Package ‘ClassComparison’

May 6, 2019

Version 3.1.8

Date 2019-05-01

Title Classes and Methods for ``Class Comparison'' Problems on Microarrays

Author Kevin R. Coombes

Maintainer Kevin R. Coombes <krc@silicovore.com>

Description Defines the classes used for ``class comparison'' problems in the OOMPA project (<http://oompa.r-forge.r-project.org/>). Class comparison includes tests for differential expression; see Simon's book for details on typical problem types.

Depends R (>= 3.0), oompaBase (>= 3.0)

Imports methods, graphics, stats, splines, Biobase

License Apache License (== 2.0)

LazyLoad yes

biocViews Microarray, DifferentialExpression, MultipleComparisons

URL http://oompa.r-forge.r-project.org/

NeedsCompilation no

Repository CRAN

Date/Publication 2019-05-06 16:40:12 UTC

R topics documented:

Bum-class *Class "Bum"*

Description

The `Bum` class is used to fit a beta-uniform mixture model to a set of p-values.

Usage

```
Bum(pvals, ...)
## S4 method for signature 'Bum'
summary(object, tau=0.01, ...)
## S4 method for signature 'Bum'
hist(x, res=100, xlab='P Values', main='', ...)
## S4 method for signature 'Bum'
image(x, ...)
## S4 method for signature 'Bum'
cutoffSignificant(object, alpha, by='FDR', ...)
## S4 method for signature 'Bum'
selectSignificant(object, alpha, by='FDR', ...)
## S4 method for signature 'Bum'
countSignificant(object, alpha, by='FDR', ...)
likelihoodBum(object)
```

Arguments

`pvals`	numeric vector containing values between `0` and `1`
`object`	object of class `Bum`
`tau`	numeric scalar between `0` and `1`, representing a cutoff on the p-values
`x`	object of class `Bum`
`res`	positive integer scalar specifying the resolution at which to plot the fitted distribution curve
`xlab`	character string specifying the label for the x axis
`main`	character string specifying the graph title

`alpha`	Either the false discovery rate (if `by = 'FDR'`) or the posterior probability (if `by = 'EmpiricalBayes'`)
`by`	character string denoting the method to use for determining cutoffs. Valid values are: • FDR • FalseDiscovery • EmpiricalBayes
`...`	extra arguments for generic or plotting routines

Details

The BUM method was introduced by Stan Pounds and Steve Morris, although it was simultaneously discovered by several other researchers. It is generally applicable to any analysis of microarray or proteomics data that performs a separate statistical hypothesis test for each gene or protein, where each test produces a p-value that would be valid if the analyst were only performing one statistical test. When performing thousands of statistical tests, however, those p-values no longer have the same interpretation as Type I error rates. The idea behind BUM is that, under the null hypothesis that none of the genes or proteins is interesting, the expected distribution of the set of p-values is uniform. By contrast, if some of the genes are interesting, then we should see an overabundance of small p-values (or a spike in the histogram near zero). We can model the alternative hypothesis with a beta distribution, and view the set of all p-values as a mixture distribution.

Fitting the BUM model is straightforward, using a nonlinear optimizer to compute the maximum likelihood parameters. After the model has been fit, one can easily determine cutoffs on the p-values that correspond to desired false discovery rates. Alternatively, the original Pounds and Morris paper shows that their results can be reinterpreted to recover the empirical Bayes method introduced by Efron and Tibshirani. Thus, one can also determine cutoffs by specifying a desired posterior probability of significance.

Value

Graphical functions (`hist` and `image`) invisibly return the object on which they were invoked.

The `cutoffSignificant` method returns a real number between zero and one. P-values below this cutoff are considered statistically significant at either the specified false discovery rate or at the specified posterior probability.

The `selectSignificant` method returns a vector of logical values whose length is equal to the length of the vector of p-values that was used to construct the `Bum` object. True values in the return vector mark the statistically significant p-values.

The `countSignificant` method returns an integer, the number of statistically significant p-values.

The `summary` method returns an object of class `BumSummary`.

Creating Objects

Although objects can be created directly using `new`, the most common usage will be to pass a vector of p-values to the `Bum` function.

Slots

`pvals`**:** numeric vector of p-values used to construct the object.

`ahat`**:** Model parameter

`lhat`**:** Model parameter

`pihat`**:** Model parameter

Methods

summary(object, tau=0.01, ...) For each value of the p-value cutoff `tau`, computes estimates of the fraction of true positives (TP), false negatives (FN), false positives (FP), and true negatives (TN).

hist(x, res=100, xlab='P Values', main='', ...) Plots a histogram of the object, and overlays (1) a straight line to indicate the contribution of the uniform component and (2) the fitted beta-uniform distribution from the observed values. Colors in the plot are controlled by `oompaColor$EXPECTED` and `oompaColor$OBSERVED`.

image(x, ...) Produces four plots in a 2x2 layout: (1) the histogram produced by `hist`; (2) a plot of cutoffs against the desired false discovery rate; (3) a plot of cutoffs against the posterior probability of coming from the beta component; and (4) an `ROC` curve.

cutoffSignificant(object, alpha, by='FDR', ...) Computes the cutoff needed for significance, which in this case means arising from the beta component rather than the uniform component of the mixture. Significance is specified either by the false discovery rate (when `by = 'FDR'` or `by = 'FalseDiscovery'`) or by the posterior probability (when `by = 'EmpiricalBayes'`)

selectSignificant(object, alpha, by='FDR', ...) Uses `cutoffSignificant` to determine a logical vector that indicates which of the p-values are significant.

countSignificant(object, alpha, by='FDR', ...) Uses `selectSignificant` to count the number of significant p-values.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Pounds S, Morris SW.
Estimating the occurrence of false positives and false negatives in microarray studies by approximating and partitioning the empirical distribution of p-values.
Bioinformatics. 2003 Jul 1;19(10):1236-42.

Benjamini Y, Hochberg Y.
Controlling the false discovery rate: a practical and powerful approach to multiple testing.
J Roy Statist Soc B, 1995; 57: 289-300.

Efron B, Tibshirani R.
Empirical bayes methods and false discovery rates for microarrays.
Genet Epidemiol 2002, 23: 70-86.

See Also

Two classes that produce lists of p-values that can (and often should) be analyzed using BUM are `MultiTtest` and `MultiLinearModel`. Also see `BumSummary`.

Examples

```
showClass("Bum")
fake.data <- c(runif(700), rbeta(300, 0.3, 1))
a <- Bum(fake.data)
hist(a, res=200)

alpha <- (1:25)/100
plot(alpha, cutoffSignificant(a, alpha, by='FDR'),
     xlab='Desired False Discovery Rate', type='l',
     main='FDR Control', ylab='Significant P Value')

GAMMA <- 5*(10:19)/100
plot(GAMMA, cutoffSignificant(a, GAMMA, by='EmpiricalBayes'),
     ylab='Significant P Value', type='l',
     main='Empirical Bayes', xlab='Posterior Probability')

b <- summary(a, (0:100)/100)
be <- b@estimates
sens <- be$TP/(be$TP+be$FN)
spec <- be$TN/(be$TN+be$FP)
plot(1-spec, sens, type='l', xlim=c(0,1), ylim=c(0,1), main='ROC Curve')
points(1-spec, sens)
abline(0,1)

image(a)

countSignificant(a, 0.05, by='FDR')
countSignificant(a, 0.99, by='Emp')
```

`BumSummary-class` *Class "BumSummary"*

Description

An implementation class. Users are not expected to create these objects directly; they are produced as return objects from the summary method for `Bum`.

Slots

`bum`: object of class `Bum`

`estimates`: data.frame

`Fhat`: numeric

Methods

show `signature(object = "BumSummary")`: Print the object, which contains a summary of the underlying `Bum` object. The summary contains a data frame with estimates of the fraction of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) at the set of p-value cutoffs specified in the call to the summary method.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

Bum

Examples

```
showClass("BumSummary")
```

`Dudoit-class` *Class "Dudoit"*

Description

An implementation of the method of Dudoit and colleagues to apply the Westfall-Young adjustment to p-values to control the family-wise error rate when analyzing microarray data.

Usage

```
Dudoit(data, classes, nPerm=1000, verbose=TRUE)
## S4 method for signature 'Dudoit,missing'
plot(x, y, xlab='T-Statistic', ylab='P-Value', ...)
## S4 method for signature 'Dudoit'
cutoffSignificant(object, alpha, ...)
## S4 method for signature 'Dudoit'
selectSignificant(object, alpha, ...)
## S4 method for signature 'Dudoit'
countSignificant(object, alpha, ...)
```

Arguments

`data`	either a data frame or matrix with numeric values, or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data.
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`nPerm`	integer scalar specifying the number of permutations to perform
`verbose`	logical scalar. If `TRUE`, prints additional output

`object`	object of class `Dudoit`
`alpha`	numeric scalar specifying the target family-wise error rate
`x`	object of class `Dudoit`
`y`	Nothing, since it is supposed to be missing. Changes to the Rd processor require documenting the missing entry.
`xlab`	character string specifying label for the x axis
`ylab`	character string specifying label for the y axis
`...`	extra arguments for generic or plotting routines

Details

In 2002, Dudoit and colleagues introduced a method to adjust the p-values when performing gene-by-gene tests for differential expression. The adjustment was based on the method of Westfall and Young, with the goal of controlling the family-wise error rate.

Value

The standard method for `plot` returns what you would expect.

The `cutoffSignificant` method returns a real number (its input value `alpha`). The `selectSignificant` method returns a vector of logical values identifying the significant test results, and `countSignificant` returns an integer counting the number of significant test results.

Objects from the Class

As usual, objects can be created by `new`, but better methods are available in the form of the `Dudoit` function. The basic inputs to this function are the same as those used for row-by-row statistical tests throughout the ClassComparison package; a detailed description can be found in the `MultiTtest` class.

The additional input determines the number, `nPerm`, of permutations to perform. The accuracy of the p-value adjustment depends on this value. Since the implementation is in R (and does not call out to something compiled like C or FORTRAN), however, the computations are slow. The default value of 1000 can take a long time with modern microarrays that contain 40,000 spots.

Slots

`adjusted.p`**:** numeric vector of adjusted p-values.

`t.statistics`**:** Object of class `numeric` containing the computed t-statistics.

`p.values`**:** Object of class `numeric` containing the computed p-values.

`groups`**:** Object of class `character` containing the names of the classes being compared.

`call`**:** Object of class `call` containing the function call that created the object.

Extends

Class `MultiTtest`, directly. In particular, objects of this class inherit methods for `summary`, `hist`, and `plot` from the base class.

Methods

cutoffSignificant(object, alpha, . . .) Determine cutoffs on the adjusted p-values at the desired significance level. In other words, this function simply returns `alpha`.

selectSignificant(object, alpha, . . .) Compute a logical vector for selecting significant test results.

countSignificant(object, alpha, . . .) Count the number of significant test results.

plot `signature(x=Dudoit, y=missing)`: ...

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Dudoit S, Yang YH, Callow MJ, Speed TP.
Statistical Methods for Identifying Differentially Expressed Genes in Replicated cDNA Microarray Experiments.
Statistica Sinica (2002), 12(1): 111-139.

Westfall PH, Young SS.
Resampling-based multiple testing: examples and methods for p-value adjustment.
Wiley series in probability and mathematics statistics. John Wiley and Sons, 1993.

See Also

`Bum`, `MultiTtest`, `SmoothTtest`

Examples

```
showClass("Dudoit")
ng <- 10000
ns <- 15
nd <- 200
fake.class <- factor(rep(c('A', 'B'), each=ns))
fake.data <- matrix(rnorm(ng*ns*2), nrow=ng, ncol=2*ns)
fake.data[1:nd, 1:ns] <- fake.data[1:nd, 1:ns] + 2
fake.data[(nd+1):(2*nd), 1:ns] <- fake.data[(nd+1):(2*nd), 1:ns] - 2

# the permutation test is slow. it really needs many more
# than 10 permutations, but this is just an example...
dud <- Dudoit(fake.data, fake.class, nPerm=10)
summary(dud)
plot(dud)
countSignificant(dud, 0.05)
```

`dwil` *Wilcoxon Density Function*

Description

Computes the density function for the Wilcoxon rank-sum distribution without centering.

Usage

```
dwil(q, m, n)
```

Arguments

`q`	vector of quantiles
`m`	number of observations in the first sample
`n`	number of observations in the second sample

Details

Computes the density function for the Wilcoxon rank-sum distribution, using exact values when both groups have fewer than 50 items and switching to a normal approximation otherwise. It was originally written for S-Plus, which still perversely insists that `m` and `n` must be less than 50. The function was retained when the OOMPA library was ported to R, since S-Plus keeps the actual rank-sum but R centers the distribution at zero. This function encapsulated the difference, allowing everything else to continue to work as it had worked previously.

Value

A vector of the same length as `q` containing (approximate or exact) values of the density function.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

`MultiWilcoxonTest`

Examples

```
dwil(51:60, 9, 3)
dwil(51:60, 9, 51)
```

MultiLinearModel-class *Class "MultiLinearModel"*

Description

Class to fit multiple (row-by-row) linear (fixed-effects) models on microarray or proteomics data.

Usage

```
MultiLinearModel(form, clindata, arraydata)
## S4 method for signature 'MultiLinearModel'
summary(object, ...)
## S4 method for signature 'MultiLinearModel'
hist(x, xlab='F Statistics', main=NULL, ...)
## S4 method for signature 'MultiLinearModel,missing'
plot(x, y, ylab='F Statistics', ...)
## S4 method for signature 'MultiLinearModel,ANY'
plot(x, y, xlab='F Statistics',
 ylab=deparse(substitute(y)), ...)
## S4 method for signature 'MultiLinearModel'
anova(object, ob2, ...)
multiTukey(object, alpha)
```

Arguments

`form`	`formula` object specifying the linear model
`clindata`	either a data frame of "clinical" or other covariates, or an `ExpressionSet`.
`arraydata`	matrix or data frame of values to be explained by the model. If `clindata` is an `ExpressionSet`, then `arraydata` can be omitted, since it is assumed to be part of the `ExpressionSet`.
`object`	object of class `MultiLinearModel`
`ob2`	object of class `MultiLinearModel`
`x`	object of class `MultiLinearModel`
`y`	optional numeric vector
`xlab`	character string specifying label for the x-axis
`ylab`	character string specifying label for the y-axis
`main`	character string specifying graph title
`...`	extra arguments for generic or plotting functions
`alpha`	numeric scalar between `0` and `1` specifying the significance level for the Tukey test.

Value

The `anova` method returns a data frame. The rows in the data frame corresponds to the rows in the `arraydata` object that was used to construct the `MultiLinearModel` objects. The first column contains the F-statistics and the second column contains the p-values.

The `multiTukey` function returns a vector whose length equals the number of rows in the `arraydata` object used to construct the `MultiLinearModel`. Assuming that the overall F-test was significant, differences in group means (in each data row) larger than this value are significant by Tukey's test for honestly significant difference. (Of course, that statement is incorrect, since we haven't fully corrected for multiple testing. Our standard practice is to take the p-values from the row-by-row F-tests and evaluate them using the beta-uniform mixture model (see `Bum`). For the rows that correspond to models whose p-values are smaller than the `Bum` cutoff, we simply use the Tukey HSD values without further modification.)

Creating Objects

Objects should be created by calling the `MultiLinearModel` function. The first argument is a `formula` specifying the linear model, in the same manner that it would be passed to `lm`. We will fit the linear model separately for each row in the `arraydata` matrix. Rows of `arraydata` are attached to the `clindata` data frame and are always referred to as "Y" in the formulas. In particular, this implies that `clindata` can not include a column already called "Y". Further, the implementation only works if "Y" is the response variable in the model.

Multiple linear models with "ExpressionSet" objects

The BioConductor packages uses an `ExpressionSet` to combine microarray data and clinical covariates (known in their context as `phenoData` objects) into a single structure. You can call `MultiLinearModel` using an `ExpressionSet` object for the `clindata` argument. In this case, the function extracts the `phenoData` slot of the `ExpressionSet` to use for the clinical covariates, and extracts the `exprs` slot of the `ExpressionSet` object to use for the array data.

Slots

`call`**:** A `call` object describing how the object was constructed.

`model`**:** The `formula` object specifying the linear model.

`F.statistics`**:** A numeric vector of F-statistics comparing the linear model to the null model.

`p.values`**:** A numeric vector containing the p-values associated to the F-statistics.

`coefficients`**:** A `matrix` of the coefficients in the linear models.

`predictions`**:** A `matrix` of the (Y-hat) values predicted by the models.

`sse`**:** A numeric vector of the sum of squared error terms from fitting the models.

`ssr`**:** A numeric vector of the sum of squared regression terms from the model.

`df`**:** A numeric vector (of length two) containing the degrees of freedom for the F-tests.

Methods

summary(object, ...) Write out a summary of the object.

hist(x, xlab='F Statistics', main=NULL, ...) Create a histogram of the F-statistics.

plot(x, ylab='F Statistics', ...) Plot the F-statistics as a function of the row index.

plot(x, y, xlab='F Statistics', ylab=deparse(substitute(y)), ...) Plot the F-statistics against the numeric vector `y`.

anova(object, ob2, ...) Perform row-by-row F-tests comparing two different linear models.

Details

The `MultiLinearModel` constructor computes row-by-row F-tests comparing each linear model to the null model Y ~ 1. In many instances, one wishes to use an F-test to compare two different linear models. For instance, many standard applications of analysis of variance (ANOVA) can be described using such a comparison between two different linear models. The `anova` method for the `MultiLinearModel` class performs row-by-row F-tests comparing two competing linear models.

The implementation of `MultiLinearModel` does not take the naive approach of using either `apply` or a `for`-loop to attach rows one at a time and fit separate linear models. All the models are actually fit simultaneously by a series of matrix operations, which greatly reduces the amount of time needed to compute the models. The constraint on the column names in `clindata` still holds, since one row is attached to allow `model.matrix` to determine the contrasts matrix.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

See Also

`anova`, `lm`, `Bum`, `MultiTtest`, `MultiWilcoxonTest`

Examples

```
showClass("MultiLinearModel")
ng <- 10000
ns <- 50
dat <- matrix(rnorm(ng*ns), ncol=ns)
cla <- factor(rep(c('A', 'B'), 25))
cla2 <- factor(rep(c('X', 'Y', 'Z'), times=c(15, 20, 15)))
covars <- data.frame(Grade=cla, Stage=cla2)
res <- MultiLinearModel(Y ~ Grade + Stage, covars, dat)
summary(res)
hist(res, breaks=101)
plot(res)
plot(res, res@p.values)

graded <- MultiLinearModel(Y ~ Grade, covars, dat)
summary(graded)

hist(graded@p.values, breaks=101)
hist(res@p.values, breaks=101)
```

```
oop <- anova(res, graded)
hist(oop$p.values, breaks=101)
```

MultiTtest-class *Class "MultiTtest"*

Description

Class to perform row-by-row t-tests on microarray or proteomics data.

Usage

```
MultiTtest(data, classes, na.rm=TRUE)
## S4 method for signature 'MultiTtest'
summary(object, ...)
## S4 method for signature 'MultiTtest'
as.data.frame(x, row.names=NULL, optional=FALSE, ...)
## S4 method for signature 'MultiTtest'
hist(x, xlab='T Statistics', main=NULL, ...)
## S4 method for signature 'MultiTtest,missing'
plot(x, y, ylab='T Statistics', ...)
## S4 method for signature 'MultiTtest,ANY'
plot(x, y, xlab='T Statistics', ylab=deparse(substitute(y)), ...)
```

Arguments

`data`	either a data frame or matrix with numeric values, or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`na.rm`	logical scalar. If `TRUE`, compute t-statistics after omitting `NA` values from individual rows of the data matrix
`object`	object of class `MultiTtest`
`x`	object of class `MultiTtest`
`y`	numeric vector
`xlab`	character string specifying the label for the x axis
`ylab`	character string specifying the label for the y axis
`main`	character string specifying the plot title
`row.names`	see the base version
`optional`	see the base version
`...`	extra arguments for generic or plotting routines

Value

The graphical routines invisibly return the object against which they were invoked.

Creating objects

Although objects can be created using `new`, the preferred method is to use the `MultiTtest` generator. In the simplest case, you simply pass in a data matrix and a logical vector assigning classes to the columns, and the constructor performs row-by-row two-sample t-tests and computes the associated (single test) p-values. To adjust for multiple testing, you can pass the p-values on to the `Bum` class.

If you use a factor instead of a logical vector, then the t-test compares the first level of the factor to everything else. To handle the case of multiple classes, see the `MultiLinearModel` class.

As with other class comparison functions that are part of the OOMPA, we can also perform statistical tests on `ExpressionSet` objects from the BioConductor libraries. In this case, we pass in an `ExpressionSet` object along with the name of a factor to use for splitting the data.

Slots

`t.statistics`**:** Object of class `numeric` containing the computed t-statistics.

`p.values`**:** Object of class `numeric` containing the computed p-values.

`df`**:** Numeric vector of the degrees of freedom per gene. Introduced to allow for missing data.

`groups`**:** Object of class `character` containing the names of the classes being compared.

`call`**:** Object of class `call` containing the function call that created the object.

Methods

summary(object, ...) Write out a summary of the object.

hist(x, xlab='T Statistics', main=NULL, ...) Produce a histogram of the t-statistics.

plot(x) Produces a scatter plot of the t-statistics against their index.

plot(x,y) Produces a scatter plot of the t-statistics in the object `x` against the numeric vector `y`.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

See Also

`matrixT`, `Bum`, `Dudoit`, `MultiLinearModel`

Examples

```
showClass("MultiTtest")
ng <- 10000
ns <- 50
dat <- matrix(rnorm(ng*ns), ncol=ns)
cla <- factor(rep(c('A', 'B'), each=25))
res <- MultiTtest(dat, cla)
summary(res)
```

```
hist(res, breaks=101)
plot(res)
plot(res, res@p.values)
hist(res@p.values, breaks=101)

dat[1,1] <- NA
mm <- matrixMean(dat, na.rm=TRUE)
vv <- matrixVar(dat, mm, na.rm=TRUE)
tt <- matrixT(dat, cla, na.rm=TRUE)
mtt <- MultiTtest(dat,cla)
```

`MultiWilcoxonTest-class`

Class "MultiWilcoxonTest"

Description

The `MultiWilcoxonTest` class is used to perform row-by-row Wilcoxon rank-sum tests on a data matrix. Significance cutoffs are determined by the empirical Bayes method of Efron and Tibshirani.

Usage

```
MultiWilcoxonTest(data, classes, histsize=NULL)
## S4 method for signature 'MultiWilcoxonTest'
summary(object, prior=1, significance=0.9, ...)
## S4 method for signature 'MultiWilcoxonTest'
hist(x, xlab='Rank Sum',
 ylab='Prob(Different | Y)', main='', ...)
## S4 method for signature 'MultiWilcoxonTest,missing'
plot(x, prior=1, significance=0.9,
 ylim=c(-0.5, 1), xlab='Rank Sum', ylab='Prob(Different | Y)', ...)
## S4 method for signature 'MultiWilcoxonTest'
cutoffSignificant(object, prior, significance, ...)
## S4 method for signature 'MultiWilcoxonTest'
selectSignificant(object, prior, significance, ...)
## S4 method for signature 'MultiWilcoxonTest'
countSignificant(object, prior, significance, ...)
```

Arguments

`data`	either a data frame or matrix with numeric values, or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data.
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`histsize`	An integer; the number of bins used for the histogram summarizing the Wilcoxon statistics. When `NULL`, each discrete rank-sum value gets its own bin.

`object`	an object of the `MultiWilcoxonTest` class.
`x`	an object of the `MultiWilcoxonTest` class.
`xlab`	character string specifying label for the x axis
`ylab`	character string specifying label for the y axis
`ylim`	Plotting limits on the y-axis
`main`	character string specifying graph title
`prior`	Prior probability that an arbitrary gene is not differentially expressed, or that an arbitrary row does not yield a significant Wilcoxon rank-sum statistic.
`significance`	Desired level of posterior probability
`...`	extra arguments for generic or plotting routines

Details

See the paper by Efron and Tibshirani.

Value

The standard methods `summary`, `hist`, and `plot` return what you would expect.

The `cutoffSignificant` method returns a list of two integers. Rank-sum values smaller than the first value or larger than the second value are statistically significant in the sense that their posterior probability exceeds the specified `significance` level given the assumptions about the `prior` probability of not being significant.

The `selectSignificant` method returns a vector of logical values identifying the significant test results, and `countSignificant` returns an integer counting the number of significant test results.

Creating Objects

As usual, objects can be created by `new`, but better methods are available in the form of the `MultiWilcoxonTest` function. The inputs to this function are the same as those used for row-by-row statistical tests throughout the ClassComparison package; a detailed description can be found in the `MultiTtest` class.

The constructor computes row-by-row Wilcoxon rank-sum statistics on the input `data`, comparing the two groups defined by the `classes` argument. It also estimates the observed and theoretical (expected) density functions for the collection of rank-sum statistics.

The additional input argument, `histsize` is usually best left to its default value. In certain pathological cases, we have found it necessary to use fewer bins; one suspects that the underlying model does not adequately capture the complexity of those situations.

Slots

`rank.sum.statistics`**:** numeric vector containing the computed rank-sum statistics.

`xvals`**:** numeric vector, best thought of as the vector of possible rank-sum statistics given the sizes of the two groups.

`theoretical.pdf`**:** numeric vector containing the theoretical density function evaluated at the points of `xvals`.

`pdf`**:** numeric vector containing the empirical density function computed at the points of `xvals`.

`unravel`**:** numeric vector containing a smoothed estimate (by Poisson regression using B-splines) of the empirical density function evaluated at `xvals`.

`groups`**:** A vector containing the names of the groups defined by `classes`.

`call`**:** object of class `call` representing the function call that created the object.

Methods

summary(object, prior=1, significance=0.9, . . .) Write out a summary of the object. For a given value of the `prior` probability of not being differentially expressed and a given significance cutoff on the posterior probability, reports the cutoffs and number of items in both tails of the distribution.

hist(x, xlab='Rank Sum', ylab='Prob(Different|Y)', main='', . . .) Plot a histogram of the rank-sum statistics, with overlaid curves representing the expected and observed distributions. Colors of the curves are controlled by `oompaColor$EXPECTED` and `oompaColor$OBSERVED`.

plot(x, prior=1, significance=0.9, ylim=c(-0.5, 1), xlab='Rank Sum', ylab='Prob(Different | Y)', . . .) Plots the posterior probability of being differentially expressed for given values of the `prior`. Horizontal lines are added at each specified `significance` level for the posterior probability.

cutoffSignificant(object, prior, significance, . . .) Determine cutoffs on the rank-sum statistic at the desired significance level.

selectSignificant(object, prior, significance, . . .) Compute a logical vector for selecting significant test results.

countSignificant(object, prior, significance, . . .) Count the number of significant test results.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Efron B, Tibshirani R.
Empirical bayes methods and false discovery rates for microarrays.
Genet Epidemiol 2002, 23: 70-86.

Pounds S, Morris SW.
Estimating the occurrence of false positives and false negatives in microarray studies by approximating and partitioning the empirical distribution of p-values.
Bioinformatics. 2003 Jul 1;19(10):1236-42.

See Also

Implementation is handled in part by the functions `dwil` and `rankSum`. The empirical Bayes results for alternative tests (such as `MultiTtest`) can be obtained using the beta-uniform mixture model in the `Bum` class.

Examples

```
showClass("MultiWilcoxonTest")
ng <- 10000
ns <- 15
nd <- 200
fake.class <- factor(rep(c('A', 'B'), each=ns))
fake.data <- matrix(rnorm(ng*ns*2), nrow=ng, ncol=2*ns)
fake.data[1:nd, 1:ns] <- fake.data[1:nd, 1:ns] + 2
fake.data[(nd+1):(2*nd), 1:ns] <- fake.data[(nd+1):(2*nd), 1:ns] - 2

a <- MultiWilcoxonTest(fake.data, fake.class)
hist(a)
plot(a)
plot(a, prior=0.85)
abline(h=0)

cutoffSignificant(a, prior=0.85, signif=0.95)
countSignificant(a, prior=0.85, signif=0.95)
```

`rankSum` *Wilcoxon Rank-Sum Statistic*

Description

Compute the Wilcoxon rank-sum statistic.

Usage

```
rankSum(data, selector)
```

Arguments

`data`	numeric vector
`selector`	logical vector the same length as `data`

Details

This is an efficient function to compute the value of the Wilcoxon rank-sum statistic without the extra overhead of the full `wilcox.test` function. It is used internally by the `MultiWilcoxonTest` class to perform row-by-row Wilcoxon tests.

Value

Returns an integer, the rank-sum of the subset of the `data` for which the `selector` is true.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

See Also

dwil, MultiWilcoxonTest

Examples

```
dd <- rnorm(100)
cc <- rep(c(TRUE, FALSE), each=50)
rankSum(dd, cc)
```

Sam-class *Class "Sam"*

Description

Implements the "Significance Analysis of Microarrays" approach to detecting differentially expressed genes.

Usage

```
Sam(data, classes, nPerm=100, verbose=TRUE)
## S4 method for signature 'Sam,missing'
plot(x, y, tracks=NULL, xlab='Expected T Statistics (Empirical)',
 ylab='Observed T Statistics', ...)
## S4 method for signature 'Sam'
summary(object, cutoff=1, ...)
## S4 method for signature 'Sam'
selectSignificant(object, cutoff=1, ...)
## S4 method for signature 'Sam'
countSignificant(object, cutoff=1, ...)
```

Arguments

`data`	Either a data frame or matrix with numeric values or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data.
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`nPerm`	An integer; the number of permutations
`verbose`	A logical flag
`x`	A `Sam` object
`y`	Nothing, since it is supposed to be missing. Changes to the Rd processor require documenting the missing entry.
`tracks`	a numeric vector
`xlab`	Label for the x axis
`ylab`	Label for the y axis

`object`	A `Sam` object
`cutoff`	A numeric value
`...`	The usual extra arguments to generic functions

Details

The SAM approach to analyzing microarray data was developed by Tusher and colleagues; their implementation is widely available. This is an independent implementation based on the description in their original paper, customized to use the same interface (and thus work with `ExpressionSet` objects) used by the rest of the ClassComparison package. The fundamental idea behind SAM is that the observed distribution of row-by-row two-sample t-tests should be compared not to the theoretical null distribution but to a null distribution estimated by a permutation test. The `Sam` constructor performs the permutation test.

Value

`summary` returns an object of class `SamSummary`.

`selectSignificant` returns a vector of logical values.

`countSignificant` returns an integer.

Creating Objects

As usual, objects can be created by `new`, but better methods are available in the form of the `Sam` function. The inputs to this function are the same as those used for row-by-row statistical tests throughout the ClassComparison package; a detailed description can be found in the `MultiTtest` class.

Slots

`t.statistics`**:** numeric vector containing the observed t-statistics.

`observed`**:** numeric vector containing the sorted observed t-statistics.

`expected`**:** numeric vector of the expected distribution of t-statistics based on a permutation test.

`sim.data`**:** numeric matrix containing all the t-statistics from all the permutations.

`call`**:** object of class `call` specifying the function call that was used to create this object.

Methods

summary(object, cutoff=1, ...) Compute a summary of the object.

plot(x, tracks=NULL, xlab='Expected T Statistics (Empirical)', ylab='Observed t Statistics', ...) Plot the observed and expected t-statistics. The `tracks` argument causes parallel lines to be drawn on either side of the quantile-quantile central line, at the specified offsets. Colors in the plot are controlled by the current values of `oompaColor$CENTRAL.LINE` and `oompaColor$CONFIDENCE.CURVE`

selectSignificant(object, cutoff=1, ...) Compute a vector that selects significant values

countSignificant(object, cutoff=1, ...) Count the number of significant values

Author(s)

Kevin R. Coombes <krc@silicovore.com>

References

Tusher VG, Tibshirani R, Chu G.
Significance analysis of microarrays applied to the ionizing radiation response.
Proc Natl Acad Sci U S A (2001) 98, 5116-5121.

See Also

Bum, MultiTtest

Examples

```
showClass("Sam")
ng <- 10000
ns <- 50
nd <- 100
dat <- matrix(rnorm(ng*ns), ncol=ns)
dat[1:nd, 1:(ns/2)] <- dat[1:nd, 1:(ns/2)] + 2
dat[(nd+1):(2*nd), 1:(ns/2)] <- dat[(nd+1):(2*nd), 1:(ns/2)] - 2
cla <- factor(rep(c('A', 'B'), each=25))

res <- Sam(dat, cla)
plot(res)

plot(res, tracks=1:3)

summary(res)
summary(res, cutoff=2)

a <- summary(res)
plot(a@significant.calls)
plot(a@significant.calls[1:300])

countSignificant(res, 1)
```

SamSummary-class *Class "SamSummary"*

Description

An implementation class. Users are not expected to create these objects directly; they are produced as return objects from the summary method for Sam.

Slots

`fdr`: numeric scalar between `0` and `1` specifying the expected false discovery rate

`hi`: Upper threshold for significance

`lo`: Lower threshold for significance

`cutoff`: numeric scalar specified in the call to the `Sam` `summary` method.

`significant.calls`: vector of logical values

`average.false.count`: The average number of false positives in the permuted data at this cutoff level.

Methods

show `signature(object = SamSummary)`: Print the object, which contains a summary of the underlying `Sam` object.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

`Sam`

Examples

```
showClass("SamSummary")
```

`significant` *Generic Methods for Significance*

Description

In the world of multiple testing that is inhabited by most microarray or protein profiling experiments, analysts frequently perform separate statistical tests for each gene or protein in the experiment. Determining cutoffs that achieve statistical significance (in a meaningful way) is an inherent part of the procedure. It is then common to select the significant items for further processing or for preparing reports, or at least to count the number of significant items. These generic functions provide a standard set of tools for selecting and counting the significant items, which can be used with various statistical tests and various ways to account for multiple testing.

Usage

```
## S4 method for signature 'ANY'
cutoffSignificant(object, ...)
## S4 method for signature 'ANY'
selectSignificant(object, ...)
## S4 method for signature 'ANY'
countSignificant(object, ...)
```

Arguments

`object`	an object that performs multiple statistical tests on microarray or proteomics data
`...`	additional arguments affecting these generic methods

Value

`cutoffSignificant` returns appropriate cutoff values that achieve specified significance criteria.

`selectSignificant` returns a logical vector, with `TRUE` values indicating items that satisfy the cutoff making them statistically significant.

`countSignificant` returns an integer, representing the number of significant items.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

`SingleGroup-class` *Class "SingleGroup"*

Description

Preliminary analysis of one group of samples for use in the `SmoothTtest` class. A key feature is the standard quality control plot.

Usage

```
SingleGroup(avg, sd, span=0.5, name='')
## S4 method for signature 'SingleGroup'
as.data.frame(x, row.names=NULL, optional=FALSE)
## S4 method for signature 'SingleGroup'
summary(object, ...)
## S4 method for signature 'SingleGroup'
print(x, ...)
## S4 method for signature 'SingleGroup'
show(object)
## S4 method for signature 'SingleGroup,missing'
plot(x, multiple=3, ccl=0, main=x@name,
 xlab='Mean', ylab='Std Dev', xlim=0, ylim=0, ...)
```

Arguments

`avg`	numeric vector of mean values
`sd`	numeric vector of standard deviations
`span`	parameter is passed onto `loess`
`name`	character string specifying the name of this object

`object`	object of class `SingleGroup`
`x`	object of class `SingleGroup`
`multiple`	numeric scalar specifying the multiple of the smoothed standard deviation to call significant
`ccl`	list containing objects of the `ColorCoding` class. If left at its default value of zero, colors are chosen automatically.
`main`	character string specifying plot title
`xlab`	character string specifying label for the x axis
`ylab`	character string specifying label for the y axis
`xlim`	Plotting limits for the x axis. If left at the default value of zero, then the limits are automatically generated
`ylim`	Plotting limits for the y axis. If left at the default value of zero, then the limits are automatically generated
`row.names`	See the base version of `as.data.frame.default`
`optional`	See the base version of `as.data.frame.default`
`...`	extra arguments for generic or plotting routines

Details

In 2001 and 2002, Baggerly and Coombes developed the smooth t-test for finding differentially expressed genes in microarray data. Along with many others, they began by log-transforming the data as a reasonable step in the direction of variance stabilization. They observed, however, that the gene-by-gene standard deviations still seemed to vary in a systematic way as a function of the mean log intensity. By borrowing strength across genes and using `loess` to fit the observed standard deviations as a function of the mean, one presumably got a better estimate of the true standard deviation.

Creating Objects

Objects can be created by calls to the `SingleGroup` constructor. Users rarely have need to create these objects directly; they are usually created as a consequence of the construction of an object of the `SmoothTtest` class.

Slots

`name`**:** character string specifying the name of this object

`avg`**:** numeric vector of mean values

`sd`**:** numeric vector of standard deviations

`span`**:** parameter used in the `loess` function to fit `sd` as a function of `avg`.

`fit`**:** list containing components `x` and `y` resulting from the `loess` fit

`score`**:** numeric vector specifying the ratio of the pointwise standard deviations to their smooth (loess) estimates

Methods

as.data.frame(x, row.names=NULL, optional=FALSE) Combine the slots containing numeric vectors into a data frame, suitable for printing or exporting.

summary(object, . . .) Write out a summary of the object.

print(x, . . .) Print the entire object. You never want to do this.

show(object) Print the entire object. You never want to do this.

plot(x, multiple=3, ccl=0, main=x@name, xlab='Mean', ylab='Std Dev', xlim=0, ylim=0, . . .) Produce a scatter plot of the standard deviations (`x@sd`) as a function of the means (`x@avg`). The appropriate multiple of the `loess` fit is overlaid, and points that exceed this multiple are flagged in a different color. Colors in the plot are controlled by the current values of `oompaColor$CENTRAL.LINE`, `oompaColor$CONFIDENCE.CURVE`, `oompaColor$BORING`, `oompaColor$BAD.REPLICATE`, and `oompaColor$WORST.REPLICATE`.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Baggerly KA, Coombes KR, Hess KR, Stivers DN, Abruzzo LV, Zhang W.
Identifying differentially expressed genes in cDNA microarray experiments.
J Comp Biol. 8:639-659, 2001.

Coombes KR, Highsmith WE, Krogmann TA, Baggerly KA, Stivers DN, Abruzzo LV.
Identifying and quantifying sources of variation in microarray data using high-density cDNA membrane arrays.
J Comp Biol. 9:655-669, 2002.

See Also

`SmoothTtest`

Examples

```
showClass("SingleGroup")
m <- rnorm(1000, 8, 2.5)
v <- rnorm(1000, 0.7)
plot(m, v)

x <- SingleGroup(m, v, name='bogus')

summary(x)

plot(x)
plot(x, multiple=2)
```

SmoothTtest-class *Class "SmoothTtest"*

Description

Implements the smooth t-test for differential expression as developed by Baggerly and Coombes.

Usage

```
SmoothTtest(stats, aname='Group One', bname='Group Two',
 name=paste(aname, 'vs.', bname))
## S4 method for signature 'SmoothTtest'
as.data.frame(x, row.names=NULL, optional=FALSE)
## S4 method for signature 'SmoothTtest'
summary(object, ...)
## S4 method for signature 'SmoothTtest,missing'
plot(x, folddiff=3, goodflag=2, badch=4, ccl=0,
 name=x@name, pch='.', xlab='log intensity', ylab='log ratio', ...)
```

Arguments

`stats`	object of class `TwoGroupStats`
`aname`	character string specifying the name of the first group
`bname`	character string specifying the name of the second group
`name`	character string specifying the name of this object
`object`	object of class `SmoothTtest`
`x`	object of class `SmoothTtest`
`row.names`	See the base version of `as.data.frame.default`
`optional`	See the base version of `as.data.frame.default`
`folddiff`	numeric scalar specifying the level of fold difference considered large enough to be indicated in the plots
`goodflag`	numeric scalar specifying the level (in standard deviation units) of the smooth t-statistic considered large enough to be indicated in the plot
`badch`	numeric scalar specifying the level of variability in single groups considered large enough to be worrisome. See the `multiple` argument to the `plot` method in the `SingleGroup` class.
`ccl`	list containing objects of class `ColorCoding`. If left at its default value of zero, colors are chosen automatically.
`pch`	default plotting character
`xlab`	character string specifying label for the x axis
`ylab`	character string specifying label for the y axis
`...`	extra arguments for generic or plotting routines

Details

In 2001 and 2002, Baggerly and Coombes developed the smooth t-test for finding differentially expressed genes in microarray data. Along with many others, they began by log-transforming the data as a reasonable step in the direction of variance stabilization. They observed, however, that the gene-by-gene standard deviations still seemed to vary in a systematic way as a function of the mean log intensity. By borrowing strength across genes and using `loess` to fit the observed standard deviations as a function of the mean, one presumably got a better estimate of the true standard deviation.

These smooth estimates are computed for each of two groups of samples being compared. They are then combined (gene-by-gene using the usual univariate formulas) to compute pooled "smooth" estimates of the standard deviation. These smooth estimates are then used in gene-by-gene t-tests.

The interesting question then arises of how to compute and interpret p-values associated to these individual tests. The liberal argument asserts that, because smoothing uses data from hundreds of measurements to estimate the standard deviation, it can effectively be treated as "known" in the t-tests, which should thus be compared against the normal distribution. A conservative argument claims that the null distribution should still be the t-distribution with the degrees of freedom determined in the usual way by the number of samples. The truth probably lies somewhere in between, and is probably best approximated by some kind of permutation test. In this implementation, we take the coward's way out and don't provide any of those alternatives. You have to extract the t-statistics (from the `smooth.t.statistics` slot of the object) and compute your own p-values in your favorite way. If you base the computations on a theoretical model rather than a permutation test, then the `Bum` class provides a convenient way to account for multiple testing.

Creating Objects

In practice, users will first use a data frame and a classification vector (or an `ExpressionSet`) to construct an object of the `TwoGroupStats` object. This object can then be handed directly to the `SmoothTtest` function to perform the smooth t-test.

Slots

`one`: object of class `SingleGroup` representing a loess smooth of standard deviation as a function of the mean in the first group of samples

`two`: object of class `SingleGroup` representing a loess smooth of standard deviation as a function of the mean in the second group of samples

`smooth.t.statistics`: numeric vector containing the smooth t-statistics

`fit`: data.frame with two columns, `x` and `y`, containing the smooth estimates of the pooled standard deviation

`dif`: numeric vector of the differences in mean values between the two groups

`avg`: numeric vector of the overall mean value

`aname`: character string specifying the name of the first group

`bname`: character string specifying the name of the second group

`name`: character string specifying the name of this object

`stats`: object of class `TwoGroupStats` that was used to create this object

Methods

as.data.frame(x, row.names=NULL, optional=FALSE) Convert the object into a data frame suitable for printing or exporting.

summary(object, ...) Write out a summary of the object.

plot(x, folddiff=3, goodflag=2, badch=4, ccl=0, name=x@name, pch='.', xlab='log intensity', ylab='log ratio', ...) Create a set of six plots. The first two plots are the QC plots from the `SingleGroup` objects representing the two groups of samples. The third plot is a scatter plot comparing the means in the two groups. The fourth plot is Bland-Altman plot of the overall mean against the difference in means (also known colloquially as an M-vs-A plot). The fifth plot is a histogram of the smooth t-statistics. The final plot is a scatter plot of the smooth t-statistics as a function of the mean intensity. Colors in the plots are controlled by the current values of `oompaColor$BORING`, `oompaColor$SIGNIFICANT`, `oompaColor$BAD.REPLICATE`, `oompaColor$WORST.REPLICATE`, `oompaColor$FOLD.DIFFERENCE`, `oompaColor$CENTRAL.LINE`, and `oompaColor$CONFIDENCE.CURVE`.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Baggerly KA, Coombes KR, Hess KR, Stivers DN, Abruzzo LV, Zhang W.
Identifying differentially expressed genes in cDNA microarray experiments.
J Comp Biol. 8:639-659, 2001.

Coombes KR, Highsmith WE, Krogmann TA, Baggerly KA, Stivers DN, Abruzzo LV.
Identifying and quantifying sources of variation in microarray data using high-density cDNA membrane arrays.
J Comp Biol. 9:655-669, 2002.

Altman DG, Bland JM.
Measurement in Medicine: the Analysis of Method Comparison Studies.
The Statistician, 1983; 32: 307-317.

See Also

`Bum`, `MultiTtest`, `SingleGroup`, `TwoGroupStats`

Examples

```
showClass("SmoothTtest")
bogus <- matrix(rnorm(30*1000, 8, 3), ncol=30, nrow=1000)
splitter <- rep(FALSE, 30)
splitter[16:30] <- TRUE
x <- TwoGroupStats(bogus, splitter)
y <- SmoothTtest(x)

opar <- par(mfrow=c(2, 3), pch='.')
plot(y, badch=2, goodflag=1)
par(opar)
```

TNoM-class *Classes "TNoM" and "fullTNoM"*

Description

Implements the "Total Number of Misclassifications" method for finding differentially expressed genes.

Usage

```
TNoM(data, classes, verbose=TRUE)
## S4 method for signature 'TNoM'
summary(object, ...)
## S4 method for signature 'TNoM'
update(object, nPerm, verbose=FALSE, ...)
## S4 method for signature 'TNoM'
selectSignificant(object, cutoff, ...)
## S4 method for signature 'TNoM'
countSignificant(object, cutoff, ...)
## S4 method for signature 'fullTNoM,missing'
plot(x, y, ...)
## S4 method for signature 'fullTNoM'
hist(x, ...)
```

Arguments

`data`	Either a data frame or matrix with numeric values or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data.
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`verbose`	logical scalar. If `TRUE`, print out intermediate results
`object`	object of class `TNoM`
`nPerm`	integer scalar specifying the number of permutations to perform
`cutoff`	integer scalar
`x`	object of class `fullTNoM`
`y`	Nothing, since it is supposed to be missing. Changes to the Rd processor require documenting the missing entry.
`...`	extra arguments to generic or plotting routines

Details

The TNoM method was developed by Yakhini and Ben-Dor and first applied in the melanoma microarray study by Bittner and colleagues (see references). The goal of the method is to detect genes that are differentially expressed between two groups of samples. The idea is that each gene serves as a potential classifier to distinguish the two groups. One starts by determining an optimal cutoff on the expression of each gene and counting the number of misclassifications that gene makes. Next, we bin genes based on the total number of misclassifications. This distribution can be compared with the expected value (by simulating normal data sets of the same size). Alternatively, one can estimate the null distribution directly by scrambling the sample labels to perform a permutation test.

The `TNoM` constructor computes the optimal cutoffs and the misclassification rates. The `update` method performs the simulations and permutation tests, producing an object of the `fullTNoM` class.

Value

`summary` returns a `TNoMSummary` object.

`update` returns a `fullTNoM` object.

`selectSignificant` returns a vector of logical values.

`countSignificant` returns an integer.

Creating Objects

Although objects of the class can be created by a direct call to new, the preferred method is to use the `TNoM` generator. The inputs to this function are the same as those used for row-by-row statistical tests throughout the **ClassComparison** package; a detailed description can be found in the `MultiTtest` class.

Slots

Objects of the `TNoM` class have the following slots:

`data`**:** The data matrix used to construct the object

`tnomData`**:** numeric vector, whose length is the number of rows in `data`, recording the minimum number of misclassification achieved using this data row.

`nCol`**:** The number of columns in `data`

`nRow`**:** The number of rows in `data`

`classifier`**:** The classification vector used to create the object.

`call`**:** The function `call` that created the object

Objects of the `fullTNoM` class have the following slots:

`dex`**:** Numeric vector of the different possible numbers of misclassifications

`fakir`**:** Numeric vector of expected values based on simulations

`obs`**:** Numeric vector of observed values

`scr`**:** Numeric vector of values based on a permutation test

`name`**:** A character string with a name for the object

Methods

Objects of the `TNoM` class have the following methods:

summary(object, ...) Write out a summary of the object, including the number of genes achieving each possible number of misclassifications.

countSignificant(object, cutoff, ...) Count the number of significant genes at the given `cutoff`.

selectSignificant(object, cutoff, ...) Get a vector for selecting the number of significant genes at the given `cutoff`.

update(object, nPerm, verbose=FALSE, ...) Perform simulation and permutation tests on the `TNoM` object.

Objects of the `fullTNoM` class have the following methods:

plot(x, ...) Plot a summary of the TNoM object. This consists of three curves: the observed cumulative number of genes at each misclassification level, along with the corresponding numbers expected based on simulations or permutation tests. The colors of the curves are controlled by the values of `oompaColor$OBSERVED`, `oompaColor$EXPECTED`, and `oompaColor$PERMTEST`

hist(x, ...) Plot a not terribly useful nor informative histogram of the results.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Bittner M, Meltzer P, Chen Y, Jiang Y, Seftor E, Hendrix M, Radmacher M, Simon R, Yakhini Z, Ben-Dor A, Sampas N, Dougherty E, Wang E, Marincola F, Gooden C, Lueders J, Glatfelter A, Pollock P, Carpten J, Gillanders E, Leja D, Dietrich K, Beaudry C, Berens M, Alberts D, Sondak V. *Molecular classification of cutaneous malignant melanoma by gene expression profiling.*
Nature. 2000 Aug 3;406(6795):536-40.

See Also

`Bum`, `MultiTtest`, `MultiWilcoxonTest`

Examples

```
showClass("TNoM")
showClass("fullTNoM")
n.genes <- 200
n.samples <- 10

bogus <- matrix(rnorm(n.samples*n.genes, 0, 3), ncol=n.samples)
splitter <- rep(FALSE, n.samples)
splitter[sample(1:n.samples, trunc(n.samples/2))] <- TRUE

tn <- TNoM(bogus, splitter)
summary(tn)

tnf <- update(tn)
```

```
plot(tnf)
hist(tnf)
```

TNoMSummary-class *Class "TNoMSummary"*

Description

An implementation class. Users are not expected to create these objects directly; they are produced as return objects from the summary method for `TNoM`.

Slots

`TNoM`**:** object of class `TNoM` ~~

`counts`**:** object of class `numeric` ~~

Methods

show `signature(object = TNoMSummary)`: Print the object, which contains a summary of the underlying `TNoM` object. In particular, the summary reports the number of genes achieving each possible number of misclassifications.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

See Also

`TNoM`

Examples

```
showClass("TNoMSummary")
```

TwoGroupStats-class *Class "TwoGroupStats"*

Description

Compute row-by-row means and variances for a data matrix whose columns belong to two different groups of interest.

Usage

```
TwoGroupStats(data, classes, name=comparison, name1=A, name2=B)
## S4 method for signature 'TwoGroupStats'
as.data.frame(x, row.names=NULL, optional=FALSE)
## S4 method for signature 'TwoGroupStats'
summary(object, ...)
## S4 method for signature 'TwoGroupStats'
print(x, ...)
## S4 method for signature 'TwoGroupStats'
show(object)
## S4 method for signature 'TwoGroupStats,missing'
plot(x, main=x@name, useLog=FALSE, ...)
```

Arguments

`data`	Either a data frame or matrix with numeric values or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data.
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`name`	A character string; the name of this object
`name1`	A character string; the name of the first group
`name2`	A character string; the name of the second group
`x`	A `TwoGroupStats` object
`row.names`	See the base version of `as.data.frame.default`
`optional`	See the base version of `as.data.frame.default`
`object`	A `TwoGroupStats` object
`main`	Plot title
`useLog`	a logical flag; should the values be log-transformed before plotting?
`...`	The usual extra arguments to generic functions

Details

This class was one of the earliest developments in our suite of tools to analyze microarrays. Its main purpose is to segregate out the preliminary computation of summary statistics on a row-by-row basis, along with a set of plots that could be generated automatically and used for quality control.

Creating Objects

Although objects of the class can be created by a direct call to new, the preferred method is to use the `TwoGroupStats` generator. The inputs to this function are the same as those used for row-by-row statistical tests throughout the ClassComparison package; a detailed description can be found in the `MultiTtest` class.

One should note that this class serves as the front end to the `SmoothTtest` class, providing it with an interface that accepts `ExpressionSet` objects compatible with the other statistical tests in the ClassComparison package.

Slots

`mean1`**:** numeric vector of means in the first group

`mean2`**:** numeric vector of means in the second group

`overallMean`**:** numeric vector of overall row means

`var1`**:** numeric vector of variances in the first group

`var2`**:** numeric vector of variances in the second group

`overallVar`**:** numeric vector of variances assuming the two groups have the same mean

`pooledVar`**:** numeric vector of row-by-row pooled variances, assuming the two groups have the same variance but different means

`n1`**:** numeric scalar specifying number of items in the first group

`n2`**:** numeric scalar specifying number of items in the second group

`name1`**:** character string specifying name of the first group

`name2`**:** character string specifying name of the second group

`name`**:** character string specifying name of the object

Methods

as.data.frame(x, row.names=NULL, optional=FALSE) Collect the numeric vectors from the object into a single dat fame, suitable for printing or exporting.

summary(object, ...) Write out a summary of the object.

print(x, ...) Print the object. (Actually, it only prints a summary, since the whole object is almost always more than you really want to see. If you insist on printing everything, use `as.data.frame`.)

show(object) Print the object (same as print method).)

plot(x, main=x@name, useLog=FALSE, ...) This function actually produces six different plots of the data, so it is usually wrapped by a graphical layout command like `par(mfrow=c(2,3))`. The first two plots show the relation between the mean and standard deviation for the two groups separately; the third plot does the same for the overall mean and variance. The fourth plot is a Bland-Altman plot of the difference between the means against the overall mean. (In the microarray world, this is usually called an M-vs-A plot.) A loess fit is overlaid on the scatter plot, and points outside confidence bounds based on the fit are printed in a different color to flag them as highly variable. The fifth plot shows a loess fit (with confidence bounds) of the difference as a function of the row index (which often is related to the geometric position of spots on a microarray). Thus, this plot gives a possible indication of regions of an array where unusual things happen. The final plot compares the overall variances to the pooled variances.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

References

Altman DG, Bland JM.
Measurement in Medicine: the Analysis of Method Comparison Studies.
The Statistician, 1983; 32: 307-317.

See Also

`MultiTtest`, `SmoothTtest`

Examples

```
showClass("TwoGroupStats")
bogus <- matrix(rnorm(30*1000, 8, 3), ncol=30, nrow=1000)
splitter <- rep(FALSE, 30)
splitter[16:30] <- TRUE

x <- TwoGroupStats(bogus, splitter)
summary(x)

opar<-par(mfrow=c(2,3), pch='.')
plot(x)
par(opar)
```

`variantT` *Classes for Variant T-tests*

Description

Classes to perform row-by-row paired or unequal variance t-tests on microarray or proteomics data.

Usage

```
MultiTtestPaired(data, classes, pairing)
MultiTtestUnequal(data, classes)
## S4 method for signature 'MultiTtestPaired'
summary(object, ...)
## S4 method for signature 'MultiTtestUnequal'
summary(object, ...)
```

Arguments

`data`	Either a data frame or matrix with numeric values or an `ExpressionSet` as defined in the BioConductor tools for analyzing microarray data.
`classes`	If `data` is a data frame or matrix, then classes must be either a logical vector or a factor. If `data` is an `ExpressionSet`, then `classes` can be a character string that names one of the factor columns in the associated `phenoData` subobject.
`pairing`	A numerical vector indicating which samples are paired.
`object`	A `MultiTtest` object
`...`	Unused; optional extra parameters for `summary`.

Creating objects

Although objects can be created using new, the better method is to use the MultiTtestPaired or MultiTtestUnequal functions. In the simplest case, you simply pass in a data matrix and a logical vector assigning classes to the columns (and, in the case of a paired t-test, a numeric vector describing the pairing), and the constructor performs row-by-row two-sample t-tests and computes the associated (single test) p-values. To adjust for multiple testing, you can pass the p-values on to the Bum class.

If you use a factor instead of a logical vector, then the t-test compares the first level of the factor to everything else. To handle the case of multiple classes, see the MultiLinearModel class.

As with other class comparison functions that are part of the OOMPA, we can also perform statistical tests on ExpressionSet objects from the BioConductor libraries. In this case, we pass in an ExpressionSet object along with the name of a factor to use for splitting the data.

Extends

Both classes extend class MultiTtest, directly. See that class for descriptions of the inherited methods and slots.

Slots

df**:** The MultiTtestUnequal class adds a slot to record e gene-by-gene degrees of freedom, which can change along with the variances.

Methods

summary signature(object = MultiTtestPaired): Write out a summary of the object.

summary signature(object = MultiTtestUnequal): Write out a summary of the object.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

References

OOMPA

See Also

Bum, MultiTtest

Examples

```
showClass("MultiTtestPaired")
showClass("MultiTtestUnequal")
ng <- 10000
ns <- 50
dat <- matrix(rnorm(ng*ns), ncol=ns)
cla <- factor(rep(c('A', 'B'), each=25))
res <- MultiTtestUnequal(dat, cla)
```

```
summary(res)
hist(res, breaks=101)
plot(res, res@p.values)

pairing <- rep(1:25, 2)
res <- MultiTtestPaired(dat, cla, pairing)
summary(res)
plot(res)
hist(res@p.values, breaks=101)
```

Index