Principal Components Analysis:

A How-To Manual for R

Emily Mankin

Introduction

Principal Components Analysis (PCA) is one of several statistical tools available for reducing the dimensionality of a data set. Its relative simplicity—both computational and in terms of understanding what's happening—make it a particularly popular tool. In this tutorial we will look at how PCA works, the assumptions required to use it, and what PCA can and cannot accomplish. Along the way, we will use the statistical coding language of R to develop a simple, but hopefully illustrative, model data set and then analyze it using PCA. The R syntax for all data, graphs, and analysis is provided (either in shaded boxes in the text or in the caption of a figure), so that the reader may follow along.

Why Use Principal Components Analysis?

The major goal of principal components analysis is to reveal hidden structure in a data set. In so doing, we may be able to

- identify how different variables work together to create the dynamics of the system
- reduce the dimensionality of the data
- decrease redundancy in the data
- filter some of the noise in the data
- compress the data
- prepare the data for further analysis using other techniques

It's difficult to give a full description of when PCA is useful, because PCA has been used in countless statistical applications. To give you some idea, here is a small sampling of applications I've come across in the last year:

PCA has been used in both evaluating and pre-processing event-related potential data. (See for example Dien's paper, "Localization of the event-related potential novelty response as defined by principal components analysis.")

PCA has been used to determine how populations of neurons divide into sub-populations and work together. (See for example Briggman's paper, "Optical Imaging of Neuronal Populations During Decision-Making.")

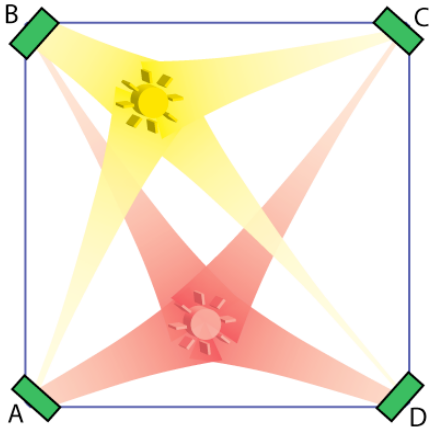


Figure 1. *Two light sources inside a box with recorders at the corner. The amount of light to reach a recorder is assumed to decay exponentially with the distance from the light source to the recorder, so that recorder A is influenced predominantly by the red light source, while recorder B is influenced predominantly by the yellow light source.*
We assume further that the amount of light recorded at each recorder is a simple sum of the amounts of light that reach it from each source.

PCA has been used to determine how risk factors combine to increase or decrease overall risk. (See for example Gu's paper, "Principal components analysis of morphological measures in the Quebec family study: Familial Correlations.")

PCA has been used in computerized face recognition. (See for example Turk's paper, "Eigenfaces for Recognition.")

A Model Data Set

We now turn to building a small model to help demonstrate what PCA is doing. Let's imagine we are given a black box, and we know that inside there are some number of light sources that emit some pattern of light over time (e.g. one could be blinking, one could be on a "dimmer" so that the intensity of light it emits varies up and down continuously, etc). We can't open the box to see the light sources, but we have the capacity to record through sensors on the edges of the box. (See Figure 1.)

Our goal is to use the data collected from the sensors to reconstruct what's happening in the box. How many light sources are in the box? Where are they located? What pattern of light does each emit? Though this is, perhaps, a contrived example, it is not entirely unlike the situation of interpreting EEG data, in which localized brain activity plays the role of the light sources, the skull the black box, and the electrodes the recorders.

We will look at how PCA helps us answer—or not—the questions above.

To begin, let's assume we can control the contents of the black box so that we can see what our data will look like. Suppose we have four recorders, one at each corner of the box. We introduce coordinates onto the box so that we have A at location (0,0), B at (0,1), C at (1,1), and D at (1,0). For now, we'll assume we have only two light sources, the first is at location (.3,.8) and its light intensity varies in a sine-wave pattern; the second is at location (.5,.2) and it varies in a cosine-wave pattern with a different period.

```
recorders = data.frame("X"=c(0,0,1,1), "Y" = c(0,1,1,0),
     row.names=c("A", "B","C","D"))

locs = data.frame("X"=c(.3,.5),"Y"=c(.8,.2))

intensities = data.frame("sine"=sin(0:99*(pi/10))+1.2,
     "cosine"= .7*cos(0:99*(pi/15))+.9)
```

We assume that the amount of light each recorder picks up decays exponentially with the distance from the light source to the recorder. We also assume that the lights combine **linearly**, that is, the sensors record a simple sum of the amount of light they receive from each source, though probably in a noisy way.

```
dists = matrix(nrow=dim(locs)[1], ncol=dim(recorders)[1],
     dimnames=list(NULL, row.names(recorders)))

for (i in 1:dim(dists)[2]){
     dists[,i]=sqrt((locs$X-recorders$X[i])^2
                    + (locs$Y-recorders$Y[i])^2)}

set.seed(500)

recorded.data = data.frame(jitter(as.matrix(intensities)%*%
     as.matrix(exp(-2*dists)),amount=0))
```

Let's look at the data we've collected. A scatter plot of the data (Figure 2a) shows that there is fairly strong correlation among all pairs of variables. This shouldn't be surprising, since we know that the four recorders are recording essentially the same data, just with different weights. We confirm this by checking the correlation matrix (see Figure 2b). Note that, while all the correlations are strong, the correlation between A and D is the strongest, followed closely by the correlation between B and C. Looking at Figure 1 should make the reason for this clear: A and D both receive a lot of input from the red light (at (.5, .2)) and are less impacted by the yellow light (at (.3, .8)), whereas B and C have just the opposite pattern. The recording from D is essentially a copy of the recording from A, making our data redundant. This should help you understand…

PCA Principle 1: In general **high correlation** between variables is a telltale sign of **high redundancy** in the data.

What about noise in the data? An assumption of PCA is that we have a reasonably high signal to noise ratio. In our case we do, because the high amplitude wave is the (important) signal; the small squiggles on top of the wave are the (to be discarded) noise. Note that "high amplitude" (our indicator for importance) really means "large variance"

and “small squiggle” (our indicator for irrelevance) is an informal synonym for “small variance.” This leads us to…

PCA Principle 2: The **most important** dynamics are the ones with the **largest variance**.

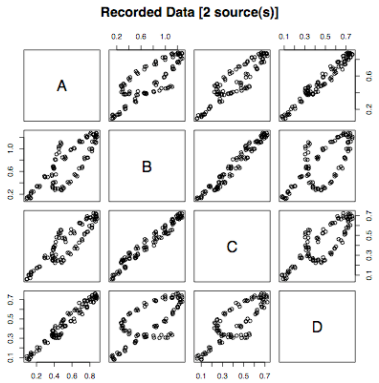
Before working with PCA you will want to ask yourself whether you believe this principle for your data. For many data sets it’s fine, but it’s worth thinking about before you throw away the small variance components. Nonetheless, we’ll go with it and plow right on.

How Does PCA Work?

This section is, in essence, a summary of Jonathon Shlens’ paper “A Tutorial on Principal Components Analysis.” If you want to understand in detail how PCA works, I highly recommend that you read his paper, which is quite clearly written and not very long.

We have now established the two principles on which PCA is based. The plan for PCA is to take our data and rewrite it in terms of new variables so that our “new data” has all the information from the original data but the redundancy has been removed and it has been organized such that the most important variables are listed first. How can we remove redundancy? Since high correlation is a mark of high redundancy, the new data should have low, or even better, zero correlation between pairs of distinct variables. To sort the new variables in terms of importance, we will list them in descending order of variance.

Figure 2

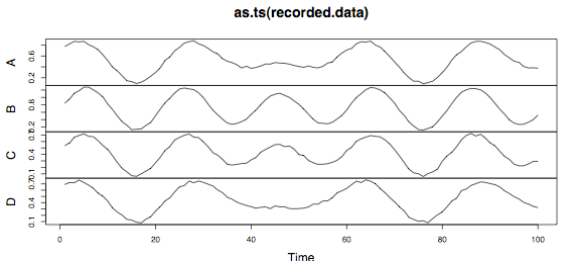


a. A scatter plot of the data reveals strong correlations among variables.
`plot(recorded.data)`

b. The correlation matrix.
`round(cor(recorded.data),2)`

```
   A    B    C    D
A 1.00 0.82 0.91 0.98
B 0.82 1.00 0.98 0.71
C 0.91 0.98 1.00 0.83
D 0.98 0.71 0.83 1.00
```

c. A time series plot of the data as recorded at each sensor. `plot.ts(recorded.data)`



Let's make this precise. Suppose we have a matrix of data, X*. Our goal is to find a matrix P such that the covariance matrix of PX is diagonal and the entries on the diagonal are in descending order. If we do this, PX will be our new data; the new variables will be linear combinations of the original variables whose weights are given by P. Because the covariance matrix is diagonal, we know that the covariance (and hence correlation) of any pair of distinct variables is zero, and the variance of each of our new variables is listed along the diagonal.

To see what we should choose for P, we'll need a few results from linear algebra that I state without proof. See Shlens' paper for details.

1. The covariance matrix of an $m \times n$ matrix, X, is an $n \times n$ symmetric matrix given by $\frac{1}{n-1}XX^T$.
2. Any symmetric $n \times n$ matrix A has a set of n orthonormal eigenvectors $(v_1, v_2, ..., v_n)$ and associated eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_n)$. This means that
   (a) For any i, $Av_i = \lambda_i v_i$
   (b) $\|v_i\| = 1$
   (c) $\langle v_i, v_j \rangle = 0$ if $i \neq j$
3. If A is an $n \times n$ symmetric matrix and E is an $n \times n$ matrix whose i^{th} column is the i^{th} eigenvector of A (where the eigenvectors are ordered in terms of decreasing value of their associated eigenvalue), then there is a diagonal matrix, D, such that $A = EDE^T$.
4. If the rows of a matrix E are orthogonal, then $E^{-1} = E^T$.

We now show how we can choose P so that the covariance matrix for PX is diagonal. Let $A=XX^T$. Let P be the matrix whose i^{th} row is the i^{th} eigenvector of A (in the notation of #3 above, this means $P=E^T$). Let's check that this gives us what we want:

$$
\begin{aligned}
Cov(PX) &= \tfrac{1}{n-1}(PX)(PX)^T \\
&= \tfrac{1}{n-1}PXX^TP^T \\
&= \tfrac{1}{n-1}PAP^T && A = XX^T \\
&= \tfrac{1}{n-1}P(EDE^T)P^T && \text{A can be rewritten this way. (See 3)} \\
&= \tfrac{1}{n-1}P(P^TDP)P^T && P = E^T \\
&= \tfrac{1}{n-1}PP^{-1}DPP^{-1} && \text{This follows from \#2, the definition of P, and \#4.} \\
&= \tfrac{1}{n-1}D
\end{aligned}
$$

---

* X must be in a specific form: each row of X should be a list of observations of one variable; each column, hence, is a single observation of all the variables. In our case, X would have 4 rows of 100 columns. Note, however, when applying the pre-installed R functions prcomp() and princomp(), it is expected that the columns are the variables. In the derivation above, X is also assumed to be in "centered" form; that is, the mean of each row is zero.

Indeed! Our goal was to find a matrix P such that the covariance matrix of PX is diagonal, and we've achieved that. Though it's not immediately clear from the derivation, by ordering the rows of P in decreasing order of associated eigenvalue, our second goal of having the variances in decreasing order is also achieved.

So that's it. To do PCA, all you have to do is follow these steps:

1. Get X in the proper form. This will probably mean subtracting off the means of each row. If the variances are significantly different in your data, you may also wish to scale each row by dividing by its standard deviation to give the rows a uniform variance of 1 (the subtleties of how this affects your analysis are beyond the scope of this paper, but in general, if you have significantly different scales in your data it's probably a good idea).
2. Calculate $A=XX^T$
3. Find the eigenvectors of A and stack them to make the matrix P.
4. Your new data is PX, the new variables (a.k.a. principal components) are the rows of P.
5. The variance for each principal component can be read off the diagonal of the covariance matrix.

Performing PCA in R

The Do It Yourself Method

It's not difficult to perform. Just follow the steps above.

```
# Obtain data in a matrix
Xoriginal=t(as.matrix(recorded.data))

# Center the data so that the mean of each row is 0
rm=rowMeans(Xoriginal)
X=Xoriginal-matrix(rep(rm, dim(X)[2]), nrow=dim(X)[1])

# Calculate P
A=X %*% t(X)
E=eigen(A,TRUE)
P=t(E$vectors)

# Find the new data and standard deviations of the principal components
newdata = P %*% X
sdev = sqrt(diag((1/(dim(X)[2]-1)* P %*% A %*% t(P))))
```

Using Built-In R Functions

R also comes with two pre-installed functions for performing PCA: prcomp() and princomp(). The primary difference between the code for doing your own PCA analysis and these two functions is that these functions expect your data to be organized with variables in columns and observations in rows (this is in keeping with the typical structure of a data.frame). They also return the new data in this form, and the principal components are given in columns instead of rows.

prcomp() and princomp() are similar functions. princomp() performs PCA using eigenvectors as described above. prcomp() uses a similar but not identical technique known as singular value decomposition (SVD). According to R help, SVD has slightly better numerical accuracy, so prcomp() is generally the preferred function. princomp() will also fail if the number of variables is larger than the number of observations. Each function returns a list whose class is prcomp or princomp, respectively. They return essentially the same information, but they use different terminology.

What each returns is summarized in the table below*:

prcomp() name	**princomp() name**	**Do it Yourself**	**Interpretation**
sdev	sdev	`sdev`	Standard deviations of each column of the rotated data
rotation	loadings	`P [or t(P)]`	The principal components.
center	center	`rm`	What got subtracted off each row or column in order to have centered data.
scale	scale		A vector of scale factors used. (If no scaling is done, prcomp() returns FALSE.)
x	scores	`newdata [or t(newdata)]`	The rotated data
	n.obs		The number of observations of each variable
	call		The call to princomp() that created the object

Because princomp() doesn't provide any significant advantages over prcomp(), we will focus only on using prcomp() for the remainder of this manual.

There are a few options that can be set when calling prcomp(). In the next section we will explore the most important option, tol. It is worth mentioning the options center and scale here. These are logical options specifying whether the data should be centered (so

* In the Do it Yourself column, taking the transpose when indicated in brackets gives an answer in the same form as is returned by prcomp() and princomp().

```
pr=prcomp(recorded.data)
pr
plot(pr)
barplot(pr$sdev/pr$sdev[1])
pr2=prcomp(recorded.data, tol=.1)
plot.ts(pr2$x)
quartz(); plot.ts(intensities)
quartz(); plot.ts(recorded.data)
quartz(); plot.ts(cbind(-1*pr2$x[,1],pr2$x[,2]))
```

each column has mean 0) or scaled (so each column has standard deviation 1) prior to performing PCA. The default for center is TRUE. The default for scale is FALSE.

What Do We Get from PCA?

Let's return to our model data set.

We start by naively applying prcomp() to the recorded data. Typing 'pr' prints the vector of standard deviations and the rotation matrix. In this case, the first principal component is -.44A - .73B - .39C - .35D.

We were hoping to be able to answer three questions about what was happening in the box. Let's see if we can.

First, we wanted to know how many different light sources there were. We start by using plot(pr). This applies the default plot method for objects of class pr, which plots the variances of each principal component against the component number. When deciding whether to reduce the dimensionality of a dataset, we look for a sharp drop off in the principal components. Here it looks like we should keep the first two components and discard the other two. That's great, since we know that that there were, in fact, two light sources. Instead of using the default plot method, I prefer to use the command barplot(pr$sdev/pr$sdev[1]), because it helps figure out how to drop the components I don't want. We can see the same precipitous drop-off in standard deviation as in the original graph, but now we can choose a tolerance that will force prcomp() to keep just the components we want. Anything between the height of the second and third bar will work; I choose 0.1.

Run prcomp() again, but this time include the option tol=0.1. What is returned will be any principal components whose standard deviation is greater than 10% of the standard deviation of the first principal component. In this case, the first two components are returned. Now let's examine our new data. Use plot.ts to plot the new data as a time series: plot.ts(pr2$x). In a separate window, plot the intensities, which we know make up the actual signal from inside the box, and in a third plot the recorded.data (See Figure 3). The bad news is that the new data is not a perfect replication of the actual signal sources. The good news is that it is better than the recorded data. In the recorded data, all four

sensors show five peaks, but in the actual signal, the sine-light has 5 peaks but the cosine light has only four. The data from the first principal component has five peaks and looks like a reflection of the sine-wave. The second principal component has only four peaks, so seems to be a better representation of the cosine-wave than any of the recorded data.

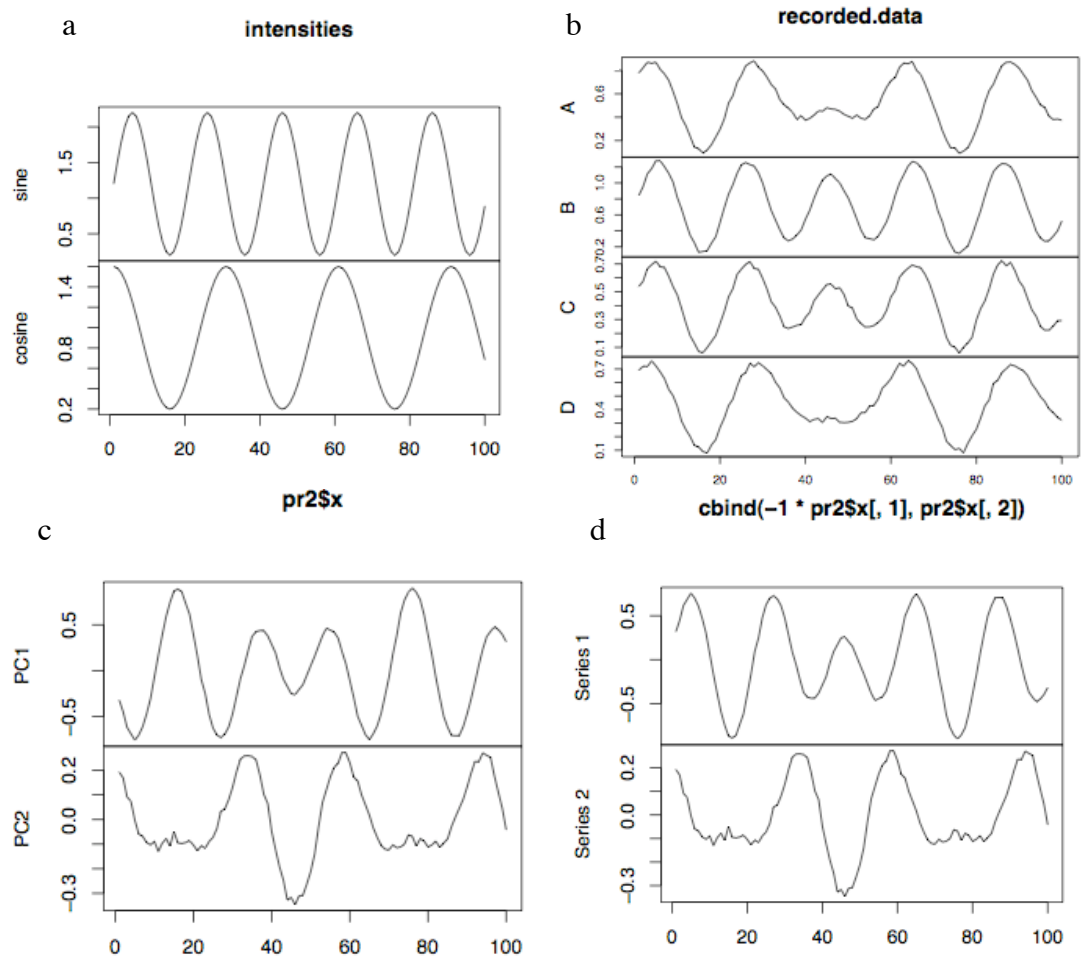


Figure 3. *Time series plots of a. the actual source pattern, b. the patterns recorded at each sensor, c. the first two principal components, d. the first two principal components, except PC1 has been replaced by its negative.*

What about the fact that PC1 appears to be upside down? This is an artifact of the fact that for any eigenvector of A, the negative of that eigenvector is also an eigenvector of A. There's no way to determine *a priori* which one to choose. Since we know what the actual signal was, we can guess that PC1 would be better if it were multiplied by negative one.

As for localizing the separate sources, it is beyond the scope of this paper to examine the efforts that have been made toward using PCA for source localization, but it is not obvious to me how PCA would achieve this.

```
od=pr$x %*% t(pr$rotation)
od2=pr2$x %*% t(pr2$rotation)
quartz(); plot.ts(recorded.data)
quartz(); plot.ts(od)
quartz(); plot.ts(od2)
```

On the data *analysis* side, it seems that the major strength of PCA is its ability to determine how many sources of variance are important and how different variables work together to produce those sources. Let's also spend a few moments considering how PCA can contribute to data *management* as well.

Going back to the derivation of PCA, we have N=PX, where N is our new data. Since we know that $P^{-1}=P^{T}$, it is easy to see that $X=P^{T}N$. Thus, if we know P and N, we can easily recover X. This is useful, because if we choose to throw away some of the smaller components—which hopefully are just noise anyway—N is a smaller dataset than X. But we can still reconstruct data that is almost the same as X.

In our example, we recorded data from four sensors, giving us 400 data points. But we also found that two principal components accounted for most of the variance. If we maintain only the rotated data from those two components and the rotation matrix P, that requires storing only 216 pieces of information, a 46% savings! When we reconstruct X, we won't get it back exactly as it was, but it will mostly be noise that we lost.

Because prcomp() works with variables in columns instead of rows as in the derivation above, the required transformation is $X=NP^{T}$, or in R syntax, X=pr$x %*% t(pr$rotation). Run the code at the top of the page. You can see that od, which is the reconstruction of X from when no principal components were discarded, is identical to the recorded.data. od2 is the reconstruction of X from only two principal components. It appears quite similar to the recorded.data, but smoothed out a little bit (see Figure 4). We will explore data reconstruction in more detail in the exercises.

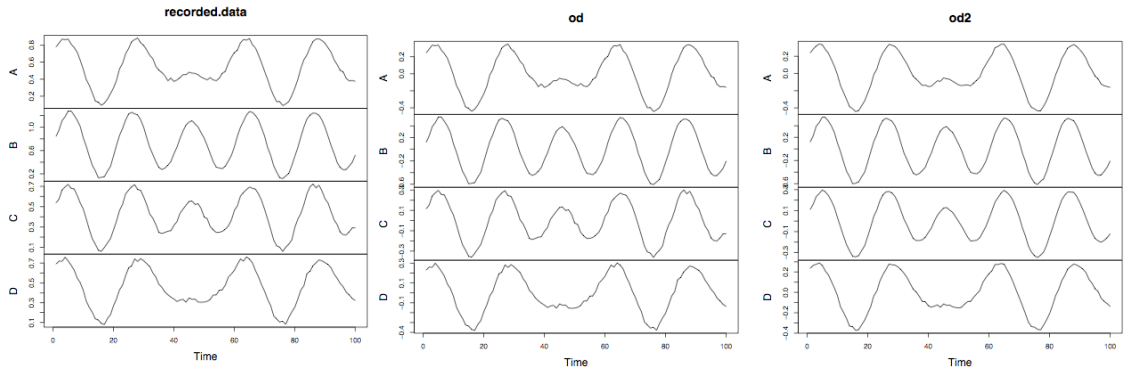


Figure 4. If no components are discarded, a perfect reconstruction (od) can be made of the original data (recorded.data). If the smallest components are discarded, the reconstruction (od2) is not exact, but is very good.

Summary

Principal components analysis is a popular tool for studying high-dimensional data. It relies on four major assumptions (Shlens):

1. Linearity. This means that the only interaction among different signal sources is that they add together. If the strength of a combined signal were the product of the strengths of contributing signals, for instance, this would be a non-linear interaction and PCA would not work.
2. The interesting dynamics have the largest variances.
3. Mean and variance are sufficient statistics. Since PCA is designed around the covariance matrix of mean-centered data, the only statistics it considers are the mean and variance. If the data cannot adequately be described by its mean and variance (e.g. it is not Gaussian or exponentially distributed), PCA will be inadequate.
4. Orthogonal components. This is a particularly strong assumption made by PCA. It is this assumption that allows PCA to be computationally straightforward, but is not a realistic assumption in many data sets.

If assumptions 2 through 4 seem untenable for your data set, consider using Independent Components Analysis (ICA). This is a computationally more difficult algorithm that re-describes data in statistically independent—but not necessarily orthogonal—components, and relies only on the linearity assumption (Shlens).

Although PCA has its limitations, it is often a useful first step in re-describing data in terms of its hidden structure. It can help decide how many dimensions are necessary for a good description of the data. By reducing the dimensions of the data, it can make analysis simpler and can also make data sets smaller without losing important information.

References

Briggman et al., Optical Imaging of Neuronal Populations During Decision-Making. Science (2005)

Dien et al., Localization of the event-related potential novelty response as defined by principal components analysis. Cognitive Brain Research (2003)

Dien et al., Introduction to principal components analysis of event-related potentials. Event related potentials: A methods handbook (2005)

Gu et al., Principal components analysis of morphological measures in the Quebec family study: Familial Correlations. American Journal of Human Biology (1997)

Maindonald and Braun. Data Analysis and Graphics Using R: An Example-based Approach. Cambridge University Press. 2007

R Development Core Team (2007). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Shlens. A Tutorial on Principal Component Analysis. Copy retrieved [04-09-2008] from: http://www.cs.cmu.edu/~elaw/papers/pca.pdf (2005)

Turk et al. Eigenfaces for Recognition. v3.espacenet.com (1991)