Principle Components Analysis for Dimensionality Reduction

Robert B. Heckendorn
University of Idaho

March 3, 2017

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 in some from one feature to some 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.

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**. 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.

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

Where $X[r]$ the the r^{th} row and Mean returns the mean of all the rows.

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.

$$X'[r] = (X[r] - Mean(X))/Stddev(X) \qquad \forall r$$

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'.

$$M = (1/R){X'}^{\intercal} \cdot X'$$

M should be a $C \times C$ matrix. Beware that this is the biased covariance because we divided by R. The unbiased covariance divides by $(R-1)$. So know what your covariance routine is producing!

1.3 Compute eigenvalues and eigenvectors of M

The covariance matrix is symmetric and so the eigenvalues are all real numbers. Eigenvalues and eigenvectors 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.

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

1.4 Normalize the eigenvectors

The length of each eigenvector should be 1. This can be done without considering the eigenvalues because an eigenvector is really 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.

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

1.6 Translate the normalized data

We now use the reduced set of eigenvectors to reduce the dimension of X. We do this by selecting the k largest eigenvalues and their associated vectors and ignoring the rest. This makes a matrix of k vectors.

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

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

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

Rotate the data back using the reduced eigenvector matrix:

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

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

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

or if Z-score

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

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 strongly influences and values near 0 are weak influences.

2 Mathematica Code

Here is the Mathematica Code from class:

```
(* get data in g.   #Rows = The number of samples.  #Cols = dimension of data *)

(* These are commands are operating on vectors of dimension #Cols *)
mx = Mean[g];      (* mean of the row vectors size=NumCols *)
sdx = StandardDeviation[g];

(* OPTIONAL: if data needs to be scaled to similar size use Z-score *)
gn = Map[(# - mx)/sdx &, g];

cc =  Transpose[gn].gn/Length[gn];      (* cc <-  get *biased* covariance between columns *)

(* give numeric answer to what are the eigenvectors *)
v = N[Eigenvectors[cc]];   (* should return normalized eigenvectors, one per row*)

(* OPTIONAL: if eigenvectors are not normalized do so here *)
```

```
evecs =  Map[Normalize, v];   (* evecs <- get normalized eigenvectors *)

(* get numeric version of eigen values *)
evalue = N[Eigenvalues[cc]]; (* evalue <- eigenvalues *)

(* strip the evecs matris to just the number of dimensions you think are important *)
newEvecs = Take[evecs, numDimensions];   (* take the first numDimensions rows *)

(* use the smaller set of vectors to select most important parts of picture *)
newImage = gn.Transpose[newEvecs];       (* take data and compress it *)

(* recover the image from the compressed newImage *)
recoveredImage = newImage.newEvecs;

(* OPTIONAL: restore picture if converted to Z score first *)
z = Map[#*sdx + mx &, recoveredImage ];
```

3 Example run

An example based an a simple 5×3 pixel "picture".

```
Read in a picture (size of Pic: 5 X 3)
 101.00000  103.00000  107.00000
 109.00000   11.00000   13.00000
  17.00000   19.00000   23.00000
  29.00000   31.00000   37.00000
  41.00000   43.00000   47.00000
Mean vector of the data points (size: 1 X 3)
  59.40000   41.40000   45.40000
Covariance Matrix (size: 3 X 3)
1450.24000  458.24000  426.24000
 458.24000 1066.24000 1074.24000
 426.24000 1074.24000 1083.84000
EigenValues (size: 1 X 3)
2516.22714 1083.82928    0.26359
EigenVectors in rows (size: 3 X 3)
   0.50606    0.61096    0.60879
  -0.86227    0.34213    0.37342
  -0.01986    0.71391   -0.69995
Encoded Pic (size: 5 X 3)
  96.18896    8.20753    0.03397
 -13.19726  -65.26800   -0.00967
 -48.77955   20.53182    0.52930
 -26.85218   19.51805   -0.94137
  -7.35997   17.01060    0.38777
Recovered pic (size: 5 X 3)
 101.00000  103.00000  107.00000
```

```
 109.00000   11.00000   13.00000
  17.00000   19.00000   23.00000
  29.00000   31.00000   37.00000
  41.00000   43.00000   47.00000
Compressed pic using only 2 dimensions (size: 5 X 2)
  96.18896    8.20753
 -13.19726  -65.26800
 -48.77955   20.53182
 -26.85218   19.51805
  -7.35997   17.01060
Recovered compressed pic (size: 5 X 3)
 101.00067  102.97575  107.02378
 108.99981   11.00690   12.99323
  17.01051   18.62213   23.37048
  28.98130   31.67206   36.34109
  41.00770   42.72316   47.27142
Compressed pic using only 1 dimension (size: 5 X 1)
  96.18896
 -13.19726
 -48.77955
 -26.85218
  -7.35997
Recovered compressed pic (size: 5 X 3)
 108.07776  100.16771  103.95892
  52.72134   33.33699   37.36564
  34.71443   11.59759   15.70348
  45.81108   24.99436   29.05265
  55.67538   36.90334   40.91932
```