Implementaton of Principal Component Analysis (PCA) in MATLAB

1 Introduction

Principal component analysis (PCA) is a statistical technique that is used to identify patterns in data and to reduce the dimensionality of the data. It does this by finding a new set of variables, called principal components, that capture the most important patterns in the data.

The first principal component is the direction in the data that captures the most variation. The second principal component is the direction in the data that captures the second most variation, and so on. Each subsequent principal component is orthogonal(perpendicular) to the previous ones, which means that they capture patterns in the data that are independent of the patterns captured by the previous components.

PCA is often used to visualize high-dimensional data, to reduce the complexity of the data, or to identify patterns in the data that might not be apparent when looking at the raw data. It is also used in a variety of applications, such as image recognition, speech recognition, and natural language processing.

Note: The concept of PCA is difficult to understand without visuals, so I will suggest you watch this video before we continue reading.

2 Method

Before we get to MATLAB, we need to know the steps in performing PCA. The steps are explained in the subsections below.

2.1 Standardize the data

It is generally a good idea to standardize the data before performing the principal component analysis (PCA). Standardization involves subtracting the mean from each feature and dividing by the standard deviation, which helps to give all the features the same scale. This is important for PCA because it allows the algorithm to treat all the features equally, rather than giving more weight to features with larger numerical values. There are a few cases where standardization may not be necessary, such as when all the features have the same scale already, or when the features have already been transformed in some way that gives them the same scale. However, in most cases, standardization is recommended as a preprocessing step before performing

PCA.
Let X be an $(n \times m)$ matrix representation of our data. That is,

$$\begin{array}{c} \quad\quad X_1 \quad\quad X_2 \quad\quad X_3 \quad \cdots \quad X_m \\ X = \begin{bmatrix} X_{11} & X_{12} & X_{13} & \cdots & X_{1m} \\ X_{21} & X_{22} & X_{23} & \cdots & X_{2m} \\ X_{31} & X_{32} & X_{33} & \cdots & X_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & X_{n3} & \cdots & X_{nm} \end{bmatrix} \end{array} \tag{1}$$

where $X_j \; for \; j = 0, 1, 2, ..., m$ represent each feature. Then each value in X_j is standardized as follows. First calculate the mean with (Equ.2).

$$\overline{X_j} = \frac{\sum_{i=0}^{n} X_{ij}}{n} \tag{2}$$

Next calculate the variance with (Equ.3).

$$Var(X_j) = \frac{\sum_{i=0}^{n} (X_{ij} - \overline{X_j})^2}{n-1} \tag{3}$$

and replace the value using (Equ.4)

$$X_{ij} = \frac{X_{ij} - \overline{X_j}}{\sqrt{Var(X_j)}} \tag{4}$$

2.2 Compute the covariance matrix

The covariance matrix is a square matrix that contains the pairwise covariances between all the variables in the dataset. It is used to measure the strength of the linear relationship between pairs of variables. Let Z represent the matrix of standardized data, then the covariance matrix ($COV(Z)$) is given by (Equ.5)

$$\begin{bmatrix} COV(X_1, X_1) & COV(X_1, X_2) & COV(X_1, X_3) & \cdots & COV(X_1, X_m) \\ COV(X_2, X_1) & COV(X_2, X_2) & COV(X_2, X_3) & \cdots & COV(X_2, X_m) \\ COV(X_3, X_1) & COV(X_3, X_2) & COV(X_3, X_3) & \cdots & COV(X_3, X_m) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ COV(X_m, X_1) & COV(X_m, X_2) & COV(X_m, X_3) & \cdots & COV(X_m, X_m) \end{bmatrix} \tag{5}$$

2.3 Compute the eigenvectors and eigenvalues of the covariance matrix

The eigenvectors and eigenvalues of the covariance matrix are used to determine the principal components of the dataset. The eigenvectors are the directions along which the data varies the most, and the eigenvalues are the magnitudes of these variations.

Let $A = COV(Z)$. To find the eigenvalues and eigenvectors of A, you can follow these steps:

1. Start by finding the characteristic polynomial of the matrix. This is a polynomial equation whose roots are the eigenvalues of the matrix. To find the characteristic polynomial, you can use the following formula:

$$det(A - \lambda I) = 0 \tag{6}$$

where A is the matrix, λ is a scalar value, and I is the identity matrix.

2. Solve the characteristic polynomial for the values of λ that make the equation equal to zero. These values are the eigenvalues of the matrix.

3. For each eigenvalue, find the corresponding eigenvector. To do this, you can use the following formula:

$$(A - \lambda I)v = 0 \tag{7}$$

where v is the eigenvector. This equation represents the condition that the matrix $(A - \lambda I)$ must be singular (i.e., its determinant must be zero) for the given eigenvalue λ.

4. Solve the equation for v. This will give you the eigenvector corresponding to the given eigenvalue.

5. Repeat the process for each eigenvalue to find all of the eigenvectors.

2.4 Select the principal components

The number of principal components to keep is typically determined by the amount of variance in the data that needs to be explained or the number of dimensions that are needed to represent the data. The principal components are the eigenvectors with the largest eigenvalues.

2.5 Transform the data

The original data can be transformed into a new set of principal components by multiplying the data by the matrix of eigenvectors. The transformed data is a new dataset with the same number of rows as the original data, but with fewer columns (the number of principal components).

2.6 Interpret the results

The principal components can be used to summarize the information in the original dataset in a more compact and interpretable form. The first principal component explains the most variance in the data, the second principal component explains the second most variance, and so on. The principal components can be plotted to visualize the patterns in the data and to identify clusters or trends.

3 PCA in MATLAB

There are several ways of finding the Principal Components of a dataset in MATLAB. In this analysis we will use the eig function, svd function and the pca function.

The dataset used in this analysis is the digits dataset which is a collection of images of handwritten digits (0-9) that is often used for machine learning and data analysis. It is included in the scikit-learn library in Python. The digits dataset consists of 1797 images, each of which is 8x8 pixels in size. Each image is represented by a 64-dimensional feature vector, with each feature representing the grayscale intensity of one pixel in the image. The dataset also includes a label for each image, indicating which digit it represents.

Before we perform any analysis on our data we need to first know what our data looks like.

```
%Let's view a few rows of the data
digits=csvread('digits.csv');
digits
```

digits = 1797×65

0	0	5	13	9	1	0	0	0	0	13	15	10	···
0	0	0	12	13	5	0	0	0	0	0	11	16	
0	0	0	4	15	12	0	0	0	0	3	16	15	
0	0	7	15	13	1	0	0	0	8	13	6	15	
0	0	0	1	11	0	0	0	0	0	0	7	8	
0	0	12	10	0	0	0	0	0	0	14	16	16	
0	0	0	12	13	0	0	0	0	0	5	16	8	
0	0	7	8	13	16	15	1	0	0	7	7	4	
0	0	9	14	8	1	0	0	0	0	12	14	14	
0	0	11	12	0	0	0	0	0	2	16	16	16	
⋮													

We can see that the data set has 1797 rows (an image per row) and 65 columns (a pixel value per column). The first 64 columns for a selected number represent eachpixels of the image and the last column is the label.

We then assign names to our variables.

```
% Assigning names to the measurements and target variable
% X represents the matrix of pixels of the images
X = digits(:,1:end-1);
% Y represents the vector of labels of the images
Y = digits(:,end);
```

Let's visualize a few images from X.

```
%ploting the first ten digits
for i = 1:10
    %create a subplot
    subplot(2, 5, i)
```

```
    %reshape the image
    rs=reshape(X(i,:), 8, 8);
    %rearrange the reshaped matrix
    imagesc(rs)
    colormap gray
    title(num2str(Y(i)))
    axis off
end
```

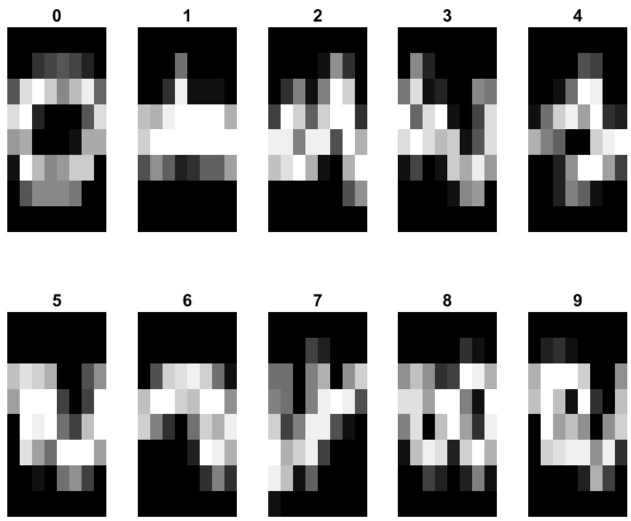

Figure 1: First Ten Numbers Plot 1

The images are not facing the right way so let's rearrange the reshaped matrix.

Note: this is just for visualization purposes and it is not a required step to perform PCA.

```
%ploting the first ten digits again
for i = 1:10
    %create a subplot
    subplot(2, 5, i)
    %reshape the image
    rs=reshape(X(i,:), 8, 8);
    %rearrange the reshaped matrix
    rsa=rs(:,end:-1:1);
    %show the image
    imagesc(imrotate(rsa,90))
    colormap gray
    title(num2str(Y(i)))
    axis off
end
```

Let's generate one final pot be for we move to the main thing. We can plot the first 3 columns of pixels against each other to see the distribution of all 1797 rows of digits.

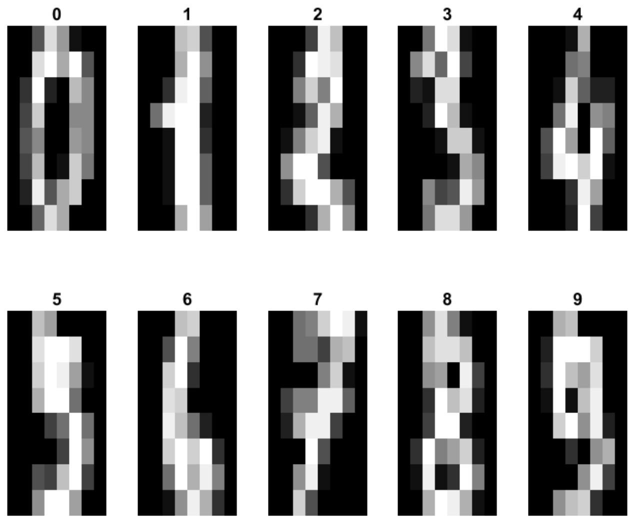

Figure 2: First Ten Numbers Plot 2

```
%this scatterPlot function takes in a table as an argument , so we
    need to covert
% the first 3 columns of matrix x into a table
Xt=array2table(X(:,1:4));
figure,sdo.scatterPlot(Xt)
end

```

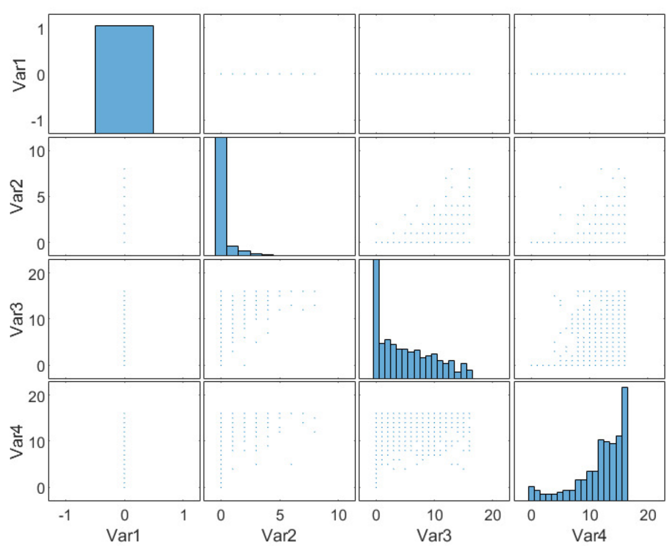

Figure 3: Scatter Plot of First Four Pixels

From Figure (3), there is no clear relationship between any pair of the 4 selected pixels. To see a clear relationship we need to include more pixels. However, If we want to plot the relationship between more than two pixels it gets a litter bit difficult. We can get away with three pixels by plotting in 3D. But, an issue arises when we want to visualize the relationship between four or more pixels at once. The digits dataset is large enough to explain and illustrate the usefulness of a PCA.

To verify that there are no clear relations, let us plot only the fourth pixel against the third pixel since there show more data points in Figure (3) above.

```
scatter(X(:,4), X(:,3),10,Y,"filled")
colormap("jet");
colorbar;
xlabel('X4'),ylabel("X3")
```

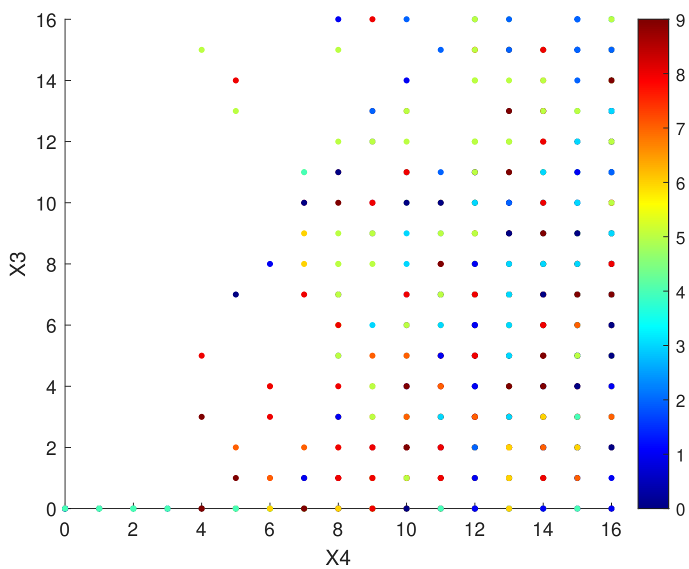

Figure 4: Scatter Plot of Fourth and Third Pixels

From Figure (4), each data point represents a number and there is no clear relationship that we can point out. Hence, we would use PCA to identify the pixels that would explain our data.

3.1 Performing PCA with the eig function

To demonstrate the concept of PCA we will follow the steps we discussed earlier.

```
%Stardardize the measurements in X
% Since the features of the dataset is on the same scale there is no
    need to standardize but if there is a need to standardize, use
    the following commented code
    % X_standard = zscore(X); % or
    % X_standard = (X-mean(X))./std(X);

%we only need to center the data
```

```
X_standard = X-mean(X);

%Compute the covariance matrix
cov_matrix = cov(X_standard);

% Compute the eigenvectors and eigenvalues
[eigenVectors, eigenValues]=eig(cov_matrix);

% Sort the eigenvalues and eigenvectors in descending order
[eigenValues, order]=sort(diag(eigenValues),'descend');
eigenVectors = eigenVectors(:,order);

% Select the number of components to keep
num_components=2;
eigenVectors_ = eigenVectors(:, 1:num_components);

%Project the data onto the principal components
projected_data = X_standard * eigenVectors_;

```

Let's make a scatter plot with the projected data.

```
scatter(projected_data(:,1), projected_data(:,2),10,Y,"filled")
colormap("jet");
colorbar;
xlabel('PC1'),ylabel("PC2")
title('EIG plot')
```

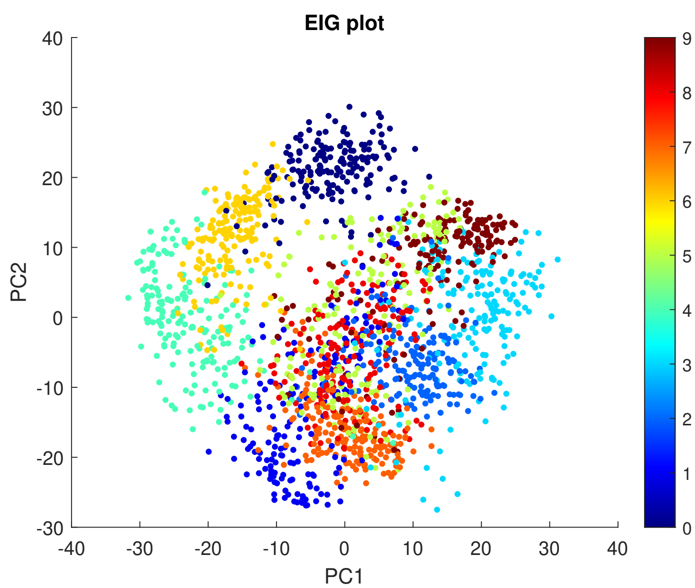

Figure 5: Scatter Plot of the First 2 Principal Components using the eig function

From Figure (5) we can see that the numbers that are alike (6 and 0) are close to each other and numbers that are not alike (4 and 3) are far apart. This is a better representation of our data than Figure (4).

Here I pick out the first 2 principal components for visualization purposes. To get the number of principal components that will capture the most variance, we will need a plot of the percentage of explained variance.

```
%calculate a cumulative sum of the sorted eigen values
cumsum_=cumsum(eigenValues);

%sum all the sorted eigen values
sum_=sum(eigenValues);

%calculate the percentages of variance expalined by the first n
    principal components
pvex= cumsum_./sum_;

%plot the percentage
figure,plot(pvex, 'k-o','Linewidth' ,2.5)
set(gca,'FontSize', 15), axis tight, grid on
set(gcf, 'Position',[1400 100 3*650 3*250])
```

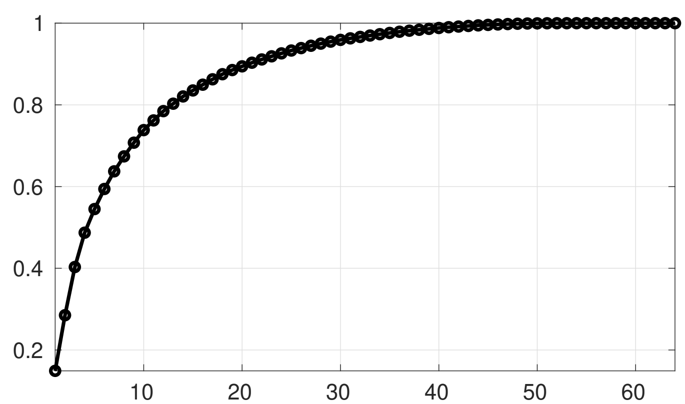

Figure 6: Scatter Plot of the First 2 Principal Components using the svd function

Let's assume we want to keep the first n components that explain at least 90% of the variance in the data. From Figure (4), the number of components to keep is 20.

Note: It is important to consider the trade-off between retaining as much of the variance in the data as possible and reducing the dimensionality of the data. In some cases, retaining a high percentage of the variance may not be necessary, and it may be more important to reduce the dimensionality for the sake of interpretability or computational efficiency.

3.2 Performing PCA with the svd function

```
%center the data
X_standard = X-mean(X);

```

```
%Compute the covariance matrix
cov_matrix = cov(X_standard);

% Compute the eigenvectors and eigenvalues
[U, Sigma, V]=svd(cov_matrix);

% Select the number of components to keep
num_components=2;
eigenVectors_ = V(:, 1:num_components);

%Project the data onto the principal components
projected_data = X_standard * eigenVectors_;
```

Let's make a scatter plot with the projected data.

```
scatter(projected_data(:,1), projected_data(:,2),10,Y,"filled")
colormap("jet");
colorbar;
xlabel('PC1'),ylabel("PC2")
title('SVD plot')
```

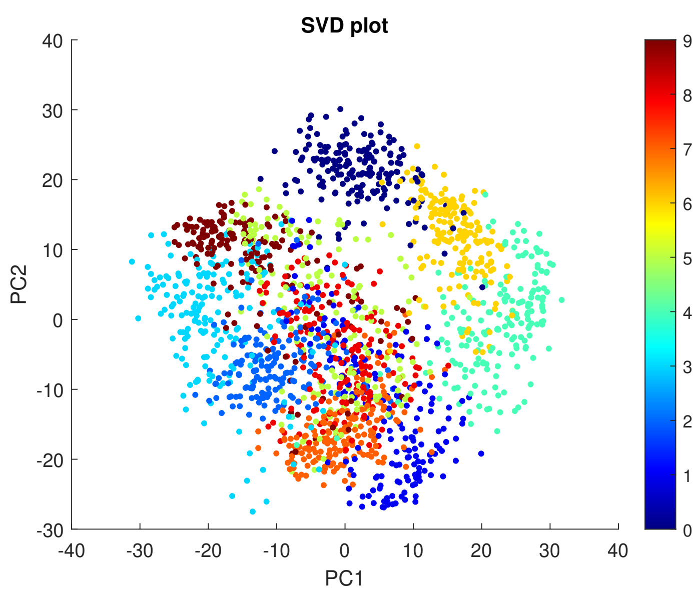

Figure 7: Scatter Plot of the First 2 Principal Components using the svd function

Figure (7) shows the same distribution of our data as Figure (5). The only difference is the PC1 of the svd function is the additive inverse of PC1 of the eig function.

To make a plot of the explained variance under the svd function, use the following code.

```
%calculate a cumulative sum of the sorted eigen values
cumsum_=cumsum(diag(Sigma));

%sum all the sorted eigen values
sum_=sum(diag(Sigma));
```

```

%calculate the percentages of variance expalined by the first n
    principal components
pvex= cumsum_./sum_;

%plot the percentage
figure,plot(pvex, 'k-o','Linewidth' ,2.5)
set(gca,'FontSize', 15), axis tight, grid on
set(gcf, 'Position',[1400 100 3*650 3*250])
```

3.3 Performing PCA with the pca function

```
% Perform PCA
[coeff, score, latent]=pca(X);

% Select the number of components to keep
num_components=2;

% Project the data onto the principal components
% since we are using the pca function, the projected data is just
    the
% selected number of components of our score
projected_data = score(:,1:num_components);
```

Let's make a scatter plot with the projected data.

```
scatter(projected_data(:,1), projected_data(:,2),10,Y,"filled")
colormap("jet");
colorbar;
xlabel('PC1'),ylabel("PC2")
title('PCA plot')
```

Figure (8) shows the same distribution as our data as Figure (5). The only difference is both PC1 and PC2 of the pca function are the additive inverse of PC1 and PC2 of eig function respectively. However, this is not always true.

To make a plot of the explained variance under the pca function, use the following code.

```
%calculate a cumulative sum of the sorted eigen values
cumsum_=cumsum(latent);

%sum all the sorted eigen values
sum_=sum(latent);

%calculate the percentages of variance expalined by the first n
    principal components
pvex= cumsum_./sum_;

%plot the percentage
figure,plot(pvex, 'k-o','Linewidth' ,2.5)
set(gca,'FontSize', 15), axis tight, grid on
set(gcf, 'Position',[1400 100 3*650 3*250])
```

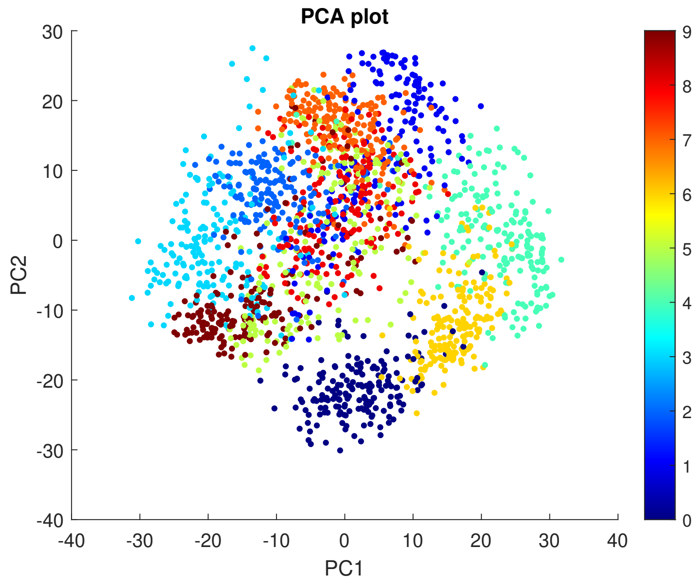

Figure 8: Scatter Plot of the First 2 Principal Components using the pca function

4 Conclusion

PCA is very important and understanding how it is been implemented is crucial. The live Script of this discussion can be downloaded from my PCA-MATLAB GitHub Repository. Good Luck!