# Microarray analysis project_edx_nishant

*Nishant Upadhyay*

*Sunday, October 19, 2014*

## INTRODUCTION

In this exercise we will analyze data taken from a typical experiment in medical science. Our example data is taken from a microarray experiment.

(1) the experiment contains totally eight samples - clumps of human cells in this case
(2) four samples have been treated with a chemical, and four are non-treated controls
(3) there are approximately 22000 variables measured on each sample

The experiment can in other words be summarized as 4 + 4 columns of samples times 22000 rows of variables. What we want to find out is which, if any, of all these 22000 variables are changing (in medscience we often say "regulated") significantly as a result of the treatment.

This may appear to be an extreme example, but think of this experiment as a model for any study where we have two groups of subjects and we measure several tens or hundreds of parameters in each group and want to see how the groups differ.

*In a microarray experiment the variables measured are usually called "probes", "genes" or "features", and each measured value is often called an "expression" value. Don't get stuck on these terms during the exercise.*

## MICROARRAYS

In medical science we often ask questions like: "How does this medicine affect the liver", "What is the difference between normal skin and a tumor in the skin", or "Is there a good candidate molecule in a tissue with inflammation that we can target with a medicine to reduce the inflammation". A common denominator is that we want to compare two or more cases, and the best experiment is naturally one that gives as much information as possible on what is going on.

More or less everything that is "going on" in a tissue or a cell in the body is controlled by proteins. Proteins are complex molecules that come in thousands of brands. Some of them are building blocks of the cell, other transmit information or store energy, and another group, the enzymes, are the work-horses that perform the actions and chemical reactions in the cell. The proper function of a cell is totally dependent on that all these proteins are present in the correct concentrations.

Almost any change that a cell experiences is reflected in increasing and/or decreasing concentrations of one or more of all these proteins. In other words, if we have a method to compare all the protein concentrations in the healthy and sick cells or tissues, we could pinpoint which are affected, and this could give clues to what has actually happened in the sick sample, especially if we are lucky to know something about the affected proteins. Measuring proteins turns out to be very difficult, but instead of the proteins we can measure mRNAs: An mRNA is another kind of molecule which is used to construct the proteins. Every protein is built using one specific kind of mRNA, and the more of that mRNA we have, the more protein is produced.

Here is where the microarray comes in. A microarray is a little plate, it almost looks like a computer chip, with thousands of microscopic chemically prepared spots on it, and each of these spots has the ability to identify one specific mRNA if you pour the properly prepared cell sap from a tissue or cell sample on it. Under a special kind of microscope, each spot will give a light signal which we can measure that is higher the higher the concentration of that mRNA is in the sample. In other words, if we measure the whole microarray we will get a lot of values for all the different mRNA concentrations in the sample, which will directly tell us something about all the protein concentrations.

Ultimately, the production of each mRNA is controlled by a gene on the chromosomes. *That is why microarray data is often termed "gene expression" values.*

Microarrays have been used extensively in medical science during the last 20 years, and there is a special database at the National Institute for Biotechnology Information in USA (NCBI) where data from lots of microarray experiments are publicly available. We are going to download our example data from there. It comes from an investigation of human vein cells treated with the inflammatory stimulus TNF, aiming at elucidating mechanisms of inflammation.

## INSTALLING PACKAGES

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. You can read everything about BioConductor on its web page, www.bioconductor.org.

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://www.bioconductor.org/biocLite.R")

This command actually runs an R program that defines the function biocLite() that is used for installing Bioconductor. (Try pasting the web address into the address line of your browser, and you will see the actual R program!)

For our analysis we need the BioConductor packages "GEOquery" and "limma". Install them like this:

biocLite("GEOquery") biocLite("limma")

Activate the packages in the normal way:

```
library(Biobase)
library(GEOquery)
```

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

```
library(limma)
```

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

```
library(gplots)
```

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

("Biobase" is the core package of Bioconductor and is always installed automatically.)

### Retrieving data

Let's start by downloading a dataset from the NCBI Gene Expression Omnibus (GEO), http://www.ncbi.nlm.nih.gov/geo/. This is a large database of microarray experiments from scientists all over the world. The example data for this exercise has the identifier "GDS1542" in this database.

```r
                  setwd("H:/Elearning/courseera videos/edX_KIx KIexploRx Explore Statistics with R/week5
gds <- getGEO('GDS1542', destdir=".")
```

```
## Using locally cached version of GDS1542 found here:
## ./GDS1542.soft.gz
```

'gds' now contains the actual measured values as well as a lot of metadata, like organism, number of genes (features) etc.

```r
class(gds)
```

```
## [1] "GDS"
## attr(,"package")
## [1] "GEOquery"
```

you learn that gds is an object of class 'GDS', defined in the library 'GEOQuery'.

```r
show(gds)
```

```
## An object of class "GDS"
## channel_count
## [1] "1"
## dataset_id
## [1] "GDS1542" "GDS1542"
## description
## [1] "Analysis of macrovascular umbilical vein endothelial cells (HUVEC) stimulated with tumor necros
## [2] "control"
## [3] "tumor necrosis factor"
## email
## [1] "geo@ncbi.nlm.nih.gov"
## feature_count
## [1] "22283"
## institute
## [1] "NCBI NLM NIH"
## name
## [1] "Gene Expression Omnibus (GEO)"
## order
## [1] "none"
## platform
## [1] "GPL96"
## platform_organism
## [1] "Homo sapiens"
## platform_technology_type
## [1] "in situ oligonucleotide"
## pubmed_id
## [1] "16617158"
## ref
## [1] "Nucleic Acids Res. 2005 Jan 1;33 Database Issue:D562-6"
## reference_series
## [1] "GSE2639"
## sample_count
## [1] "8"
## sample_id
## [1] "GSM50777,GSM50778,GSM50779,GSM50780"
## [2] "GSM50781,GSM50782,GSM50783,GSM50784"
## sample_organism
## [1] "Homo sapiens"
## sample_type
## [1] "RNA"
## title
## [1] "Tumor necrosis factor effect on macrovascular umbilical vein endothelial cells"
## type
## [1] "Expression profiling by array" "agent"
## [3] "agent"
## update_date
## [1] "Nov 03 2006"
## value_type
## [1] "count"
## web_link
## [1] "http://www.ncbi.nlm.nih.gov/geo"
## An object of class "GEODataTable"
## ****** Column Descriptions ******
##     sample                 agent
## 1 GSM50777               control
## 2 GSM50778               control
## 3 GSM50779               control
## 4 GSM50780               control
## 5 GSM50781 tumor necrosis factor
## 6 GSM50782 tumor necrosis factor
## 7 GSM50783 tumor necrosis factor
## 8 GSM50784 tumor necrosis factor
##                                                                   description
## 1        Value for GSM50777: HUVEC_control experiment 1; src: HUVEC culture
## 2        Value for GSM50778: HUVEC_control experiment 2; src: HUVEC culture
## 3        Value for GSM50779: HUVEC_control experiment 3; src: HUVEC culture
## 4        Value for GSM50780: HUVEC_control experiment 4; src: HUVEC culture
## 5 Value for GSM50781: HUVEC_TNF-stimulated experiment 1; src: HUVEC culture
## 6 Value for GSM50782: HUVEC_TNF-stimulated experiment 2; src: HUVEC culture
## 7 Value for GSM50783: HUVEC_TNF-stimulated experiment 3; src: HUVEC culture
## 8 Value for GSM50784: HUVEC_TNF-stimulated experiment 4; src: HUVEC culture
## ****** Data Table ******
##      ID_REF IDENTIFIER GSM50777 GSM50778 GSM50779 GSM50780 GSM50781
## 1 1007_s_at       DDR1    124.1    113.4    107.3    127.6      128
## 2   1053_at       RFC2       29     74.1     32.3     36.7     35.7
## 3    117_at      HSPA6     26.4      4.3     13.1     17.7     23.9
## 4    121_at       PAX8    358.9    282.5    314.5    238.5    367.8
## 5 1255_g_at     GUCA1A     30.7     14.8       21     12.1     21.4
##   GSM50782 GSM50783 GSM50784
## 1    104.2     93.4    109.5
## 2       29     33.1     50.7
## 3     18.3     23.2     19.1
## 4    279.1    307.3    262.5
## 5     16.1     31.7     11.7
## 22278 more rows ...
```

You will be presented a lot of information about this experiment, and at the end the description of the samples (Columns descriptions) and the beginning of the actual Data Table. Each row in the data table are the measured values for one probe (spot) on the microarray. The cryptic name of this probe is in the first column (ID__REF). The gene (and protein produced from that gene) each probe detects is in the second column (IDENTIFIER).

The GEOData is a specialized class, for GEO derived datasets only. *For our further analyses we need to convert it to something that fits the general functions of Bioconductor. This is the ExpressionSet class, and there is a special function to do this conversion.* Try:

```r
eset <- GDS2eSet(gds)
```

```
## File stored at:
## C:\Users\Nishant\AppData\Local\Temp\Rtmp0wuQ8U/GPL96.annot.gz

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error
## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in readLines(con, 1): seek on a gzfile connection returned an
## internal error

## Warning in read.table(con, sep = sep, header = header, nrows = sampleRows,
## : seek on a gzfile connection returned an internal error

## Warning in read.table(con, sep = sep, header = header, nrows = sampleRows,
## : seek on a gzfile connection returned an internal error

## Warning in read.table(file = file, header = header, sep = sep, quote =
## quote, : seek on a gzfile connection returned an internal error
```

```
##To get an overview of eset, just type:
show(eset)
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 22283 features, 8 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: GSM50777 GSM50778 ... GSM50784 (8 total)
##   varLabels: sample agent description
##   varMetadata: labelDescription
## featureData
##   featureNames: 1007_s_at 1053_at ... AFFX-TrpnX-M_at (22283
##     total)
##   fvarLabels: ID Gene title ... GO:Component ID (21 total)
##   fvarMetadata: Column labelDescription
## experimentData: use 'experimentData(object)'
##   pubMedIds: 16617158
## Annotation:
```

The ExpressionSet contains basically the same information as the GDS, but structured differently.

```
#Let's currently just extract the actual values as a standard matrix:
expdata <- exprs(eset)

#Verify the size of the matrix:
dim(expdata)
```

```
## [1] 22283     8
```

```
#There are thousands of probes (rows) and much fewer samples (columns).
#Let's also get a glimpse of the data:
head(expdata)
```

```
##           GSM50777 GSM50778 GSM50779 GSM50780 GSM50781 GSM50782 GSM50783
## 1007_s_at    124.1    113.4    107.3    127.6    128.0    104.2     93.4
## 1053_at       29.0     74.1     32.3     36.7     35.7     29.0     33.1
## 117_at        26.4      4.3     13.1     17.7     23.9     18.3     23.2
## 121_at       358.9    282.5    314.5    238.5    367.8    279.1    307.3
## 1255_g_at     30.7     14.8     21.0     12.1     21.4     16.1     31.7
## 1294_at       56.2     50.0     59.9     48.7     68.7     63.0     59.7
##           GSM50784
## 1007_s_at    109.5
## 1053_at       50.7
## 117_at        19.1
## 121_at       262.5
## 1255_g_at     11.7
## 1294_at       59.2
```

```
#We lost the gene (IDENTIFIER) column from the table,
#but don't worry about that for now.
```

### Filtering

```r
#Sometimes there is missing data in real experiments,
#so let's check for that:
sum(is.na(expdata))
```

```
## [1] 1
```

```r
#Apparently there is one data point missing. Let's find the row
#in order to filter it away.
w <- which(apply(is.na(expdata), 1, sum) > 0 )

#Now we know which row to remove.Filter away that row
temp <- expdata[-w, ]
```

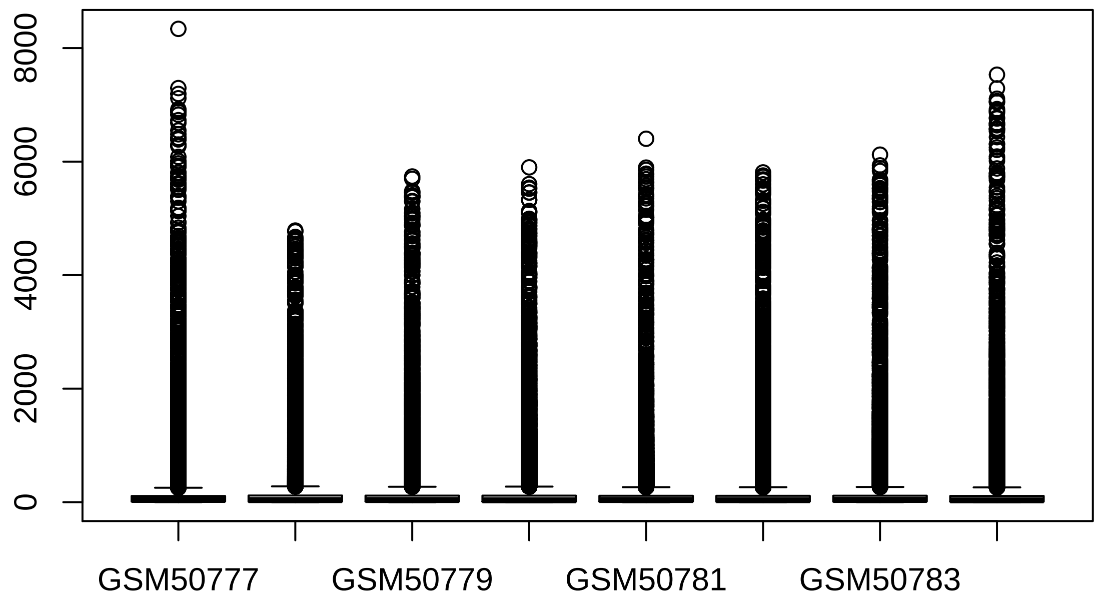
## Filtering continued

Verify that you removed exactly one row using a proper R function!

```r
#Then redefine expdata to the filtered result:
expdata <- temp
```

Now check how the data looks using a box plot:

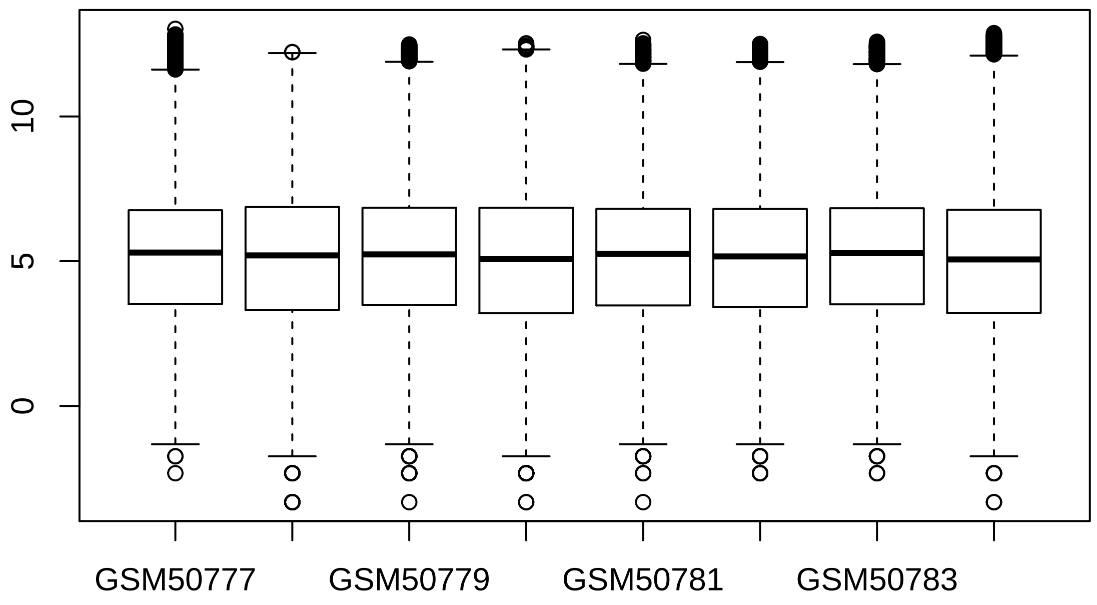
```r
boxplot(as.data.frame(expdata))
```



Oops, this looks strange! Almost all values are close to zero and there seems to be an extended tail of outlying high values. *Actually, microarray data is not at all normal distributed.* But if we make the 2-logarithm of data it is much better:

```
# Normalising microarray data by taking logs
logdata <- log2(expdata)

#Check again if it improved:
boxplot(as.data.frame(logdata))
```



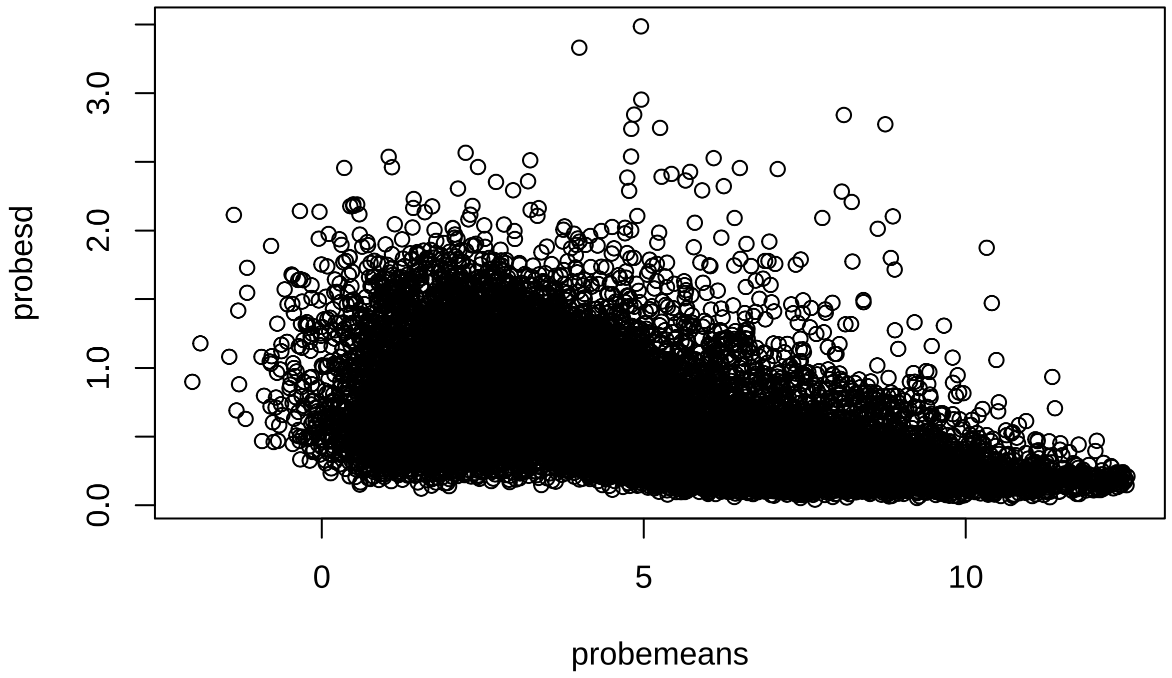
This is much better. Furthermore, the mean of all the samples are approximately the same, showing that on a global scale, every microarray worked equally well technically.

In a meaningful experiment with many variables, the purpose is to detect the (relatively small) subset of variables that change significantly between the conditions studied. Naively, we could now just make a lot of t-tests, one for each probe and see which are significantly changing. But this will give us a lot of false positives just by chance, even if there is no true change of any probe between the samples.

How many false positives would you expect when investigating changes in 22000 probes between identical samples at a p-value threshold of 0.05? ans-22000*0.05=1100

Thus, we want to eliminate as many variables as possible that we do not consider biologically meaningful before doing statistical testing. Furthermore in our kind of experiment with microarrays, data tend to be very noisy when the signal is very low. Let's have a look on that by plotting the standard deviation against the mean for every probe:

```r
probemeans <- apply(logdata, 1, mean)
probesd <- apply(logdata, 1, sd)
plot(probemeans, probesd)
```



It is clear that standard deviation is higher at lower means. Some probes have veryy high standard deviations - these may be truly changing, something we will investigate further on.

```
#Let's first eliminate the 25% of the probes that have weakest signal:
q25 <- quantile(probemeans, 0.25)
whichtosave <- which(probemeans > q25)
q25logdata <- logdata[whichtosave,]
```

Since we are only interested in probes that change, and thus have high variability, we can also remove those with very low variability. A way to do that is to filter on the inter-quartile range (using the IQR function). Keep only those with an IQR above 1.5:

```
mydata <- q25logdata[apply(q25logdata, 1, IQR) > 1.5, ]
```

```
#How many variables remain?
dim(mydata)[1]
```

```
## [1] 475
```

**Data exploration**

At this point it would be interesting to do a *Principal Component Analysis (PCA)*. This basically transforms the data to find new orthogonal variables that explain most of the variation in the dataset.

This variation can be analysed either between probes (rows) or samples (columns). We will use the function prcomp() for the PCA. It does the analysis between rows. For our purpose the samples are most interesting to compare, so we need to interchange columns and rows in mydata. This is also called to "transpose" the matrix.

Find out how you can transpose mydata and put the result in tdata? tdata <- aperm(mydata, c(2,1)) or tdata <- t(mydata) or tdata <- aperm(mydata)

```
# transposing data
tdata <- t(mydata)
```

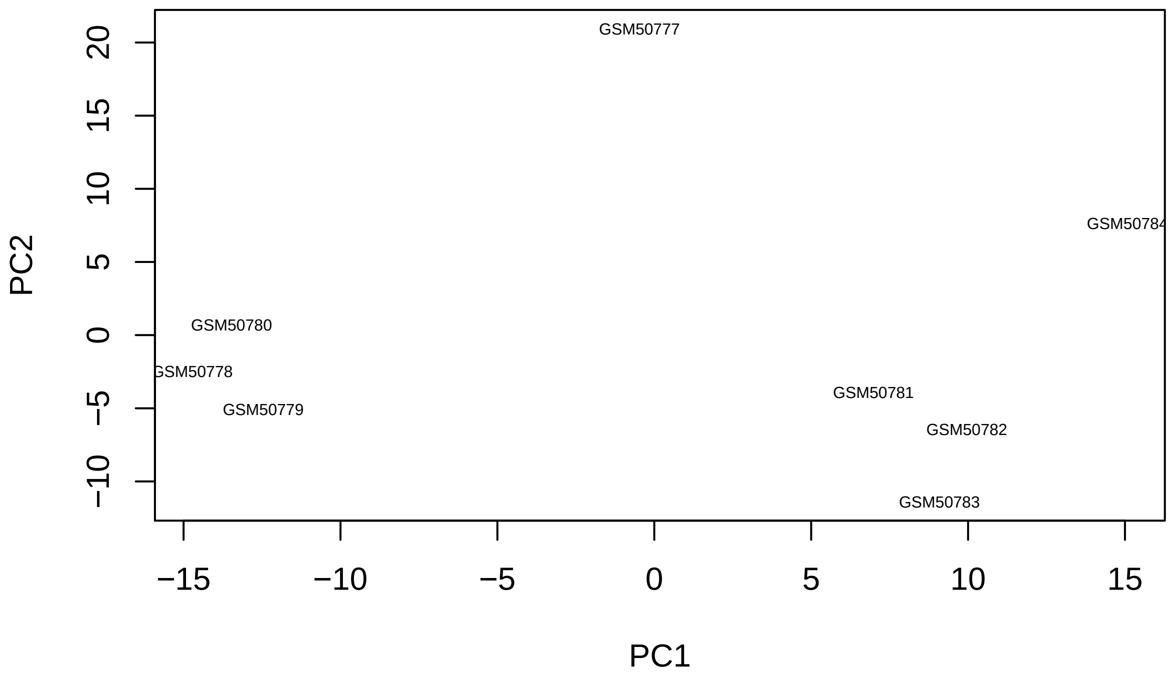
```
#Now we use function prcomp() on the transposed data:
pca <- prcomp(tdata, scale=T)
```

```
#We can look at the explanatory value of the principal components:
summary(pca)
```

```
## Importance of components:
##                            PC1     PC2    PC3    PC4     PC5     PC6   PC7
## Standard deviation     12.0216 10.1076 8.6016 7.3741 6.12051 5.99130 5.157
## Proportion of Variance  0.3043  0.2151 0.1558 0.1145 0.07886 0.07557 0.056
## Cumulative Proportion   0.3043  0.5193 0.6751 0.7896 0.86843 0.94400 1.000
##                              PC8
## Standard deviation     7.631e-15
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

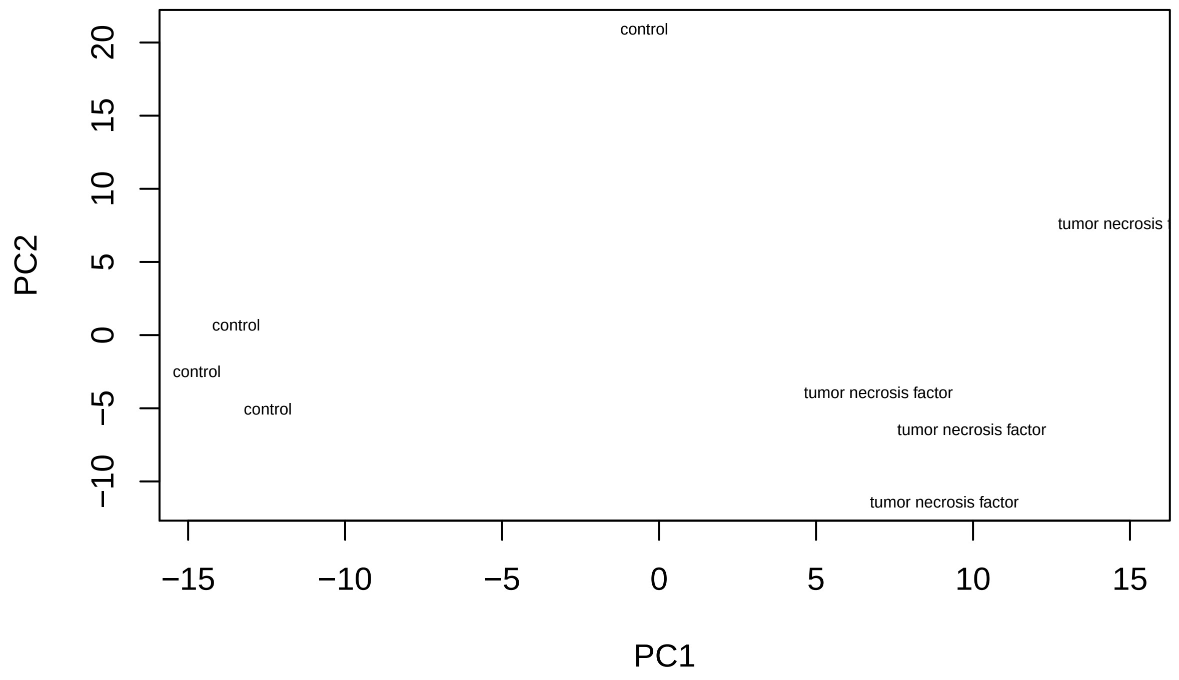
The first component explains the largest part of the variance, but not all. In the best of worlds, this accounts for the difference between our experimental conditions, otherwise we have some unknown batch effect that dominate the experiment.

```
#We can plot the samples in relation to the first two components:
plot(pca$x, type="n")
text(pca$x, rownames(pca$x), cex=0.5)
```



More interesting is maybe to see which experimental condition they belong to. For that purpose we extract this from the ExpressionSet (remember that?, use show(eset) again. . . )

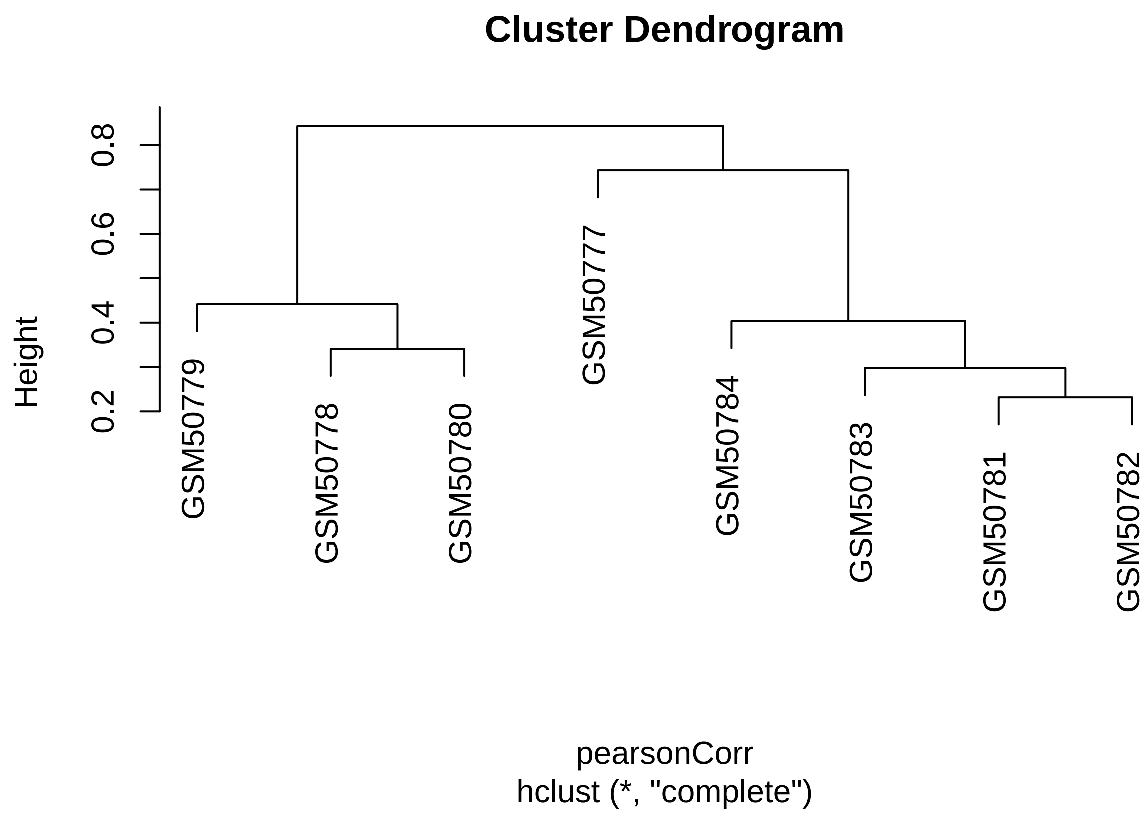
```r
conditions <- phenoData(eset)$agent
plot(pca$x, type="n")
text(pca$x, labels=conditions, cex=0.5)
```



Luckily the first component divides the samples by condition. However, in one sample something else seems to be going on, sending it away along the second component. That can be worth remembering when judging the final results.

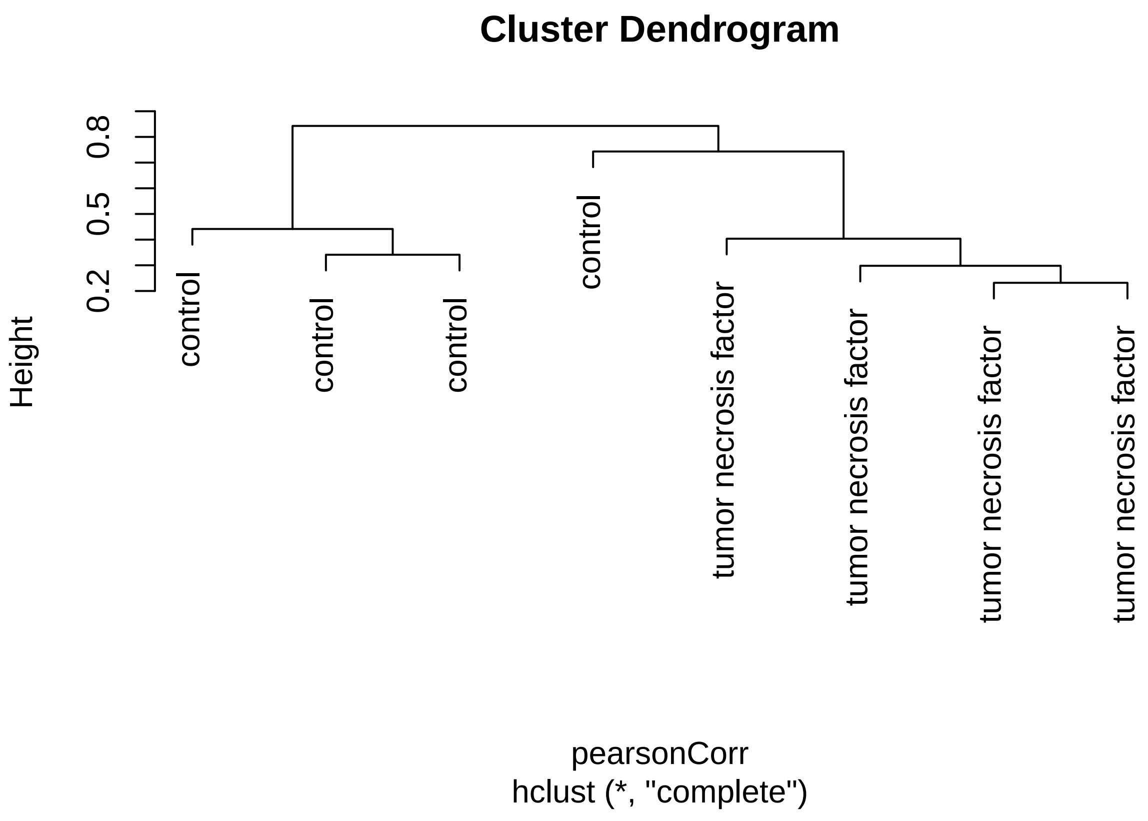
Another informative plot is to do a *dendrogram of correlation between the samples*. First we make a correlation matrix between the samples, than a hierarchical clustering is performed:

```
pearsonCorr <- as.dist(1 - cor(mydata))
hC <- hclust(pearsonCorr)
plot(hC, labels = sampleNames(eset))
```



The heights of the branches indicate how distant the samples are.

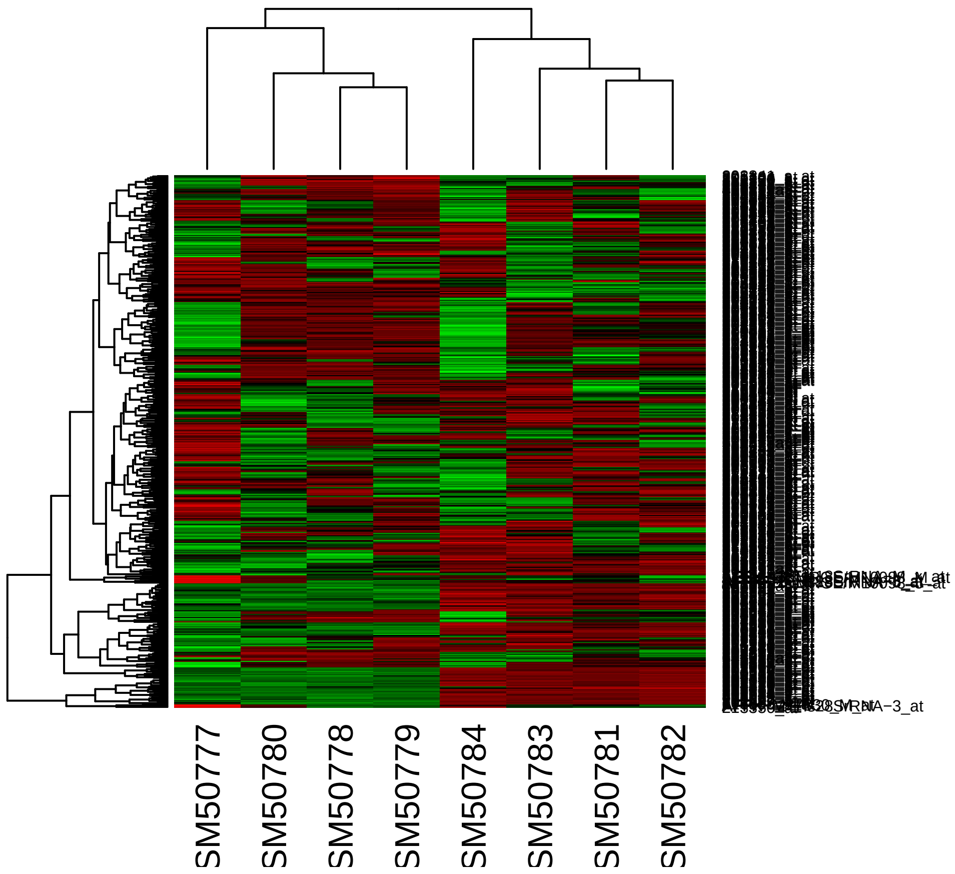
```r
#Recall which sample was in each condition by putting condition as labels:
plot(hC, labels = conditions)
```



The two groups and the sample "in between" are even more evident here than in the PCA.

*Another useful visualization of large datasets is the* **heatmap**. R clusters the data both on rows columns with this command:

```
library(gplots)
heatmap(mydata, col=greenred(100))
```



Red corresponds to high, green to low values. The dendrogram on top is basically the same as we produced earlier, in a slightly different order. Towards the bottom of the heatmap are the probes clustered that discriminate the two conditions clearly.

Let's now find the probes change significantly between the conditions. There are several tools to do that, more or less advanced. The most simplistic would be to do a t-test for every probe. This is however not a good idea. We have a lot of probes, and the few samples will give the estimate of variance low precision in many cases and give us many false negatives and positives.

However, it turns out that the variance of probes with approximately the same expression level is rather similar, and hence one can let probes "borrow" variance from each other to get better variance estimates. *Such a method is employed in the limma package (LInear Models for Microarrays).*

**limma** needs to see the whole dataset, including the high variance probes, to do correct variance estimations. Thus we will go back and use our ExpressionSet containing all data.In addition to the actual data, **limma needs a model matrix, basically information on which conditions each sample represents. R has a standard function for defining model matrices,model.matrix**. It uses the ~ operator to define dependencies.

Each input variable should be a factor, so let's first make a factor out of the conditions (agent in our ExpressionSet):

```
condfactor <- factor(eset$agent)
```

```
#Construct a model matrix, and assign the names to the columns:
design <- model.matrix(~0+condfactor)
```

```
#For the columns of the design matrix you could use any names.
#I choose "ctrl" for the control samples, and "tnf" for the
#chemically treated.
colnames(design) <- c("ctrl", "tnf")
```

```
#Check how the matrix looks:
design
```

```
##   ctrl tnf
## 1    1   0
## 2    1   0
## 3    1   0
## 4    1   0
## 5    0   1
## 6    0   1
## 7    0   1
## 8    0   1
## attr(,"assign")
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$condfactor
## [1] "contr.treatment"
```

Note that the first 4 samples have '1' in the control level, and the following 4 have the '1' in the other level. If we had had a multifactor experiment (ANOVA style data), we would have included more levels and assigned them in appropriate combinations to the samples.

```
#The next command estimates the variances:
fit <- lmFit(eset, design)
```

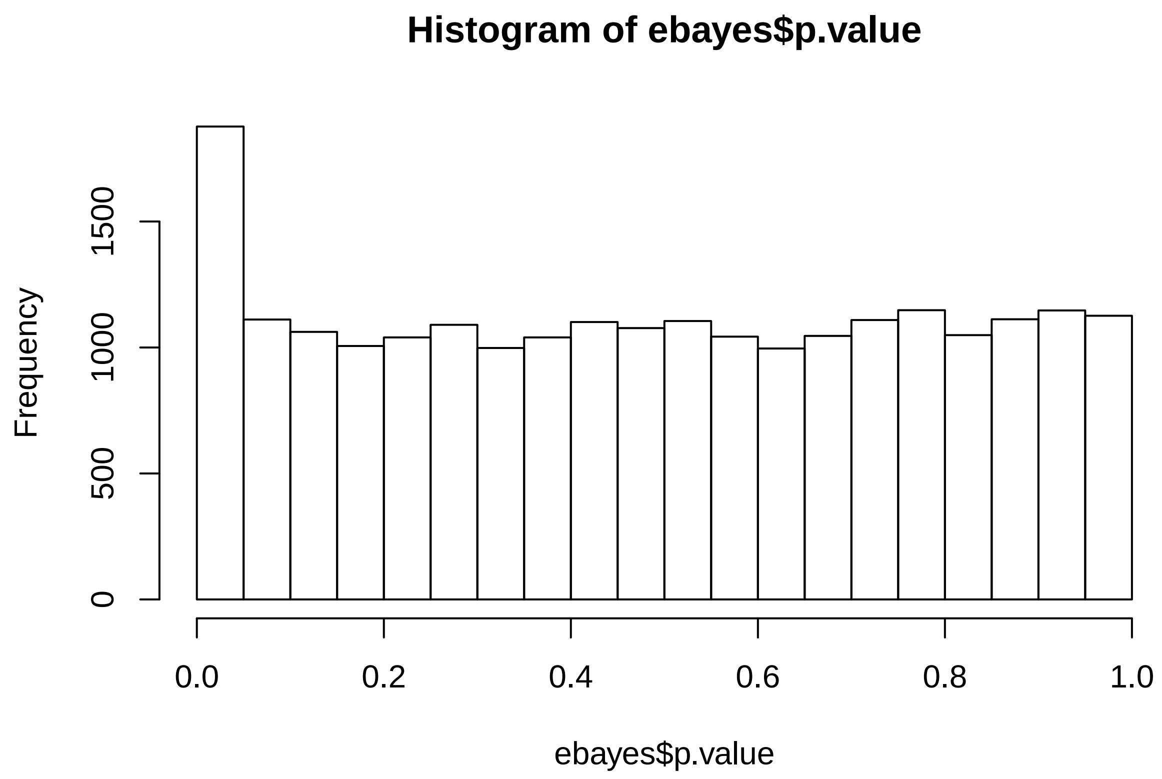
```
#Now we need to define the conditions we want to compare - trivial
#in this case since there are only two conditions:
contrastmatrix <- makeContrasts(tnf - ctrl,levels=design)
```

```
#The following commands calculate the p-values for the differences
#between the conditions defined by the contrast matrix:
fit <- contrasts.fit(fit, contrastmatrix)
ebayes <- eBayes(fit)
```

```
#As so often in R we can use show(ebayes) or just ebayes to see what the result contains:
ebayes
```

```
## An object of class "MArrayLM"
## $coefficients
## 1007_s_at   1053_at    117_at    121_at 1255_g_at
##    -9.325    -5.900     5.750     5.575     0.575
## 22278 more rows ...
##
## $stdev.unscaled
## 1007_s_at   1053_at    117_at    121_at 1255_g_at
## 0.7071068 0.7071068 0.7071068 0.7071068 0.7071068
## 22278 more rows ...
##
## $sigma
## [1] 12.195816 16.257896  6.817563 48.620276  8.437985
## 22278 more elements ...
##
## $df.residual
## [1] 6 6 6 6 6
## 22278 more elements ...
##
## $cov.coefficients
##             Contrasts
## Contrasts    tnf - ctrl
##   tnf - ctrl        0.5
##
## $genes
##                  ID                                 Gene title
## 1007_s_at 1007_s_at discoidin domain receptor tyrosine kinase 1
## 1053_at     1053_at replication factor C (activator 1) 2, 40kDa
## 117_at       117_at        heat shock 70kDa protein 6 (HSP70B')
## 121_at       121_at                                paired box 8
## 1255_g_at 1255_g_at     guanylate cyclase activator 1A (retina)
##           Gene symbol Gene ID UniGene title UniGene symbol UniGene ID
## 1007_s_at        DDR1     780          <NA>           <NA>       <NA>
## 1053_at          RFC2    5982          <NA>           <NA>       <NA>
## 117_at          HSPA6    3310          <NA>           <NA>       <NA>
## 121_at           PAX8    7849          <NA>           <NA>       <NA>
## 1255_g_at      GUCA1A    2978          <NA>           <NA>       <NA>
##                                                                        Nucleotide Title
## 1007_s_at                                 Human receptor tyrosine kinase DDR gene, complete cds
## 1053_at                     Human replication factor C, 40-kDa subunit (A1) mRNA, complete cds
## 117_at                                                    Human heat-shock protein HSP70B' gene
## 121_at                                                                     H.sapiens Pax8 mRNA
## 1255_g_at Homo sapiens guanylate cyclase activating protein (GCAP) gene exons 1-4, complete cds
##                GI GenBank Accession Platform_CLONEID Platform_ORF
## 1007_s_at 1753221           U48705             <NA>         <NA>
## 1053_at   1590810           M87338             <NA>         <NA>
## 117_at      35221           X51757             <NA>         <NA>
## 121_at      38425           X69699             <NA>         <NA>
## 1255_g_at  623404           L36861             <NA>         <NA>
##           Platform_SPOTID Chromosome location
## 1007_s_at            <NA>             6p21.3
## 1053_at              <NA>            7q11.23
## 117_at               <NA>               1q23
## 121_at               <NA>               2q13
## 1255_g_at            <NA>             6p21.1
##                                                     Chromosome annotation
## 1007_s_at               Chromosome 6, NC_000006.12 (30880909..30900156)
## 1053_at     Chromosome 7, NC_000007.14 (74231502..74254458, complement)
## 117_at                Chromosome 1, NC_000001.11 (161524246..161526897)
## 121_at    Chromosome 2, NC_000002.12 (113215997..113278921, complement)
## 1255_g_at               Chromosome 6, NC_000006.12 (42155406..42180056)
##
## 1007_s_at                                                          ATP binding///collagen bi
## 1053_at
## 117_at
## 121_at    DNA binding///DNA binding///RNA polymerase II core promoter sequence-specific DNA binding//
## 1255_g_at
##
## 1007_s_at
## 1053_at
## 117_at
## 121_at    anatomical structure morphogenesis///branching involved in ureteric bud morphogenesis///cel
## 1255_g_at
##
## 1007_s_at extracellular space///extracellular vesicular exosome///integral component of plasma membra
## 1053_at                                                                                     DNA rep
## 117_at           colocalizes_with COP9 signalosome///blood microparticle///centriole///cytoplasm///cy
## 121_at
## 1255_g_at                                                       photoreceptor disc membrane///photor
##                                                                                    GO:Function ID
## 1007_s_at GO:0005524///GO:0005518///GO:0005518///GO:0046872///GO:0005515///GO:0038062///GO:0004714
## 1053_at                                                    GO:0005524///GO:0003677///GO:0005515
## 117_at                          GO:0005524///GO:0042623///GO:0019899///GO:0031072///GO:0051082
## 121_at    GO:0003677///GO:0003677///GO:0000979///GO:0005515///GO:0003700///GO:0004996///GO:0044212
## 1255_g_at                                                                 GO:0005509///GO:0008048
##
## 1007_s_at
## 1053_at
## 117_at
## 121_at    GO:0009653///GO:0001658///GO:0071371///GO:0007417///GO:0042472///GO:0001822///GO:0003337//
## 1255_g_at
##                                                                                      GO:Component I
## 1007_s_at                              GO:0005615///GO:0070062///GO:0005887///GO:0005886///GO:00432
## 1053_at                                                                        GO:0005663///GO:00056
## 117_at    colocalizes_with GO:0008180///GO:0072562///GO:0005814///GO:0005737///GO:0005829///GO:00700
## 121_at                                                     NOT GO:0005730///GO:0005654///GO:00056
## 1255_g_at                                                      GO:0097381///GO:0001917///GO:00058
## 22278 more rows ...
##
## $Amean
## 1007_s_at   1053_at    117_at    121_at 1255_g_at
##  113.4375   40.0750   18.2500  301.3875   19.9375
## 22278 more elements ...
##
## $method
## [1] "ls"
##
## $design
##   ctrl tnf
## 1    1   0
## 2    1   0
## 3    1   0
## 4    1   0
## 5    0   1
## 6    0   1
## 7    0   1
## 8    0   1
## attr(,"assign")
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$condfactor
## [1] "contr.treatment"
##
##
## $contrasts
##       Contrasts
## Levels tnf - ctrl
##   ctrl         -1
##   tnf           1
##
## $df.prior
## [1] 0.85814
##
## $s2.prior
## [1] 34.10144
##
## $var.prior
## [1] 0.4691884
##
## $proportion
## [1] 0.01
##
## $s2.post
## [1]  134.39377  235.51266   44.93038 2072.40612   66.55760
## 22278 more elements ...
##
## $t
##   1007_s_at     1053_at      117_at      121_at   1255_g_at
## -1.13755988 -0.54370068  1.21314529  0.17318985  0.09967449
## 22278 more rows ...
##
## $df.total
## [1] 6.85814 6.85814 6.85814 6.85814 6.85814
## 22278 more elements ...
##
## $p.value
## 1007_s_at   1053_at    117_at    121_at 1255_g_at
## 0.2934809 0.6038631 0.2651982 0.8675027 0.9234536
## 22278 more rows ...
##
## $lods
## 1007_s_at   1053_at    117_at    121_at 1255_g_at
## -4.611886 -4.846650 -4.574738 -4.917754 -4.923293
## 22278 more rows ...
##
## $F
## [1] 1.294042474 0.295610433 1.471721492 0.029994724 0.009935004
## 22278 more elements ...
##
## $F.p.value
## [1] 0.2934809 0.6038631 0.2651982 0.8675027 0.9234536
## 22278 more elements ...
```

```
#A lot of stuff here, one of the most interesting is the p.value.
#Let's make a histogram of the p values:
hist(ebayes$p.value)
```



You see that the number of probes are enriched close to p = 0.00. This surplus is due to the genes that are significantly changing between the conditions. If the data had a skewed distribution we might see an accumulation at the p = 1.00 end, indicating a problem with our data.

If the data were completely random, the p values would be equally distributed from 0 to 1, thus if we from random data picked the probes that had $p < 0.05$ as "significant", we would pick exactly 1/20 of all probes, all being false positives. Unfortunately, there is no way to discriminate the surplus true probes from the false positives. This is problem with any study where a lot of variables are tested, for instance in large sociology studies.

However, there are ways to regulate the p-value cut off in order to have control over the false positives. A common way is the **Benjamini-Hochberg adjustment**. **This adjustment allows you to set a limit to how large fraction of false positive variables (false discovery rate, or FDR) you accept in the results.**

```
#This adjustment, at 5% FDR, is built into the decideTests function:
results <- decideTests(ebayes)

#decideTests produces 0, 1, or -1 for each probes, telling if that
#probe did not pass the test (0), or was significantly
#increased (1), or decreased (-1)
```

How can we display the number of probes that passed?

1.>sum(results == 1 | results == -1)

Correct - results==1 gives TRUE for every result that is 1, results==-1 TRUE for every result that is -1, and by | we combine so that either TRUE is TRUE for each probe. Then every TRUE counts as 1 in sum()

or

2.> length(which(results != 0))

Correct - we count the number of values in the row number array in B.

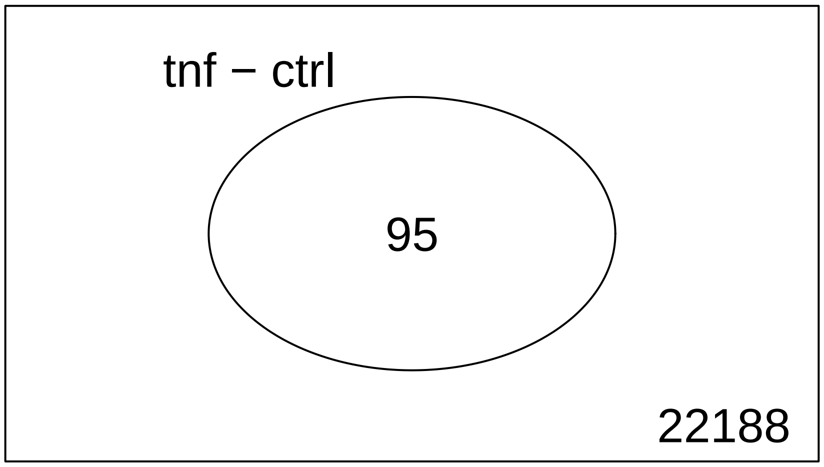
```
#displays the number of probes that passed
length(which(results != 0))
```

```
## [1] 95
```

There are other ways to control the rate of false positives in a multiple test experiment–**Holm-Bonferroni**

**Data exploration continued**

```
#A neat function to display the result of just two conditions is the Venn diagram:
vennDiagram(results)
```



```r
#Extract the original data for the most changing probes:
resData <- exprs(eset)[results != 0,]
```

```r
#Add gene symbols as row names:

geneSymbol <- as.array(fData(eset)[,"Gene symbol"])
gs <- geneSymbol[c(which(results != 0))]
rownames(resData) <- gs
```

```r
#Add p-values in an extra column:

pvalues <- ebayes$p.value[results != 0,]
resData <- cbind(resData, pvalues)
```

```r
#And add-p values corrected for multiple testing (q-values):

adj.pvalues <- p.adjust(ebayes$p.value, method="BH")
adj.pvalues <- adj.pvalues[results != 0]
resData <- cbind(resData, adj.pvalues)
```

The character values are surrounded by quotes.

Which parameter could we add to write.table to avoid that? —-‘quote=FALSE’

```r
#Write output to a file:


write.table(resData, "most_regulated.txt",sep="\t", quote = FALSE)
```

**Now we have generated both quality control graphs and a table of probes that change significantly between the samples. It’s time for the medical scientists to take over and let them make the biological conclusions.**