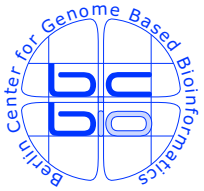
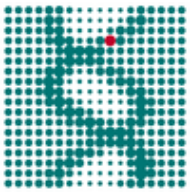


Max Planck Institute for Molecular Genetics
Computational Diagnostics Group @ Dept. Vingron
Ihnestrasse 63-73, D-14195 Berlin, Germany
http://compdiag.molgen.mpg.de/



Similarities of Ordered Gene Lists

User's Guide to the Bioconductor Package *OrderedList*

Stefanie Scheid, Claudio Lottaz, Xinan Yang, and Rainer Spang

email: Claudio.Lottaz@klinik.uni-regensburg.de

Technical Report
Nr. 2006/01

Abstract

This is the vignette of the Bioconductor compliant package *OrderedList*. We describe the methods and functions to explore the similarity between two lists of ordered genes.

Contents

1 Introduction **2**

2 Methods **3**

2.1 Similarity score 3

2.2 Tuning α 5

2.3 Comparing lists *without* expression data 5

3 Comparing Two Expression Studies **7**

3.1 `prepareData`: Combining two studies into one expression set 7

3.2 `OrderedList`: Detecting similarities of two expression studies 10

4 Comparing Two Ordered Lists **15**

4.1 `compareLists`: Detecting similarities of two ordered gene lists 15

5 Bibliography **20**

Chapter 1

Introduction

The methods of package *OrderedList* provide a *comparison of comparisons*. Say, we compare two gene expression studies. Both are comparisons of two states. Preferably, one state relates to a good outcome or prognosis and the other one relates to a bad outcome. For each study separately, we might conduct a two-sample test per gene to discover differentially expressed genes. Although each single study might not necessarily reveal significant changes, we observe considerable overlap in the top-ranking genes. Hence, we wish to compare the results of the two comparisons.

We assign a *similarity score* to a comparison of two ranked (ordered) gene lists. The similarity score is based on the number of overlapping genes in the top ranks. For each rank, the size of overlap is computed. The final score is in principle a weighted sum of these values, with more weight put on the top ranks. In the following chapter, we briefly review the methods introduced in Yang et al. (2006) [5].

Acknowledgements

This research has been supported by BMBF grants 031U109/209 and 01GR0455 of the German Federal Ministry of Education and Research. In addition X.Y. was supported by a DAAD-Fellowship.

Chapter 2

Methods

2.1 Similarity score

Data sets. We start with the analysis of two gene expression studies A and B. We assume that the two studies were either measured on the same platform or that the two sets of probes can be mapped onto each other such that the ith probe of study A corresponds to the ith probe of study B. Both studies comprise the same number of probes.

In each study, the samples divide into at least two distinct classes and we have to choose which two classes are to be compared. Within each study, a gene-wise test on the difference of class means is conducted. Appropriate tests are for example the common t-test or just the log ratio test, that is difference of means. In any case, a large positive test score corresponds to up-regulation and a large negative value to down-regulation. The genes within each study are sorted according to their test scores. Top ranks correspond to highly up-regulated genes and bottom ranks to highly down-regulated genes. These two rankings are the first stage of our analysis: the ordered gene lists G_A and G_B.

Computing the overlap. For each rank n, $n = 1, \ldots, \#\text{genes}$, we count the number of genes that appear in both ordered lists up to position n. Table 2.1 provides an artificial example for the top 10 ranks. The values $O_n(G_A, G_B)$ denote the size of the overlap at position n.

Preliminary similarity score. The ingredients of the preliminary version of the weighted similarity score are the total overlap and the weights. The total overlap of position n is defined as the overlap of up-regulated genes $O_n(G_A, G_B)$ as in Table 2.1 plus the overlap of down-regulated genes $O_n(f(G_A), f(G_B))$, where $f(\cdot)$ refers to the flipped list with down-regulated genes on top. The total overlap A_n at position n is given as:

$$A_n = O_n(G_1, G_2) + O_n(f(G_1), f(G_2)). \tag{2.1}$$

Table 2.1: Overlap $O_n(G_A, G_B)$ of two ordered lists G_A and G_B for the first 10 ranks. The entries of G_A and G_B are randomly chosen Affymetrix probe IDs.

Rank n	G_A	G_B	$O_n(G_A, G_B)$
1	**1771_at**	761_at	0
2	32344_at	**32623_at**	0
3	**222_at**	**1771_at**	1
4	**32623_at**	8993_at	2
5	32793_at	**31569_at**	2
6	**1124_at**	**1124_at**	3
7	**31569_at**	2371_at	4
8	32648_at	312_at	4
9	31636_at	**222_at**	5
10	31355_at	9921_at	5

The weights w_α are chosen to decay exponentially with rank n:

$$w_\alpha = \exp\{-\alpha n\}. \tag{2.2}$$

The parameter α is needed to tune the weights: a smaller α puts more weight on genes further down the list. We shall see later how to choose an appropriate α. The similarity score S'_α is defined as the sum over all weighted overlaps:

$$S'_\alpha(G_A, G_B) = \sum_{n=1}^{\#\text{genes}} \exp\{-\alpha n\} A_n. \tag{2.3}$$

As the weights decrease towards zero for large n, the summation usually stops before reaching rank $n = \#\text{genes}$.

Final similarity score. The definition of the final version of similarity score $S_\alpha(G_A, G_B)$ needs a second parameter besides α:

$$S_\alpha(G_A, G_B) = \max\left\{ \beta\, S'_\alpha\,(G_A, G_B), \quad (1-\beta)\, S'_\alpha\,(G_A, f(G_B)) \right\}, \tag{2.4}$$

with $\beta \in \{0.5, 1\}$. Parameter β is set by the user:

- $\beta = 1$: The class labels of the two studies match. That is, the first class label of study A has the same interpretation as the first class label of study B. The same principle applies for the second class labels. For example, both studies might compare a good to a bad prognosis group. Likewise, both might investigate the same cancer subtypes. Here orientation of the ordered lists is similar: genes on top are up-regulated, genes at the bottom are down-regulated.

– $\beta = 0.5$: The class labels do not match. For example, study A compares different outcomes while study B compares different tissues. Now, the orientation of the two lists is not clear. Thus we take into account both the similarity of the originally ordered lists as well as the similarity of one list to the other list in flipped orientation.

2.2 Tuning α

Choosing a value for parameter α has two effects: it defines the weighing scheme for each rank but also how many ranks are taken into account, that is how far down the lists we evaluate the overlap. Each choice will yield a different similarity score, yet we do not know whether the score deviates substantially from a score based on random lists. Thus we propose a simple tuning procedure: we evaluate the distribution of observed scores and random scores to decide which choice of α leads to reliable scores. To this end, we go back to the original expression data of the two underlying studies. The distribution of observed scores is derived by drawing sub-samples of samples within each class of each study. In the current implementation, we draw 80% sub-samples and then repeat the whole comparison, that is we derive rankings based on the sub-sampled data for each study and re-compute the similarity score. Similarly, the random scores are derived by randomly shuffling the samples within each study. We repeat this procedure B times for each choice of α. Thus, for each α we receive the distributions of B observed and B random scores. We evaluate the *separability* of the two score disributions by applying the pAUC-score [3]. The pAUC-score evaluates the overlap of two distributions. A high score relates to good separation. Hence we choose α such that is provides us with observed scores that separate clearly from random scores. The significance is evaluated by computing an empirical p-value for the *median* observed score based on the set of random scores.

2.3 Comparing lists *without* expression data

We provide a function for comparing only two ranked lists of (gene) identifiers, for which the underlying gene expression data is not at hand. The scoring method is essentially the same. However, we cannot simulate a distribution of observed scores as the sub-sampling of the expression data is not possible. Thus, we cannot find an optimal α. At least we can compute random scores by comparing one list to the randomly shuffled second list. Based on the random scores, an empirical p-value is computed for the observed score. One might then choose an α leading to a significant similarity.

Note a second peculiarity when comparing two lists only. When gene expression data is at hand, the genes are ranked from the most up-regulated to the most down-regulated genes and we have to compute the overlap within the top ranks (up-regulated) and within the

bottom ranks(down-regulated). We call this strategy *two-sided*. However, when comparing lists, we might have a ranking with highly induced genes on top and not induced genes at the lower end. The induced genes are either up- or down-regulated. In this case we only want to compare the top of the lists in order to find significant overlap of induced genes. This is particularly important for experimental contexts, where only top genes in the lists are interesting for biological reasons. We call this strategy *one-sided*. In Chapter 4 we introduce a function working on two lists, for which one-sided or two-sided comparisons can be selected.

Chapter 3

Comparing Two Expression Studies

3.1 `prepareData`: Combining two studies into one expression set

```
prepareData(eset1, eset2, mapping = NULL)
```

The function prepares a collection of two expression sets of class *ExpressionSet* and/or Affy batches of class *AffyBatch* to be passed on to the main function `OrderedList`. For each data set, one has to specify the variable in the corresponding phenodata from which the grouping into two distinct classes is done. The data sets are then merged into one 'ExpressionSet' together with the rearranged phenodata. If the studies were done on different platforms but a subset of genes can be mapped from one chip to the other, this information can be provided via the 'mapping' argument.

Please note that both data sets have to be pre-processed beforehand, either together or independently of each other. The preprocessed gene expression values have to be on an additive scale, that is logarithmic or log-like scale.

The two inputs `eset1` and `eset2` are named lists with five elements:

- `data`: Object of class `ExpressionSet` or `AffyBatch`.
- `name`: Character string with comparison label.
- `var`: Character string with phenodata variable. Based on this variable, the samples for the two-sample testing will be extracted.
- `out`: Vector of two character strings with the levels of `var` that define the two clinical classes. The order of the two levels must be identical for all studies. Ideally, the first entry corresponds to the "bad" and the second one to the "good" outcome level.

– `paired`: Logical - `TRUE` if samples are paired (e.g. two measurements per patients) or `FALSE` if all samples are independent of each other. If data are paired, the paired samples need to be in (whatever) successive order. Thus, the first sample of one condition must match to the first sample of the second condition and so on.

The optional argument `mapping` is a data frame containing one named vector for each study. The vectors are comprised of probe IDs that fit to the rownames of the corresponding expression set. For each study, the IDs are ordered identically. For example, the kth row of `mapping` provides the label of the kth gene in each single study. If all studies were done on the same chip, no mapping is needed (default).

We illustrate the use of function `prepareData` with an application on the exemplary data sets stored in `data(OL.data)`. The data contains a list with three elements: `breast`, `prostate` and `map`. The first two are expression sets of class *ExpressionSet* taken from the breast cancer study of Huang et al. (2003) [2] and the prostate cancer study of Singh et al. (2002) [4]. Both data sets were preprocessed as described in Yang et al. (2006) [5] and contain only a random subsample of the original probes. We further removed unneeded samples from both studies. The labels of the `breast` expression set were extended with 'B' to create two data sets where the probe IDs differ but can be mapped onto each other. The mapping is stored in the data frame `map`, which consists of the two probe ID vectors.

For illustration, we combine the two studies pretending that we need a mapping. The first outcome of both studies relate to bad prognosis, that is "*Rec*urrence vs. *N*on-*Rec*urrence" for the prostate cancer study and "*high* risk vs. *low* risk of relapse" for the breast cancer study.

```
> library(OrderedList)
> data(OL.data)
> OL.data$breast

ExpressionSet (storageMode: lockedEnvironment)
assayData: 1000 features, 30 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 00291004 00291087 ... 00291352 (30 total)
  varLabels: Extension Risk Recurrence
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
```

```
> OL.data$prostate

ExpressionSet (storageMode: lockedEnvironment)
assayData: 1000 features, 21 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: T24__tumor T01__tumor ... T23__tumor (21 total)
  varLabels: sample outcome class
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:

> OL.data$map[1:5,]

   prostate      breast
1   1414_at   1414_at_B
2 1796_s_at 1796_s_at_B
3   1131_at   1131_at_B
4   1316_at   1316_at_B
5   1624_at   1624_at_B

> A <- prepareData(
+ eset1=list(data=OL.data$prostate,name="prostate",var="outcome",out=c("Rec","NRec"),paired
+ eset2=list(data=OL.data$breast,name="breast",var="Risk",out=c("high","low"),paired=FALSE)
+ mapping=OL.data$map
+ )
> A

ExpressionSet (storageMode: lockedEnvironment)
assayData: 1000 features, 51 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: T59__tumor.1 T26__tumor.1 ... 00291352.2 (51 total)
  varLabels: outcome dataset class paired
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
```

3.2 `OrderedList`: Detecting similarities of two expression studies

```
OrderedList(eset, B = 1000, test = "z", beta = 1, percent = 0.95,
verbose = TRUE, alpha = NULL, min.weight = 1e-5)
```

Function `OrderedList` aims for the comparison of comparisons: given two combined expression studies the function produces a gene ranking for each study and quantifies the overlap by computing the weighted similarity scores as introduced in Chapter 2. The final list of overlapping genes consists of those probes that contribute a certain percentage to the overall similarity score.

The input arguments are:

- `eset`: Expression set containing the two studies of interest.
- `B`: Number of internal sub-samples needed to optimize α.
- `test`: String, one of `"fc"` (log ratio = log fold change), `"t"` (t-test with equal variances) or `"z"` (t-test with regularized variances). The z-statistic is implemented as described in Efron et al. (2001) [1].
- `beta`: Either 1 or 0.5. In a comparison where the class labels of the studies match, we set `beta=1`. For example, in each single study the first class relates to bad prognosis while the second class relates to good prognosis. If a matching is not possible, we set `beta=0.5`. For example, we compare a study with good/bad prognosis classes to a study, in which the classes are two types of cancer tissues.
- `percent`: The final list of overlapping genes consists of those probes that contribute a certain percentage to the overall similarity score. Default is `percent=0.95`. To get the full list of genes, set `percent=1`.
- `verbose`: Logical value for message printing.
- `alpha`: A vector of weighting parameters. If set to `NULL` (the default), parameters are computed such that the top 100 to the top 2500 ranks receive weights above `min.weight`.
- `min.weight`: The minimal weight to be taken into account while computing scores.

We apply function `OrderedList` with default values to our combined data set. The result is an object of class *OrderedList* for which print and plot function exist. For the result see Figures 3.1 to 3.3. The sorted list of overlapping genes is stored in `$intersect`.

```
> x <- OrderedList(A, empirical=TRUE)

Simulating score distributions...
          0%.....:.....:.....:.....:.....100%
  Random: ------------------------------------------------- please wait...
Observed: -------------------------------------------------

Computing empirical confidence intervals...
     Top: -------------------------------------------------
  Bottom: -------------------------------------------------

> x

Similarity of Ordered Gene Lists
 Comparison          : breast~prostate
 Number of genes     : 1000
 Test statistic      : z
 Number of subsamples: 1000
 beta = 1 -> corresponding labels could be matched in different studies
-------------------------------------
 Optimal regularization parameter: alpha = 0.03837642
 Lists are more alike in direct order
 Weighted overlap score:  227.432
 Significance of similarity: p-value = 0.05894106
 Number of genes contributing 95 % to similarity score: 82

> x$intersect[1:5]

[1] "1011_s_at/1011_s_at_B" "1022_f_at/1022_f_at_B" "1036_at/1036_at_B"
[4] "103_at/103_at_B"       "1060_g_at/1060_g_at_B"
```

Calling `OrderedList` with the `empirical` option set to true, causes *OrderedList* to compute empirical bounds for expected overlaps shown in Figure 3.3. By default, this is switched off and underestimated bounds deduced from a hypergeometric distribution are drawn.

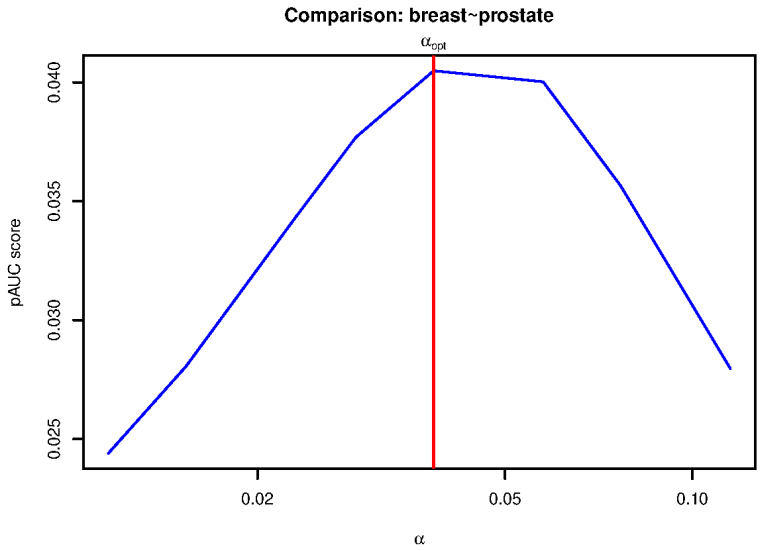


Figure 3.1: `plot(x,"pauc")`: Option `"pauc"` selects the plot of pAUC-scores, based on which the optimal α is chosen. The pAUC-score measure the separability between the two distributions of observed and random similarity scores. The similarity scores depend on α and thus α is chosen where the pAUC-scores are maximal. The optimal α is marked by a vertical line.

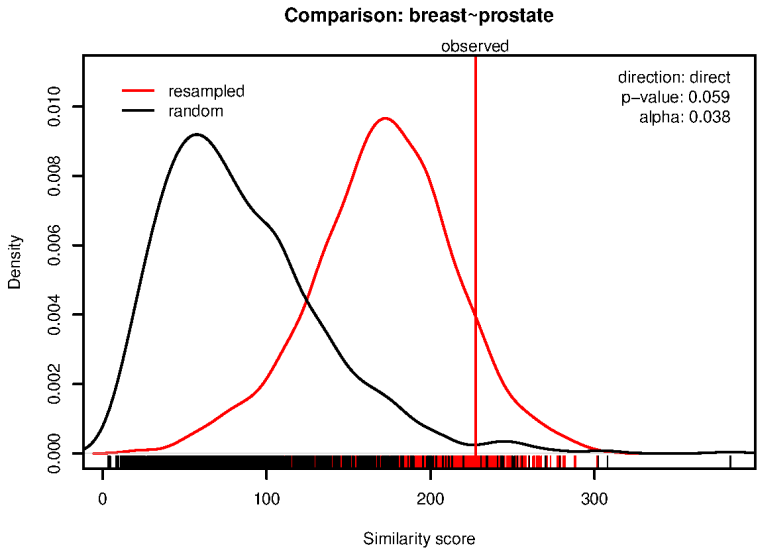


Figure 3.2: `plot(x,"scores")`: Shown are kernel density estimates of the two score distributions underlying the pAUC-score for optimal α. The red curve correspondence to simulated observed scores and the black curve to simulated random scores. The vertical red line denotes the actually observed similarity score. The bottom rugs mark the simulated values. The two distributions got the highest pAUC-score of separability and thus provide the best signal-to-noise separation.

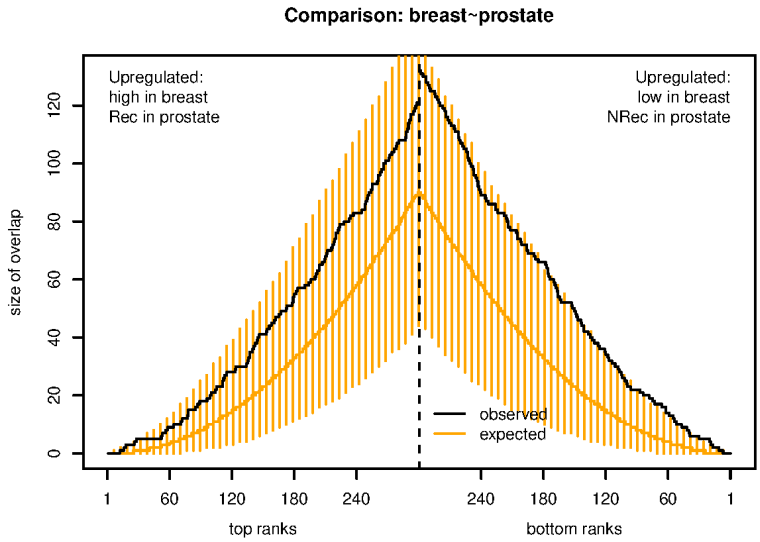


Figure 3.3: `plot(x,"overlap")`: Displayed are the numbers of overlapping genes in the two gene lists. The overlap size is drawn as a step function over the respective ranks. Top ranks correspond to up-regulated and bottom ranks to down-regulated genes. In addition, the expected overlap and 95% confidence intervals derived empirically from the subsampling are shown.

Chapter 4

Comparing Two Ordered Lists

4.1 `compareLists`: Detecting similarities of two ordered gene lists

```
compareLists(ID.List1, ID.List2, mapping = NULL, two.sided = TRUE, B =
1000, alphas = NULL, min.weight = 1e-5, invar.q = 0.5)
```

The two lists received as arguments are matched against each other according to the given mapping. The comparison is performed from both ends by default. Permutations of lists are used to generate random scores and compute empirical p-values. The evaluation is also performed for the case the lists should be reversed.

The input arguments are:

- `ID.List1`: First ordered list of identifiers to be compared.
- `ID.List2`: Second ordered list to be compared, must have the same length as `ID.List1`.
- `mapping`: Maps identifiers between the two lists. This is a matrix with two columns. All items in `ID.List1` must match to exactly one entry of column 1 of the mapping, each element in `ID.List2` must match exactly one element in column 2 of the mapping. If mapping is `NULL`, the two lists are expected to contain the same identifiers and there must be a one-to-one relationship between the two.
- `two.sided`: Whether the score is to be computed considering both ends of the list, or just the top members.
- `B`: The number of permutations used to estimate empirical p-values.

- `alphas`: A set of α candidates to be evaluated. If set to `NULL`, `alphas` are determined such that reasonable maximal ranks to be considered result.
- `min.weight`: The minimal weight to be considered.
- `invar.q`: The fraction of list elements expected to be invariant.

Although `compareLists` is not limited to the use with lists deduced from whole-genome gene expression data, the following aspect is inspired by this application. In whole-genome gene expression data, a large fraction of genes is expected to be invariant in most biologically reasonable comparisons. This hipothesis is for instance used in noramlization of microarray data. In gene lists ordered according to differential expression invariant genes always end up in the middle of the lists. Therefore, they do not influence the similarity score as we define it for the *OrderedList* package. In order to account for this effect when generating random scores, we exclude the fraction of invariant genes determined by `invar.q` from the shuffling for the generation of the similarity score's null distribution. The default value of 50% for `invar.q` is a underestimate typically used in normalization. It may be reconsidered from case to case.

For illustration, we generate two lists from the gene IDs stored in `OL.data$map`. We pretend the lists were already ordered. For the second list, we shuffle within the first 500 ranks and within the last 500 ranks to get some overlap.

```
> list1 <- as.character(OL.data$map$prostate)
> list2 <- c(sample(list1[1:500]),sample(list1[501:1000]))
> y <- compareLists(list1,list2)

  Simulating random scores...
  0%.....:.....:.....:.....:.....100%
  --------------------------------------------------

> y

List comparison
  Assessing similarity of     : top and bottom ranks
  Length of lists             : 1000
  Quantile of invariant genes : 0.5
  Number of random samples    : 1000
--------------------------------------
      Genes      Scores p.values Rev.Scores Rev.p.values
0.115   100    6.488854    0.283   0.000000            1
```

```
0.077   150   19.757828    0.331   0.000000           1
0.058   200   45.980481    0.343   0.000000           1
0.038   300  150.639693    0.364   0.000000           1
0.029   400  345.161784    0.416   0.000000           1
0.023   500  657.646441    0.476   0.000000           1
0.015   750 2159.386018    0.600   7.062765           1
```

The returned object of class *listComparison* can be explored by a plot function providing a series of overlap plots similar to Figure 3.3 and a series of random score distributions similar to Figure 3.2. The print function returns the table above summarizing the results. Now we might want to choose a specific α possibly leading to a significant score and extract the resulting set of intersecting list identifiers. This is done by applying function `getOverlap`:

```
getOverlap(x, max.rank = 100, percent = 0.95)
```

The inputs are:

- `x`: An object of class *listComparison*.
- `max.rank`: The maximum rank to be considered.
- `percent`: The final list of overlapping genes consists of those probes that contribute a certain percentage to the overall similarity score. Default is `percent=0.95`. To get the full list of genes, set `percent=1`.

Note that we have two results per α: the similarity score for the comparison of the originally ordered lists and the reversed score for the comparison of one original to one reversed list. Function `getOverlap` chooses the direction with the higher similarity score. In our example above, the direct comparison is clearly the right choice. Following the example, we set the number of genes to 100 and extract the overlapping IDs. In the first 100 top ranks and the first 100 bottom ranks we find a set of 17 overlapping IDs. The result object is of class *listComparisonOverlap*, for which again print and plot functions exist, see Figures 4.1 and 4.2.

```
> z <- getOverlap(y)
> z

List comparison
  Assessing similarity of              : top and bottom ranks
  Length of lists                      : 1000
  Number of random samples             : 1000
-----------------------------------------------------------
```

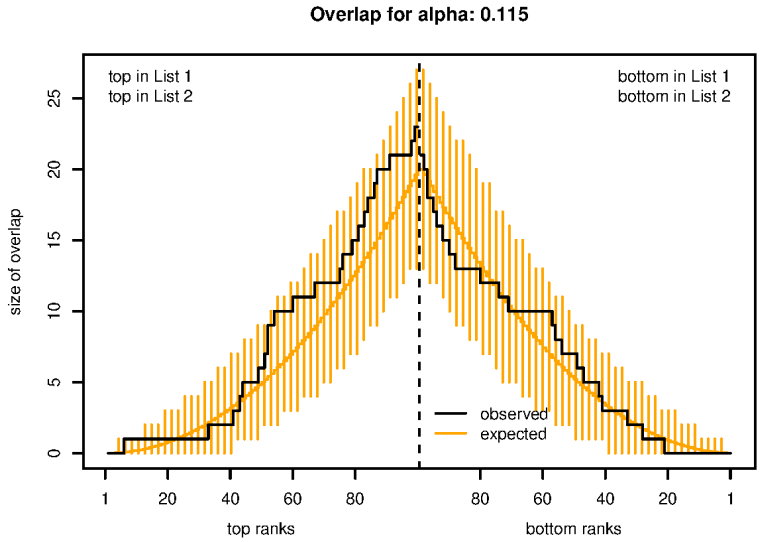


Figure 4.1: `plot(z)`: Displayed are the numbers of overlapping genes in the two gene lists. The overlap size is drawn as a step function over the respective ranks. Top ranks correspond to up-regulated and bottom ranks to down-regulated genes. In addition, the expected overlap and 95% confidence intervals derived from a hypergeometric distribution are shown.

```
  Lists are more alike in direct order
  Chosen regularization parameter       : alpha = 0.115 ( 100 genes)
  Weighted overlap score                : 6.488854
  Significance of similarity            : p-value = 0.283
  Score percentage for common entries   : 95
  Entries contributing score percentage : 17
```

```
> z$intersect[1:5]
```

```
[1] "1076_at"   "1131_at"   "1195_s_at" "1226_at"   "123_at"
```

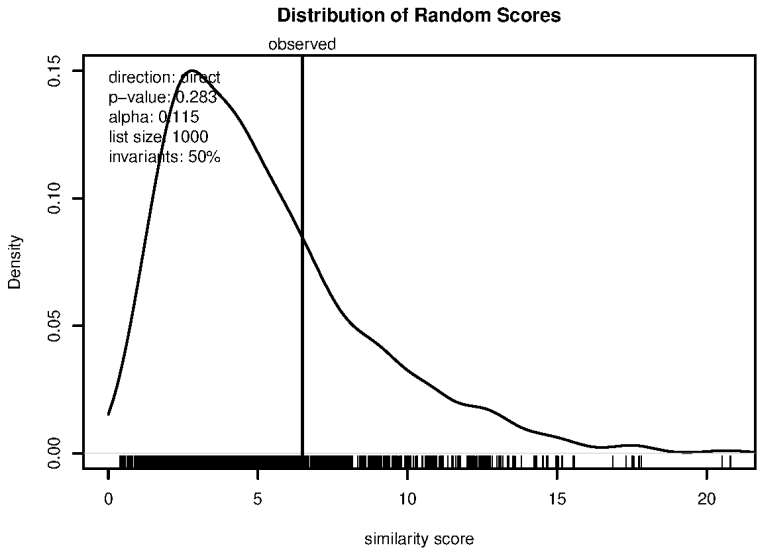


Figure 4.2: `plot(z,"scores")`: Shown are kernel density estimates of the distribution of random similarity scores. The observed score is marked by the vertical line.

Chapter 5

Bibliography

[1] Efron B, Tibshirani R, Storey JD, and Tusher V (2001): "Empirical Bayes analysis of a microarray experiment", *Journal of the American Statistical Society* **96**, 1151–1160.

[2] Huang E, Cheng S, Dressman H, Pittman J, Tsou M, Horng C, Bild A, Iversen E, Liao M, Chen C, West M, Nevins J, and Huang A (2003): "Gene expression predictors of breast cancer outcomes", *Lancet* **361**, 1590–1596.

[3] Pepe MS, Longton G, Anderson GL, and Schummer M (2003): "Selecting Differentially Expressed Genes from Microarray Experiments", *Biometrics* **59**, 133–142.

[4] Singh D, Febbo PG, Ross K, Jackson DG, Manola J, Ladd C, Tamayo P, Renshaw AA, D'Amico AV, Richie JP, Lander E, Loda M, Kantoff PW, Golub TR, and Sellers WR (2002): "Gene expression correlates of clinical prostate cancer behavior", *Cancer Cell* **1**, 203–209.

[5] Yang X, Bentink S, Scheid S, and Spang R (2006): "Similarities of ordered gene lists", to appear in *Journal of Bioinformatics and Computational Biology.*