Assignment (1): PCA Algorithm Implementation

Course Name: ***Neural Networks***
Course Code: ***CSE440***

Submitted to:

Dr. Mahmoud Khalil
Eng. Amr Yassin

Faculty of Engineering, Computer Engineering and Software Systems

Student Information

Student Name:
Ahmed Mamdouh Mohammed

Student ID:
16P6020

Table of Contents

1.0 INTRODUCTION:

Principal Component Analysis (PCA) is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. In this report, I will discuss how I implemented PCA using Python with NumPy and Matplotlib. I implemented everything from scratch and compared my results to the results of the PCA method provided by the SciKit-Learn library by comparing the produced plots from my output to the produced plots of the SciKit-Learn's PCA method output. This was done to check if my results are valid.

The input feature vectors have a shape of (N x D) where N is the number of data points and D is the number of dimensions. The output shall be (N x K) where K is the number of dimensions after reduction.

I assumed that the dimensions after reduction (K) will always be equal to D. To be able to plot the output data on a 2D scatter plot, I only took into consideration the two principle components which have the highest variability. I also assumed that the dimension of the input feature vectors (D) can be any value, but I will only use the first two dimensions of features to plot the input data on a 2D scatter plot, this is so that I could be able to have a sense of what the input data may look like. To test my implementation of PCA, I used the Iris dataset which is provided through the SciKit-Learn library.

In the next section, I will explain the main parts of the code without going into too much details as the code itself has a lot of comments to explain all the nitty-gritty details.

2.0 Code Walkthrough:

2.1 Main Function:

The main function contained the most general parts of the script. At first, I loaded the Iris dataset into two array variables separating the features from the class labels. Secondly, I plotted the first two features of the Iris dataset using matplotlib. Then, I called the “apply_pca” function passing to it the input matrix of features, the function then returns the projected features (principle components). I plotted the first and second principle components using matplotlib. At the end, I used Scikit-learn’s PCA function on the Iris dataset and plotted its output in order to test my own output and compare it against a standard PCA implementation.

```
199
200  def main():
201
202      """ Main entry to the program """
203
204      # input_matrix = initialize_random_matrix((1000,5))  # Random data points
205      # if(input_matrix is None):
206      #     return
207
208      # Loading the iris dataset
209      iris_dataset = datasets.load_iris()
210      input_matrix = iris_dataset.data
211      labels = iris_dataset.target
212
213      # Plotting first feature against the second feature
214      display_2D_scatter_plot(input_matrix, "Input Dataset", "X1", "X2", labels)
215
216      # Applying my implementation of PCA to the Iris dataset
217      pc_matrix = apply_pca(input_matrix)
218
219      # Plotting the two principle components with highest variability
220      display_2D_scatter_plot(pc_matrix, "Projected Dataset", "PC1", "PC2", labels)
221
222      # Applying Scikit's PCA function to the dataset and plotting its output to test my output
223      pc_matrix_scikit = apply_pca_scikit(input_matrix)
224      display_2D_scatter_plot(pc_matrix_scikit, "Projected Dataset Using SciKit", "PC1", "PC2", labels)
225
```

2.2 Applying PCA Function:

The “apply_pca” function takes the input matrix of features and normalizes it. Then, the normalized matrix is passed to a function called “calculate_covariance” which calculates the covariance matrix. The covariance matrix is then used to calculate the eigenvectors and eigenvalues using the “eig” function provided by “np.linalg”. The eigenvectors and eigenvalues are passed as parameters to the “sort_eigenvectors” function which returns a list of sorted eigenvectors in which the eigenvectors at the beginning of the list contain much more variability (higher eigenvalues) than the eigenvectors towards the end of the list (lower eigenvalues). Finally, the eigenvectors list is used to multiply each eigenvector by the input normalized matrix in order to calculate the principle components. The principle components are appended to a list called “pc_list” in the same order of their eigenvectors and the “pc_list” is returned.

```
def apply_pca(input_matrix : np.ndarray):
    """
    This function applies the steps of the PCA algorithm to a numpy array of
    shape (n,d) and returns a numpy array of shape (n,k)

    PARAMS:
        input_matrix - The numpy array of shape (n,d) in which PCA will be applied to

    RETURNS:
        pc_list - A list of k principle components each of shape (n,1)
        (k == d for this script)

    """

    # normalizing the input_matrix
    normalized_matrix = normalize_matrix(input_matrix)

    # calculating the Covariance matrix
    covariance_matrix = calculate_covariance(normalized_matrix)

    # calculating the eigenvalues "v" and eigenvectors "u"
    v, u = np.linalg.eig(covariance_matrix)

    # Sorting the eigenvectors such that the vectors with higher eigenvalues are placed first
    eigenvectors_list = sort_eigenvectors(v, u, k = input_matrix.shape[1]) # values, vectors, number of dimensions

    # calculating the principle components and appending them to the pc_list
    pc_list = []

    for vector in eigenvectors_list:
        PC = normalized_matrix.dot(vector)
        PC = PC.reshape((PC.shape[0], 1))
        pc_list.append(PC)

    return pc_list
```

2.3 Plotting Function:

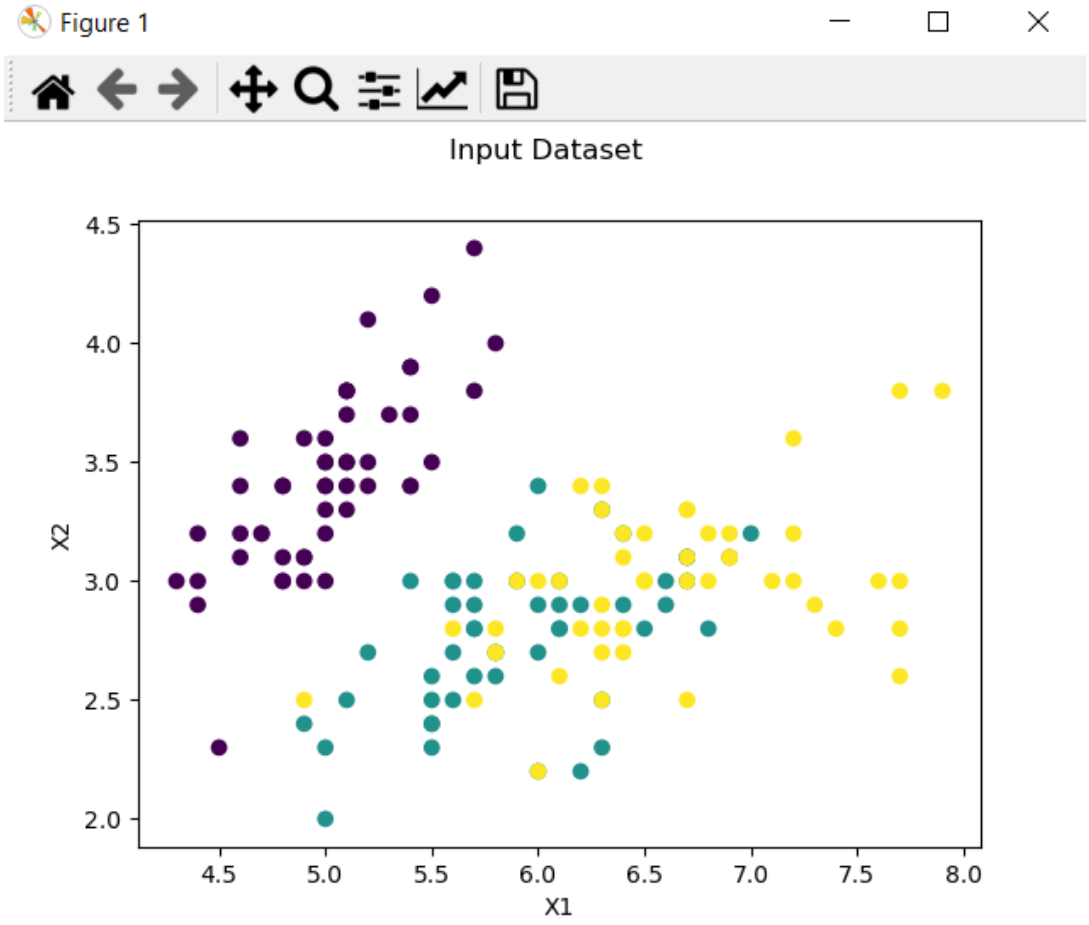
The “display_2D_scatter_plot” function is used to plot either the input or the output from the PCA. It takes parameters such as title, xlabel and ylabel depending on the required plot. The most important parameter is the dataset which is the data to be plotted. Only two dimensions are taken into consideration in the plot.

```
def display_2D_scatter_plot(dataset, title, xlabel, ylabel, labels = None):
    """
    This function displays the input dataset as a form of scatter plot assuming
    that the dataset is 2D only.

    PARAMS:
        dataset - a numpy array of shape (n,2) which has n data points and
        only 2 dimensions

        title - string holding the title of the figure to be displayed

        xlabel - string holding the label for the x axis of the plot

        ylabel - string holding the label for the y axis of the plot

        labels - the target class for each observation. This is used for visualization

    """

    plt.figure()

    if(type(dataset) == list): # Because I store the principle components in a list data structure
        plt.scatter(dataset[0], dataset[1], c = labels)
    else:
        plt.scatter(dataset[:,0], dataset[:,1], c = labels)

    plt.suptitle(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show()
```

3.0 Input/Output Plots:

3.1 Input Plot:

I used the Iris dataset as the input in this script. It has a shape of (150 x 4) so in order to plot it, I only took into consideration the first two features in my scatter plot. The colors of the observations on the plot correspond to the class label of the observation. The plot is shown below.



3.2 Output Plot:

The output of my PCA function is a list of 4 principle component arrays each of which have a shape of (150, 1). I only took into consideration the 2 principle components resulted from the eigenvectors corresponding to the highest eigenvalues. These two principle components show a lot of variability in the data which means they would result in a good representation. My output plot is shown below.