Machine Learning: Unsupervised Learning — Principal Component Analysis

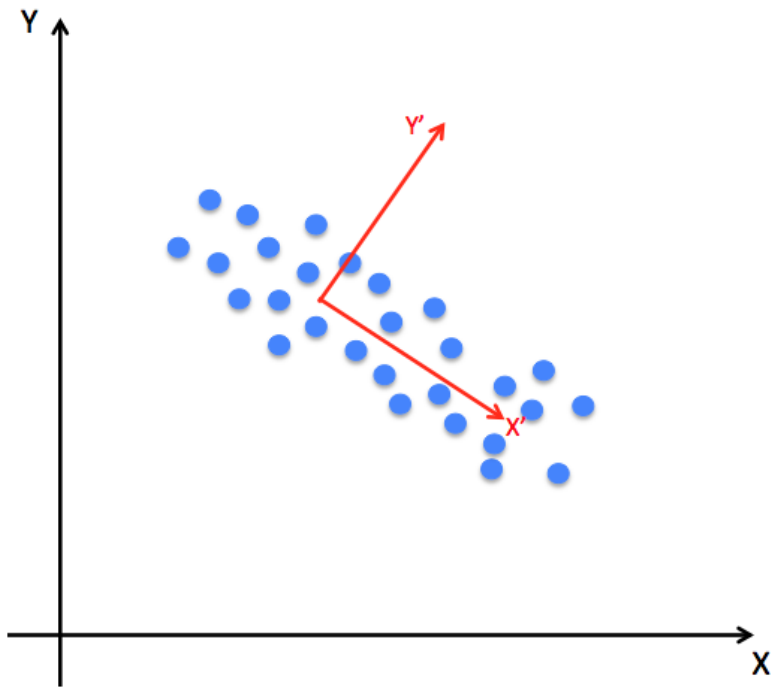
Michele Cavaioni Follow
Feb 7, 2017 · 4 min read

The Principal Component Analysis (PCA) is a method for feature selection that turns a set of correlated variables into the underlying set of orthogonal (latent) variables, that are not measured directly but are driving the phenomenon that we are measuring.

In a given dataset, the PCA method finds a new coordinate system that is obtained from the old one only by translation and rotation.

It first moves the center of the coordinate system to the center of the data, then it moves the x-axis into the principal axis of variation, which is the one where we see the most variation relative to all the data points.

Finally, it moves the other axis into an orthogonal, less important, direction of variation.



PCA provides vectors and not functions (contrary for example to linear regression), therefore any type of data sets can be represented by it.

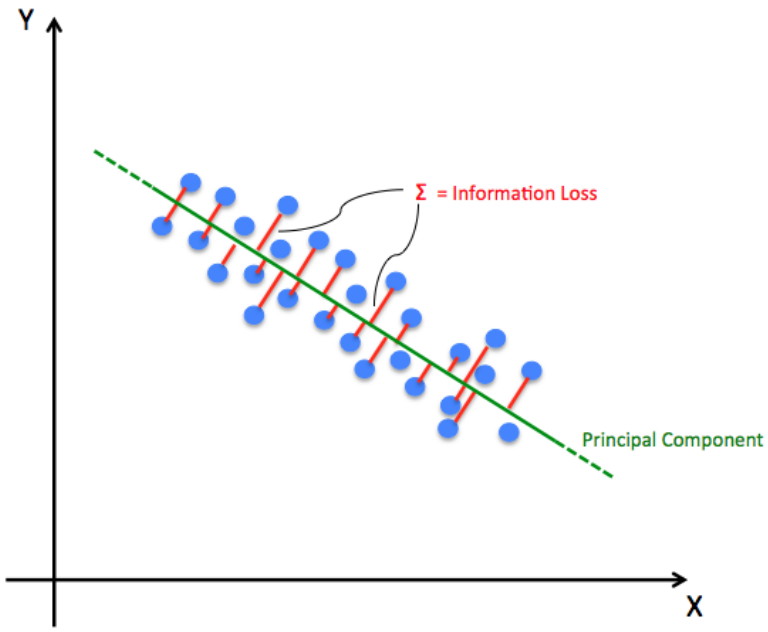
As mentioned earlier, when applying PCA in the context of dimensionality reduction, we basically want to select a smaller number of features that are driving the patterns. We detect the "composite features" (which are actually called "principal components") that more directly probe the underlying phenomenon.

So, let's introduce the concept of "***Principal Component***" (PC).

While with a regression we are trying to predict an output variable with respect to the value of an input variable, with a Principal Component we try to come up with a direction in the data (not to predict anything) where we can project the data onto, while loosing the minimum amount of information.

Once again, the Principal Component of a dataset is the **direction that has the largest variance**, (where variance is intended as the "spread" of a data distribution), because it can retain the maximum amount of information in the original data, minimizing the information loss.

The amount of information loss is the sum of the distance of each point to the principal component line.



Applying PCA to all the features allows us to derive the principal components involved.

PCA is a systematized way to transform input features into principal components, using them as new features.

These principal components are the directions in the data that maximize variance when we project the original data to them.

The more variance of the data along a principal component, the higher the principal component is ranked. Each principal component is orthogonal to each other, without overlapping, so they are a sort of independent features.

PCA helps visualize high-dimensional data, it reduces noise and finally makes other algorithms to work better because we are injecting fewer inputs.

A successful use of PCA is seen in facial recognition problem.

In this case the principal components that are the result of PCA are called ***eigenfaces***.

This method is very effective since picture of faces have high input dimensionality (many pixels) and faces have certain general patterns that can be captured in smaller number of dimensions (i.e. two eyes and the mouth in the bottom).

The dilemma is given by the selection of principal components. Unfortunately there is no written rule on how many of these should be, but a good practice is to train on different number of PCs and see how the accuracy responds. Then finally stop adding when it becomes apparent that adding more PCs doesn't buy much more discrimination.

It certainly is not good to perform feature selection before running PCA, as it defeats the purpose of having many features from which PCA combines information.

However, in a multi-class classification problem like this one, where we have more than 2 labels, accuracy is a less intuitive metric, while instead **F1-score** provides better explanation for the results.

The F1-score in fact, considers both the *precision* and the *recall* of the test to compute the score. The *precision* is the number of correct positive results divided by the number of all positive results; the *recall* instead is the number of correct positive results divided by the number of positive results that should have been returned.

Precision = TruePositive/(TruePositive+FalsePositive)

Recall = TruePositive/(TruePositive+FalseNegative)

The F1 score can be interpreted as a weighted average of the precision and recall.

Finally, one thing to note is that adding more PCs does not necessarily mean improving the F1-score, as it can deteriorate after a certain point.

. . .

This blog has been inspired by the lectures in the Udacity's Machine Learning Nanodegree. (http:www.udacity.com)