Using affycoretools

James W. MacDonald

June 16, 2014

1 Overview

This package is made up of various 'wrapper' functions that I use to help automate some of the more routine aspects of my job as a Biostatistician in a microarray core facility. Since the vast majority of analyses I do are based on data from Affymetrix GeneChips, the focus of these functions are also directed towards this platform. This does not however preclude these functions from being extended to other platforms including other oligonucleotide arrays as well as spotted cDNA arrays.

2 Introduction

For most analyses, I follow the precepts of *Literate Statistical practice* [1]. The basic idea being that the document used to present an analysis is also what is used to *do* the analysis. To do this, I use Emacs/ESS (*Emacs speaks statistics*) [2] and an `Sweave` document [3].

An `Sweave` document is a file (with an .Rnw extension) that contains text and LaTeX markup that will be used to create (usually) a PDF document, as well as R code that will be used to create plots or tables in the resulting document, and/or to provide finished output suitable for presentation to your client(s). Usually the `Sweave` document does both. Note that this vignette (as are most BioConductor vignettes) is produced using an `Sweave` document.

The learning curve for LaTeX and the other markup required to create a functional `Sweave` document can be steep. However, it is well worth the effort to learn for anybody who is routinely required to do statistical analyses and present the results to others. For anybody interested in learning about `Sweave`, the two best sources of information (in my opinion) are the Sweave User Manual, and just about any BioConductor vignette. The .Rnw file for most BioC vignettes can be found in the R-Home/library/¡packagename¿/doc directory. In addition, there is an example `Sweave` document in the example directory for this package that is the basis for most of the analyses I do.

Because I do all of my analyses using `Sweave`, most of the functions in this package are designed for both interactive and non-interactive use. In addition, most functions will output information that may be useful to present in the resulting text document.

3 Interactive Analyses

3.1 Quality Control

For these examples, I will be using some data that were generated in our microarray core. The experiment is a simple comparison of two different cell lines, one of which is sensitive to a particular treatment, whereas the other is not. One set of samples was prepared using the Affymetrix *in vitro translation* kit, whereas the other set of samples was prepared using the NuGen Ovation kit. There are three biological replicates for each sample type. The celfiles can be found in the examples directory of this package (R-home/library/affycoretools/example).

For some analyses it may not be necessary to generate a report detailing the analysis, or you simply may want to do a quick quality check to ensure the raw data are of high enough quality to proceed with the analysis. In this case we can just do some quality control plots and compute expression measures using `affystart`.

The `affystart` function may be used to compute `rma`, `gcrma` or `mas5` expression values. In the case of `mas5` expression values, the output (written to a text file in the working directory) includes the P/M/A calls and associated p-values.

```
library("affycoretools")
pd <- new("AnnotatedDataFrame",
          data = read.table("pdata.txt", header = TRUE, row.names = 1))
eset <- affystart(groups = rep(1:4, each = 3),
                  groupnames = unique(paste(pData(pd)[,1],
                  pData(pd)[,2], sep = "-")),
                  phenoData = pd)
```

```
## Warning:  replacing previous import by 'utils::head' when loading 'hgfocuscdf'
## Warning:  replacing previous import by 'utils::tail' when loading 'hgfocuscdf'

##
```

Figure 1 is the usual density plot – we have found that this plot is one of the more informative quality control plots available, at least for `rma`. Any chips with high background (curve shifted to the right) invariably need to be re-done, which in this case usually means re-fragmenting the cRNA and re-hybridizing to a new chip. With these data I expect much more variability due to the differences in the *IVT* kits that were used for the two sample sets. However, a case could be made that sample12 needs to be re-done.

Figure 2 is an RNA degradation plot – this is supposed to give some idea of how much degradation of mRNA occured, and how well the *IVT* step went. This plot is moderately useful, but not nearly as informative as the density plot. Here we can see that the slope of the lines for the two groups is quite different, indicating that the two *IVT* kits give distinctly different results.

Figure 3 is a plot of the first two principal components from a principal components analysis (PCA). Basically, this is used to show the overall structure of the data. This is another very useful plot. In most cases we expect replicate samples to group together, indicating general similarity in overall expression patterns. It may be difficult however to determine from this plot how closely samples are grouping – for instance, the NuGen samples appear to be quite well separated on the y-axis. To determine how meaningful this separation is, we need a `screeplot`.

Figure 4 shows the `screeplot` for this PCA. Each bar shows how much of the overall variance is captured by each principal component. Here we can see that the first PC captures the vast majority of the variance, which indicates that the separation of the samples on the y-axis (the second PC) is actually quite small, so the samples are grouping fairly tightly.

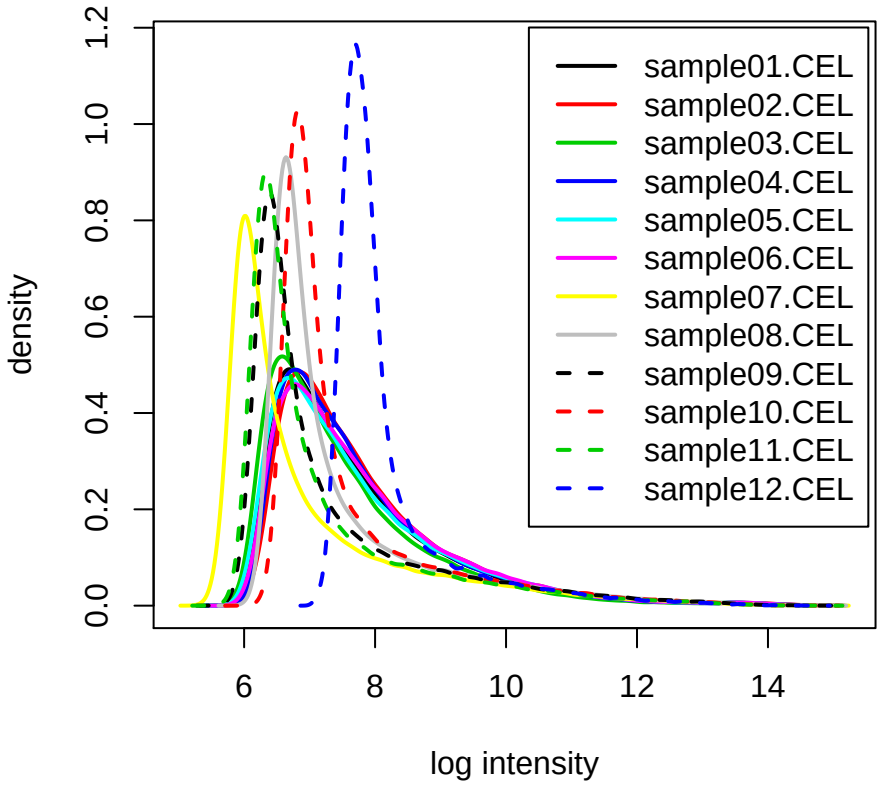

Figure 1: Density plot

The `affystart` function calls three other functions to make these plots (`plotHist`, `plotDeg`, and `plotPCA`), which can all be called individually to make just one of these plots, or in the case of `plotPCA`, to make either the PCA or the `screeplot`.

3.2 Computing Differential Expression

After checking the quality control plots (and maybe looking at other QC plots that are available in the *affyPLM* package), the next step is to make comparisons and output lists of differentially expressed genes. Because of the obvious differences between the two sample sets, it is probably preferable to compute expression values separately and then combine the data.

```
eset1 <- affystart(filenames = list.celfiles()[1:6],
                   plot = FALSE, pca = FALSE)
eset2 <- affystart(filenames = list.celfiles()[7:12],
                   plot = FALSE, pca = FALSE)
eset <- new("ExpressionSet",
            exprs = cbind(exprs(eset1), exprs(eset2)),
            phenoData = pd,
```

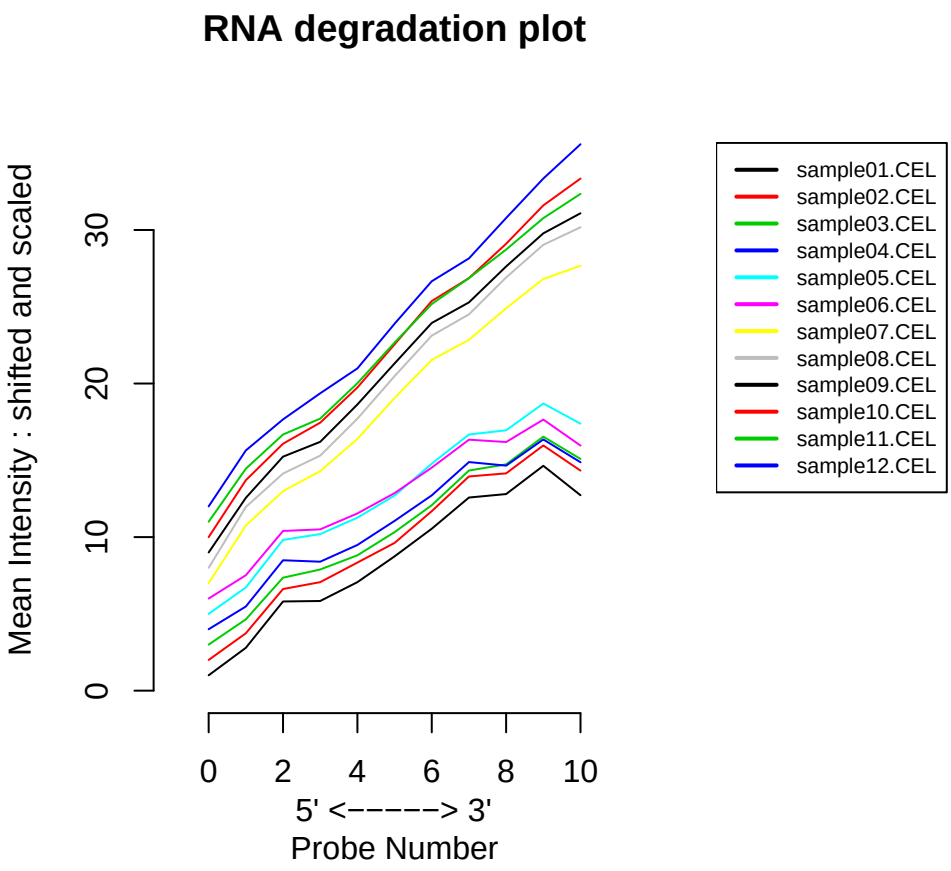

Figure 2: RNA degradation plot

```
            annotation = annotation(eset1))
```

I do most of my analyses using the *limma* package. I find that this package is capable of analyzing most of the experiments that I see. I also like to use the *annaffy* package for creating output to give to my clients. The HTML tables that can be produced using this package can either be posted on the web or an intranet, or simply emailed to the client. Because I use both of these packages together on a regular basis, some of my functions are designed to link the results from a *limma* analysis to the *annaffy* package.

The data set we are using for this vignette was originally produced in order to see how comparable the results from the two *IVT* kits were. One way to make this comparison is to fit a linear model, compute contrasts of sensitive and insensitive samples for each sample set, and then look for genes that are significant in both contrasts.

First, we filter the data, removing those genes that appear not to be expressed in either sample. The criterion here is at least three of the samples have to have expression values greater than 2^6.

```
library(genefilter)

##
## Attaching package:  'genefilter'
```

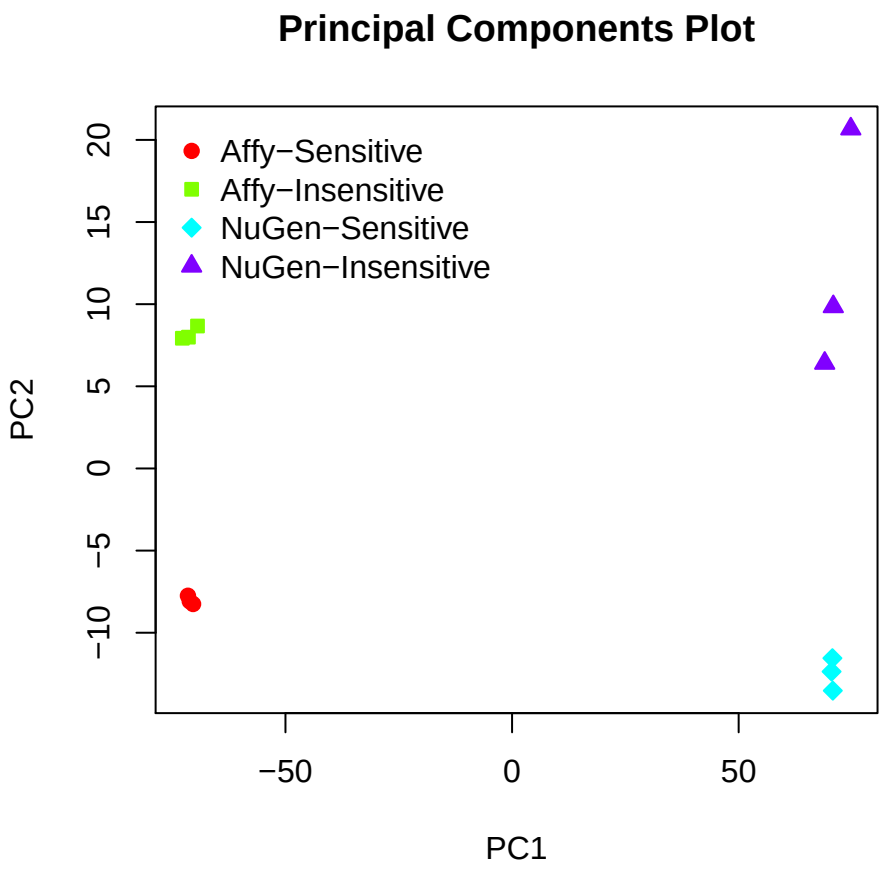

Figure 3: PCA plot

```
##
## The following object is masked from 'package:base':
##
##     anyNA
```

```
f1 <- kOverA(3, 6)
filt <- filterfun(f1)
index <- genefilter(eset, filt)
eset <- eset[index,]
```

After filtering out the 'unexpressed' genes, we fit a linear model and extract contrasts of interest. Explaining the following code is beyond the scope of this vignette – for more information on fitting linear models using *limma*, please see the "LIMMA User's guide".

```
library(limma)
```

```
##
## Attaching package:  'limma'
##
```

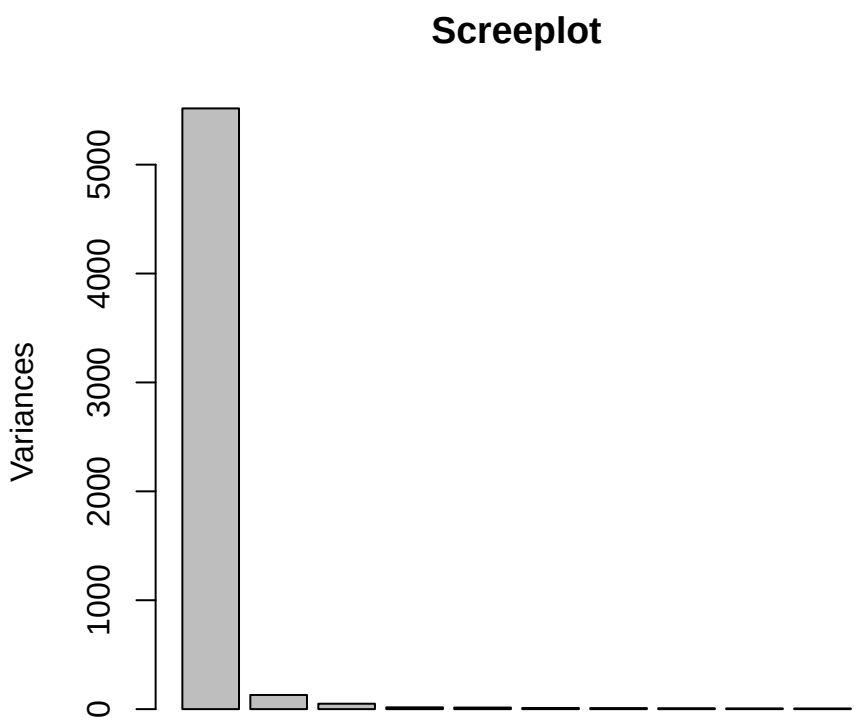

Figure 4: Screeplot

```
## The following object is masked from 'package:BiocGenerics':
##
##     plotMA
```

```
grps <- paste(pData(eset)[,1],
              pData(eset)[,2], sep = ".")
design <- model.matrix(~ 0 + factor(grps))
colnames(design) <- levels(factor(grps))
ugrps <- unique(grps)
contrasts <- matrix(c(1, -1, 0, 0, 0, 0, 1, -1),
                    ncol = 2, dimnames = list(ugrps,
                              paste(ugrps[c(1,3)], ugrps[c(2,4)],
                                    sep = " - ")))
fit <- lmFit(eset, design)
fit2 <- contrasts.fit(fit, contrasts)
```

```
## Warning:  row names of contrasts don't match col names of coefficients
```

```
fit2 <- eBayes(fit2)
```

Printing out the design and contrast matrices may be helpful:

```
design
```

```
##    Affy.Insensitive Affy.Sensitive NuGen.Insensitive NuGen.Sensitive
## 1                 0              1                 0                0
## 2                 0              1                 0                0
## 3                 0              1                 0                0
## 4                 1              0                 0                0
## 5                 1              0                 0                0
## 6                 1              0                 0                0
## 7                 0              0                 0                1
## 8                 0              0                 0                1
## 9                 0              0                 0                1
## 10                0              0                 1                0
## 11                0              0                 1                0
## 12                0              0                 1                0
## attr(,"assign")
## [1] 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$`factor(grps)`
## [1] "contr.treatment"
```

```
contrasts
```

```
##                   Affy.Sensitive - Affy.Insensitive NuGen.Sensitive - NuGen.Insensitive
## Affy.Sensitive                                    1                                   0
## Affy.Insensitive                                 -1                                   0
## NuGen.Sensitive                                   0                                   1
## NuGen.Insensitive                                 0                                  -1
```

Once we have fit the model and extracted contrasts of interest, the next step is to output some results. We might first want to look at a Venn diagram that shows how many genes were differentially expressed in each sample. Note here that if we use the `vennCounts` function in *limma* with *include* = "both", then we will select genes that are significant in both comparisons, but without requiring the genes be differentially expressed in the same direction. In this case it does not make sense to count a gene as being differentially expressed in both sample sets unless the direction is the same. In other words, if a given gene appears to be upregulated in the sensitive samples when we use the Affy *IVT* kit, but downregulated in the sensitive samples when we use the NuGen Ovation kit, it does not make sense to say that the results agree (e.g., are in the intersection of the Venn diagram). Therefore, we will use the `vennCounts2` function, with *method* = "same", which will require the same direction as well.

```
rslt <- decideTests(fit2)
vc <- vennCounts2(rslt, method = "same")
vennDiagram(vc, cex = 0.8)
```

Figure 5 shows the Venn diagram for this analysis.

At this point we may wish to output lists of the genes that are unique to each comparison, as well as the genes that are common to both. To do this, we use the `vennSelect` function.

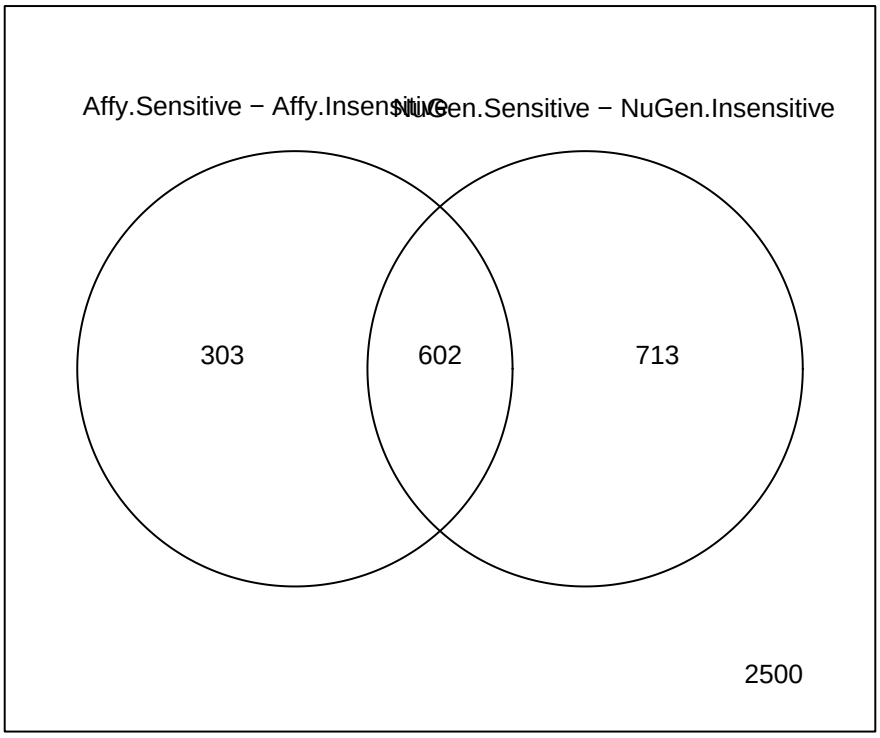

Figure 5: Venn Diagram

```
vennSelect(eset, design, rslt, contrasts, fit2)
```

This will output both HTML and text files containing the gene names, links to various online databases (for the HTML files), and the expression values for the samples in question. The file names will be extracted from the column names of the `TestResults` object (produced as a result of calling `decideTests` above). Note that `decideTests` uses the column names of the contrasts matrix to make the column names of the `TestResults` object, so it is important to set up the contrasts matrix with reasonable names. Reasonable

being defined here as:

- Something that will make sense as the name for the resulting tables.
- Names that will be acceptable as part of a filename for your particular operating system.

Alternatively, we may simply want to output lists of genes that are significant in each of the contrasts at a given p-value and/or fold change.

```
limma2annaffy(eset, fit2, design,
              contrasts, annotation(eset),
              pfilt = 0.05)
```

This will output two HTML tables containing all genes that are significant at an adjusted p-value of 0.05 (default multiplicity correction using false discovery rate [4]). The gene lists will be sorted in descending p-value order, so theoretically the more 'interesting' genes will be at the top of the list. We have the option of outputting text files as well. I generally do so, because it is not uncommon for my clients to want to open these files in a spreadsheet program and do some further exploration, and the HTML tables tend not to work well.

Another analysis that we may wish to perform (although it doesn't make much sense here), is to look for Gene Ontology terms that are 'enriched' in the set of significant genes. The *GOstats* package is quite useful for this sort of analysis, but the output is not always as compact as one might like. My clients generally just want to see a list of GO terms that are enriched, as well as the p-values associated with each term.

We can get the Affy probe IDs for the genes in the intersection of the Venn diagram, and then use those IDs to look for GO terms that are 'enriched' in that set of of probes.

```
index1 <- vennSelect(x = rslt, indices.only = TRUE)[[3]]
probids <- unique(getLL(featureNames(eset)[index1],
                        annotation(eset)))
univ <- unique(getLL(featureNames(eset),
                     annotation(eset)))
params <- new("GOHyperGParams", geneIds = probids,
              universeGeneIds = univ,
              annotation = annotation(eset),
              conditional = TRUE, ontology = "MF")
hyp <- hyperGTest(params)
htmlReport(hyp, file = "GO MF terms.html",
           categorySize = 10)
```

This function outputs an HTML file that can be opened using a web browser.

4 Non-Interactive Analyses

It is relatively simple to write a vignette for interactive analysis using a given package. It is not simple to do the same for a non-interactive analysis because by definition there is no active interaction with R. Therefore, instead of trying to explain things in this vignette, I have placed an example `Sweave` document in the examples directory of this package (R-home/library/affycoretools/examples) that will re-create the above analyses and output a PDF file as well as the HTML and text files. Between this example and the Sweave User Manual, it should be relatively straightforward to figure out how to do something similar for your own analyses.

5 Session information

The version of R and packages loaded when creating this vignette were:

```
R version 3.1.0 Patched (2014-05-21 r65711)
Platform: x86_64-apple-darwin13.1.0 (64-bit)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] limma_3.20.5         genefilter_1.46.1    hgfocuscdf_2.14.0    affycoretools_1.36.1
 [5] GO.db_2.14.0         RSQLite_0.11.4       DBI_0.2-7            AnnotationDbi_1.26.0
 [9] GenomeInfoDb_1.0.2   affy_1.42.2          Biobase_2.24.0       BiocGenerics_0.10.0

loaded via a namespace (and not attached):
 [1] AnnotationForge_1.6.1     BBmisc_1.6                BSgenome_1.32.0
 [4] BatchJobs_1.2             BiocInstaller_1.14.2      BiocParallel_0.6.1
 [7] BiocStyle_1.2.0           Biostrings_2.32.0         Category_2.30.0
[10] DESeq2_1.4.5              Formula_1.1-1             GOstats_2.30.0
[13] GSEABase_1.26.0           GenomicAlignments_1.0.1   GenomicFeatures_1.16.2
[16] GenomicRanges_1.16.3      Hmisc_3.14-4              IRanges_1.22.9
[19] KernSmooth_2.23-12        MASS_7.3-33               Matrix_1.1-4
[22] PFAM.db_2.14.0            R.methodsS3_1.6.1         R.oo_1.18.0
[25] R.utils_1.32.4            R2HTML_2.2.1              RBGL_1.40.0
[28] RColorBrewer_1.0-5        RCurl_1.95-4.1            Rcpp_0.11.2
[31] RcppArmadillo_0.4.300.8.0 ReportingTools_2.4.0      Rsamtools_1.16.1
[34] VariantAnnotation_1.10.2  XML_3.98-1.1              XVector_0.4.0
[37] affyio_1.32.0             annaffy_1.36.0            annotate_1.42.0
[40] biomaRt_2.20.0            biovizBase_1.12.1         bit_1.1-12
[43] bitops_1.0-6              brew_1.0-6                caTools_1.17
[46] cluster_1.15.2            codetools_0.2-8           colorspace_1.2-4
[49] dichromat_2.0-0           digest_0.6.4              edgeR_3.6.2
[52] evaluate_0.5.5            fail_1.2                  ff_2.2-13
[55] foreach_1.4.2             formatR_0.10              gcrma_2.36.0
[58] gdata_2.13.3              geneplotter_1.42.0        ggbio_1.12.5
[61] ggplot2_1.0.0             gplots_2.13.0             graph_1.42.0
[64] grid_3.1.0                gridExtra_0.9.1           gtable_0.1.2
[67] gtools_3.4.1              highr_0.3                 hwriter_1.3
[70] iterators_1.0.7           knitr_1.6                 lattice_0.20-29
[73] latticeExtra_0.6-26       locfit_1.5-9.1            munsell_0.4.2
[76] oligoClasses_1.26.0       plyr_1.8.1                preprocessCore_1.26.1
[79] proto_0.3-10              reshape2_1.4              rtracklayer_1.24.2
[82] scales_0.2.4              sendmailR_1.1-2           splines_3.1.0
[85] stats4_3.1.0              stringr_0.6.2             survival_2.37-7
```

```
[88] tools_3.1.0              xtable_1.7-3             zlibbioc_1.10.0
```

References

[1] Anthony Rossini. Literate Statistical Analysis. In Kurt Hornik and Friedrich Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2001, Technische Universität Wien, Vienna, Austria*, 2001. ISSN 1609-395X. URL: http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/.

[2] Anthony Rossini, Richard Heiberger, Rodney Sparapani, Martin Maechler, and Kurt Hornik. Emacs Speaks Statistics: A Multiplatform, Multipackage Development Environment for Statistical Analysis. *Journal of Computational and Graphical Statistics*, 13:247–261, 2004.

[3] Friedrich Leisch. Dynamic generation of statistical reports using literate data analysis. In W. Haerdle and B. Roenz, editors, *Compstat 2002 - Proceedings in Computational Statistics*, pages 575 – 580. Physika Verlag, 2002.

[4] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57:289–300, 1995.