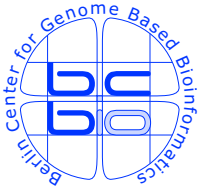



Max Planck Institute for Molecular Genetics
Computational Diagnostics Group @ Dept. Vingron
Ihnestrasse 63-73, D-14195 Berlin, Germany
http://compdiag.molgen.mpg.de/

User's Guide to the R Package *compdiagTools*

# Toolbox for Performing and Illustrating Microarray Data Analyses

**Stefanie Scheid, Jochen Jäger, Claudio Lottaz**
**email:** `first.last@molgen.mpg.de`

Technical Report
Nr. 2005/01

## Abstract

This is the vignette of the Bioconductor compliant package *compdiagTools*, a collection of valuable and handy functions for performing and illustration of gene expression data analyses.

# Contents

# Chapter 1

# Preprocessing Utilities

## Author

The functions discussed here are written by Stefan Bentink and Dennis Kostka, this description by Claudio Lottaz[1].

## 1.1 Overview

The functions described here are used in the CompDiag pipeline for processing microarray datasets acquired using Affymetrix GeneChip microarrays. We rely on the Bioconductor packages *affy*, *vsn* and *affyPLM*. Furthermore, meta-data packages for the particular chips are needed.

The preprocessing of a dataset is performed by calling the `runPreprocessing`. This function needs a path, where the processed data should be written and a file which contains full paths to all CEL-files to be included:

```
runPreprocessing{"/project/gene_expression/analyses/Yeoh02")
```

When the cel-file-list is omitted, the file "celfiles.lst" in the base path is expected to contain this list. The cel-file-list contains one entry per line holding a short name before and a complete path after the single tab allowed in each entry. E.g.:

```
...
E2A-PBX1-C8     /public_datasets/Yeoh02/rawdata/E2A-PBX1-C8.CEL.gz
E2A-PBX1-C9     /public_datasets/Yeoh02/rawdata/E2A-PBX1-C9.CEL.gz
E2A-PBX1-R1     /public_datasets/Yeoh02/rawdata/E2A-PBX1-R1.CEL.gz
Hyperdip-50-1   /public_datasets/Yeoh02/rawdata/Hyperdip-50-#1.CEL.gz
Hyperdip-50-10  /public_datasets/Yeoh02/rawdata/Hyperdip-50-#10.CEL.gz
```

[1]Contact: `claudio.lottaz@molgen.mpg.de`

```
Hyperdip-50-11  /public_datasets/Yeoh02/rawdata/Hyperdip-50-#11.CEL.gz
Hyperdip-50-12  /public_datasets/Yeoh02/rawdata/Hyperdip-50-#12.CEL.gz
Hyperdip-50-13  /public_datasets/Yeoh02/rawdata/Hyperdip-50-#13.CEL.gz
Hyperdip-50-14  /public_datasets/Yeoh02/rawdata/Hyperdip-50-#14.CEL.gz
Hyperdip-50-2   /public_datasets/Yeoh02/rawdata/Hyperdip-50-#2.CEL.gz
...
```

Here is a rough description of our preprocessing protocol:

**Background correction:** No explicit background correction is performed. The theoretical model used for normalization in *vsn* incorporates a model for background correction. Moreover, we did not observe useful effects due to the explicit background correction step, we have used in former versions of our preprocessing.

**Normalization:** Normalization is performed on the probe level using the variance stabilization and calibration method implemented in the Bioconductor package *vsn*. This method uses a asinh transformation (instead of the log) which renders the variance of probe intensities approximately independent of their expected expression levels. For each chip an offset and a scaling factor are estimated, assuming that a fair fraction of probes are not differentially expressed across the study. Given the computational complexity of this method, parameters are estimated on a random subset of probes and then used to transform the entire arrays.

**Probe set summary:** To compute expression levels for probe sets based on the corresponding probes, we apply the median polish method on the arsinh normalized data. For each probe set a robust additive model is fitted across the arrays, possibly taking into account the different sensitivity of the probe cells via a probe effect. For this step we use the *affyPLM* package, because the methods in this package provide residuals for each probe reflecting the quality of the model fit.

## 1.2 Directory structure

The data generated during preprocessing is saved in a predefined directory structure of subdirectories ind the base directory provided to `runPreprocessing`:

- `./logs`: a log file containing progress messages is written here. Existing files are not overwritten.
- `./normalized`: Normalized data as well as images for quality control are saved here.

- `./simpleaffy`: Standard MAS5-like analysis is performed only for quality control. The results are saved here.
- `./summarized`: Results from the summary step including images for quality control are saved here.

The final result is written into a Bioconductor expression set and saved into the base directory if not specified otherwise.

## 1.3 Graphics utilities

During preprocessing, we routinely generate plot and images useful for quality control. The corresponding function can also be used directly. By default they generate bitmap output in PNG format, but upon request they can also generate PDF output. PDF output is much better in quality but needs more memory. Memory is not an issue for particular documents but for routine generation of several images per chip in large datasets.

### 1.3.1 `boxplt.cpd` and `boxplt.plmset.cpd`- box plots on expression levels and standard errors

When microarray experiments are consistently measured, we expect similar distributions of expression levels after normalization. Box plots give a nice overview on distributions by showing particular quantiles over many distributions in one plot. Moreover, outliers are visible but of less interest for quality control in the context of microarray studies. In a well performed and normalized dataset the boxes of the different chips are neatly aligned (Figure 1.1, left hand side). Furthermore, we can consider the distributions of residuals for each microarray. We want to see very narrow distributions around zero, meaning that the model fit used for summary did not need to correct the actual measurements excessively (Figure 1.1, right hand side).

### 1.3.2 `wimage.cpd` - 2 dimensional residual plots

This plot allows for the chip-wise illustration of the correction by the model fit for summary. The same residuals used in `boxplt.plmset.cpd` are drawn as color codes against their position on the chip (Figure 1.2). This reveals locally consistent corrections by the linear model. This could happen, for instance, when dirt causes bad measurements in a whole region of a chip, or when some technical problem leads to a gradiant over the chip.

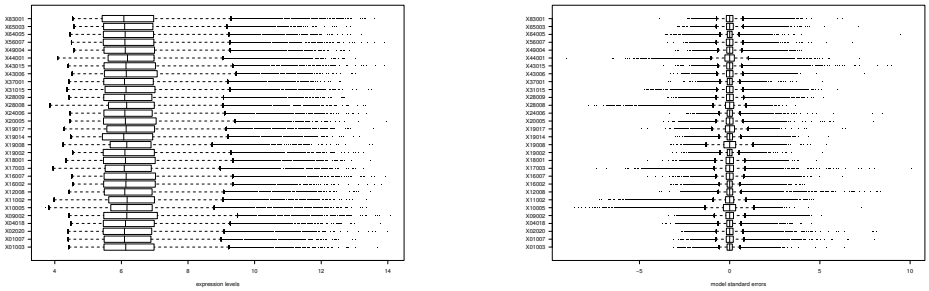


**Figure 1.1:** Boxplots on expression levels (left) and residuals (right).

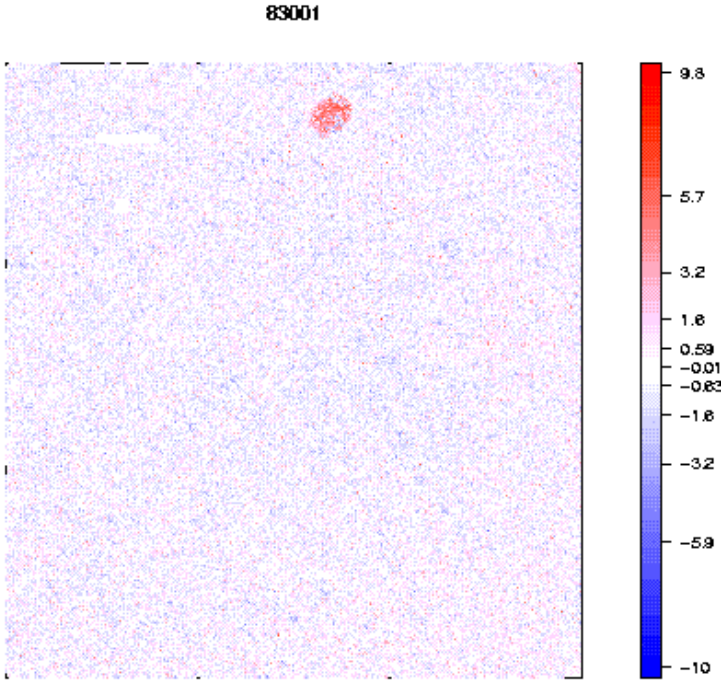



**Figure 1.2:** Chip-wise residual-location-image.

### 1.3.3 `MvsA.cpd` - rank residual plots

The rank-residual-plots generated by `MvsA.cpd` are generated as follows: expression levels from a reference chip (one of the function arguments) are subtracted from the chip to be investigated (another function argument) to form the residuals. These residuals are ranked according to the expression level measured in the investigated chip. Such plots are useful to check whether the variance stabilization procedure has worked.

The scatter plot as shown in Figure 1.3 shows the variance of genes depending on their epxression level as a dark band around the x axis. The band should be the same width from left to right. As a further help to judge this property, we draw in blue three times median absolute deviance for a running window across the expression ranks. These should ideally form a horizontal line. The red line around zero should also be horizontal (at zero). This is a loess fit to the drawn dots.

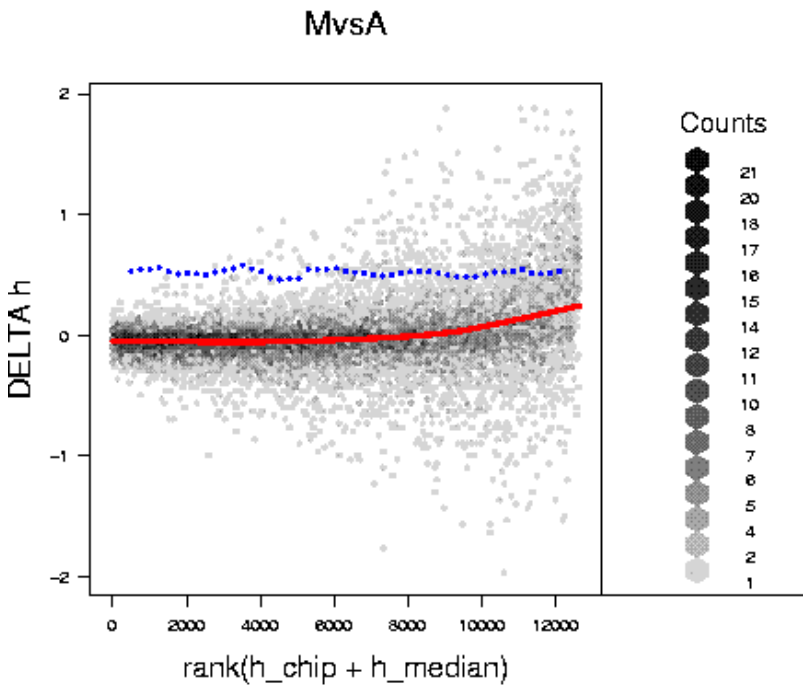



**Figure 1.3:** Chip-wise rank-residual-plot.

## 1.4 Incremental preprocessing

There are two reasons for incremental normalization and preprocessing. On one hand, very large datasets can only be preprocessed using large amounts of memory. For instance, datasets measured with modern whole-genome microarrays can usually not be processed on 32-bit linux machines when they have more than 100 Chips, because the available maximal memory of 2GB is not sufficient. On the other hand, when diagnostic signatures are to be published, they have to be stable for new patients that are diagnosed after the establishment of the signature. For these two reasons, we include the `addonPreprocessing` method to the *compdiagTools* package.

The idea is to collect a core set of samples first and use these for establishing a scale. This scale consists of probe-effects used in median polishing and vsn parameters. Both sets of parameters are stored by the *runPreprocessing* method into the preprocessing description slot of expression sets, the corresponding list element is called `val` and the `addonPreprocessing` method relies on the package *docval* to compute the expression levels for external chips.

In addition to the straight computation of expression levels for external chips, the here described method can also generate the corresponding images for quality control, as they are suggested by `runPreprocessing`. I.e., the rank-residual plots and the residual-location-plots. Not all of the whole-project quality control plots are drawn, because they would need the complete reading of all raw data, which is not feasible for large datasets. Actually, only the boxplot for summarized expression levels is updated for the external chips in order to compare the distributions of the expression levels of external chips with those of the core.

# Chapter 2

# `optimTree` - Rearranging dendrograms

## Author

The `optimTree` bundle is written by Stefanie Scheid[1].

## 2.1 Introduction

The two-dimensional hierarchical clusterplot is one of the most favourite tools to visualize expression data. The left plot of Figure 2.1 shows such a plot for the lymphoma data set in [Alizadeh et al.(2000)Alizadeh, Eisen, Davis, Ma, Lossos, Rosenwald, Boldrick, Sa
The data set stored in `data(lymphoma)` in package *vsn* comprises eight cDNA samples, the values are log ratios of tissue to reference expression. For our purposes, we filtered for the 100 genes with highest variance. The center of the clusterplot is filled with expression data coded in colours. Here, bright yellow corresponds to large positive log ratios (up-regulation) and bright blue to large negative log ratios (down-regulation).

Rows and columns of the expression matrix are reordered independently of each other. The dendrograms on top and on the left side of Figure 2.1 depict the resulting ordering schemes. For each of the two dimensions, we conduct hierarchical clustering with average linkage using Euclidean distances between samples or genes respectively. For computation, we apply the R function `hclust` twice and pass the resulting `hclust` objects on to the plotting function `clusterplot`.

Clusterplots are a comfortable tool for visualizing a data set or exploring a set of candidate genes. However, their interpretation is not trouble-free. A closer inspec-

[1]Contact: `stefanie.scheid@molgen.mpg.de`

tion of the left plot in Figure 2.1 reveals a typical problem: We immediately desire to rearrange the columns because the two left-most and the two right-most samples appear to belong together. They are all coloured mainly in blue and, what is more relevant, they actually belong to the same disease entity (CLL) while the samples in between belong to another entity (DLCL). Regarding the dendrogram on top, we could simply swap it at its third node from top. Thus, all CLL columns are gathered on the left hand side which is similar to the right plot in Figure 2.1. This swapping is not illegal, it does not change the information collected in the plot in any way. To the human eye however, the new ordering "makes more sense". Of course, we can perform many swappings while keeping a valid dendrogram. With $n$ samples, the dendrogram has $n-1$ nodes and we are allowed to flip each node, thus leading to $2^{n-1}$ possible dendrogram permutations. One half of these just mirrors the other half, which reduces the number of unique permutations to $2^{n-2}$ for our purposes. For the lymphoma data set, $n = 8$ leads to 64 possible dendrograms. The following chapter introduces the R function `optimTree` which is designed to find the optimal dendrogram permutation with respect to a certain measure.

## 2.2 Implementation

The search for an optimal dendrogram rearrangement works as follows: We enumerate all possible permutations. For each resulting dendrogram, we read the samples from left to right and sum over distances between neighboring samples, disregarding any feature of the underlying dendrogram. The optimal dendrogram is the one with the smallest sum of distances, that is the dendrogram where similar (close) samples are arranged in proximity. Consider the toy example of four leafs and the distance matrix shown in Table 2.1. Based on the distance matrix, hierarchical clustering using `hclust` is performed.

Four samples lead to $2^2 = 4$ possible dendrograms. These are shown in Figure 2.2. The number on top of each plot is the sum of pairwise distances. For the first plot, this is the sum of distances between leafs 4 and 2, 2 and 1, 1 and 3, resulting in $9 + 1 + 5 = 15$. The optimal dendrogram is the second one with a distance sum of 6.

The `optimTree` bundle contains two main functions that affect dendrograms: `optimTree` and `optimTree.dendrogram`. Function `optimTree.plot` is a convenient wrapper to plot gene expression data, see Section 2.3. Function `optimTree` works on `hclust` objects but needs the underlying distance matrix as additional input. It returns an `hclust` object with minimal sum of distances. In case the minimum is not unique, `optimTree` returns the first minimum. The output object has an additional slot `sumDist` where the sum of pairwise distances is stored. The optional argument

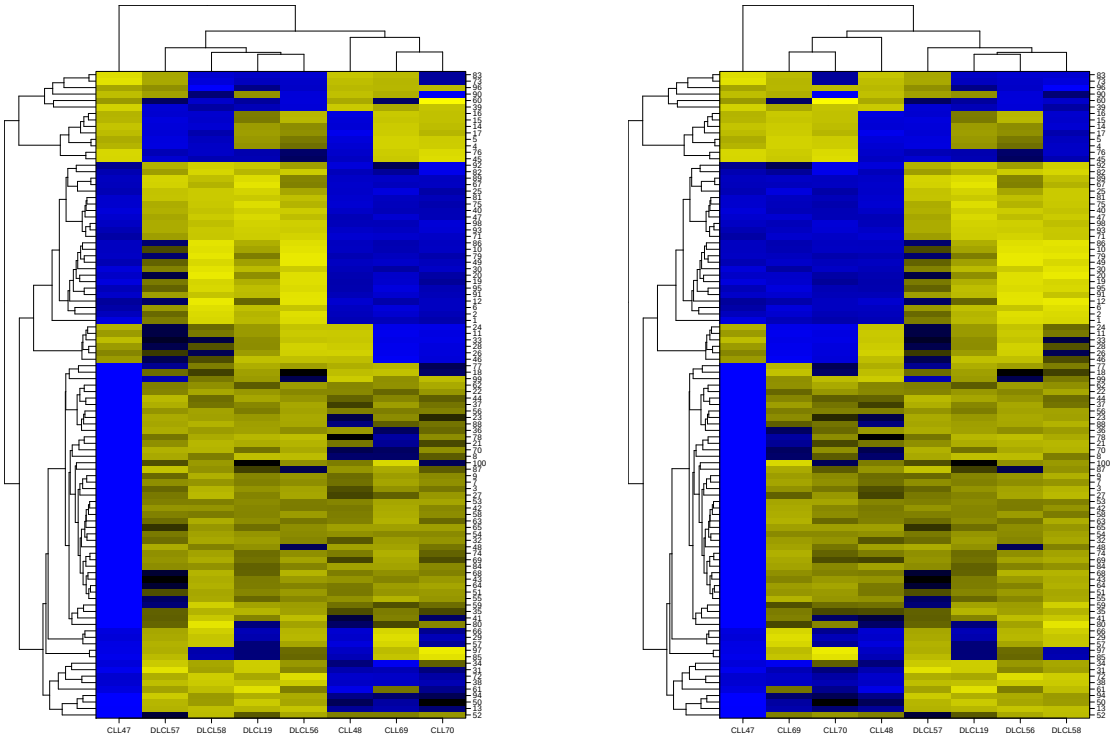



**Figure 2.1:** Two-dimensional hierarchical clusterplots of lymphoma data. Left plot: Default ordering of samples using `hclust`. Right plot: Optimal ordering of samples using `optimTree`.

| | 1 | 2 | 3 |
|---|---|---|---|
| 2 | 1 | | |
| 3 | 5 | 2 | |
| 4 | 3 | 9 | 7 |

**Table 2.1:** Artificial distance matrix with four samples.

`all=TRUE` in `optimTree` invokes the output of the list of all possible dendrograms. The default is set to `all=FALSE` to output only the optimal tree.

Continuing the lymphoma example above, Figure 2.3 shows the first four dendrograms with smallest sums of distances. The underlying matrix of Euclidean distances between samples is given in Table 2.2. Function `optimTree` returns the first dendrogram in Figure 2.3 which cannot simply be passed on to `clusterplot`. Within `clusterplot`, function `as.dendrogram` is called which transforms the `hclust` into a `dendrogram` object but does not keep the new ordering. Here, function `optimTree.dendrogram` must be applied. The function uses `reorder` and returns a `dendrogram` object with preserved optimal ordering. Figure 2.4 shows results of successive application of `optimTree` and `optimTree.dendrogram` to the lymphoma `hclust` object. For easily plotting gene expression data without `clusterplot` see Section 2.3.

| | CLL47 | CLL48 | CLL69 | CLL70 | DLCL19 | DLCL56 | DLCL57 |
|---|---|---|---|---|---|---|---|
| CLL48 | 30.69 | | | | | | |
| CLL69 | 32.75 | 18.74 | | | | | |
| CLL70 | 31.74 | 17.32 | 10.3 | | | | |
| DLCL19 | 35.98 | 19.76 | 21.64 | 19.79 | | | |
| DLCL56 | 38.08 | 22.53 | 23.78 | 22.43 | 9.09 | | |
| DLCL57 | 35.42 | 16.46 | 20.26 | 19.43 | 11.09 | 14.36 | |
| DLCL58 | 40.08 | 22.17 | 24.51 | 23.77 | 11.01 | 9.19 | 12.16 |

**Table 2.2:** Euclidean distances of lymphoma samples.

## 2.3 Plotting expression data

The plotting function `optimTree.plot` is a convenient wrapper that produced both expression plots in Figure 2.1. It includes source code taken from package *clusterplot* by Mark Wilkinson.

The input is either an expression set of class *exprSet* or a matrix containing expression values in the common ordering, that is genes in rows and sam-

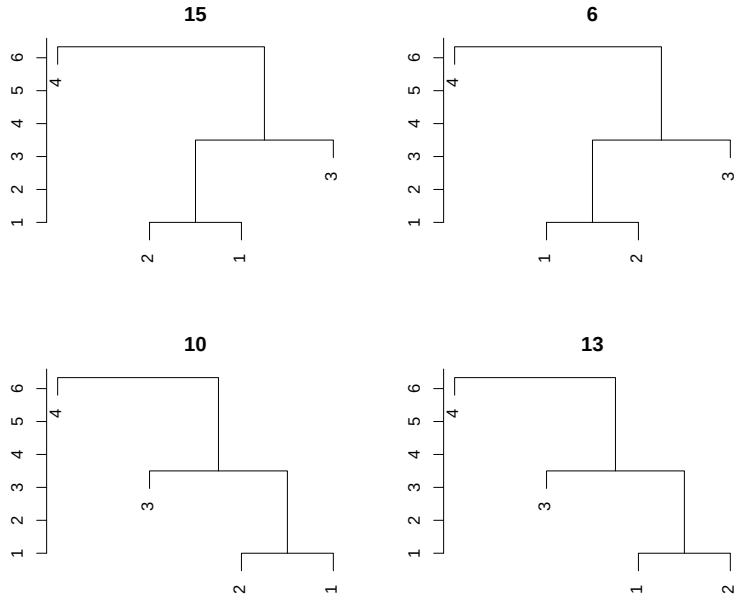



**Figure 2.2:** All valid dendrograms resulting from the example distance matrix. Values on top are sums of distances between neighboring leafs.

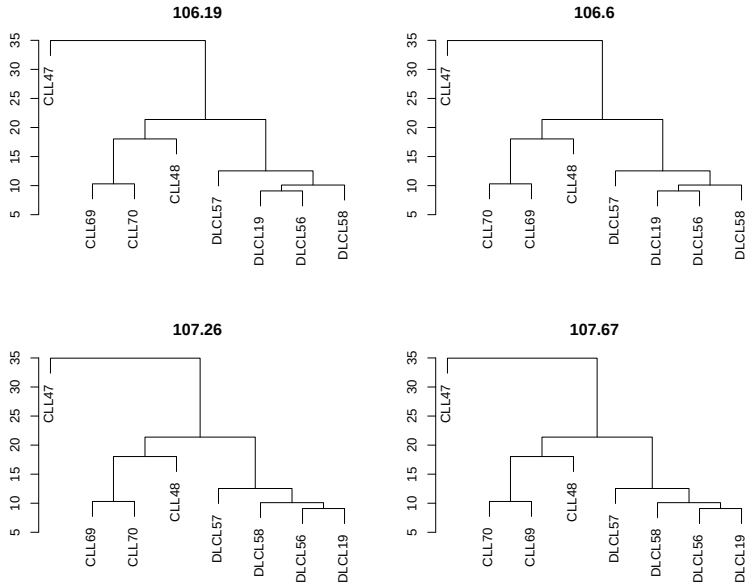



**Figure 2.3:** Four dendrograms with smallest sums of distances for the lymphoma data.

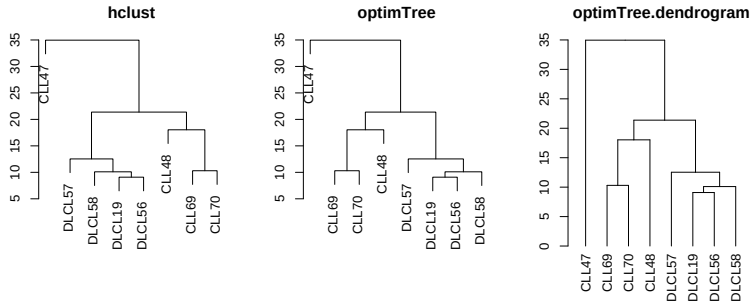



**Figure 2.4:** Dendrograms of lymphoma data with distances stored in a `dist` object `x`. Left plot: Result of `y <- hclust(x)`. Center plot: Result of `z <- optimTree(y,x)`. Right plot: Result of `optimTree.dendrogram(z)`.

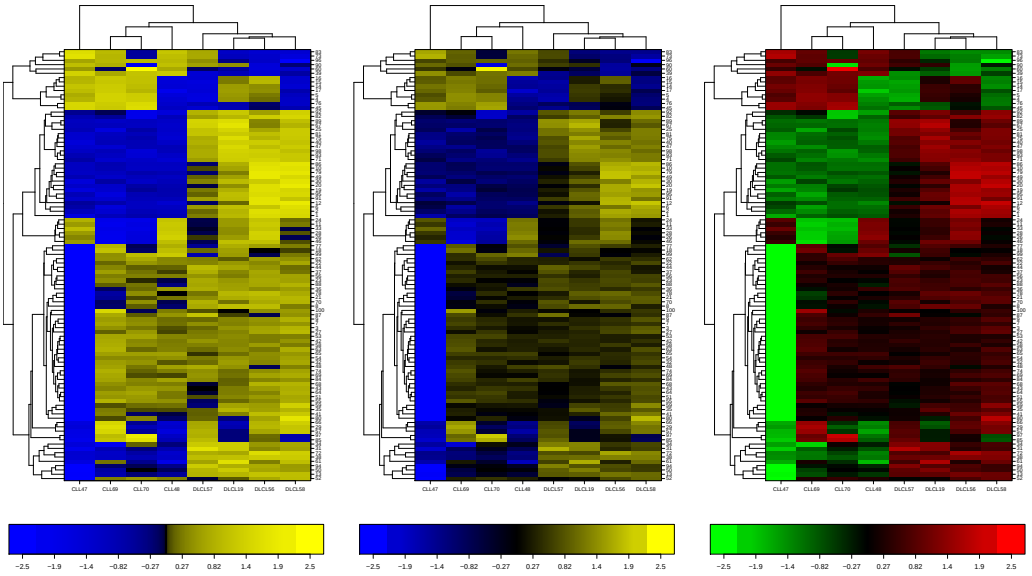


**Figure 2.5:** Three available color schemes for `optimTree.plot`. Left plot: `col="color1"`. Center plot: `col="color2"`. Right plot: `col="color3"`.

ples in columns. Currently, the function uses Euclidean distances both between genes and between samples and performs hierarchical clustering with average linkage. Argument `optimize=TRUE` invokes the search for the optimal dendrogram. If `optimize=FALSE`, the default `hclust` output will be processed. These two options lead to Figure 2.1. Please note, that the optimization is limited to sample size 15. If necessary, change the default argument `max.samp=15`.

Additional arguments affect the plotting style. With `row.scale=TRUE`, the expression matrix is scaled gene-wise before plotting (but after clustering). With `zlim`, we define the color range. The default is 2. Hence, all (scaled) expression values exceeding $\pm 2$ are encoded in the brightest color. If `row.scale=FALSE`, a higher `zlim` value may be reasonable. Argument `col` can be set to `"color1"` (default), `"color2"` or `"color3"`. Finally, argument `barplot=TRUE` produces the encoding color bar using package *GLAD*. See Figure 2.5 for the resulting images.

## 2.4 Limitations

Function `optimTree` enumerates all valid dendrograms that emerge from the given one. If the number of samples is large, the computation is time-consuming if not impossible. With 20 samples, 262144 permutations are possible. Computations with

more than 20 samples might not be feasible any more. For the plotting default, see Section 2.3.

For larger data sets or optimal rearrangement of genes, an algorithmical approach is needed. The simpliest search algorithm starts with the given dendrogram and enumerates all neighbors (dendrograms with one flipped node). If a neighbor has lower sum of distances, this one is kept as the next starting point. The search results in a dendrogram with minimal sum of distances among its neighbors. However, it is not necessarily the optimal choice. The implementation of heuristic search algorithms for solving the problem of dendrogram rearrangement is part of future research.

# Chapter 3

# Evaluating Gene Lists

## Author

The functions discussed here written by Martin Held, Matthias Maneck, Jochen Jäger and Claudio Lottaz, this description by Claudio Lottaz[1].

## 3.1 `geneLister` - generating gene lists in HTML

Many analysis methods for microarray data generate lists of genes: supervised classification compute signatures, unsupervised class finding techniques report lists of supporting genes and differential gene expression detection directly aims for lists of induced genes. Since thus generated gene lists are often rather long, they are of little use without supporting annotations.

The `geneLister` exploits Bioconductor meta-data packages for the microarrays used in a particular analysis, to generate annotated lists of genes. It can gake lists of probe-sets as well as objects from *twilight*[2] analyses as input. The provided list holds gene symbols and names as well as corresponding identifiers from various databases including GenBank, UniGene, Gene Ontology and KEGG. The database identifiers are generated as direct links to the corresponding Internet sites.

```
 \small
> library(hgu95av2)
> ids <- ls(hgu95av2ACCNUM)[1:20]
> geneLister(ids, "hgu95av2", HTML=FALSE)
```

[1] Contact: `claudio.lottaz@molgen.mpg.de`

[2] differential gene expression and local FDR computation package from Bioconductor

By default the method generates HTML code and returns it as a character string. This HTML code is well suited to be part of an HTML page. The function also generates anchors for each probe-set, so that links directly to a gene of interest can be generated.

## 3.2 geneImager - images with expression data

The function `geneImager` generates an image for further illustration of gene expression patterns present in a particular gene set of interest. Similar as the suggestion by Eisen and others [2], an image with one row per gene and one column per sample is drawn. Hierarchical clustering and dendrograms can be added optionally, just as scaling and annotations fpr genes/samples. The following image is generated as follows:

```
library(ALL)
library(twilight)
cls <- ifelse(substr(as.character(pData(ALL)[," BT" ], 1, 1)==" B" , 1, 2)
tw <- twilight.pval(ALL, cls)
gl <- rownames(tw$result)[1:50]
geneImager(exprs(ALL)[gl,])
```

The `geneImager` can return HTML code with a clickable map. This map defines clickable regions on the annotations of th genes. The links generated lead to anchors produced by the `geneLister`. In order to use this clickable map, the image must not be scaled.

## 3.3 gokeggLister - overrepresentation analysis

A common method to explore gene lists on a more global view is to determine overrepresented functional groups of genes. Probe-sets are attributed to one or more biological processes, molecular function and cellular location with respect to the Gene Ontology [Ashburner et al.(2000)Ashburner, Ball, Blake, Botstein, Butler, Cherry, Davis, Dolinski e
as well as pathways in the Kyoto Encyclopedia of Genes and Genomes [Kanehisa(1996)].
For each of these terms we can compute a probability that more non randomly many genes with that given annotation are present in a gene list of interest. This probability is computed with respect to the hypergeometrical distribution.

For the list of genes `gl`, generate in the in the `geneImager` example, overrepresentation analysis as described in the paragraph above is computed as easy as this:

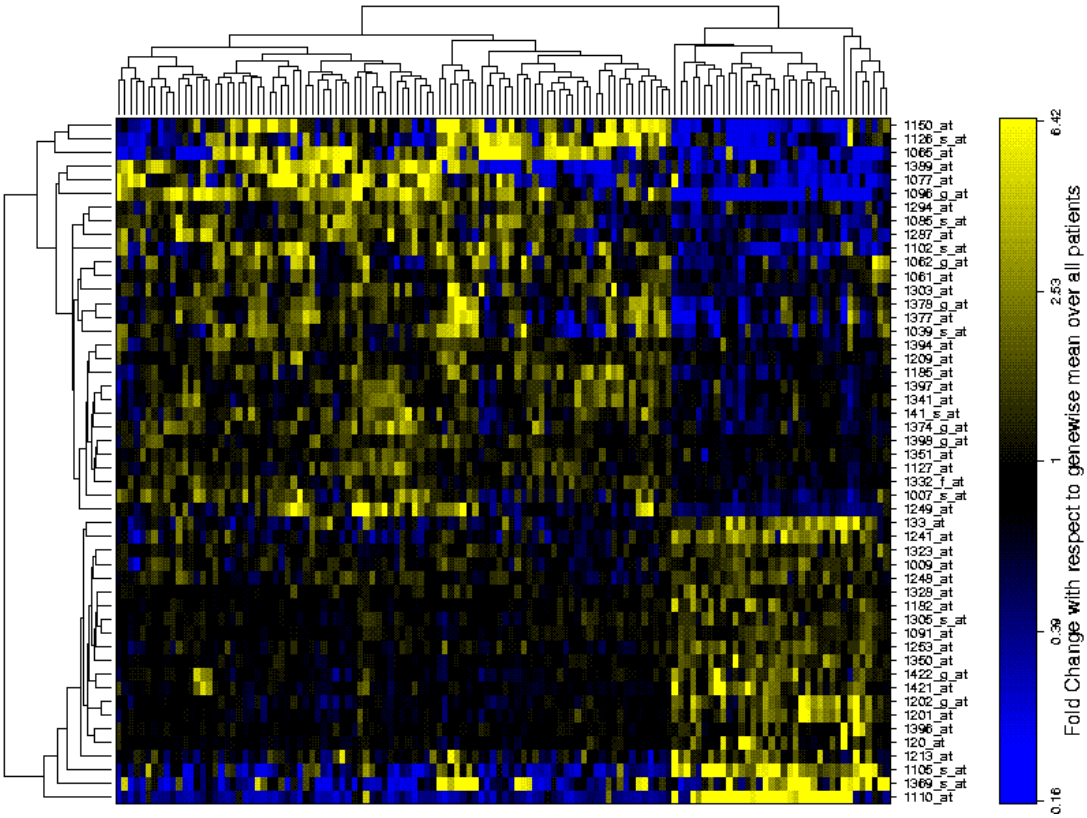



**Figure 3.1:** Gene expression patterns illustrated as color codes in an image. Illustrated on leukemia data published in [1]

```
gokeggLister(gl)
```

This command also returns a peace of HTML code to be included into a HTML page. The HTML code encodes a table with all detected terms at a p-value below 0.1.

As an additional feature `gokeggLister` generates induced graphs for detected GO terms (not for KEGG) as shown in Figure 3.2. From nodes detected as relevant, all parants are searched up to the root. Such trees give an expression, whether overrepresented GO nodes are closely related or not. The HTML code

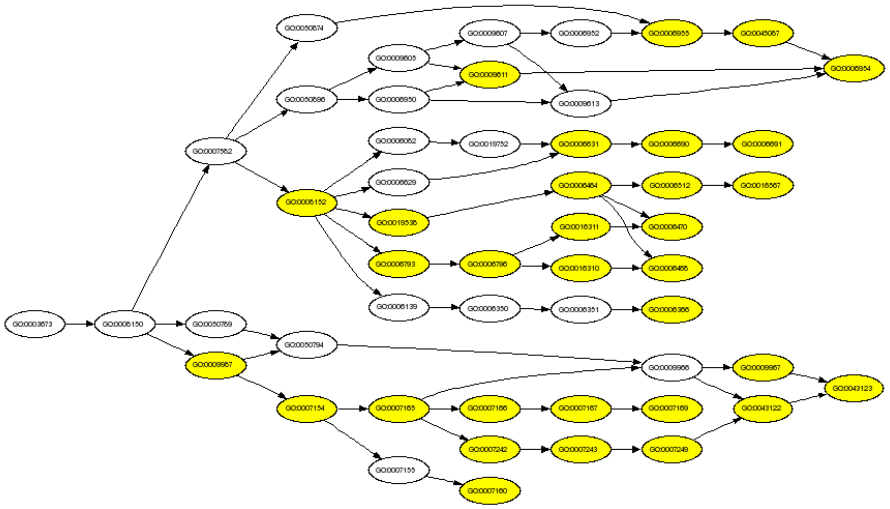


**Figure 3.2:** Induced biological process GO graph for upregulated genes in `gl`

returned by `gokeggLister` includes a clickable map on this graph. In order to use the graph feature just described, the graph visualization package `Rgraphviz` [Gansner & North, 2000] be installed on your system.

There is no correction for multiple testing although in most categories more than 1000 hypotheses are tested. Therefore, only very low p-values should be interpreted. Correction for multiple testing is non-trivial since almost all of the hypotheses are wildly dependent.

# Chapter 4

# Unsupervised Methods

## Author

Claudio Lottaz[1].

## 4.1 Consensus Clustering

The function `consensusCluster` implements the method suggested in [Monti(2003)]. We have also implemented an image method for the computed consensus matrix.

This function performs standard k-means clustering on the data matrix provided. It returns the result as is from the k-means method. In addition a consensus matrix is computed from splits of subsamplings of the data. In this matrix each row corresponds to a observation and so does each column. Rows and columns are ordered in same way such that observations from the same overall cluster are adjacent. In each position of the matrix, the ratio is stored, how often the two corresponding samples are attributed to the same cluster by k-means.

For stable clusterings, we expect that the consensus matrix just holds values close to zero and close to one. square blocks of ones are expected along the diagonal of the matrix. As summary statistics, a value per cluster and a value per observation and cluster is computed. To measure the stability of a cluster, the average consensus values from the consensus matrix are taken for all pairs of observations where both partners belong to the cluster. Moreover, for observation o and cluster C, the average consensus values is computed for all other members of cluster C.

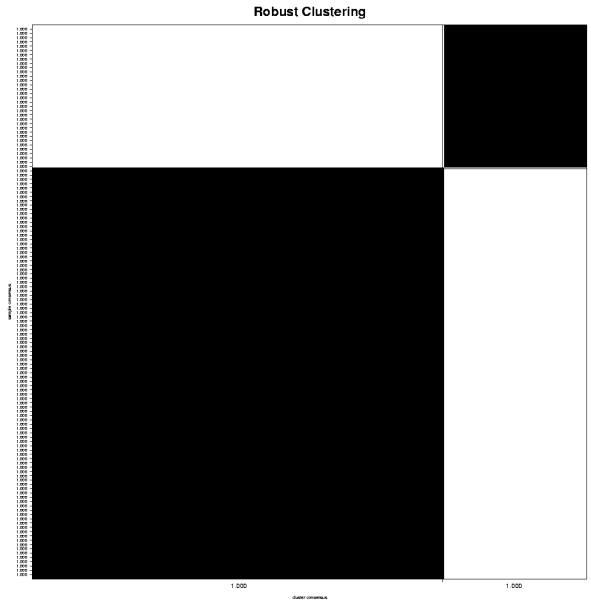
Here we compute an example useing the ALL data by Chiaretti et al. [1]. In this dataset we expect a very clear split between T-cell and B-cell leukemias. The following computes the corresponding consensus matrix:

[1]Contact: `claudio.lottaz@molgen.mpg.de`

```
> library(ALL)
> data(ALL)
> vars <- apply(exprs(ALL), 1, var)
> e <- exprs(ALL)[rank(vars) > 12525, ]
> consClust <- consensusCluster(e, nclass = 2, nsamplings = 500)

.................................................
```

Thereby we choose only the 100 highest variance genes for clustering. With the following command the corresponding consensus matrix is plotted:

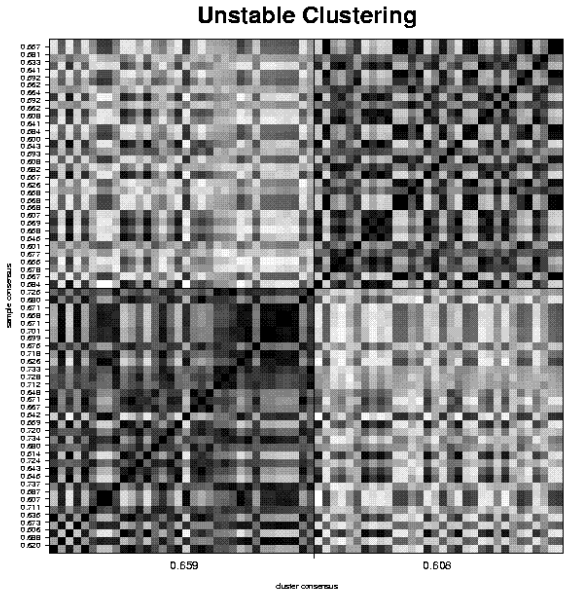
```
> image(consClust, main = "Robust Clustering", col = grey(seq(1,
+     0, -0.01)), outfile = "fig_consClust_robust.png")

null device
          1
```




When we restrict the dataset to B-cells only, it is much more difficult to detect a clear split and thus the corresponding plot is blurred out as well:

```
> samples <- 1:66
> vars <- apply(exprs(ALL)[, samples], 1, var)
> e <- exprs(ALL)[rank(vars) > 12525, samples]
> consClust <- consensusCluster(e, nclass = 2, nsamplings = 500)

.................................................
```

```
> image(consClust, main = "Unstable Clustering", col = grey(seq(1,
+     0, -0.01)), outfile = "fig_consClust_shaky.png")

null device
          1
```




Monti et al. have evaluated this method on a large lymphoma dataset in 2004 [Monti(2004)].

# Chapter 5

# Odds and Ends

## Author

Various authors have contributed the functions discussed here, the maintainer is Claudio Lottaz[1].

## 5.1 getPubmedEntry - getting citation information

This function takes a single PMID as an argument, queries Pubmed through the Internet (http://www.ncbi.nlm.nih.gov/entrez/query.fcgi) and returns the retrieved data including the abstract in a list. The names of the list correspond to the medline abbreviations.

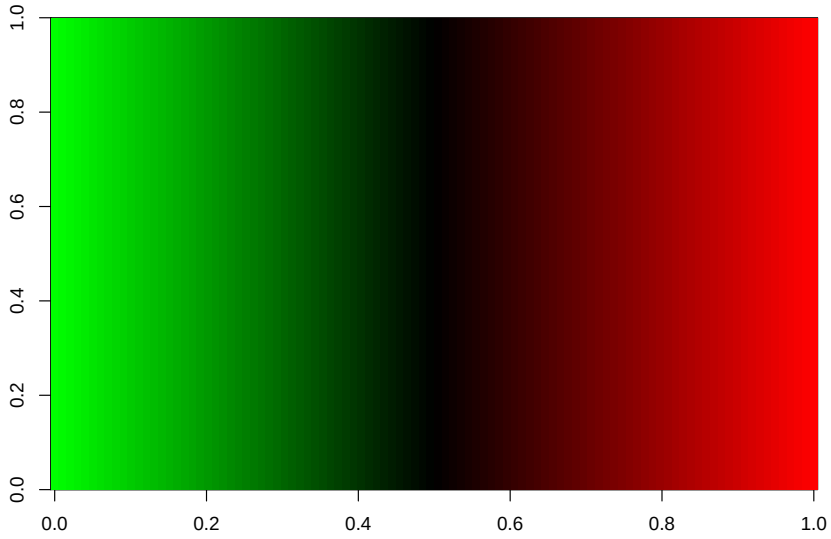
```
> x <- getPubmedEntry(11674852)
> names(x)

 [1] "PMID" "OWN"  "STAT" "DA"   "DCOM" "LR"   "PUBM" "IS"   "VI"
[10] "IP"   "DP"   "TI"   "PG"   "AB"   "AD"   "FAU"  "AU"   "LA"
[19] "PT"   "PL"   "TA"   "JT"   "JID"  "SB"   "MH"   "EDAT" "MHDA"
[28] "AID"  "PST"  "SO"

> x$TI

[1] "Resampling method for unsupervised estimation of cluster validity."
```

[1] Contact: claudio.lottaz@molgen.mpg.de

## 5.2 `greenred.colors` - colors for green-red cluster plots

This function uses the `hsv` to generate a gradiant of colors adequate for red-green cluster plots. The call allows to specify the number of shades to be generated. The folowing image illustrates the colors generated.

```
> image(matrix(1:100, ncol = 1), col = greenred.colors())
```




## 5.3 `factorplot` - Visualize factor like data

This function takes a matrix or dataframe and regards the content as factor entries. It then visualizes this matrix so that each factor level gets a unique color and a legend is plotted to the right hand side.

```
> library(golubEsets)
> factorplot(pData(golubTrain)[, c(2, 4, 5, 7, 8, 9, 11)])
```

# Chapter 6

# Bibliography

[Alizadeh et al.(2000)Alizadeh, Eisen, Davis, Ma, Lossos, Rosenwald, Boldrick, Sabet et al.] Alizadeh, A. A., M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, **2000**. *403*(6769), 503–11.

[Ashburner et al.(2000)Ashburner, Ball, Blake, Botstein, Butler, Cherry, Davis, Dolinski et al.] Ashburner, M., C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, et al. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nat Genet*, **2000**. *25*(1), 25–29.

[1] S Chiaretti, X Li, R Gentleman et al. Gene expression profile of adult T-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, **2004**, *103*(7).

[2] Eisen, M. B., P. T. Spellman, P. O. Brown, and D. Botstein, Cluster analysis and display of genome-wide expression patterns. *Proc Natl Acad Sci U S A*, **1998**. *95*(25), 14863–14868.

[Gansner & North, 2000] Gansner, E. R. & North, S. C. (2000) An open graph visualization system and its applications to software engineering. *Software Practice and Experience,* **30** (11), 1203–1233.

[Kanehisa(1996)] Kanehisa, M. Toward pathway engineering: a new database of genetic and molecular pathways. *Sci & Tech Japan*, **1996**. *59*, 34–8.

[Monti(2003)] S. Monti, P. Tamayo, J. P. Mesirov, and T. R. Golub. Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data. *Machine Learning*, *52*:1-2, **2003**, pp. 91-118.

[Monti(2004)] S. Monti, K. J. Savage, J. L. Kutok, F. Feuerhake, P. Kurtin, M. Mihm, B. Wu, L. Pasqualucci, D. Neuberg, R. C. Aguiar, P. Dal Cin, C. Ladd, G. S. Pinkus, G. Salles, N. L. Harris, R. Dalla-Favera, M. Habermann, J. C. Aster, T. R. Golub, and M. A. Shipp. Molecular profiling of diffuse large B-cell lymphoma identifies robust subtypes including one characterized by host inflammatory response, *Blood*, Nov. **2004**.