

Data analysis

Lesson 9 – Prof. Calogero

Correction of the exercise of lesson 8

Step 1: understand what I want → step 2: organize things → start!

```
command.R
1 #Install in your Rstudio Bioconductor DESeq2 package.
2 #Following DESeq2 vignette: detect the genes called differential expressed compar
3 #DMSO versus acute osi.
4 #Thresholds: |logFC| ≥ 1 & adj-p-val ≤ 0.05
5 #DMSO versus cronic osi
6 #Thresholds: |logFC| ≥ 1 & adj-p-val ≤ 0.05
7 #Filter osi_log2CPM.csv using DE from acute osi and plot UMAP
8 #There is any difference with respect to what can be observed using all genes?
9 #Filter osi_log2CPM.csv using DE from cronic osi and plot UMAP
10 #There is any difference with respect to what can be observed using all genes?
11 #Filter osi_log2CPM.csv using combined DE from cronic/acute osi and plot UMAP
12 #There is any difference with respect to what can be observed using all genes?
13
14 #created acute and cronic folders
15 #placing bulk RNAseq acute and cronic in the corresponding folder
16 #created sc_ctrl and sc_acute folders
17 #placing sc data in the correspondig folders. N.B. These data are log2CPM transfo
18 |
```

Comment of the exercise

- Compare the differentially expressed genes in DMSO vs acute or DMSO vs chronic treatment. In both cases, I look for an adjusted p-value < 0.05 → I accept the possibility of having 5% of genes that are in my list of differentially expressed genes by chance. Moreover, I use an absolute logFC >= 1 or below 0.1. Why this threshold? This is mainly related to the sensitivity and the numerosity of our sample. Sensitivity is referred to qPCR; since I work with a small number of samples, the variability is high, so qPCR reads what I have. Many of the genes I selected will not be validated because the noise can be so high that they will disappear when I'll try to look at them. This threshold is reasonable to look at almost all my samples.
- I want to use these differentially expressed genes to see how the acute/chronic treatment will change the UMAP plot.

Data must be reorganized: since I expect to use data for many years, it is important to have them all together. In the folder "acute" there is a file "acute.txt", which contains the name of the gene associated to 3 acute conditions (the same is for chronic). Then there are the sc_acute and the sc_ctrl, where there are the log2CPM.csv files.

```
Terminal 1 - ~/Library/CloudStorage/OneDrive-Personal/Documents/courses/data_analysis_2022/2022-05-
1, "TTTGAGAGTACA", "TTGTTALGATG", "TTTATLGGTGG", "TTTATLTTTAT", "TTTATGAGAGTAA", "TTTATGAGAGTAA",
2, "TTTCAGGCTAGC", "TTTCCTGCTCTC", "TTTCGGTATGAC", "TTTCCTGCTATAC", "TTTCCTTTCGCTA", "TTTCCTTTCGCTA",
3, "TTTATGAGAGC", "TTTATTCACAA", "TTTTCAGATCCC", "TTTTCGGGAGC"
4, "TSPANG", 6.43504682561757, 5.78282156415226, 5.77757370360712, 5.61706697933612
5, 8.856704624346, 5.69869759220078, 6.40724618198128, 5.94947379310455, 5.81018812608675, 5.82944
6, 80334782, 5.7829428044512, 6.57244748552769, 5.85053587072132, 5.86874092630731, 5.8364645158033
7, 6.50876593982187, 5.90107578688662, 5.79124412648285, 6.48711923966604, 5.87970340553882, 5.604
8, 8529592465, 5.87260537597081, 5.80108790653671, 5.79693704225315, 6.42154835792246, 5.9063626302
```

If I want to see the first two rows of a file (the prof opens osi_log2CPM.csv), the command is `head -n 2 osi_log2CPM.csv`. The file is a comma-separated file and the information related to the differential expression is tab-delimited.

In sc analyses I have the EnsemblID:symbolID (in RNAseq data it is the opposite).

- I would like to reorganize data of the bulk, to render them identical to the sc ones, because it is convenient: file dimensions are different. I have to upload the acute.txt in R (to do so I can move to the folder or, as done below, use the ./acute/ line):

```
acute <- read.table("./acute/acute.txt", header=T, sep="\t", row.names=1)
```

- To take the row names:
`Rn <- rownames(acute)`
- Verify what has been loaded with `head(rn)`
- `Tmp <- strsplit(rn, ":")`
- `Head(tmp)` gives me a list made of 2 elements (as `rn`): the first one is the symbol, while the second one is the EnsemblID.

```
18 getwd()
19 acute <- read.table("./acute/acute.txt", header=T, sep="\t", row.names=1)
20 rn <- rownames(acute)
21 head(rn)
22 tmp <- strsplit(rn, ":")
23
24
24.1 (Top Level) :
```

```
[5]
1] "A3GALT2" "ENSG00000184389"

[6]
1] "A4GALT" "ENSG00000128274"
```

Apply works on matrices or on dataframes, while sapply works on vectors, and it returns vectors. L-apply can be used on whatever element, but it will return a list. Since I know that strsplit works because it provides me a list where each element is a vector of 2 characters, I can use sapply creating a function: the string function(x) indicates that the function must be applied to each content of the parameter. Sapply goes through the list, taking all the elements at a time and applying the function on them.

```
Tmp1 <- sapply(strsplit(rn, ":"), function(x)x[1])
```

This function will extract the first 2 elements of each element of the list and will save them in tmp1. Verify with head(tmp1).

```
19 getwd()
20 acute <- read.table("./acute/acute.txt", header=T, sep="\t", row.names=1)
21 rn <- rownames(acute)
22 head(rn)
23 tmp1 <- sapply(strsplit(rn, ":"), function(x)x[1])
24
24.1 (Top Level) :
```

```
Console Terminal Jobs
R 4.1.0 - ~/Library/CloudStorage/OneDrive-Personal/Documents/courses/data_analysis_2022/2022-05-11/data
[6]
1] "A4GALT" "ENSG00000128274"

tmp1 <- sapply(strsplit(rn, ":"), function(x)x[1])
head(tmp1)
1] "A1BG" "A1CF" "AZM" "AZM1"
5] "A3GALT2" "A4GALT"
1]
```

- Paste is a way how I combine elements: as seen in the command below, I assign to tmp the swapping of the 2 elements for the value x. then, out of the function, I send tmp out back (this last step is not necessary to be done because tmp is the last variable, so it will be returned as well: if I write another variable after tmp, the last variable will be given as output instead of tmp. Return allows me to see tmp instead of the last variable that I created). Then I run the function and I see that data are reorganized.

```
18 getwd()
19 acute <- read.table("./acute/acute.txt", header=T, sep="\t", row.names=1)
20 rn <- rownames(acute)
21 head(rn)
22 rn1 <- sapply(strsplit(rn, ":"), function(x){
23   tmp <- paste(x[2], x[1], sep=":")
24   return(tmp)
25 })
26
26.5 (Top Level) :
```

```
head(rn1)
1] "ENSG00000121410:A1BG"
2] "ENSG00000148584:A1CF"
3] "ENSG00000175899:AZM"
4] "ENSG00000166535:AZM1"
5] "ENSG00000184389:A3GALT2"
6] "ENSG00000128274:A4GALT"
```

- Now, I can build a function giving the same output on another variable. The parameter, in this case, is called my.rownames, which is going to be split. First, I execute the swapping function and then rn. Head(tmp) gives me the following output.

```

27
28 swapping <- function(my.rownames){
29   rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
30     tmp <- paste(x[2], x[1], sep=":")
31     return(tmp)
32   })
33   return(rn1)
34 }
35
36 tmp <- swapping(my.rownames-rn)
37 (TopLevel) :

```

```

> head(tmp)
[1] "ENSG00000121410:A1BG"
[2] "ENSG00000148584:A1CF"
[3] "ENSG00000175899:A2M"
[4] "ENSG00000166535:A2ML1"
[5] "ENSG00000184389:A3GALT2"
[6] "ENSG00000128274:A4GALT"
>

```

- As an alternative, swapping can get my dataframe as input, it reads the row names into a variable (my.rownames), it string-splits the variable with the ":" and reorganizes data in a way that the 1st element becomes the 2nd and vice versa. Finally, as my vector contains the row names, I reassign the new row names to the row names of my data frame. Then I return my data frame.

```

22
23 swapping <- function(my.df){
24   my.rownames <- rownames(my.df)
25   rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
26     tmp <- paste(x[2], x[1], sep=":")
27     return(tmp)
28   })
29   rownames(my.df) <- rn1
30   return(my.df)
31 }
32
33 acute.swapped <- swapping(acute)
34
35

```

```

R 4.1.0 - ~/Library/CloudStorage/OneDrive-Perso

```

	acute2	acute3
ENSG00000121410:A1BG	15	5
ENSG00000148584:A1CF	10	2
ENSG00000175899:A2M	0	0
ENSG00000166535:A2ML1	245	244
ENSG00000184389:A3GALT2	1	0
ENSG00000128274:A4GALT	1780	1424

```

>

```

- Since the function works, I can put it at the beginning of my script and apply it on the file of the chronic treatment, verifying the output with **header(chronic)**:

```

34 chronic <- read.table("../chronic/chronic.txt", header=T, sep="\t", row.names=1)
35 chronic.swapped <- swapping(chronic)
36

```

```

R 4.1.0 - ~/Library/CloudStorage/OneDrive-Personal/Docum

```

	chronic2	chronic3
ENSG00000121410:A1BG	54	16
ENSG00000148584:A1CF	7	9
ENSG00000175899:A2M	33	7
ENSG00000166535:A2ML1	993	912
ENSG00000184389:A3GALT2	0	2
ENSG00000128274:A4GALT	3057	2822

```

>

```

So, the first part is to swap the row names to obtain two tables that will be used in the next step. The next step is to find differentially expressed genes for the acute/chronic treatment.

- Load DESeq2 in R
- Take the acute.swapped and convert it into a matrix, because this format is needed for the next passages. Command: `acute.swapped <- as.matrix(acute.swapped)` in order to make all the elements of the same type. From `acute.swapped` I have to take the covariate elements, which are related to the column names. `Dimnames` gives a list of the elements of the matrix present in the columns. `Dimnames(acute.swapped)[[2]]`, instead, gives me a vector: the 1st square is the 1st element of the list, while with the double `[[]]` I inter the list. The output is a data frame. Finally, I can assign a factor to the conditions → 2 levels are obtained (ctrl and trt). Then, `rownames(coldata) <- dimnames(acute.swapped)[[2]]`
- The output is an object of the class “DESeq”.

```
40 acute.swapped <- as.matrix(acute.swapped)
41
42 coldata <- data.frame(samples=dimnames(acute.swapped)[[2]],
43                       condition=c(rep("ctrl", 3), rep("trt",3)))
44
45
```

samples	condition
1 ctrl1	ctrl
2 ctrl2	ctrl
3 ctrl3	ctrl
4 acute1	trt
5 acute2	trt
6 acute3	trt

```
38 #DE for acute
39 library(DESeq2)
40 acute.swapped <- as.matrix(acute.swapped)
41
42 coldata <- data.frame(samples=dimnames(acute.swapped)[[2]],
43                       condition=c(rep("ctrl", 3), rep("trt",3)))
44
45
```

samples	condition
1 ctrl1	ctrl
2 ctrl2	ctrl
3 ctrl3	ctrl
4 acute1	trt
5 acute2	trt
6 acute3	trt

```
4 dds <- DESeqDataSetFromMatrix(countData = acute.swapped,
5                               colData = coldata,
6                               design = ~ condition)
7
8
```

```
> dds <- DESeqDataSetFromMatrix(countData = acute.swapped,
+                               colData = coldata,
+                               design = ~ condition)
> dds
class: DESeqDataSet
dim: 18774 6
metadata(1): version
assays(1): counts
rownames(18774): ENSG00000121410:A1BG ENSG00000148584:A1CF ...
               ENSG00000159840:ZYX ENSG00000074755:ZZEF1
rowData names(0):
colnames(6): ctrl1 ctrl2 ... acute2 acute3
colData names(2): samples condition
```

- To see the final results, the command is `dds <- DESeq(dds)` and then `res <- results(dds)`. The string can be changed in this way:

```
37
38 #DE for acute
39 library(DESeq2)
40
41 swapped <- acute.swapped
42 swapped <- as.matrix(swapped)
43 coldata <- data.frame(samples=dimnames(swapped)[[2]],
44                       condition=c(rep("ctrl", 3), rep("trt",3)))
45 rownames(coldata) <- dimnames(swapped)[[2]]
46 dds <- DESeqDataSetFromMatrix(countData = swapped,
47                               colData = coldata,
48                               design = ~ condition)
49 dds <- DESeq(dds)
50 res <- results(dds)
51
```

- Since these commands work, I can put them in a unique function and execute DESeq on the acute and chronic data frames.

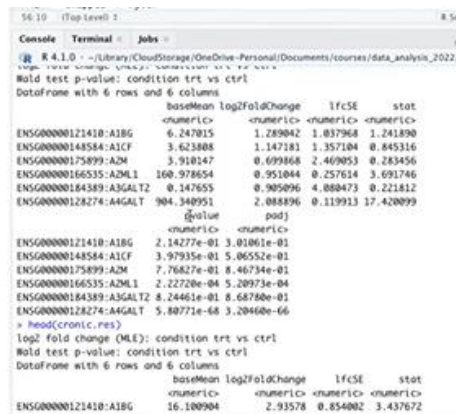
```
40 #executing deseq on dataframe made af 3 ctrl and 3 trt
41 mydeseq <- function(my.df){
42   swapped <- my.df
43   swapped <- as.matrix(swapped)
44   coldata <- data.frame(samples=dimnames(swapped)[[2]],
45                         condition=c(rep("ctrl", 3), rep("trt",3)))
46   rownames(coldata) <- dimnames(swapped)[[2]]
47   dds <- DESeqDataSetFromMatrix(countData = swapped,
48                                 colData = coldata,
49                                 design = ~ condition)
50   dds <- DESeq(dds)
51   res <- results(dds)
52   return(res)
53 }
54
55
```

```

54 = }
55
56 acute.res <- mydeseq(acute.swapped)
57 cronic.res <- mydeseq(cronic.swapped)
58
59

```

- I see the files with head. In the output, BaseMean is the number of mean counts associated to each gene (they are not transformed), the log2FC, the standard error, the statistics from which I derive the p-value and the adjusted one. The columns I am interested in are the adjusted p-value and the log2FC.



```

54.10 (Top Level: 1) 8.50
Console Terminal Jobs
R 4.1.0 - (Library/CloudStorage/OneDrive-Personal/Documents/courses/data_analysis_2022/)
log2 fold change (MLE): condition trt vs ctrl
Wald test p-value: condition trt vs ctrl
Dataframe with 6 rows and 6 columns
  baseMean log2FoldChange lfcSE stat
<numeric> <numeric> <numeric> <numeric>
ENSG00000121410-A1BG 6.247015 1.289042 1.037968 1.241890
ENSG00000148584-A1CF 3.623808 1.147181 1.357104 0.845316
ENSG00000175899-A2M 3.910147 0.699868 2.469053 0.283456
ENSG00000166535-A2ML1 168.978654 0.951044 0.257614 3.691746
ENSG00000184389-A3GALT2 0.147655 0.905096 4.080473 0.221812
ENSG00000128274-A4GALT 904.340951 2.088896 0.119913 17.420099
  p.value padj
<numeric> <numeric>
ENSG00000121410-A1BG 2.14277e-01 3.01061e-01
ENSG00000148584-A1CF 3.97935e-01 5.06552e-01
ENSG00000175899-A2M 7.76827e-01 8.46734e-01
ENSG00000166535-A2ML1 2.22720e-04 5.20973e-04
ENSG00000184389-A3GALT2 8.24461e-01 8.68700e-01
ENSG00000128274-A4GALT 5.80771e-68 3.20460e-66
> head(cronic.res)
log2 fold change (MLE): condition trt vs ctrl
Wald test p-value: condition trt vs ctrl
Dataframe with 6 rows and 6 columns
  baseMean log2FoldChange lfcSE stat
<numeric> <numeric> <numeric> <numeric>
ENSG00000121410-A1BG 16.100904 2.93578 0.854002 3.437672

```

- Now, I can add a simple filter: I must filter the rows in order to select the genes having a log2FC >= 1. This can be done with the following command. Which gives me the number of rows that contain the adjusted p-value < 0.01 and the number of rows with the log2FC, intersect selects the same elements. Moreover, I have to discard everything that is associated to "NA" in my data frame. The meaning of the command with the "!" is: is NA present in my file? If yes (which corresponds to the exclamation point), discard it.

```

dds <- DESeq(dds)
res <- results(dds)
res <- res[!is.na(res$log2FoldChange),]
res <- res[which((res$padj <= 0.05) & (abs(res$log2FoldChange) >= 1)),]
return(res)
}

acute.res <- mydeseq(acute.swapped)
cronic.res <- mydeseq(cronic.swapped)

```

- The dimensions of the two files changed. When I do differential expression this way, that can happen is that I discard a gene in one group because the threshold changes. Example: in the acute treatment the log2FC is 0.999 → the gene will be discarded. In the chronic, the log2FC is 1.1 → the gene is maintained. The best way would be to "treat" the data in the same way, with a system that ranks them together. This can be done only if needed: in my case, the goal is to take the 4000 genes present in the acute treatment and filter the sc to see if I get a separation, and then I have to do the same thing on the chronic treatment. If chronic and acute treatments are partially overlapped is not a problem for me until now.
- Now, I have to filter the treated sample of the sc using the list of differentially expressed genes in the acute or chronic treatment. I load the csv file and look at its dimensions. Then, I filter the rownames on the bases of the sc data. The dimensions

```

#working on scRNA
sc <- read.csv("../sc_acute/osi_log2CPM.csv", header=T, row.names=1)#779
sc.acute <- sc[which(rownames(sc) %in% rownames(acute.res)),]

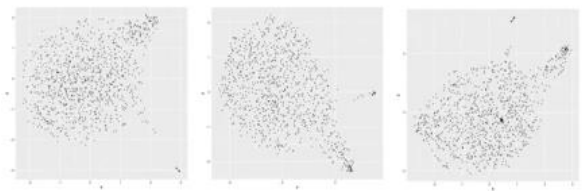
```

are smaller. Then, the same thing can be done on the chronic treatment file and look again at the dimensions.

- The next step is data visualization. The matrix must be transposed because the rows need to be the cells: umap is done on the cells, so rows must be the cells; random_state = 111 is decided arbitrary; n_epochs = 1000 guarantees a good conversion of the algorithm; min_dis = 0.05 and neighbours = 15. Then, I create a dataframe and do the ggplot2. I run the command on the chronic treatment and on the full sc dataset, in order to compare the differences.

```
66
67 library(umap)
68 library(ggplot2)
69 umap.out <- umap(t(sc.acute), random_state=111, n_epochs = 1000, min_dist=0.
70 f=data.frame(x=as.numeric(umap.out$layout[,1]),y=as.numeric(umap.out$layout[
71 #plotting UMAP
72 sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.5)
73 #pdf("ctrl_umap_m0.05_n10.pdf")
74 print(sp)
75 #dev.off()
76
```

- The 3 pdf files (sc, acute and chronic) are put in a PowerPoint slide: the set of genes that is present in the sc file is less separated than in the other 2 files. Another thing that can be done is to take the genes in common (chronic + acute) or the acute or chronic genes only: this may help to hypothesize if the set of cells that seem to separate from the others has a meaning or not.
- Once rownames(acute) has been reorganized, in order to see only the symbols I do write the command below. Writelines writes, on each row, the symbol I generated.



```
9
9 #de results
1 acute.res <- mydeseq(acute.swapped)$4027
2 tmp <- apply(strsplit(rownames(acute.res), ":"), function(x)x[2])
3 writelines(tmp, "acute_sym.txt")
4
```

Omics Net offers different possibilities to visualize interactions related to functional relationships among the things I analyse. I load my genes, whose ID type is "official gene symbol" → proceed. I can investigate protein-protein interactions, miRNA-gene, metabolite or TF-gene (which is the best one in this case). TRRUST validates the experimental data for the relationship between TF and targets. As output, there are different networks, which can be observed in a 2D or 3D way. In the Node Explorer there are the nodes associated to each gene. I can look to the TF characterized by the highest connectivity as the most interesting one for my experiment. Omics Net helps me to identify upstream regulators of the list of genes that are upregulated in my analysis. In "Reactome" I can see which sets of genes are affected by the drug treatment.