MAMA: a 9 in 1 R package for Meta-Analysis of MicroArray

Ivana Ihnatova

October 1, 2010

Contents

Part I
Introduction

This paper provides a user guide to R-package MAMA. The package implements nine different methods that have been proposed in meta-analysis of microarray and are designed to identify differentially expressed genes.

In here, we will demonstrate the features of the package with an example of meta-analysis in cancer microarray data, the comparison of expression profiles in MSI (microsatelite instable) and MSS (microsatelite stable) colon cancer. We gathered three microarray data from public databases. The data are stored in object `DataColonHalf`

The guide starts with package and sample data loading.

```
> rm(list = ls(all = TRUE))
> options(width = 60)
> library(MAMA)

gdata: read.xls support for 'XLS' (Excel 97-2004)
gdata: files ENABLED.

gdata: Unable to load perl libaries needed by
gdata: read.xls()
gdata: to support 'XLSX' (Excel 2007+) files.

gdata: Run the function 'installXLSXsupport()'
gdata: to automatically download and install the perl
gdata: libaries needed to support Excel XLS and XLSX
gdata: formats.

> load(url("http://math.muni.cz/~xihnatov/DataColonHalf.RData"))
> ls()

[1] "australia" "denmark"   "japan"
```

The original data sets have been preprocessed and subsampled in order to reduce the computational complexity. All data sets have been normalized and are in log_2-scale. The corresponding sample sizes tor the three datasets (`denmark` [1], `australia` [1] and `japan` [2]) are 77 (39 MSI and 38 MSS), 36 (5 MSI and 31 MSS) and 41 (16 MSI and 25 MSS), respectively. In all expression profiles we have selected the same set of 500 genes for analysis.

Each of the datasets is stored as an ExpressionSet object - a specific container for microarray data and experimental metadata. The detailed information about this object can be found at [3]. Gene expression data matrix can be obtained by function `exprs()` and function `pData()` return a data frame with samples description (class labels).

A different method is used in each of parts below and parts are written to be independed from each other, so you can directly move to method that are of your interest. Meta-analysis usually consist of three steps: Data preparation (and its transformation if necessary), Detection of differentially expressed genes and Extraction and visualization of results.

Part II
Methods that combine p-values

Introduction

In this part we will focus on methods that combine p-values [4], [5]. These methods are inspired by Fisher's S-statistic published in 1925 [6]. We usually obtain two measurements of significance of change in gene expression: value of test-statistic and p-value. These methods combine the p-values from study-specific analysis and combine them into one p-value in sense of sum of logs. Methods differ in test statistic that is used to calculate the study-specific p-value.

Usage

Data preparation

When using this implementation we have to merge all gene expression data matrices (*exprs()*) and class labels vectors (*pData()[,]*) to two lists.

```
> esets <- list(exprs(denmark), exprs(australia),
+     exprs(japan))
> classes <- list(pData(denmark)[, 1], pData(australia)[,
+     1], pData(japan)[, 2])
```

Detecting differentially expressed genes

Functions `pvalcombination` and `pvalcombination.paired` provide meta-analysis based on combination of p-values. The former is designed for unpaired data and the latter for paired design of microarray experiments. Because, our data sets are unpaired, we will use `pvalcombination`. The function requires: a list of gene expression data matrices (`esets`), a list of vectors of class labels (`classes`), type of test statistics (`moderated`) and threshold for significance (`BHth`). It returns list of indices of selected genes. Three possible values for argument `moderated` are available: `"t"` for common t-test, `"limma"` for moderated t-test used in limma package [7] and `"SMVar"` for moderated t-test defined in SMVar package [8].

```
> pvalt <- pvalcombination(esets, classes, moderated = "t",
+     BHth = 0.01)
```

```
    DE    IDD   Loss    IDR    IRR
160.00  43.00  15.00  26.88  11.36
```

Several characteristics which have been defined in meta-analysis of microarray (especially for methods which combine p-values or effect sizes). This characteristics are outprinted by the function. `DE` denotes number of significant genes in meta-analysis. `IDD` represents Integration Driven Discoveries, it means genes which are significant in meta-analysis but not in any of the individual studies alone. Other way round, if a gene is significant only in individual data sets but not in meta-analysis, it is called Integration Driven Revision and `Loss` is a number of such genes. `IDR` and `IRR` are percentages of Integration Driven Discoveries and Integration Driven Revisions in identified differentially expressed genes (`DE`).

Results

```
> summary(pvalt)

              Length Class  Mode
study1        113    -none- numeric
study2          8    -none- numeric
study3         59    -none- numeric
AllIndStudies 132    -none- numeric
Meta          160    -none- numeric
TestStatistic 500    -none- numeric
```

This object is a list with six slots. *Study1* to *Study3* are numeric vectors with indices of differentially expressed genes in data sets 1 to 3. *AllIndStudies* is a vector of indices of differentially expressed genes in at least one data set. Differentially expressed genes found by meta-analysis have their indices stored in *Meta*. And finally, a slot called *TestStatistic* is a vector with test statistics in meta-analysis.

Part III
Methods that combine effect sizes

Introduction

Methods that combine effect size use hierarchical model:

$$y_i = \theta_i + \epsilon_i, \epsilon_i \sim N(0, \sigma_i^2)$$

$$\theta_i = \mu + \delta_i, \delta_i \sim N(0, \tau_i^2),$$

where μ is true difference in mean expression between two classes, y_i denotes the measure effect for study i, with $i = 1, .., k$, τ^2 represents the between study variability, σ_i^2 denotes the within study variability. The analysis is different depending on whether a fixed-effect model (FEM) or a random-effect model (REM) is deemed appropriate. Under a FEM, $\tau = 0$ is assumed, otherwise a REM need to be fit. The estimates of the overall effect μ are different depending on which model is used.

Two papers dealing with effect size combination as method for meta analysis of microarray have been published [4] and [9]. They differ in effect size definition and implementation.

Method presented in [4] offers three variants of effect sizes (classical and moderated T-test) and uses explicitly random-effect model. It is implemented as two functions `EScombination` for unpaired data and `EScombination.paired` for paired data.

On the other hand, in [9] the effect size is defined as Hedge's and Olkin's g and both random-effect and fixed-effect are available. Package *GeneMeta* [10] implements this method.

Algorithm

1. Data recoding.
2. Effect size calculation in each data set.
3. Decision between random-effect model (REM) and fixed-effect model (FEM).
4. Model application.

Usage

Because there are two different ways of implementation for using combination of effect size method on microarray data sets, we will discuss them separately.

Implementation from metaMA package

Data preparation This method requires two lists, one containing the data matrices (*exprs()*) and the other one the corresponding vectors of group labels (*pData()[,]*).

```
> esets <- list(exprs(denmark), exprs(australia),
+     exprs(japan))
> classes <- list(pData(denmark)[, 1], pData(australia)[,
+     1], pData(japan)[, 2])
```

Detecting differentially expressed genes As we have unpaired data, we are going to use function `EScombination`. This function has four arguments: a list of gene expression data matrices (`esets`), a list of class labels vectors (`classes`), effect size definition (`moderated`) and a threshold for false discovery rate (FDR) (`BHth`). Three possible values for `moderated` are available: `"t"` for common t-test, `"limma"` for moderated t-test used in limma package [7] and `"SMVar"` for moderated t-test defined in SMVar package [8].

```
> ESt <- EScombination(esets, classes, moderated = "t",
+     BHth = 0.01)
```

```
    DE    IDD   Loss    IDR    IRR 
109.00  28.00  51.00  25.69  38.64 
```

Function `EScombination` prints several measures defined in meta-analysis of microarray. `DE` denotes number of significant genes in meta-analysis. `IDD` represents Integration Driven Discoveries, it means genes which are significant in meta-analysis but not in any of the individual studies alone. Other way round, if a gene is significant only in individual data sets but not in meta-analysis, it is called Integration Driven Revision and `Loss` is a number of such genes. `IDR` and `IRR` are percentages of Integration Driven Discoveries and Integration Driven Revisions in identified differentially expressed genes (`DE`).

```
> summary(ESt)

               Length Class  Mode
study1         113    -none- numeric
study2           8    -none- numeric
study3          59    -none- numeric
AllIndStudies  132    -none- numeric
Meta           109    -none- numeric
TestStatistic  500    -none- numeric
```

This object is a list with six slots. *Study1* to *Study3* are indices of differentially expressed genes in data sets 1 to 3. *AllIndStudies* is a vector of indices of differentially expressed genes in at least one data set. Differentially expressed genes found by meta-analysis have their indices stored in *Meta*. And finally, a slot called *TestStatistic* is a vector with test statistics (”combined effect size”) in meta-analysis.

Implemenetation from GeneMeta package

Data preparation Before calculating effect sizes we have to create vectors with class labels in form of 1's and 0's. 1 is supposed to be for diseased samples and 0 for normal samples. In data sets used as example in this document 1 refers to MSI samples and 0 to MSS.

```
> ph1 <- pData(denmark)[, 1]
> levels(ph1) <- c(1, 0)
> pData(denmark)[, 1]

 [1]  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI
[12]  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI
[23]  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI  MSI
[34]  MSI  MSI  MSI  MSI  MSI  MSI  MSS  MSS  MSS  MSS  MSS
[45]  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS
[56]  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS
[67]  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS  MSS
Levels:  MSI  MSS

> ph1

 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[29] 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[57] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Levels: 1 0

> ph2 <- pData(australia)[, 1]
> levels(ph2) <- c(1, 0)
> ph3 <- pData(japan)[, 2]
> levels(ph3) <- c(1, 0)
```

ph1, ph2 and ph3 are numeric vectors containing class labels for data sets denmark, australia and japan. These vectors are needed as arguments for functions which provide effect size and its variability estimates.

Detecting differentially expressed genes Functions getdF, dstar and sigmad estimate effect size and its variability for a individual data set, therefore we have to use them three-times. For denmark data set

```
> d.den <- getdF(denmark, ph1)
> d.adj.den <- dstar(d.den, length(ph1))
> var.d.adj.den <- sigmad(d.adj.den, sum(ph1 ==
+     0), sum(ph1 == 1))
> head(d.adj.den)
```

```
[1]  0.4835090 -0.1882238 -0.2740332 -0.4466879 -0.9850491
[6] -1.1694108
```

```
> head(var.d.adj.den)
```

```
[1] 0.05347487 0.05218687 0.05244444 0.05325247 0.05825761
[6] 0.06083683
```

and for other two data sets

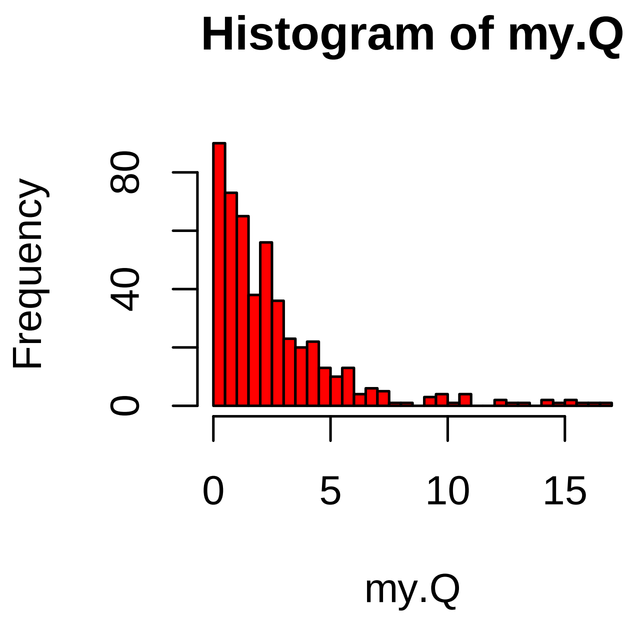
```
> d.aus <- getdF(australia, ph2)
> d.adj.aus <- dstar(d.aus, length(ph2))
> var.d.adj.aus <- sigmad(d.adj.aus, sum(ph2 ==
+     0), sum(ph2 == 1))
> d.jap <- getdF(japan, ph3)
> d.adj.jap <- dstar(d.jap, length(ph3))
> var.d.adj.jap <- sigmad(d.adj.jap, sum(ph3 ==
+     0), sum(ph3 == 1))
```

Function getdF has two arguments: the data set (a ExpressionSet object or a matrix) and class labels (a factor or numeric vector with 1 and 0) and computes estimates of standardized mean difference, found in Hedge and Olkin's [11]. Function dstar corrects the estimates for sample size bias, therefore its second argument is sample size of the data set. Function sigmad calculates the estimate of variance of unbiased effect size. For calculation, the user has to provide effect size estimates and sample size of each class.

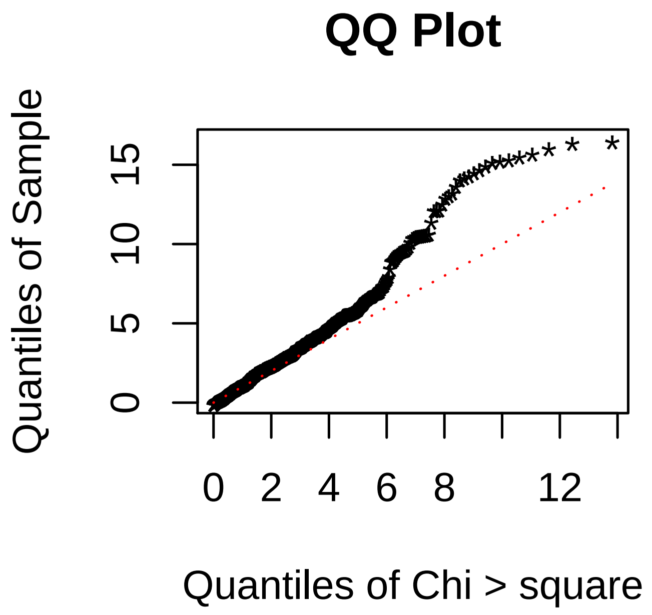
Now, we are going to use Chochran's Q statistic [12] to test between-study variability, so we can decide whether we should be considering random-effect (REM) or fixed-effect model(FEM) for the data.

Function f.Q provides a straightforward calculation of Cochran's Q statistic. If the null hypothesis that the between-study variance is equal to zero (data are well modeled by a fixed effects design) then the estimated Q values will have approximately a chi-squared distribution with degrees of freedom equal to the number of studies minus one. We are going to look at mean and histogram of Q statistics. Later we will compare quantiles of Q to quantiles of chi-square distribution.

```
> mymns <- cbind(d.adj.den, d.adj.aus, d.adj.jap)
> myvars <- cbind(var.d.adj.den, var.d.adj.aus,
+     var.d.adj.jap)
> my.Q <- f.Q(mymns, myvars)
> mean(my.Q)

[1] 2.576469

> hist(my.Q, breaks = 50, col = "red")
```



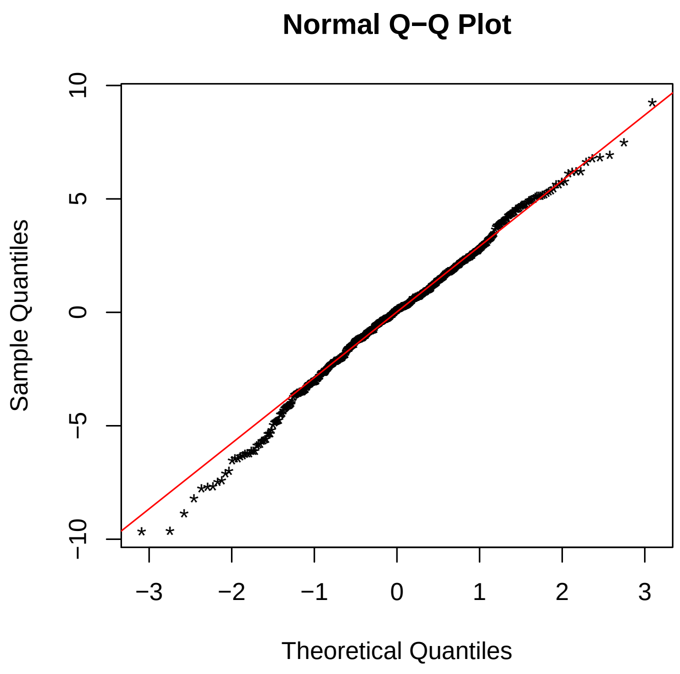
```
> num.studies <- 3
> plotQvsChi(my.Q, num.studies)
```



According to Q-Q plot the hypothesis seems to be valid and fixed-effect model (FEM) should be used. However, we are going to use random-effect model (REM) too, so we can see if there is any difference in estimates of combined effect size.

The computation is simpler for FEM than for REM. Functions `mu.tau2` and `var.tau2` estimate combined effect size (`mu.tau2`) and variance (`var.tau2`). Each effect size is a weighted average of the effects for the individual data sets divided by its standard error. The weights are the reciprocal of the estimated variances.

```
> muFEM = mu.tau2(mymns, myvars)
> sdFEM = var.tau2(myvars)
> ZFEM = muFEM/sqrt(sdFEM)
> qqnorm(ZFEM, pch = "*")
> qqline(ZFEM, col = "red")
```



Plotting the quantiles of the effects we can see that the presumption of approximate Normality seems to be appropriate.

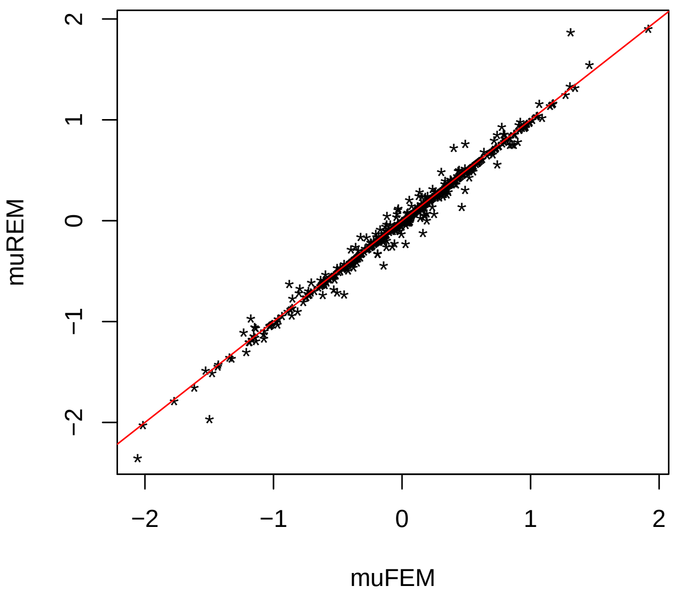
In REM we have to account between-study variability (τ^2). Function `tau2.DL` provides DerSimonian's and Laird's [13] estimates of τ^2 from Cochran's Q. It has two addional arguments: number of studies (`num.studies`) and weights (`my.weights=1/myvars`). We add between-study variability to estimated variance (`myvars`) and calculate the combined effect size like in FEM.

```
> num.studies <- 3
> my.tau2.DL <- tau2.DL(my.Q, num.studies, my.weights = 1/myvars)
> myvarsDL <- myvars + my.tau2.DL
> muREM <- mu.tau2(mymns, myvarsDL)
> varREM <- var.tau2(myvarsDL)
> ZREM <- muREM/sqrt(varREM)
```

`muFEM` or `muREM` are numeric vectors with estimated combined (overall) effect size for a gene in FEM or REM. The estimated standard error of overall effect size for each gene is stored in numeric vectors: `varFEM` or `varREM`. We will test significance of overall effect size by Z-score (`ZFEM` or `ZREM`) defined as mean divided by standard error.

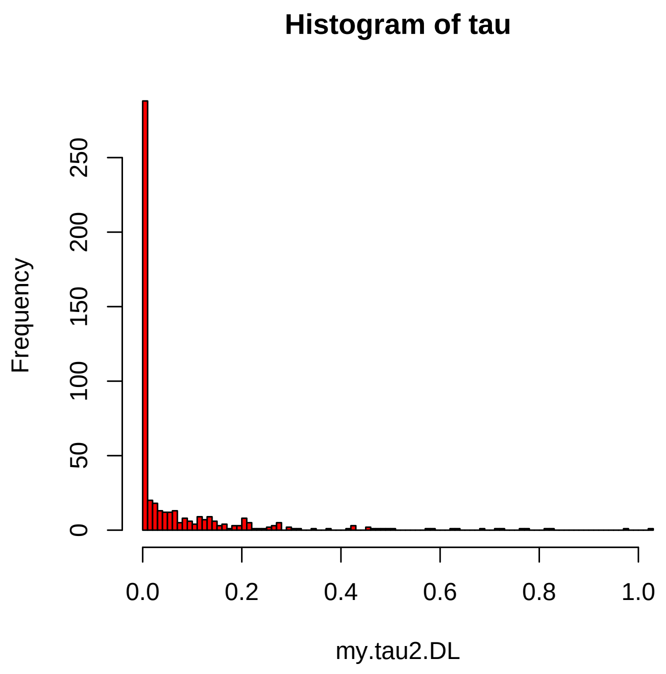
We can easily compare FEM estimates and REM estimates

```
> plot(muFEM, muREM, pch = "*")
> abline(0, 1, col = "red")
```



We do not see much difference here. Actually, for most of the genes the τ^2 is estimated as zero.

```
> hist.tau <- hist(my.tau2.DL, col = "red", breaks = 100,
+     main = "Histogram of tau")
```



Results The procedure described in details above is also implemented in function `zScores`. The arguments of this function are a list of expression sets (`esets`) and a list of classes (`classes`). Argument `useREM` chooses between REM and FEM.

```
> esets <- list(denmark, australia, japan)
> classes <- list(ph1, ph2, ph3)
> theScores <- zScores(esets, classes, useREM = FALSE)
> round(theScores[1:2, ], 3)
```

```
            zSco_Ex_1 zSco_Ex_2 zSco_Ex_3   zSco MUvals
217562_at       2.091    -0.542     0.881  1.865  0.326
203766_s_at    -0.824    -0.085     0.855 -0.196 -0.034
            MUsds Qvals df Qpvalues Chisq Effect_Ex_1
217562_at   0.175 1.964  2    0.375 0.062       0.484
203766_s_at 0.174 1.379  2    0.502 0.845      -0.188
            Effect_Ex_2 Effect_Ex_3 EffectVar_Ex_1
217562_at        -0.262       0.283          0.053
203766_s_at      -0.041       0.275          0.052
            EffectVar_Ex_2 EffectVar_Ex_3
217562_at            0.233          0.103
203766_s_at          0.232          0.103
```

We get a matrix (`theScores`) with the following columns:

- *Effect_Ex_* are the unbiased estimates of the effect (*d.adj.*)

- *EffectVar_Ex_* are the estimated variances of the unbiased effects (*var.d.adj.*)
- *zSco_Ex_* are the unbiased estimates of the effects divided by their standard deviation
- *Qvals* are the Q statistics (*my.Q*) and df is the number of combined experiments minus one
- *MUvals* and *MUsds* are equal to *muFEM* and *sdFEM* (the overall mean effect size and its standard deviation)
- *zSco* are the z scores (*ZFEM*)
- *Qpvalues* is for each gene the probability that a chi-square distribution with *df* degree of freedom has a higher value than its *Q* statistic
- *Chisq* is the probability that a chi-square distribution with 1 degree of freedom has a higher value than *zSco2*

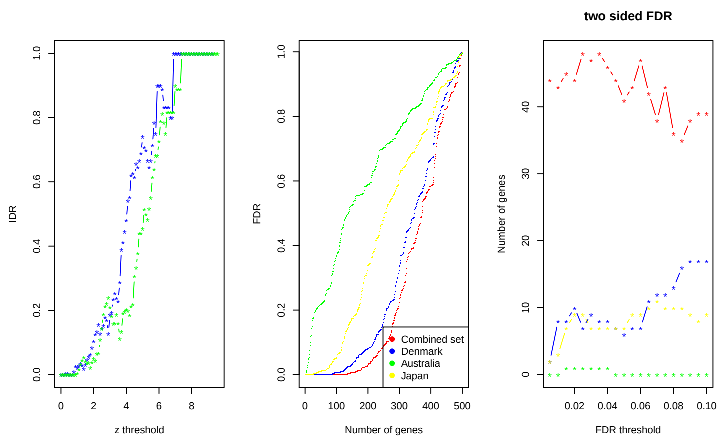
Function `zScoresFDR` implements SAM [?] type analysis to estimate the false discovery rate (FDR).

```
> ScoresFDR <- zScoreFDR(esets, classes, useREM = FALSE,
+     nperm = 50, CombineExp = 1:3)
> names(ScoresFDR)

[1] "pos"       "neg"       "two.sided"

> round(ScoresFDR$pos[1:2, ], 3)

            zSco_Ex_1 FDR_Ex_1 zSco_Ex_2 FDR_Ex_2 zSco_Ex_3
217562_at       2.091    0.084    -0.542    1.083     0.881
203766_s_at    -0.824    1.219    -0.085    1.020     0.855
            FDR_Ex_3   zSco   FDR MUvals MUsds Qvals df
217562_at      0.585  1.865 0.102  0.326 0.175 1.964  2
203766_s_at    0.586 -0.196 1.052 -0.034 0.174 1.379  2
            Qpvalues Chisq
217562_at      0.375 0.062
203766_s_at    0.502 0.845
```

Function `plotES` provides several visualizations of the results. Specifying `which=1` will plot so called *IDRplot*. This plot shows the fraction of the genes that have a higher effect size than the threshold for the combined Z-score , but not for any of the data set specific Z-scores. Genes with combined Z-score > 0 and < 0 are plotted separately. Selection `which=2` will plot the number of genes and the corresponding FDR for the two sided situation. If the user is more interested in the number of genes that are below a given threshold for the FDR, he decides for `which=3`. It shows for each study (indicated by different colors) and various thresholds for the FDR (x axis) the number of genes that are below this threshold in the given study but above in all other studies are shown (y axis). If numeric vector is used that all figure specified in the vectors are plot.

Argument `legend.names` is a character vector with names of the date set used in legends and `colors` is a vector of colors to be used for plotting.

```
> plotES(theScores, ScoresFDR, num.studies = 3,
+     legend.names = c("Combined set", "Denmark",
+         "Australia", "Japan"), colors = c("red",
+         "blue", "green", "yellow"), which = 1:3)
```



Part IV
Similarity of Ordered Gene Lists (SOGL)

Introduction

Similarity of Ordered Gene Lists is another method for meta-analysis of microarray. It is call as ”comparison of comparisons” by its authors [?].

Briefly, it assigns a similarity score to a comparison of two ranked (ordered) gene lists. The score is based on the number of overlapping genes in the top ranks. It computes the size of overlap for each rank. The final score is a weighted sum of these values, with more weight put on the top ranks.

Algorithm

1. Required data sets - two data sets with same set of genes (or genes which can be mapped to each other) are required.

2. Ranking of genes - The genes are then ranked based on gene-wise test on difference of class mean. There is only one assumption about test result: a large positive test score corresponds to up-regulation and a large negative value to down-regulation.

3. Computing the overlap - for each rank (from 1 to number of genes) we count the number of genes that appear in both ordered lists up to that

position. It is denoted as $O_n(G_A, G_B)$, where G_A and G_B refer to ordered gene lists.

4. Preliminary similarity score - First we compute a total overlap A_n at position n given as $O_n(G_A, G_B) + O_n(f(G_A), f(G_B))$, where $f()$ means flipped list (down-regulated genes on top). Later we add weights ($w_\alpha = e^{-\alpha.n}$)to it and we sum it up to preliminary score. Parameter α is needed to tune the weights: a smaller α puts more weight on genes Further down the list. Implementation can choose an appropriate α itself.

5. Final similarity score - it takes two possibilities into account. The possibilities are: the class labels of the two data sets match or do not match.

The algorithm above is valid for meta-analysis in which expression data are also available. However, we can analyze only two ordered gene list without expression data. It has two peculiarities: we can not use same approach for calculating the significance of overlap and we can not be sure if genes are ranked from the most up-regulated to the most down-regulated. Please see [14] for more details.

Usage

Data preparation

We will use only first two datasets (`denmark` and `australia`) and they need to be processed by function `dataSOGL`, so they can be merged into one "ExpressionSet" object with function `prepareData`. Function `dataSOGL` requires a ExpressionSet object (`data`), a column number for *pData* to be used as class labels (`group`), a name for class labels (`groupname`) and microarray platform for *annotation* (`annotation`). Function `prepareData` has three aguments: `eset1`, `eset2` and `mapping`. `eset1` and `eset2` are lists consisting of: a data set as a ExpressionSet object (`data`), name of the data set (`name`), name of the class labels (`var`), numeric vector of class labels used in data set (`out`) and a indicator whether paired data are present (`paired`). `mapping` is a two column data frame with probe IDs of `eset1` and `eset2`. 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.

```
> denmarkSOGL <- dataSOGL(data = denmark, group = 1,
+     groupname = "satelite", annotation = "hgu133plus2")
> australiaSOGL <- dataSOGL(data = australia, group = 1,
+     groupname = "satelite", annotation = "hgu133plus2")
> A <- prepareData(eset1 = list(data = denmarkSOGL,
+     name = "colon_cancer1", var = "groupname",
+     out = c(1, 2), paired = FALSE), eset2 = list(data = australiaSOGL,
+     name = "colon_cancer2", var = "groupname",
+     out = c(1, 2), paired = FALSE), mapping = NULL)
```

Detecting differentially expressed genes

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. The final list of overlapping genes consists of those probes that contribute a certain percentage to the overall similarity score. We can choose three different statistics for gene ranking: t-test with equal variances, log ratio (log fold change) or Z-score (chosen explicitly, t-test with regularized variances). We apply function `OrderedList` with default values to our combined data set.

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

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

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

> x.z

Similarity of Ordered Gene Lists
 Comparison          : colon_cancer1~colon_cancer2
 Number of genes     : 500
 Test statistic      : z
 Number of subsamples: 1000
 beta = 1 -> corresponding labels could be matched in different studies
--------------------------------------
 Optimal regularization parameter: alpha = 0.02878231
 Lists are more alike in direct order
 Weighted overlap score:  1137.552
 Significance of similarity: p-value = 0.000999001
 Number of genes contributing 95 % to similarity score: 231
```

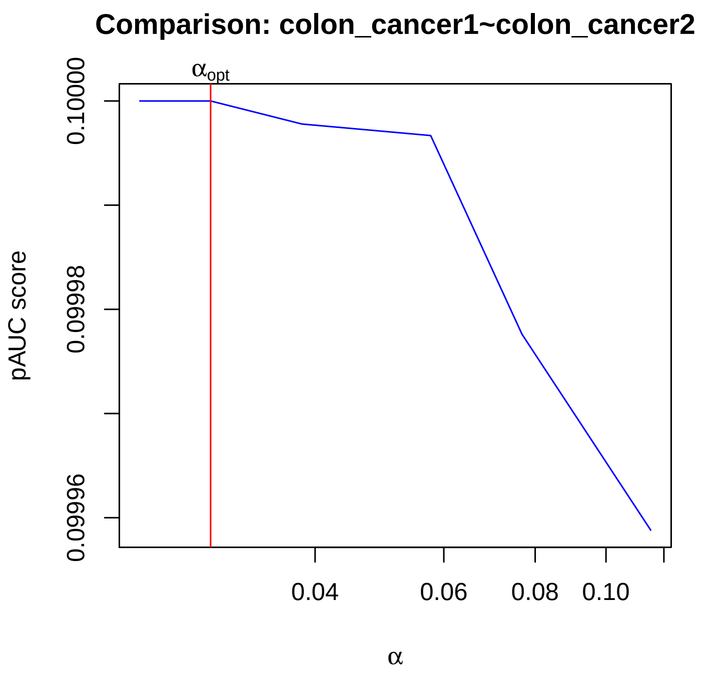
Results

The result is an object of class OrderedList for which print and plot function exist. Output from print function can be seen above and plot function is used below. The sorted list of overlapping genes is stored in `$intersect`.

```
> x.z$intersect[1:12]

 [1] "1552281_at"   "1552365_at"   "1552485_at"
 [4] "1552621_at"   "1552680_a_at" "1553033_at"
 [7] "1553986_at"   "1554394_at"   "1554508_at"
[10] "1554999_at"   "1555086_at"   "1556055_at"
```

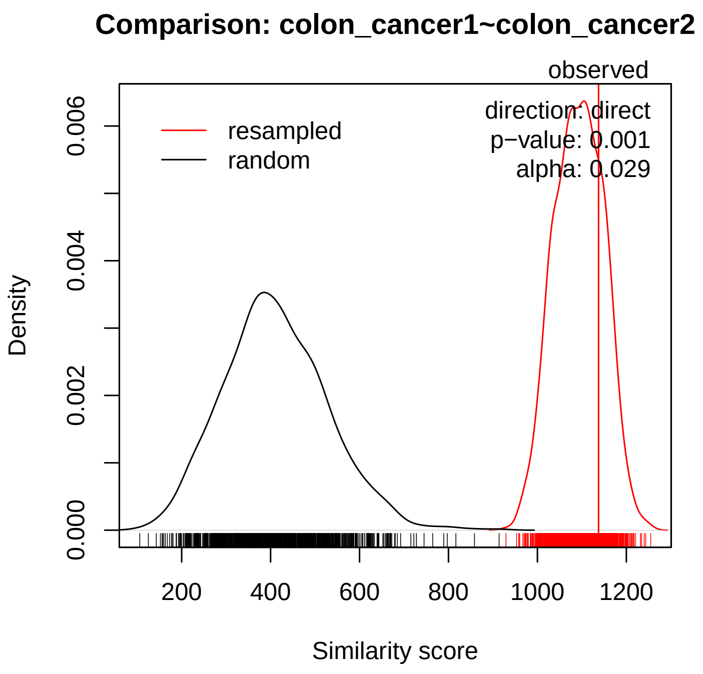
Calling `OrderedList` with the `empirical` option set to `true`, causes OrderedList to compute empirical bounds for expected overlaps. By default, this is switched off and underestimated bounds deduced from a hypergeometric distribution are used.

```
> plot(x.z, "pauc")
```



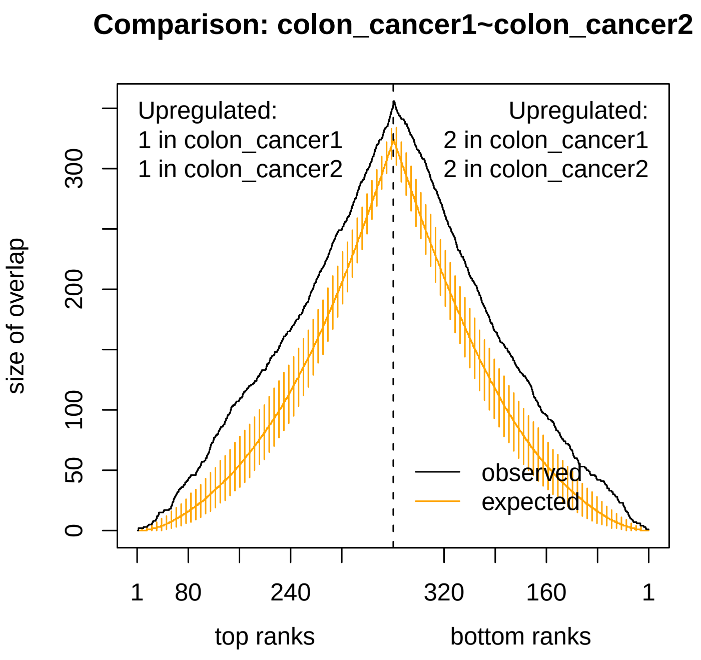
This is a plot of pAUC scores based on α selection. The separability between the two distributions of observed and random similarity score is measured by pAUC score. α is chosen where the pAUC score is maximal. It is marked by a vertical line.

```
> plot(x.z, "scores")
```



The red curve correspondence to kernel density estimate of simulated observed scores and the black curve to kernel density of simulated random scores. The actually observed similarity score is denoted by the vertical red line. The bottom rugs mark the simulated values.

```
> plot(x.z, "overlap")
```



This plot displays the numbers of overlapping genes in the two gene lists. It is drawn as step function over the respective ranks. Top ranks correspond to up-regulated and bottom ranks to down-regulated genes. The expected overlap and confidence intervals are shown too. They are derived empirically form the subsampling or deduced from a hypergeometric distribution, it depends on parameter `empirical`.

Notes

We can also compare directly two ordered gene lis via functions: `CompareLists` and `getOverlap`. Please see [14] for details.

Part V
RankProduct

Introduction

RankProduct is a non-parametric statistic that detects up-regulated and down-regulated genes under one condition against another condition. In our sample data set we look for difference in expression between MSI and MSS colon cancer.

It focuses on genes which are consistently highly ranked in a number of lists, for example genes that are regularly found among top up-regulated genes in many microarray studies. It assumes that under the null hypothesis that the order of all items is random then the probability of finding a certain item among the top r of n items in a list is $p = r/n$. Rank product is defined by multiplying these probabilities $RP = \prod_i \frac{r_i}{n_i}$, where r_i is the rank of the item in the i-th list and n_i is the total number of the items on i-th list. The smaller the RP value the smaller the probability that the observation of the item at the top of the lists is due to chance. It is equivalent to calculating the geometric mean rank. A list of up- or down-regulated genes are selected based on the estimated percentage of false positive prediction (pfp), it is known as false discovery rate (FDR), too.

Algorithm

Algorithm of the method has five steps:

1. Fold-change ratio is calculated in each data set.
2. Ranks are assigned (1 for the highest value) according to fold-change ratio. r_{gi} is rank of gene g in comparison i, where i is from 1 to K, where K is sum of products of number of slides in groups.
3. RankProduct for a gene (RP_g) is calculated as $\prod_i r_{gi}^{1/K}$
4. l permutations of expression values at each microarray slide is performed and all previous steps repeated. We obtain $RP_g^{(l)}$
5. Step 4 is repeated L times to estimate the distribution of $RP_g^{(l)}$. This distribution is used to calculate p-value and pfp for each gene.

Usage

Data preparation

In order to run a rank product meta-analysis, users need to call function `RPadvance`. They both require three arguments: `data`, `cl` and `origin`. The first required argument, `data`, is the matrix (or data frame) containing the gene expression data that should be analyzed. Each of its rows corresponds to a gene, and each column corresponds to a sample. Second and third argument, `cl` and `origin`, are vectors of length `ncol(data)` containing the class labels of the samples or the origin labels of the samples. Function `mergedata` returns a list with three slots corresponding to arguments described above. `class.col` argument is a numeric vector indicating which columns of *pData* should be used as class labels. First number refers to first data set etc.

```
> rankdata <- mergedata(denmark, australia, japan,
+     class.col = c(1, 1, 2))
> rankdata$cl
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[28] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[55] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1
[82] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
[109] 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
[136] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
> rankdata$origin
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [28] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [55] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 [82] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[109] 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[136] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

In `cl` all 1's refer to MSI samples and all 2's to MSS samples. Similarly in `origin`, 1 belongs to samples from first data set (`denmark`), 2 from second data set (`australia`) and 3 from `japan` study. You can choose different numbers for labels, but same numbers are always treated like same samples from same class or with same origin.

Detecting differentially expressed genes

In this section, we show how the rank product method can be applied to detect differentially expressed gene in our data sets in sence of meta-analysis. It means we will get two separate lists (up- and down-regulated genes separately) not two such lists for each data set. For each gene, one pfp (percentage of false prediction) is computed and used to select significant genes. We can run meta-analysis by

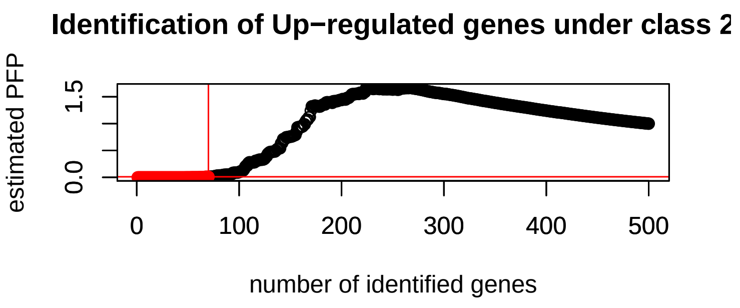
```
> RP.out <- RPadvance(rankdata$dat, rankdata$cl,
+     rankdata$origin, num.perm = 50, logged = TRUE,
+     na.rm = FALSE, gene.names = rownames(exprs(denmark)),
+     plot = FALSE)
```

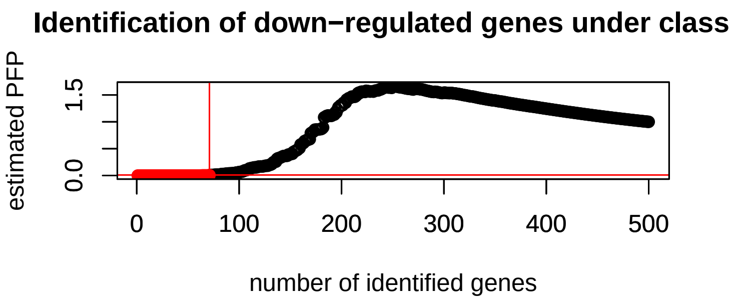
```
 The data is from  3 different origins

Rank Product analysis for two-class case

Warning: Expected classlabels are 0 and 1. cl will thus be set to 0 and 1.

Starting  50 permutations...
Computing pfp...
```

The data are log-transformed, therefore we set `logged=TRUE`. The number of permutations is default set to 100, you can change it to higher number, if you wish more precise estimates of the pfp. The argument `plot=FALSE` will prevent the graphical display of the estimated pfp vs. number of identified genes. We will use function `plotRP` for a such display.

Results

```
> plotRP(RP.out, cutoff = 0.01)
```





The function `plotRP` graphicaly displays the estimated pfp vs. number of identified genes using the output from `RPadvance`. If cutoff (the maximum accepted pfp) is specified, identified genes are marked in red.

```
> RankRes <- topGene(RP.out, cutoff = 0.01)

Table1: Genes called significant under class1 < class2

Table2: Genes called significant under class1 > class2

> head(round(RankRes$Table1, 3))

            gene.index RP/Rsum FC:(class1/class2) pfp
228030_at          254  15.955              0.234   0
228915_at          462  26.810              0.407   0
206239_s_at         77  28.268              0.344   0
243669_s_at        237  30.786              0.465   0
213880_at          258  40.144              0.520   0
213385_at          213  43.146              0.456   0
            P.value
228030_at         0
228915_at         0
206239_s_at       0
243669_s_at       0
213880_at         0
213385_at         0

> head(round(RankRes$Table2, 3))
```

```
          gene.index RP/Rsum FC:(class1/class2) pfp P.value
205242_at        257  31.645              3.092   0       0
37145_at         154  34.843              2.678   0       0
209301_at        164  37.957              2.276   0       0
206442_at        280  42.928              2.927   0       0
206391_at        168  49.136              1.836   0       0
204818_at        277  50.370              2.216   0       0
```

The function `topGene` is used to output a table of the identified genes from the output object from function `RPadvance`. Table contains genes according to other arguments. It is obligatory to specify either the `cutoff` (the desired significance of the identification) or `num.gene` (the number of top genes identified), otherwise a error message will be printed and the function will be stopped. If cutoff is selected, user needs to choose between `pfp` (percentage of false prediction) or `pval` (p-value). `pfp` is the default setting, which is selected when no selection is made.

Two tables are output, listing identified up- (Table1: class 1 $<$ class 2) and down- (Table2: class1 $>$ class 2) regulated genes. There are 5 columns in the table

1. *gene.index* is the gene index in the original data set
2. *RP/Rsum* is the computed rank product for each gene
3. *FC:(class1/class2)* is the computed fold change of the average expression levels under two conditions, which would be converted to the original scale using input logbase (default value is 2) if `logged=TRUE` is specified
4. *pfp* is the estimated pfp value for each gene in the list if that gene serves as the cutoff point
5. *P.value* is the associated P-values for each gene

Notes

By combining data sets from different origins together, the test gets increased power, which leads to more identified genes. For more information see also [15].

Part VI
Z-statistic - posterior mean differential expression

Introduction

The main idea of this method is that one can use data from one study to construct a prior distribution of differential expression and thus utilize the posterior mean differential expression, weighted by variances, whose distribution is standard normal distribution due to classic Bayesian probability calculation.

It is based on assumption that gene expression is normally distributed with mean μ_g and SD σ_g^2 and that we can estimate σ_g^2 by pooling together all genes with similar levels of mean intensity. The difference in gene expression is tested by

$$Z = \frac{D}{\sigma_D} = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \sim N(0,1),$$

where $\overline{X}_1$ and $\overline{X}_2$ denotes mean gene expression values in classes, σ_1^2 and σ_2^2 denotes the estimated SD in classes and n_1 and n_2 denotes the number of samples in classes.

Usage

Data preparation

Because the same number of samples in each class and study is used in primary publication of the method [16], we will first look at number of samples in our data.

```
> table(pData(denmark)[, 1])

 MSI  MSS
  39   38

> table(pData(australia)[, 1])

 MSI  MSS
   5   31

> table(pData(japan)[, 2])

MSI MSS
 16  25
```

The smallest value in the tables above is 5, therefore we will randomly choose 5 samples in each class and data set. Function `dataZ` performs such data reduction. It has four required arguments: a data set as ExpressionSet object (`data`), number of column of *pData* slot with class labels (`group`), number of samples to be selected (`nsamp`) and name for class labels (`varname`). We need to merge the data sets into one `mergeExprSet` object created by function `mergeExprs` from R package *MergeMaid*.

```
> denmarkZ <- dataZ(data = denmark, group = 1, nsamp = 5,
+     varname = "satelite")
> australiaZ <- dataZ(australia, 1, 5, "satelite")
> japanZ <- dataZ(japan, 2, 5, "satelite")
> library(MergeMaid)
> merged <- mergeExprs(denmarkZ, australiaZ, japanZ)
```

Now, we can proceed to detecting differentially expressed genes.

Detecting differentially expressed genes

We apply this method by

```
> z.stat <- Zscore(merged)
```

```
Pheno data is assumed to be in the first column of phenoData slot
0 marked as 0
1 marked as 1
Contrast will be 1 - 0
```

Results

```
> head(round(z.stat, 3))
```

```
             Zscore Pvalue
1552281_at    4.609  0.000
1552365_at   -7.066  0.000
1552485_at   -3.123  0.002
1552502_s_at -1.798  0.072
1552546_a_at -0.088  0.930
1552553_a_at -1.512  0.131
```

Only values of Z-statistic (`Zscore`) and their p-values (`Pvalue`) are provided by function `Zscore`.

Notes and discussion

This implementation expects either same microarray platform or same scale of expression values (like after POE transformation [17]) in all data sets.

Part VII
TSP-clasiffier

Introduction

This method has been originally described in [18]. A top scoring pair (TSP) is a pair of genes whose relative ranks can be used to classify arrays according to a binary phenotype. A top scoring pair classifier has three advantages over standard classifiers:

1. the classifier is based on the relative ranks of genes and is more robust to normalization and preprocessing,
2. the classifier is based on a pair of genes and is likely to be more interpretable than a more complicated classifier,
3. a classifier based on a small number of genes lends itself diagnostic tests based on PCR that are both more rapid and cheaper than classifiers based on a large number of genes.

Usage

In this section we will demonstrate the use of the functions made for meta-analysis of example data sets. We will show how to calculate top scoring pair, how to calculate p-values for significance and how to plot TSP objects.

Data preparation

We are going to use function `mergedata` again. Please see Data preparation section of RankProduct part for details.

```
> tspdata <- mergedata(denmark, australia, japan,
+     class.col = c(1, 1, 2))
```

Detecting differentially expressed genes

Function `tspcalc` calculates top scoring gene pair. It has two arguments: `dat` and `grp`. `dat` can be either an m genes by n samples matrix of expression data or an ExpressionSet object. There are also two posibilities for `grp`: A group indicator in character or numeric form or an integer indicating the column of *pData()* to use as the group indicator. We use gene expression data matrix and vector of numeric class labels.

```
> tsp <- tspcalc(dat = tspdata$dat, grp = tspdata$cl)
```

We can compute the significance of a top scoring pair, too. It calculates "how strong a top scoring pair is".

The function `tspsig` performs a permutation test with the null hypothesis that no TSP exists in the data set. It permutes the group labels B times and calculates a null TSP score for each time. The p-value is then the total number of null TSP scores that exceed the observed TSP score plus 1 divided by $B + 1$. A progress bar indicates the time left in the calculation. You have to again specify the data expression matrix, class labels and additionally the number of permutations. You can also set the seed for permutations to make results reproducible.

```
> out <- tspsig(tspdata$dat, tspdata$cl, B = 50)
```

```
          |2%      |20%      |40%      |60%      |80%      |100%
Progress: ||||||||||||||||||||||||||||||||||||||||||||||||||
```

Results

Function `tspcalc` returns a tsp object.

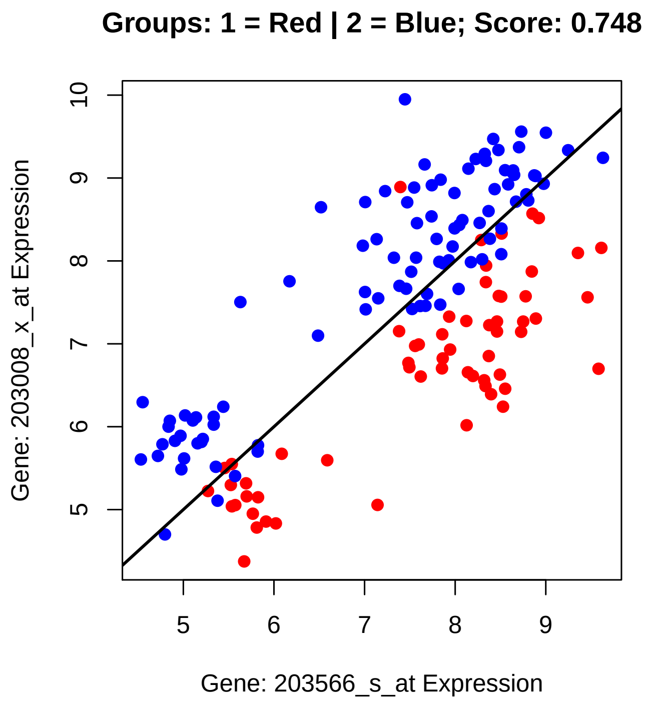
```
> tsp
```

```
tsp object with: 1 TSPs
Pair:                TSP Score         Tie-Breaker                Indices
TSP 1 :          0.75                NA                                   243 415
```

In the output above each row refers to one top scoring pair. *TSP Score* is TSP score as defined in [18], essentially it is the empirical average of sensitivity and specificity for the pair. *Tie-Breaker* denotes the tie-breaking score described in [19]. Briefly, each expression value is ranked within its array, then a rank difference score is calculated for each pair of genes. Finally, *Indices* gives the rows of the gene expression matrix that define a top scoring pair.

```
> tspplot(tsp)
```

```
Number of TSPs:  1
TSP 1
```



The `tspplot` accepts a tsp object and returns a TSP plot. The figure plots the expression for the first gene in the TSP pair versus the expression for the second gene in the TSP pair across arrays. The user defined groups are plotted in the colors red and blue. The score for the pair is shown across the top of each plot. If there is more than one TSP, hitting return will cycle from one TSP to the next.

```
> summary(out)
```

```
           Length Class  Mode
p           1     -none- numeric
nullscores 50     -none- numeric
```

```
> out$p
```

```
[1] 0.01960784
```

```
> out$nullscores

 [1] 0.3248227 0.3446809 0.3177305 0.3159574 0.3645390
 [6] 0.3017730 0.3475177 0.3503546 0.3060284 0.3234043
[11] 0.2783688 0.3609929 0.3265957 0.3095745 0.3152482
[16] 0.3386525 0.3081560 0.3138298 0.3336879 0.3202128
[21] 0.3351064 0.3113475 0.3010638 0.3035461 0.2872340
[26] 0.3000000 0.3248227 0.3166667 0.4156028 0.2897163
[31] 0.3851064 0.3521277 0.3078014 0.3687943 0.3113475
[36] 0.3049645 0.3563830 0.3095745 0.3124113 0.3010638
[41] 0.2957447 0.2769504 0.3347518 0.3120567 0.3290780
[46] 0.3340426 0.3900709 0.2893617 0.3659574 0.3173759
```

p and *nullscores* are two the most interesting elements of output from `tspsig` function. The former is the significance of TSP and the latter contains top scores observed in permutations.

Part VIII
VennMapping

Introduction

VennMapping [20] is a method based on Venn diagrams and contingency tables. It looks for number of common genes in pairs of gene lists, statistical significance of observed match and returns also names of the common genes.

Algorithm

Algorithm of this method consists of three steps:

1. Calculation of fold-change in each data set.
2. Selection of significant (interesting) genes.
3. Comparison of gene lists pairs.

Usage

Data preparation

Function `fold.change` calculates mean fold-change in one data set. It has two arguments: data set (e.g. `denmark`) and column number of *pData* slot with class labels to be used. It assumes data are on log_2 scale.

```
> fc.d <- fold.change(denmark, 1)
> fc.a <- fold.change(australia, 1)
> fc.j <- fold.change(japan, 2)
> FC <- cbind(fc.d, fc.a, fc.j)
```

Function `gene.select` selects significant/interesting genes from mean fold-change matrix with rows referring to genes and columns to data sets. The user has to specify (apart from mean fold-change matrix) a cutoff for selection. The cutoff is on log_2 scale, too. We chose 1 for genes with at least 2-fold change in expression.

```
> list <- gene.select.FC(FC, 1)
> summary(list)

     Length Class  Mode
fc.d 33     -none- character
fc.a 27     -none- character
fc.j 35     -none- character
```

Object `list` is a list in which each slot contains names of selected genes in one study. For example from the print above 33 genes have been selected in `denmark` data set.

Detecting differentially expressed genes

Now, we can move on comparison of selected gene lists in pairs of data sets. There are three functions to perform such a analysis: `conting.tab`, `Z` and `gene.list`. `conting.tab` returns contingency table with number of common genes. `Z` provides Z statistic to measure significance of observed number of common genes and `gene.list` outputs table with names of common genes. All of them have one argument same - it is a list object with names of selected genes in individual data sets. For function `Z` one additional argument is necessary - the number of genes involved in meta-analysis (calculated by `length(rownames(exprs(denmark)))`).

```
> conting.tab(list)

     fc.d fc.a fc.j
fc.d   NA   12   16
fc.a   12   NA    7
fc.j   16    7   NA

> Z(list, n = length(rownames(exprs(denmark))))

         fc.d     fc.a     fc.j
fc.d       NA 7.920245 9.320174
fc.a 7.869850       NA 3.821593
fc.j 9.340196 3.854327       NA

> gene.list(list)

     fc.d
fc.d NA
fc.a "205009_at;206239_s_at;37145_at;205044_at;213385_at;228030_at;205242_at;204818_at;206
fc.j "202803_s_at;230964_at;213915_at;206239_s_at;1556055_at;1552281_at;37145_at;209301_at
     fc.a
fc.d "205009_at;206239_s_at;37145_at;205044_at;213385_at;228030_at;205242_at;204818_at;206
fc.a NA
fc.j "206239_s_at;209583_s_at;37145_at;228030_at;206442_at;210143_at;230793_at"
     fc.j
fc.d "202803_s_at;230964_at;213915_at;206239_s_at;1556055_at;1552281_at;37145_at;209301_at
fc.a "206239_s_at;209583_s_at;37145_at;228030_at;206442_at;210143_at;230793_at"
fc.j NA
```

Part IX
MAP-Matches

Introduction

Meta-Analysis Pattern Matches (MAP-Matches) [21] is a method that extends VennMapping [20] and meta-profiling [22]. It is designed to analyze more distinct microarray data (search for common molecular mechanism in all types of cancer). It assumes same gene set in all data sets.

Algorithm

Algorithm of this method has five steps:

1. Calculation of T-statistic for each two classes in each data set.
2. Building matrix of T-statistics (T-matrix) with rows referring to genes and columns to pairs of classes and data set.
3. Selection of threshold for T-statistic.
4. Transformation of T-matrix into a binary matrix: 1 for T-statistics above threshold, 0 for T-statistics below threshold.
5. Statistical analysis of transformed T-matrix (more details in Usage section).

Usage

Data preparation

The analysis starts with calculation of T-statistics. Function `meta.test` returns a list with two slots: matrix of test statistics (`test`) and matrix of p-values (`p`). In each of the matrices rows correspond to genes and columns to data sets. We need only `test` slot for this method. Argument `class.col` is a numeric vector indicating which column of *pData* should be used and `data.names` is a character vector with names of the data sets.

```
> stat.real <- meta.test(denmark, australia, japan,
+     class.col = c(1, 1, 2), data.names = c("denmark",
+         "australia", "japan"))$test
> colnames(stat.real) <- c("Denmark", "Australia",
+     "Japan")
```

Detecting differentially expressed genes

The do not select significant genes in each study we only set threshold for T-statistics. We decided for 98 % quantile (same in [21]).

```
> stat <- c(stat.real)
> quan <- T.select(stat)
> T.default <- quan["98.00%"]
```

Now, we transform `stat.real` (T-matrix) into a binary matrix. We replace T-statistics above threshold with 1 and below with 0.

```
> value.dis <- apply(stat.real, MARGIN = c(1, 2),
+     function(x) ifelse(abs(x) > T.default, 1,
+         0))
> rownames(value.dis) <- featureNames(denmark)
> head(value.dis)
```

```
             Denmark Australia Japan
217562_at          0         0     0
203766_s_at        0         0     0
1554394_at         0         0     0
212662_at          0         0     0
1555370_a_at       0         0     0
240574_at          0         0     0
```

Each row `value.dis` is called a meta-analysis pattern. We are going to analyze their occurrence, significance and genes they occur at. Function `ratio` provides basic summarization of `value.dis`.

```
> results <- ratio(value.dis)
> summary(results)
```

```
         Length Class  Mode
n         3     -none- numeric
X.string 23     -none- character
p.strong  6     -none- numeric
p.soft    6     -none- numeric
```

In `results` we can find: number of genes with T-statistic sufficiently high in each study *n*, patterns observed in data (*X.String*), probability of observing strong match (*p.strong*) and probability of observing soft match (*p.soft*). We say two patterns match strongly if they are equal. The rule for soft match is weaker as only 1's in patterns must match.

Function `MAPmatrix` calculates a matrix with rows corresponding to patterns and four columns: unique patterns that are being observed in our data (*uniqe.pat*), number of observed soft matches with the pattern (*n.soft*), number of observed strong matches (*n.strong* and number of 1's in the pattern *n.sig*).

```
> MAPmat <- MAPmatrix(value.dis)
> MAPmat
```

```
    unique.pat n.soft n.strong n.sig
100        100     18       12     1
010        010      4        2     1
001        001      8        3     1
101        101      5        4     2
111        111      1        1     3
110        110      2        1     2
```

Only pattern with multiply 1's are connected with common molecular mechanism and we will focus on them in the rest of analysis.

```
> MAPmat2 <- MAPmat[MAPmat$n.sig > 1, ]
> unique.pat <- as.character(MAPmat2[, 1])
```

We assume that sufficiently high number of strong matches may provides evidence of common molecular mechanism. Functions `MAPsig1` and `MAPsig2` perform statistical analysis to answer whether we observe significant number of matches or not. The statistical analysis can be done in two ways (both based on permutation testing): we either permute columns of T-matrix (in binary form) or permute class labels in data sets and repeat the whole procedure with same threshold for T-statistics. The former is implemented in `MAPsig1` and the latter in `MAPsig2`. Function `test.group.shuffle` calculates T-statistics with permuted class label repeatedly.

```
> p1 <- MAPsig1(unique.pat, value.dis, iter = 1000)
```

```
1          2          3          4          5          6          7          8          9
```

```
> p1
```

```
  p.soft p.strong
1  0.000    0.000
2  0.001    0.001
3  0.012    0.150
```

```
> den.shuf <- test.group.shuffle(data = denmark,
+     dataname = "Denmark")
> aus.shuf <- test.group.shuffle(data = australia,
+     dataname = "Australia")
> jap.shuf <- test.group.shuffle(data = japan, dataname = "Japan",
+     var = 2)
> dataset <- c("Denmark", "Australia", "Japan")
> p2 <- MAPsig2(dataset, value.dis, unique.pat,
+     B = 100)
```

```
> p2
```

```
  permu.soft permu.strong
1          0          0.0
2          0          0.0
3          0          0.1
```

Both *p1* and *p2* have same structure: rows refer to patterns and columns to statistical signifficance of observed soft (*p.soft* or *permu.soft*) or strong (*p.strong* or *permu.strong*) matches.

Results

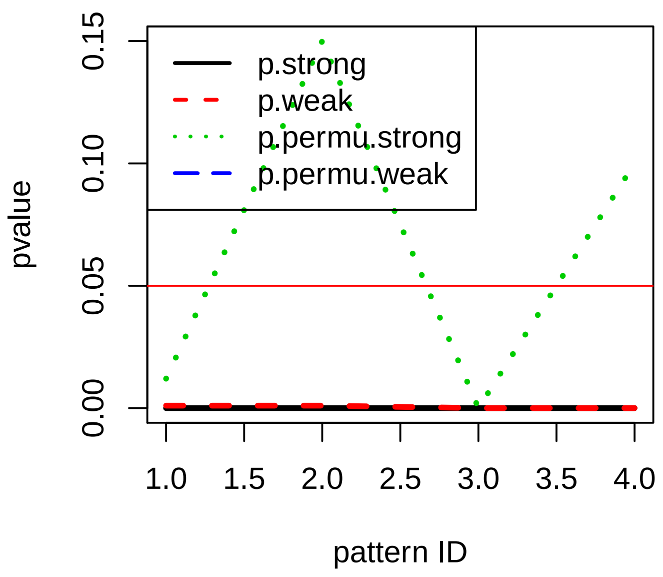
Finally, we will bind all necessary outputs together.

```
> resx <- cbind(MAPmat2, p1, p2)
> colnames(resx) <- c(colnames(MAPmat2), "p.strong",
+     "p.weak", "p.permu.strong", "p.permu.weak")
> intx <- t(as.matrix(resx[which(resx[, 4] < 0.06),
+     ]))
> t(resx)
```

```
               101     111     110
unique.pat     "101"   "111"   "110"
n.soft         "5"     "1"     "2"
n.strong       "4"     "1"     "1"
n.sig          "2"     "3"     "2"
p.strong       "0.000" "0.001" "0.012"
p.weak         "0.000" "0.001" "0.150"
p.permu.strong "0"     "0"     "0"
p.permu.weak   "0.0"   "0.0"   "0.1"
```

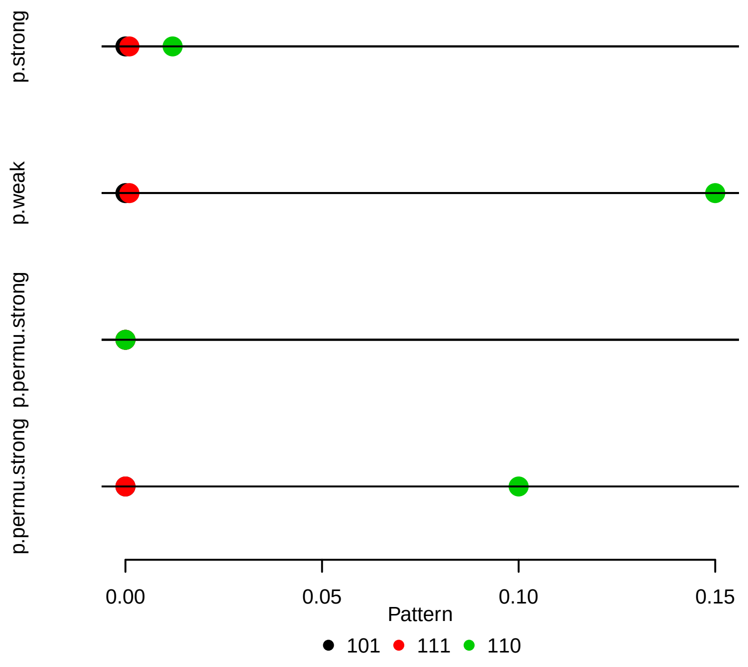
We can plot p-values by

```
> plotpattern(resx, method = 1)
```



or

```
> plotpattern(resx, method = 2)
```



Until now, we have only found out that for some patterns there is significantly high count of strong or soft matches being observed. Obviously we want to know expression of which genes is changed in these patterns. Function `MAP.genes` returns a list in which each slot contains list of genes involved in one pattern. If argument `files` is set to `TRUE` a files with gene names are also saved.

```
> probs <- MAP.genes(resx, value.dis, files = FALSE)
> names(probs) <- rownames(resx)
> summary(probs)

    Length Class  Mode
101 5      -none- character
111 1      -none- character
110 2      -none- character
```

probs is a list with each slot referring to one pattern and list of gene names is stored there. The pattern has been observed at these genes.

If there is annotation package available for microarray platform used in meta-analysis we can create a HTML annotation of found patterns by

```
> library(annaffy)
> library(hgu133plus2.db)
> MAP.HTMLanno(resx, probs, "hgu133plus2.db")
```

Part X METRADISC

Introduction

METRADISC [23] is unique among rank-based methods (like Rank Product or TSP) because it provides an estimate of heterogeneity as one of its outputs. Additionally the method can deal with genes which are being measured in only some of the studies. The implementation available in MAMA package is restricted to genes common in all microarray studies analyzed.

Algorithm

1. Gene Ranking - In microarray analysis we usually test samples for a large number of genes. The results provide for each gene a test statistic and its statistical significance (p-value). Therefore we can rank the tested genes in each study based on direction in expression change and statistical significance. If there are n genes being tested, the highest rank n is given to the gene that shows the lowest p-value and it is up-regulated in diseased samples. Then follow all other up-regulated genes ranked according to increasing p-value. These are followed by down-regulated genes and the lowest rank (1) is given to gene that shows the lowest p-value and is down-regulated in diseased samples. Genes with equal p-values are assigned tied ranks.

2. The Average Rank and Heterogeneity metrics - In this step we compute a average rank and heterogeneity metrics. The average rank R^* is defined as $R^* = \frac{\sum_{i=1}^{s} R_i}{s}$, where R_i is the rank of the gene in study i and s is total number of studies ($i = 1, 2, ..., s$). The heterogeneity metrics Q^* is given by formula $Q^* = \sum_{i=1}^{s}(R_i - R^*)^2$, it is actually generalization of Cochran's Q statistic.

3. Monte Carlo permutation test - To obtain statistical significance for average rank and heterogeneity metrics we randomly permute the ranks of each study and the stimulated metrics are calculated. Then we repeat the procedure to generate null distribution for the metrics. Each variable is then tested against the corresponding null distribution. We are interested genuinely in four statistical significances: for high average rank, for low average rank, for high heterogeneity and for low heterogeneity. Distinction between high and low average rank is important as we want to keep the direction of effect in mind. Ignoring it can lead to spurious results that a gene is consistently significant even if it is up-regulated in one study and down-regulated in second one. On the other hand, statistically low heterogeneity may suggest consistent results among different studies. The statistical significance for high average rank (R^*) is defined as the percentage of simulated metrics that exceed or are equal to the observed (R^*). The statistical significance for low average rank (R^*) is defined as the per-

centage of simulated metrics that are below or equal to the observed (R^*). Significance of heterogeneity is defined analogously.

Usage

Data preparation

We will start with computing test statistic and p-value for each gene and data set. Function `meta.test` returns a list with two slots: data frame of test statistics and data frame of p-values. In each of the matrices rows correspond to genes and columns to data sets. Argument `class.col` is a numeric vector indicating which column of *pData* should be used and `data.names` is a character vector with names of the data sets.

```
> metra <- meta.test(denmark, australia, japan,
+     class.col = c(1, 1, 2), data.names = c("denmark",
+         "australia", "japan"))
> head(metra$test)

                denmark  australia      japan
217562_at    -2.1666481  1.0669144 -0.9286918
203766_s_at   0.8318955  0.1014991 -0.9459710
1554394_at    1.2176000  4.0282590  1.1763280
212662_at     1.9755557  1.3046695  2.5983263
1555370_a_at  4.3678119 -0.6042763  2.7174563
240574_at     5.1746999  1.6404196  1.8830312
```

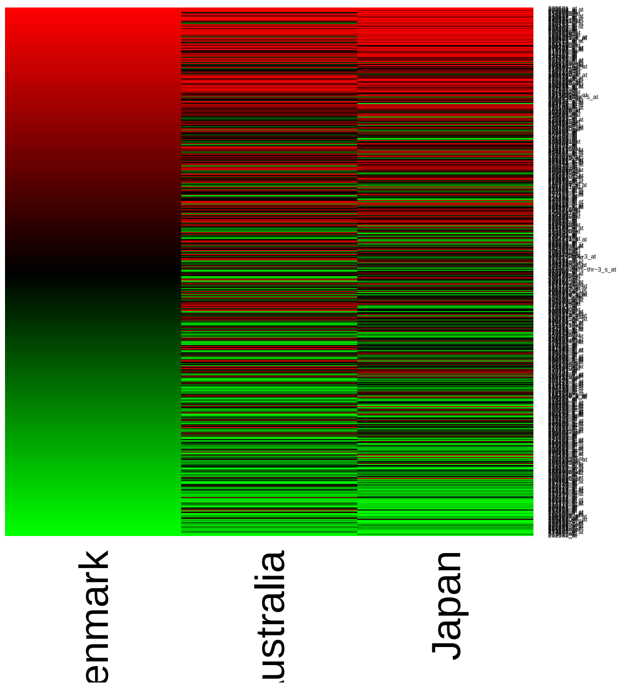
Detecting differentially expressed genes

Now, we can proceed to ranking genes. Function `rank.genes.adv` ranks the genes as described in Algorithm section above.

```
> RANK <- rank.genes.adv(metra)
> head(RANK)

             denmark australia japan
217562_at        105       368   156
203766_s_at      326       257   153
1554394_at       348       488   380
212662_at        390       397   442
1555370_a_at     473       170   445
240574_at        484       424   419
```

The genes ranks can be visualized by

```
> RANK2 <- RANK[order(RANK[, 1]), ]
> colnames(RANK2) <- c("Denmark", "Australia", "Japan")
> heatcol <- colorRampPalette(c("Green", "Black",
+     "Red"))(100)
> metaheat(as.matrix(RANK2), col = heatcol)
```



The next step is to compute average rank R^* and heterogeneity metric Q^* for each gene.

```
> RQ <- compute.RQ(RANK)
> head(round(RQ, 1))

             r.star  q.star
217562_at     209.7 38904.7
203766_s_at   245.3 15168.7
1554394_at    405.3 10762.7
212662_at     409.7  1592.7
1555370_a_at  362.7 56072.7
240574_at     442.3  2616.7
```

And finally we use function `MCtest` to perform Monte Carlo permutation test. Function requires the observed ranks (`RANK`), observed average rank and heterogeneity metric (`RQ`) and number of permutations (`nper`) as arguments. Number of permutations depends on the required accuracy for the final p-values. $1/nper$ is the accuracy for the final p-values. For example with 1000 permutations the p-values are calculated with three decimal places.

```
> MC <- MCtest(RANK, RQ, nper = 1000)

100     200     300     400     500     600     700     800     900     1000

> head(MC)
```

```
             R.high R.low Q.high Q.low
217562_at     0.666 0.337  0.457 0.543
203766_s_at   0.499 0.505  0.740 0.260
1554394_at    0.029 0.971  0.814 0.186
212662_at     0.015 0.985  0.971 0.029
1555370_a_at  0.101 0.900  0.305 0.695
240574_at     0.008 0.992  0.949 0.051
```

Results

The command below creates a character vector of genes with significant average ranks and low heterogeneity. The selected threshold for statistical significance is 0.01.

```
> METRA <- c(rownames(MC)[MC[, 1] < 0.01 & MC[,
+     4] < 0.01], rownames(MC)[MC[, 2] < 0.01 &
+     MC[, 4] < 0.01])
> METRA[1:10]
```

```
 [1] "234207_at"   "225802_at"   "230964_at"   "236223_s_at"
 [5] "201279_s_at" "231829_at"   "230621_at"   "239539_at"
 [9] "238812_at"   "228030_at"
```

Part XI
Results combination

In this part we are going to compare and combine outputs from all methods so we can look and changes in gene expression in various ways.

We are going to start with lists of differentially expressed genes, because this is the only one output common for all methods mentioned in this vignette. We will merge all lists into one variable via function `join.DEG`. The function requires a complete list of genes involved in meta-analysis so it can map indices to gene names like for example function `pvalcombination` provides. Function `featureNames()` returns a character vector with genes present in the ExpressionSet object. Because the same set of genes was measured in each data set we can arbitrarily choose one data set.

```
> lists <- join.DEG(pvalt, ESt, ScoresFDR, x.z,
+     RankRes, z.stat, tsp, probs, genenames = featureNames(denmark),
+     type = c(1, 1, 3, 4, 5, 6, 7, 8), cutoff = 0.01)
> names(lists) <- c("PvalCom", "ESCom", "ESCom2",
+     "OrderedList", "RankProduct", "Z-stat", "TSP",
+     "MAP")
> summary(lists)
```

```
            Length Class  Mode
PvalCom     160    -none- character
ESCom       109    -none- character
ESCom2      163    -none- character
```

```
OrderedList 231    -none- character
RankProduct 140    -none- character
Z-stat      150    -none- character
TSP           2    -none- character
MAP           6    -none- character
```

Now, we will transform this list to a binary matrix where rows refer to genes and columns to method and 1 means that the gene was identified as a differentially expressed gene in the method. Function `make.matrix` provides such transformation.

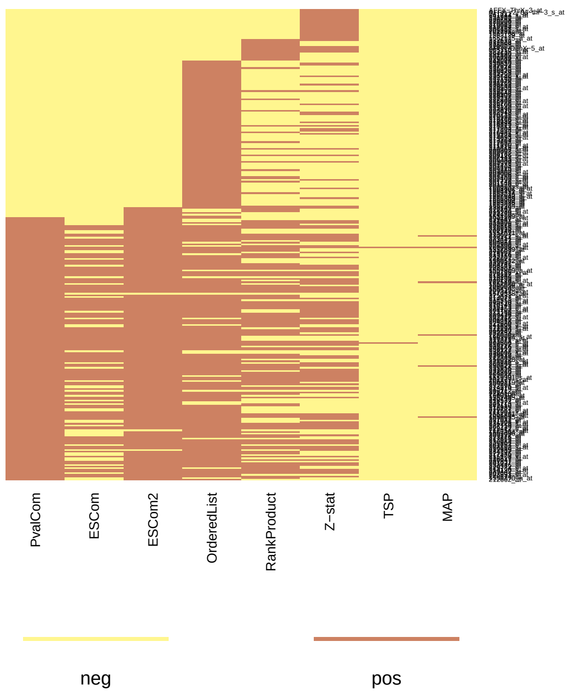
```
> MAT <- make.matrix(lists)
> MAT[1:5, 1:5]
```

```
             PvalCom ESCom ESCom2 OrderedList RankProduct
212662_at          1     0      1           1           0
1555370_a_at       1     0      1           0           1
240574_at          1     1      1           1           1
203553_s_at        1     1      1           1           0
207607_at          1     0      1           1           1
```

It is very popular to visualize results of microarray analysis as a heatmap. A heatmap is a graphical representation for a numeric matrix where values are presented as colors. Gene expression values are usually used in microarray analysis. In these pictures colors go continuously from green (for down-regulation) through black (for no change in gene expression) to red (for up-regulation). There are several R-packages which implement plotting heatmaps in slightly different way. Functions `metaheat` and `metaheat2` are modification of two of them, so a discrete set of colors (only two in `metaheat` but even several in `metaheat2`) can be used with an appropriate legend.

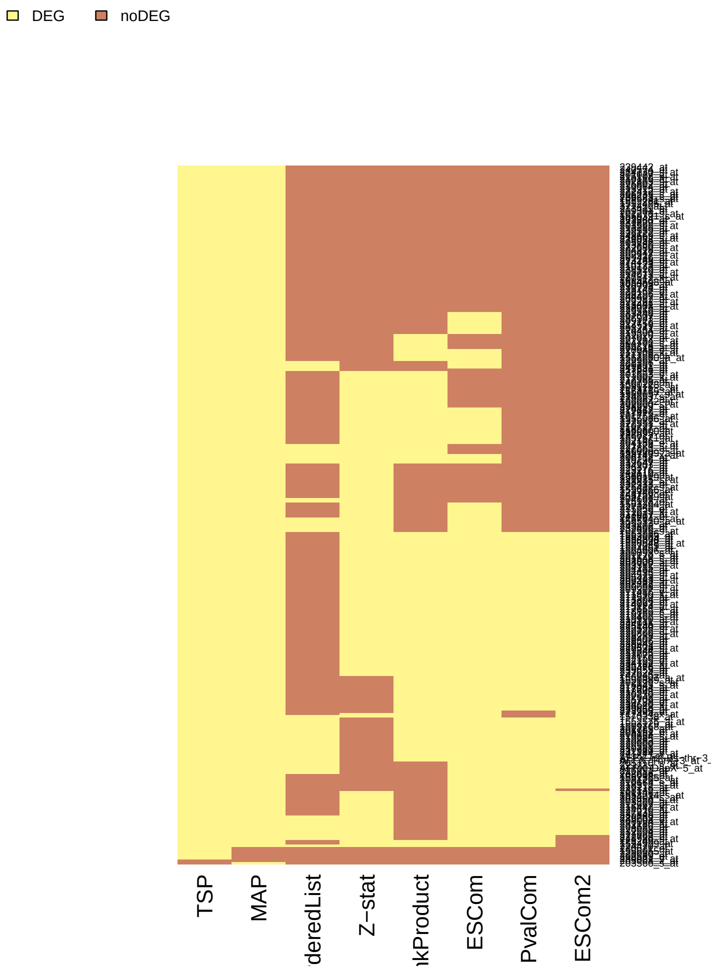
Function `metaheat` has three arguments: a data matrix (`MAT`), a number defining position of legend (`legend=1` is legend drawn below the picture) and vector of colors (`col`).

```
> metaheat(MAT, legend = 1, col = c("khaki1", "lightsalmon3"))
```



Function `metaheat2` has as many arguments as `heatmap.2` form gplots package and two more. Argument `legend.names` is a character vector with labels to be used in legend. Setting `discret=TRUE` will indicate that legend for discrete values should be drawn.

```
> metaheat2(MAT, col = c("khaki1", "lightsalmon3"),
+     legend.names = c("DEG", "noDEG"), discrete = TRUE,
+     trace = "none", dendrogram = "none")
```



The user can perform cluster analysis on `MAT` to search for similarities between methods or genes.

We can look at number of genes found by number of methods by

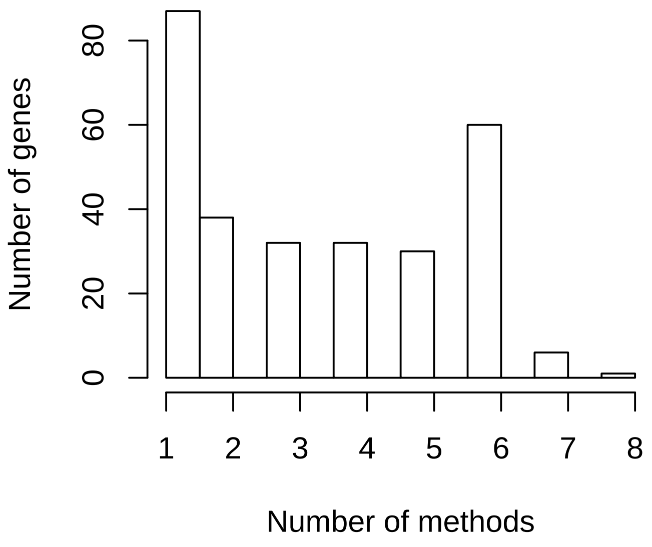
```
> dim(MAT)
```

```
[1] 286   8
```

According to the outsprint above, eight different methods have found 217 differentially expressed genes.

The histogram below shows that the most of the genes have been selected in only one method.

```
> n.met <- apply(MAT, 1, sum)
> hist(n.met, main = "", xlab = "Number of methods",
+      ylab = "Number of genes", xlim = c(1, 8))
```



`n.met` is a numeric vector of number of methods that identified the gene as differentially expressed.

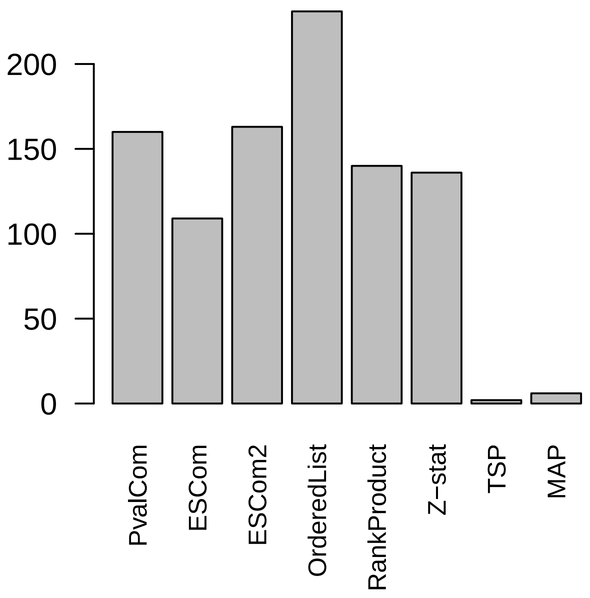
Next, we can look for example how many genes have been found as differentially expressed in at least 6 methods.

```
> dim(MAT[n.met > 5, ])
```

```
[1] 67  8
```

On the other hand, we can find out how many genes have been found by a method.

```
> n.gen <- apply(MAT, 2, sum)
> barplot(n.gen, cex.names = 0.8, las = 2)
```



Function `contig.tab` provides a number of genes common in two gene lists. It can be applied to `lists`, too.

```
> TAB <- conting.tab(lists)
> TAB[1:5, 1:5]

            PvalCom ESCom ESCom2 OrderedList RankProduct
PvalCom          NA   109    157         139         108
ESCom           109    NA    109         102          84
ESCom2          157   109     NA         140         111
OrderedList     139   102    140          NA         113
RankProduct     108    84    111         113          NA
```

Expression of one gene

In this section we are going to focus on one gene and to look at its expression change from different points of view. The different points of view are represented by different approaches used in the methods.

First we will join all the available results to one list and then select only rows for one gene.

```
> results <- join.results(pvalt, ESt, theScores,
+     ScoresFDR$two.sided, x.z, RankRes, z.stat,
+     probs, MC, RQ, type = c(1, 1, 5, 5, 2, 3,
+         5, 4, 5, 5), genenames = rownames(exprs(denmark)))
> gene <- metagene("203008_x_at", results)
> gene
```

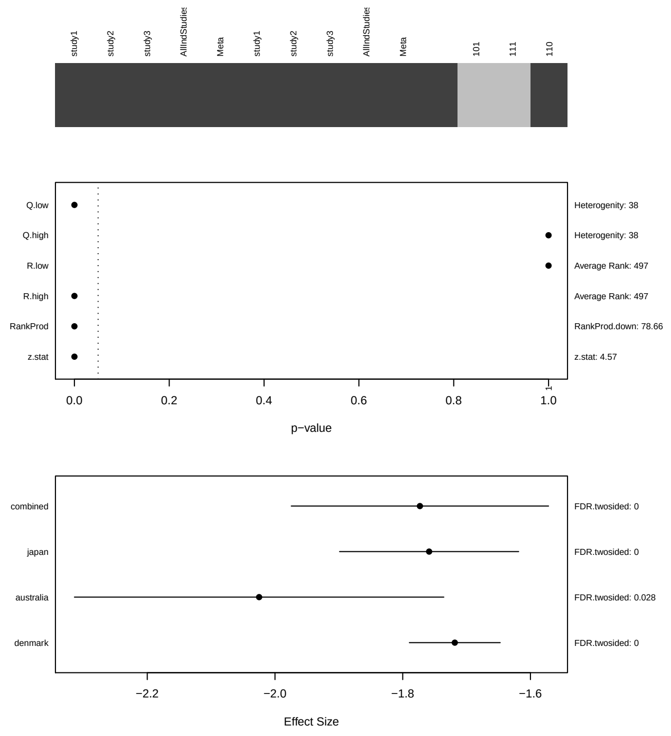
```
[[1]]
       study1        study2        study3 AllIndStudies 
      1.00000       1.00000       1.00000       1.00000 
         Meta TestStatistic 
      1.00000      -8.92453 

[[2]]
       study1        study2        study3 AllIndStudies 
     1.000000      1.000000      1.000000      1.000000 
         Meta TestStatistic 
     1.000000      8.674749 

[[3]]
     zSco_Ex_1      zSco_Ex_2      zSco_Ex_3           zSco 
   -6.44329249    -3.76525489    -4.69659562    -8.80275939 
        MUvals          MUsds          Qvals             df 
   -1.77309287     0.20142467     0.26260012     2.00000000 
      Qpvalues          Chisq    Effect_Ex_1    Effect_Ex_2 
    0.87695460     0.00000000    -1.71847860    -2.02486588 
   Effect_Ex_3 EffectVar_Ex_1 EffectVar_Ex_2 EffectVar_Ex_3 
   -1.75867849     0.07113324     0.28920365     0.14021890 

[[4]]
 zSco_Ex_1   FDR_Ex_1  zSco_Ex_2   FDR_Ex_2  zSco_Ex_3 
-6.4432925  0.0000000 -3.7652549  0.0275000 -4.6965956 
  FDR_Ex_3       zSco        FDR     MUvals      MUsds 
 0.0000000 -8.8027594  0.0000000 -1.7730929  0.2014247 
     Qvals         df   Qpvalues      Chisq 
 0.2626001  2.0000000  0.8769546  0.0000000 

[[5]]
[1] TRUE

[[6]]
        gene.index            RP/Rsum FC:(class1/class2) 
          415.0000            78.6607             0.5803 
               pfp            P.value 
            0.0000             0.0000 

[[7]]
              Zscore       Pvalue
203008_x_at 4.565183 4.990581e-06

[[8]]
  101   111   110 
FALSE FALSE  TRUE 

[[9]]
R.high  R.low Q.high  Q.low 
     0      1      1      0 
```

```
[[10]]
r.star q.star
   497     38
```

```
> save(gene, file = "gen.RData")
```

This output provides much of the information available on the gene through all the described methods. It is a rather complicated structure, so we will try to represent it graphically in comprehensible form.

```
> plotgene(gene, type = c(1, 1, 2, 3, 4, 5, 6, 7,
+     8, 9), datalabels = c("denmark", "australia",
+     "japan", "combined"))
```



The picture above shows in top part occurrence of gene in ... , in list of overlapping genes in SOGL method and in gene lists with observed MAP (Meta-Analysis Pattern). The dark box means that the gene is present in the list. Values from objects: `pvalt`, `ESt`, `x.z` and `probs` are used in here.

The middle part is dedicated to p-values available in meta-analysis. Specific values of the statistics can be found on the right side of the chart. The vertical dashed line denotes the signifficance threshold 5%. P-values from `MC`, `RankRes` and `z.stat` are drawn in here.

Combination of effect size is plotted in the bottom graph. The point marks the effect size. Horizontal lines denote the variance of effect size. Statistical significance of the difference in gene expression (FDR adjusted) can be found on the right side of the chart. This graph uses values from `theScores` and `ScoresFDR`.

References

[1] Jorissen, R. N., Lipton, L., Gibbs, P., Chapman, M. et al. 2008, *DNA copy-number alterations underlie gene expression differences between microsatellite stable and unstable colorectal cancers*, Clinical Cancer Research, Vol. 14, pp. 8061-8069

[2] Watanabe, T., Kobunai, T., Toda, E., Yamamoto, Y. et al. 2006, *Distal colorectal cancers with microsatellite instability (MSI) display distinct gene expression profiles that are different from proximal MSI cancers* Cancer Research, Vol.66, no. 20, pp. 9804-9808

[3] Falcon, S., Morgan, M. and Gentleman, R. 2007, *An introduction to Biocinductor's ExpressionSet class*, available at: http://www.bioconductor.org/packages/2.2/bioc/vignettes/Biobase/inst /doc/ExpressionSetIntroduction.pdf

[4] Marot, G., Foulley, J.L., Mayer, C.D.,Jaffrézic, F. 2009, *Moderated effect size and P-value combinations for microarray meta-analyses*, Bioinformatrics, Vol. 25 no. 20 2009, pp. 2692-2699

[5] Rhodes, D.R., Barrette, T.R., Rubin, M. A., Ghosh, D. a Chinnaiyan, A. M. 2002, *Meta-Analysis of Microarrays: Interstudy Validation of Gene Expression Profiles Reveals Pathway Dysregulation in Prostate Cancer*, CANCER RESEARCH 62, pp: 4427-4433

[6] Fisher, R.A. 1925, *Statistical methods for research*, Oliver and Boyd, Edinburgh

[7] Smyth, G. K. 2004, *Linear models and empirical Bayes methods for assessing differential expression in microarray experiments*, Statistical Applications in Genetics and Molecular Biology 3, No. 1, Article 3

[8] Jaffrézic, F., Marot, G., Degrelle, S., Hue, I., Foulley, J.L. 2007, *A structural mixed model for variances in differential gene expression studies*, Genetical Research, Vol. 89, pp. 19-25.

[9] Choi, J.K., Yu, U., Kim, S. a Yoo, O.J. 2003, *Combining multiple microarray studies and modeling interstudy variation*, Bioinformatics, Vol. 19, Suppl. 1 2003, pp. i84-i90

[10] Gentleman, R., Rauschhaupt, M., Huber, W., a Lusa L. 2008, *Meta-analysis for Microarray Experiments*, dostupné na: http://www.bioconductor.org/packages/2.3/bioc/vignettes/GeneMeta /inst/doc/GeneMeta.pdf

[11] Hedged, V. L. a Olkin, I. 1985, *Statistical Methods for Metaanalysis*, Academic Press, Orlando

[12] Cochran, B.G. 1954, *The combination of estimates from different experiments*, Biometrics, Vol. 10, pp. 101-129

[13] DerSimonian, R., a Laird, N. M. 1986, *Meta-analysis in clinical trials*, Controlled Clinical Trials, Vol. 7, pp. 177-188

[14] Scheid, S., Lottaz, C., Yang, X. a Spang, R. 2006, *Similarities of Ordered Gene Lists User's Guide to the Bioconductor Package OrderedList 1.11.3*, dostupné na: http://www.bioconductor.org/packages/2.5/bioc/vignettes/OrderedList/inst/doc/tr_2006_01.pdf

[15] Hong, F., Breitling, R., McEntee,C. W., Wittner, B. S., Nemhauser, J. L. a Chory, J. 2006, *RankProd: a bioconductor package for detecting differentially expressed genes in meta-analysis*, Bioinformatics, Vol. 22, no. 22 2006, pp. 2825-2827

[16] Wang, J., Coombes, K. R., Highsmith, W. E., Keating, M. J. a Abruzzo, L. V. 2004, *Differences in gene expression between B-cell chronic lymphocytic leukemia and normal B cells: a meta-analysis of three microarray studies*, Bioinformatics, Vol. 20, no. 17 2004, pp. 3166-3178

[17] Ghosh, D. a Choi, H. 2009, *metaArray package for meta-analysis of microarray data*, dostupné na: http://bioconductor.org/packages/2.5/bioc/vignettes/metaArray/inst/doc/metaArray.pdf

[18] Geman, D., d'Avignon, Ch., Naiman, D. Q. a Winslow, R.L. 2004, *Classifying Gene Expression Profiles from Pairwise mRNA Comparisons*, Statistical Applications in Genetics and Molecular Biology 2004, Vol. 3, Issue 1, Article 19

[19] A.C. Tan, D.Q. Naiman, L. Xu, R.L. Winslow, D. Geman, *Simple decision rules for classifying human cancers from gene expression profiles*, Bioinformatics, 21: 3896-3904, 2005.

[20] Smid, M., Dorssers, L. C. J. a Jenster, G. 2003, *Venn Mapping: clustering of heterologous microarray data based on the number of co-occurring differentially expressed genes*, Bioinformatics, Vol. 19, no. 16 2003, pp. 2065-2071

[21] Yang, X., Bentink, S. a Spang, R. 2005, *Detecting Common Gene Expression Patterns in Multiple Cancer Outcome Entities*, Biomedical Microdevices, Vol.7:3, pp. 247-251

[22] Rhodes, D. R., Yu, J., Shanker, K., Deshpande, N., Varambally, R., Ghosh, D., Barrette, T., Pandey, A. a Chinnaiyan, A. M. 2004, *Large-scale meta-analysis of cancer microarray data identifies common transcriptional profiles of neoplastic transformation and progression*, PNAS, Vol. 101, no. 25, pp. 9309-9314

[23] Zintzaraz, E a Ioannidis, J.P.A. 2008, *Meta-analysis for ranked discovery datasets: Theoretical framework and empirical demonstration for microarrays*, Computational Biology and Chemistry 32, pp. 39-47