RNA sequencing Project_Edx_nishant20102014

Nishant Upadhyay

Monday, October 20, 2014

This project consists of working through a somewhat realistic scenario where we try to analyze the results from an RNA sequencing experiment using various BioConductor packages for R.

Note :some codes not working.I have written them outside the r-chunk

WHAT IS RNA SEQUENCING?

Before RNA sequencing, RNA is extracted from a sample and usually treated with a protocol to enrich for a certain population of RNA molecules, such as the protein-coding messenger RNAs or the micro-RNAs. (This is done to get rid of ribosomal RNA molecules that will often comprise upwards of 90% of the RNA molecules).

The selected RNA population is then reverse transcribed to cDNA, which is fragmented and gets prepared for sequencing by attaching adapters - molecular "handles" that are recognized in the sequencing process.

The resulting "cDNA library" can be sequenced in a process that will randomly sample fragments from the library and report their nucleotide sequences. There will typically be several millions of sequences (usually called reads) of 50-200 nucleotides (although they can be much longer for certain lower-throughput platforms).

When the sequences have been obtained from the instrument, the following steps are typically performed (assumed that there is a reference genome for the organism that the sample came from; reference-free or de novo analysis is a very interesting topic that unfortunately falls outside the scope of this course.):

1. Mapping or "alignment" to a reference genome. In this step, the raw sequence files (often in FASTQ format) are matched to a reference genome to find out where they may have originated from. The output is an alignment file (often in SAM or BAM format - the latter is just a compressed, binary version of the former) that records where each read matches the reference genome.
2. Counting or some other type of quantification. In this step, the number of reads matching each gene (or some other feature such as exon) is calculated with the help of a feature annotation file (often in GTF format) which details where different genes or other features start and end in the genome.
3. Differential expression analysis is often performed at this point if the aim of the experiment is to compare different groups, such as cases and controls in a disease study, treatments vs controls, different tissues etc.
4. Gene ontology analysis and pathway analysis are often performed on the results of the differential expression analysis.

In the core version of this exercise, we will start from step 3 which means that we will perform a differential expression analysis between cancerous and healthy tissue, visualize the results and finally do a very simple gene ontology and pathway analysis on the differentially expressed genes.

In the extended version, which will not work on Windows, we will also do steps 1-2: download raw data files, align (map) them to a reference genome, and count the number of sequences coming from each annotated gene. The idea is to show you a fairly complete workflow for going from raw data to some sort of biologically relevant results, with everything done inside of R.

WHAT IS BIOCONDUCTOR?

BioConductor is a project that aims to develop and make available R functionality for bioinformatics, that is, the computational analysis of biological problems. It's a kind of umbrella that collects R libraries (also called packages) from people and research groups all over the world. BioConductor has a peer review process that each package has to undergo in order to get into the project.

In order to install BioConductor packages, you use a procedure that is a bit different from the usual install.packages() or R CMD INSTALL (or whatever you use for standard R packages). You start by writing:

source("http://bioconductor.org/biocLite.R")

Subsequently you can install the BioConductor packages that you are interested in by giving the command biocLite("PACKAGE NAME").

To load a package after you have installed it, just type library("PACKAGE NAME"), as usual.

For this assignment, we will need to install the BioConductor packages DESeq2, biomaRt, org.Hs.eg.db and topGO, as well as the non-BioConductor R packages calibrate and pheatmap. Those brave souls that want to do the downloading, mapping and counting themselves also need the Rsubread package.

DIFFERENTIAL EXPRESSION ANALYSIS

The core exercise starts here!

Analysis of differentially expressed genes based on RNA-seq is a field undergoing a rapid development. The R-based *DESeq2 package* is one of the methods considered to be the most reliable at this point in time. Other popular R-based and popular packages are edgeR, samr and BaySeq.

Install and load DESeq2 by doing:

source("http://bioconductor.org/biocLite.R") biocLite("DESeq2")

```
library("DESeq2")
```

```
## Loading required package: GenomicRanges

## Warning: package 'GenomicRanges' was built under R version 3.1.1

## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##     xtabs
##
## The following objects are masked from 'package:base':
##
```

```
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, do.call, duplicated, eval, evalq, Filter, Find, get,
##     intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rep.int, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unlist
##
## Loading required package: IRanges

## Warning: package 'IRanges' was built under R version 3.1.1

## Loading required package: GenomeInfoDb
## Loading required package: Rcpp

## Warning: package 'Rcpp' was built under R version 3.1.1

## Loading required package: RcppArmadillo

## Warning: package 'RcppArmadillo' was built under R version 3.1.1
```

There is an introduction to DESeq2 that you can access from within R by giving the command

vignette("DESeq2")

This shows a so-called vignette; a kind of documentation format used for R packages. It's better to open vignettes that come with the package you have loaded in R rather than googling for the documentation; the vignette describes the version of the package that you actually have, and some R packages change quickly.

Apart from the steps described here, the vignette contains more details about how DESeq2 works and how to interpret the various plots generated by the program. But right now we are mostly interested in getting to a list of genes that are differentially expressed between the tumor and healthy samples.

If you haven't done the optional exercise above, download the file called **count_table.txt** from the link: https://courses.edx.org/c4x/KIx/KIexploRx/asset/count_table.txt and read it into your R session using the following command:

```
setwd("H:/Elearning/courseera videos/edX_KIx KIexploRx Explore Statistics with R/week5_visiting the res
counts <- read.delim("count_table.txt")
```

We'll start by defining the information about the experiment that we want to use in the analysis. Assuming that we are using the table we just created (let's assume it's called counts) with samples named C02, C03, C06, N02, N03 and N06, where the first three are tumor samples and the others samples from healthy tissue, we can express the experimental information as follows:

```
exp.info <-data.frame(patient=rep(c("02","03","06"),2), condition=c(rep("C",3),rep("N",3)))

rownames(exp.info) <- colnames(counts)
```

That is, there are three patients (02, 03 and 06) and two conditions (C and N). The row names in the exp.info table need to be the same as the column names in the count table so that DESeq2 can match them up with each other. We will assume that we have made sure that the samples are in the same order in both tables!

```
#Now we can create a new DESeqDataSet object:

dds <- DESeqDataSetFromMatrix(countData=counts, colData=exp.info,design=~patient+condition)
```

After executing this command, you might get an information message from R starting with "Usage note". Don't worry - it's normal.

The design parameter described the design of the experiment, that is, which columns in the experiment info table we want to take into consideration. These are called factors. By default, the factor that is listed last (condition in our case) will be the one that results will be reported for by default. (Don't worry of you don't understand what that means.)

Sanity checking the data

It's common to visualize gene expression data using heat maps diagrams which represent values in a matrix with a grid where each tile has a color that reflects its value. This can be done using R's image() command, but it's often more interesting to use pheatmap(), which performs a clustering (grouping) of the matrix both on the x and y axes. The DESeq vignette/user guide recommends variance stabilization of the data before clustering and visualization. Many statistical methods, such as linear (Pearson) correlation, assume that the variance of the data does not depend on the mean. So we should start by doing:

```
library(pheatmap)
```

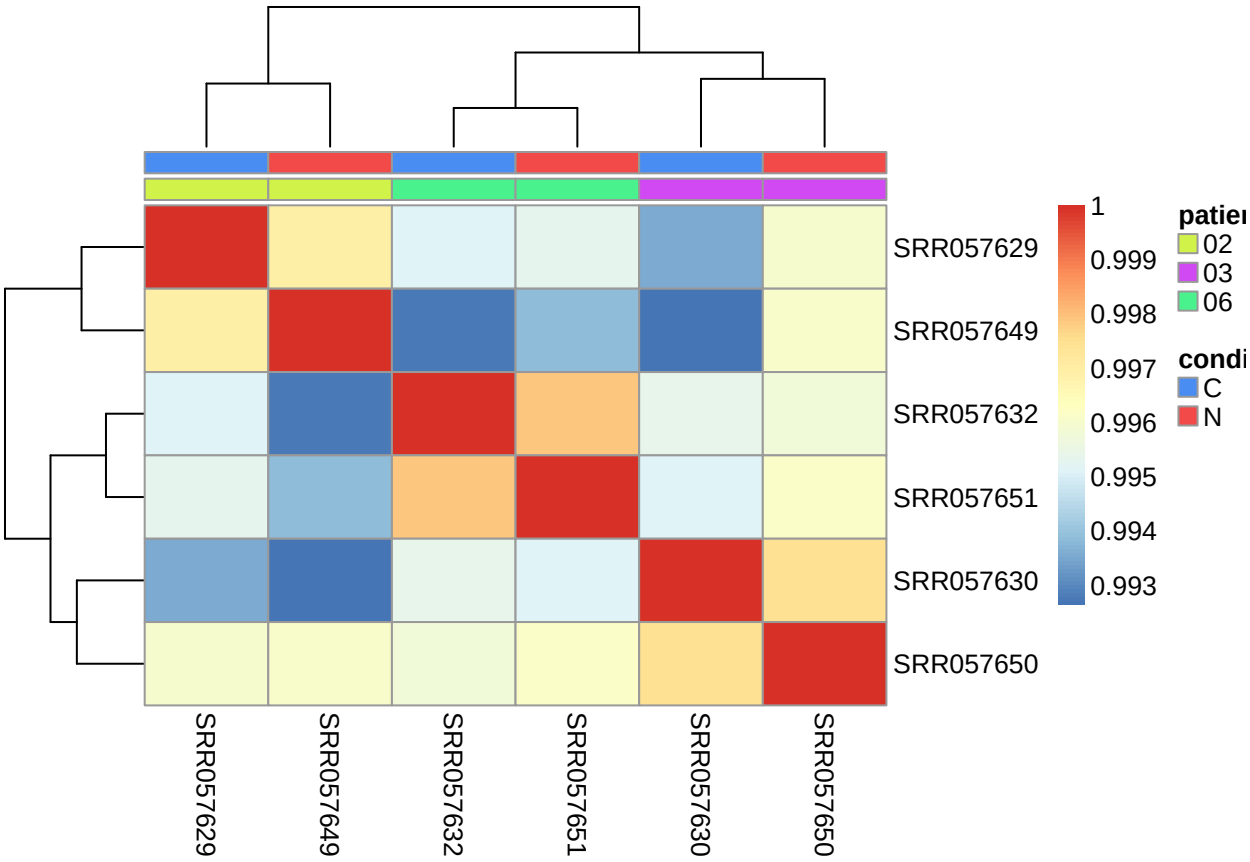
```
## Warning: package 'pheatmap' was built under R version 3.1.1
```

```
r <- rlogTransformation(dds)
```

which performs a regularized log transform that amounts to one way of doing a variance stabilization. In some versions of DESeq2, this command is called rlog, so try that if rlogTransformation doesn't work.

Now we can look at the correlations between the six samples:

```
pheatmap(cor(assay(r)),annotation=exp.info)
```

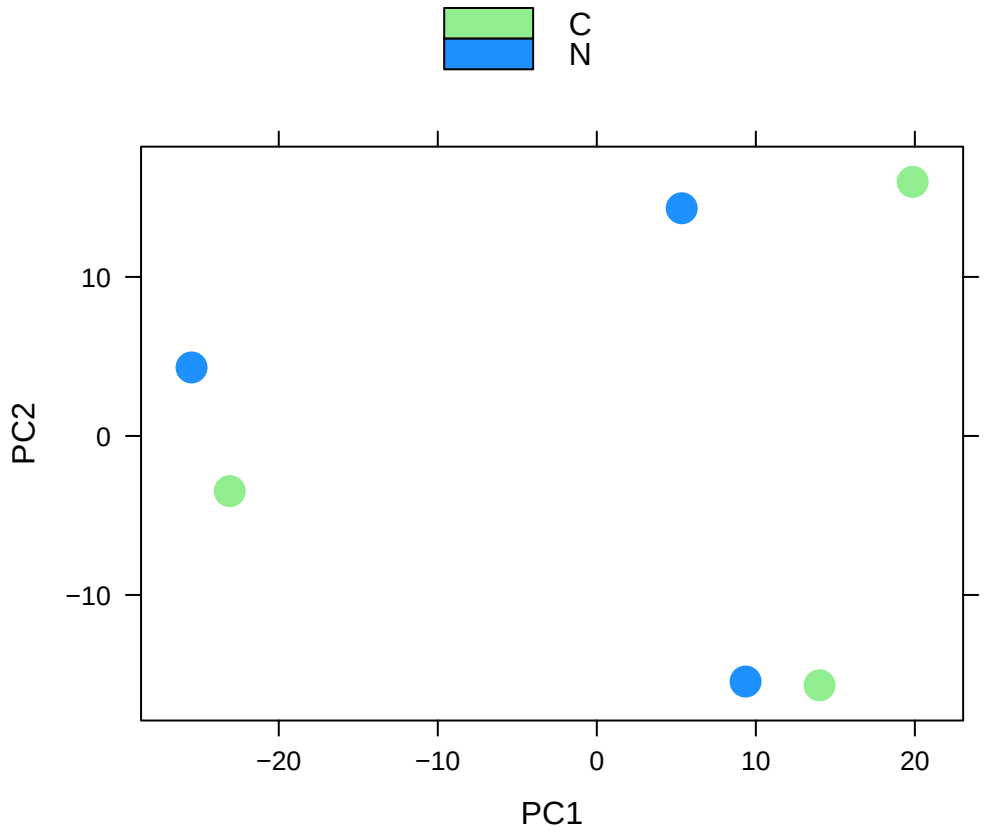


What does this heat map tell you, in general terms, about the gene expression profiles of the six samples? Note that the annotation option makes pheatmap annotate the heat map with patient and condition information.

Do the samples cluster **more by individual**

Another way to check how the expression levels in the samples relate to each other would be to use a principal component analysis plot:

```
print(plotPCA(r,intgroup="condition"))
```



which will draw a PC plot where samples are colored by condition (tumor or normal). By changing "condition" to "patient", you can get samples colored by patient ID instead.

Differential expression analysis

The actual differential expression testing between genes is done by

```
dds <-DESeq(dds)
```

```
## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

This function looks simple but actually contains many steps which you can read more about in the vignette if you want. We can obtain differential expression results by:

```
res <- results(dds)
```

This will yield a table with pvalues and pvalues adjusted for multiple comparisons for each transcript. We can filter this table on, for example, the adjusted p value, and keep those transcripts where this value is less than 0.01:

```
res.sig <- res[which(res$padj<0.01),]
res.sig <- res.sig[order(res.sig$padj),]
```

Now we can write the gene list to a file:

```
write.table(res.sig,file="significant_diff_expr.txt", sep="\t",quote=F,row.names=F)
```

Now run the differential expression analysis commands again, but this time omit the "patient" factor from the design formula, that is, with condition as the only factor:

```
dds.alt <- DESeqDataSetFromMatrix(countData=counts, colData=exp.info,design=~condition)
dds.alt <-DESeq(dds.alt)
```

```
## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

```
res.alt <- results(dds.alt)
res.alt.sig <- res.alt[which(res.alt$padj<0.01),]
```

Check how many differentially expressed genes you get now.

With only condition as a factor rather than both patient and condition, does DESeq2 determine that you get–**less differentially expressed genes**

Getting common gene symbols and names

You might want to get standard gene names and gene symbols rather than the ENSEMBL ID:s. This is possible to do by parsing the annotation GTF file that you have downloaded or by using the org.Hs.eg.db package, but we will show another way that uses biomaRt package from BioConductor. Install it as usual:

biocLite("biomaRt")

```
library(biomaRt)
```

Tell biomaRt that we want use the ENSEMBL databases for the human genome:

```
mart <- useDataset("hsapiens_gene_ensembl", useMart("ensembl"))
ensembl_genes <- rownames(res.sig)
```

Now fetch common gene symbols (called "external IDs" here) corresponding to each ENSEMBL ID:

output <- getBM(filters="ensembl_gene_id", attributes=c("ensembl_gene_id", "external_gene_id"), values=ensembl_genes,mart=mart)

Now you can add the common names to your table of significantly differentially expressed genes:

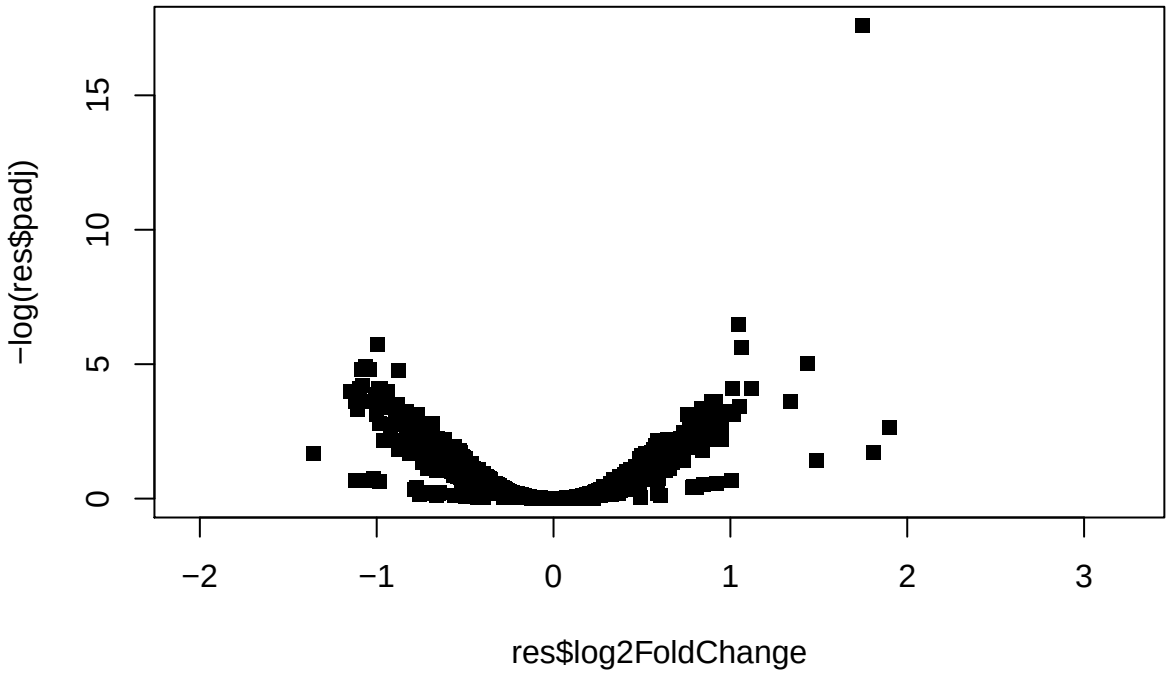
res.sig$GeneSymbol <- output$external_gene_id

Plots

A common way to look at differential expression between two conditions is to use a volcano plot. Here, genes are plotted as points in a diagram where the y axis represents the negative logarithm of the p value for the gene being differentially expressed meaning that the higher up a gene is in the plot, the better (lower) the p value and the x axis represents log fold change (expression ratio) for the gene between the conditions. The fold change varies between 0 and infinity, but the log fold change can take on any positive or negative values. This means that the points will lie distributed (often fairly symmetrically) around zero on the x axis. These plots will often show a characteristic pattern that resembles an erupting volcano.

Start by plotting all the data points in this way:

```
plot(res$log2FoldChange,-log(res$padj),pch=15)
```




```
# pch=15 turns the points into squares
```

Then replot the differentially expressed genes (padj < 0.01) in red:

points(res.sig$log2FoldChange$, $-log(res.sig$padj),col="red",pch=15)

If you want to add gene names to the plot, you can use the textxy() function from the calibrate package.

install.packages("calibrate")

```
library("calibrate")
```

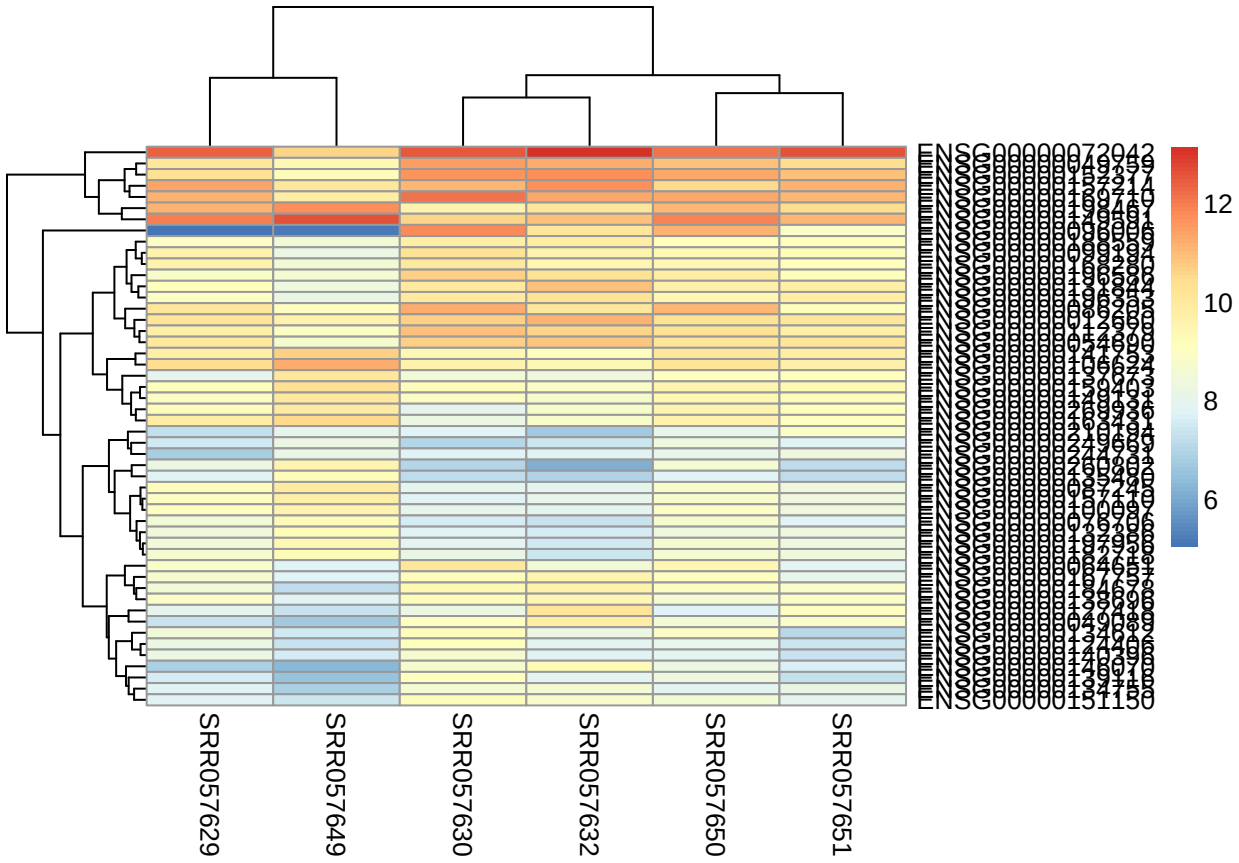
```
## Warning: package 'calibrate' was built under R version 3.1.1
```

```
## Loading required package: MASS
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:biomaRt':
##
##     select
```

textxy(res.sig$log2FoldChange, -log(res.sig$padj),res.sig$GeneSymbol)

We can make a slightly different heat map that shows the expression level in each sample for 50 differentially expressed gene

```
select = order(res$padj)[1:50] # choose the ones with lowest padj
pheatmap(assay(r[select,],scale="none"))
```



Gene ontology and pathway analysis

Finally, let's try some simple gene ontology (GO) and pathway analysis. There are quite a few BioConductor packages for this (as you will find if you spend a few minutes browsing http://www.bioconductor.org/packages/2.12/bioc/) but we will take a brief look at one of those, topGO.

Start by installing the two packages in the usual BioConductor way.

biocLite("topGO")

topGO

topGO is a Gene Ontology enrichment analysis package. It needs us to provide a list of all the genes we have analyzed, and a measure of their significance in the DE analysis. We will use the adjusted p value, padj, for this.

```
library(topGO)
```

```
## Loading required package: graph
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
##
## Loading required package: GO.db
## Loading required package: AnnotationDbi

## Warning: package 'AnnotationDbi' was built under R version 3.1.1

##
## Attaching package: 'AnnotationDbi'
##
## The following object is masked from 'package:MASS':
##
##     select

## Warning: package 'RSQLite' was built under R version 3.1.1

## Loading required package: DBI

## Warning: package 'DBI' was built under R version 3.1.1

##
## Loading required package: SparseM

## Warning: package 'SparseM' was built under R version 3.1.1

##
## Attaching package: 'SparseM'
##
## The following object is masked from 'package:base':
##
##     backsolve

##
## groupGOTerms:    GOBPTerm, GOMFTerm, GOCCTerm environments built.
```

```
##
## Attaching package: 'topGO'
##
## The following objects are masked from 'package:GenomicRanges':
##
##     score, score<-
##
## The following objects are masked from 'package:IRanges':
##
##     members, score, score<-
```

```
all <- res$padj
names(all) <- rownames(res)
all <- na.omit(all)
```

Then we can create a topGOdata object which we can then manipulate in different ways. You can choose between looking at CC (cellular component), MF (molecular function) or BP (biological process).

biocLite("org.Hs.eg.db")

```
library("org.Hs.eg.db")
```

```
##
```

```
GOdata <- new("topGOdata",ontology = "CC", allGenes = all, geneSel=function(p) p < 0.01, description ="
```

```
##
## Building most specific GOs ..... ( 813 GO terms found. )
##
## Build GO DAG topology .......... ( 1046 GO terms and 2043 relations. )
##
## Annotating nodes ............... ( 2898 genes annotated to the GO terms. )
```

Then we can perform a test to assess significant overrepresentation of genes in GO categories:

```
resultFisher <- runTest(GOdata, algorithm = "classic", statistic = "fisher")
```

```
##
##           -- Classic Algorithm --
##
##       the algorithm is scoring 95 nontrivial nodes
##       parameters:
##           test statistic:  fisher
```

The results can be displayed in a table:

```
GenTable(GOdata, classicFisher = resultFisher, topNodes = 10)
```

```
##         GO.ID                                        Term Annotated
## 1  GO:0042584                chromaffin granule membrane         2
## 2  GO:0042583                          chromaffin granule         3
```

```
## 3  GO:0030173        integral component of Golgi membrane         8
## 4  GO:0031228       intrinsic component of Golgi membrane         9
## 5  GO:0005871                              kinesin complex        11
## 6  GO:0030140       trans-Golgi network transport vesicle        16
## 7  GO:0031234 extrinsic component of cytoplasmic side ...        16
## 8  GO:0030667                  secretory granule membrane        20
## 9  GO:0019897      extrinsic component of plasma membrane        27
## 10 GO:0005875              microtubule associated complex        31
##    Significant Expected classicFisher
## 1            1     0.01        0.0069
## 2            1     0.01        0.0103
## 3            1     0.03        0.0273
## 4            1     0.03        0.0307
## 5            1     0.04        0.0374
## 6            1     0.06        0.0539
## 7            1     0.06        0.0539
## 8            1     0.07        0.0670
## 9            1     0.09        0.0895
## 10           1     0.11        0.1021
```

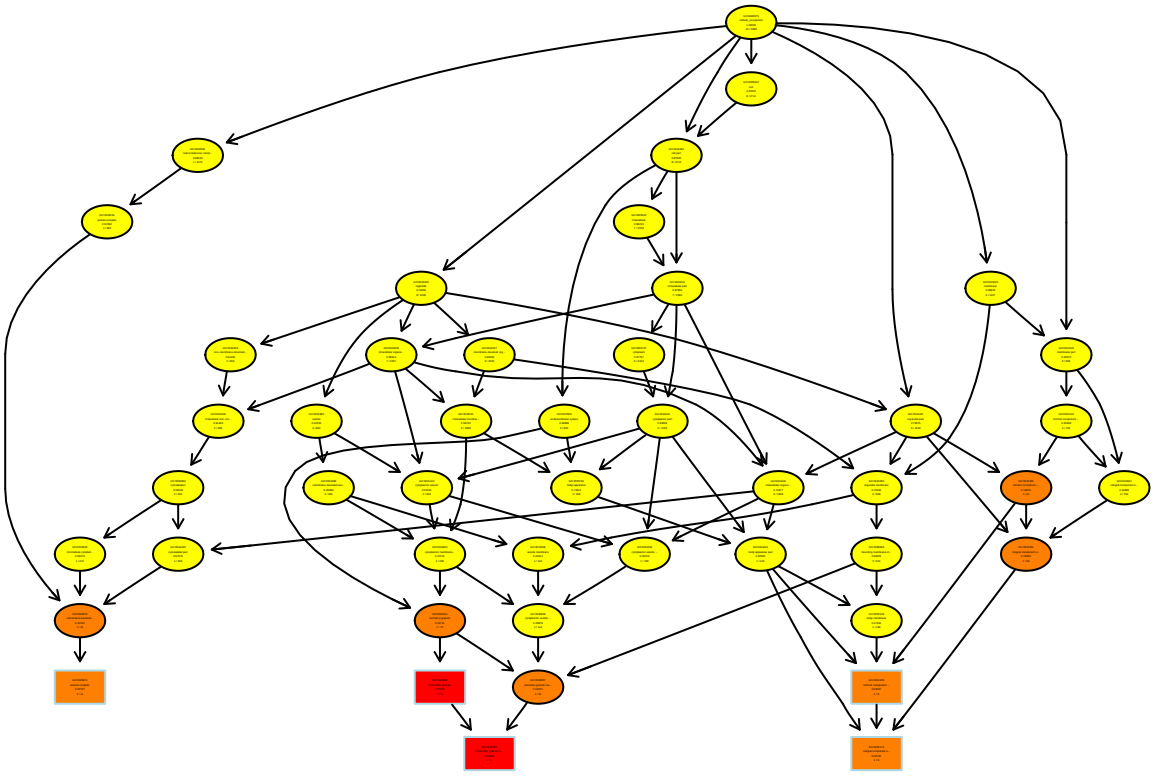
If we want to get fancy, we can also plot the results as a nice graph, but we need to install another package for that:

biocLite('Rgraphviz')

```
library(Rgraphviz)
```

```
## Loading required package: grid
##
## Attaching package: 'grid'
##
## The following object is masked from 'package:topGO':
##
##     depth
```

```
showSigOfNodes(GOdata, score(resultFisher),firstSigNodes=5,useInfo='all')
```



```
## $dag
## A graphNEL graph with directed edges
## Number of Nodes = 47
## Number of Edges = 87
##
## $complete.dag
## [1] "A graph with 47 nodes."
```

What theme do the most enriched GO terms relate to? ans–**Physical components (e g membranes, organelles and extracellular matrix)**

References on RNA sequencing

Wang et al. RNA-Seq: a revolutionary tool for transcriptomics. Nature Rev Genet 10:5763, 2009.

Mortazavi et al. Mapping and quantifying mammalian transcriptomes by RNASeq. Nature 5:621628,2008.

Auer and Doerge. Statistical Design and Analysis of RNA Sequencing Data. Genetics 185:405-416, 2010.