Package 'saps'

October 9, 2015

Title Significance Analysis of Prognostic Signatures

Description Functions implementing the Significance Analysis of Prognostic Signatures method (SAPS). SAPS provides a robust method for identifying biologically significant gene sets associated with patient survival. Three basic statistics are computed. First, patients are clustered into two survival groups based on differential expression of a candidate gene set. P_pure is calculated as the probability of no survival difference between the two groups. Next, the same procedure is applied to randomly generated gene sets, and P_random is calculated as the proportion achieving a P_pure as significant as the candidate gene set. Finally, a pre-ranked Gene Set Enrichment Analysis (GSEA) is performed by ranking all genes by concordance index, and P_enrich is computed to indicate the degree to which the candidate gene set is enriched for genes with univariate prognostic significance. A SAPS_score is calculated to summarize the three statistics, and optionally a Q-value is computed to estimate the significance of the SAPS_score by calculating SAPS_scores for random gene sets.

Version 2.0.0

biocViews BiomedicalInformatics, GeneExpression, GeneSetEnrichment, DifferentialExpression, Survival

Maintainer Daniel Schmolze <saps@schmolze.com>

Depends R (>= 2.14.0), survival

Imports piano, survcomp, reshape2

Suggests snowfall, knitr

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

Author Daniel Schmolze [aut, cre], Andrew Beck [aut], Benjamin Haibe-Kains [aut]

NeedsCompilation no

R topics documented:

`saps-package` *Implements Significance Analysis of Prognostic Signatures (SAPS), a robust method for determining prognostically significant gene sets*

Description

`saps` will usually be the only function needed.

References

Beck AH, Knoblauch NW, Hefti MM, Kaplan J, Schnitt SJ, et al. (2013) Significance Analysis of Prognostic Signatures. PLoS Comput Biol 9(1): e1002875.doi:10.1371/journal.pcbi.1002875

`calculatePEnrichment` *Compute P_enrichment*

Description

This function performs a pre-ranked gene set enrichment analysis (GSEA) to evaluate the degree to which a candidate gene set is overrepresented at the top or bottom extremes of a ranked list of concordance indices. This function is normally called by `saps`.

Usage

```
calculatePEnrichment(rankedGenes, candidateGeneSet, cpus, gsea.perm = 1000)
```

Arguments

rankedGenes	An *nx1* matrix of concordance indices for *n* genes. Generally this will be the z-score returned by rankConcordance. The row names should contain gene identifiers.
candidateGeneSet	A *1xp* matrix of *p* gene identifiers. The row name should contain a name for the gene set.
cpus	This value is passed to the runGSA function in the **piano** package. For multi-core CPUs, this value should be set to the number of cores (which will significantly improve the computational time).
gsea.perm	The number of permutations to be used in the GSEA. This value is passed to runGSA.

Value

The function returns a matrix with the following columns:

P_enrichment	the enrichment score
direction	either 1 or -1 depending on the direction of association

References

Beck AH, Knoblauch NW, Hefti MM, Kaplan J, Schnitt SJ, et al. (2013) Significance Analysis of Prognostic Signatures. PLoS Comput Biol 9(1): e1002875.doi:10.1371/journal.pcbi.1002875

Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, et al. (2005) Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. Proc Natl Acad Sci USA 102: 15545-15550.

See Also

saps runGSA

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create two random genesets of 5 genes each
set1 <- sample(colnames(dat), 5)
set2 <- sample(colnames(dat), 5)

genesets <- rbind(set1, set2)
```

```
# tweak data for first 5 patients for set1
dat[1:5, set1] <- dat[1:5, set1]+10

# rank all genes by concordance index
ci <- rankConcordance(dat, time, followup)[,"z"]

# set1 should achieve significance
p_enrich <- calculatePEnrichment(ci, genesets["set1",,drop=FALSE], cpus=1)
p_enrich

# set2 should not
p_enrich <- calculatePEnrichment(ci, genesets["set2",,drop=FALSE], cpus=1)
p_enrich
```

`calculatePPure` *Compute P_pure*

Description

This function stratifies patients into two groups via k-means clustering (k=2) on an *nxp* matrix consisting of *n* patients and *p* genes in the candidate prognostic set. It is normally called by `saps`.

Usage

```
calculatePPure(geneData, survivalTimes, followup)
```

Arguments

`geneData`	An *nxp* matrix consisting of *n* patients and *p* genes in the candidate prognostic geneset.
`survivalTimes`	A vector of survival times. The length must equal the number of rows *n* in `geneData`.
`followup`	A vector of 0 or 1 values, indicating whether the patient was lost to followup (0) or not (1). The length must equal the number of rows (i.e. patients) in `geneData`.

Value

A list with the following elements:

`p_pure`	A log-rank p-value indicating the probability that the two groups show no survival difference.
`cluster`	Vector of assigned cluster (1 or 2) for each patient using the supplied candidate prognostic geneset.

References

Beck AH, Knoblauch NW, Hefti MM, Kaplan J, Schnitt SJ, et al. (2013) Significance Analysis of Prognostic Signatures. PLoS Comput Biol 9(1): e1002875.doi:10.1371/journal.pcbi.1002875

See Also

saps

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create random genesets of 5 genes
set1 <- sample(colnames(dat), 5)

# get gene data for set1
set1_data <- dat[, set1]

# shouldn't achieve significance
p_pure <- calculatePPure(set1_data, time, followup)
p_pure$p_pure

# alter expression data for first 5 patients for set1
dat[1:5, set1] <- dat[1:5, set1]+10

set1_data <- dat[, set1]

# now p_pure should be significant
p_pure <- calculatePPure(set1_data, time, followup)
p_pure$p_pure
```

---

calculatePRandom *Compute P_random*

Description

This function randomly samples gene sets, and calculates P_pure (via calculatePPure) for each one. P_random is the proportion of randomly sampled gene sets achieving a P_pure at least as significant as the provided p_pure. This function is normally called by saps.

Usage

```
calculatePRandom(dataSet, sampleSize, p_pure, survivalTimes, followup,
  random.samples = 10000)
```

Arguments

dataSet	A matrix, where the column names are gene identifiers and the values are gene expression levels. Each row should contain data for a single patient.
sampleSize	The desired size for the randomly sampled gene sets.
p_pure	The candidate P_pure against which to compare the P_pure values for the randomly generated gene sets.
survivalTimes	A vector of survival times. The length must equal the number of rows in dataSet.
followup	A vector of 0 or 1 values, indicating whether the patient was lost to followup (0) or not (1). The length must equal the number of rows (i.e. patients) in dataSet.
random.samples	The number of random gene sets to sample.

Value

A list with the following elements:

p_random	The proportion of randomly sampled gene sets with a calculated p_pure at least as significant as the provided p_pure.
p_pures	A vector of calculated p_pure values for each randomly generated geneset.

References

Beck AH, Knoblauch NW, Hefti MM, Kaplan J, Schnitt SJ, et al. (2013) Significance Analysis of Prognostic Signatures. PLoS Comput Biol 9(1): e1002875.doi:10.1371/journal.pcbi.1002875

See Also

saps

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# relatively low threshold
p_pure <- 0.05

p_random <- calculatePRandom(dat, 5, p_pure, time, followup, random.samples=100)
p_random$p_random
hist(p_random$p_pures)
length(p_random$p_pures[p_random$p_pures <= p_pure])

# set a more stringent threshold
```

```
p_pure <- 0.001

p_random <- calculatePRandom(dat, 5, p_pure, time, followup, random.samples=100)
length(p_random$p_pures[p_random$p_pures <= p_pure])
```

`calculateQValue` *Compute saps q-value*

Description

This function computes the saps q-value for a candidate prognostic geneset by computing the saps score for randomly generated genesets and determining the proportion at least as significant as the saps score for the candidate set. This function is normally called by `saps`.

Usage

```
calculateQValue(dataSet, sampleSize, survivalTimes, followup, saps_score,
  random.samples, qvalue.samples, cpus, gsea.perm, rankedGenes)
```

Arguments

`dataSet`	A matrix, where the column names are gene identifiers and the values are gene expression levels. Each row should contain data for a single patient.
`sampleSize`	The desired size for the randomly sampled gene sets.
`survivalTimes`	A vector of survival times. The length must equal the number of rows in `dataSet`.
`followup`	A vector of 0 or 1 values, indicating whether the patient was lost to followup (0) or not (1). The length must equal the number of rows (i.e. patients) in `dataSet`.
`saps_score`	The saps score for the candidate geneset. The q-value is calculated as the proportion of random saps scores whose absolute value is >= to the provided saps score.
`random.samples`	The number of random gene sets to sample during the calculation of `P_random`.
`qvalue.samples`	The number of random gene sets to sample for purposes of computing the q-value.
`cpus`	An integer that specifies the number of cpus/cores to be used when calculating `P_enrichment`. If greater than 1, the **snowfall** package must be installed or an error will occur.
`gsea.perm`	The number of permutations to be used when calculating `P_enrich`. This is passed to the `runGSA` function in the **piano** package.
`rankedGenes`	A vector of ranking scores for each gene in `dataSet`. Ordinarily this will be the z-scores obtained by a call to `rankConcordance`.

Value

The function returns a list with two elements:

`q_value` the calculated q-value.

`random_saps_scores`
a vector of individual saps scores for each randomly generated geneset.

References

Beck AH, Knoblauch NW, Hefti MM, Kaplan J, Schnitt SJ, et al. (2013) Significance Analysis of Prognostic Signatures. PLoS Comput Biol 9(1): e1002875.doi:10.1371/journal.pcbi.1002875

See Also

saps

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# borderline significant saps score
saps_score <- 1.3

rankedGenes <- rankConcordance(dat, time, followup)[,"z"]

q_value <- calculateQValue(dat, 5, time, followup, saps_score, random.samples=100,
     qvalue.samples=10, cpus=1, gsea.perm=1000, rankedGenes)

q_value$q_value
random_scores <- abs(q_value$random_saps_scores)
hist(random_scores)
length(random_scores[random_scores > saps_score])
```

---

plotEnrichment *Plot concordance indices for a geneset*

Description

This function draws concordance indices for a given geneset relative to the concordance indices for all the genes in the dataset (i.e., the degree of enrichment for the geneset).

Usage

```
plotEnrichment(geneset, rankedGenes)
```

Arguments

geneset	A geneset as returned by saps.
rankedGenes	A vector of concordance index z-scores. Usually this will be the rankedGenes element returned by saps.

Value

The function is used for side-effects (drawing a plot). No value is returned.

See Also

saps rankConcordance

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create two random genesets of 5 genes each
set1 <- sample(colnames(dat), 5)
set2 <- sample(colnames(dat), 5)

genesets <- rbind(set1, set2)

# run saps
results <- saps(genesets, dat, time, followup, random.samples=100)

set <- results$genesets[["set1"]]

# p_enrich should not be significant
plotEnrichment(set, results$rankedGenes)

# increase expression levels for set1 for first 5 patients
dat[1:5, set1] <- dat[1:5, set1]+10

# run saps again
results <- saps(genesets, dat, time, followup, random.samples=100)

set <- results$genesets[["set1"]]
```

```
# now it should be significant
plotEnrichment(set, results$rankedGenes)
```

---

plotKM *Plot Kaplan-Meier curves for a gene set*

Description

Plots Kaplan-Meier survival curves for a given gene set using the cluster labels generated during the computation of p_pure to stratify patients into two survival groups. The function is a wrapper for km.coxph.plot in the **survcomp** package.

Usage

```
plotKM(geneset, survivalTimes, followup, title = NA, y.label = NA,
  x.label = NA, p.text = NA, ...)
```

Arguments

geneset	A geneset as returned by saps.
survivalTimes	A vector of survival times, as used in a call to saps.
followup	A vector of 0 or 1 values, indicating whether the patient was lost to followup (0) or not (1), as used in a call to saps.
title	The plot title. Defaults to "Kaplan-Meier curves for geneset [geneset["name"]]".
y.label	The y-axis label. Defaults to "Probability of survival".
x.label	The x-axis label. Defaults to "Overall survival".
p.text	Text to display in the lower left hand corner. Defaults to displaying p_pure and p_pure_adj.
...	Additional arguments to be passed to km.coxph.plot

Value

The function is used for side-effects (drawing a plot). No value is returned.

See Also

saps calculatePPure km.coxph.plot

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create two random genesets of 5 genes each
set1 <- sample(colnames(dat), 5)
set2 <- sample(colnames(dat), 5)

genesets <- rbind(set1, set2)

# run saps
results <- saps(genesets, dat, time, followup, random.samples=100)

set <- results$genesets[["set1"]]

# KM plots should not seperate at this point
plotKM(set, time/365, followup, x.label="Overall survival (years)")

# increase expression levels for set1 for first 5 patients
dat[1:5, set1] <- dat[1:5, set1]+10

# run saps again
results <- saps(genesets, dat, time, followup, random.samples=100)

set <- results$genesets[["set1"]]

# KM plots should now seperate
plotKM(set, time/365, followup, x.label="Overall survival (years)")
```

---

plotRandomDensity *Draw density plot of* p_pure *values for random gene sets*

Description

This function retrieves the p_pure values for the random gene sets generated during the computation of p_random for a given gene set. These are drawn as a density plot, with the value of p_pure for the gene set indicated. The value of p_random for the gene set is displayed as well.

Usage

```
plotRandomDensity(geneset, ...)
```

Arguments

geneset	A geneset as returned by saps.
...	Additional arguments to be passed to plot

Value

The function is used for side-effects (drawing a plot). No value is returned.

See Also

saps calculatePRandom

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create two random genesets of 5 genes each
set1 <- sample(colnames(dat), 5)
set2 <- sample(colnames(dat), 5)

genesets <- rbind(set1, set2)

# run saps
results <- saps(genesets, dat, time, followup, random.samples=100)

set <- results$genesets[["set1"]]

# should not be significant
plotRandomDensity(set)

# increase expression levels for set1 for first 5 patients
dat[1:5, set1] <- dat[1:5, set1]+10

# run saps again
results <- saps(genesets, dat, time, followup, random.samples=100)

set <- results$genesets[["set1"]]

# now it should be significant
plotRandomDensity(set)
```

plotSapsScoreDensity *Draw density plot of* saps_score *values for random gene sets*

Description

This function retrieves the saps_score values for the random gene sets generated during the computation of saps_qvalue for a given gene set. These are drawn as a density plot, with the value of saps_score for the gene set indicated.

Usage

```
plotSapsScoreDensity(geneset, ...)
```

Arguments

geneset	A geneset as returned by saps.
...	Additional arguments to be passed to plot

Value

The function is used for side-effects (drawing a plot). No value is returned.

See Also

saps calculateQValue

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create two random genesets of 5 genes each
set1 <- sample(colnames(dat), 5)
set2 <- sample(colnames(dat), 5)

genesets <- rbind(set1, set2)

# increase expression levels for set1 for first 5 patients
dat[1:5, set1] <- dat[1:5, set1]+10

# run saps and compute q-values
results <- saps(genesets, dat, time, followup, random.samples=100,
```

```
                compute_qvalue=TRUE, qvalue.samples=10)

set <- results$genesets[["set1"]]

# qvalue.samples=10 is too small to achieve significance
plotSapsScoreDensity(set)
```

---

rankConcordance *Compute concordance indices*

Description

Computes concordance indices for a gene expression data set, and returns the concordance index and the z-score.

Usage

```
rankConcordance(dataset, survivalTimes, followup)
```

Arguments

dataset	A matrix, where the column names are gene identifiers and the values are gene expression levels. Each row should contain data for a single patient.
survivalTimes	A vector of survival times. The length must equal the number of rows (i.e. patients) in dataset.
followup	A vector of 0 or 1 values, indicating whether the patient was lost to followup (0) or not (1). The length must equal the number of rows (i.e. patients) in dataSet.

Details

This function is a wrapper for concordance.index in the **survcomp** package. It applies the latter over the columns of dataset to calculate concordance indices and the corresponding z-score for each gene.

Value

The function returns a matrix with two columns:

cindex	concordance index estimate.
z	z-score of the concordance index estimate.

and as many rows as dataset. The row names contain the gene identifiers.

See Also

saps concordance.index

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

ci <- rankConcordance(dat, time, followup)
z <- ci[,"z"]
range(z)
hist(z)
```

---

saps *Compute SAPS statistics*

Description

This is the main user interface to the **saps** package, and is usually the only function needed.

Usage

```
saps(candidateGeneSets, dataSet, survivalTimes, followup,
  random.samples = 1000, cpus = 1, gsea.perm = 1000,
  compute_qvalue = FALSE, qvalue.samples = 1000, verbose = TRUE)
```

Arguments

candidateGeneSets	A matrix with at least one row, where each row represents a gene set, and the column values are gene identifiers. The row names should contain unique names for the gene sets. The column values may contain NA values, since in general gene sets will have differing lengths.
dataSet	A matrix, where the column names are gene identifiers (in the same format as the values in candidateGeneSets) and the values are gene expression levels. Each row should contain data for a single patient.
survivalTimes	A vector of survival times. The length must equal the number of rows (i.e. patients) in dataSet.
followup	A vector of 0 or 1 values, indicating whether the patient was lost to followup (0) or not (1). The length must equal the number of rows (i.e. patients) in dataSet.
random.samples	An integer that specifies how many random gene sets to sample when computing P_random. Defaults to 1000.

cpus	An integer that specifies the number of cpus/cores to be used when calculating P_enrichment. If greater than 1 (the default), the **snowfall** package must be installed or an error will occur.
gsea.perm	The number of permutations to be used when calculating p_enrich. This is passed to the runGSA function in the **piano** package. Defaults to 1000.
compute_qvalue	A boolean indicating whether to include calculation of the saps q_value. Setting this to TRUE will significantly increase the computational time.
qvalue.samples	An integer that specifies how many random gene sets to sample when computing the saps q_value. Defaults to 1000.
verbose	A boolean indicating whether to display status messages during computation. Defaults to TRUE.

Details

saps provides a robust method for identifying biologically significant gene sets associated with patient survival. Three basic statistics are computed. First, patients are clustered into two survival groups based on differential expression of a candidate gene set. p_pure is calculated as the probability of no survival difference between the two groups.

Next, the same procedure is applied to randomly generated gene sets, and p_random is calculated as the proportion achieving a p_pure as significant as the candidate gene set. Finally, a pre-ranked Gene Set Enrichment Analysis (GSEA) is performed by ranking all genes by concordance index, and p_enrich is computed to indicate the degree to which the candidate gene set is enriched for genes with univariate prognostic significance.

A saps_score is calculated to summarize the three statistics, and optionally a saps_qvalue is computed to estimate the significance of the saps_score by calculating the saps_score for random gene sets.

Value

The function returns a list with the following elements:

rankedGenes	Vector of concordance index z-scores for the genes in dataSet, named by gene identifier.
geneset.count	The number of gene sets analyzed.
genesets	A list of genesets (see below).
saps_table	A dataframe summarizing the adjusted and unadjusted saps statistics for each geneset analyzed. The dataframe contains the following columns: size, p_pure, p_random, p_enrich Each row summarizes a single geneset. Note that the saps statistics are stored with each individual geneset as well; this table is provided simply for convenience.

genesets is in turn a list with the following elements:

name	The name of the geneset.
size	The number of genes in the geneset.
genes	Vector of gene labels for this geneset.

`saps_unadjusted`
: Vector with elements `p_pure`, `p_random`, `p_enrich`, `saps_score`, and `saps_qvalue` containing the respective unadjusted p-values.

`saps_adjusted`
: Vector with elements `p_pure`, `p_random`, `p_enrich`, `saps_score`, and `saps_qvalue` containing the respective p-values adjusted for multiple comparisons.

`cluster`
: Vector of assigned cluster (1 or 2) for each patient using this candidate geneset.

`random_p_pures`
: Vector of p_pure values for each random geneset generated during the computation of p_random.

`random_saps_scores`
: Vector of saps_score values for each random geneset generated during the computation of saps_qvalue.

`direction`
: Direction (-1 or 1) of the enrichment association for this geneset.

References

Beck AH, Knoblauch NW, Hefti MM, Kaplan J, Schnitt SJ, et al. (2013) Significance Analysis of Prognostic Signatures. PLoS Comput Biol 9(1): e1002875.doi:10.1371/journal.pcbi.1002875

See Also

`survdiff` `concordance.index` `runGSA`

Examples

```
# 25 patients, none lost to followup
followup <- rep(1, 25)

# first 5 patients have good survival (in days)
time <- c(25, 27, 24, 21, 26, sample(1:3, 20, TRUE))*365

# create data for 100 genes, 25 patients
dat <- matrix(rnorm(25*100), nrow=25, ncol=100)
colnames(dat) <- as.character(1:100)

# create two random genesets of 5 genes each
set1 <- sample(colnames(dat), 5)
set2 <- sample(colnames(dat), 5)

genesets <- rbind(set1, set2)

# compute saps
results <- saps(genesets, dat, time, followup, random.samples=100)

# check results
saps_table <- results$saps_table
saps_table[1:7]

# increase expression levels for set1 for first 5 patients
dat[1:5, set1] <- dat[1:5, set1]+10
```

```
# run again, should get significant values for set1
results <- saps(genesets, dat, time, followup, random.samples=100)

# check results
saps_table <- results$saps_table
saps_table[1:7]
```

Index