5 - Data normalization and visualization

TOPICS	
DATE	@April 13, 2022 2:00 PM
LAST EDITED BY	
LAST EDITED TIME	@April 28, 2022 5:01 PM
MADE BY	
PROF	Calogero
Recording	
STATUS	☐

summary of already explained points

bulk

- q: is cue provoking a change in transcriptome?
- RSEM: after we have annotation, it calculates relative expression of all isoforms and associated reads that are from various isoform. it uses for each isoform the exons specific for that isoform to assign fraction of reads to gene isoform. in general in gene level we can make a summary of the overall the countable all the isoforms. actually most people are not interested in this step because it is complex.
- What we get at the end: we get from this part a count table that contain information of the counts for your experiment in terms of number of reads associated to the specific gene or the RNAs or the UMI for specific gene
- no zero-inflation
- q: is a subpopulation of cells in the sample changing upon treatment/cue?
- we need R1(contain information that are related to the cell code and R2 (both info on sequence and cell/transcript of origin)
- less amount of RNA → quality is always a bit worse than in bulk
- counting out pcr artifacts is crucial and the UMI is what allows us to do so
- zero-inflated: many transcripts will seem to have 0 expression if they have low expression → count table in the sparse form. we need to convert it into a dense matrix w all 0's w cell ranger

both

- map reads on the genome → STAR which extremely fast (we can map discontinuous elements: transcript to genome in each exons which disconnected by introns and other stuff)

Next step:

- Perform annotation of the peaks → you need gtf file: tab-delimited file containing annotation for each genomic coordinates (UCSC and Ensemble, one is gene centric and the other is transcript centric).
- end: number of unique UMI for a gene

At this point:

- we have done all the quality controls
- we have a table for bulk wit columns for different samples / a table for sc w columns for different cells

Today we focus on bulk: we'd like to visualize our data in a way that allows us to see whether there are significant differences among our samples in terms of differential gene expression.

Before visualization, the first step is normalization of the data: different formats. The standard is **TPM (transcripts per million**). Since we are not yet skilled in R we are going to use a different type of normalization, **CPM (counts per million)**. TPM is used today in place of the previous normalization method, FPKM. The following video is what the professor used to explain this topic.

https://www.youtube.com/watch?v=TTUrtCY2k-w&t=474s

Let's start with RPKM (Reads Per Kilobase Million).

There are imaginary an RNA-seq data with three replicates (Rep 1, 2 and 3) for a genome with four genes (A-D) which are different in terms of some characteristics such as size, for example, A is 2 kilobase, B is 4 kb, C is 1 kb and D is very large with 10 kb.
Looking at the replicates, Rep 1 and 2 are very similar whereas Rep 3 is higher than others,
but what is considered is gene D is very little expressed, it means that you are not able to get any information about it. Also replicate three seems to be more rich in reads with respect to the other, and if you look at the gene A and the gene B you find out that B is twise as long as gene A, hence the number of reads of gene B is double compare to gene A.

- Sum all the reads and divide by 1 million (10 in the example for simplicity)
- Divide all read values by the "per million" factor → scale for read depth

Gene Name	Rep1 Counts	Rep2 Counts	Rep3 Counts
A (2kb)	10	12	30
B (4kb)	20	25	60
C (1kb)	5	8	15
D (10kb)	0	0	1

STEP 1: Normalize for read depth

Why the size of the genes is important?

Because the number of fragments is related to the size of the genes for example if you have 4 kb you generate double number of fragments with respect to the 2kb ; and it would be difficult to analyze because in real experiments we have more and longer genes.

In order to solve this problem we can summarize all columns by addition of all reads in each Rep, for example we have 35 reads for the first replication , 45 and 106 for the Rep 2 and 3 respectively. Then for representing the total number of reads, we use sort of conversion, we represent 35 reads as a number referred to 1 million reads in typical situation; but in this specific case because we have few genes, we are scaling the total read counts by 10 instead of 1,000,000. So now that our reference value is going to be 3.5 , 4.5 and 10.6 for Rep 1, 2 and 3 respectively .

Gene Name	Rep1 Counts	Rep2 Counts	Rep3 Counts
A (2kb)	10	12	30
B (4kb)	20	25	60
C (1kb)	5	8	15
D (10kb)	0	0	1
Total reads:	35	45	106
Tens of reads:	3.5	4.5	10.6

tens of reads (or million of reads depending on the division factor) (10/3.5….)

Now, we divide each of the various number of each Rep by reference value that refer to the column and by doing this, we are actually removing the effect that each column has more read with respect to the others and

finally the three replicates seems to have relatively similar number over reads after this normalization.

RPM - scaled using the "per million" factors.

Gene Name	Rep1 RPM	Rep2 RPM	Rep3 RPM
A (2kb)	2.86	2.67	2.83
B (4kb)	5.71	5.56	5.66
C (1kb)	1.43	1.78	1.43
D (10kb)	0	0	0.09

There is still difference between 2 genes A related to their size (2 kb or 4 kb) and we need to do correct this phenomenon.

STEP 2: Normalize for gene length

For correction the differences between length of genes, what we simply do is deviding the value by the size of the gene that we are considering; for example the first row is going to be divided by 2 (2 kb for gene A) and the second row is devided by 4 and so on.

Reads are scaled for depth (M) and gene length (K).

Gene Name	Rep1 RPKM	Rep2 RPKM	Rep3 RPKM
A (2kb)	1.43	1.33	1.42
B (4kb)	1.43	1.39	1.42
C (1kb)	1.43	1.78	1.42
D (10kb)	0	0	0.009

Now you can see that not only differences among each columns but in each row is disappeared and this is RPKM kind of normalization that is summarized by this formula:

FPKM is the same as RPKM but for paired-end reads.

$$\mathrm{FPKM}_i = \frac{X_i}{\left(\frac{\widetilde{l}_i}{10^3}\right)\left(\frac{N}{10^6}\right)} = \frac{X_i}{\widetilde{l}_i N} \cdot 10^9$$

The interpretation of FPKM is as follows: if you were to sequence this pool of RNA again, you expect to see FPKM_i fragments for each thousand bases in the feature for every $N/10^6$ fragments you've sequenced. It's basically just the rate of fragments per base multiplied by a big number (proportional to the number of fragments you sequenced) to make it more convenient.

li

“XI” is an expression in the count of gene “I”, and is actually divided by the length of the genes divided by 1000 because bases of our genes are represented as a number of kilo (1000) and the N is the total number of reads of columns that is going to be divided by 1 milion.

So if we organize the data, we have the ratio between expression level of our genes divided by the length of genes and the number of reads of the column multiplied by 10 to the power of 9 .

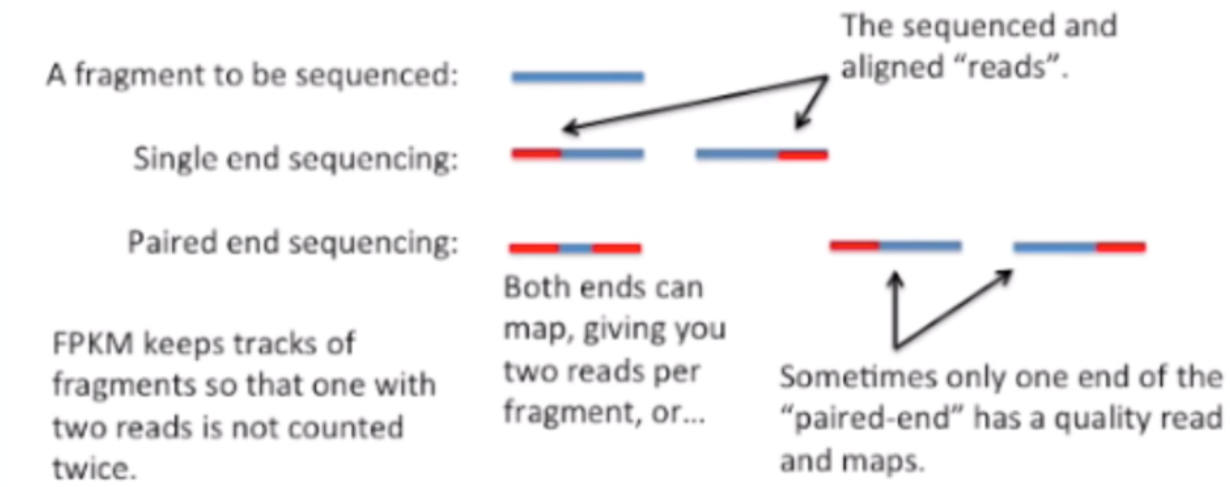
To summarize this normalization calculate the scaling factor by the column and then subsequently calculate the difference between the size of the gene in the row.

Differences between RPKM and FPKM :

RPKM : Reads Per Kilobase Million

FPKM: Fragments Per Kilobase Million

Also RPKM is related to single end RNA-seq whereas **FPKM** althogh is very similare to RPKM, used for paired end RNA-seq .



TPM :

In this normalization, the first thing that we do is actually normalizing our data set by the size of our genes then after having done that we calculate the scaling factor for each column and we do the normalization as we have done before.

💡 **TPM** (today standard for normalization): Transcripts per million.

- *First*, divide each count by the gene length
- *Second*, normalize for sequencing depth

$$\mathrm{TPM}_i = \frac{X_i}{\widetilde{l}_i} \cdot \left(\frac{1}{\sum_j \frac{X_j}{\widetilde{l}_j}} \right) \cdot 10^6$$

TPM has a very nice interpretation when you're looking at transcript abundances. As the name suggests, the interpretation is that if you were to sequence one million full length transcripts, TPM is the number of transcripts you would have seen of type i, given the abundances of the other transcripts in your sample. The last "given" part is important. The denominator is going to be different between experiments, and thus is also sample dependent which is why you cannot directly compare TPM between samples. While this is true, TPM is probably the most stable unit across experiments, though you still shouldn't compare it across experiments.

This kind of normalization (TPM transcript per million) is better since it allows to do a ranking of gene expression. If we have three samples, we expect a change in ranking position to happen when a gene is differentially expressed. For example that if our gene is very high express is going to be ranked very high in expression and if our gene is very little expressed is going to be ranked very low.

The important point is differentially expression of genes cause to change the ranking position in the treated samples and by doing this normalization you are still able to see differences that exist between different treatments.

The advantage of TPM is that all samples have the same “total size” after normalization. This allows for a better comparison of the samples.

Differences between two normalization:

what actually made the two normalization procedure different is after normalization, the three summary of the columns for the RPKM are totally different whereas in TPM, all samples have the same “total size” after normalization and this allows for a better comparison of the samples.

RPKM

Gene Name	Rep1 RPKM	Rep2 RPKM	Rep3 RPKM
A (2kb)	1.43	1.33	1.42
B (4kb)	1.43	1.39	1.42
C (1kb)	1.43	1.78	1.42
D (10kb)	0	0	0.009
Total:	4.29	4.5	4.25

.. the sums of each column are very different.

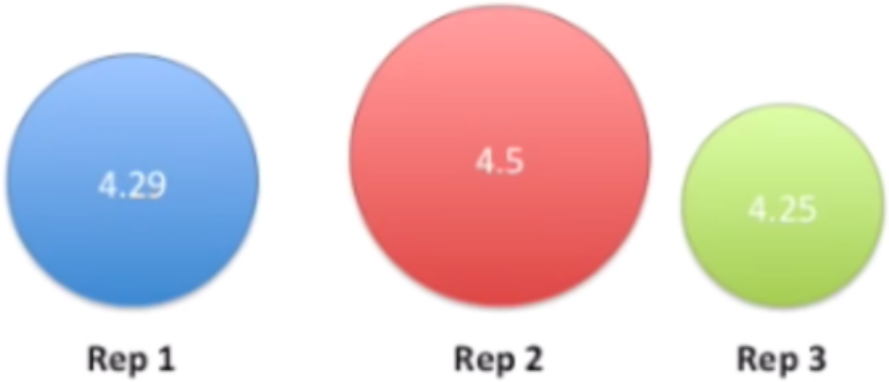
looking at the first one you can see that each pie has a different size so it is comolicated to do comparison. looking at TPM all thesummary are the same and all the pie will have the same size and can be compared to each other and you can evaluate if there are differences among the different samples

TPM

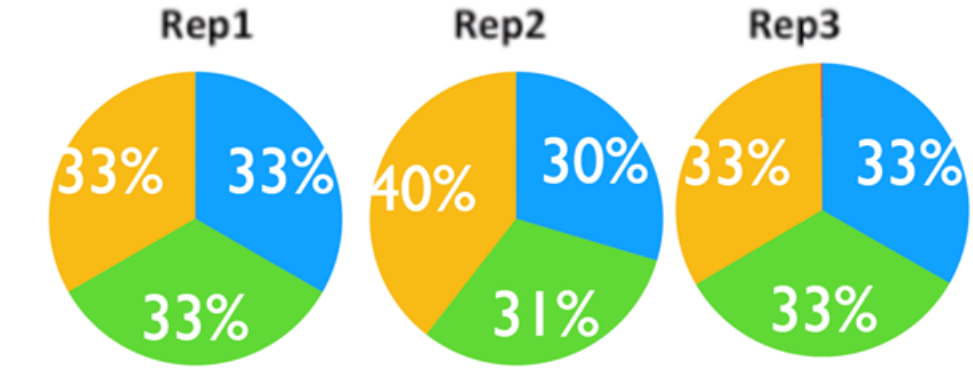
Gene Name	Rep1 TPM	Rep2 TPM	Rep3 TPM
A (2kb)	3.33	2.96	3.326
B (4kb)	3.33	3.09	3.326
C (1kb)	3.33	3.95	3.326
D (10kb)	0	0	0.02
Total:	10	10	10

As you can see in the example, comparing two pie charts are difficult cause of different size of each parts .

PRKM:



TPM:



In the moodle we can find a zip file with data from an experiment done on egfr-mutated lung cancer cells (PC9) which is sensitive to an inhibitor named Osimertinib. In this experiment we have three samples which is called mock samples treated with DMSO and then we have 2 different treatment conditions:

mock samples untreated one

- Acute (500nM Osim for 24h)
- Drug tolerant persisters (DTP) 500 nM osimertinib for 21 days which simulating what is actually happening in long treatment

We want to see if the 9 replicates are relatively homogeneous but they are changing in the various condition with respect to their treated

To do this exercise, you have to install the Bioconductor package edgeR that allow you to do differential expression analyzing and it has a specific function that will allow you to convert the directly your count matrix into a CPM matrix via Docker and also converting your count table into CPM normalized table.

You can find a command to use load the data in to your environment because the file is outside you have to bring into R and your matrix has to set in to a variable that is your matrix.

this only after opening Rstudio and setting the folder containing the file as the working directory.

So what we do is we define our variables X and you can use equal or you can use the arrow depending on what you prefer is the same. Then the command is :

`x = read.table ("filename", sep = "\t", header = T, row.names = 1)` header = T each column has a name

- I t is really important to set working directory the folder where your file is located because in the other case you get an error that is unable to open the folder.
- `\t` means setup
- `header = T, row.names = 1` about how we manage the structure of our table and row names is the parameter that ask you which column contain the row name that in your case is the number one

Now your X is going to be a data frame that is specific type of matrix and edgeR might ask you to convert the dataframe to a matrix which is :

```
X= as.matrix(x)row.names=1
```

Then use the CPM function on the x file. At this point we have a table which convert in CPM and then we will save the CPM converted table

we can use:

```
write.table (x, "cpm.txt", sep = "\t", col.names = NA)
```

what is considering is that the first column is considered as a full column with the first element that is empty

Visualizing experiment data

Install the R package dev.tools into Docker with `library(dev.tools)`

R packages consists of a collection of R functions, data sets and compiled code which adds value to the existing R-functionalities. They are stored in the 'library' directory in the R-environment and developed by the community.

So these are going to be 2 exercise:

1- the first one is compression and saving the data

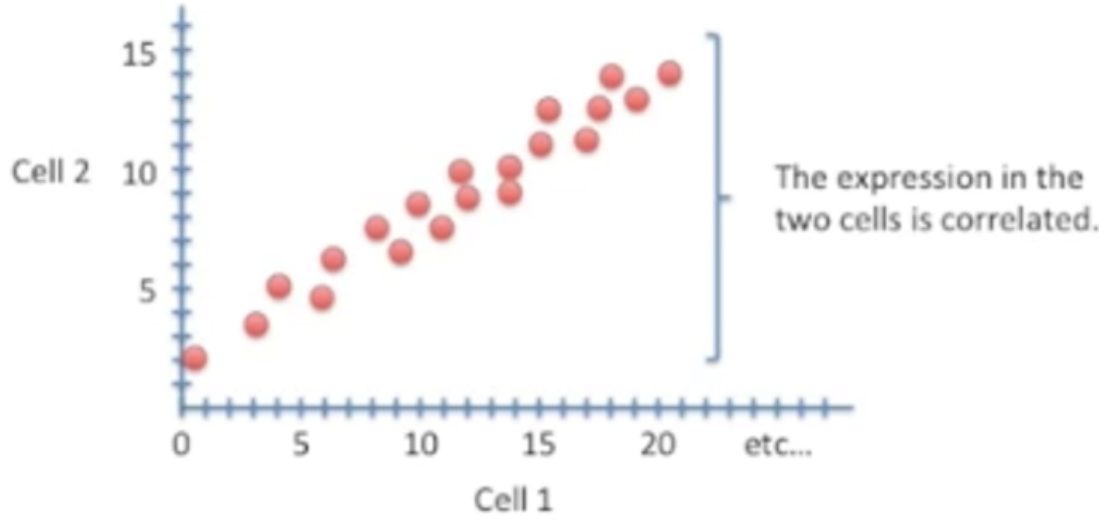
2- the second one is installing dev.tools and it is important because we need it to install a specific library for the second part

PCA

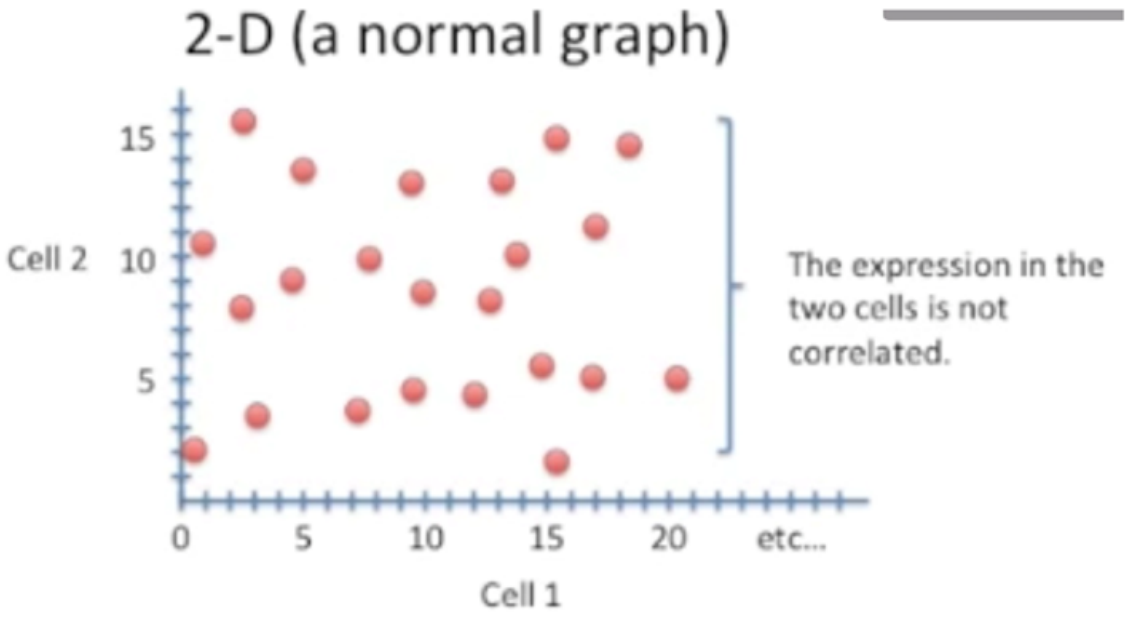
represent data in a reduced space (since our experiments are multidimensional and therefore impossible to represent in only three dimensions without dimensional reduction) based on **variance** among data. The following video is what the professor used to explain this topic.

https://www.youtube.com/watch?v=_UVHneBUBW0

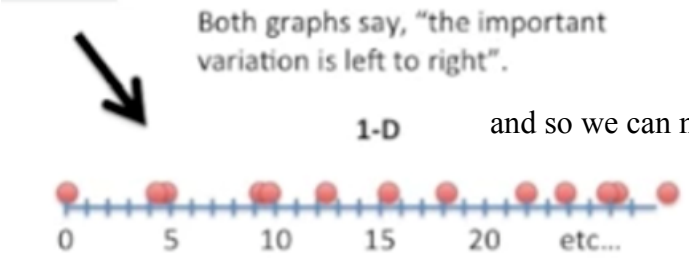
Here there is an example about genes(dots) of 2 different cells and what is really obvious is these genes are linearly related to each other and two gene as well.



In this example there is no relationship between the expression of two cells

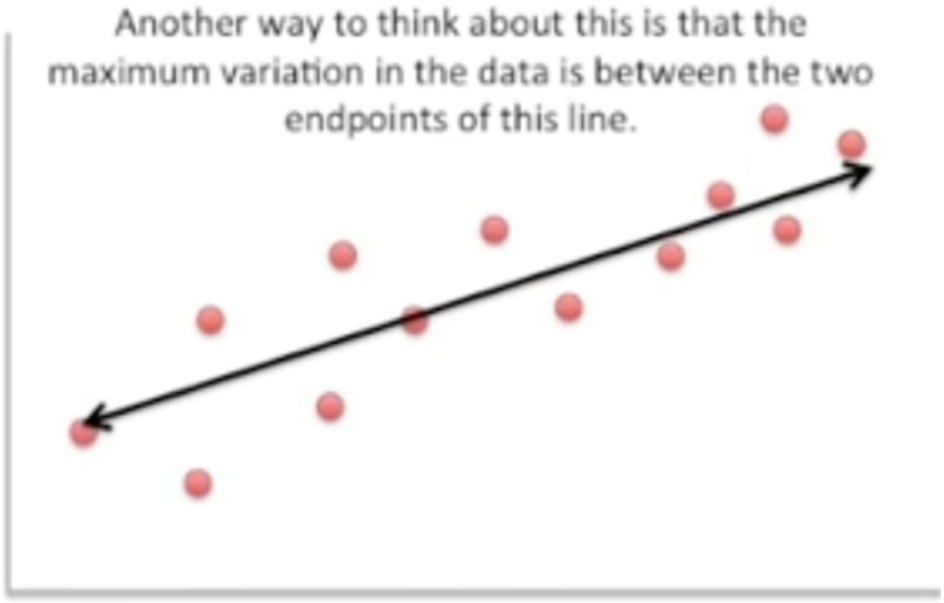


In reality yo don't need to dimension to represent the similarity of the two cells because the majority of differences related to cell one there for representation is smaller .

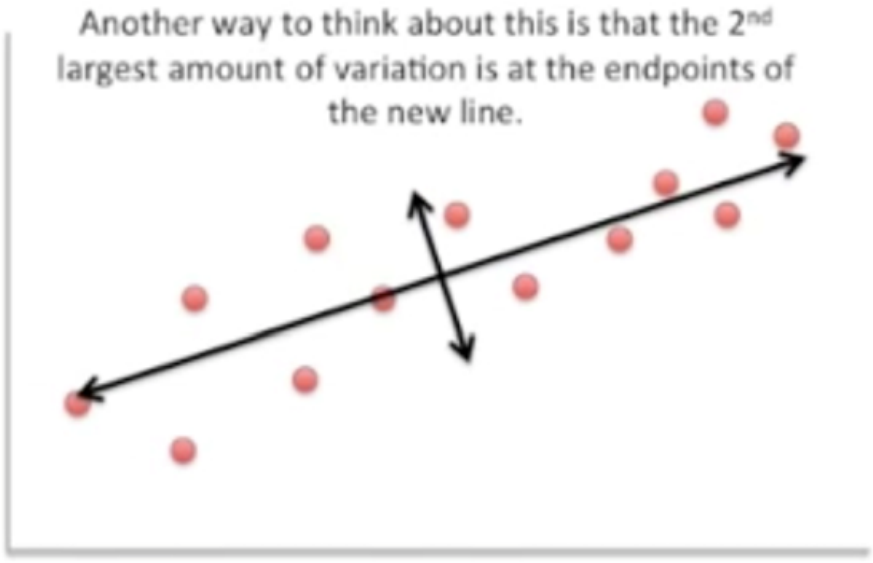


and so we can move to 1 dimension

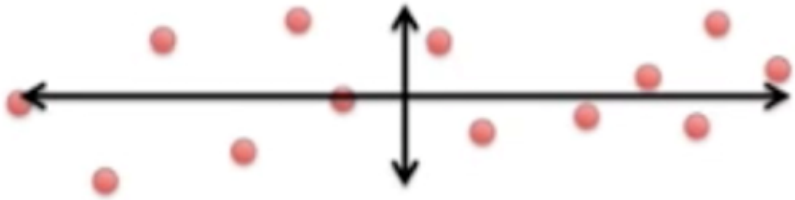
Now in this example there are various genes related to two cells. the first thing we plot two cells with various genes and then we plot the line that indicate the direction of the greatest variation that you can represent the data set. so in other word this graph show what is the most important variation associated to your data.



So what we can do to define the other genes that are in remote position compared the line, use second line that is perpendicular respect to the first one. it is obvious that the size of two variants ids different. what is considering if we have three samples we have another segments that is going to be perpendicular to second one that give you the third dimension.



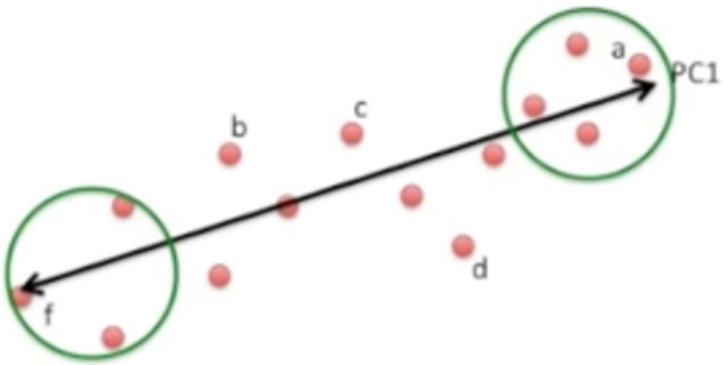
For just focusing on variants, the first segment became X axis and the perpendicular line become Y axis, so now cell 1 and 2 disappeared . changing the reference: loosing the cells and using the variants



principle component one and principle component two

You can have a lot of samples and different variants as well.

According to this example not all genes are important to estimate the variants that is represented, the largest and lowest value are have more effect on PC1 whereas genes b, c and d are less important in finding the variants.



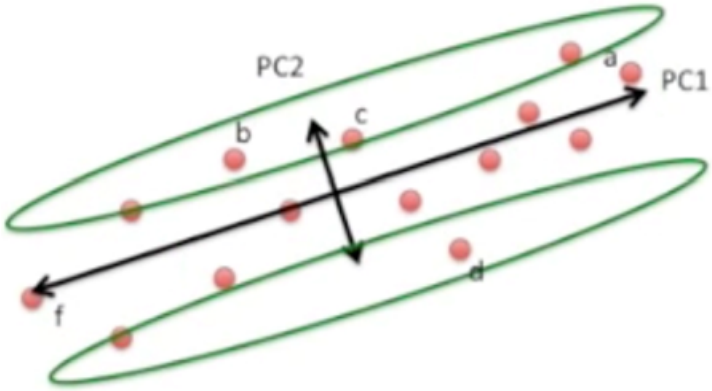
About second dimension is different, b, c and d are more important than a.

and you do this by assining to each gene a value taht define how important it is for each segment

have a number of dimension equal to the number of variants, in reality 2-3 are enough tor epresent the data

So depending on the component, importance in the representation of variants is given by a different setup genes.

The weights are called “loadings” adn an array of “loadings” for a certain PC is called an “eigenvector” This will be the starting point for the next lesson but how do i calculate the weight



Homework solution

1. Install the package “edgeR” to do differential expression analysis in the docker
   a. Write a Rscript with the command suggested in the edgeR page

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("edgeR")
```

   b. Then copy it onto the docker. The command is `docker cp path\of\the\source\file.R lastdockerimage:/home/`
   c. Do a commit
   d. Run the docker and verify that there is your new file in the homee
   e. Give the permission to execute the file (`chmod +x file.R`)
   f. Execute the file (`Rscript file.R`) and let it install the package

so starting fromt he table with the count and having now the two table for the two dimensions:
Cell PC1 score= (read count dalla prima tabella * influence dalla tabella relativa alla prima dimensione (si ripete tutto per ogni dimensione))+ si somma per ogni gene

I obtain a single value associated to the cell but this single value depends more from the gene which are more relevant for that particular dimension. So a summary of the expression weighted for their imprtance
Then repeat for the principal component 2 (the second dimenson)

So by doing this at the end obtain a conversion from genes to cells in the graph we will not have Cell1vs cell2 and the dots representing genes but PC1 vs PC2 and the dot representing the cell. Putting the other cells we will see more clearly the correlation of the cells and on which PC1 there is the big difference. So you will see a difference with the untreated condition but how much it is different.

```
PS C:\Users\eleon\desktop> docker cp C:\Users\eleon\Desktop\command_edgeR.R 699cb7989aa9:/home/
PS C:\Users\eleon\desktop> docker commit 699cb7989aa9 r4:v.0.01
sha256:374bc074902f4f7430f9e84b1f9a090b2fd10dbea690c45d778f203eaee50187
PS C:\Users\eleon\desktop> docker run -it r4:v.0.01
root@55ade60e1dbe:/# cd home
root@55ade60e1dbe:/home# ls
command_edgeR.R  command_umap.R  docker
root@55ade60e1dbe:/home# chmod +x command_edgeR.R
root@55ade60e1dbe:/home# rscript command_edgeR.R
bash: rscript: command not found
root@55ade60e1dbe:/home# Rscript command_edgeR.R
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
trying URL 'https://cloud.r-project.org/src/contrib/BiocManager_1.30.16.tar.gz'
Content type 'application/x-gzip' length 262502 bytes (256 KB)
==================================================
```

2. Write an R script to
   a. convert the file on moodle (dataset.txt) in an R object (i.e. a variable) and put it in the R environment

> ❗ If you try to do it inside R-studio, set as working directory the folder where the file is, otherwise it gets an error
>
> The command is `setwd("path/of/your/choice")`

```
x <- read.table ("nameoffile.txt", sep ="\t", header = TRUE, row.names = 1)
# x is now a dataframe
# convert then the dataframe into a matrix
y <- data.matrix(x, rownames.force = NA)
```

   b. Convert the csv (here we have a tab delimited txt file) into a cpm table (again in txt format)

```
z <- edgeR::cpm(y)
```

   c. Save it on a file on the folder where you are now

```
write.table(z, file = “L5cpm.txt”, sep = “\t”, col.names = NA)
```

   (The matrix we have is a tab-delimited matrix, for this reason we use “\t” as separator).

If you want to better understand what is happening, instead of writing a script, do it interactively on Rstudio (without the last part of the saving). This is what you get.

```
> setwd("C:/Users/eleon/Desktop/data_analysis")
> x <- read.table("L5dataset.txt", sep = "\t", header = TRUE, row.names = 1)
>
> y <- data.matrix(x, rownames.force = NA)
> z <- edgeR::cpm(y)
> class (x)
[1] "data.frame"
> Class (y)
Error in Class(y) : could not find function "Class"
>
> class (y)
[1] "matrix" "array"
> class (z)
[1] "matrix" "array"
> head (y)
         PC9.DMSO1  PC9.DMSO2  PC9.DMSO3 PC9.osiacute1 PC9.osiacute2 PC9.osiacute3  PC9.osiDTP1  PC9.osiDTP2 PC9.osiDTP3
A1BG      2.134688   5.929364   4.331146     8.0276302     15.762958     5.8535371   19.4726655   54.6598821   16.444272
A1CF      1.652831   4.144646   2.499168     4.9450323     10.044213     2.4764460   14.1457653    7.4760577    9.132462
A2M       8.212231   0.000000   0.000000    14.9416440      0.000000     0.3590596    0.3616486   33.3906184    7.940929
A2ML1   122.758423 120.174707  83.076498   167.3363296    245.611585   244.6464245  723.9662025  993.4183288  912.677045
A3GALT2   0.000000   0.000000   0.000000     0.9536797      1.925191     0.9640867    2.8911717    0.9671412    2.889548
A4GALT  293.882219 427.545732 311.207997  1333.0639420   1780.162943  1424.6011540 2322.4511550 3057.6903860 2822.636641
> head (z)
         PC9.DMSO1  PC9.DMSO2  PC9.DMSO3 PC9.osiacute1 PC9.osiacute2 PC9.osiacute3 PC9.osiDTP1 PC9.osiDTP2 PC9.osiDTP3
A1BG    0.05785255 0.12585758 0.11147612    0.20589165    0.34118237   0.145725237  0.56151065  1.35822392  0.36454014
A1CF    0.04479366 0.08797488 0.06432421    0.12682957    0.21740261   0.061651729  0.40790501  0.18576989  0.20245038
A2M     0.22256107 0.00000000 0.00000000    0.38322141    0.00000000   0.008938877  0.01042844  0.82971157  0.17603622
A2ML1   3.32689677 2.55084662 2.13824360    4.29182115    5.31615600   6.090532547 20.87617307 24.68509776 20.23241984
A3GALT2 0.00000000 0.00000000 0.00000000    0.02445986    0.04166992   0.024001175  0.08336936  0.02403215  0.06405611
A4GALT  7.96455166 9.07515080 8.00994894   34.19025646   38.53085318  35.465794001 66.96982827 75.97945792 62.57281248
```

3. You have to write all this on a Rscript and then mount the folder where this script it onto the docker and from the docker give the permission to run this file (`chmod +x`) and then run it

```
docker run -itv C:\Users\eleon\Desktop\data_analysis:/home/data_analysis r4:v.0.01
```

Here for example I mounted my data_analysis folder inside the home of the docker. Then I checked the file with nano and gave the `chmod +x` before running it with `Rscript nameofthefile`.

```
root@8b3894cacf65:/home/data_analysis# ls
fastqc_v0.11.9.zip                  L1sample1100k_S1_L001_R2_001.fastq        L4GSM4679532_P01
L1ra100k.R1_fastqc.html             L1sample1100k_S1_L001_R2_001_fastqc.html  L5dataset.txt
L1ra100k.R1_fastqc.zip              L1sample1100k_S1_L001_R2_001_fastqc.zip   L5.R
L1ra100k.R1.fastq.gz                L2dataset0.zip                            README.md
L1sample1100k_S1_L001_I1_001.fastq  L2dataset1
L1sample1100k_S1_L001_R1_001.fastq  L2dataset1.zip
root@8b3894cacf65:/home/data_analysis# nano L5.R
root@8b3894cacf65:/home/data_analysis# nano L5.R
root@8b3894cacf65:/home/data_analysis# chmod +x L5.R
root@8b3894cacf65:/home/data_analysis# Rscript L5.R
```

If you do `ls`, you should see your new file. Then you're done.

```
root@8b3894cacf65:/home/data_analysis# Rscript L5.R
root@8b3894cacf65:/home/data_analysis# ls
fastqc_v0.11.9.zip       L1ra100k.R1.fastq.gz              L1sample1100k_S1_L001_R2_001.fastq       L2dataset0.zip  L4GSM4679532_P01  L5.R
L1ra100k.R1_fastqc.html  L1sample1100k_S1_L001_I1_001.fastq  L1sample1100k_S1_L001_R2_001_fastqc.html  L2dataset1      L5cpm.txt         README.md
L1ra100k.R1_fastqc.zip   L1sample1100k_S1_L001_R1_001.fastq  L1sample1100k_S1_L001_R2_001_fastqc.zip   L2dataset1.zip  L5dataset.txt
root@8b3894cacf65:/home/data_analysis#
```

4. Install dev.tools for visualization of the experiment inside the docker (already installed in Rstudio in the first lesson). To do so you have first to install some system packages.
   a. run your docker by mounting the folder with all your data on it
   b. create a new .sh file with nano; write this in it:

```
apt-get update
apt-get update
apt-get install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
apt-get update && apt install -y libudunits2-dev libgdal-dev
apt-get update
apt-get -y install gfortran
apt-get -y install build-essential
apt-get -y install fort77
apt-get -y install xorg-dev
apt-get -y install liblzma-dev  libblas-dev gfortran
apt-get -y install gobjc++
apt-get -y install aptitude
apt-get -y install libbz2-dev
apt-get -y install libpcre3-dev
aptitude -y install libreadline-dev
apt-get -y install libcurl4-openssl-dev
```

c. Save and exit the file

d. Then type ./"nameofthefile".sh and run the shell command (as explained in the lesson 4 for the bash command of cellranger)

e. exit the docker and commit the changes

To install devtools packages, you have to create a new R script with the following command:

```
install.packages("devtools")
```

Then you save the Rscript.

You have to install devtools inside the docker and so you have to copy the Rscript file inside the docker with `docker cp path\of\the\source\file.R lastdockerimage:/home/`

Then you have to commit the changes with `docker commit`. Finally you do `docker run`, go inside home with `cd home`, give the permision to run the Rscript with `chmod +x nameofthefile` and then execute the Rscript with `Rscprit nameofthefile.` At the end you do again `docker commit` to save installation of devtools.

To see if it has correctly installed devtools (or any other package) go to `usr/local/lib/R/site-library` and do `ls`