Principal Component Analysis (PCA)

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

Importantly, the dataset on which PCA technique is to be used must be scaled. The results are also sensitive to the relative scaling. As a layman, it is a method of summarizing data. Imagine some wine bottles on a dining table. Each wine is described by its attributes like colour, strength, age, etc. But redundancy will arise because many of them will measure related properties. So what PCA will do in this case is summarize each wine in the stock with less characteristics.

Intuitively, Principal Component Analysis can supply the user with a lower-dimensional picture, a projection or "shadow" of this object when viewed from its most informative viewpoint.

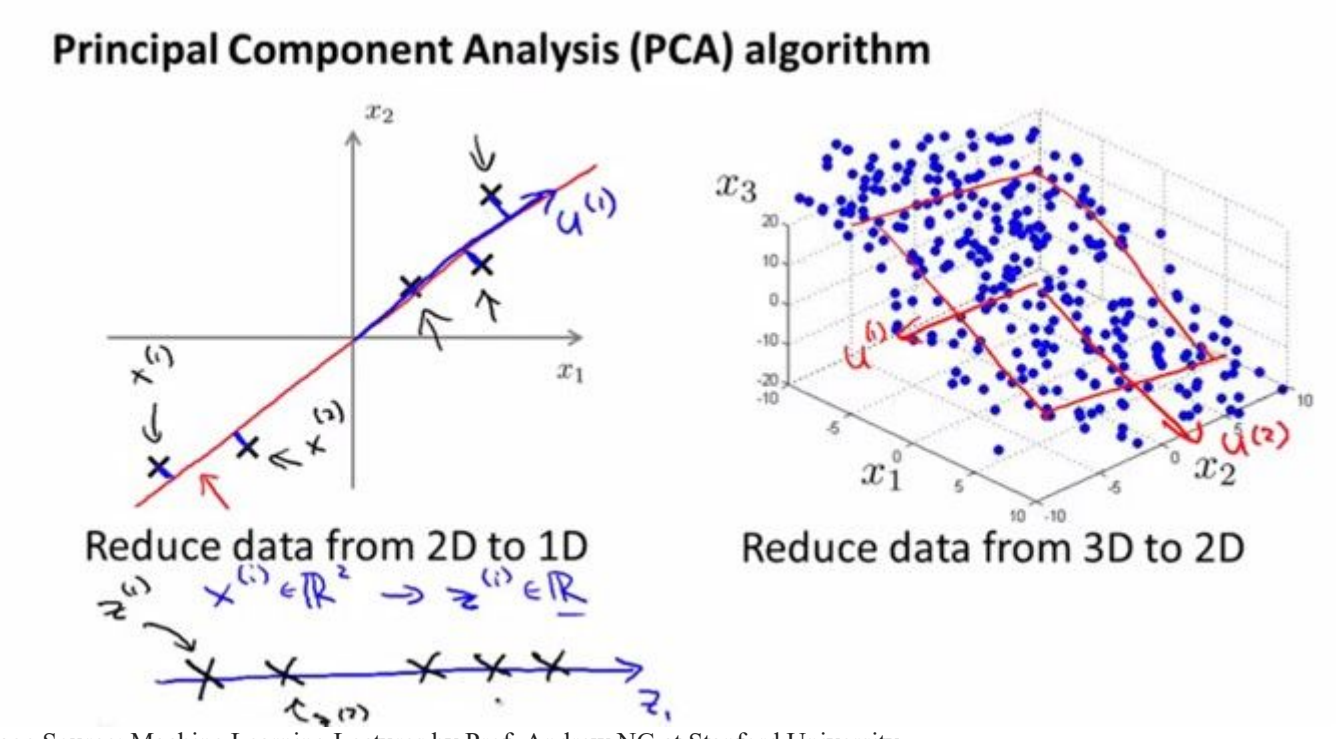

Image Source: Machine Learning Lectures by Prof. Andrew NG at Stanford University

- **Dimensionality**: It is the number of random variables in a dataset or simply the number of features, or rather more simply, the number of columns present in your dataset.

- **Correlation**: It shows how strongly two variable are related to each other. The value of the same ranges for –1 to +1. Positive indicates that when one variable increases, the other increases as well, while negative indicates the other decreases on increasing the former. And the modulus value of indicates the strength of relation.
- **Orthogonal**: Uncorrelated to each other, i.e., correlation between any pair of variables is 0.
- **Eigenvectors**: Eigenvectors and Eigenvalues are in itself a big domain, let's restrict ourselves to the knowledge of the same which we would require here. So, consider a non-zero vector v. It is an eigenvector of a square matrix A, if Av is a scalar multiple of v. Or simply:
  $Av = \lambda v$
  Here, v is the eigenvector and λ is the eigenvalue associated with it.
- **Covariance Matrix**: This matrix consists of the covariances between the pairs of variables. The (i,j)th element is the covariance between i–th and j–th variable.

Properties of Principal Component

Technically, a principal component can be defined as a linear combination of optimally-weighted observed variables. The output of PCA are these principal components, the number of which is less than or equal to the number of original variables. Less, in case, when we wish to discard or reduce the dimensions in our dataset. The PCs possess some useful properties which are listed below:

1. The PCs are essentially the linear combinations of the original variables, the weights vector in this combination is actually the eigenvector found which in turn satisfies the principle of least squares.
2. The PCs are orthogonal, as already discussed.
3. The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance.

The least important PCs are also sometimes useful in regression, outlier detection, etc.

Implementing PCA on a 2-D Dataset

Step 1: Normalize the data: First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become x– and all Y become y–. This produces a dataset whose mean is zero.

Step 2: Calculate the covariance matrix: Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

Please note that $Var[X_1] = Cov[X_1,X_1]$ and $Var[X_2] = Cov[X_2,X_2]$.

Step 3: Calculate the eigenvalues and eigenvectors: Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. λ is an eigenvalue for a matrix A if it is a solution of the characteristic equation:

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

Where, I is the identity matrix of the same dimension as A which is a required condition for the matrix subtraction as well in this case and '*det*' is the determinant of the matrix. For each eigenvalue λ, a corresponding eigen-vector v, can be found by solving:

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

Step 4: Choosing components and forming a feature vector: We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.
Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.

$$Feature\ Vector = (eig_1, eig_2)$$

Step 5: Forming Principal Components:

This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

$$NewData = FeatureVector^T \times ScaledData^T$$

Here,

NewData is the Matrix consisting of the principal components,

FeatureVector is the matrix we formed using the eigenvectors we chose to keep, and

ScaledData is the scaled version of original dataset

('T' in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa. In particular, a 2x3 matrix has a transpose of size 3x2)

If we go back to the theory of eigenvalues and eigenvectors, we see that, essentially, eigenvectors provide us with information about the patterns in the data. In particular, in the running example of 2-D set, if we plot the eigenvectors on the scatterplot of data, we find that the principal eigenvector (corresponding to the largest eigenvalue) actually fits well with the data. The other one, being perpendicular to it, does not carry much information and hence, we are at not much loss when deprecating it, hence reducing the dimension.

All the eigenvectors of a matrix are perpendicular to each other. So, in PCA, what we do is represent or transform the original dataset using these orthogonal (perpendicular) eigenvectors instead of representing on normal *x* and *y* axes. We have now classified our data points as a combination of contributions from both *x* and *y*. The difference lies when we actually disregard one or many eigenvectors, hence, reducing the dimension of the dataset. Otherwise, in case, we take all the eigenvectors in account, we are just transforming the co-ordinates and hence, not serving the purpose.

Applications of Principal Component Analysis

PCA is predominantly used as a dimensionality reduction technique in domains like facial recognition, computer vision and image compression. It is also used for finding patterns in data of high dimension in the field of finance, data mining, bioinformatics, psychology, etc.

PCA for images

You must be wondering many a times how can a machine read images or do some calculations using just images and no numbers. We will try to answer a part of that now. For simplicity, we will be restricting our discussion to square images only. Any square image of size *NxN* pixels can be represented as a *NxN* matrix where each element is the intensity value of the image. (The image is formed placing the rows of pixels one after the other to form one single image.) So if you have a set of images, we can form a matrix out of these matrices, considering a row of pixels as a vector, we are ready to start principal component analysis on it.

Say we are given an image to recognize which is not a part of the previous set. The machine checks the differences between the to-be-recognized image and each of the principal components. It turns out that the process performs well if PCA is applied and the differences are taken from the 'transformed' matrix. Also, applying PCA gives us the liberty to leave out some of the components without losing out much information and thus reducing the complexity of the problem.

For image compression, on taking out less significant eigenvectors, we can actually decrease the size of the image for storage. But to mention, on reproducing the original image from this will lose out some information for obvious reasons.

Usage in programming

For application of PCA, you can hard-code the whole process in any programming language, be it C++, R, Python, etc. or directly use the libraries made available by contributors. However, it is recommended to hard-code in case the problem is not too complex so that you actually get to see what exactly is happening in the back-end when the analysis is being done and also understand the corner cases. Just for instance, in R, there are libraries called princomp, HSAUR, prcomp, etc. which can be used for direct application.

Referance:

https://www.dezyre.com/data-science-in-python-tutorial/principal-component-analysis-tutorial

https://en.wikipedia.org/wiki/Principal_component_analysis