PCA

Sebastian Heucke

9/30/2019

Contents

PCA

This tutorial is an R markdown version of https://www.youtube.com/watch?v=0Jp4gsfOLMs&list=PLblh5JKOoLUJJpBNfk8_YadPwDTO2SCbx&index=1

The topics of the tutorial:

- How to use **prcomp()** function to do PCA
- How to draw a PCA plot using **base graphics** and **ggplot2**
- How to determine how much variation each principal component accounts for.
- How to examine the loading scores to determine what variables have the largest effect on the graph.

First, let's generate a fake dataset that we can use in the demonstration. We will make a matrix of data with 10 samples where we measured 100 genes in each sample. The first 5 samples will be "wt" or "wild type" samples. The last 5 samples will be "ko" or "knock-out" samples. The rows represent the genes, usually you'd have things like *Sox9*, but since this is a fake dataset, we have gene1, gene2,. . . , gene100.

```r
data.matrix <- matrix(nrow=100, ncol=10)

# create and name samples as columns
colnames(data.matrix) <- c(
 paste("wt", 1:5, sep=""),
 paste("ko", 1:5, sep=""))

# genenames 1..100 for rows
rownames(data.matrix) <- paste("gene", 1:100, sep="")

# give the fake genes fake read counts using poission distribution
for (i in 1:100) {
 wt.values <- rpois(5, lambda=sample(x=10:1000, size=1))
 ko.values <- rpois(5, lambda=sample(x=10:1000, size=1))

 data.matrix[i,] <- c(wt.values, ko.values)
}
```

```r
# look at first 6 rows
head(data.matrix)
```

```
##       wt1 wt2 wt3 wt4 wt5  ko1 ko2 ko3 ko4  ko5
## gene1 671 682 656 677 684 1002 932 981 925  947
## gene2 821 853 819 868 832  529 546 551 527  501
## gene3 816 838 770 824 771  423 435 444 400  387
## gene4 557 578 555 526 540  485 480 457 463  501
## gene5 540 513 519 506 474   66  65  58  64   57
## gene6 195 216 197 202 201  968 981 953 927 1003
```

```r
# dimension of the matrix
dim(data.matrix)
```

```
## [1] 100  10
```

With the data call **prcomp() to do a PCA on our data.

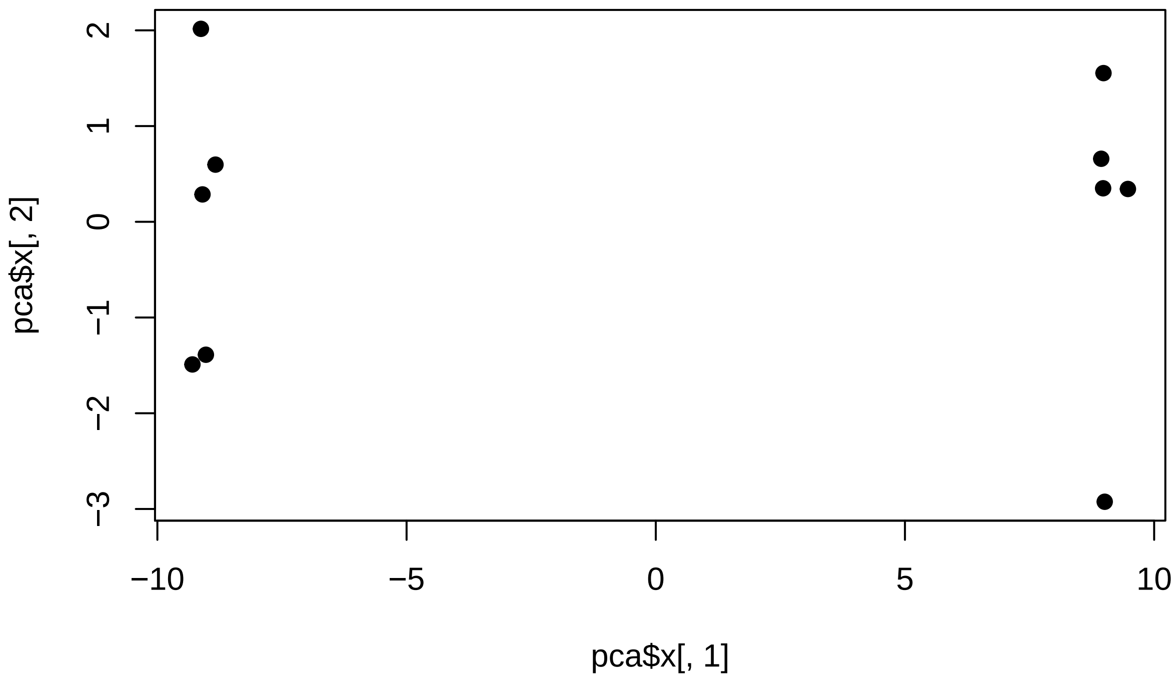
```r
pca <- prcomp(t(data.matrix), scale=TRUE)
```

The goal is to draw a graph that shows how the samples are related (or not related) to each other. By default, **prcomp()** expects the samples to be rows and the genes to be columns. Since the samples in our data matrix are columns, and the genes (variables) are rows we have to transpose the matrix using the **t()** function.

The function **prcomp()** returns:

- **x** contains the principal components (PCs) for drawing a graph. Here we are using the first two columns in **x** to draw a 2-D plot that uses the first two PCs. Remember, since there are 10 samples, there are 10 PCs. The first PC accounts for the most variation in the original data (gene expression across all 10 samples), the 2nd PC accounts for the second most variation and so on.

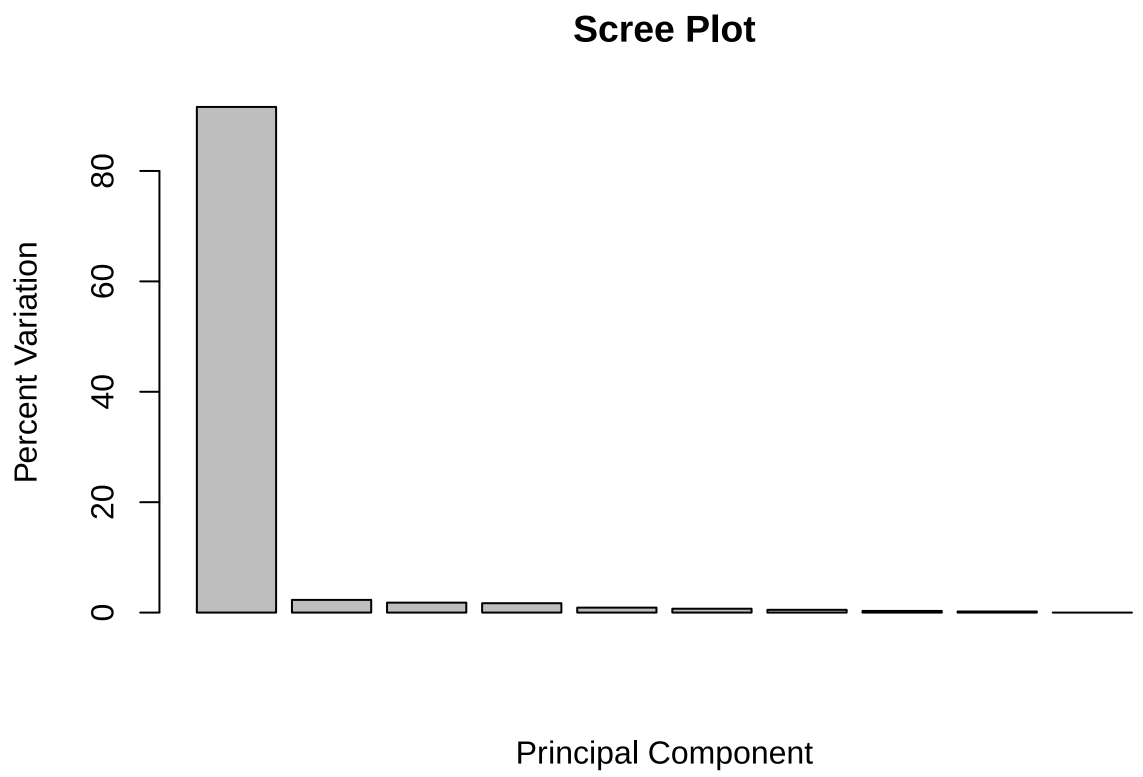
```r
plot(pca$x[,1], pca$x[,2])
```



PC1 is on the x-axis and PC2 is on the z-axis. 5 of the samples are on one side of the graph and the other 5 are one the other side of the graph.

To get a sence of how meaningful these clusters are, let's see how much variation in the original data PC1 accounts for. To do this we use the square of **sdev**, which stands for "standard deviation", to calculate how much variation in the original data each principal component accounts for. since the percentage of variation that each PC accounts for is way more interesting then the actual value, we calculate the percentages and plotting the percentages with **barplot()**.

```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
barplot(pca.var.per, main="Scree Plot", xlab="Principal Component",
ylab="Percent Variation")
```



PC1 accounts for almost all of the variation in the data. This means there is a big difference between the two clusters.

Using ggplot2 for fancier looking plot

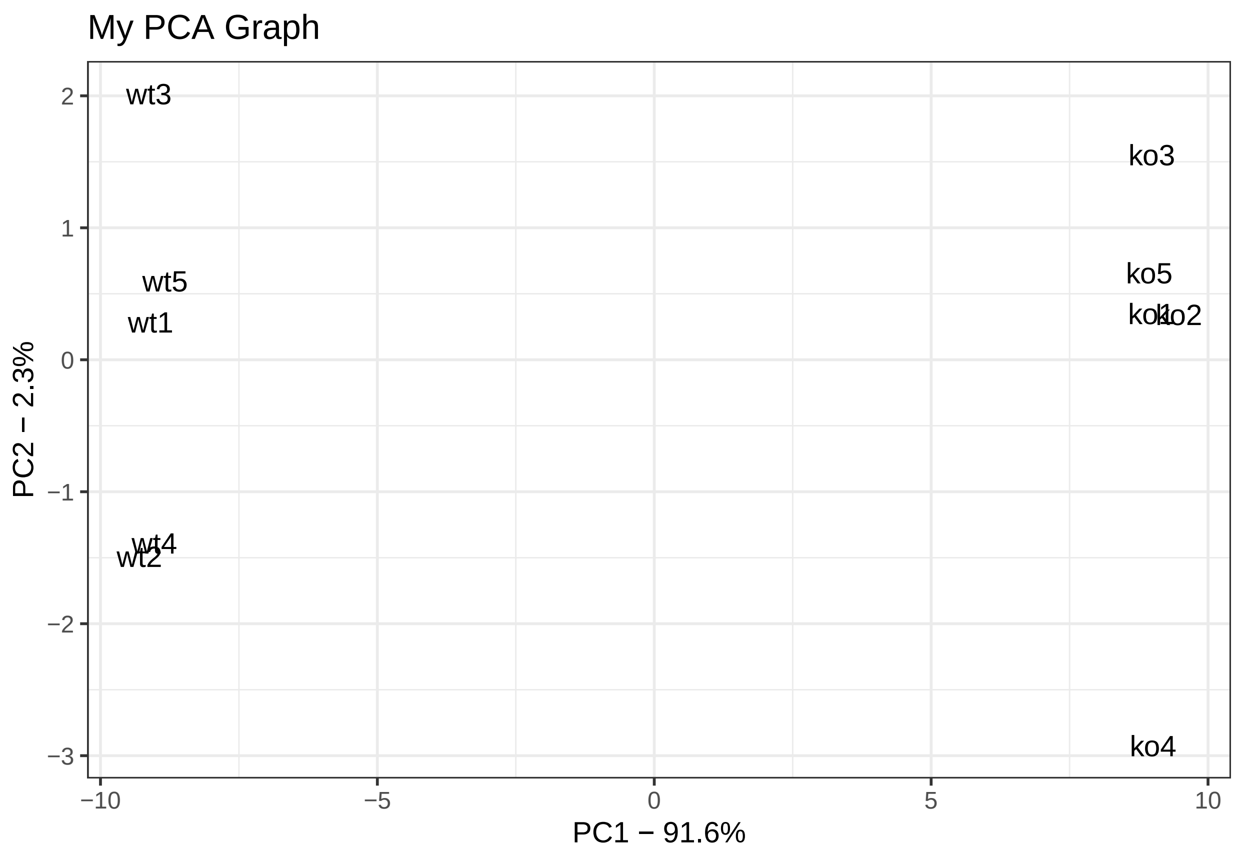
First, format the data the way **ggplot2** likes it. One column with the sample ids. Two columns for the X and Y coordinates for each sample.

```
library(ggplot2)
pca.data <- data.frame(Sample=rownames(pca$x),
 X=pca$x[,1],
 Y=pca$x[,2])
pca.data
```

```
##     Sample         X          Y
## wt1    wt1 -9.094859  0.2860832
## wt2    wt2 -9.297018 -1.4900833
## wt3    wt3 -9.126602  2.0155430
## wt4    wt4 -9.026778 -1.3887668
## wt5    wt5 -8.834447  0.5972785
## ko1    ko1  8.976039  0.3499602
## ko2    ko2  9.474057  0.3427847
## ko3    ko3  8.983676  1.5532556
## ko4    ko4  9.007959 -2.9244866
## ko5    ko5  8.937974  0.6584315
```

We have one row per sample. Each row has a sample ID and X/Y coordinates for that sample.

```r
ggplot(data=pca.data, aes(x=X, y=Y, label=Sample)) +
 geom_text() +
 xlab(paste("PC1 - ", pca.var.per[1], "%", sep="")) +
 ylab(paste("PC2 - ", pca.var.per[2], "%", sep="")) +
 theme_bw() +
 ggtitle("My PCA Graph")
```



The X-axis tells us what percentage of the variation in the original data that PC1 accounts for. The Y-axis tells us what percentage of the variation in the original data that PC2 accounts for. Now the samples are labeled, so we know which ones are on the left and the right.

In the first part of the **ggplot()** call, we pass in the **pca.data** dataframe and tell **ggplot** which columns contain the X and Y coordinates and which column has the sample labels. Then we use **geom_text()** to tell **ggplot** to plot the labels, rather than dots or some other shape. Then we use **xlab()** and **ylab()** to add X and Y axis labels. Here I'm using the **paste()** function to combine the percentage of variation with some text to make the labels look nice. Calling **theme_bw()** makes the graph's background white. Lastly we add a title to the graph using **ggtitle()**.

Next, let's look at how to use loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot.

The **prcomp()** function calles the loading scores **rotaion**. There are loading scores for each PC. Here I'm just going to look at the loading scores for PC1, since it accounts for 92% of the variation in the data.

Genes that push samples to the left side of the graph will have large negative values and genes that push samples to the right will have large positive values. since we're interested in both sets of genes, we'll use the **abs()** function to sort based on the number's magnitude rather than from high to low. Then we get the names of the top 10 genes with the largest loading score magnitudes and lastly we see which of these genes have positive loading scores, these push the "ko" samples to the right side of the graph and we see genes which have negative loading scores, these push the "wt" samples to the left side of the graph.

```
loading_scores <- pca$rotation[,1]
gene_scores <- abs(loading_scores) # get the magnitudes
gene_score_ranked <- sort(gene_scores, decreasing=TRUE) # sort form high to low
top_10_genes <- names(gene_score_ranked[1:10])
top_10_genes # show the names of the top 10 genes
```

```
##  [1] "gene23"  "gene57"  "gene97"  "gene62"  "gene100" "gene65"  "gene37"
##  [8] "gene53"  "gene72"  "gene6"
```

```
pca$rotation[top_10_genes,1] # show the scores (and +/- sign)
```

```
##     gene23     gene57     gene97     gene62    gene100     gene65
## -0.1044606 -0.1044593  0.1044506  0.1044434  0.1044222 -0.1044131
##     gene37     gene53     gene72      gene6
##  0.1044050  0.1043816 -0.1043721  0.1043652
```

Session information

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.2.0  knitr_1.23     devtools_2.1.0 usethis_1.5.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1        pillar_1.4.2      compiler_3.6.1
##  [4] prettyunits_1.0.2 remotes_2.1.0     tools_3.6.1
##  [7] testthat_2.1.1    digest_0.6.20     pkgbuild_1.0.5
## [10] pkgload_1.0.2     tibble_2.1.3      evaluate_0.14
## [13] memoise_1.1.0     gtable_0.3.0      pkgconfig_2.0.2
## [16] rlang_0.4.0       cli_1.1.0         yaml_2.2.0
## [19] xfun_0.8          dplyr_0.8.3       withr_2.1.2
## [22] stringr_1.4.0     desc_1.2.0        fs_1.3.1
## [25] tidyselect_0.2.5  rprojroot_1.3-2   grid_3.6.1
## [28] glue_1.3.1        R6_2.4.0          processx_3.4.1
## [31] rmarkdown_1.14    sessioninfo_1.1.1 purrr_0.3.2
## [34] callr_3.3.1       magrittr_1.5      backports_1.1.4
## [37] scales_1.0.0      ps_1.3.0          htmltools_0.3.6
## [40] assertthat_0.2.1  colorspace_1.4-1  labeling_0.3
## [43] stringi_1.4.3     lazyeval_0.2.2    munsell_0.5.0
## [46] crayon_1.3.4
```

The document was processed on 2019-09-30.