SysBio Day 1 Hands On

Contents

Note

Here you'll find the code for the exercises of day 1. Do not worry if your code from following the live coding session is not exactly the same, there are always several ways to perform the same operation. If you find trouble trying to run anything when coding, remember that the documentation and/or Google are your best friends.

1) Installing packages

Dependencies for today's exercises include:

Package	source	Description
`biomaRt`	Bioconductor	Get molecular annotation for features (gene, proteins, etc)
`Biobase`	Bioconductor	Handle Expression-Sets (gene expression data)
`GEOquery`	Bioconductor	Get data from GEO
`qusage`	Bioconductor	Read GMT files

Depending on your R version, bioconductor packages are installed in different ways. You can check you R version using `R.version$version.string`

```
# For R version < 3.5
source("https://www.bioconductor.org/biocLite.R")

biocLite("biomaRt")
biocLite("Biobase")
biocLite("GEOquery")
biocLite("qusage")
```

```
# For R version > 3.5
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
```

```
BiocManager::install("biomaRt")
BiocManager::install("Biobase")
BiocManager::install("GEOquery")
BiocManager::install("qusage")
```

Let's now load the libraries:

```
library("biomaRt")
library("Biobase")
library("GEOquery")
library("qusage")
```

2) Loading data set from GEO

You can visit the GEO entry for `GSE116436` at

> https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE116436

`GSE116436` is the GEO series of the whole data set. It comprehends transcriptome profiling upon **60 cell lines using 15 anti-cancer agents** with different time points (2, 6, 24 hrs) and drug concentrations (incl. null concentration, i.e. control). A short description and experimental overview are shown. For further details, you can read the original article:

> Monks A et al. The NCI Transcriptional Pharmacodynamics Workbench: A Tool to Examine Dynamic Expression Profiling of Therapeutic Response in the NCI-60 Cell Line Panel. Cancer Res 2018 Dec 15;78(24):6807-6817. PMID: 30355619

You can also click on one sample (transcriptome profile) to see **sample metadata**.

> For instance, GSM3231645

Importantly, this **data set is large (7.5k samples)**. Thus the GEO series is stratify in subseries by each drug:

GEO accession	Drug
GSE116437	5-Azacytidine
GSE116438	bortezomib
GSE116439	cisplatin
GSE116440	dasatinib
GSE116441	doxorubicin
GSE116442	erlotinib
GSE116443	geldanamycin
GSE116444	gemcitabine
GSE116445	lapatinib
GSE116446	paclitaxel
GSE116447	sirolimus
GSE116448	sorafenib
GSE116449	sunitinib
GSE116450	topotecan
GSE116451	vorinostat

Let's start with one subset of the data to understand how it works.

We choose the `GEO accession number` of one subserie. For instance, we choose lapatinib (GSE116445).

We use the following command line to download the data

```
lapatinib <- getGEO("GSE116445", destdir="/tmp", AnnotGPL=TRUE)[[1]]
```

```
## Warning: 1168 parsing failures.
##   row           col           expected               actual         file
## 10161 UniGene title 1/0/T/F/TRUE/FALSE Clone HQ0117 PRO0117 literal data
## 10161 UniGene ID    1/0/T/F/TRUE/FALSE Hs.670442            literal data
## 10179 UniGene title 1/0/T/F/TRUE/FALSE Transcribed locus    literal data
## 10179 UniGene ID    1/0/T/F/TRUE/FALSE Hs.621370            literal data
## 10347 UniGene title 1/0/T/F/TRUE/FALSE Transcribed locus    literal data
## ..... ............. .................. ..................... .............
## See problems(...) for more details.
```

3) Exploring the data object

After this, we have a R object named `lapatinib`. This object contains the gene expression data and sample metadata. To explore this object, we can use:

```
class(lapatinib)

# It is a Expression-Set, which is a special object to store 'expression data'
is(lapatinib)

# What is the dimension of this bunch of data?
dim(lapatinib)

# Which attributes does the object have?
attributes(lapatinib)
```

How do the features and samples look like?

```
# Features = probes
head(featureNames(lapatinib), 5)

# Samples = transcriptome profiles (under certain experimental conditions)
head(sampleNames(lapatinib), 5)
```

Wrap-up

You can see that both `Features` and `Samples` are described throughby **identifiers**...

In order to analyse and interpret this data, we need to map this identifiers to the genes and experimental conditions of each sample.

4) Subsetting the data

For the day 2 practice, we have subset the data set in a specific manner. We have choosen the relevant profiles as those perturbed with the highest drug concentration and the untreated samples (as control) and latest time point, expecting to see the strongest patterns. To do so, we first get the sample metadata for these two features: drug concentration and time point

```
# the sample metadata contains the drug concentrations within tags
sample_tags = as.character(lapatinib$title)

## Drug concentration
# Extract the part describing drug concentration
```

```
drug_conc = sapply(sample_tags, function(tag){
  strsplit(tag, split="_")[[1]][3]
  })
# Replace nanoMolar with empty string
drug_conc = gsub("nM$", "", drug_conc)
# Convert to numbers
drug_conc = as.numeric(drug_conc)
# Max conc
drug_conc.max = max(drug_conc)
# Min conc
drug_conc.min = min(drug_conc)

## Time point: basically the same
drug_tpoint = sapply(sample_tags,function(tag){
  strsplit(tag, split="_")[[1]][4]
  })
drug_tpoint = gsub("h$", "", drug_tpoint)
drug_tpoint = as.numeric(drug_tpoint)
drug_tpoint.max = max(drug_tpoint)
```

Finally we subset the data

```
selected_samples = (drug_conc == drug_conc.max | drug_conc == drug_conc.min) & drug_tpoint == drug_tpoi
lapatinib = lapatinib[, selected_samples]

dim(lapatinib)
```

5) Access to the gene expression and annotation data

Expression-Sets are special objects to store gene expression data and metadata associated to both genes and samples. All this information is store in a unique `R object`.

The Gene Expression data: Gene expression data is store as a quantitative matrix. This matrix contains measurements of the gene-level expression. **Metainformation for samples and features**: Two `data.frames` contains metadata for samples and features.

To access to the gene expression data, we use `exprs()`.

```
e_data = exprs(lapatinib)
head(e_data)
```

`Expresion-Set` objects store the metadata information within the object itself. To access to this information, there are two main functions from `Biobase` package:

- `pData()` : Access to phenotypeData. The phenotypeData is the sample metadata described in the GEO entry of each sample.
- `fData()` : Access to featureData. The featureData is complementary information for each one of the features (probes).

Let's try them out!

Sample metadata

```
p_data = pData(lapatinib)
head(p_data)
```

Feature metadata

```
f_data = fData(lapatinib)
head(f_data)
```

6) Identifier conversion

Microarrays are High-Throughput platforms throughby gene-level expression is measured using one or a few probes for the same gene.

For instance, we could know which platform was used in our dataset:

```
# GEO platform identifier
annotation(lapatinib)
# Or using the sample metadata
unique(pData(lapatinib)$hyb_protocol)
```

In order to create a fit matrix of Genes x Samples, we are going to collapse the intensity of one or more probes into one single value that represents the expression of the gene. For this, we need to create a vector that map each probe to its corresponding gene

```
# Loading Mart object for Ensembl Homo sapiens data
mart = useMart('ENSEMBL_MART_ENSEMBL', 'hsapiens_gene_ensembl',
               host = "www.ensembl.org", ensemblRedirect=F)

query = featureNames(lapatinib)
matching_table = getBM(c('affy_hg_u133a_2', 'external_gene_name'),
                       filters='affy_hg_u133a_2', values=query, mart=mart)
colnames(matching_table) = c('probe', 'gene')
```

Workaround (if Ensembl servers are down)

```
# Quick and dirty
matching_table = data.frame(1:length(f_data$ID))
matching_table$probe = f_data$ID
# Getting unique IDs (first gsymbol)
matching_table$gene = gsub('[///].*',  '', f_data$`Gene symbol`)
matching_table$gene = gsub('[-].*',  '', matching_table$gene)
matching_table = matching_table[, -1]
```

Renaming columns

```
# Convert columns names to something readable
our_columns <- colnames(e_data)
our_columns <- p_data[our_columns, 'title']
colnames(e_data) <- our_columns
```

Collapsing probe intensities to gene-level expression

```
# Transform into a data.frame
df = as.data.frame(e_data)

# Matching Identifiers
gsymbols = as.character(sapply(rownames(df), function(x){
  aux = matching_table[matching_table$probe == x, 'gene'][1]
```

```
  }))

# We add another column to the data.frame which describe the corresponding gene
df$gene = gsymbols

# Aggregate values for the same gene by the average
df2 = aggregate(. ~gene, data=df, mean)

# Replace row.names with genes
rownames(df2) = as.character(df2$gene)

# Removing unmapped GeneSymbols
df2 = df2[!rownames(df2) == '', ]

# Remove the column with the genes
df2 = df2[, which(colnames(df2) != "gene")]
```

7) Saving the data

```
# Saving the expression data
write.csv(df2, 'my_data.csv')

# Let us save the annotations for the upcoming days
write.csv(p_data, 'my_p_data.csv')
```

8) Retrieving pathway annotation

Let's take as an example a curated gene set (C2) from MSigDB.

Download the `.gmt` file from the Biocarta curated pathways and subset the list of genes corresponding to the MAPK pathway (`BIOCARTA_MAPK_PATHWAY`). > Alternatively, you can search directly on the website and dowload only that > specific list of genes(here)

```
gsets = read.gmt('c2.cp.biocarta.v7.1.symbols.gmt')
gset = gsets[['BIOCARTA_MAPK_PATHWAY']]
```

9) Obtaining PPI network

Now we will download the protein-protein interaction network from OmniPath.

```
url = 'http://omnipathdb.org/interactions?genesymbols=1'
opath = unique(read.table(url, sep='\t', header=T)[, c(3, 4)])

# Subsetting the PPI to members of the pathawy
pathway = opath[opath[, 1] %in% gset & opath[, 2] %in% gset, ]
```

10) Mapping expression to pathway

Finally, we extract the median expression of all nodes in our pathway and save them in a file as well as the network itself.

```
unique_genes = unique(c(as.character(pathway[, 1]),
                        as.character(pathway[, 2])))
```

```
measurements = data.frame()

# Extracting  median of measurements of pathway members
for(g in unique_genes){
  measurements[g, 1] = rowMedians(as.matrix(df2[g, ]))
}

# Storing the results
write.table(pathway, 'pathway.txt', sep='\t', quote=F, row.names=F)
write.csv(measurements, 'measurements.csv', quote=F)
```