Quantitative Understanding in Biology
3.4 Principal Component Analysis

Jinhyun Ju

April 12th, 2015

1 Introduction

Principal component analysis (PCA) is a powerful yet simple method widely used for analyzing high dimensional datasets. When dealing with datasets such as gene expression measurements, one of the biggest challenges is just the size of the data itself. Transcriptome wide gene expression data usally have 10,000+ measurements per sample and commonly used sequence variation datasets have around 600,000 $\sim$ 900,0000 measurements per sample. The high dimensionality not only makes it difficult to perform statistical analyses on the data, but also makes the visualization and exploration of the data challenging. Imagine trying to find a pattern from these datasets which cannot be fully visualized because the number of datapoints exceeds the number of pixels on your monitor. This is when you want to use PCA to represent the given dataset in lower dimensions to visualize it in 2-D or 3-D and identify any apparent patterns that might be hidden in the high dimensions.

2 The core concept: Change of basis

Let us think about the problem that we are facing with a simple example that involves the measurement of 5 genes $g_1, ..., g_5$. In this case, we would have 5 measurements for each sample and they can be represented as points in a 5 dimensional space. For example, if the measurements for sample 1 (x_1) were: $g_1 = 1, g_2 = 2.3, g_3 = 3.7, g_4 = 0.2, g_5 = 0.3$, we can represent x_1 like this (Just like we would define a point on a two dimensional space with x-y axes as [x,y]) :

$$x_1 = \begin{pmatrix} 1 \\ 2.3 \\ 3.7 \\ 0.2 \\ 0.3 \end{pmatrix} \tag{1}$$

It might be quite obvious, but this means that the point x_1 is defined as 1 unit in the g_1 direction, 0.5 units in the g_2 direction, 2 in the g_3 direction and so on,

$$x_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} * 1 + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} * 2.3 + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} * 3.7 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} * 0.2 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} * 0.3 \tag{2}$$

where the set of 5 vectors are the "basis set" for the given data. If all of the 5 genes are independent of each other (or orthogonal if we talk in vector terms) we would have to look at the data as it is, since knowing the expression level of g_1 will not give us any information about other genes. In such a scenario, reducing the dimensions would just mean dropping a measurement of a gene and losing the information about that gene's expression level. However, if there is an underlying driving force influencing the expression levels of multiple genes, (for example, pathway activation that bumps up the expression levels of g_1 and g_2 together, or a feedback interaction where a high level of g_5 suppresses the expression level of g_3 and g_4) we don't have to focus on the expression levels individually. This would mean that by knowing the expression level of g_1 we can get some sense of the expression level of g_2, and from the level of g_3 we can guess the levels of g_4 and g_5. So let's say for the sake of simplicity, that our 1 sample captures this relationship perfectly (this will probably never happen in real life, but anyways...) and represent our sample in a more compact way. Something like this might do the job.

$$x_1 = \begin{pmatrix} 1 \\ 2.3 \\ 0 \\ 0 \\ 0 \end{pmatrix} * 1 + \begin{pmatrix} 0 \\ 0 \\ 3.7 \\ 0.2 \\ 0.3 \end{pmatrix} * 1 \tag{3}$$

What this means is that we are representing our sample x_1 as a two dimensional point (1,1) in the new space defined by the relationships between genes. This is also called "change of basis" since we are changing the basis to represent our samples in a different space. A better representation would be to normalize the basis vectors to have a unit norm like this where the normalization factor just gets multiplied into the coordinates in the space:

$$x_1 = \begin{pmatrix} 0.3987261 \\ 0.9170701 \\ 0 \\ 0 \\ 0 \end{pmatrix} * 2.507897 + \begin{pmatrix} 0 \\ 0 \\ 0.99528556 \\ 0.05379922 \\ 0.08069883 \end{pmatrix} * 3.717526 \tag{4}$$

This is basically what happens when we reduce the dimensionality with PCA. PCA will look for "redundancy" in the measurements and will represent the data with a new basis that minimizes the redundancy of measurements. A key thing to remeber is, that here we assumed the new basis are the ground truth because I said so, but in reality PCA will first calculate the covariance matrix using all the samples and then calculate the new basis vectors by calculating the eigen vectors of the covariance matrix.

3 Correlation and Covariance

In order to understand PCA, we need to introduce the concept of covariance in relation to correlation. The variance of a sample can be calculated by

$$E[(x - \bar{x})^2)] = s^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

Covariance can be thought of as variance with 2 variables, which takes the form as follows:

$$E[(x - \bar{x})(y - \bar{y})] = cov(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

If this looks awfully familiar you are not mistaken. Correlation between two variables is just a scaled form of covariance, which only takes values between -1 and 1.

$$cor(x, y) = \frac{cov(x, y)}{\sigma_x, \sigma_y}$$

So just like correlation, covariance will be close to 0 if the two variables are independent and will take a positive value if they tend to move in the same direction and a negative value if the opposite is true.

4 PCA example in R

PCA in R is very simple once you get familiar with the outputs of the functions that perform PCA. However, to understand how it is calculated, let us begin with calculating the results without the PCA function. As mentioned in the above section, the first step would be to calculate the covariance matrix for each measurement against every other measurement from the samples. For this, we would need some samples, and our good friend **`rnorm()`** can help us out with this. First, let's look at what a covariance matrix for two independent measurements will look like.

```
x <- rnorm(300,0,1)
y <- rnorm(300,0,1)

samples <- cbind(x,y)

cov(samples)

##            x          y
## x 0.92179814 0.01086061
## y 0.01086061 0.99076068
```

For independent variables, the covariance matrix will have the variance of each variable on the diagonal and the covariance terms for each variable pair on the off-diagnol. So the matrix shown above is representing this:

$$\begin{pmatrix} cov(x,x) = var(x) & cov(x,y) \\ cov(y,x) & cov(y,y) = var(y) \end{pmatrix} \tag{5}$$

and we check this by calculating each entry separately.

```
var(x)

## [1] 0.9217981

var(y)

## [1] 0.9907607

cov(x,y)

## [1] 0.01086061
```

In summary, the diagonal values will tell us how much variation each variable has by itself, and the off-diagonal terms will tell us how much each variable pair tends to move together.

Here the two variables were independent, thus the off-diagonal terms are close to 0. Now let's see what happens when two variables are not independent.

```
x <- 2 + rnorm(300,0,1)
y <- 0.5 + 1.5*x + rnorm(300,0,1)

samples <- cbind(x,y)

cov(samples)
```

```
##          x        y
## x 1.068901 1.623963
## y 1.623963 3.308765
```

Now we have strong off-diagonal terms that indicate the relationship between x and y, which is not very surprising since the value of y was generated based on the value of x.

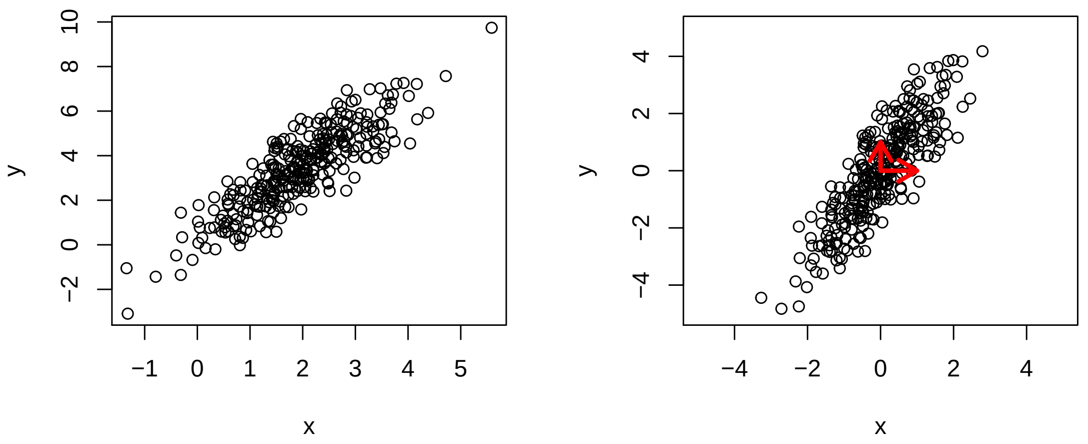
```
par(mfrow = c(1,2))
example.data <- cbind(x,y)

# plot the data as it is
plot(example.data[,1],example.data[,2], xlab = "x", ylab = "y")

# center the data
example.data.center <- scale(example.data, center = TRUE, scale = FALSE)

# plot centered data with basis vectors
plot(example.data.center[,1], example.data.center[,2],
     xlim = c(-5,5), ylim = c(-5,5), xlab = "x", ylab = "y")

arrows(x0=0, y0=0, x1 = 1, y1 = 0, col = "red", lwd = 3,length =0.15)
arrows(x0=0, y0=0, x1 = 0, y1 = 1, col = "red", lwd = 3,length =0.15)
```



When we plot the data the structure is quite obvious. In this case we don't really need to reduce the dimensions since a 2-D plot is easy to generate. However, for the sake of demonstration (and the lack of ability to plot 4 or 5 dimensional data) let us try to reduce this 2 dimensional dataset into a single dimension without losing too much information. The most valuable information in this dataset is probably the correlation between x and y (since there is not much left if you take that relationship out... just normal errors). So it seems like a good idea to keep that information in the single dimension that we have. Let's first center our data to (0,0) to make it easier to draw vectors.

On the right plot, the data is centered and represented with the basis vectors (1,0) and (0,1) shown as the red arrows. In order to capture the relationship between x and y and representing the data in 1-D we would probably use a vector that goes along the diagonal of the data. The direction along the diagonal explains the the largest amount of variance in the data (has the largest spread along its direction) and if we project each data point onto this vector we wont be losing too much information about the relationship between x and y. Let's find out the exact direction of that vector by using pca in R. There are two functions in R that are commonly used to perform pca: prcomp() and princomp(). Although they are doing the samething, they use slightly different methods to calculate the outcomes and prcomp() happens to use the method that is faster and is computationally less expensive. So let's use prcomp() to do our calculations.

```r
# when you use prcomp, your input data should have measrued variables
# in columns and individual samples/points as rows
# (N samples x G genes (or measurements) )
```

```
pca.result <- prcomp(example.data.center)
```

That was easy, but what is saved in the result?

```
str(pca.result)

## List of 5
##  $ sdev    : num [1:2] 2.04 0.465
##  $ rotation: num [1:2, 1:2] -0.465 -0.885 -0.885 0.465
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "x" "y"
##   .. ..$ : chr [1:2] "PC1" "PC2"
##  $ center  : Named num [1:2] -2.00e-17 -1.41e-17
##   ..- attr(*, "names")= chr [1:2] "x" "y"
##  $ scale   : logi FALSE
##  $ x       : num [1:300, 1:2] -2.865 0.308 -1.614 0.845 3.194 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "PC1" "PC2"
##  - attr(*, "class")= chr "prcomp"
```

You can see that there are 5 different results saved in the variable pca.result.

```
# "sdev" contains information about the fraction of variation
# explained by a certain principal component.
pca.result$sdev

## [1] 2.039981 0.464913

(pca.result$sdev / sum(pca.result$sdev))*100

## [1] 81.43981 18.56019
```

What is shown here is the percentage of variance explained by each principal component. This means that the first PC explains 74% of the variation in the data, and the second component explains about 26% of the variation and so on.

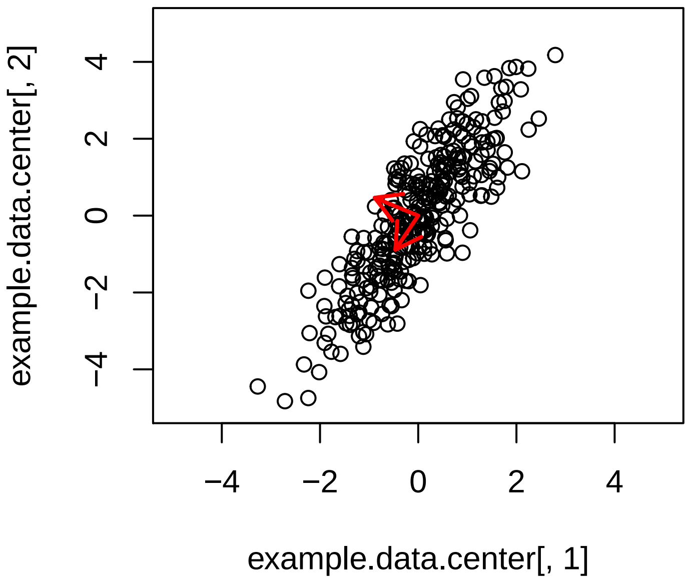
```
# "rotation" contains the directions
# of principal components in each of its columns.

pca.result$rotation

##          PC1        PC2
## x -0.4649094 -0.8853582
## y -0.8853582  0.4649094
```

```
plot(example.data.center[,1], example.data.center[,2],
     xlim = c(-5,5), ylim = c(-5,5))

arrows(x0=0, y0=0,
       x1 = pca.result$rotation[1,1],
       y1 = pca.result$rotation[2,1],
       col = "red", lwd = 2,length =0.15)
arrows(x0=0, y0=0,
       x1 = pca.result$rotation[1,2],
       y1 = pca.result$rotation[2,2],
       col = "red", lwd = 2,length =0.15)
```



We can see that the first PC is the direction along the diagonal.

```
# "center" contains the mean for each data column
# (in our case it would be close or equal to 0 since we centered the data).

pca.result$center

##             x             y
## -1.998401e-17 -1.406282e-17
```

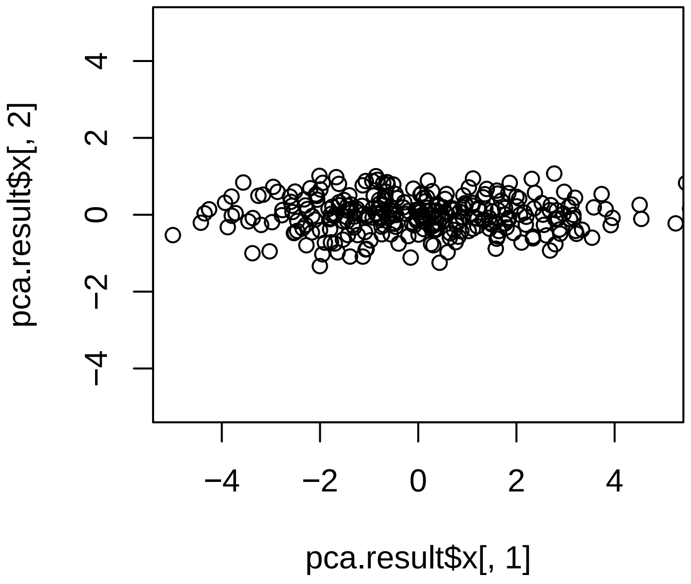
```r
# "scale" contains information about whether or not we gave the option to scale
# (divide it by its standard deviation) the data.

pca.result$scale
```

```
## [1] FALSE
```

```r
# "x" contains the coordinate values of the data projected
# to a principal component in each of its columns.

plot(pca.result$x[,1],pca.result$x[,2],xlim = c(-5,5), ylim = c(-5,5))
```



You can see that the representation of the data looks like a rotation using the diagonal of the original data as the first axis. So if we are interested in only keep 1-D of the data without losing too much information, our best shot would be to keep the first column of the projected data pca.result$x[,1].

5 Further Reading

For a more detailed and intuitive explanation on PCA, I highly recommend the tutorial written by Jonathon Shlens: `http://arxiv.org/pdf/1404.1100.pdf`