

ExpressionView

Andreas Lüscher

May 26, 2010

Contents

1	Introduction	1
2	Loading the gene expression data	1
3	Find biclusters	2
3.1	Iterative Signature Algorithm (ISA)	2
3.2	Algorithms of the <code>Biclust</code> package	2
3.3	External clustering programs	3
4	Order	3
5	Export	4
6	Visualize	4
7	Using ExpressionView with non-gene expression data	7
8	Session information	8

1 Introduction

Clustering genes according to their expression profiles is an important task in analyzing microarray data. In this tutorial, we explain how to use ExpressionView, an R package designed to interactively explore biclusters identified in gene expression data, in conjunction with the Iterative Signature Algorithm (ISA) [1] and the biclustering methods available in the `Biclust` package [2].

2 Loading the gene expression data

The `ExpressionView` package requires the gene expression data to be available in the form of a BioConductor `ExpressionSet`. In this tutorial we will use the BioConductor sample data from a clinical trial in acute lymphoblastic leukemia provided by the `ALL` package.

```
> library(ALL)
> library(hgu95av2.db)
> data(ALL)
```

The data set contains 128 samples and 12625 features.

3 Find biclusters

There are many biclustering algorithms described in the literature [3]. All of them aim to reduce the complexity of the gene expression data by identifying suitable groups of genes and conditions that are co-expressed. In this tutorial show how to use **ExpressionView** with some of the available biclustering algorithms.

3.1 Iterative Signature Algorithm (ISA)

The ISA [1] for gene expression data is implemented in the **eisa** package:

```
> library(eisa)
```

To run the ISA for the given data set, simply call the **ISA** function on the **ExpressionSet** object:

```
> set.seed(5)
> modules <- ISA(ALL)
```

Depending on your computing resources, this should take roughly two minutes. If you do not want to wait that long, you can shorten the calculation by selecting the thresholds for genes and conditions:

```
> threshold.genes <- 2.7
> threshold.conditions <- 1.4
> set.seed(5)
> modules <- ISA(ALL, thr.gene = threshold.genes,
  thr.cond = threshold.conditions)
```

If you leave the thresholds undefined, as in the first example, the ISA runs with the default values, i.e., **thr.gene**=c(2,2.5,3,3.5,4) and **thr.cond**=c(1,1.5,2,2.5,3). In both cases, the random number generator is initialized manually using **set.seed(5)**, to give reproducible results. The **isa** function returns an **ISAModules** object. Typing its name returns a brief summary of the results:

```
> modules
```

An **ISAModules** instance.

```
Number of modules: 5
Number of features: 3427
Number of samples: 128
Gene threshold(s): 2.7
Conditions threshold(s): 1.4
```

This object can be directly used with the functions of the **ExpressionView** package. See Section 4 for details.

3.2 Algorithms of the Biclust package

The **biclust** package implements several biclustering algorithms in a unified framework. It uses the **Biclust** class to store a set of biclusters. Let us use the Plaid Model Bicluster Algorithm [4] on the **ALL** data set

```
> library(biclust)
> biclusters <- biclust(exprs(ALL), BCPlaid(), fit.model = ~m +
  a + b, verbose = FALSE)
```

Biclust objects can be directly used with the **ExpressionView** functions. Alternatively, they can be converted to **ISAModules** objects, using the standard **as** R function:

```
> as(biclusters, "ISAModules")
```

An ISAModules instance.

```
Number of modules: 3
Number of features: 12625
Number of samples: 128
Gene threshold(s):
Conditions threshold(s):
```

results an ISAModules object.

3.3 External clustering programs

Since the structure of biclustering results is independent of the applied method, it is straightforward to import results obtained from external clustering programs and convert them to **ISAModules**. To illustrate the conversion, let us consider the sample data and **randomly** assign the 12625 genes and 128 samples to 5 modules. The resulting modules can be described by two binary matrices

```
> modules.genes <- matrix(as.integer(runif(nrow(ALL) *
  length(modules)) > 0.8), nrow = nrow(ALL))
> modules.conditions <- matrix(as.integer(runif(ncol(ALL) *
  length(modules)) > 0.8), nrow = ncol(ALL))
```

indicating if a given gene *i* is contained in module *j* if `modules.genes[i,j] ≠ 0`. Using these matrices, it is straightforward to create an **ISAModules** object:

```
> new("ISAModules", genes = modules.genes, conditions = modules.conditions,
  rundata = data.frame(), seeddata = data.frame())
```

An ISAModules instance.

```
Number of modules: 5
Number of features: 12625
Number of samples: 128
Gene threshold(s):
Conditions threshold(s):
```

4 Order

To present the tens of possibly overlapping biclusters in a visually appealing form, it is necessary to reorder the rows and columns of the gene expression matrix in such a way that biclusters form contiguous rectangles. Since for more than two mutually overlapping biclusters, it is in general impossible to find such an arrangement, one has to make concessions. In contrast methods that propose to repeat rows and columns as necessary to achieve this goal [5], we prefer to optimize the arrangement within the original data by maximizing the area of the largest contiguous biclusters.

The **OrderEV** function implemented in the **ExpressionView** package determines the optimal order of the gene expression matrix for a given set of biclusters. It can be called with **ISAModules** or **Bi-clust** objects as the first argument:

```
> library(ExpressionView)
> optimalorder <- OrderEV(modules)
```

The result is a list containing various mappings between the original data and the optimal arrangement. Note that the genes and the samples can be ordered separately. Apart from reordering the full gene expression matrix, the algorithm also determines the best arrangement of individual biclusters. The mapping of the genes and the samples contained in bicluster *i* can be accessed by

```
> optimalorder$genes[i + 1]
> optimalorder$samples[i + 1]
```

The first elements of the lists contain the optimal ordering of the complete matrix. By default, the `OrderEV` function runs for roughly one minute, this might not be sufficient to find an appropriate order for data containing many overlapping biclusters. The status of the ordering is stored in

```
> optimalorder$status
```

```
$genes
[1] 1 1 1 1 1 1
```

```
$samples
[1] 1 1 1 1 1 1
```

If the status is set to 1, the algorithm has found the optimal solution. A 0 indicates that the calculation could not be terminated within the given timeframe. The `OrderEV` function accepts two additional parameters to circumvent the problem of partial alignment: One can start the ordering from a given initial configuration, i.e., the result of a previous arrangement by defining the *initialorder* argument

```
> optimalorderp <- OrderEV(modules, initialorder = optimalorder,
  maxtime = 120)
```

and one can increase the time limit by specifying *maxtime*. Note that the time is indicated in seconds and cannot be smaller than 1.

5 Export

The `ExportEV` function allows the user to combine the available data and export it to an XML file that can be read by the Flash applet:

```
> ExportEV(modules, ALL, optimalorder, filename = "file.evf")
```

The function gathers the data contained in the `ExpressionSet` *ALL*, orders it according to the optimal arrangement *optimalorder* and adds the biclusters defined in *modules*. The output is an uncompressed XML file that can be opened with any text viewer. We have chosen to use the extension *.evf* (for ExpressionView file) for the data files. This extension is associated with the stand-alone version of the viewer, so that one can simply double-click on such a file to launch the program and load the data. The file association is the reason why we do not use the *.xml* extension. A description of the XML layout can be found on the ExpressionView website at <http://www.unil.ch/cbg/ExpressionView>. Before exporting the data, the `ExportEV` function automatically calculates GO [6] and KEGG [7] enrichments for the given biclusters.

6 Visualize

The ExpressionView Flash applet can be launched from the R environment:

```
> LaunchEV()
```

Video tutorials describing how to use the applet can be found on the ExpressionView website at <http://www.unil.ch/cbg/ExpressionView>. The screenshot shown in Fig. 1 and the description below illustrate the main features of the applet:

- a** Opens an ExpressionView data file. Note that before opening a new data file, you should restart the applet, i.e., refresh your browser window.
- b** Exports the current view to a pdf file. The file also includes the title (o) of the gene expression data.
- c** Exports the data of the currently viewed module (=bicluster) to a CSV file, that can be opened as a spreadsheet.
- d** In inspect mode, you can use the mouse to explore the gene expression data. The information about the data under the mouse pointer is shown in the Info Panel (t).
- e, f** Zoom and pan modes allow you to restrict the view to a particular part of the gene expression data.
- e** In zoom mode, you can also use keyboard shortcuts: **a** to auto-zoom onto the modules and **e** to see the whole data. In addition to the simple zoom-in feature, you can also use the mouse to select the rectangular area you want to have a closer look at.
- f** Pan mode.
- g, h, i, j** Module highlighting and viewing. It is in general impossible to present mutually overlapping biclusters as single rectangles. They are made up of a collection of rectangles. The ordering algorithm used in the R package realigns the gene expression matrix in a way that maximizes the total area of the largest rectangle in every bicluster. The outlines of these parts are drawn in a slightly brighter color than the background, making them easily recognizable.
- g, h** Modules are highlighted as the user moves the mouse over the gene expression data. The two check boxes allow you to choose between highlighting all the parts of a module (Filling) or alternatively only the largest rectangle (Outline). You can also turn it off completely. For data sets with many modules, it can be helpful to restrict highlighting to Outline.
- i, j** Similar to the highlighting, these two check boxes allow you to show either all the parts of the modules (Filling) or only the largest rectangles (Outline). By shift-clicking one of the check-boxes you can switch between showing only the modules or only the gene expression data.
- k** Sets the visibility of the modules layer. Moving the slider to the left fades out the gene expression data, thus focusing on the Biclusters, while towards the opposite direction, the gene expression data moves to the foreground.
- l** Realigns the windows at their initial positions.
- m** Puts the program in fullscreen mode. Note that due to security reasons, it is impossible to enter text in this mode. On Mac OS X, a bug in Flash player prevents you from exporting data in fullscreen mode.
- n** Opens the ExpressionView website, from where you can download sample files and tutorials.
- o** Description and dimensions of the data set.
- p** Modules navigator. The Global tab is always available and shows the complete gene expression data. Additional tabs appear as you open individual modules. To close a module, simply move the mouse over the tab and click the close button that appears.
- q, r, s** Selected genes, samples and modules. The highlighting reflects the selection in the tables (w). The selection is maintained when switching tabs (p).
- q** Selected genes (=probes).

- r** Selected samples (=conditions).
- s** Selected modules (=biclusters).
- t** Info panel showing the data associated with the current mouse position. The GO and KEGG list contain the five most significant categories and pathways associated with the modules under the mouse pointer.
- u** Lists the selected genes, samples and modules, together with the intersecting modules.
- u1** Opens intersecting modules.
- u2** Clears the selection.
- v** Lists the selected GO categories and KEGG pathways
- w** List navigator. Note that depending on the view (p), the lists only show genes and samples contained in the currently viewed module. Modules can also be opened by double-clicking on the corresponding row. The Experiment tab contains a brief description of the data.
- x** Searches the tables for a given expression and restricts the view to the matching entries. The search function uses Perl-style **regular expressions**. By default, the search functions is applied to the whole table. To restrict it to a particular column, shift-click the corresponding column header.
- z** Select a column header to sort the entries according to that column. Shift-click to restrict the search function to that column.

7 Using ExpressionView with non-gene expression data

While ExpressionView is designed to work with gene expression data available in the form of a Bioconductor `ExpressionSet`, it can also be used to visualize other data. Let us for instance use in-silico data generated by the `isa` package with dimensions 50×500 containing 10 overlapping modules:

```
> library(ExpressionView)
> m <- 50
> n <- 500
> M <- 10
> data <- isa.in.silico(num.rows = m, num.cols = n,
  num.fact = M, noise = 0.1, overlap.row = 5)[[1]]
> modules <- isa(data)
```

The `ExportEV` uses the named list provided by the `description` variable to label the data. First, let us annotate the rows and columns of the data set

```
> rownames(data) <- paste("row", seq_len(nrow(data)))
> colnames(data) <- paste("column", seq_len(ncol(data)))
```

Next, we assign the meta data associated with the rows of the data matrix. In this example we use 5 tags labelled "row tag":

```
> rowdata <- outer(1:nrow(data), 1:5, function(x,
  y) {
  paste("row description (", x, ", ", y, ")",
    sep = "")
})
```

```

})
> rownames(rowdata) <- rownames(data)
> colnames(rowdata) <- paste("row tag", seq_len(ncol(rowdata)))

```

And similarly for the columns, using 10 “column tags”:

```

> coldata <- outer(1:ncol(data), 1:10, function(x,
  y) {
    paste("column description (", x, ", ", y,
          ")", sep = "")
  })
> rownames(coldata) <- colnames(data)
> colnames(coldata) <- paste("column tag", seq_len(ncol(coldata)))

```

To finish the description, we add some general information and merge it with the above tables to get a single named list:

```

> description <- list(experiment = list(title = "Title",
  xaxislabel = "x-Axis Label", yaxislabel = "y-Axis Label",
  name = "Author", lab = "Address", abstract = "Abstract",
  url = "URL", annotation = "Annotation", organism = "Organism"),
  coldata = coldata, rowdata = rowdata)

```

When dealing with gene expression data, the `xaxislabel` is equal to “genes” and the `yaxislabel` is “samples”. Finally, we export the data set to an ExpressionView file:

```

> ExportEV(modules, data, filename = "file.evf",
  description = description)

```

Simply load this file with the Flash applet and check where the various labels appear.

8 Session information

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

- R version 2.11.0 (2010-04-22), x86_64-pc-mingw32
- Locale: LC_COLLATE=English_United States.1252, LC_CTYPE=English_United States.1252, LC_MONETARY=English_United States.1252, LC_NUMERIC=C, LC_TIME=English_United States.1252
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, tools, utils
- Other packages: ALL 1.4.7, AnnotationDbi 1.10.1, biclust 0.9.1, Biobase 2.8.0, bitops 1.0-4.1, Category 2.14.0, caTools 1.10, colorspace 1.0-1, DBI 0.2-5, eisa 1.0.0, ExpressionView 1.1.0, genefilter 1.30.0, hgu95av2.db 2.4.1, isa2 0.2.1, MASS 7.3-6, org.Hs.eg.db 2.4.1, RSQLite 0.9-0, vcd 1.2-8
- Loaded via a namespace (and not attached): annotate 1.26.0, GO.db 2.4.1, graph 1.26.0, GSEABase 1.10.0, KEGG.db 2.4.1, RBGL 1.24.0, splines 2.11.0, survival 2.35-8, XML 3.1-0, xtable 1.5-6

References

- [1] Sven Bergmann, Jan Ihmels, and Naama Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review E*, 67:031902, 2003.
- [2] Sebastian Kaiser and Friedrich Leisch. A toolbox for bicluster analysis in R. Technical Report 028, Department of Statistics, University of Munich, 2008.
- [3] S.C. Madeira and A.L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1:24–45, 2004.
- [4] Heather Turner, Trevor Bailey, and Wojtek Krzanowski. Improved biclustering of microarray data demonstrated through systematic performance tests. *Computational Statistics & Data Analysis*, 48:235–254, 2004.
- [5] Gregory Grothaus, Adeel Mufti, and TM Murali. Automatic layout and visualization of biclusters. *Algorithms for Molecular Biology*, 1:15, 2006.
- [6] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25 – 29, 2000.
- [7] Minoru Kanehisa, Susumu Goto, Shuichi Kawashima, Yasushi Okuno, and Masahiro Hattori. The KEGG resource for deciphering the genome. *Nucleic Acids Research*, 32:D277–D280, 2004.