SmithLabArray

Tim F. Rayner

February 12, 2011

Cambridge Institute of Medical Research
University of Cambridge

Contents

1 Introduction

The **SmithLabArray** package has been developed to help simplify the import, normalisation, filtering, visualisation and analysis of gene expression data using R and Bioconductor. Most of the package functions were written primarily to support Affymetrix data files, although many are more widely applicable. All of the following discussion assumes that you have started up R and loaded the package:

```
> library("SmithLabArray")
```

2 Data import and annotation

This section deals with the import of Affymetrix data and its annotation with accompanying clinical data. The Affymetrix hybridisation system generates CEL files containing probe intensities ("raw" data). Our sample hybridisation pipeline also generates files containing clinical annotation ("ARR" files) from a spreadsheet we create for each hybridisation batch. These files can be used to annotate your CEL data. Alternatively, the CEL file data may be annotated using the clinical data stored in our local clinical database (known for our purposes as "ClinStudyWeb").

2.1 Importing CEL files with ARR files

The `importAffyData` function is the starting point for most data analyses. It reads in a list of ARR files, links them to their corresponding CEL files, reads in those raw data files, and annotates them using the information in the ARR files. This has the advantage that you are not reliant on a connection to the clinical database, and for simple analyses it will often be sufficient. However, for a far greater breadth of clinical annotation, alongside an ability to update that annotation as new information becomes available, the user should consider using the clinical database as described in Section 2.2.

```
> cels <- importAffyData(arr.files=dir(pattern=".*.ARR"))
```

2.2 Annotating CEL files using the clinical database

The ARR files discussed in the previous section typically only contain the most basic patient data required to run simple analyses. For access to clinical and other test results, emergent patient groupings, expanded QC scoring and more, the user should use the `affyClinWeb` function to read in a set of CEL files and annotate them using the clinical database. As new clinical data is deposited in the database, you will want to update this annotation and the `reannoClinWeb` function has been provided for this purpose:

```
> # Initial import:
> cels <- affyClinWeb(files=dir(pattern=".*.CEL"))
>
> # Later...
> cels <- reannoClinWeb(cels)
```

3 Quality Control

Quality control of microarray data is an important step in any analysis, but one which is hard to fully automate. We have developed a QC analysis pipeline (see below), but it is important to remember that it is not infallible. In particular, we generally recommend that you visually inspect each array image (typically available as a JPG file) for defects. Note however that this will typically have already been done for you, with the results stored in the database as a "visual_check" score. If you use the `affyClinWeb` function from Section 2.2 these scores will be available as part of your AffyBatch phenoData (NA indicates no problems found).

3.1 Basic QC checks

The **arrayQualityMetrics** package from Bioconductor implements a range of different quality control measures. We have selected a handful of the most widely applicable, and wrapped them all up in a function called `runAffyQC`:

```
> runAffyQC(cels)
```

The currently recommended workflow is this:

1. Import the CEL file data.
2. Run initial QC checks using `runAffyQC`.
3. Discard obvious outlier arrays.
4. Normalise the data using `vsnrma` (Section 4).
5. Run QC checks on the normalised data using the `arrayQualityMetrics` function from the package of the same name.
6. Take note of any normalised outliers, and observe how they behave in subsequent visualisation and analysis. Discard if necessary. If any samples are discarded it is advisable to re-normalise your data.

3.2 Running QC checks on large data sets

The `runAffyQC` function is very memory- and CPU-intensive. It should typically only be used where the number of hybridisations being checked is less than around 100. For more than 100 hybridisations, you can try using the `batchQC` function which splits a large set of CEL files into smaller subsets, running QC checks on each subset in turn:

```
> batchQC(cels, size=50)
```

4 Data normalisation

For basic data normalisation we generally use the `vsnrma` function to apply a variance-stabilising normalisation (VSN). You should refer to the **vsn** package documentation for further information. There are several more advanced considerations which may come into play when normalising your data, and they are discussed below.

4.1 Normalising large data sets

Normalising large data sets (more than around 100 hybridisations) can be very memory-intensive. Luckily the VSN algorithm allows us to create a reference normalisation using a set of arrays, and then normalise subsequent array batches to that reference. This feature has been used in the `batchVSN` function which splits the input data into batches of a specified size and normalises each in turn:

```
> eset <- batchVSN(cels, size=50)
```

4.2 Correcting for batch effects with ComBatEset

We frequently notice that some batches of hybridisations exhibit a strong batch effect, i.e., that they tend to cluster together even when normalised. This batch effect can be corrected by applying the ComBat algorithm to the normalised data set[1]. The `ComBatEset` function provides a convenient way to apply this algorithm:

```
> eset <- ComBatEset(eset)
```

4.3 Combining Affymetrix cartridge and GeneTitan data

Affymetrix has introduced a new high-througput format, known as GeneTitan, for their array platforms. The GeneTitan system, while using the same probe sequences as the older cartridge arrays, differs considerably in the physical layout of those probes on the chip. This poses a slight problem for co-normalisation of cartridge and GeneTitan data sets. We have generated a mapping between the two platforms which allows us to use VSN to combine the data sets. This co-normalisation is implemented by the `titanVSN` function:

```
> titan.cels <- affyClinWeb(files=dir(pattern=".*.CEL"))
> combined <- titanVSN(cels, titan.cels)
```

Note that the combined data set will still show a strong batch effect, so you will almost certainly need to apply the ComBat algorithm to correct for it (see Section 4.2):

```
> combined <- ComBatEset(combined)
```

As of the time of writing, only the HuGene ST and MoGene ST platforms are supported by this function, but more platforms are easily added upon request.

5 Data filtering

Data filtering can be a useful way to remove noisy probesets from subsequent analysis. Historically, filtering was particularly useful for data normalised using MAS5 or RMA, in which the log transformation of the data tends to amplify noise from low-intensity probe signals. For the most part, we tend not to bother filtering VSN-normalised data since this algorithm doesn't generate such a marked noise amplification. However, for particularly delicate analyses (differential splicing being a case in point) it may be worth applying a filter

to your post-normalisation data set. Note that since normalisation algorithms tend to assume that they will be operating on the complete data set, we do not support filtering the raw data before normalisation.

5.1 Filtering by DABG

The older Affymetrix array platforms, which used both perfect-match (PM) and mis-match (MM) probes for each target sequence, supported a detection-call algorithm as part of the MAS5 software. This algorithm cannot be applied to Gene ST arrays which only have PM probes. Instead, Affymetrix have provided a "Detection Above Background" (DABG) algorithm as part of their APT suite of tools. See `http://www.affymetrix.com/partners_programs/programs/developer/tools/powertools.affx` for a description, and to download these tools. You will need to refer to the Affymetrix documentation for help running these tools; they are outside the scope of this vignette. If you wish to persue this option you will need to use the APT apt-probeset-summarize tool to generate a report using the DABG algorithm, and then feed the output summary file into the `filterByDABG` function alongside your normalised ExpressionSet object:

```
> filtered <- filterByDABG(eset,
+     dabgfile="/complete/path/to/dabg.summary.txt")
```

6 Data export

While R and Bioconductor are powerful tools, they are not all-powerful, and they certainly aren't user-friendly. For those users who wish to export their data in a format they can use with other tools, read on...

6.1 Exporting data to MeV

The MultiExperiment Viewer (MeV) is a Java-based application available from the Dana-Farber Cancer Institute (`http://www.tm4.org/mev/`). Its flexible data import can easily handle the tab-delimited data files generated by the `output2MEV` function. This function will annotate probesets using gene symbols taken from a designated annotation package, and will write out the clinical data annotation as part of the export file header:

```
> output2MEV(filtered, file="output.txt",
+     libname="hugene10sttranscriptcluster.db")
```

7 Gene expression analysis

We can barely scratch the surface of the options available in Bioconductor when it comes to expression analysis. The analysis functions in the **SmithLabArray** package are based around those in the **limma** package. While the latter set of functions requires that you create a design matrix, — and also a contrast matrix in most circumstances — the **SmithLabArray** functions attempt to construct such matrices for you, greatly simplifying the procedure. However,

for full control over the statistical tests run, it is still worth consulting the **limma** documentation and performing the analysis by hand.

The `deriveSignature` function will take an ExpressionSet, the name of an experimental factor column from that object's phenoData, and the name of the factor value (condition) of interest. For example, using the **ALL** dataset available as part of Bioconductor:

```
> library("ALL")
> data("ALL")
> selSamples <- ALL$mol.biol %in% c("ALL1/AF4", "E2A/PBX1")
> ALLs <- ALL[, selSamples]
> sigs <- deriveSignature(ALLs, "mol.biol", "ALL1/AF4", lfc = 0)
```

The *sigs* object is a list containing two data frames; one for probesets upregulated in the condition of interest, and one with downregulated probesets.

Note that comparing two conditions such as this is the simplest possible case. When more than one condition exists within an experimental factor (e.g. comparing multiple immune system cell types) the `deriveSignature` function will calculate the intersection of the genelists generated by comparing the condition of interest to each of the other conditions in turn. In this way, the returned signature will represent those genes uniquely up- and downregulated in the designated condition, compared to all other conditions.

This package provides two other functions which may be of interest to the user. The first, `getVariableSignatures`, iterates over the conditions in the specified phenoData factor, running `deriveSignature` on each. The second, `getAllSignatures`, attempts to identify all phenoData factors of interest and then run `getVariableSignatures` on each. This second function is inherently less reliable than the other functions in the section, owing to the wide variety of information which may be stored in a typical ExpressionSet phenoData. It should therefore be used with considerable caution.

8 Data visualisation

The Bioconductor package provides a huge range of data visualisation tools. In this section we discuss a couple of functions which provide a simple introduction to this field.

8.1 Simple heirarchical cluster heat maps

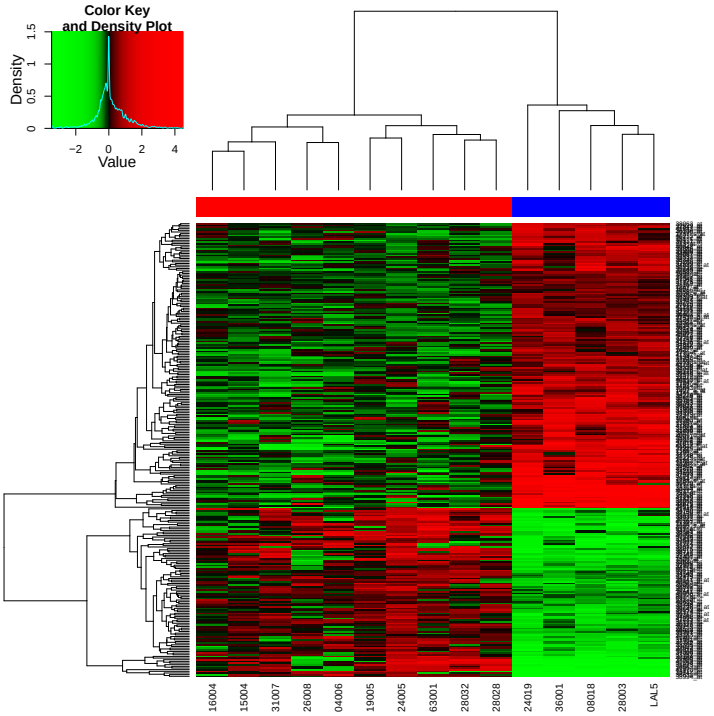
There are two functions for drawing cluster heat maps in the **SmithLabArray** package. The first and most general of these is `quantileColorHM`, which is simply a wrapper for the `heatmap.2` function from the **gplots** package. It ensures an even colour distribution throughout the heatmap by adjusting the colours used based on quantiles calculated from the intensity distribution of the input data. It also provides an option to median-centre the data before clustering.

We can collapse our differential gene signature list into a simple character vector of probeset identifiers:

```
> wanted <- as.character(unlist(lapply(sigs, subset, select = "id",
+     drop = TRUE)))
```

We then use that character vector to draw a heatmap using a subset of the main ExpressionSet:

```
> ALLs <- ALLs[featureNames(ALLs) %in% wanted, ]
> data(hmcol)
> quantileColorHM(ALLs, col = rgcol, center = TRUE, trace = "none",
+     ColSideColors = ifelse(ALLs$mol.biol == "ALL1/AF4", "red",
+         "blue"))
```



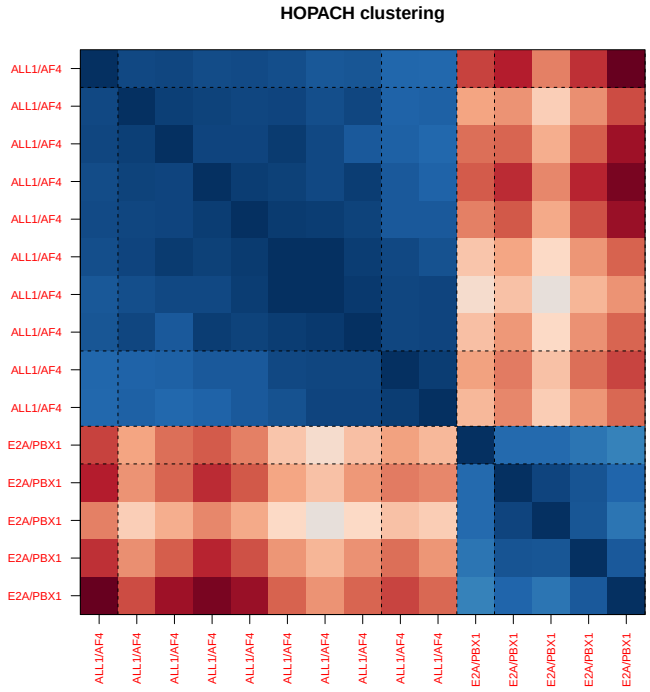
The stark contrast between groups shown in this heatmap is unsurprising, since we have deliberately plotted only those genes which are differentially expressed between the two groups.

A second function, `drawMadFilteredHeatMap`, is available as a way to reduce the size of the data set being clustered. This function calculates the most variable genes by median absolute deviation (MAD) and passes them to the `quantileColorHM` function. The default is to take the top 1500 genes, but this can of course be changed if desired.

8.2 HOPACH detection of optimal group numbers

The HOPACH clustering method works a little differently to the heirarchical clustering described in Section 8.1; instead of calculating distances between samples and using those to build a dendrogram of relatedness, HOPACH will attempt to build the "ideal" clustering based on optimising within-cluster correlations. See the hopach package for more information on this algorithm. The `plotArrayHopach` function gives a simple way to generate HOPACH cluster heatmaps. Using this function with a kmax=2 argument is recommended:

```
> hobj <- plotArrayHopach(ALLs, covariates = c("mol.biol"), kmax = 2,
+     col = hmcol)
```



Dashed lines indicate the boundaries between optimal clusters. The clustering figure here shows that there remain potentially significant subgroups within the main two *ALLs* clusters.

9 Conclusions

We have discussed the main features of the **SmithLabArray** package, and seen how they fit into a basic data analysis workflow. As has already been mentioned, these features only address a very small subset of what is possible with R and Bioconductor. As your analyses become more sophisticated you will likely want to read the vignettes for the various packages used here, in particular **limma**, **Biobase**, **affy**, **hopach** and **gplots**. These vignettes will help you develop your own approaches to analysing your data.

10 Acknowledgements

With thanks to members of the Smith Lab for helpful suggestions, code snippets, and bug reports, and the Bioconductor project for creating and maintaining an indispensable set of tools.

References

[1] W. E. Johnson, A. Rabinovic, and C. Li (2007). Adjusting batch effects in microarray expression data using Empirical Bayes methods. *Biostatistics* 8(1):118–127

11 Session information

The version number of R and packages loaded for generating the vignette were:

```
R version 2.12.0 (2010-10-15)
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

locale:
[1] C

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

other attached packages:
[1] ALL_1.4.7           SmithLabArray_0.5.2 limma_3.6.6
[4] Biobase_2.10.0

loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.12.0  DBI_0.2-5             RColorBrewer_1.0-2
 [4] RSQLite_0.9-3         XML_3.2-0             affy_1.28.0
 [7] affyio_1.18.0         annotate_1.28.0       cluster_1.13.2
[10] gdata_2.8.0           genefilter_1.32.0     gplots_2.8.0
[13] grid_2.12.0           gtools_2.6.2          hopach_2.10.0
[16] lattice_0.19-13       preprocessCore_1.12.0 splines_2.12.0
[19] survival_2.36-1       tools_2.12.0          vsn_3.18.0
[22] xtable_1.5-6
```