# Common microarray analyses in Bioconductor

Mark Dunning

November 14, 2013

The GOstats pacakge can be used to test the association of Gene Ontology (GO) terms to genes in a gene list. Testing is done using a Hypergeometric test after defining a gene universe and a set of interesting genes. This package demonstrates the usage of GOstats for a breast cancer experiment using Illu- mina data. The starting point for the practical is a normalised expression matrix, therefore the same commands in the practical can be applied to data from other technologies.

## 1 Dealing with public data

### 1.1 Retrieving public data from GEO

The data from this experiment comprises nine paired tumor/normal colon tissues on Illumina HT12_v3 gene expression Beadchips. These data were generated to inform a comparison of technologies for microRNA profiling. However, we will only use the mRNA data here.

**Use Case:** Download the dataset with GEO GSE33126.

```
> library(GEOquery)
> library(limma)
> library(RColorBrewer)
> url <- "ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SeriesMatrix/GSE33126/"
> filenm <- "GSE33126_series_matrix.txt.gz"
> if (!file.exists("GSE33126_series_matrix.txt.gz")) download.file(paste(url,
+     filenm, sep = ""), destfile = filenm)
> gse <- getGEO(filename = filenm)
```

We need to explore the data to ensure that they have been normalised and are on an appropriate scale for analysis.

**Use Case:** Are the data on the $\log_2$ scale? Are they normalised? Look at the phenotypic data stored with the object and find information about the sample groups in the study and patient IDs.

```
> head(exprs(gse))
> exprs(gse) <- log2(exprs(gse))
> boxplot(exprs(gse), outline = FALSE)
> pData(gse)[1:2, ]
> SampleGroup <- pData(gse)$source_name_ch1
> Patient <- pData(gse)$characteristics_ch1.1
```

### 1.2 Downloading data from ArrayExpress

**Use Case:** Download the dataset E-TABM-25

```
> library(ArrayExpress)
> ETABM25.affybatch = ArrayExpress("E-TABM-25")
> ETABM25.affybatch
```

```
> print(ETABM25.affybatch)
> sampleNames(ETABM25.affybatch)
> colnames(pData(ETABM25.affybatch))
> head(exprs(ETABM25.affybatch))
> boxplot(log2(exprs(ETABM25.affybatch)), outline = FALSE)
```

### 1.3 Using Bioconductor data packages

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("breastCancerVDX")
> library(breastCancerVDX)
> data(vdx)
> pData(vdx)
```

## 2 Clustering

**Use Case:** Perform a hierarchical clustering on the samples using Euclidean distance and average linkage agglomeration. Are there any distinct groups in the data? Try using the correlation based similarity measure and complete linkage. Does the choice of method effect your clustering?

```
> par(mfrow = c(1, 2))
> clust.euclid = hclust(dist(t(exprs(gse))), method = "complete")
> clust.cor = hclust(as.dist(1 - cor(exprs(gse))), method = "complete")
> plot(clust.euclid, label = SampleGroup)
> plot(clust.cor, label = SampleGroup)
```

## 3 Principal Components Analysis and MDS

Principal components analysis (PCA) is a data reduction technique that allows to simplify multidimensional data sets to 2 or 3 dimensions for plotting purposes and visual variance analysis.

**Use Case:** Perform PCA

```
> library(ggplot2)
> distMat <- dist(t(exprs(gse)))
> pca <- prcomp(distMat)
> summary(pca)
> pcRes <- data.frame(pca$rotation, SampleGroup, Sample = rownames(pca$x),
+     Patient)
> ggplot(pcRes, aes(x = PC1, y = PC2, col = SampleGroup,
+     label = Patient)) + geom_point() + geom_text(vjust = 0,
+     alpha = 0.5)
> ggplot(pcRes, aes(x = SampleGroup, y = PC1, fill = SampleGroup)) +
+     geom_boxplot()
> ggplot(pcRes, aes(x = SampleGroup, y = PC2, fill = SampleGroup)) +
+     geom_boxplot()
```

**Use Case:** MDS

```
> cols <- SampleGroup
> cols <- gsub("tumor", "orange", cols)
> cols <- gsub("normal", "steelblue", cols)
> par(mfrow = c(1, 1))
> plotMDS(exprs(gse), col = cols)
```

```
> type <- as.factor(pData(gse)$source_name_ch1)
> levels(type) <- brewer.pal("Set1", n = length(levels(type)))
> WGCNA:::plotDendroAndColors(clust.cor, colors = as.character(type))
> design <- model.matrix(~0 + as.factor(pData(gse)$source_name_ch1))
> colnames(design) <- c("Normal", "Tumour")
> fit <- lmFit(exprs(gse), design)
> cMat <- makeContrasts(DE = Tumour - Normal, levels = design)
> fit2 <- contrasts.fit(fit, cMat)
> efit <- eBayes(fit2)
```

## 4 Producing a heatmap

A heatmap is often used to visualise differences between samples. Each row represents a gene and each column is an array and colours indicate the expression levels of genes. Both samples and genes with similar expression profile are clustered together. Drawing a heatmap in R uses a lot of memory and can take a long time, therefore reducing the amount of data to be plotted is usually recommended. Typically, data are filtered to in- clude the genes which tell us the most about the biological variation. Note that selection of such genes is done without using prior knowledge about the samples.
In R, the apply function is shorthand for running the same function repeatedly for each row or column in a matrix. For example, if you want to calculate the mean or standard deviation for each gene across all samples, or calculations for all expression values in an array.

**Use Case:** Make a heatmap using the 100 most variable genes in the experiment according to their inter-quartile range (IQR). Can you see any structure in the data?

```
> IQRs = apply(exprs(gse), 1, IQR)
> highVarGenes = order(IQRs, decreasing = T)[1:100]
> Symbols <- as.character(fData(gse)$Symbol[highVarGenes])
> heatmap(as.matrix(exprs(gse)[highVarGenes, ]), ColSideColors = cols,
+     labRow = Symbols)
```

The heatmap function can be customised in many ways to make the output more infor- mative. For example, the labRow and labCol parameters can be used to give labels to the rows (genes) and columns (arrays) of the heatmap. Similarly, ColSideColors and RowSide- Colors give coloured labels, often used to indicate different groups which are know in ad- vance. See the help page for heatmap for more details.

## 5 Obtaining annotation information

Annotation information can be retrieved from Bioconductor to assist in the interpretation of array data. In our case we are using the Illumina Human Version 1 chip, which has its an- notation in the illuminaHumanv1 package. Similar packages are available for Human6 version 2 (illuminaHumanv2) and Mouse6 (illuminaMousev1) or affy chips. The illuminaHumanv1 package contains a series of 'environment' objects, each of which contains mappings between each probe on the Human6 chip and the genome. For example, the mapping between each probe and Entrez IDs is provided by theilluminaHumanv1ENTREZID environment. Due to the design of this chip, not all probes will map to a Entrez ID, in which case the value of NA is returned. A list of all such environments can be retrieved by running the command illuminaHumanv1(). The environments are split into two sections; environment that provide mappings from probes on the array to the genome ("Mappings found for probe based rda files"), and mappings which go from the genome to probes on the array ("Mappings found for non-probe based rda files").

**Use Case:** Load the illuminHumanv3.db annotation package. What mappings can be obtained for every probe?

```
> library(illuminaHumanv3.db)
> illuminaHumanv3()
```

The `mget` function can be used to retrieve annotation information for each probe from a particular environment. The parameters passed to the function are the manufacturer IDs and the environment we want to look up. It is usually a good idea to also specify `ifnotfound=NA`. This prevents the function from falling over if we pass an identifier that it can't find in the environment.

**Use Case:** Retrieve the chromosome, refseq and entrez ID for each gene

```
> ids = rownames(exprs(gse))[1:10]
> chr = mget(ids, illuminaHumanv3CHR, ifnotfound = NA)
> refseq = mget(ids, illuminaHumanv3REFSEQ, ifnotfound = NA)
> entrez = mget(ids, illuminaHumanv3ENTREZID, ifnotfound = NA)
> anno = cbind(Ill_ID = as.character(ids), Chr = as.character(chr),
+     RefSeq = as.character(refseq), Entrez = entrez)
> anno
```

Environments can also be used to find which probes on the array map to a particular feature from a database, such as a GO term, pathway or gene symbol.

**Use Case:** Which probes on this chip are part of the cell cycle using the GO (GO:0007049) or KEGG (04110)?

```
> cellCycleProbesGO = mget("GO:0007049", illuminaHumanv3GO2PROBE)
> cellCycleProbesKEGG = mget("04110", illuminaHumanv3PATH2PROBE)
> cellCycleProbesGO[[1]][1:10]
> length(cellCycleProbesKEGG[[1]])
```

## 6 Gene-Ontology analysis

### 6.1 Non-specific filtering

We are now going to create a gene universe by removing genes for will not contribute to the subsequent analysis. Such filtering is done without regarding the phenotype variables - hence a "non-specific" filter. The Illumina Human6 chip contains 47,293 probes, but less than half of these have enough detailed information to useful for a GO analysis. Therefore we restrict the dataset to only probes for which we have a Entrez ID. It is also recommended to select probes with sufficient variability across samples to be inter- esting; as probes with little variability will no be interesting to the question we are trying to answer. The interquartile-range of each probe across all arrays is commonly used for this with a cut-off of 0.5.

**Use Case:** Create the gene universe of all genes with Entrez ID and with sufficient variation across samples. How big is the universe?

```
> entrezIds = mget(rownames(exprs(gse)), illuminaHumanv3ENTREZID,
+     ifnotfound = NA)
> haveEntrezId = names(entrezIds)[sapply(entrezIds, function(x) !is.na(x))]
> entrezSubset = exprs(gse)[haveEntrezId, ]
> entrezIQR = apply(entrezSubset, 1, IQR)
> selected = entrezIQR > 0.5
> nsFiltered = entrezSubset[selected, ]
> universeIds = unlist(mget(rownames(nsFiltered), illuminaHumanv3ENTREZID,
+     ifnotfound = NA))
```

Remember that the size of the universe can have an effect on the analysis. If the universe is made artificially large by including too many uninformative probes, the p-values for the GO terms will appear more significant.

### 6.2 Selecting genes of interest and performing Hypergeometric test

We now test the genes in the universe to see which ones have significant differences be- tween the two groups. For this, we use the rowttests function implemented in the GOstats package, which performs a t-test for each row with respect to a factor. The p-values of the test can be extracted, with one p-value given for each probe.

**Use Case:** Do a t-test for each probe between the two groups of samples we have identified. How many probes are significant at the 0.05 level? Define a list of genes to be used in the hypergeometric test by finding the Entrez Ids for these significant probes.

```
> library(GOstats)
> library(genefilter)
> fac = as.factor(pData(gse)$source_name_ch1)
> ttests = rowttests(as.matrix(nsFiltered), fac)
> smPV = ttests$p.value < 0.05
> pvalFiltered = nsFiltered[smPV, ]
> dim(pvalFiltered)
> selectedEntrezIds = unlist(mget(rownames(pvalFiltered),
+     illuminaHumanv3ENTREZID, ifnotfound = NA))
```

The hyperGTest function is used to do the hypergeometric test for GO terms. Rather than passing a long list of parameters to the function. An object of type GOHyperGParams is created to hold all the parameters we need to run the hypergeometric test. This object can then be passed to hyperGTest multiple times without having to re-type the parameters each time. The meanings of these parameters are as follows:

- geneIds - The list of identifiers for the genes that we have selected as interesting
- universeGeneIds - The list of identifiers resulting from non-specific filtering
- annotation - The name of the annotation package that will be used
- ontology - The name of the GO ontology that will be tested; either BP, CC or MF
- pvaluecutoff - p-value that we will use to select significant GO terms
- testDirection - Either "over" or "under" for over or under represented terms respectively
- conditional - A more sophisticated form of hypergeometric test, which takes the relationships between terms in the GO graph can be used if this is set to TRUE. For this practical we will keep conditional = FALSE

**Use Case:** Do a hypergeometric test to find which GO terms are over-represented in the filtered list of genes. How many GO terms are significant with a p-value of 0.05?

```
> params = new("GOHyperGParams", geneIds = selectedEntrezIds,
+     universeGeneIds = universeIds, annotation = "illuminaHumanv3",
+     ontology = "BP", pvalueCutoff = 0.05, conditional = FALSE,
+     testDirection = "over")
> hgOver = hyperGTest(params)
> hgOver
```

The summary function can be used to view the results of the test in matrix form. The rows of the matrix are arranged in order of significance. The p-value is shown for each GO term along with with total number of genes for that GO term, number of genes we would be expect to appear in the gene list by chance and that number that were observed. A descriptive name is also given for each term. The results can also be printed out to a HTML report using htmlReport.

**Use Case:** View the results of the top 20 GO terms and create a HTML report.

```
> summary(hgOver)[1:20, ]
```

GOstats also has the facility to test for KEGG pathways and chromosome bands which are over-reprsented. The procedure of creating a gene universe and set of selected genes is the same. However, we have to use a different object for the parameters, as not all

**Use Case:** Repeat the hypergeometric test for chromsome bands and KEGG pathways

```
> keggParams = new("KEGGHyperGParams", geneIds = selectedEntrezIds,
+     universeGeneIds = universeIds, annotation = "illuminaHumanv3",
+     pvalueCutoff = 0.05, testDirection = "over")
> keggHgOver = hyperGTest(keggParams)
> summary(keggHgOver)
> chrParams = new("ChrMapHyperGParams", geneIds = selectedEntrezIds,
+     universeGeneIds = universeIds, annotation = "illuminaHumanv3",
+     pvalueCutoff = 0.05, testDirection = "over", conditional = TRUE)
> chrHgOver = hyperGTest(chrParams)
> summary(chrHgOver)
```

## 7 Creating a classifer

```
> anov <- apply(exprs(nki), 1, function(x) t.test(x ~ pData(nki)$er)$p.value)
> threshold <- quantile(anov, 3000/length(anov))
> sub.Expression <- exprs(nki)[which(anov < threshold),
+     ]
> naVals <- apply(sub.Expression, 1, function(x) any(is.na(x)))
> sub.Expression <- sub.Expression[-which(naVals), ]
> library(class)
> res <- knn(train = t(sub.Expression), test = t(sub.Expression),
+     cl = pData(nki)$er, k = 5)
> er.ap <- table(res, pData(nki)$er)
> er.ap <- 1 - sum(diag(er.ap))/sum(er.ap)
```

```
> library(pamr)
> dat <- exprs(vdx)
> x <- NULL
> x$x <- dat
> dat <- pamr.knnimpute(x)$x
> gN <- as.character(fData(vdx)$Gene.symbol)
> gI <- featureNames(vdx)
> sI <- sampleNames(vdx)
> train.dat <- list(x = dat, y = pData(vdx)$er, genenames = gN,
+     geneid = gI, sampleid = sI)
```

```
> model <- pamr.train(train.dat, n.threshold = 100)
```

```
> model.cv <- pamr.cv(model, train.dat, nfold = 10)
> model.cv
> pamr.plotcv(model.cv)
```

```
> Delta <- 8
> pamr.plotcen(model, train.dat, Delta)
> pamr.confusion(model.cv, Delta)
> pamr.plotcvprob(model, train.dat, Delta)
> pamr.geneplot(model, train.dat, Delta)
```

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("breastCancerMAINZ")
> library(breastCancerMAINZ)
> data(mainz)
> pData(mainz)
> table(pamr.predict(model, exprs(mainz), Delta), pData(mainz)$er)
> boxplot(pamr.predict(model, exprs(mainz), Delta, type = "posterior")[,
+     1] ~ pData(mainz)$er)
```

## 8 Survival

```
> plot(survfit(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
+     pData(vdx)$er))
> survdiff(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
+     pData(vdx)$er)
> library(genefu)
> data(ssp2003)
> intSubtype <- intrinsic.cluster.predict(sbt.model = ssp2003,
+     data = t(exprs(vdx)), annot = fData(vdx), do.mapping = TRUE)$subtype
> plot(survfit(Surv(pData(vdx)$t.dmfs, pData(vdx)$e.dmfs) ~
+     intSubtype))
```