genefilter

November 11, 2009

R topics documented:

`allNA` *A filter function to determine if all elements of a vector are NA.*

Description

`allNA` evaluates to `FALSE` if all elements of its argument are `NA`. `anyNA` evaluates to `FALSE` if any of the elements of its argument are `NA`.

Usage

```
allNA(x)
anyNA(x)
```

Arguments

x	The vector to test.

Value

FALSE if all elements of x are NA.

Author(s)

R. Gentleman

See Also

pOverA

Examples

```
allNA(NA)
allNA(1)
anyNA(1)
anyNA(NA)
```

Anova *A filter function for Analysis of Variance*

Description

Anova returns a function of one argument with bindings for cov and p. The function, when evaluated, performs an ANOVA using cov as the covariate. It returns TRUE if the p value for a difference in means is less than p.

Usage

```
Anova(cov, p=0.05, na.rm=TRUE)
```

Arguments

cov	The covariate. It must have length equal to the number of columns of the array that Anova will be applied to.
p	The p-value for the test.
na.rm	If set to TRUE any NA's will be removed.

Details

The function returned by Anova uses lm to fit a linear model of the form lm(x ~ cov), where x is the set of gene expressions. The F statistic for an overall effect is computed and if it has a *p*-value less than p the function returns TRUE, otherwise it returns FALSE for that gene.

Value

Anova returns a function with bindings for cov and p that will perform a one-way ANOVA.

The covariate can be continuous, in which case the test is for a linear effect for the covariate.

Author(s)

R. Gentleman

See Also

kOverA, lm

Examples

```
set.seed(123)
af <- Anova(c(rep(1,5),rep(2,5)), .01)
af(rnorm(10))
```

---

coxfilter *A filter function for univariate Cox regression.*

Description

A function that performs Cox regression with bindings for surt, cens, and p is returned. This function filters genes according to the attained p-value from a Cox regression using surt as the survival times, and cens as the censoring indicator. It requires survival.

Usage

```
coxfilter(surt, cens, p)
```

Arguments

surt	Survival times.
cens	Censoring indicator.
p	The p-value to use in filtering.

Value

Calls to the coxph function in the survival library are used to fit a Cox model. The filter function returns TRUE if the p-value in the fit is less than p.

Author(s)

R. Gentleman

See Also

Anova

Examples

```
set.seed(-5)
sfun <- coxfilter(rexp(10), ifelse(runif(10) < .7, 1, 0), .05)
ffun <- filterfun(sfun)
dat <- matrix(rnorm(1000), ncol=10)
out <- genefilter(dat, ffun)
```

---

cv *A filter function for the coefficient of variation.*

Description

cv returns a function with values for a and b bound. This function takes a single argument. It computes the coefficient of variation for the input vector and returns TRUE if the coefficient of variation is between a and b. Otherwise it returns FALSE

Usage

```
cv(a=1, b=Inf, na.rm=TRUE)
```

Arguments

a	The lower bound for the cv.
b	The upper bound for the cv.
na.rm	If set to TRUE any NA's will be removed.

Details

The coefficient of variation is the standard deviation divided by the absolute value of the mean.

Value

It returns a function of one argument. The function has an environment with bindings for a and b.

Author(s)

R. Gentleman

See Also

pOverA, kOverA

Examples

```
set.seed(-3)
cvfun <- cv(1,10)
cvfun(rnorm(10,10))
cvfun(rnorm(10))
```

dist2 — *Calculate an n-by-n matrix by applying a function to pairs of columns of an m-by-n matrix.*

Description

Calculate an n-by-n matrix by applying a function to pairs of columns of an m-by-n matrix.

Usage

```
dist2(x, fun=function(a,b) mean(abs(a-b), na.rm=TRUE), diagonal=0)
```

Arguments

x	A matrix, or any object x for which ncol(x) and x[,j] return appropriate results.
fun	A symmetric function of two arguments that may be columns of x.
diagonal	The value to be used for the diagonal elements of the resulting matrix.

Details

With the default value of fun, this function calculates for each pair of columns of x the mean of the absolute values of their differences (which is proportional to the L1-norm of their difference). This is a distance metric.

The implementation assumes that fun is symmetric, fun(a,b)=fun(b,a). Hence, the returned matrix is symmetric. fun(a,a) is not evaluated, instead the value of diagonal is used to fill the diagonal elements of the returned matrix.

A use for this function is the detection of outlier arrays in a microarray experiment. Assume that each column of x can be decomposed as $z + \beta + \epsilon$, where z is a fixed vector (the same for all columns), ϵ is vector of nrow{x} i.i.d. random numbers, and β is an arbitrary vector whose majority of entries are negligibly small (i.e. close to zero). In other words, Dz the probe effects, ϵ measurement noise and β differential expression effects. Under this assumption, all entries of the resulting distance matrix should be the same, namely a multiple of the standard deviation of ϵ. Arrays whose distance matrix entries are way different give cause for suspicion.

Value

A symmetric matrix of size n x n.

Examples

```
z = matrix(rnorm(15693), ncol=3)
dist2(z)
```

`eSetFilter` *A function to filter an eSet object*

Description

Given a Bioconductor's ExpressionSet object, this function filters genes using a set of selected filters.

Usage

```
eSetFilter(eSet)
getFilterNames()
getFuncDesc(lib = "genefilter", funcs = getFilterNames())
getRdAsText(lib)
parseDesc(text)
parseArgs(text)
showESet(eSet)
setESetArgs(filter)
isESet(eSet)
```

Arguments

`eSet`	`eSet` an ExpressionSet object
`lib`	`lib` a character string for the name of an R library where functions of interests reside
`funcs`	`funcs` a vector of character strings for names of functions of interest
`text`	`text` a character of string from a filed (e. g. description, argument, ..) filed of an Rd file for a fucntion
`filter`	`filter` a character string for the name of a filter function

Details

A set of filters may be selected to filter genes in through each of the filters in the order the filters have been selected

Value

A logical vector of length equal to the number of rows of 'expr'. The values in that vector indicate whether the corresponding row of 'expr' passed the set of filter functions.

Author(s)

Jianhua Zhang

See Also

`genefilter`

Examples

```
if( interactive() ) {
  data(sample.ExpressionSet)
  res <- eSetFilter(sample.ExpressionSet)
}
```

`filterfun` *Creates a first FALSE exiting function from the list of filter functions it is given.*

Description

This function creates a function that takes a single argument. The filtering functions are bound in the environment of the returned function and are applied sequentially to the argument of the returned function. When the first filter function evaluates to `FALSE` the function returns `FALSE` otherwise it returns `TRUE`.

Usage

```
filterfun(...)
```

Arguments

`...`	Filtering functions.

Value

`filterfun` returns a function that takes a single argument. It binds the filter functions given to it in the environment of the returned function. These functions are applied sequentially (in the order they were given to `filterfun`). The function returns `FALSE` (and exits) when the first filter function returns `FALSE` otherwise it returns `TRUE`.

Author(s)

R. Gentleman

See Also

`genefilter`

Examples

```
set.seed(333)
x <- matrix(rnorm(100,2,1),nc=10)
cvfun <- cv(.5,2.5)
ffun <- filterfun(cvfun)
which <- genefilter(x, ffun)
```

`findLargest` *Find the Entrez Gene ID corresponding to the largest statistic*

Description

Most microarrays have multiple probes per gene (Entrez). This function finds all replicates, and then selects the one with the largest value of the test statistic.

Usage

```
findLargest(gN, testStat, data = "hgu133plus2")
```

Arguments

`gN`	A vector of probe identifiers for the chip.
`testStat`	A vector of test statistics, of the same length as `gN` with the per probe test statistics.
`data`	The character string identifying the chip.

Details

All the probe identifiers, `gN`, are mapped to Entrez Gene IDs and the duplicates determined. For any set of probes that map to the same Gene ID, the one with the largest test statistic is found. The return vector is the named vector of selected probe identifiers. The names are the Entrez Gene IDs.

This could be extended in different ways, such as allowing the user to use a different selection criterion. Also, matching on different identifiers seems like another alternative.

Value

A named vector of probe IDs. The names are Entrez Gene IDs.

Author(s)

R. Gentleman

See Also

`sapply`

Examples

```
library("hgu95av2.db")
set.seed(124)
gN <- sample(ls(hgu95av2ENTREZID), 200)
stats <- rnorm(200)
findLargest(gN, stats, "hgu95av2")
```

gapFilter *A filter to select genes based on there being a gap.*

Description

The gapFilter looks for genes that might usefully discriminate between two groups (possibly unknown at the time of filtering). To do this we look for a gap in the ordered expression values. The gap must come in the central portion (we exclude jumps in the initial Prop values or the final Prop values). Alternatively, if the IQR for the gene is large that will also pass our test and the gene will be selected.

Usage

```
gapFilter(Gap, IQR, Prop, na.rm=TRUE, neg.rm=TRUE)
```

Arguments

Gap	The size of the gap required to pass the test.
IQR	The size of the IQR required to pass the test.
Prop	The proportion (or number) of samples to exclude at either end.
na.rm	If TRUE then NA's will be removed before processing.
neg.rm	If TRUE then negative values in x will be removed before processing.

Details

As stated above we are interested in

Value

A function that returns either TRUE or FALSE depending on whether the vector supplied has a gap larger than Gap or an IQR (inter quartile range) larger than IQR. For computing the gap we want to exclude a proportion, Prop from either end of the sorted values. The reason for this requirement is that genes which differ in expression levels only for a few samples are not likely to be interesting.

Author(s)

R. Gentleman

See Also

ttest, genefilter

Examples

```
set.seed(256)
x <- c(rnorm(10,100,3), rnorm(10, 100, 10))
y <- x + c(rep(0,10), rep(100,10))
tmp <- rbind(x,y)
Gfilter <- gapFilter(200, 100, 5)
ffun <- filterfun(Gfilter)
genefilter(tmp, ffun)
```

genefilter *A function to filter genes.*

Description

genefilter filters genes in the array expr using the filter functions in flist. It returns an array of logical values (suitable for subscripting) of the same length as there are rows in expr. For each row of expr the returned value is TRUE if the row passed all the filter functions. Otherwise it is set to FALSE.

Usage

```
genefilter(expr, flist)
```

Arguments

expr	A matrix or ExpressionSet that the filter functions will be applied to.
flist	A list of filter functions to apply to the array.

Details

This package uses a very simple but powerful protocol for *filtering* genes. The user simply constructs any number of tests that they want to apply. A test is simply a function (as constructed using one of the many helper functions in this package) that returns TRUE if the gene of interest passes the test (or filter) and FALSE if the gene of interest fails.

The benefit of this approach is that each test is constructed individually (and can be tested individually). The tests are then applied sequentially to each gene. The function returns a logical vector indicating whether the gene passed all tests functions or failed at least one of them.

Users can construct their own filters. These filters should accept a vector of values, corresponding to a row of the expr object. The user defined function should return a length 1 logical vector, with value TRUE or FALSE. User-defined functions can be combined with filterfun, just as built-in filters.

Value

A logical vector of length equal to the number of rows of expr. The values in that vector indicate whether the corresponding row of expr passed the set of filter functions.

Author(s)

R. Gentleman

See Also

genefilter, kOverA

Examples

```
set.seed(-1)
f1 <- kOverA(5, 10)
flist <- filterfun(f1, allNA)
exprA <- matrix(rnorm(1000, 10), ncol = 10)
ans <- genefilter(exprA, flist)
```

`genefinder` *Finds genes that have similar patterns of expression.*

Description

Given an `ExpressionSet` or a `matrix` of gene expressions, and the indices of the genes of interest, `genefinder` returns a `list` of the `numResults` closest genes. The user can specify one of the standard distance measures listed below. The number of values to return can be specified. The return value is a `list` with two components: genes (measured through the desired distance method) to the genes of interest (where X is the number of desired results returned) and their distances.

Usage

```
genefinder(X, ilist, numResults=25, scale="none", weights, method="euclidean")
```

Arguments

`X`	A numeric `matrix` where columns represent patients and rows represent genes.
`ilist`	A `vector` of genes of interest. Contains indices of genes in matrix X.
`numResults`	Number of results to display, starting from the least distance to the greatest.
`scale`	One of "none", "range", or "zscore". Scaling is carried out separately on each row.
`weights`	A vector of weights applied across the columns of `X`. If no weights are supplied, no weights are applied.
`method`	One of "euclidean", "maximum", "manhattan", "canberra", "correlation", "binary".

Details

If the `scale` option is "range", then the input matrix is scaled using `genescale()`. If it is "zscore", then the input matrix is scaled using the `scale` builtin with no arguments.

The method option specifies the metric used for gene comparisons. The metric is applied, row by row, for each gene specified in `ilist`.

The "correlation" option for the distance method will return a value equal to 1-correlation(x).

See `dist` for a more detailed description of the distances.

Value

The returned value is a `list` containing an entry for each gene specified in `ilist`. Each `list` entry contains an array of distances for that gene of interest.

Author(s)

J. Gentry and M. Kajen

See Also

genescale

Examples

```
set.seed(12345)

#create some fake expression profiles
m1 <- matrix (1:12, 4, 3)
v1 <- 1
nr <- 2

#find the 2 rows of m1 that are closest to row 1
genefinder (m1, v1, nr, method="euc")

v2 <- c(1,3)
genefinder (m1, v2, nr)

genefinder (m1, v2, nr, scale="range")

genefinder (m1, v2, nr, method="manhattan")

m2 <- matrix (rnorm(100), 10, 10)
v3 <- c(2, 5, 6, 8)
nr2 <- 6
genefinder (m2, v3, nr2, scale="zscore")
```

---

genescale *Scales a matrix or vector.*

Description

genescale returns a scaled version of the input matrix m by applying the following formula to each column of the matrix:

$$y[i] = (x[i] - min(x))/(max(x) - min(x))$$

Usage

```
genescale(m, axis=2, method=c("Z", "R"), na.rm=TRUE)
```

Arguments

m	Input a matrix or a vector with numeric elements.
axis	An integer indicating which axis of m to scale.
method	Either "Z" or "R", indicating whether a Z scaling or a range scaling should be performed.
na.rm	A boolean indicating whether NA's should be removed.

Details

Either the rows or columns of m are scaled. This is done either by subtracting the mean and dividing by the standard deviation ("Z") or by subtracing the minimum and dividing by the range.

Value

A scaled version of the input. If m is a matrix or a dataframe then the dimensions of the returned value agree with that of m, in both cases the returned value is a matrix.

Author(s)

R. Gentleman

See Also

genefinder,scale

Examples

```
m <- matrix(1:12, 4, 3)
genescale(m)
```

---

half.range.mode *Mode estimation for continuous data*

Description

For data assumed to be drawn from a unimodal, continuous distribution, the mode is estimated by the "half-range" method. Bootstrap resampling for variance reduction may optionally be used.

Usage

```
half.range.mode(data, B, B.sample, beta = 0.5, diag = FALSE)
```

Arguments

data	A numeric vector of data from which to estimate the mode.
B	Optionally, the number of bootstrap resampling rounds to use. Note that B = 1 resamples 1 time, whereas omitting B uses data as is, without resampling.
B.sample	If bootstrap resampling is requested, the size of the bootstrap samples drawn from data. Default is to use a sample which is the same size as data. For large data sets, this may be slow and unnecessary.
beta	The fraction of the remaining range to use at each iteration.
diag	Print extensive diagnostics. For internal testing only... best left FALSE.

Details

Briefly, the mode estimator is computed by iteratively identifying densest half ranges. (Other fractions of the current range can be requested by setting `beta` to something other than 0.5.) A densest half range is an interval whose width equals half the current range, and which contains the maximal number of observations. The subset of observations falling in the selected densest half range is then used to compute a new range, and the procedure is iterated. See the references for details.

If bootstrapping is requested, `B` half-range mode estimates are computed for `B` bootstrap samples, and their average is returned as the final estimate.

Value

The mode estimate.

Author(s)

Richard Bourgon <bourgon@stat.berkeley.edu>

References

- DR Bickel, "Robust estimators of the mode and skewness of continuous data." *Computational Statistics & Data Analysis* 39:153-163 (2002).
- SB Hedges and P Shah, "Comparison of mode estimation methods and application in molecular clock analysis." *BMC Bioinformatics* 4:31-41 (2003).

See Also

`shorth`

Examples

```
## A single normal-mixture data set

x <- c( rnorm(10000), rnorm(2000, mean = 3) )
M <- half.range.mode( x )
M.bs <- half.range.mode( x, B = 100 )

if(interactive()){
hist( x, breaks = 40 )
abline( v = c( M, M.bs ), col = "red", lty = 1:2 )
legend(
       1.5, par("usr")[4],
       c( "Half-range mode", "With bootstrapping (B = 100)" ),
       lwd = 1, lty = 1:2, cex = .8, col = "red"
       )
}

# Sampling distribution, with and without bootstrapping

X <- rbind(
           matrix( rnorm(1000 * 100), ncol = 100 ),
           matrix( rnorm(200 * 100, mean = 3), ncol = 100 )
           )
M.list <- list(
               Simple = apply( X, 2, half.range.mode ),
```

```
                BS = apply( X, 2, half.range.mode, B = 100 )
                )

if(interactive()){
boxplot( M.list, main = "Effect of bootstrapping" )
abline( h = 0, col = "red" )
}
```

---

kOverA *A filter function for k elements larger than A.*

Description

kOverA returns a filter function with bindings for k and A. This function evaluates to TRUE if at least k of the arguments elements are larger than A.

Usage

```
kOverA(k, A=100, na.rm=TRUE)
```

Arguments

A	The value you want to exceed.
k	The number of elements that have to exceed A.
na.rm	If set to TRUE any NA's will be removed.

Value

A function with bindings for A and k.

Author(s)

R. Gentleman

See Also

pOverA

Examples

```
fg <- kOverA(5, 100)
fg(90:100)
fg(98:110)
```

maxA — *A filter function to filter according to the maximum.*

Description

maxA returns a function with the parameter A bound. The returned function evaluates to TRUE if any element of its argument is larger than A.

Usage

```
maxA(A=75, na.rm=TRUE)
```

Arguments

A	The value that at least one element must exceed.
na.rm	If TRUE then NA's are removed.

Value

maxA returns a function with an environment containing a binding for A.

Author(s)

R. Gentleman

See Also

pOverA

Examples

```
ff <- maxA(30)
ff(1:10)
ff(28:31)
```

nsFilter — *Non-Specific Filtering of Features in an ExpressionSet*

Description

This function identifies and removes features that appear to be less informative. Use cases for this function are: variable selection for subsequent sample clustering or classification tasks; independent filtering of features used in subsequent hypothesis testing, with the aim of increasing the detection rate (please see Details).

Usage

```
nsFilter(eset, require.entrez = TRUE, require.GOBP = FALSE,
    require.GOCC = FALSE, require.GOMF = FALSE,
    remove.dupEntrez = TRUE, var.func = IQR, var.cutoff = 0.5,
    var.filter = TRUE, filterByQuantile=TRUE,
    feature.exclude="^AFFX", ...)
varFilter(eset, var.func = IQR, var.cutoff = 0.5, filterByQuantile=TRUE)
featureFilter(eset, require.entrez=TRUE,
    require.GOBP=FALSE, require.GOCC=FALSE,
    require.GOMF=FALSE, remove.dupEntrez=TRUE,
    feature.exclude="^AFFX")
```

Arguments

eset	an ExpressionSet object
require.entrez	If TRUE, require that all probe sets have an Entrez Gene ID annotation. Probe sets without such an annotation will be filtered out.
require.GOBP	If TRUE, require that all probe sets have an annotation to at least one GO ID in the BP ontology. Probe sets without such an annotation will be filtered out.
require.GOCC	If TRUE, require that all probe sets have an annotation to at least one GO ID in the CC ontology. Probe sets without such an annotation will be filtered out.
require.GOMF	If TRUE, require that all probe sets have an annotation to at least one GO ID in the MF ontology. Probe sets without such an annotation will be filtered out.
remove.dupEntrez	If TRUE and there are multiple probe sets mapping to the same Entrez Gene ID, then the probe set with the largest value of var.func will be retained and the others removed.
var.func	A function that will be used to assess the variance of a probe set across all samples. This function should return a numeric vector of length one when given a numeric vector as input. Probe sets with a var.func value less than var.cutoff will be removed. The default is IQR.
var.cutoff	A numeric value to use in filtering out probe sets with small variance across samples. See the var.func argument and the details section below.
var.filter	A logical indicating whether or not to perform variance based filtering. The default is TRUE.
filterByQuantile	Logical: whether the variance-filter cutoff threshold should be interpreted as a quantile. Defaults to TRUE; if set to FALSE the cutoff value is used directly "as is".
feature.exclude	A character vector of regular expressions. Any probe sets identifiers (return value of featureNames(eset)) that match one of the specified patterns will be filtered out. The default value is intended to filter out Affymetrix quality control probe sets.
...	Unused, but available for specializing methods.

Details

Marginal type I errors: Independent filtering of features used in subsequent hypothesis testing can increase the detection rate at the same *marginal* type I error, as detailed in the following. Call U^I the stage 1 filter statistic, U^{II} the stage 2 test statistic for differential expression. Sufficient conditions for marginal type-I error control are:

- U^I the overall (across all samples) variance or mean, and U^{II} the t-statistic (or any other scale and location invariant statistic);
- U^I the overall mean, and U^{II} the moderated t-statistic (as in limma's `eBayes` function);
- U^I a sample-class label independent function (e.g. overall mean, median, variance, IQR), and U^{II} the Wilcoxon rank sum statistic.

In each of these cases, the value of U^I for the k-th feature must depend on the data for the k-th feature only, not on any other features.

Experiment-wide type I error: Marginal type-I error control provided by the conditions above is sufficient for control of the family wise error rate (FWER). Note, however, that common false discovery rate (FDR) methods depend not only on the marginal behaviour of the test statistics under the null hypothesis, but also on their joint distribution. The joint distribution can be affected by filtering. The effect of this is negligible in many cases in practice, but this depends on the dataset and the filter used, and the assessment is in the responsibility of the data analyst. For a more comprehensive discussion, please see the reference (Bourgon et al.).

Annotation Based Filtering Arguments `require.entrez`, `require.GOBP`, `require.GOCC`, and `require.GOMF` turn on a filter based on available annotation data. The annotation package is determined by calling `annotation(eset)`.

Duplicate Probe Removal If `remove.dupEntrez=TRUE`, probes determined by your annotation to be pointing to the same gene will be compared, and only the probe with the highest `var.func` value will be retained.

Variance Based Filtering The `var.filter`, `var.func`, `var.cutoff` and `varByQuantile` arguments control numerical cutoff-based filtering. The intention is to remove uninformative probe sets, representing genes that were not expressed at all. Probes for which `var.func` returns `NA` are removed. The default `var.func` is `IQR`, which is defined as `rowQ(eset, ceiling(0.75 * ncol(eset))) - rowQ(eset, floor(0.25 * ncol(eset)))`; this choice is motivated by the observation that unexpressed genes are detected most reliably through their low variability across samples. Additionally, `IQR` is robust to outliers (see note below). The default `var.cutoff` is `0.5` and is motivated by the rule of thumb that in many tissues only 40% of genes are expressed. Of course, if you believe in a different approach to numerical filtering you can choose another function as `var.func`, or turn off numerical filtering by setting `var.filter=FALSE`.

Note that by default the numerical-filter cutoff is interpreted as a quantile, so leaving the default values intact would filter out 50% of the genes remaining at this stage. If you prefer to set the cutoff at some absolute threshold, change the value of `varByQuantile` to `FALSE`, and modify `var.cutoff` accordingly.

Note also that variance filtering is performed last, so that (if `varByQuantile=TRUE` and `remove.dupEntrez=TRU`
the final number of genes does indeed exclude precisely the `var.cutoff` fraction of unique genes remaining after all other filters were passed.

The stand-alone function `varFilter` does only numerical filtering, and returns an `ExpressionSet`. `featureFilter` does only feature based filtering and duplicate removal, and returns an expression set as well. Duplicate removal is hard-coded to retain the highest-IQR probe for each gene.

Value

For `nsFilter` a list consisting of:

`eset`	the filtered `ExpressionSet`
`filter.log`	a list giving details of how many probe sets where removed for each filtering step performed.

For both `varFilter` and `featureFilter` the filtered `ExpressionSet`.

Note

`IQR` is a reasonable variance-filter choice when the dataset is split into two roughly equal and relatively homogeneous phenotype groups. If your dataset has important groups smaller than 25% of the overall sample size, or if you are interested in unusual individual-level patterns, then `IQR` may not be sensitive enough for your needs. In such cases, you should consider using less robust and more sensitive measures of variance (the simplest of which would be `sd`).

Author(s)

Seth Falcon (somewhat revised by Assaf Oron)

References

R. Bourgon, R. Gentleman, W. Huber, Independent filtering increases power for detecting differentially expressed genes, Technical Report.

Examples

```
library("hgu95av2.db")
library("Biobase")
data(sample.ExpressionSet)
ans <- nsFilter(sample.ExpressionSet)
ans$eset
ans$filter.log

## skip variance-based filtering
ans <- nsFilter(sample.ExpressionSet, var.filter=FALSE)

a1 <- varFilter(sample.ExpressionSet)
a2 <- featureFilter(sample.ExpressionSet)
```

---

pOverA *A filter function to filter according to the proportion of elements larger than A.*

Description

A function that returns a function with values for `A`, `p` and `na.rm` bound to the specified values. The function takes a single vector, `x`, as an argument. When the returned function is evaluated it returns `TRUE` if the proportion of values in `x` that are larger than `A` is at least `p`.

Usage

```
pOverA(p=0.05, A=100, na.rm=TRUE)
```

Arguments

A	The value to be exceeded.
p	The proportion that need to exceed A for TRUE to be returned.
na.rm	If TRUE then NA's are removed.

Value

pOverA returns a function with bindings for A, p and na.rm. This function evaluates to TRUE if the proportion of values in x that are larger than A exceeds p.

Author(s)

R. Gentleman

See Also

cv

Examples

```
ff<- pOverA(p=.1, 10)
ff(1:20)
ff(1:5)
```

---

rowFtests *t-tests and F-tests for rows or columns of a matrix*

Description

t-tests and F-tests for rows or columns of a matrix

Usage

```
rowttests(x, fac, tstatOnly = FALSE)
colttests(x, fac, tstatOnly = FALSE)
fastT(x, ig1, ig2, var.equal = TRUE)

rowFtests(x, fac, var.equal = TRUE)
colFtests(x, fac, var.equal = TRUE)
```

Arguments

x	Numeric matrix. The matrix must not contain NA values. For rowttests and colttests, x can also be an ExpressionSet.
fac	Factor which codes the grouping to be tested. There must be 1 or 2 groups for the t-tests (corresponding to one- and two-sample t-test), and 2 or more for the F-tests. If fac is missing, this is taken as a one-group test (i.e. is only allowed for the t-tests). The length of the factor needs to correspond to the sample size: for the row* functions, the length of the factor must be the same as the number of columns of x. for the col* functions, it must be the same as the number of rows of x. If x is an ExpressionSet, then fac may also be a character vector of length 1 with the name of a covariate variable in x.
tstatOnly	A logical variable indicating whether to calculate parametric p-values. If TRUE, just the t-statistics are returned. This can be considerably faster.
ig1	The indices of the columns of x that correspond to group 1.
ig2	The indices of the columns of x that correspond to group 2.
var.equal	A logical variable indicating whether to treat the variances in the samples as equal. If 'TRUE', a simple F test for the equality of means in a one-way analysis of variance is performed. If 'FALSE', an approximate method of Welch (1951) is used, which generalizes the commonly known 2-sample Welch test to the case of arbitrarily many samples.

Details

If fac is specified, rowttests performs for each row of x a two-sided, two-class t-test with equal variances. fac must be a factor of length ncol(x) with two levels, corresponding to the two groups. The sign of the resulting t-statistic corresponds to "group 1 minus group 2". If fac is missing, rowttests performs for each row of x a two-sided one-class t-test against the null hypothesis 'mean=0'.

rowttests and colttests are implemented in C and are reasonably fast and memory-efficient. fastT is an alternative implementation, in Fortran, possibly useful for certain legacy code. rowFtests and colFtests are currently implemented using matrix algebra in R. Compared to the *ttests functions, they are slower and use more memory.

Value

A data.frame with columns statistic, p.value (optional in the case of the t-test functions) and dm, the difference of the group means (only in the case of the t-test functions). The row.names of the data.frame are taken from the corresponding dimension names of x.

The degrees of freedom are provided in the attribute df. For the F-tests, if var.equal is 'FALSE', nrow(x)+1 degree of freedoms are given, the first one is the first degree of freedom (it is the same for each row) and the other ones are the second degree of freedom (one for each row).

Author(s)

Wolfgang Huber <huber@ebi.ac.uk>

References

B. L. Welch (1951), On the comparison of several mean values: an alternative approach. Biometrika, *38*, 330-336

See Also

mt.teststat

Examples

```
x  = matrix(runif(970), ncol=97)
f2 = factor(floor(runif(ncol(x))*2))
f7 = factor(floor(runif(ncol(x))*7))

r1 = rowttests(x)
r2 = rowttests(x, f2)
r7 = rowFtests(x, f7)

## compare with pedestrian tests
about.equal = function(x,y,tol=1e-10)
  stopifnot(all(abs(x-y) < tol))

for (j in 1:nrow(x)) {
  s1 = t.test(x[j,])
  about.equal(s1$statistic, r1$statistic[j])
  about.equal(s1$p.value,   r1$p.value[j])

  s2 = t.test(x[j,] ~ f2, var.equal=TRUE)
  about.equal(s2$statistic, r2$statistic[j])
  about.equal(s2$p.value,   r2$p.value[j])

  dm = -diff(tapply(x[j,], f2, mean))
  about.equal(dm, r2$dm[j])

  s7 = summary(lm(x[j,]~f7))
  about.equal(s7$statistic$value, r7$statistic[j])
}

## colttests
c2 = colttests(t(x), f2)
stopifnot(identical(r2, c2))

## missing values
f2n = f2
f2n[sample(length(f2n), 3)] = NA
r2n = rowttests(x, f2n)
for(j in 1:nrow(x)) {
  s2n = t.test(x[j,] ~ f2n, var.equal=TRUE)
  about.equal(s2n$statistic, r2n$statistic[j])
  about.equal(s2n$p.value,   r2n$p.value[j])
}
```

---

rowpAUCs-methods *Rowwise ROC and pAUC computation*

Description

Methods for fast rowwise computation of ROC curves and (partial) area under the curve (pAUC) using the simple classification rule `x > theta`, where `theta` is a value in the range of `x`

Usage

```
rowpAUCs(x, fac, p=0.1, flip=TRUE, caseNames=c("1", "2"))
```

Arguments

x	ExpressionSet or numeric matrix. The matrix must not contain NA values.
fac	A factor or numeric or character that can be coerced to a factor. If x is an ExpressionSet, this may also be a character vector of length 1 with the name of a covariate variable in x. fac must have exactly 2 levels. For better control over the classification, use integer values in 0 and 1, where 1 indicates the "Disease" class in the sense of the Pepe et al paper (see below).
p	Numeric vector of length 1. Limit in (0,1) to integrate pAUC to.
flip	Logical. If TRUE, both classification rules x > theta and x < theta are tested and the (partial) area under the curve of the better one of the two is returned. This is appropriate for the cases in which the classification is not necessarily linked to higher expression values, but instead it is symmetric and one would assume both over- and under-expressed genes for both classes. You can set flip to FALSE if you only want to screen for genes which discriminate Disease from Control with the x > theta rule.
caseNames	The class names that are used when plotting the data. If fac is the name of the covariate variable in the ExpressionSet the function will use its levels as caseNames.

Details

Rowwise calculation of Receiver Operating Characteristic (ROC) curves and the corresponding partial area under the curve (pAUC) for a given data matrix or ExpressionSet. The function is implemented in C and thus reasonably fast and memory efficient. Cutpoints (theta are calculated before the first, in between and after the last data value. By default, both classification rules x > theta and x < theta are tested and the (partial) area under the curve of the better one of the two is returned. This is only valid for symmetric cases, where the classification is independent of the magnitude of x (e.g., both over- and under-expression of different genes in the same class). For unsymmetric cases in which you expect x to be consistently higher/lower in of of the two classes (e.g. presence or absence of a single biomarker) set flip=FALSE or use the functionality provided in the ROC package. For better control over the classification (i.e., the choice of "Disease" and "Control" class in the sense of the Pepe et al paper), argument fac can be an integer in [0,1] where 1 indicates "Disease" and 0 indicates "Control".

Value

An object of class rowROC with the calculated specificities and sensitivities for each row and the corresponding pAUCs and AUCs values. See rowROC for details.

Methods

Methods exist for rowPAUCs:

```
signature(x="matrix", fac="factor")
```

rowPAUCs **rowPAUCs** `signature(x="matrix", fac="numeric")`

rowPAUCs `signature(x="ExpressionSet")`

rowPAUCs `signature(x="ExpressionSet", fac="character")`

Author(s)

Florian Hahne <fhahne@fhcrc.org>

References

Pepe MS, Longton G, Anderson GL, Schummer M.: Selecting differentially expressed genes from microarray experiments. *Biometrics. 2003 Mar;59(1):133-42.*

See Also

rocdemo.sca, rocdemo.sca, rocdemo.sca

Examples

```
library(Biobase)
data(sample.ExpressionSet)

r1 = rowttests(sample.ExpressionSet, "sex")
r2 = rowpAUCs(sample.ExpressionSet, "sex", p=0.1)

plot(area(r2, total=TRUE), r1$statistic, pch=16)
sel <- which(area(r2, total=TRUE) > 0.7)
plot(r2[sel])

## this compares performance and output of rowpAUCs to function pAUC in
## package ROC
if(require(ROC)){
  ## performance
  myRule = function(x)
    pAUC(rocdemo.sca(truth = as.integer(sample.ExpressionSet$sex)-1 ,
         data = x, rule = dxrule.sca), t0 = 0.1)
  nGenes = 200
  cat("computation time for ", nGenes, "genes:\n")
  cat("function pAUC: ")
  print(system.time(r3 <- esApply(sample.ExpressionSet[1:nGenes, ], 1, myRule)))
  cat("function rowpAUCs: ")
  print(system.time(r2 <- rowpAUCs(sample.ExpressionSet[1:nGenes, ],
  "sex", p=1)))

  ## compare output
  myRule2 = function(x)
   pAUC(rocdemo.sca(truth = as.integer(sample.ExpressionSet$sex)-1 ,
                    data = x, rule = dxrule.sca), t0 = 1)
  r4 <-  esApply(sample.ExpressionSet[1:nGenes, ], 1, myRule2)
  plot(r4,area(r2), xlab="function pAUC", ylab="function rowpAUCs",
  main="pAUCs")

  plot(r4, area(rowpAUCs(sample.ExpressionSet[1:nGenes, ],
  "sex", p=1, flip=FALSE)), xlab="function pAUC", ylab="function rowpAUCs",
  main="pAUCs")

  r4[r4<0.5] <- 1-r4[r4<0.5]
  plot(r4, area(r2), xlab="function pAUC", ylab="function rowpAUCs",
  main="pAUCs")
 }
```

rowROC-class *Class "rowROC"*

Description

A class to model ROC curves and corresponding area under the curve as produced by rowpAUCs.

Objects from the Class

Objects can be created by calls of the form `new("rowROC", ...)`.

Slots

data: Object of class `"matrix"` The input data.

ranks: Object of class `"matrix"` The ranked input data.

sens: Object of class `"matrix"` Matrix of senitivity values for each gene at each cutpoint.

spec: Object of class `"matrix"` Matrix of specificity values for each gene at each cutpoint.

pAUC: Object of class `"numeric"` The partial area under the curve (integrated from 0 to `p`.

AUC: Object of class `"numeric"` The total area under the curve.

factor: Object of class `"factor"` The factor used for classification.

cutpoints: Object of class `"matrix"` The values of the cutpoints at which specificity ans sensitivity was calculated. (Note: the data is ranked prior to computation of ROC curves, the cutpoints map to the ranked data.

caseNames: Object of class `"character"` The names of the two classification cases.

p: Object of class `"numeric"` The limit to which `pAUC` is integrated.

Methods

show **`signature(object="rowROC")`** Print nice info about the object.

[**`signature(x="rowROC", j="missing")`** Subset the object according to rows/genes.

plot **`signature(x="rowROC", y="missing")`** Plot the ROC curve of the first row of the object along with the `pAUC`. To plot the curve for a specific row/gene subsetting should be done first (i.e. `plot(rowROC[1])`.

pAUC **`signature(object="rowROC", p="numeric", flip="logical")`** Integrate area under the curve from `0` to `p`. This method returns a new `rowROC` object.

AUC **`signature(object="rowROC")`** Integrate total area under the curve. This method returns a new `rowROC` object.

sens **`signature(object="rowROC")`** Accessor method for sensitivity slot.

spec **`signature(object="rowROC")`** Accessor method for specificity slot.

area **`signature(object="rowROC", total="logical")`** Accessor method for pAUC slot.

Author(s)

Florian Hahne <fhahne@fhcrc.org>

References

Pepe MS, Longton G, Anderson GL, Schummer M.: Selecting differentially expressed genes from microarray experiments. *Biometrics. 2003 Mar;59(1):133-42.*

See Also

rowpAUCs

Examples

```
library(Biobase)
require(genefilter)
data(sample.ExpressionSet)
roc <- rowpAUCs(sample.ExpressionSet, "sex", p=0.5)
roc
area(roc[1:3])

if(interactive()) {
par(ask=TRUE)
plot(roc)
plot(1-spec(roc[1]), sens(roc[2]))
par(ask=FALSE)
}

pAUC(roc, 0.1)
roc
```

---

rowSds *Row variance and standard deviation of a numeric array*

Description

Row variance and standard deviation of a numeric array

Usage

```
rowVars(x, ...)
rowSds(x, ...)
```

Arguments

x	An array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame.
...	Further arguments that get passed on to rowMeans and rowSums.

Details

These are very simple convenience functions, the main work is done in rowMeans and rowSums. See the function definition of rowVars, it is very simple.

Value

A numeric or complex array of suitable size, or a vector if the result is one-dimensional. The 'dimnames' (or 'names' for a vector result) are taken from the original array.

Author(s)

Wolfgang Huber http://www.ebi.ac.uk/huber

See Also

rowMeans and rowSums

Examples

```
a = matrix(rnorm(1e4), nrow=10)
rowSds(a)
```

---

shorth *A location estimator based on the shorth*

Description

A location estimator based on the shorth

Usage

```
shorth(x, na.rm=FALSE, tie.action="mean", tie.limit=0.05)
```

Arguments

x	Numeric
na.rm	Logical. If TRUE, then non-finite (according to is.finite) values in x are ignored. Otherwise, presence of non-finite or NA values will lead to an error message.
tie.action	Character scalar. See details.
tie.limit	Numeric scalar. See details.

Details

The shorth is the shortest interval that covers half of the values in x. This function calculates the mean of the x values that lie in the shorth. This was proposed by Andrews (1972) as a robust estimator of location.

Ties: if there are multiple shortest intervals, the action specified in ties.action is applied. Allowed values are mean (the default), max and min. For mean, the average value is considered; however, an error is generated if the start indices of the different shortest intervals differ by more than the fraction tie.limit of length(x). For min and max, the left-most or right-most, respectively, of the multiple shortest intervals is considered.

Rate of convergence: as an estimator of location of a unimodal distribution, under regularity conditions, the quantity computed here has an asymptotic rate of only $n^{-1/3}$ and a complicated limiting distribution.

See half.range.mode for an iterative version that refines the estimate iteratively and has a builtin bootstrapping option.

Value

The mean of the x values that lie in the shorth.

Author(s)

Wolfgang Huber http://www.ebi.ac.uk/huber, Ligia Pedroso Bras

References

- G Sawitzki, "The Shorth Plot." Available at http://lshorth.r-forge.r-project.org/TheShorthPlot.pdf
- DF Andrews, "Robust Estimates of Location." Princeton University Press (1972).
- R Grueble, "The Length of the Shorth." Annals of Statistics 16, 2:619-628 (1988).
- DR Bickel and R Fruehwirth, "On a fast, robust estimator of the mode: Comparisons to other robust estimators with applications." Computational Statistics & Data Analysis 50, 3500-3530 (2006).

See Also

half.range.mode

Examples

```
x = c(rnorm(500), runif(500) * 10)
methods = c("mean", "median", "shorth", "half.range.mode")
ests = sapply(methods, function(m) get(m)(x))

if(interactive()) {
  colors = 1:4
  hist(x, 40, col="orange")
  abline(v=ests, col=colors, lwd=3, lty=1:2)
  legend(5, 100, names(ests), col=colors, lwd=3, lty=1:2)
}
```

---

tdata *A small test dataset of Affymetrix Expression data.*

Description

The tdata data frame has 500 rows and 26 columns. The columns correspond to samples while the rows correspond to genes. The row names are Affymetrix accession numbers.

Usage

```
data(tdata)
```

Format

This data frame contains 26 columns.

Source

An unknown data set.

Examples

```
data(tdata)
```

`ttest` *A filter function for a t.test*

Description

`ttest` returns a function of one argument with bindings for `cov` and `p`. The function, when evaluated, performs a t-test using `cov` as the covariate. It returns `TRUE` if the p value for a difference in means is less than `p`.

Usage

```
ttest(m, p=0.05, na.rm=TRUE)
```

Arguments

`m`	If `m` is of length one then it is assumed that elements one through `m` of `x` will be one group. Otherwise `m` is presumed to be the same length as `x` and constitutes the groups.
`p`	The p-value for the test.
`na.rm`	If set to `TRUE` any `NA`'s will be removed.

Details

When the data can be split into two groups (diseased and normal for example) then we often want to select genes on their ability to distinguish those two groups. The t-test is well suited to this and can be used as a filter function.

This helper function creates a t-test (function) for the specified covariate and considers a gene to have passed the filter if the p-value for the gene is less than the prespecified `p`.

Value

`ttest` returns a function with bindings for `m` and `p` that will perform a t-test.

Author(s)

R. Gentleman

See Also

`kOverA`, `Anova`, `t.test`

Examples

```
dat <- c(rep(1,5),rep(2,5))
set.seed(5)
y <- rnorm(10)
af <- ttest(dat, .01)
af(y)
af2 <- ttest(5, .01)
af2(y)
y[8] <- NA
af(y)
af2(y)
y[1:5] <- y[1:5]+10
af(y)
```

Index