Lesson 5 – data analysis

Prof. Calogero

SUMMARY OF THE PREVIOUS TOPICS

RNAseq → fastq → R1.

scRNAseq → fastq → R1 (UMI) + R2 (gene).

In a conventional fastQC test, I see how the reads are. Independently from the kind of approach I use, the next step is to map the reads on the genome (made of exons and introns) using STAR and, then, to perform an annotation of the genome, in order to associate a specific position of the genome to its function: this is done with GTF.

Biotype = which type of molecule is located in a specific region. This information is given by the European GTF, differently from the US one.

After GTF, RSEM is used: it calculates the relative expression of the isoforms and associates a portion of exon to the specific isoform that is considered.

In case of scRNA the situation is more complicated, due to PCR artifacts. In this case, isoforms are not considered, while exonic reads and transcript-associated reads are. To count genes, only the 3'-end is considered; to do so, the UMI is observed. It is random → for the same cDNA, there are not identical UMIs. At the end, I obtain a count table, which gives me the number of genes associated to a read (RNAseq) and the number of cells associated to a UMI (scRNAseq). Sc is 0-inflated: there are a lot of genes that do not have a UMI associated; this does not mean that they are not expressed, because the oligo-dT may not associate to the poly-A (this mainly happens in less expressed genes). The count table is in a sparse form, which is however not useful to us: I need to obtain a dense matrix, which is done by cellranger.

At this situation, there is a table for bulk and a table for sc.

NEW TOPIC

<u>Data normalization</u>

The ideal method is the conversion into transcripts per million, but I am going to obtain the cpm (counts per million). Why do I have to normalize? Because the number of reads mapped to each sample is never identical → normalization is useful to homogenize this number. Unnormalized data keep the variability of the sample.

In the slide there are 4 different genes: gene A is 2 kb, gene B is 4 kb, gene C is 1 kb and gene D is very large (10 kb). There are 3 replicates: regarding gene A, replicates 1 and 2 are very similar, while the third one is bigger; this also happens for the other genes. Gene D is very little expressed → no information will be obtained. There are 2 situations:

1) Replicate 3 is richer in reads than the others.
2) Gene A and B are one the double of the other, but this is due to their dimensions: a longer gene has a higher number of reads mapping on it.

In order to remove this effect, can work with different steps:

1) *Normalization for read depth*: I make the sum of the replicates for each gene the reads are divide the number by ten instead of a million, because my sample is smaller. I obtain 3.5, 4.5 and 10.6. Then, each value of the replicate is divided for the tens of reads → as I scale everything on the basis of the full number of reads present in a column, I remove the effect of a column to have more reads than

the other ones. The 3 replicates seem to have similar number of reads after this normalization, but there is still a difference between gene A and B. How can I do this correction?

2) *Normalization for gene length*: the replicates are divided to the gene length → the replicates are very similar to each other. This approach is called RPM, and it is summarized by a formula.

$$\widetilde{l}_i = l_i - \mu_{FLD} + 1$$

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

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.

- Xi = expression, in counts, of gene "I". it is divided by the length of the gene/1000.
- N = total number of reads present in a column; it is divided by 1 million.
- If I reorganize the data, I will obtain FPKM = expression level of the gene "I"/(length of the gene*number of genes present in a column) * 10^9

This normalization calculates the scaling factor by the columns and, then, the difference between the size of genes.

RPKM = reads per million per kilobase; FPKM is referred to the fragment: in this case I evaluate the effect in pair-end reads. Given 2 reads, the fragment is defined by the 2 reads together. I don't count the 2 reads independently but once, because they belong to the same transcript. In single-end sequencing I sequence on one side only, while in paired-end I sequence in both sides, counting the reads once.

$$\widetilde{l}_i = l_i - \mu_{FLD} + 1$$

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

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.

The way single cells are counted and normalized occurs through TPM = transcript per million. In this case, I divide each count by the gene size → I calculate the scaling factor for each column and normalize. Results are very similar, but this kind of normalization is very different from RPKM: the expression is divided by the length, and the normalized length is divided to the expression of all the other genes. A highly expressed gene will be highly ranked and vice versa. Ranking does not take into consideration the level of expression, but the relative position in the column: if I have 3 replicates for a condition, I expect that my gene roughly stays more or less (because of fluctuations); if it is differentially expressed, it will change the ranking position in the treated sample. So, despite normalization, I will still be able to see differences among the conditions. CPM (counts per million) works by ranking the expression of the gene with respect to the expression of all the other genes, without considering their length. This approach is noisier than TPM, but it still works well. CPM fits well if the gene length is similar, like in case of microRNAs, made of 20 nucleotides. If I work with RNAseq, CPM is clean as well, but it is better to work with TPM.

What makes the 2 normalization procedures different?

After normalization, if I do the sum of the columns, in RPKM I obtain 3 different values. If I imagine that the summary is the full size of a tart, each gene being a section of the tart, each tart has a different size → it is difficult to compare the genes with each other, because the pies have different sizes and they are not comparable. In TPM, on the other hand, the summaries are the same → the pies will have the same size, and samples will be comparable. This representation is useful to evaluate to see if there are differences between samples during an experiment.

<u>Exercise</u>

On Moodle there is an experiment related to an EGFR-mutated lung cancer cell line (deletion of exon 19), which is sensitive to the inhibitor Osimertinib. The 3 samples are called MOCK (cells are treated with DMSO, the diluent of Osimertinib), acute treated and DTP. I want to see if the 9 replicates are homogeneous to each other, but different from the others. How is the 21-day treated sample different from the one treated for 24 hours or from the NT one? Before looking to the genes changing, I want to represent the 9 samples on the

same space, which is not possible simply looking at the transcription profile. In the docker container, go to Bioconductor page and search for edgeR, which has a function allowing to convert the count matrix into a cpm one. I have to install this package on the docker and convert my matrix into a cpm-normalized table, saving it on a folder.

Command: x = read.table("name/of/the/file", sep = "\t", header = T, row.names = 1)

Set as working directory the folder where the file is! Row.names is the parameter that asks "which column has the row names?". The conversion is done with x = as.matrix(x)

On x I apply the cpm function. Once the table has been created, I have to convert it into a .cpm one.

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

Col.names considers the first element of the column as empty, in order to align the rows with the columns.

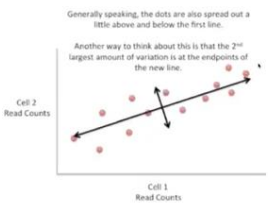
Once I install edgeR, I have to commit the changes!

On the R of the docker, install dev.tools, which can be loaded with library(dev.tools). If the output is 0, this means that the download did not go well → I have to go back and see what missed and install it.

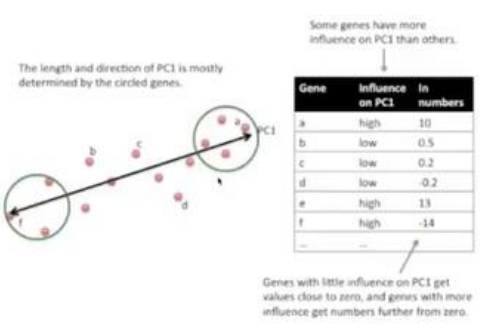
<u>Visualizing experiment data</u>

PCA = principal component visualization = 2D representation of a 3D space, still able to represent the information in a reduced space. What we will try to do is to represent the data (in my case, 9 columns) in a reduced space, which will tell me what are the similarities and the dissimilarities between samples. They are given by the variance of expression present among samples (big variance = different samples; small variance = similar samples). The 9 samples will be aggregated on the basis of the relative variance of expression.

- In this example, the dots are genes → cells are linearly related.
- In this other case, there is not a linear relationship.
- In this example, I don't need the 2 dimensions to represent similarities or dissimilarities of genes, because they are mainly related to cell 1. A 2D can be reduced to a 1D, in this case.



PCA example: there are various genes (a-i) and the reads associated to 2 cells. the genes are plotted for cell 1 and 2; the line indicates the direction of the greatest variance that can represent the dataset. It tells me what is the most important variation that I can obtain from these data. Once I identified the first variance, the second one is orthogonal. Practically, samples will be projected to the 2 dimensions of the variance, which have a different size (the 1st one is bigger, and it represents most of the sample; the 2nd one is smaller, and it represents what is left). Given 3 samples, 3 dimensions will be needed, and so on (in case of 4 samples, the 4th dimension is not visually seen).



Gene	Influence on PC1	In numbers
a	high	10
b	low	0.5
c	low	0.2
d	low	-0.2
e	high	13
f	high	-14
...	...	

Once the orientation of the diagonals has been defined, the names of the axes must be eliminated → the 2 diagonals are the x and y axes. When I change the reference, I don't look at the cells anymore, but at the variances. PC1 = principal component 1 (the most variant one); PC2 = principal component 2. As far as I move on the component, the information of the variance is progressively lost. If there are no differences between the 2 samples, the 2 components are very similar, and this is an indication that there are no differences among samples. The dimensions are given according to the number of samples I have (9 samples = 9 dimensions, but the first 2 or 3 are able to completely represent my

dataset). What I want from my experiment is that each cell is going to be represented by the variance of PC1 and the variance of PC2, but to reach this aim there is another step. Looking at the first dimension, not all the genes can estimate the variance: genes a and f are on the extremities, while b, c and d are outside. I can generate a score telling me how important is a specific gene in giving me the variance. F is going to be – while a is going to be + (being at the extremities); b and c have a small influence on the variance. This can be done on the second dimension too, but the weight given by each gene is different than the weight given on the first dimension. Depending on the component, the representation of the variance is given by the set of genes. Given the weight given by each gene in PC1 and PC2, I can represent cell 1 doing a summary of gene expression according to the weight: gene count*influence (in case of gene a, it is 10) + … for all the genes → I obtain a single value, which is associated to a specific cell and that mainly depends on the "heaviest" gene in the first dimension. The same thing is done in PC2. Given 12 for PC1 and 6 for PC2, I can locate cell 1 on the graph. Using this approach, I convert my information from genes to cells; a cell is the summary of the genes weighted by their importance in each of their specific component. If I make the same calculation for the 2nd cell, I see that it has a completely different PC1 than cell 1. Cell 3 is similar to cell 1 → little variance, but they are different from cell 2.

Weights are called "loadings", and the vector of loading is called "eigenvector". This will be the starting point for the next lesson.