Bioconductor basics tutorial

Sandrine Dudoit and Robert Gentleman

June 24, 2002

1 Preliminaries

This document is a brief tutorial for three packages released as part of the Bioconductor project. These and all other Bioconductor packages (including data packages) can be downloaded from the Bioconductor website, `www.bioconductor.org`. The specific packages that will be considered include `Biobase`, `genefilter`, and `annotate`. These packages provide basic tools needed to process DNA microarray expression data. Finally, at the end of the tutorial, we briefly demonstrate the use of standard R clustering functions. A more complete overview of machine learning using R is given in another tutorial and also in the lectures on classification and clustering. Other packages used in this tutorial can be obtained from CRAN `www.cran.r-project.org`.

More can be learned about specific functions from the R help system; either `help(name)` or `?name` will yield the help files. Bioconductor packages have additional documentation in the form of vignettes and HowTo's that are supplied in the `/inst/doc` directory of each package. These documents can also be obtained from the Bioconductor website. In addition, there are many other resources on how to use R, including R manuals on CRAN. Documentation and textbooks for S–Plus, such as Venables and Ripley (1999), are also appropriate.

You will need to load the following R and Bioconductor packages.

```
R> library(XML)
R> library(Biobase)
R> library(annotate)
R> library(genefilter)
R> library(golubEsets)
R> library(ctest)
R> library(MASS)
R> library(cluster)
```

2 Golub et al. dataset

The functionality of the three packages is demonstrated using the Affymetrix gene expression dataset of Golub et al. (1999). The data are available from the authors in an online repos-

itory (http://www-genome.wi.mit.edu/mpr/data_set_ALL_AML.html) and were included in an R data package, golubEsets, suitable for use in this tutorial. This dataset comes from a study of gene expression in two types of acute leukemias: acute lymphoblastic leukemia (ALL) and acute myeloid leukemia (AML). Gene expression levels were measured using Affymetrix high–density oligonucleotide arrays (HU6800 chip) containing probes for approximately 6,800 human genes and ESTs. The chip actually contains 7,129 different probe sets; some of these map to the same genes and others are there for quality control purposes. The data comprise 47 cases of ALL (38 B–cell ALL and 9 T–cell ALL) and 25 cases of AML. Samples are divided into a learning set with 38 observations and a test set of 34 observations.

In this tutorial, we make the assumption that the data have been suitably pre–processed, i.e., image analysis, normalization, and computation of expression measures were already performed. You are referred to either the affy package from Bioconductor or to dChip www.dchip.org for relevant software and documentation for pre–processing of Affymetrix data, and to marrayNorm for pre–processing of cDNA array data.

3 Biobase: Classes exprSet and phenoData

We can obtain the expression measures in the form of exprSets by using the data function

```
R> data(golubTrain)
R> data(golubTest)
R> data(golubMerge)
```

The following commands provide information on the object golubTrain (similarly for golubTest and golubMerge)

```
R> class(golubTrain)

[1] "exprSet"

R> slotNames(golubTrain)

[1] "exprs"       "se.exprs"    "phenoData"   "description" "annotation"
[6] "notes"

R> golubTrain

Expression Set (exprSet) with
        7129 genes
        38 samples
                 phenoData object with 11 variables and 38 cases
         varLabels
                Samples: Samples
                ALL.AML: ALL.AML
```

```
        BM.PB: BM.PB
        T.B.cell: T.B.cell
        FAB: FAB
        Date: Date
        Gender: Gender
        pctBlasts: pctBlasts
        Treatment: Treatment
        PS: PS
        Source: Source
```

To extract only information on the mRNA samples

```
R> phenoTrain <- phenoData(golubTrain)
R> class(phenoTrain)

[1] "phenoData"

R> slotNames(phenoTrain)

[1] "pData"     "varLabels"

R> phenoTrain

    phenoData object with 11 variables and 38 cases
    varLabels
        Samples: Samples
        ALL.AML: ALL.AML
        BM.PB: BM.PB
        T.B.cell: T.B.cell
        FAB: FAB
        Date: Date
        Gender: Gender
        pctBlasts: pctBlasts
        Treatment: Treatment
        PS: PS
        Source: Source
```

Data on only the first 3 chips and 10 genes can be obtained using the subsetting operator "["

```
R> golubTrain[1:3, 1:10]

Expression Set (exprSet) with
    3 genes
    10 samples
        phenoData object with 11 variables and 10 cases
```

```
varLabels
       Samples: Samples
       ALL.AML: ALL.AML
       BM.PB: BM.PB
       T.B.cell: T.B.cell
       FAB: FAB
       Date: Date
       Gender: Gender
       pctBlasts: pctBlasts
       Treatment: Treatment
       PS: PS
       Source: Source
```

The matrix of expression measures can be accessed using either the general `@` operator or the `slot` function from the `methods` package. It can also be accessed using the accessor function `exprs` provided in `Biobase`.

```
R> X <- golubTrain@exprs
R> X <- slot(golubTrain, "exprs")
R> X <- exprs(golubTrain)
R> dim(X)

[1] 7129   38
```

The `@` operator and `slot` function can also be used to modify slots of these objects

```
R> slot(golubTrain, "annotation")

[1] "affy"

R> slot(golubTrain, "annotation") <- "HU6800"
R> golubTrain@annotation

[1] "HU6800"
```

4 Genefilter

4.1 Preliminary gene filtering

One of the first tasks that must be carried out when analyzing expression data is to filter out those genes that are unlikely to be of interest. The approach taken in `genefilter` is to separate the task of specifying the filters that will be used from the task of applying them. Users must first specify which filtering functions they want to use, assemble them using `filterfun`, and then apply them using `genefilter`.

Following Golub et al. (1999) (Pablo Tamayo, pers. comm.), we will apply three pre–processing steps to the normalized matrices of intensity values available on the website: (i) thresholding: floor of 100 and ceiling of 16,000; (ii) filtering: exclusion of genes with $\max / \min \leq 5$ or $(\max - \min) \leq 500$, where max and min refer respectively to the maximum and minimum intensities for a particular gene across mRNA samples; (iii) base–10 logarithmic transformation. In addition, we will (iv) standardize the data so that the expression measures for each array have mean 0 and variance one across genes. After applying these steps to the training set, the expression measures can be summarized by a $3,051 \times 38$ matrix $\mathbf{X} = (x_{gi})$, where x_{gi} denotes the expression level for gene g in mRNA sample i.

The following commands may be used to apply steps (i)–(iv) to the training set and store the resulting combined expression and sample data in the `exprSet` object `golubTrainSub`.

(i) Thresholding.

```
R> X <- exprs(golubTrain)
R> X[X < 100] <- 100
R> X[X > 16000] <- 16000
```

The matrix `X` now has all values that were less than 100 replaced by 100 and all values that were larger than 16,000 replaced by 16,000.

(ii) Filtering. Now we want to select those genes that satisfy the second requirement of Tamayo. We first define the filter function `mmfilt` and use the two main functions, `filterfun` and `genefilter`.

```
R> mmfilt <- function(r = 5, d = 500, na.rm = TRUE) {
+      function(x) {
+          minval <- min(x, na.rm = na.rm)
+          maxval <- max(x, na.rm = na.rm)
+          (maxval/minval > r) && (maxval - minval > d)
+      }
+  }
R> mmfun <- mmfilt()
R> ffun <- filterfun(mmfun)
R> good <- genefilter(X, ffun)
R> sum(good)

[1] 3051

R> X <- X[good, ]
```

(iii) Log–transformation.

```
R> X <- log10(X)
```

(iv) Standardization. Finally, we standardize the chips and create a new instance of the `exprSet` class

```
R> X <- scale(X)
R> golubTrainSub <- golubTrain[good, ]
R> slot(golubTrainSub, "exprs") <- X
R> golubTrainSub

Expression Set (exprSet) with
        3051 genes
        38 samples
                 phenoData object with 11 variables and 38 cases
         varLabels
                Samples: Samples
                ALL.AML: ALL.AML
                BM.PB: BM.PB
                T.B.cell: T.B.cell
                FAB: FAB
                Date: Date
                Gender: Gender
                pctBlasts: pctBlasts
                Treatment: Treatment
                PS: PS
                Source: Source
```

4.2 Filtering genes by t–test

It is of interest to identify genes that are differentially expressed in ALL and AML patients. We will filter genes according to their nominal p–values for a two–sample Welch t–test comparing expression measures in ALL and AML patients. We can use the `ttest` filter function from the `genefilter` package to identify genes with nominal p–values less than 0.0001. Note that this is a very rough analysis. These p–values are nominal and should be adjusted to account for multiple testing; the `multtest` package can be used for more in depth multiple testing procedures.

```
R> cl <- golubTrainSub$ALL.AML
R> table(cl)

cl
ALL AML
 27  11

R> ffun <- filterfun(ttest(cl, p = 1e-04))
R> smallp<- genefilter(exprs(golubTrainSub), ffun )
R> sum(smallp)
```

```
[1] 163
```

Let us compute a number of statistics for these 163 genes, such as, ALL and AML mean expression levels, t–statistics, and nominal p–values.

```
R> X2 <- exprs(golubTrainSub[smallp, ])
R> stats <- apply(X2, 1, function(z) {
+      res <- t.test(z ~ cl)
+      c(res$estimate, res$statistic, res$parameter, res$p.value)
+  })
R> stats <- data.frame(t(stats))
R> names(stats) <- c("ALL mean", "AML mean", "t-statistic", "df",
+      "p-value")
R> stats <- stats[order(stats$p), ]
```

5 Annotate

We will now use functions from the `annotate` package to relate expression data to biological metadata from databases such as LocusLink and PubMed. Note that this part of the Bioconductor project is undergoing very rapid development and may well be different by the time you read this. Please consult the online documentation and relevant vignettes and HowTo's.

5.1 LocusLink query

To obtain LocusLink IDs (LocusID) for the 163 genes with small p–values, first use the `read.annotation` function to read in annotation data from the file corresponding to the HU6800 chip used in this experiment. This function returns the environment into which the annotation data are read. The LocusLink IDs for a subset of interesting genes can be obtained using the `multiget` function. LocusLink can then be queried for these genes using the `locuslinkByID` function.

```
ll <- read.annotation("hgu68ll") # Read in locus link annotation data
gname <- row.names(X2)           # Names of genes with small p--value
LocusID<-multiget(gname, env=ll) # Locus link IDs
locuslinkByID(LocusID)           # Query LocusLink
```

The LocusLink page is shown in the browser for the first accession number in LocusID. Other results will be accessible through the "View" pull–down menu on the LocusLink page.

5.2 PubMed query

Given a set of PubMed IDs (PMID), the `pubmed` function can either have a browser display a PubMed page for those identifiers, or return an `XMLDocument` object with the same information. For instance, to view entries for the following three PMIDs

```
pubmed("11695507","9923029","11769896",disp="browser")
```

To store the corresponding abstracts in (a list of) objects of class `pubMedAbst`

```
x <- pubmed("11695507","9923029","11769896")
a <- xmlRoot(x)
numAbst <- length(xmlChildren(a))
absts1 <- list()
for (i in 1:numAbst) {
  absts1[[i]] <- buildPubMedAbst(a[[i]])
}
```

To view and access various slots of these objects, including the title slot

```
absts1[[1]]
for(i in 1: length(absts1))
  print(articleTitle(absts1[[i]]))
```

The function `pm.getabst` combines a number of the above tasks. It takes as arguments gene names and a character string identifying the appropriate annotation for the experiment. PubMed IDs are obtained for the corresponding gene names and PubMed is queried with these IDs. The corresponding abstracts are downloaded and converted into R objects of class `pubMedAbst` using the `buildPubMedAbst` function as well as functions from the `XML` package. The ALL and AML expression dataset was obtained using the Affymetrix HU6800 chip, thus the annotation to be used is `"hgu68"`. Correspondence between Affymetrix names (e.g. D10495_at) and PMIDs (e.g. "11822877", "11748588", "11676480", "11466390", "8357834", "8188219", "7925449") is specified in the file `hgu68pmed` from the `annotate` package. Note that Bioconductor now provides data packages containing various annotation files for PubMed and LocusLink queries, as well as chromosomal location information (e.g. `hgu6800` and `hgu95A` packages). To obtain PubMed entries corresponding to the genes with the 5 smallest p–values

```
R> g5 <- gname[1:5]
R> absts2 <- pm.getabst(g5, "hgu68")
```

`absts2` is a list of length the number of genes, where each entry is itself a list of objects of class `pubMedAbst` for each gene. The `pm.titles` function can be used to extract abstract titles and store these in a list with length the number of genes.

```
R> pm.titles(absts2)

[[1]]
[1] "Identification of two novel human putative serine/threonine kinases, VRK1 and VRK2,

[[2]]
[1] "An important role for protein kinase C-delta in human keratinocyte migration on der
```

```
[2] "Regulation of human eosinophil NADPH oxidase activity: a central role for PKCdelta.
[3] "Physical and functional interactions between protein tyrosine phosphatase alpha, PI
[4] "Inhibition of H2 histamine receptor-mediated cation channel opening by protein kina
[5] "Molecular and biochemical characterization of a recombinant human PKC-delta family
[6] "Assignment of the protein kinase C delta polypeptide gene (PRKCD) to human chromoso
[7] "Bradykinin induces translocation of the protein kinase C isoforms alpha, epsilon, a

[[3]]
[1] "Prediction of the coding sequences of unidentified human genes. III. The coding seq

[[4]]
[1] "[The change in the plasma contents of adrenomedullin and endothelin in burn patient
[2] "Expression and function of adrenomedullin and its receptors in Conn's adenoma cells
[3] "Comparison of intravenous adrenomedullin with atrial natriuretic peptide in patient
[4] "Expression of adrenomedullin by human granulosa lutein cells and its effect on prog
[5] "Adrenomedullin."
[6] "Discovery of adrenomedullin in rat ischemic cortex and evidence for its role in exa
[7] "Genomic structure of human adrenomedullin gene."
[8] "Cloning and characterization of cDNA encoding a precursor for human adrenomedullin.

[[5]]
[1] "Human Nopp140, which interacts with RNA polymerase I: implications for rRNA gene tr
[2] "Cell-cycle-dependent alterations of a highly phosphorylated nucleolar protein p130
```

Text searching can then be done using the function `pm.abstGrep` which relies on the R function `grep`. It returns a list (of length the number of genes) of logical vectors, indicating which abstracts contain the desired word for the corresponding gene. For example, to see which abstracts contain the word protein (with lower or upper case p)

```
R> sapply(absts2, function(x) pm.abstGrep("[Pp]rotein", x))

[[1]]
[1] TRUE

[[2]]
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE

[[3]]
[1] TRUE

[[4]]
[1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE

[[5]]
[1] TRUE TRUE
```

5.3 Chromosome location

To obtain the chromosomal locations of the genes with small p–values

```
R> chroms <- read.annotation("hgu68chrom")
R> whichC <- multiget(gname, env = chroms, iffail = NA)
R> cc <- as.numeric(unlist(whichC))
```

5.4 Creation of HTML report

Finally, the function `ll.htlmpage` can be used to create an HTML report of our analysis. Given LocusLink accession numbers, it will produce an HTML file with one row per gene and a clickable entry which opens the LocusLink webpage for that gene. Other information can be displayed using the `othernames` argument as shown below.

```
R> res <- cbind(gname, cc, signif(stats, 3))
R> names(res) <- c("Gene name", "Chromosome", names(stats))
R> ll.htmlpage(LocusID, filename = "golub.html", title = "Golub et al. data, genes wit
+      othernames = res, table.head = c("LocusID", names(res)),
+      table.center = TRUE)
```

The report is stored in the file `golub.html` of your current working directory. In case you forgot the directory, type `getwd()`.

6 Cluster analysis

6.1 Distances

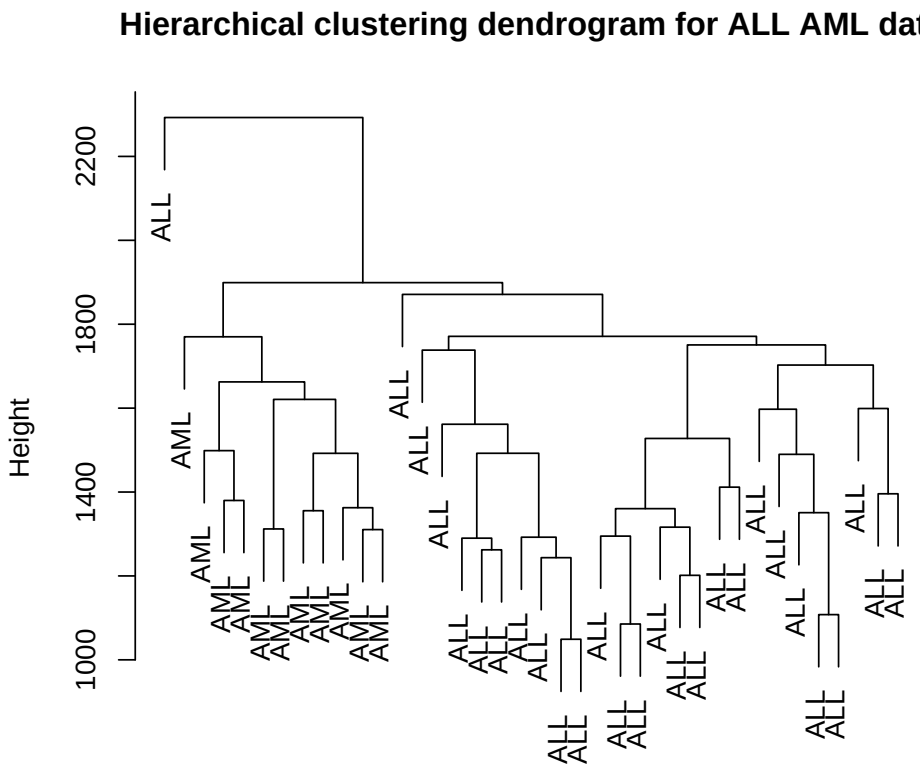
One of the most important issues that must be addressed when analyzing gene expression data is the choice of a distance or similarity measure. To compare two genes across a set of samples (or two samples across a set of genes) we must have some notion of distance between these objects and define a suitable function for measuring this distance. There are many different measures that are available. In the interest of using standard software, we will use either the Euclidean or Manhattan distances in this tutorial. However, we do note that this decision is important and should generally be given much more consideration than we have time for here.

6.2 Clustering methods

We now give a very brief overview of clustering functions in R; many more clustering methods are available in R than described here. Depending on the question of interest, one could cluster the samples to see which ones group together or we could be interested in clustering genes – again to see which group together.
Two broad categories of clustering algorithms are the agglomerative algorithms and the partitioning algorithms. Agglomerative hierarchical clustering can be performed using the `hclust` function in the `mva` package.

First, `dist` is used to compute distances. This function takes a matrix as its first argument and computes the distances between the rows of the matrix. We will use the expression data matrix from `golubTrainSub` for this purpose and if we want to cluster samples then we must take the transpose of this matrix. The matrix can be obtained from the `exprSet` object `golubTrainSub` using the accessor function `exprs`. The commands below are used to carry out hierarchical clustering using the Manhattan distance metric and to plot the corresponding dendrogram with nodes labeled according to the ALL/AML status.

```
R> dgTr <- dist(t(exprs(golubTrainSub)), method = "manhattan")
R> hcgTr <- hclust(dgTr, method = "average")
R> plot(hcgTr, labels = golubTrainSub$ALL.AML, main = "Hierarchical clustering dendrog
+      sub = "Average linkage, Manhattan distance for scaled arrays, G=3,051 genes")
```



Next, we consider partitioning methods. These include `pam`, partitioning around medoids, which is available in the `cluster` package, and `kmeans` which is available in the `mva` package. These methods require the user to specify how many clusters are wanted. The algorithms then use different strategies to apportion the samples to the clusters.

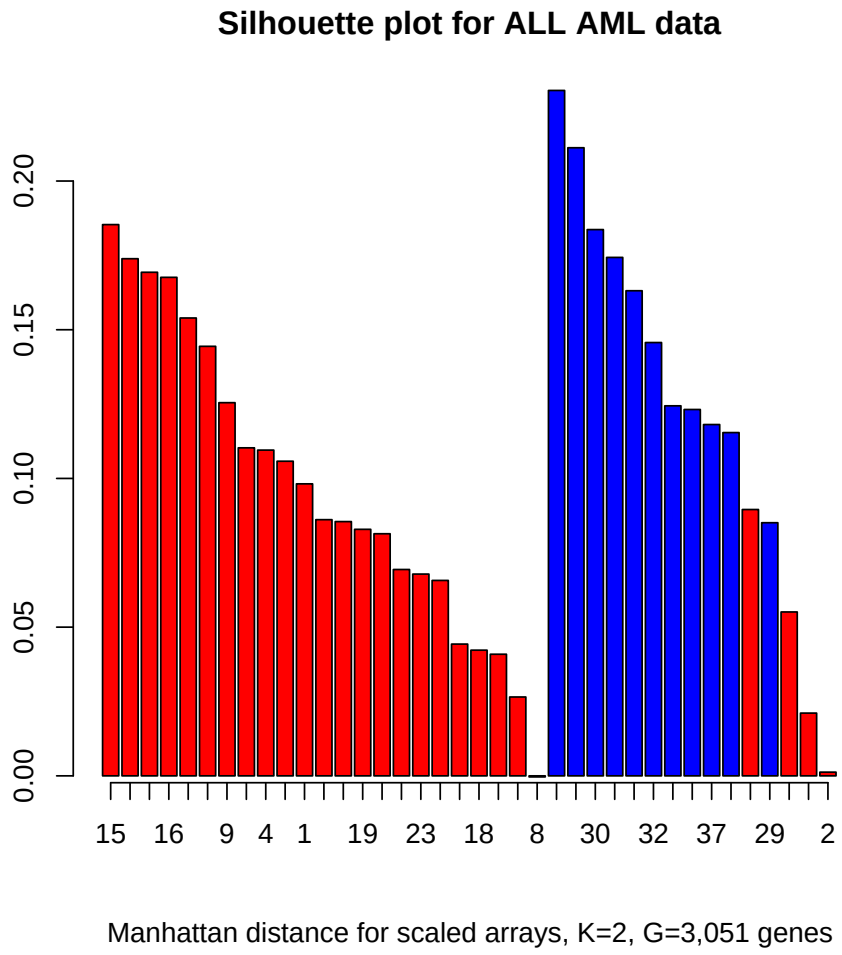
For pam silhouette plots are easily produced. For the interpretation of these read the documentation (? pam) or Kaufman and Rousseeuw (1990). In the next example (as in many other examples) we set the seed for the random number generator. This is done to ensure that the results are always the same. These algorithms have some random components that may give rise to very different results. One may want to consider the average (in some sense) of multiple runs of these algorithms.

```
R> set.seed(12345)
R> pmgTr <- pam(dgTr, k = 2, diss = TRUE)
R> kmgTr <- kmeans(t(exprs(golubTrainSub)), centers = 2)
R> table(pmgTr$clust, golubTrainSub$ALL.AML)

    ALL AML
  1  23   0
  2   4  11

R> table(kmgTr$clust, golubTrainSub$ALL.AML)

    ALL AML
  1   2  11
  2  25   0
```

The two tables allow us to compare the clustering results to the truth.

For silhouette plots

```
R> sils <- pmgTr$silinfo[[1]][, 3]
R> ord <- as.numeric(row.names(pmgTr$silinfo[[1]]))
R> barplot(sils, col = c(2, 4)[as.numeric(cl[ord])], main = "Silhouette plot for ALL A
+      sub = "Manhattan distance for scaled arrays, K=2, G=3,051 genes")
```



References

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M.L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

Leonard Kaufman and Peter Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. John Wiley & Sons, 1990. ISBN 0471878766.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S–PLUS*. Springer, 1999.