# Creating annotated output with affycoretools and ReportingTools

James W. MacDonald*

April 24, 2017

## Contents



*jmacdon@u.washington.edu

# 1 Overview

The *affycoretools* package is intended to help people easily create useful output from various analyses. While affycoretools was originally intended for those using Affymetrix microarrays, this is no longer the case. While some functions remain Affy-centric, most are now much more general, and can be used for any microarray or RNA-Seq experiment.

# 2 Introduction

This package has evolved from my work as a service core biostatistician. I routinely analyze very similar experiments, and wanted to create a way to minimize all the cutting and pasting of code that I found myself doing. In addition, I wanted to come up with a good way to make an analysis reproducible, in the sense that I (or somebody else) could easily re-create the results.

In the past this package relied on the *annaffy* package, and was intended to be used in concert with a 'Sweave' document that contained both the code that was used to analyze the data, as well as explanatory text that would end up in a pdf (or HTML page) that could be given to a client. In the intervening period, people have developed other, better packages such as *knitr* and *ReportingTools* that make it much easier to create the sort of output I like to present to my clients.

# 3 Using affycoretools

For this section we will be using the `sample.ExpressionSet` data set that comes with the *Biobase* package. Remember that you can always run this code at home by doing this:

```
library(knitr)
purl(system.file("doc/RefactoredAffycoretools.Rnw", package="affycoretools"))
```

And then you will have a file called RefactoredAffycoretools.R in your working directory that you can either `source` or open with *RStudio* or *Emacs/ESS*, and run by chunk or line by line.

We first load and rename the data:

```
suppressMessages(library(affycoretools))
data(sample.ExpressionSet)
eset <- sample.ExpressionSet
eset
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 26 samples
##   element names: exprs, se.exprs
## protocolData: none
## phenoData
##   sampleNames: A B ... Z (26 total)
##   varLabels: sex type score
##   varMetadata: labelDescription
## featureData: none
```

```
## experimentData: use 'experimentData(object)'
## Annotation: hgu95av2
```

This *ExpressionSet* object is a truncated data set, based on an Affymetrix HG-U95av2 array. There are 26 samples and 500 probesets. We will use the *phenoData* to fit a linear model using *limma*. *comment: We will not cover any aspects of fitting a linear model here; the limma User's Guide covers this topic in depth. In addition, this analysis isn't meant to be correct in any sense; we are just doing this to get some data to annotate and output.*

```
suppressMessages(library(limma))
pd <- pData(phenoData(eset))
design <- model.matrix(~0+type+sex, pd)
colnames(design) <- gsub("type|sex", "", colnames(design))
contrast <- matrix(c(1,-1,0))
colnames(contrast) <- "Case vs control"
fit <- lmFit(eset, design)
fit2 <- contrasts.fit(fit, contrast)
fit2 <- eBayes(fit)
topTable(fit2, 1)[,1:4]
```

```
##                              logFC   AveExpr        t      P.Value
## 31667_r_at                763.1646  735.1835 24.84768 1.429233e-18
## AFFX-HSAC07_X00351_M_st   193.4776  197.6336 23.57778 4.770893e-18
## 31375_at                  368.1255  395.6733 22.26916 1.761763e-17
## 31466_at                  198.3558  195.4360 20.62148 1.011904e-16
## 31597_r_at               1590.9543 1634.2481 20.58837 1.049424e-16
## 31440_at                  597.5251  659.0559 20.51108 1.142742e-16
## 31396_r_at               2344.2449 2504.3938 20.29175 1.457511e-16
## AFFX-hum_alu_at          8005.2783 8681.4000 19.45532 3.768484e-16
## 31391_at                  380.7353  401.1064 19.35000 4.258204e-16
## AFFX-HSAC07_X00351_3_at 4771.0705 4869.6635 19.34299 4.293033e-16
```

At this point we can generate a data.frame, but this data.frame has no annotation, such as gene names or symbols, etc, that say what each probeset is measuring. The *MArrayLM* object that we are calling 'fit2', is capable of containing these data, and will append those data to the `topTable` output.

```
suppressMessages(library(hgu95av2.db))
gns <- select(hgu95av2.db, featureNames(eset), c("ENTREZID","SYMBOL","GENENAME"))
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
## There are one-to many mappings here, so we just
## removed duplicates in a very naive way.
gns <- gns[!duplicated(gns[,1]),]
fit2$genes <- gns
topTable(fit2, 1)[,1:3]
```

```
##                                    PROBEID ENTREZID   SYMBOL
## 31667_r_at                       31667_r_at    10002    NR2E3
## AFFX-HSAC07_X00351_M_st AFFX-HSAC07_X00351_M_st     <NA>     <NA>
## 31375_at                           31375_at     <NA>     <NA>
```

```
## 31466_at                                 31466_at     3128 HLA-DRB6
## 31597_r_at                             31597_r_at     1978 EIF4EBP1
## 31440_at                                 31440_at     6932     TCF7
## 31396_r_at                             31396_r_at     4440     MSI1
## AFFX-hum_alu_at                   AFFX-hum_alu_at     <NA>     <NA>
## 31391_at                                 31391_at     9001     HAP1
## AFFX-HSAC07_X00351_3_at AFFX-HSAC07_X00351_3_at     <NA>     <NA>
```

After adding the annotation data to the `MArrayLM` object, the `topTable` output now contains the appropriate annotation data for each probeset. At this point we can output an HTML table that contains these data.

```
suppressMessages(library(ReportingTools))
htab <- HTMLReport("afile", "My cool results")
publish(topTable(fit2, 1), htab)
finish(htab)
```

```
## [1] "./afile.html"
```

And now we have a HTML table called 'afile.html' in our working directory, that contains the data for our top 10 genes. This table is not particularly interesting, and the *ReportingTools* package already has functionality to just do something like

```
htab <- HTMLReport("afile2", "My cool results, ReportingTools style")
publish(fit2, htab, eset, factor = pd$type, coef = 1, n = 10)
finish(htab)
```

```
## [1] "./afile2.html"
```

and it will automatically generate an annotated table, with some extra plots that show the different groups, and we didn't even have to use `topTable` directly. However, the default plots in the HTML table are a combination of dotplot and boxplot, which I find weird (see afile2.html if you are running this code yourself). Since *ReportingTools* is easily extensible, we can make changes that are more pleasing.

```
d.f <- topTable(fit2, 2)
out <- makeImages(df = d.f, eset = eset, grp.factor = pd$type, design = design,
                  contrast = contrast, colind = 1, repdir = ".")
htab <- HTMLReport("afile3", "My cool results, affycoretools style")
publish(out$df, htab, .mofifyDF = list(entrezLinks, affyLinks))
finish(htab)
```

```
## [1] "./afile3.html"
```

Note that there are two differences in the way we did things. First, we create a `data.frame`, and then decorate it with the plots using the `makeImages` function. This will by default create dotplots (or you can specify boxplots). For the plots to fit in an HTML table, there are no axis labels. However, each plot is also a link, and if you click on it, a larger plot with axis labels will be presented. See 'afile3.html', if you are running this code yourself.

All the little files that get created can get pretty messy, so the default is to put everything into a 'reports' subdirectory, so your working directory stays clean. For this example we over-ride the defaults so we do not have to go searching in subdirectories for our tables.

An alternative parameterization that probably makes more sense is to fit coefficients for each sex/treatment combination.

```
grps <- factor(apply(pd[,1:2], 1, paste, collapse = "_"))
design <- model.matrix(~0+grps)
colnames(design) <- gsub("grps", "", colnames(design))
contrast <- matrix(c(1,-1,0,0,
                     0,0,1,-1,
                     1,-1,-1,1),
                   ncol = 3)
colnames(contrast) <- c("Female_Case vs Female_Control",
                        "Male_Case vs Male_Control",
                        "Interaction")
fit <- lmFit(eset, design)
fit2 <- contrasts.fit(fit, contrast)
fit2 <- eBayes(fit2)
fit2$genes <- gns
```

With this parameterization we can look at intra-sex differences, as well as the interaction (looking for sex-specific changes). This now means that we have a total of three HTML tables to output, which makes things a bit more complex to present. Luckily, this is pretty simple to accomplish. For this step we will now use the default 'reports' subdirectory to keep everything straight. In addition, we will trim down the output a bit.

```
## get a list containing the output for each comparison
out <- lapply(1:3, function(x) topTable(fit2, x))
## process the output to add images
htab <- lapply(1:3, function(x){
    tmp <- HTMLReport(gsub("_", " ", colnames(contrast)[x]), colnames(contrast)[x], "./reports")
    tmp2 <- makeImages(out[[x]], eset, grps, design, contrast, x)
    publish(tmp2$df, tmp, .modifyDF = list(affyLinks, entrezLinks))
    finish(tmp)
    return(tmp)
})

## Now make an index.html file to contain links to the various comps
idx <- HTMLReport("index", "Links to our stuff")
publish(hwriter::hwrite("Univariate comparisons", br = TRUE, header = 2), idx)
publish(Link(htab), idx)
finish(idx)
```

```
## [1] "./index.html"
```

Now there will be an index.html file in the current directory that has individual links to each of the three comparisons we made. This is nice, as you only have to point a client or PI to a single link that they can use to explore all the results.

We are often asked to create a Venn diagram showing overlap between groups. This is pretty simple to do, but it would be nicer to have an HTML version with clickable links, so your PI or end user can see what genes are in each cell of the Venn diagram. As an example, we can generate a Venn diagram comparing

overlapping genes between the male and female comparisons.

```
collist <- list(1:2)
venn <- makeVenn(fit2, contrast, design, collist = collist, adj.meth = "none")
vennlnk <- vennPage(venn, "venn_diagram", "Venn diagram")
```

The `makeVenn` function also returns a `vennCounts` object that we can use in our *knitr* document to generate a Venn diagram there as well (1).

```
vennDiagram(venn[[1]]$venncounts, cex = 0.9)
```

And we can add a link to our index page quite easily.

```
idx <- HTMLReport("index","Links to our stuff")
publish(hwriter::hwrite("Univariate comparisons", br = TRUE, header = 2), idx)
publish(Link(htab), idx)
publish(hwriter::hwrite("Venn diagrams", br = TRUE, header = 2), idx)
publish(Link("Venn1", vennlnk), idx)
finish(idx)
```

```
## [1] "./index.html"
```

There is similar functionality for presenting the results of a GO hypergeometric analysis (`makeGoTable`), and GSEA analysis, based on the `romer` function in *limma* (`runRomer` and `outputRomer`).

## 4 Session information

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

```
toLatex(sessionInfo())
```

- R version 3.4.0 (2017-04-21), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Running under: `Windows Server 2012 R2 x64 (build 9600)`
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.38.0, Biobase 2.36.0, BiocGenerics 0.22.0, IRanges 2.10.0, ReportingTools 2.16.0, S4Vectors 0.14.0, affycoretools 1.48.0, hgu95av2.db 3.2.3, knitr 1.15.1, limma 3.32.0, org.Hs.eg.db 3.4.1
- Loaded via a namespace (and not attached): AnnotationFilter 1.0.0, AnnotationForge 1.18.0, AnnotationHub 2.8.0, BSgenome 1.44.0, BiocInstaller 1.26.0, BiocParallel 1.10.0, BiocStyle 2.4.0, Biostrings 2.44.0, Category 2.42.0, DBI 0.6-1, DESeq2 1.16.0, DelayedArray 0.2.0, Formula 1.2-1, GGally 1.3.0, GO.db 3.4.1, GOstats 2.42.0, GSEABase 1.38.0, GenomeInfoDb 1.12.0, GenomeInfoDbData 0.99.0, GenomicAlignments 1.12.0, GenomicFeatures 1.28.0, GenomicRanges 1.28.0, Hmisc 4.0-2, KernSmooth 2.23-15, Matrix 1.2-9, OrganismDbi 1.18.0, PFAM.db 3.4.1, ProtGenerics 1.8.0, R.methodsS3 1.7.1, R.oo 1.21.0, R.utils 2.5.0, R6 2.2.0, RBGL 1.52.0, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.1-2, Rcpp 0.12.10, Rsamtools 1.28.0, SummarizedExperiment 1.6.0, VariantAnnotation 1.22.0, XML 3.98-1.6, XVector 0.16.0, acepack 1.4.1,

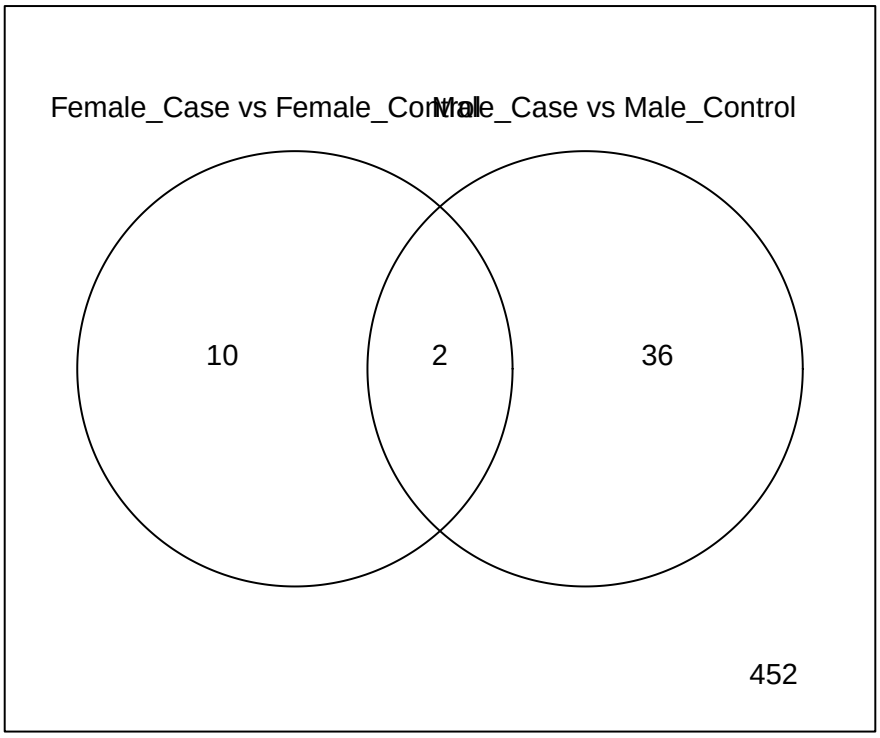


Figure 1: Venn diagram

affy 1.54.0, affyio 1.46.0, annotate 1.54.0, backports 1.0.5, base64enc 0.1-3, biomaRt 2.32.0, biovizBase 1.24.0, bit 1.1-12, bitops 1.0-6, caTools 1.17.1, checkmate 1.8.2, cluster 2.0.6, codetools 0.2-15, colorspace 1.3-2, compiler 3.4.0, data.table 1.10.4, dichromat 2.0-0, digest 0.6.12, edgeR 3.18.0, ensembldb 2.0.0, evaluate 0.10, ff 2.2-13, foreach 1.4.3, foreign 0.8-67, gcrma 2.48.0, gdata 2.17.0, genefilter 1.58.0, geneplotter 1.54.0, ggbio 1.24.0, ggplot2 2.2.1, gplots 3.0.1, graph 1.54.0, grid 3.4.0, gridExtra 2.2.1, gtable 0.2.0, gtools 3.5.0, highr 0.6, htmlTable 1.9, htmltools 0.3.5, htmlwidgets 0.8, httpuv 1.3.3, httr 1.2.1, hwriter 1.3.2, interactiveDisplayBase 1.14.0, iterators 1.0.8, lattice 0.20-35, latticeExtra 0.6-28, lazyeval 0.2.0, locfit 1.5-9.1, magrittr 1.5,

matrixStats 0.52.2, memoise 1.1.0, mime 0.5, munsell 0.4.3, nnet 7.3-12, oligoClasses 1.38.0, plyr 1.8.4, preprocessCore 1.38.0, reshape 0.8.6, reshape2 1.4.2, rmarkdown 1.4, rpart 4.1-11, rprojroot 1.2, rtracklayer 1.36.0, scales 0.4.1, shiny 1.0.2, splines 3.4.0, stringi 1.1.5, stringr 1.2.0, survival 2.41-3, tibble 1.3.0, tools 3.4.0, xtable 1.8-2, yaml 2.1.14, zlibbioc 1.22.0