Principle Components Analysis (PCA) for Dimensionality Reduction

Robert B. Heckendorn
University of Idaho

February 23, 2022

Principle Components Analysis (PCA) is an unsupervised machine learning technique for dimensionality reduction.

Assume you have R samples of data each with C features in an $R \times C$ matrix called X. That is, each sample of C dimensional data is in a row. The matrix could represent any kind of data in which there are C features in each of R samples. This data could even be a matrix of gray levels of a picture where each row of gray level pixels is a point in C dimensional space. The data is usually correlated from one feature to other feature (column to another) but not necessarily adjacent. That is, the data usually occupies some much lower dimension. Somewhere in the data are the essential features. In fact, the essential features may be a linear combination of the features given. If they are, we can find these combinations by using Principle Component Analysis (PCA). In fact, PCA can tell us the relative contribution from various orthogonal components so we can concern ourselves only with the most salient features. Sometimes reducing the dimensionality can be used to eliminate noise and help focus an algorithm on learning the important parts of the data. This is unsupervised learning in that we do not have answer categories given for data. We are looking for the important orthogonal directions in the data.

1 The PCA Algorithm for Compression

Here is a recapitulation of the algorithm in the book but with more detail and some example data in detail. I will assume that some data such as the list of eigenvectors comes stored as **one eigenvector per row**. This is the way the vectors are presented in the Matrix Library. The algorithms can be adjusted to work with an assumption that eigenvectors are in columns, but that was not what I chose here because of the way C/C++ stores matrices. Be sure to check the code you are using to get the alignment of vectors correct.

1.1 Center the data.

This is done by computing the mean position of the R points in C dimensional space and then subtracting that mean from each row.

$$X'[r] = X[r] - MeanByCol(X) \qquad \forall r$$

Where $X[r]$ the the r^{th} row and Mean returns a vector of the means of the each of the columns.

If the data in different dimensions are on different scales like 0 to 10 inches vs 0 to 1000000 years then scaling with Z-score is advised:

$$X'[r] = (X[r] - MeanByCol(X))/StddevByCol(X) \qquad \forall r$$

This makes sure each dimension (column) is scaled within that dimension.

1.2 Compute covariance matrix.

Compute the covariance matrix M using the normalized data X':

$$M = (1/R)X'^{\mathsf{T}} \cdot X'$$

M should be a $C \times C$ matrix. **Beware that M is the covariance matrix only if X is centered about the mean** as we have done. Also, this is the **biased covariance because we divided by R.** The unbiased covariance divides by $(R-1)$. So know which your covariance routine is producing. Read the fine print!

1.3 Compute eigenvalues and eigenvectors of M

The covariance matrix is symmetric and so the eigenvalues W_i are all real numbers. Eigenvalues, W_i, and eigenvectors V_i, come out as pairs, one eigenvalue for each eigenvector and are often computed together in one routine. The eigenvectors indicate the direction of the variation. The eigenvalues indicate the amount of variation. When calling an eigenvector routine know if the eigenvectors are normalized or not. Their real purpose is to give a direction so they are often normalized. Also know if they are sorted by magnitude of eigenvalue. We will be concerned with the largest eigenvalues.

$$V = eigenvector(M) \qquad W = eigenvalues(M)$$

$$V_i \in \mathbb{R}^C \text{ and } W_i \in \mathbb{R}$$

So V is a $C \times C$ matrix which represents C vectors of C columns. I will assume the eigenvectors are stored as rows in the matrix.

1.4 Normalize the eigenvectors

The magnitude of each eigenvector should be 1. This can be done without considering the eigenvalues because an eigenvector is only a direction and magnitude is not important. If they are not normalized for some reason you can normalize them.

$$V'[r] = Normalize(V[r]) \qquad \forall r$$

1.5 Sort eigenvectors by eigenvalue

We will consider only the eigenvectors with the K largest eigenvalues in magnitude. We can do this by sorting the eigenvalue/eigenvector pairs by eigenvalue magnitude. They may already be sorted. Check your documentation. Then taking the K largest. K is often chosen so the sum of the largest K eigenvalues contain most of the sum of the eigenvalues and the remaining eigenvalues are of little value.

$$\widehat{V} = Max_k(W, V')$$

$\widehat{V}$ is now a $K \times C$ matrix.

We now have a reduced set of eigenvectors to reduce the dimension of X. We have done this by selecting the K largest eigenvalues and their associated vectors and ignoring the rest. This makes $\widehat{V}$ a matrix of K row vectors.

1.6 Translate the normalized data

$$X'' = X' \cdot \widehat{V}^{\top}$$

The resulting matrix X'' is $R \times K$ in size rather than $R \times C$. It is smaller! It is this reduced dimension matrix X'' that we could apply our machine learning algorithms to and see if they work better. This approach has been used in facial recognition for instance.

1.7 Recovering Data from Compressed Data

Machine Learning doesn't require you recover the image from its compressed form. It is generally just used as a first step in classifying. But to show how the compressed form is capturing key attributes of the original data we can recover the image, but it requires: the compressed image X'', the selected eigen vectors $\widehat{V}$, and the scaling information $StddevByCol(X)$ and $MeanByCol(X)$.

First rotate the data back using the reduced eigenvector matrix:

$$X^* = X'' \cdot \widehat{V}$$

This works because $\widehat{V}$ is a rotation operation and the inverse of a rotation is the transpose: $\widehat{V}^{\top}$. Since X'' is $R \times K$ and $\widehat{V}$ is $K \times C$ matrix X^* is $R \times C$.

Then move the data back from where it was centered to its old position: Note the $MeanByCol(X)$ is the original mean.

$$X^*[r] + MeanByCol(X) \qquad \forall r$$

Or if Z-score:

$$(X^*[r] * StddevByCol(X)) + MeanByCol(X) \qquad \forall r$$

The result should now be $R \times C$ matrix. But it is not exactly the same because $K < C$ and this causes some data to be lost, but because we kept the K largest eigenvalues we kept most of the sources of variance in out data. Note that if $K = C$ then we should get the exact data back, ignoring tiny errors in the arithmetic.

1.8 The Component Matrix

Component analysis tells you the "weight" of each of the original dimension's involvement in the new axes. In this case multiply each of the eigenvectors by the square root of the corresponding eigenvalue. The Matrix of scaled eigenvectors is sometimes called the Component Matrix.

$$V_i \sqrt{W_i} \quad \forall i$$

Values in each vector that are near 1 are strong influences and values near 0 are weak influences.