



University Of Ibn-Khaldoun Tiaret

Report of Practical Work

# Principal Component Analysis Algorithm

Option: Master 1 Year SE

Course: Algorithm Complexity

**Remise Le 12/12/2018**

**Presented By:**

Mr. Younes Charfaoui

Mr. Bourbai Ismail

**Subject Responsible:**

Mr. Chenine Abdelkader

**Year of 2018-2019**

# Chapter I

**Problem:** Extracting Principal Components for a given dataset.

## **The Problem:**

In machine learning the problem is that the data is in a  $n$  dimension and this not a problem for the mathematics, it's a problem for human that can't see this dimension, and also a problem for the machines, since we are in  $n$  dimension means that more computation are going to happen, the Principal Component Analysis (PCA) comes here to play the role of reducing the dimensionality of the data so we can visualize it in 2D or 3D, and the machine going to do less computation for finding the best machine learning model.

Principal component analysis is a dimensionality reduction technique; it lets you distill multi-dimensional data down to fewer dimensions, selecting new dimensions that preserve variance in the data as best it can.

**The way the algorithm work is like following:**

- Standardize the data (By default it's the responsibility of the machine learning engineer).
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix.
- Sort eigenvalues in descending order and choose the  $k$  eigenvectors that correspond to the  $k$  largest eigenvalues where  $k$  the number of dimensions of the new feature subspace is ( $k \leq d$ ).
- Construct the projection matrix  $W$  from the selected  $k$  eigenvectors.
- Transform the original dataset  $X$  via  $W$  to obtain a  $k$ -dimensional feature subspace  $Y$ .

## **Theory behind PCA:**

**Standardizing the Data:** Most of the times, our dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Euclidian distance between two data points in their computations, this is a problem. If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units, 5kg and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

Whether to standardize the data prior to a PCA on the covariance matrix depends on the measurement scales of the original features. Since PCA yields a feature subspace that maximizes the variance along the axes, it makes sense to standardize the data, especially, if it was measured on different scales.

**Computing Eigenvectors and Eigenvalues:** The eigenvectors and eigenvalues of a covariance matrix represent the "core" of a PCA: The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes.

Covariance Matrix can be calculated as following:

$$\frac{1}{n-1} \sum_{i=1}^n (X - \bar{x})(Y - \bar{y})$$

The Eigen value and Eigen vector of a matrix can be calculated as following:

$$(A - \lambda I)X = 0$$

Where  $\lambda$  is the Eigenvalue and  $X$  is The Eigenvector and the preceding equation means that:

$$\det(A - \lambda I) = 0$$

**Sort eigenvalues:** The typical goal of a PCA is to reduce the dimensionality of the original feature space by projecting it onto a smaller subspace, where the eigenvectors will form the axes. In order to decide which eigenvector(s) can be dropped without losing too much information for the construction of lower-dimensional subspace, we need to inspect the corresponding eigenvalues: The eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data; those are the ones that can be dropped. In order to do so, the common approach is to rank the eigenvalues from highest to lowest in order to choose the top  $k$  eigenvectors.

**Construct the projection matrix:** It's about time to get to the really interesting part: The construction of the projection matrix that will be used to transform the dataset onto the new feature subspace. Although, the name "projection matrix" has a nice ring to it, it is basically just a matrix of our concatenated top  $k$  eigenvectors.

**Transform the original dataset:** In this last step we will use the  $d * k$ -dimensional projection matrix  $W$  to transform our samples onto the new subspace via the equation  $Y = X \times W$ , where  $Y$  is matrix of our transformed samples.

# Chapter II

## Algorithm Description

### Specification:

- Inputs:
  - **Number components:** An Integer Number of Components or dimensions the user want to perceive in the new data
  - **Data:** The Data Matrix of Numbers the user want to reduce its dimensionality.
  - **Scaling:** A Boolean value indicates weather to standardize the data or not.
- Output:
  - The Data Matrix transformed to new dimensionality.

### Program:

```
1. import numpy as np
2. from sklearn.preprocessing import StandardScaler
3.
4. def pca(number_components, data , is_scaled = True):
5.     if not is_scaled:
6.         scaler = StandardScaler()
7.         data = scaler.fit_transform(data)
8.
9.         # calculate the covariance matrix
10.        cov_matrix = np.cov(data.T)
11.
12.        # calculate the eigen things
13.        eig_vals , eig_vect = np.linalg.eig(cov_matrix)
14.
15.        # codes are the same
16.        eig_pairs = [(np.abs(eig_vals[i]) , eig_vect[:,i]) for i in range(len(eig_vals))]
17.
18.        # Sorting All of Them
19.        eig_pairs.sort(key = lambda x : x[0] , reverse= True)
20.
21.        # Choosing The Highest Ones
22.        final = []
23.        for i in range(number_components):
24.            final.append(eig_pairs[i][1].reshape(data.shape[1],1))
25.
```

```
26.     # Creating the Projection Matrix
27.     projection_matrix = np.hstack((final))
28.
29.     # transforming the data
30.     Y = data.dot (projection_matrix)
31.
32.     return Y
```

## Tools Used:

During this practical work we have used:

- Python as a programming language to implements PCA Algorithm.
- Numpy library for doing math calculations like matrix multiplication, covariance matrix calculations and so one, this library it's highly optimized for math operations.
- Matplotlib library for visualizing the data and graphs (plotting in general).
- Sklearn library, from which we used Standard Scaler for scaling our data in a proper range, and The PCA Algorithm to compare our result to it's.
- Pandas library which is in the background of this algorithm, it help us load the data from different source (CSV, TSV, Excel) and convert it into numpy matrix we can us in our algorithm.

## Comparing to Already Implemented PCA: