`GeneSelector` package vignette

Martin Slawski *
Anne-Laure Boulesteix †

Sylvia Lawry Centre, Hohenlindenerstr. 1, D-81677 Munich, Germany

1 Statistical and Bioinformatical background

One of the most important aspects of microarray data analysis is the detection of genes that are differentially expressed, e.g., in different experimental conditions or in individuals with different phenotypes. The results of microarray studies are usually the starting point for further more expensive and time-consuming experiments, which involve only a small number of candidate genes that turned out to be 'most promising' in the previous step Aerts et al. [2006]. Considering the induced efforts and costs, finding an accurate ranking of genes that reduces the fraction of false positives under the top-ranking genes is a crucial challenge in microarray data analysis.

The name `GeneSelector` refers to the following filter mechanism: Starting from a large set of candidate genes, it stepwise excludes more and more genes until one ends up with a very sparse set, called 'selected' genes. Its elements have the following property. Given some threshold rank that is still considered biologically relevant, the ranks of the selected genes are better than or at least equal to the threshold in all rankings performed using different methods and different perturbed data sets. In a word, they are consistently identified as differentially expressed.

Several rankings are generated using two approaches. On the one hand, the original dataset is slightly perturbed, by randomly leaving one- or several array(s) out, adding noise or swapping class labels. This procedure is repeated many times. On the other hand, one can use several ranking methods or test statistics - the current implementation now features fifteen different procedures, including both traditional and modern approaches (see an overview in Section 2). The first approach tries to mimic a changed data situation and is therefore helpful for assessing stability of the results (Pavlidis et al. [2003], Qiu et al. [2007]), while the second one follows the spirit of a 'sensitivity analysis'. Almost all of the considered test statistics rely on idealized assumptions and it is hard to check whether they hold for a particular dataset at hand. In this context, using several statistics is beneficial.

Furthermore, a major topic in `GeneSelector` is stability. If results tremendously change when the data are slightly perturbed, they are little credible. We thank In fact, microarray data are suspected to yield 'noise discovery' Ioannidis [2005]. To gain insight into stability, `GeneSelector` computes several stability measures as discussed in the diploma thesis of Elisabeth Gnatowski with the title 'Stability of methods for the analysis of differential gene expression' Gnatowski [2007] which served as a starting point in the development of `GeneSelector`.

Let $\boldsymbol{X} = (x_{ji})_{\substack{j=1,\ldots,p \\ i=1,\ldots,n}}$ denote the matrix of gene expression values (relative or absolute

*Martin.Slawski@campus.lmu.de
†http://www.slcmsr.net/boulesteix

intensity values) and $\boldsymbol{y} = (y_i)$ the vector of class labels with indices $i = 1, \ldots, n$ for the observations (arrays) and $j = 1, \ldots, p$ for the genes. For notational convenience, we set $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$. All the procedures considered here return a triple $t(\mathcal{D}) = (\boldsymbol{s}, \boldsymbol{p}, \boldsymbol{r})$, where

- $\boldsymbol{s}$ is a $p \times 1$ vector of test statistics for each gene,
- $\boldsymbol{p}$ is the corresponding vector of p-values given the statistic provides one,
- $\boldsymbol{r}$ is a vector of *ranks* for each gene, where rank 1 is attributed to the gene being 'most distant' from the null hypothesis, i.e. most differentially expressed.

In this context, $t(\cdot)$ can in general be applied to the independent two-sample case, the paired two-sample case and the one-sample case.
In the `GeneSelector` package, the focus is on the ranks $\boldsymbol{r}$. The idea behind `GeneSelector` is to exclude as many genes as possible from the subset of candidate genes for differential expression, in order to avoid false positives. *selected* genes are those present at the top of the ranking with various featured procedures $t(\cdot)$ as well as with B *perturbed* data sets, which are denoted as $\{\widetilde{\mathcal{D}}\}_{b=1}^{B}$. The `GeneSelector` package creates perturbed data sets in order to take the stability of the findings into account. By 'perturbed data sets' we mean data sets that result from

- leaving out samples,
- swapping class labels ($\widetilde{\boldsymbol{X}} = \boldsymbol{X}$), but $\boldsymbol{y}$ changes),
- generating bootstrap replicates,
- adding noise.

Results from 'perturbed' data sets and original dataset ($\mathcal{D}$) can be aggregated in `GeneSelector` (separately for each different procedure) in two ways (s. below), obtaining an aggregated ranking $\boldsymbol{r}^{\text{agg}}(\mathcal{D}; t(\cdot))$. After defining (or determining) some 'threshold rank' $1 \leq \theta \ll p$, the `GeneSelector` procedure can be stated as follows:

1. Choose a set of test procedures for differential expressions $\{t_k(.)\}_{k=1}^{K}$. For instance, one may choose $t_1(.)$ as the ordinary t-test procedure and $t_2(.)$ as the foldchange procedure.
2. Apply each $t_k(.)$ to the original dataset $\mathcal{D}$ and to different perturbed data sets $\{\widetilde{\mathcal{D}}_b\}_{b=1}^{B}$
3. Aggregate the resulting rankings to obtain $\boldsymbol{r}_k^{\text{agg}},\ k = 1, \ldots, K$.
4. Define an order on the set of test procedures $t_{(1)}, \ldots, t_{(K)}$ so that $t_{(1)}$ is attributed most importance, $t_{(2)}$ second most and so on.
5. The final ranking follows now the following criteria

   (1) 'Selection' in $t_{(1)}$

   $\vdots$

   (K) 'Selection' in $t_{(K)}$

   (K+1) Rank in $t_{(1)}$

   $\vdots$

   (2K) Rank in $t_{(K)}$

where 'Selection' means that the respective rank is below θ

The parameter θ can be interpreted as the highest rank one is still interested in. For instance, if one expects 10 differentially expressed genes, θ could be set to that value, but it should usually taken higher, because the procedure outlined above will normally produce less 'selected' genes than θ.
Admittedly, the choice of θ is not easy. One possibilty of finding this value adaptively consists of using multiple testing procedures (s. example below).

2 Overview of featured test procedures

There is a vast literature about test statistics to be used in the microarray setting. Here, we provide a selection of 15 procedures. Of course, this selection is non-exhaustive, but we have tried to include a good 'mixture' of (sometimes completely different) approaches. They can be categorized in the following way (the names of the corresponding functions implementing them are written in typewriter):

- the naive approaches: Foldchange (`RankingFC`), a 'gap' statistic (`RankingGap`)
- 'ordinary' statistics: (usual) t-statistic (`RankingTstat`), Welch statistic (`RankingWelch`), Wilcoxon statistic (`RankingWilcoxon`)
- hierarchical bayesian models: Baldi-Long t-statistic (`RankingBaldiLong`, Baldi and Long [2001]), Fox-Dimmic t-statistic (`RankingFoxDimmic`, Fox and Dimmic [2006]), the B-statistic (`RankingBstat`, Lönnstedt and Speed [2002]), 'moderated' t-statistic (`RankingLimma`, Smyth [2004])
- two mixture model approaches named `RankingEbam` and `RankingWilcEbam`, respectively (Efron et al. [2001]; Efron and Tibshirani [2002]) that are connected with Significance Analysis for Microarrays (SAM, Tusher et al. [2001]) that is also included (`RankingSam`).
- permutation-based t-statistics (`RankingPermutation`)
- 'modified' t-statistics: shrinkage t-statistic (`RankingShrinkageT`, Opgen-Rhein and Strimmer [2007]), soft-threshold t-statistic (`RankingSoftthresholdT`, Wu [2005])

3 An artifical dataset example

3.1 Generating the data set

For demonstration purposes, we use a simulated datsets containing $2,000$ genes. It can be characterized in the following manner.

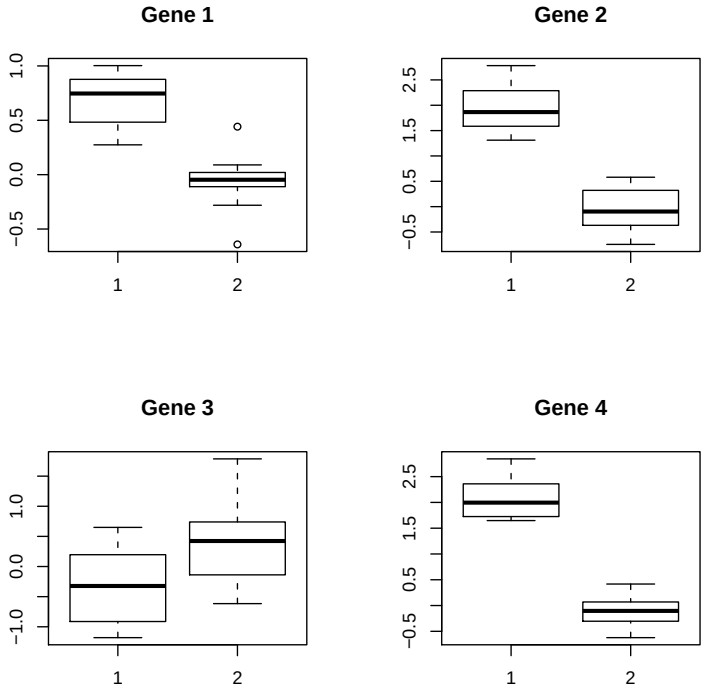
- $\boldsymbol{X}$ is drawn from a multivariate normal distribution with zero mean vector and covariance matrix $\boldsymbol{\Sigma}$.
- $\boldsymbol{\Sigma}$ is drawn randomly from an Inverse Wishart distribution.
- $\boldsymbol{y}$ consists of ten samples for each class, in the following regarded as two independent samples.
- The first 40 genes (rows) of $\boldsymbol{X}$ are differentially expressed, where the differences $\Delta_j, j = 1, \ldots, 40$ in mean between the two classes were simulated independently according to a normal distribution with variance 0.9.

We access the data using the lines:

```
> data(toydata)
> yy <- as.numeric(toydata[1, ])
> xx <- as.matrix(toydata[-1, ])
> dim(xx)

[1] 2000   20

> table(yy)

yy
 1  2
10 10
```

Knowing that the first genes are actually differentially expressed, we make boxplots of the first four genes:

```
> par(mfrow = c(2, 2))
> for (i in 1:4) boxplot(xx[i, ] ~ yy, main = paste("Gene", i))
```



3.2 Ranking the genes

We now peform 6 rankings from six methods: `ordinaryT` (ordinary t-test), `Baldi-LongT` (Baldi-Long t-statistic), `FoxDimmicT` (Fox-Dimmic t-statistic), `SAM`, `Wilcoxon`, `WilcEbam` and a modified version of the Wilcoxon statistic derived from a mixture model.

```
> ordT <- RankingTstat(xx, yy, type = "unpaired")
> BaldiLongT <- RankingBaldiLong(xx, yy, type = "unpaired")
> FoxDimmicT <- RankingFoxDimmic(xx, yy, type = "unpaired")
> sam <- RankingSam(xx, yy, type = "unpaired")
> wilcox <- RankingWilcoxon(xx, yy, type = "unpaired", pvalues = TRUE)
> wilcoxefron <- RankingWilcEbam(xx, yy, type = "unpaired")
```

The resulting objects are all instances of the class `GeneSelector`.
To get some class information, we use the commands:

```
> class(ordT)

[1] "GeneRanking"
attr(,"package")
[1] "GeneSelector"

> getSlots("GeneRanking")

          x           y   statistic     ranking        pval        type
   "matrix"    "factor"   "numeric"   "numeric"    "vector" "character"
     method
"character"

> str(ordT)

Formal class 'GeneRanking' [package "GeneSelector"] with 7 slots
  ..@ x        : num [1:2000, 1:20] 1 2.78 -1.18 2.79 -2.95 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2000] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:20] "arr1" "arr2" "arr3" "arr4" ...
  ..@ y        : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
  ..@ statistic: Named num [1:2000] 13.09 10.4 9.53 -8.38 -7.93 ...
  .. ..- attr(*, "names")= chr [1:2000] "4" "11" "2" "26" ...
  ..@ ranking  : int [1:2000] 4 11 2 26 5 9 23 38 28 40 ...
  ..@ pval     : Named num [1:2000] 1.23e-10 4.83e-09 1.85e-08 1.26e-07 2.79e-07 ...
  .. ..- attr(*, "names")= chr [1:2000] "4" "11" "2" "26" ...
  ..@ type     : chr "unpaired"
  ..@ method   : chr "ordinaryT"
```

The following 'convenience' methods are available:

```
> show(ordT)

Ranking by ordinaryT
number of genes: 2000

> summary(ordT)

        statistic  p_values
Min.     -8.37800 1.233e-10
1st Qu.  -1.00300 1.184e-01
Median   -0.07951 3.616e-01
Mean     -0.04111 4.042e-01
3rd Qu.   0.88980 6.671e-01
Max.     13.09000 9.992e-01
```

```
> toplist(ordT)

  index statistic        pvals
1     4   13.0879 1.232978e-10
```

The last command yields the top-ranking genes according to the chosen procedure.

3.3 Perturbed data sets

One major feature of the `GeneSelector` package is the incorporation of 'stability' aspects. A procedure $t(\cdot)$ as defined above is called *stable* if it produces similar results from perturbed data sets, i.e if

$$t(\mathcal{D}) \approx t(\widetilde{\mathcal{D}}_1) \approx \ldots \approx t(\widetilde{\mathcal{D}}_B).$$

For stability assessment, we first generate a `FoldMatrix`.

```
> loo <- GenerateFoldMatrix(xx, yy, k = 1)
> show(loo)

number of removed samples per replicate: 1
number of replicates: 20
constraints: minimum classize for each class: 9
```

The last command produces data sets where one sample has been removed. We plug this into the method `GetRepeatRanking` to repeat the ranking 20 times, i.e. for each removed observation:

```
> loor_ordT <- GetRepeatRanking(ordT, loo)
> loor_BaldiLongT <- GetRepeatRanking(BaldiLongT, loo)
> loor_FoxDimmicT <- GetRepeatRanking(FoxDimmicT, loo)
> loor_sam <- GetRepeatRanking(sam, loo)
> loor_wilcox <- GetRepeatRanking(wilcox, loo)
> loor_wilcoxefron <- GetRepeatRanking(wilcoxefron, loo)
```

The object `loo` can also be used in the following manner:

```
> ex1r_ordT <- GetRepeatRanking(ordT, loo, scheme = "Labelexchange")
```

`scheme = "Labelexchange"` means that instead of leaving one observation out per iteration, they are given the opposite class label.
We can also use bootstrapping, e.g. :

```
> boot <- GenerateBootMatrix(xx, yy, maxties = 3, minclassize = 5,
+     repl = 30)
> show(boot)

number of bootstrap replicates: 30
constraints: minimum classize for each class: 5
              maximum number of ties per observation: 3

> boot_BaldiLongT <- GetRepeatRanking(BaldiLongT, boot)
```

... or add noise, e.g.:

```
> noise_ordT <- GetRepeatRanking(ordT, varlist = list(genewise = TRUE,
+     factor = 1/10))
```

Note that *no* matrix (`Boot` or `Fold`) is provided for the variant with added noise.
To get a toplist that takes into account all iterations, we can use:

```
> toplist(loor_ordT, show = FALSE)

original dataset:

   index statistic        pvals
4      4 13.087900 1.232978e-10
11    11 10.404717 4.833338e-09
2      2  9.533551 1.853163e-08
26    26 -8.378361 1.261238e-07
5      5 -7.927116 2.791221e-07
9      9 -7.744184 3.880996e-07
23    23  7.392767 7.402187e-07
38    38 -6.986973 1.592778e-06
28    28 -6.786421 2.345378e-06
40    40  6.421584 4.808461e-06
```

As an exploratory tool to examine the difference between original and perturbed data sets, `plot` commands have been defined:

From figure 1, it is obvious that the top ranks are relatively stable as compared to higher ranks. Unsurprisingly, stability depends massively on the method used to create perturbed data sets. In this example, the bootstrap rankings are far more scattered around the optimum least squares line than the Jackknife rankings.

To combine several schemes, one can use the 'join' functionality

```
> perturb_ordT <- join(ex1r_ordT, noise_ordT)
> show(perturb_ordT)

30 rankings with scheme 'combined'
Method used: ordinaryT
```

3.4 Objective stability measures

Instead of using informal visual methods, one can compute several stability measures.

3.4.1 Multiple linear regression model

The preferred stability measure is based on a (multivariate) linear regression model

$$\boldsymbol{R} = \boldsymbol{Z}\boldsymbol{\Gamma} + \boldsymbol{\Xi}$$

where the $p \times B$ 'response' matrix $\boldsymbol{R} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_B]$ contains in its columns the vector of ranks resulting from the perturbed data sets $\widetilde{\mathcal{D}}_1, \ldots, \widetilde{\mathcal{D}}_B$ and the $p \times 2$ regressor matrix $\boldsymbol{Z}$ is given as $\boldsymbol{Z} = [1, \boldsymbol{r}(\mathcal{D})]$. The ranking $\boldsymbol{r}(\mathcal{D})$ resulting from the original dataset is thus considered as 'covariate'. The $2 \times B$ matrix $\boldsymbol{\Gamma}$ of regression coefficients has to

```
> par(mfrow = c(2, 2))
> plot(loor_ordT, col = "blue", pch = ".", cex = 2.5)
> plot(ex1r_ordT, col = "blue", pch = ".", cex = 2.5)
> plot(boot_BaldiLongT, col = "blue", pch = ".", cex = 2.5)
> plot(noise_ordT, frac = 1/10, col = "blue", pch = ".", cex = 2.5)
```

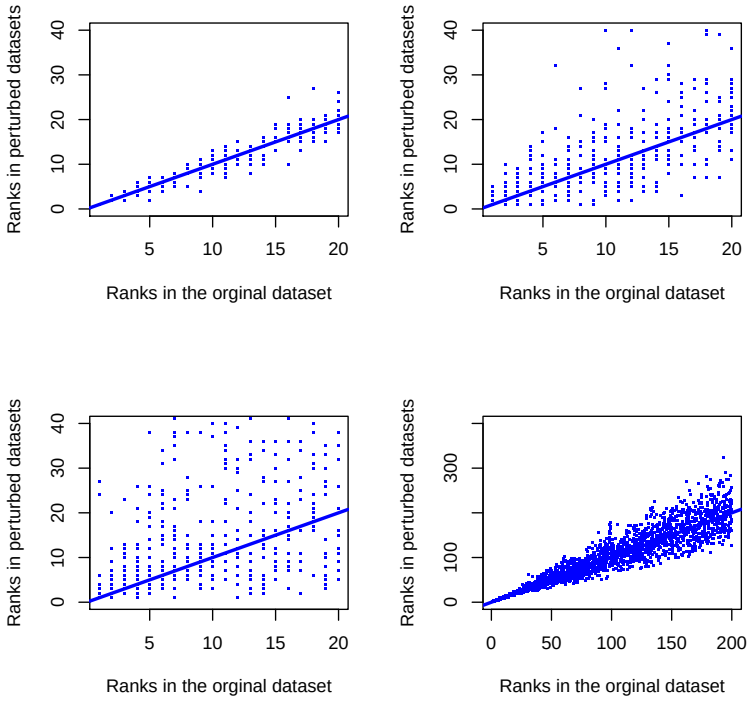


Figure 1: Scatterplots of rankings from perturbed datasets vs. rankings from original dataset. Top, left: Removal of one array per iteration (Jackknife). Top, right: Exchange of one class label per iteration. Bottom, left: Bootstrap. Bottom, right: Addition of noise.

be estimated. Additionally, a weight vector $\boldsymbol{w}$ is used to attribute more importance to the top ranks. An estimator for $\boldsymbol{\Gamma}$ is obtained by minimizing the weighted least squares criterion

$$\widehat{\boldsymbol{\Gamma}} = \underset{\boldsymbol{\Gamma} \in \mathbb{R}^{2 \times B}}{\arg\min} \ \mathrm{tr}((\boldsymbol{R} - \boldsymbol{Z}\boldsymbol{\Gamma})^\top \boldsymbol{W} (\boldsymbol{R} - \boldsymbol{Z}\boldsymbol{\Gamma}))$$

with $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{w})$. Stability is then measured via a goodness- of-fit criterion for linear models.

```
> stab_lm_ordT <- GetStabilityLm(loor_ordT, decay = "linear")
> show(stab_lm_ordT)

Stability measure: weighted linear regression,
 weighting:  based on ranks ,  linear weight decay
 multivariate R2 is:  0.9549876

> stab_lm_BaldiLongT <- GetStabilityLm(loor_BaldiLongT, decay = "linear")
> show(stab_lm_BaldiLongT)
```

```
Stability measure: weighted linear regression,
 weighting:  based on ranks ,  linear weight decay
 multivariate R2 is:  0.9522463

> stab_lm_FoxDimmicT <- GetStabilityLm(loor_FoxDimmicT, decay = "linear")
> show(stab_lm_FoxDimmicT)

Stability measure: weighted linear regression,
 weighting:  based on ranks ,  linear weight decay
 multivariate R2 is:  0.9405184

> stab_lm_sam <- GetStabilityLm(loor_sam, decay = "linear")
> show(stab_lm_sam)

Stability measure: weighted linear regression,
 weighting:  based on ranks ,  linear weight decay
 multivariate R2 is:  0.9555079

> stab_lm_wilcox <- GetStabilityLm(loor_wilcox, decay = "linear")
> show(stab_lm_wilcox)

Stability measure: weighted linear regression,
 weighting:  based on ranks ,  linear weight decay
 multivariate R2 is:  0.6866364

> stab_lm_wilcoxefron <- GetStabilityLm(loor_wilcoxefron, decay = "linear")
> show(stab_lm_wilcoxefron)

Stability measure: weighted linear regression,
 weighting:  based on ranks ,  linear weight decay
 multivariate R2 is:  0.9147122
```

This shows that all six methods are of comparable stability.

3.4.2 Overlap score

As an alternative, the Overlap Score (Lottaz et al. [2006]) can be used.
Its computation depends on weights that decrease exponentially with the rank:

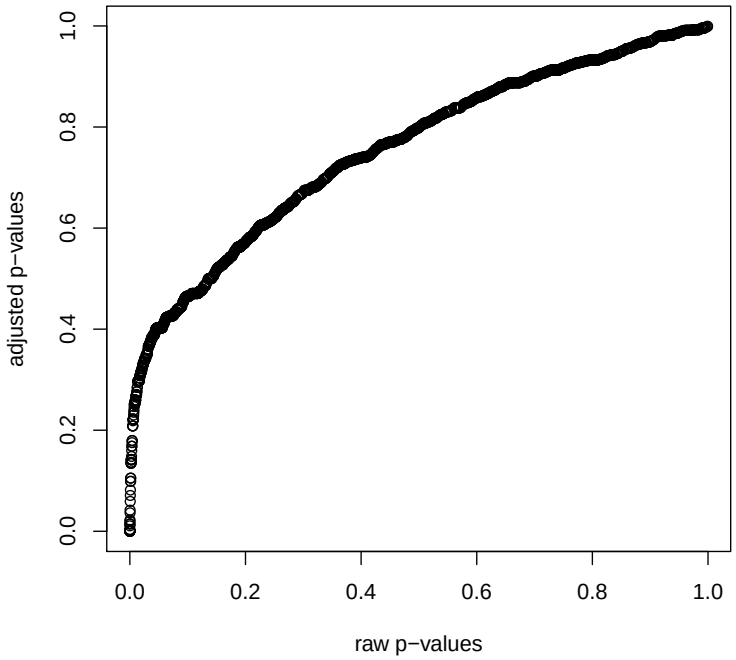
$$w(r) = \exp(-\alpha r)$$

One possible approach uses information from (adjusted) p-values incorporated in a nonlinear least squares regression to find an appropriate value for α.

$$\alpha^{\text{opt}} = \arg\min_{\alpha} \sum_{r=1}^{p} (p(r) - (1 - \exp(-\alpha r)))^2,$$

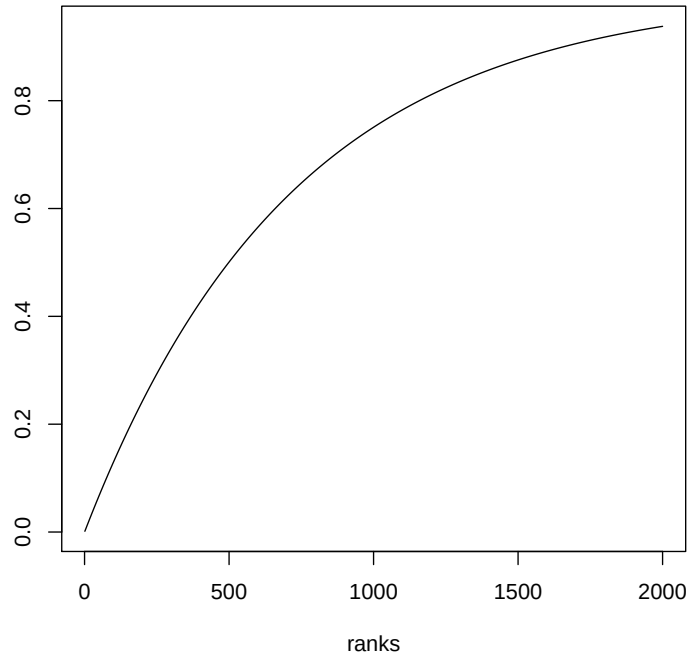
where $p(r)$ is the p-value belonging to rank r.
To follow this procedure, we use the lines:

```
> ordT_adjpval <- AdjustPvalues(ordT@pval, method = "BH")
> plot(ordT@pval, ordT_adjpval, xlab = "raw p-values", ylab = "adjusted p-values")
```



We can then use adjusted p-values for a nonlinear least squares Regression

```
> alphaopt <- GetAlpha(ordT@ranking, ordT_adjpval)
> plot(1:length(ordT@ranking), 1 - exp(-alphaopt * (1:length(ordT@ranking))),
+     type = "l", xlab = "ranks", ylab = "")
> stab_overlap_ordT <- GetStabilityOverlap(loor_ordT, decay = "exponential",
+     alpha = alphaopt)
> plot(stab_overlap_ordT)
```



3.4.3 Recovery score

As a third stabilty measure, the Recovery Score (Pavlidis et al. [2003]) is implemented:

```
> rs_ordT <- RecoveryScore(loor_ordT, method = "BH")
> print(rs_ordT)
```

```
        1         2         3         4         5         6         7         8
1.0000000 1.0000000 1.0000000 0.9047619 1.0000000 1.0000000 1.0000000 0.9090909
        9        10        11        12        13        14        15        16
0.8076923 0.9545455 0.9500000 0.6562500 1.0000000 1.0000000 1.0000000 1.0000000
       17        18        19        20
1.0000000 1.0000000 0.9500000 1.0000000
```

3.5 Aggregating the results from different procedures

After stability assessment, ranks from different perturbed data sets and the original dataset can be aggregated in two ways.

3.5.1 Bayesian approach

The first method uses a bayesian model:

```
> agg_ordT <- AggregateBayes(loor_ordT, stab_lm_ordT, tau = 1)
> agg_BaldiLongT <- AggregateBayes(loor_BaldiLongT, stab_lm_BaldiLongT,
+     tau = 1)
> agg_FoxDimmicT <- AggregateBayes(loor_FoxDimmicT, stab_lm_FoxDimmicT,
```

```
+     tau = 1)
> agg_sam <- AggregateBayes(loor_sam, stab_lm_sam, tau = 1)
> agg_wilcox <- AggregateBayes(loor_wilcox, stab_lm_wilcox, tau = 1)
> agg_wilcoxefron <- AggregateBayes(loor_wilcoxefron, stab_lm_wilcoxefron,
+     tau = 1)
```

3.5.2 Simple aggregation

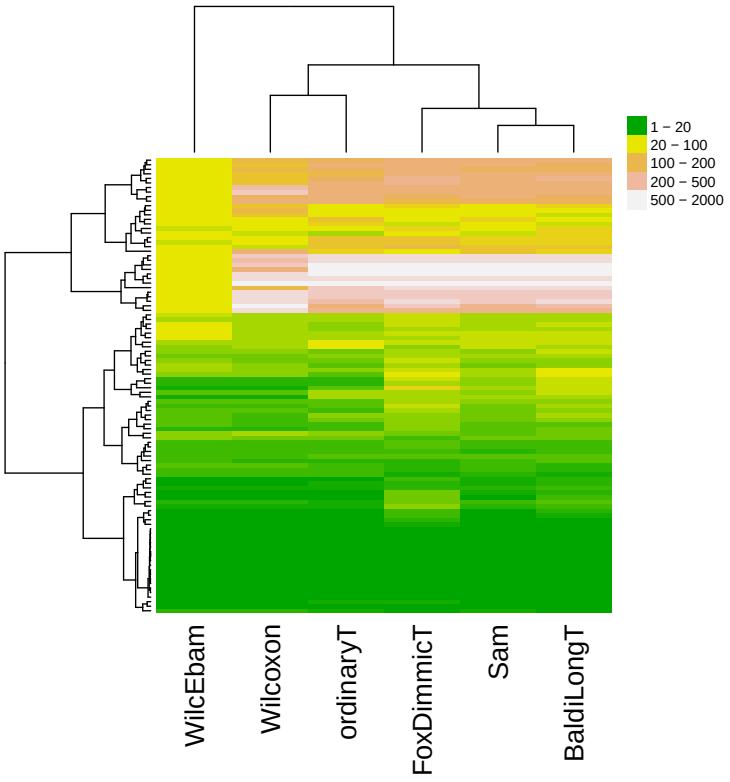
The hyperparameter `tau` controls the 'confidence' in the ranking based on the original dataset. For faster (and similar results), one uses 'AggregateSimple':

```
> agg_simple <- AggregateSimple(loor_BaldiLongT, stab_lm_BaldiLongT,
+     aggregatefun = "mean")
```

3.5.3 Visualization via Heatmap

Different procedures $t(\cdot)$ can be visually compared using a heatmap, that clusters genes and procedures simultaneously.

```
> statlist <- list(agg_ordT, agg_BaldiLongT, agg_FoxDimmicT, agg_sam,
+     agg_wilcox, agg_wilcoxefron)
> HeatmapMethods(statlist, ind = 1:100)
```



Note that the `Heatmap` depicts only the first 100 genes (argument `ind`) and that the objects have to be collected in the list `statlist`.

3.6 The gene selector

As a last step, we run the `GeneSelector`. The idea is to select those genes which are not well-ranked by all procedures.

Let us assume that SAM is our preferred statistic, followed by Wilcoxon, Baldi-Long, FoxDimmic, OrdinaryT and WilcEbam, then the vector defining the order for `statlist` has the following form:

```
> ordstat <- c(4, 5, 2, 3, 1, 6)
```

The `ordstat` vector defines the order of statistics in the procedure described in the procedure described above. It is also used to establish a final ranking with respect to 'selection' firstly and ranks secondly, with 'selection' in the highest ranked statistic counting most. We set the threshold θ to 50 (`threshold = "user", maxrank = 50`), that is 'selection' means that the gene is on the top 50 genes in all considered statistics. This yields:

```
> gk50 <- GeneSelector(statlist, ind = NULL, indstatistic = ordstat,
+     threshold = "user", maxrank = 50)
```

We can also define the threshold based on a multiple testing procedure that is applied to the raw pvalues of the first statistic in `ordstat` (here, this is SAM). The threshold is then the number of genes with p-values $\geq$ `maxpval`.

```
> gpval <- GeneSelector(statlist, ind = NULL, indstatistic = ordstat,
+     threshold = "BH", maxpval = 0.05)
```

The results can now be visualized as follows:

```
> show(gk50)

GeneSelector run with gene rankings from the following statistics:
Sam
Wilcoxon
BaldiLongT
FoxDimmicT
ordinaryT
WilcEbam
Number of genes below threshold rank  50  in all statistics: 29

> show(gpval)

GeneSelector run with gene rankings from the following statistics:
Sam
Wilcoxon
BaldiLongT
FoxDimmicT
ordinaryT
WilcEbam
Number of genes below threshold rank  21  in all statistics: 11

> toplist(gpval)
```

```
  index        pvals
1     4 6.306192e-05
```

```
> SelectedGenes(gpval)
```

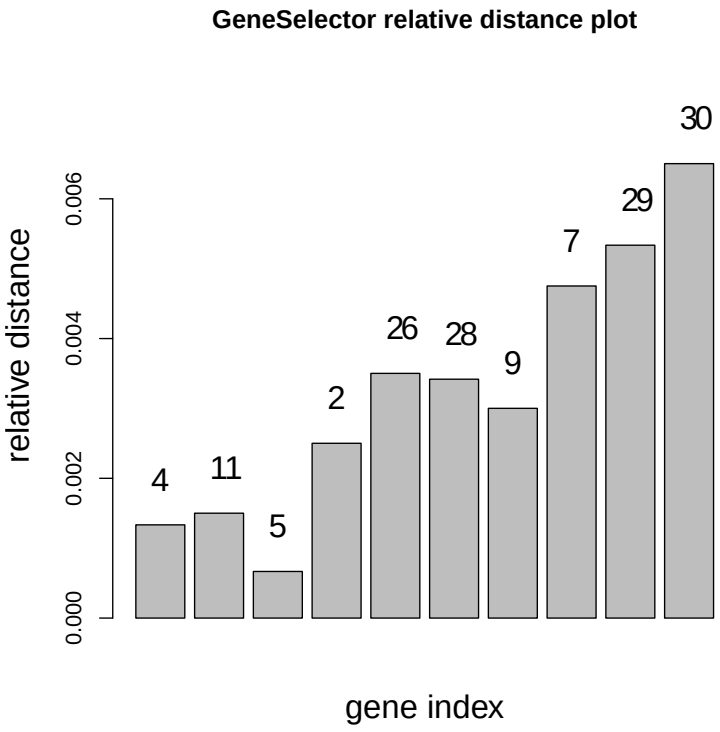
```
  index        pvals
1     4 6.306192e-05
```

The multiple testing procedure sets the threshold rank more strictly. Note that we have still one positive gene in `gkpval`. The following plot shows (as a barplot) the absolute relative distance for the top-ranked genes defined as

$$d_{\ell_1}((r_{j,t_1},\ldots,r_{j,t_K})) = \sum_{k=1}^{K}(r_{j,t_k}-1),$$

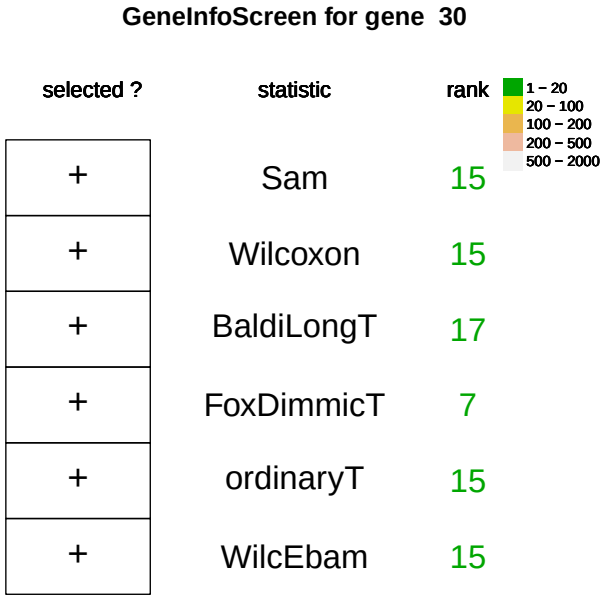
an ℓ_1-distance from the best possible result, rank 1 for all considered procedures $t_1(\cdot),\ldots,t_K(\cdot)$.

```
> plot(gpval)
```



If one wants to get information about a specific gene (here gene indexed 30) one uses:

```
> GeneInfoScreen(gpval, which = 30)
```



References

S. Aerts, D. Lambrechts, S. Maity, P. Van Loo, B. Coessens, F. DeSmet, L.-C. Tranchevent, B. DeMoor, P. Marynen, B. Hassan, P. Carmeliet, and Y. Moreau. Gene prioritization through genomic data fusion. *Nature Biotechnology*, pages 537–544, 2006.

Pierre Baldi and Anthony D Long. A Bayesian framework for the analysis of microarray expression data: regularized t-test and statistical inferences of gene changes. *Bioinformatics*, 17:509–519, 2001.

Bradley Efron and Robert Tibshirani. Empirical Bayes Methods and False Discovery Rates for Microarrays. *Genetic Epidemiology*, 23:70–86, 2002.

Bradley Efron, Robert Tibshirani, John D Storey, and Virginia Tusher. Empirical Bayes Analysis of a Microarray Experiment. *Journal of the American Statistical Association*, 96:1151–1160, 2001.

Richard J Fox and Matthew W Dimmic. A two sample Bayesian t-test for microarray data. *BMC Bioinformatics*, 7:126, 2006.

E. Gnatowski. Stability of methods for the analysis of differential gene expression. *Diploma thesis, Department of Statistics, Ludwig-Maximimilians-University Munich*, 2007.

J. P. A. Ioannidis. Microarrays and molecular research: noise discovery? *The Lancet*, 365:454–455, 2005.

Ingrid Lönnstedt and Terry Speed. Replicated Microarray Data. *Statistica Sinica*, 12: 31–46, 2002.

Claudio Lottaz, Xinan Yang, Stefanie Scheid, and Rainer Spang. OrderedList - a Bioconductor package for detecting similarity in ordered gene lists. *Bioinformatics*, 22:2315–2316, 2006.

Rainer Opgen-Rhein and Korbinian Strimmer. Accurate Ranking of Differentially Expressed Genes by a Distribution-Free Shrinkage Approach. *Statistical Applications in Genetics and Molecular Biology*, 6:Iss.1, Art.9, 2007.

P. Pavlidis, Q. Li, and W. Stafford Noble. The effect of replication on gene expression microarray experiments. *Bioinformatics*, 19:1620–1627, 2003.

X. Qiu, Y. Xiao, A. Gordon, and A. Yakovlev. Assessing stability of gene selection in microarray data analysis. *BMC Bioinformatics*, pages 7–50, 2007.

Gordon K Smyth. Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3, 2004.

Virginia Goss Tusher, Robert Tibshirani, and Gilbert Chu. Significance analysis of microarrays applied to the ionizing radiation response. *PNAS*, 98:5116–5121, 2001.

Baolin Wu. Differential gene expression using penalized linear regression models: The improved SAM statistic. *Bioinformatics*, 21:1565–1571, 2005.