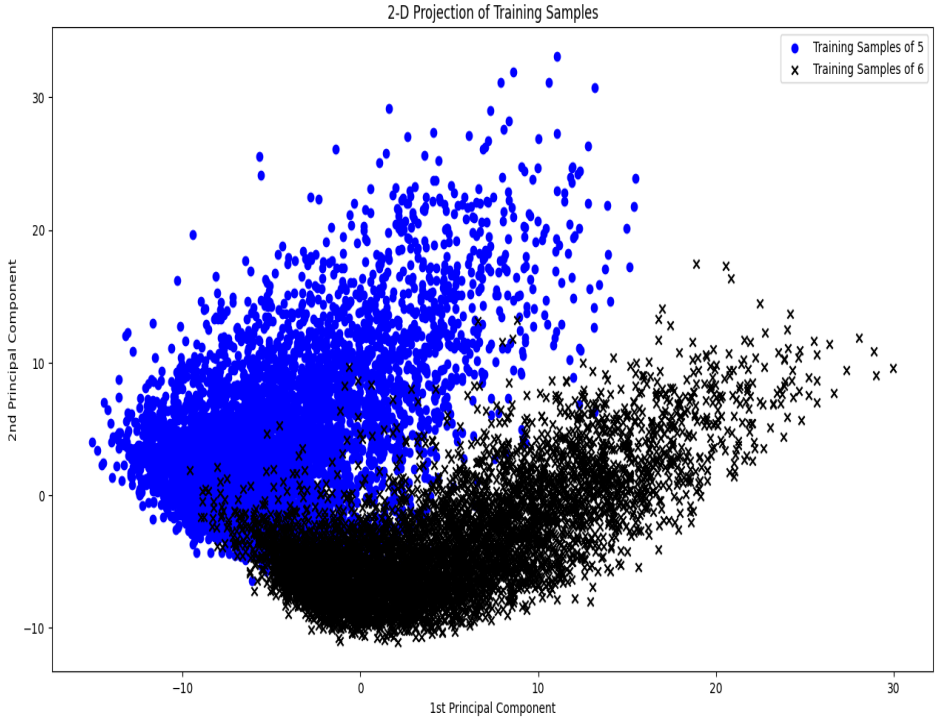
  *Eigenvalues*: [ 5.14108597e+01 4.21602555e+01 2.97938050e+01 1.95741803e+01
  1.90185714e+01 1.50982938e+01 1.29295188e+01 1.09471527e+01…….
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00] These are the eigenvalues after rearranging them in descending order.

**Task 3: Dimension reduction using PCA**

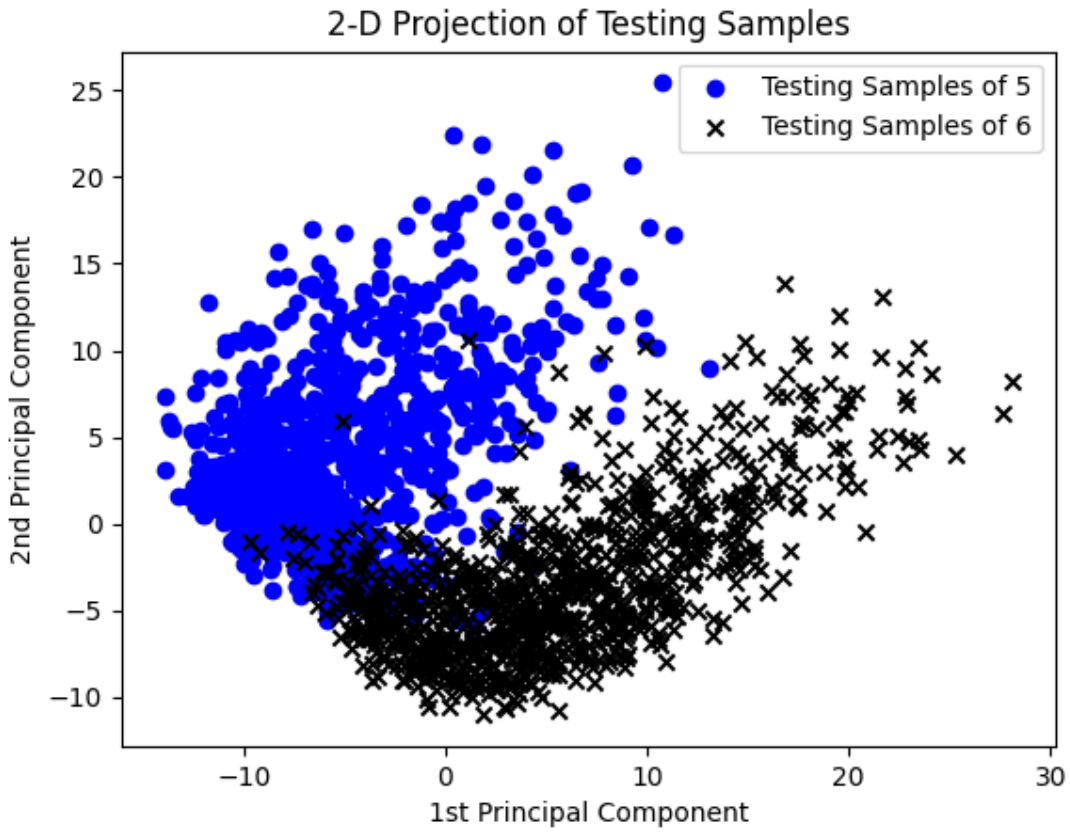
- *2-D Projections*: Only the 2-D projections of the samples on the first and second principal components are considered, these projections represent the new representations of the samples.

The eigenvalues of the selected two components are 5.14108597e+01(first component) and 4.21602555e+01(second component).

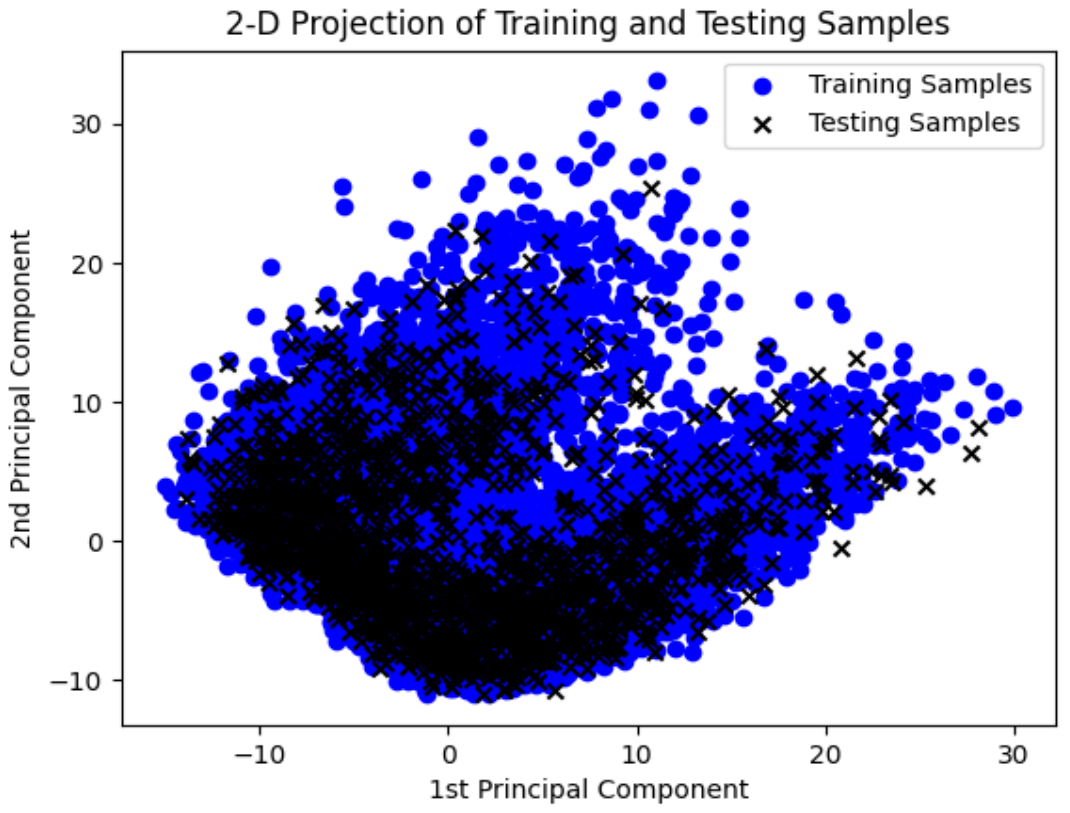
- *Data Visualization*: Both the training and testing data sets are visualized in this 2-D space. Observations were made regarding how the two classes are clustered in this reduced-dimensional space.




This image contains the scatter plot of all the training samples

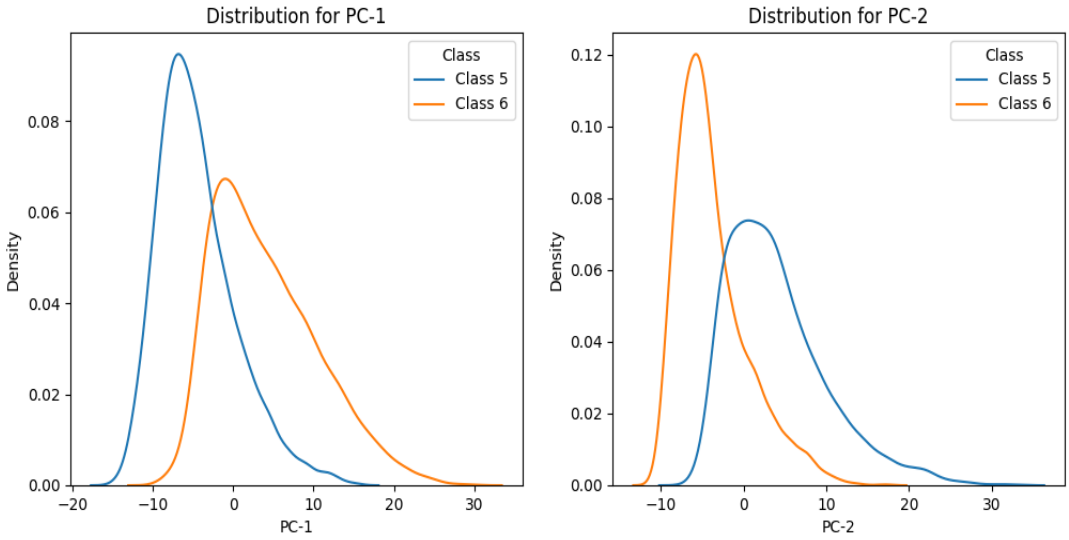



This image contains the scatter plot of all the training samples




This image contains the scatter plot of all the training and testing samples

- *Distribution Analysis*: Here based on the below plots we can see that the data is normally distributed.




**Task 4: Density Estimation**

- *Parameters for Gaussian Distributions*: Using the training data, the parameters for 2-D Gaussian distributions for each class, are estimated. Hence, two distributions one for each digit are obtained.

```
mean_train_5 = np.mean(projections_train_5,axis=0)
cov_matrix_train_5 = np.cov(projections_train_5, rowvar=False)
mean_train_6 = np.mean(projections_train_6,axis=0)
cov_matrix_train_6 = np.cov(projections_train_6, rowvar=False)
```

The above code is used to calculate the parameters for each class, are the values of the parameters are as follows:

Class '5' :

- mean_train_5 = [-4.45320748 4.06951377]
- cov_matrix_train_5 = [[23.39792743 15.13683929]
  [15.13683929 36.44222332]]

Class '6' :

- mean_train_6 = [4.07922233 -3.72775171]
- cov_matrix_train_6 = [[42.26796632 17.9467385]
  [17.9467385 18.33394357]]

**Task 5: Bayesian Decision Theory for Optimal Classification**

- *Minimum-Error-Rate Classification*: Bayesian Decision Theory using the estimated distributions is used to classify the images. This classification process is used to determine the optimal class assignment for both the training and testing datasets.
- *Accuracy Results*: The accuracy of the model is computed for both the training and testing data sets and the accuracy is as follows:

  **Training Set Accuracy**: 94.27639121615663%

  **Testing Set Accuracy**: 93.83783783783784%

## Conclusion:

In this phase of the project, a series of data pre-processing and classification is implemented to classify handwritten digits '5' and '6'. The key findings include:

- ✓ Feature normalization ensured data consistency and prepared it for analysis.
- ✓ The 2-D structure of the data was exposed through PCS and dimension reduction, providing insights into class distribution.
- ✓ The parameters for each class's 2-D Gaussian distributions were determined by density estimation.
- ✓ Obtained accurate and reliable results for both training and testing datasets, 94.27639121615663% and 93.83783783783784% respectively, based on the estimated distributions using Bayesian Decision Theory.

There are many broad applications for such ML techniques in pattern recognition and classification models, the model can be further refined and optimized to enhance accuracy and efficiency when it is applied to real time data.