Package 'shiftR'

October 14, 2022

Type Package

Title Fast Enrichment Analysis via Circular Permutations

Version 1.5

Date 2019-03-21

Description Fast enrichment analysis for locally correlated statistics
via circular permutations.
The analysis can be performed at multiple significance thresholds
for both primary and auxiliary data sets with
efficient correction for multiple testing.

BugReports https://github.com/andreyshabalin/shiftR/issues

URL https://github.com/andreyshabalin/shiftR

License LGPL-3

Imports parallel

Suggests knitr, rmarkdown, pander

VignetteBuilder knitr

NeedsCompilation yes

Author Andrey A Shabalin [aut, cre] (<https://orcid.org/0000-0003-0309-6821>),
Edwin J C G van den Oord [aut]

Maintainer Andrey A Shabalin <andrey.shabalin@gmail.com>

Repository CRAN

Date/Publication 2019-03-22 09:50:03 UTC

R topics documented:

shiftR-package 2
cramerV 2
enrichmentAnalysis 3
getOffsets 5
matchDatasets 6
shiftrPermBinary 8

shiftrPrepare 10
simulate 11
singlePermutation 12

Index **14**

`shiftR-package` *Fast Enrichment Analysis via Circular Permutations*

Description

Fast enrichment analysis for locally correlated statistics via circular permutations. The analysis can be performed at multiple significance thresholds for both primary and auxiliary data sets with with efficient correction for multiple testing.

Details

Package:	shiftR
Type:	Package
License:	LGPL-3
Depends:	methods

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

See Also

See the main function `shiftrPermBinary` for more info.

Run `browseVignettes("shiftR")` for the vignette.

`cramerV` *Calculate Cramer's V (phi) Coefficient*

Description

This functions calculates Cramer's V coefficient for overlap of two binary data sets.

Usage

```
cramerV(sum12, sum1, sum2, len)
```

Arguments

len	Total number of elements in each data set.
sum1	Number of active features in data set 1.
sum2	Number of active features in data set 1.
sum12	Number of simultaneously active features in the data sets.

Value

Returns the Cramer's V coefficient.

Note

The parameters can be single values or vectors.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

Examples

```
# Zero score for perfect independence
cramerV(100,10000,10000,1000000)

# Positive score for increased overlap
cramerV(150,10000,10000,1000000)

# Negative score for decreased overlap
cramerV( 50,10000,10000,1000000)

# We can input a vector for sum12
cramerV(99:101,10000,10000,1000000)
```

---

enrichmentAnalysis *Fast Enrichment Testing via Circular Permutations on Non-Binary Outcomes*

Description

This function performs enrichment analysis on two sets of matching test statistics. The circular permutation scheme accounts for possible local correlation of test statistcs. The testing is performed using the quantile thresholds provided for each data set.

For every permutation the enrichment is measure with Cramer's V coefficient. The maximum/minimum coefficient across all considered thresholds is recorded. It is then compared with the maximum/minimum coefficient observed without permuting the data.

For matching data sets calculated at different genomic locations please use matchDatasets.

Usage

```
enrichmentAnalysis(
    pvstats1,
    pvstats2,
    percentiles1 = NULL,
    percentiles2 = NULL,
    npermute,
    margin = 0.05,
    threads = 1)
```

Arguments

pvstats1	The vector of statistics for primary data set. The statistics must be p-value like, i.e. smaller is better.
pvstats2	The vector of statistics for auxiliary data set. The statistics must be p-value like, i.e. smaller is better.
percentiles1	These quantile thresholds are used to cut off top results in the primary data set for matching with the top results in the auxiliary. Can be omitted if the vector pvstats1 is binary.
percentiles2	Same as percentiles1, but for the other data set.
npermute	Number of permutations to perform.
margin	The minimum offset in the circular permutation to consider. Can be a fraction of total number of values or an integer count of values. Passed in the call of getOffsetsRandom for generation of offsets.
threads	The number of CPU cores to use for calculations. Set to TRUE to use all cores. Multithreading is turned off by default.

Value

Returns a list with:

overallPV	The p-values for the overall test across all thresholds. The p-values are for enrichment, depletion, and two-sided test for both.
byThresholdPV	The p-values for tests for each individual threshold. The p-values provided for enrichment, depletion, and two-sided test.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

Examples

```
### Data size
n = 1e5

### Generate vectors of test statistics with local correlation
```

```
window = 1000
pvstats1 = diff(cumsum(runif(n+window)), lag = window)
pvstats2 = diff(cumsum(runif(n+window)), lag = window)

# Add a bit of dependence
pvstats1 = pvstats1 + 0.5 * pvstats2

# test top 0.1, 1, 3, 5, and 10 percent

percentiles1 = c(0.001, 0.01, 0.03, 0.05, 0.1)
percentiles2 = c(0.001, 0.01, 0.03, 0.05, 0.1)

# The offset margin

margin = 0.05

# Set the number of permutations
# to the maximum

npermute = 1e3


enr = enrichmentAnalysis(
        pvstats1,
        pvstats2,
        percentiles1,
        percentiles2,
        npermute,
        margin ,
        threads = 2)

# View the results
enr
```

`getOffsets` *Generate Random or Uniformly Spaced Permutation Offsets*

Description

This functions generate offsets for permutation analysis with `shiftrPermBinary`. Random, uniformly spaced, and complete sets are available via `getOffsetsRandom`, `getOffsetsUniform`, and `getOffsetsAll` functions respectively.

The function `getNOffestsMax` calculates the maximum number of permutations (given the margin).

Usage

```
getOffsetsRandom(n, npermute, margin = 0.05)
getOffsetsUniform(n, npermute, margin = 0.05)
getOffsetsAll(n, margin)

getNOffsetsMax(n, margin)
```

Arguments

n	Number of features in the permuted sets.
npermute	The number of offsets to be generated (number of permutations).
margin	Offsets by less than margin*n or more than (1-margin)*n are not generated.

Value

Returns a set of permutation offsets for use in shiftrPermBinary function.
The set of offsets is

1. random for getOffsetsRandom,
2. uniformly spaced for getOffsetsUniform, or
3. all possible for getOffsetsAll.

The function getNOffestsMax returns the maximum number of permutations (given the margin).

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

Examples

```
### Number of features, permutations, and margin
n = 100
npermute = 20
margin = 0.1

### Maximum number of permutations
# Should be 81 (from 10 to 90)
getNOffsetsMax(n, margin)

### Random offsets
getOffsetsRandom(n, npermute, margin)

### Uniformly spaced offsets
getOffsetsUniform(n, npermute, margin)

### All possible offsets
getOffsetsAll(n, margin)
```

---

matchDatasets *Match Two Data Sets by Location*

Description

The goal of this function is to match records in the data sets for subsequent enrichment analysis.

For each record in the primary data set (data1) it finds the record in the auxiliary data set (data1) which overlap with it or lie within the flanking distance (flank). If multiple such auxiliary record are found, we select the one with the center closest to the center of the primary record. If no such record is available, no matching is made for the primary record.

Usage

```
matchDatasets(data1, data2, flank = 0)
```

Arguments

data1	A data frame with the primary data set, must have at least 4 columns: 1. Chromosome name. 2. Start position. 3. End position. 4. P-value or test statistic. 5. Optional additional columns.
data2	A data frame with the auxiliary data set. Must satisfy the same format criteria as the primary data set.
flank	Allowed distance between matched records. Set to zero to require overlap.

Value

Returns a list with matched data sets.

data1	The primary data sets without unmatched records.
data2	The auxiliary data set records matching those in data1 above. Note that some auxiliary records can get duplicated if they are the best match for multiple records in the primary data.

Note

For a technical reason, the chromosome positions are assumed to be no greater than 1e9.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

Examples

```
data1 = read.csv(text =
"chr,start,end,stat
chr1,100,200,1
chr1,150,250,2
chr1,200,300,3
chr1,300,400,4
chr1,997,997,5
chr1,998,998,6
chr1,999,999,7")

data2 = read.csv(text =
"chr,start,end,stat
chr1,130,130,1
```

```
chr1,140,140,2
chr1,165,165,3
chr1,200,200,4
chr1,240,240,5
chr1,340,340,6
chr1,350,350,7
chr1,360,360,8
chr1,900,900,9")

# Match data sets exactly.
matchDatasets(data1, data2, 0)

# Match data sets with a flank.
# The last records are now matched.
matchDatasets(data1, data2, 100)
```

`shiftrPermBinary` *Fast Enrichment Testing on Binary Outcomes via Circular Permutations*

Description

This function performs very fast feature enrichment permutation testing between two binary data sets. Circular permutations are used instead of simple permutations to preserve local dependence of test statistics. The input data sets can be preprocessed with `shiftrPrepareLeft` and `shiftrPrepareRight` functions.

Usage

```
shiftrPermBinary(
    left,
    right,
    offsets,
    alsoDoFisher = TRUE,
    returnPermOverlaps = FALSE)
```

Arguments

`left`	The first vector of binary (0/1) outcomes. For repeated use it can be preprocessed with `shiftrPrepareLeft` function.
`right`	The second vector of binary (0/1) outcomes. For repeated use it can be preprocessed with `shiftrPrepareRight` function.
`offsets`	Vector of offsets, can be generated by `getOffsetsRandom`, `getOffsetsUniform`, or `getOffsetsAll`.
`alsoDoFisher`	If `TRUE`, also perform Fisher exact test (via `fisher.test`).
`returnPermOverlaps`	If `TRUE` return overlap counts under all tested permutations.

Value

Returns a list with:

nfeatures	Number of features in input data sets.
lfeatures	Number of active features in the left data set.
rfeatures	Number of active features in the right data set.
overlap	Number of features simultaneously active in both data sets.
overlapUnderNull	Expected value of overlap if input data sets were independent.
enrichment	Enrichment ratio, equal to overlap / overlapUnderNull
permPVenrich	Permutation p-value for enrichment (one-sided).
permPVdeplete	Permutation p-value for depletion (one-sided).
permPV	Permutation p-value for depletion (two-sided).
permZ	Permutation z-statistic, calculated by fitting normal distribution to the overlap values under permutations. Positive values indicate enrichment.
fisherTest	Fisher exact test, as output by fisher.test
fisherMat	Input 2x2 matrix for Fisher exact test.
overlapsPerm	Vector of length npermute with overlap values under permutations.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

See Also

This function essentially involves npermute calls of singlePermutation function and calculation of summary statistics and p-values.

Examples

```
### Number of features
nf = 1e6
npermute = 10000

### Generate data sets
# The vector of a few common active feature to create dependence
common = sample(c(0L,1L), size = nf, replace = TRUE, prob = c(0.999,0.001))

# Left and right data sets with the common active features
lset = sample(c(0L,1L), size = nf, replace = TRUE, prob = c(0.8,0.2)) | common
rset = sample(c(0L,1L), size = nf, replace = TRUE, prob = c(0.8,0.2)) | common

offsets = getOffsetsUniform(n = nf, npermute = npermute)

show(head(offsets))
show(tail(offsets))
```

```
z = shiftrPermBinary(lset, rset, offsets)

show(z)
```

---

shiftrPrepare *Prepare Data for Fast Circular Permutation Analysis*

Description

The concept of circular permutations is symmetric with respect to the input data sets. The algorithm for circular permutation calculation is, however, not symmetric with respect to two datasets and thus the required data preprocessing is also different. For simplicity, we call the data sets 'left' and 'right'.

Usage

```
shiftrPrepareLeft(set)
shiftrPrepareRight(set)
```

Arguments

set	A 0/1 vector defining selected (genomic) features. The 'left' and 'right' sets must have equal length. The enrichment of their overlap can be assessed w ith shiftrPermBinary function.

Value

Returns objects of class fcpLeft and fcpRight respectively. The returned objects are used in singlePermutation and shiftrPermBinary functions.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

See Also

See codeshiftrPermBinary function and the respective example.

Examples

```
### Number of features
nf = 1e6

### Generate left and right sets
lset = sample(c(0L,1L), size = nf, replace = TRUE)
rset = sample(c(0L,1L), size = nf, replace = TRUE)

# Prepare binary sets:
```

```
lbin = shiftrPrepareLeft(lset)
rbin = shiftrPrepareRight(rset)

### Check object sizes
# Notice asymetry in binary object sizes

object.size(lset)
object.size(rset)
object.size(lbin)
object.size(rbin)
```

`simulate` *Generate Artificial Data for Tests and Illustrations*

Description

These functions generate two artificial data sets with local dependence of observations.

Usage

```
simulateNumeric(n, corWithin, corAcross = 0)
simulateBinary(n, corWithin, corAcross = 0)
```

Arguments

`n`	Total number of elements in each data set.
`corWithin`	Correlation of adjacent observations within each data set.
`corAcross`	Correlation of observations across data sets.

Value

Returns the Cramer's V coefficient.

Note

The `simulateNumeric` function generates two data sets with elements having standard normal distribution.

The `simulateBinary` function generates data sets with 0/1 values by thresholding the numeric data sets from `simulateNumeric`.

The `simulatePValues` function generates data sets of p-values by applying `pnorm` to the data sets from `simulateNumeric`.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

Examples

```
n = 100000
sim = simulateNumeric(n, 0.5, 0.3)

# Means should be close to 0 (zero)
mean(sim$data1)
mean(sim$data2)

# Variances should be close to 1
var(sim$data1)
var(sim$data2)

# Correlation of adjacent observations
# should be close to 0.5
cor(sim$data1[-1], sim$data1[-n])
cor(sim$data2[-1], sim$data2[-n])

# Correlation between data sets
# should be close to 0.3
cor(sim$data1, sim$data2)
```

---

singlePermutation *Count Feature Overlap Under a Permutation*

Description

This function performs fast feature overlap count under a circular permutation. The input data sets must be preprocessed with `shiftrPrepareLeft` and `shiftrPrepareRight` functions.

Usage

```
singlePermutation(left, right, offset)
```

Arguments

`left`	Feature set prepared with `shiftrPrepareLeft` function.
`right`	Feature set prepared with `shiftrPrepareRight` function.
`offset`	Offset of one feature set relative to another. See the example below for clarity. Zero indicate no offset, i.e. simply count feature overlap.

Value

Returns count of feature overlap under a circular permutation.

Author(s)

Andrey A Shabalin <andrey.shabalin@gmail.com>

Examples

```
### Number of features
nf = 1e6

### Generate left and right sets
lset = sample(c(0L,1L), size = nf, replace = TRUE)
rset = sample(c(0L,1L), size = nf, replace = TRUE) | lset

# Prepare binary sets:
lbin = shiftrPrepareLeft(lset)
rbin = shiftrPrepareRight(rset)

### count feature overlap
# R calculations
overlapS = sum(lset & rset)
# Binary calculations
overlapF = singlePermutation(lbin, rbin, 0)

message("Feature overlap: ",
        overlapS, " / ", overlapF,
        " (slow/fast count)")
stopifnot( overlapS == overlapF )


### Count overlap with offset
offset = 2017
# R calculations
overlapOS = sum(lset[ c((offset+1):nf, 1:offset)] & rset)
# Binary calculations
overlapOF = singlePermutation(lbin, rbin, offset)

message("Feature overlap at offset: ",
        overlapOS, " / ", overlapOF,
        " (slow/fast count)")
stopifnot( overlapOS == overlapOF )
```

Index

∗ **shiftR**
shiftR-package, 2

cramerV, 2

enrichmentAnalysis, 3

fisher.test, *8, 9*

getNOffsetsMax *(*getOffsets*)*, 5
getOffsets, 5
getOffsetsAll *(*getOffsets*)*, 5
getOffsetsRandom, *4*
getOffsetsRandom *(*getOffsets*)*, 5
getOffsetsUniform *(*getOffsets*)*, 5

matchDatasets, *3*, 6

pnorm, *11*

shiftR-package, 2
shiftrPermBinary, *2*, *5, 6*, 8, *10*
shiftrPrepare, 10
shiftrPrepareLeft, *8*, *12*
shiftrPrepareLeft *(*shiftrPrepare*)*, 10
shiftrPrepareRight, *8*, *12*
shiftrPrepareRight *(*shiftrPrepare*)*, 10
simulate, 11
simulateBinary *(*simulate*)*, 11
simulateNumeric *(*simulate*)*, 11
simulatePValues *(*simulate*)*, 11
singlePermutation, *9, 10*, 12