Package 'jazzPanda'

March 18, 2025

Type Package

Title Finding spatially relevant marker genes in image based spatial transcriptomics data

Version 0.99.6

Date 2024-07-25

LazyData FALSE

Depends R (>= 4.4.0)

Imports spatstat.geom, dplyr, glmnet, caret, foreach, stats, magrittr, doParallel, BiocParallel, methods, BumpyMatrix,SpatialExperiment

VignetteBuilder knitr

Suggests BiocStyle, knitr, rmarkdown, spatstat, Seurat, statmod, corrplot, ggplot2, ggraph, ggrepel, gridExtra, reshape2, igraph, jsonlite, vdiffr, patchwork, ggpubr, tidyr, SpatialFeatureExperiment, ExperimentHub, TENxXeniumData, SingleCellExperiment, SFEData, Matrix, data.table, scran, scater, grid, testthat (>= 3.0.0)

Description This package contains the function to find marker genes for image-based spatial transcriptomics data. There are functions to create spatial vectors from the cell and transcript coordiantes, which are passed as inputs to find marker genes. Marker genes are detected for every cluster by two approaches. The first approach is by permtuation testing, which is implmented in parallel for finding marker genes for one sample study. The other approach is to build a linear model for every gene. This approach can account for multiple samples and backgound noise.

License GPL-3

URL https://github.com/phipsonlab/jazzPanda, https://bhuvad.github.io/jazzPanda/

BugReports https://github.com/phipsonlab/jazzPanda/issues

biocViews Spatial, GeneExpression, DifferentialExpression, StatisticalMethod, Transcriptomics

RoxygenNote 7.3.2

Encoding UTF-8

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/jazzPanda

git_branch devel

git_last_commit 7b643fe

git_last_commit_date 2025-03-02

Repository Bioconductor 3.21

Date/Publication 2025-03-18

Author Melody Jin [aut, cre] (ORCID: <https://orcid.org/0000-0002-2222-0958>)

Maintainer Melody Jin <jin.m@wehi.edu.au>

Contents

jazzPanda-package *jazzPanda: A hybrid approach to find spatially relevant marker genes in image-based spatial transcriptomics data*

Description

jazzPanda pacakge provides hybrid approaches to prioritize marker genes that uses the spatial coordinates of gene detections and cells making up clusters. We propose a binning approach `get_vectors` that summarises the number of genes and cells within a cluster as spatial vectors. We have developed two approaches to detect and prioritize marker genes. The first approach `compute_permp` is based on correlation coefficients between genes and cluster spatial vectors, where significance of the marker genes are assessed through permutation. The second approach `lasso_markers` is based on lasso regularisation and linear modeling of our defined spatial vectors. This second approach is more flexible and can account for multiple samples and background noise.

Author(s)

Melody Jin <jin.m@wehi.edu.au>

.check_binning *helper function to check the input of binning*

Description

helper function to check the input of binning

Usage

```
.check_binning(bin_param, bin_type, w_x, w_y)
```

Arguments

`bin_param`	A numeric vector indicating the size of the bin. If the `bin_type` is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the `bin_type` is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
`bin_type`	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
`w_x`	A numeric vector of length two specifying the x coordinate limits of enclosing box.
`w_y`	A numeric vector of length two specifying the y coordinate limits of enclosing box.

Value

the length of total bins

`.check_valid_input` *helper function to check the inputs passed to marker detection function*

Description

helper function to check the inputs passed to marker detection function

Usage

```
.check_valid_input(
  gene_mt,
  cluster_mt,
  sample_names,
  n_fold = 10,
  background = NULL
)
```

Arguments

`gene_mt`	A matrix contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene. The column names must be specified and refer to the genes. This can be the output from the function `get_vectors`.
`cluster_mt`	A matrix contains the number of cells in a specific cluster in each grid. Each row refers to a grid, and each column refers to a cluster. The column names must be specified and refer to the clusters. Please do not assign integers as column names. This can be the output from the function `get_vectors`.
`sample_names`	A vector specifying the names for the samples.
`n_fold`	Optional. A positive number giving the number of folds used for cross validation. This parameter will pass to `cv.glmnet` to calculate a penalty term for every gene.
`background`	Optional. A matrix providing the background information. Each row refers to a grid, and each column refers to one category of background information. Number of rows must equal to the number of rows in `gene_mt` and `cluster_mt`. Can be obtained by only providing coordinates matrices `cluster_info`. to function `get_vectors`.

Value

a list of two matrices with the following components

`n_clusters`	Number of clusters
`cluster_names`	a vector of strings giving the name of the clusters

.check_valid_names *helper function to check the names of gene/cluster/sample*

Description

helper function to check the names of gene/cluster/sample

Usage

```
.check_valid_names(x, x_name)
```

Arguments

x	A character vector to check naming
x_name	A name specifying the type of x, for message purpose only

Value

A character vector of same length with valid names

.compute_observation *Compute observation statistic for permutation framework*

Description

Compute observation statistic for permutation framework

Usage

```
.compute_observation(
  x,
  cluster_info,
  correlation_method,
  n_cores,
  test_genes,
  bin_type,
  bin_param,
  w_x,
  w_y,
  use_cm
)
```

Arguments

x	a named list (of transcript detection coordinates) or SingleCellExperiment or SpatialExperiment or SpatialFeatureExperiment object. If a named list is provided, every list element is a dataframe containing the transcript detection coordinates and column names must include "feature_name" (gene name), "x" (x coordinate), "y" (y coordinate). The list names must match samples in cluster_info.
cluster_info	A dataframe/matrix containing the centroid coordinates, cluster label and sample for each cell.The column names must include "x" (x coordinate), "y" (y coordinate), "cluster" (cluster label) and "sample" (sample). It is strongly recommended to use syntactically valid names for columns clusters and samples. If invalid names are detected, the function make.names will be employed to generate valid names. A message will also be displayed to indicate this change.
correlation_method	A parameter pass to cor, indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman": can be abbreviated.
n_cores	A positive number specifying number of cores used for parallelizing permutation testing. Default is one core (sequential processing).
test_genes	A vector of strings giving the name of the genes you want to test correlation for.
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the bin_type is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the bin_type is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
w_x	a numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	a numeric vector of length two specifying the y coordinate limits of enclosing box.
use_cm	A boolean value that specifies whether to create spatial vectors for genes using the count matrix and cell coordinates instead of the transcript coordinates when both types of information are available. The default setting is FALSE.

Value

A named list with the following components

obs.stat	A matrix contains the observation statistic for every gene and every cluster. Each row refers to a gene, and each column refers to a cluster
gene_mt	contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene.

`.compute_permutation` *Compute permutation statistics for permutation framework*

Description

Compute permutation statistics for permutation framework

Usage

```
.compute_permutation(
  cluster_info,
  perm.size = 1000,
  correlation_method = "pearson",
  bin_type,
  bin_param,
  n_cores = 1,
  w_x,
  w_y,
  gene_mt,
  cluster_names
)
```

Arguments

`cluster_info`	A dataframe/matrix containing the centroid coordinates and cluster label for each cell.The column names should include "x" (x coordinate), "y" (y coordinate), and "cluster" (cluster label).
`perm.size`	A positive number specifying permutation times
`correlation_method`	A parameter pass to `cor`, indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman": can be abbreviated.
`bin_type`	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
`bin_param`	A numeric vector indicating the size of the bin. If the `bin_type` is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the `bin_type` is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
`n_cores`	A positive number specifying number of cores used for parallelizing permutation testing. Default is one core (sequential processing).
`w_x`	a numeric vector of length two specifying the x coordinate limits of enclosing box.
`w_y`	a numeric vector of length two specifying the y coordinate limits of enclosing box.
`gene_mt`	A matrix contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene.
`cluster_names`	A list of strings giving the name and order of the clusters

Value

A matrix with permutation statistics

`.convert_data` *Convert SingleCellExperiment/SpatialExperiment/SpatialFeatureExperiment objects to list object for jazzPanda.*

Description

This function takes an object of class SingleCellExperiment, SpatialExperiment or SpatialFeature-Experimentreturns and returns a list object that is expected for the `get_vector` functions.

Usage

```
.convert_data(x, sample_names, test_genes)
```

Arguments

`x`	a SingleCellExperiment or SpatialExperiment or SpatialFeatureExperiment object
`sample_names`	a vector of strings giving the sample names
`test_genes`	A vector of strings giving the name of the genes you want to create gene vector.

Value

outputs a list object with the following components

`trans_lst`	A list of named dataframes. Each dataframe refers to one sample and shows the transcript detection coordinates for each gene. The name matches the input sample_names
`cm_lst`	A list of named dataframes containing the count matrix for each sample. The name matches the input sample_names

---

`.create_cor_mg_result` *Create a marker gene result object for correlation approach*

Description

This function creates a structured output object named 'cor_mg_result' for storing the permutation results. The object contains three matrices:

Usage

```
.create_cor_mg_result(obs.stat, perm.pval, perm.pval.adj)
```

Arguments

obs.stat	A matrix containing the correlation coefficients for each pair of genes and cluster vectors.
perm.pval	A matrix containing the raw permutation p-value for each pair of genes and cluster.
perm.pval.adj	A matrix containing the adjusted permutation p-value for each pair of genes and cluster.

Value

An S3 object of class 'cor_mg_result' which includes three matrices.

.create_lm_mg_result *Create a marker gene result object for linear modelling approach*

Description

This function creates a structured output object named 'glm_mg_result' for storing the marker gene results. The object contains two data frames: top results and full results.

Usage

```
.create_lm_mg_result(top_result, full_result)
```

Arguments

top_result	A data frame containing top results.
full_result	A data frame containing full results.

Value

An S3 object of class 'glm_mg_result' which includes both results data frames.

.get_cluster_vectors *Create spatial vectors for clusters*

Description

Create spatial vectors for clusters

Usage

```
.get_cluster_vectors(
  cluster_info,
  bin_length,
  bin_type,
  bin_param,
  w_x,
  w_y,
  sample_names
)
```

Arguments

cluster_info	A dataframe/matrix containing the centroid coordinates, cluster label and sample for each cell.The column names must include "x" (x coordinate), "y" (y coordinate), "cluster" (cluster label) and "sample" (sample).
bin_length	A positive integer giving the length of total bins
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the bin_type is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the bin_type is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
w_x	A numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	A numeric vector of length two specifying the y coordinate limits of enclosing box.
sample_names	a vector of strings giving the sample names

Value

a matrix contains the cell count in each grid. Each row refers to a grid, and each column refers to a cluster.

.get_gene_vectors_cm *Create spatial vectors for genes from count matrix and cell coordinates*

Description

This function will build gene vectors with count matrix and cell locations

Usage

```
.get_gene_vectors_cm(
  cluster_info,
  cm_lst,
  bin_type,
  bin_param,
  test_genes,
  w_x,
  w_y
)
```

Arguments

cluster_info	A dataframe/matrix containing the centroid coordinates, cluster label and sample for each cell.The column names must include "x" (x coordinate), "y" (y coordinate), "cluster" (cluster label) and "sample" (sample).
cm_lst	A list of named matrices containing the count matrix for each sample The name must match the sample column in cluster_info. If this input is provided, the cluster_info must be specified and contain an additional column "cell_id" to link cell location and count matrix. Default is NULL.
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the bin_type is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the bin_type is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
test_genes	A vector of strings giving the name of the genes you want to test. This will be used as column names for one of the result matrix gene_mt.
w_x	A numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	A numeric vector of length two specifying the y coordinate limits of enclosing box.

Value

a matrix contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene.

.get_gene_vectors_tr *Create spatial vectors for genes from transcript coordinates*

Description

This function will build gene vectors based on the transcript coordinates of every gene

Usage

```
.get_gene_vectors_tr(
  trans_lst,
  test_genes,
  bin_type,
  bin_param,
  bin_length,
  w_x,
  w_y
)
```

Arguments

trans_lst	If specified, it is a list of named dataframes. Each dataframe refers to one sample and shows the transcript detection coordinates for each gene. Optional parameter.
test_genes	A vector of strings giving the name of the genes you want to test. This will be used as column names for one of the result matrix gene_mt.
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the bin_type is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the bin_type is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
bin_length	A positive integer giving the length of total bins
w_x	A numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	A numeric vector of length two specifying the y coordinate limits of enclosing box.

Value

a matrix contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene.

.get_lasso_coef *help function to get lasso coefficient for every cluster for a given model*

Description

help function to get lasso coefficient for every cluster for a given model

Usage

```
.get_lasso_coef(
  i_gene,
  gene_mt,
  vec_cluster,
  cluster_names,
  n_fold = 10,
  n_samples,
  sample_names
)
```

Arguments

i_gene	Name of the current gene
gene_mt	A matrix contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene. The column names must be specified and refer to the genes. This can be the output from the function get_vectors.
vec_cluster	A matrix of the spatial vectors for clusters.
cluster_names	A vector of strings giving the name of clusters
n_fold	Optional. A positive number giving the number of folds used for cross validation. This parameter will pass to cv.glmnet to calculate a penalty term for every gene.
n_samples	A positive number giving the number samples
sample_names	A vector specifying the names for the sample

Value

a list of two matrices with the following components

coef_df	A matrix giving the lasso coefficient of each cluster
lambda.1se	the lambda.1se value of best fitted model

---

compute_permp *Calculate a p-value for correlation with permutation.*

Description

This function will run permutation framework to compute a p-value for the correlation between the vectorised genes and clusters each cluster for one sample.

Usage

```
compute_permp(
  x,
  cluster_info,
  perm.size,
  bin_type,
  bin_param,
  test_genes,
  correlation_method = "pearson",
  n_cores = 1,
  correction_method = "BH",
  w_x,
  w_y,
  use_cm = FALSE
)
```

Arguments

x	a SingleCellExperiment or SpatialExperiment or SpatialFeatureExperiment object
cluster_info	A dataframe/matrix containing the centroid coordinates and cluster label for each cell.The column names should include "x" (x coordinate), "y" (y coordinate), and "cluster" (cluster label).
perm.size	A positive number specifying permutation times
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the bin_type is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the bin_type is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
test_genes	A vector of strings giving the name of the genes you want to test correlation for. gene_mt.
correlation_method	A parameter pass to cor indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman": can be abbreviated.
n_cores	A positive number specifying number of cores used for parallelizing permutation testing. Default is one core (sequential processing).
correction_method	A character string pass to p.adjust specifying the correction method for multiple testing .
w_x	a numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	a numeric vector of length two specifying the y coordinate limits of enclosing box.

use_cm A boolean value that specifies whether to create spatial vectors for genes using the count matrix and cell coordinates instead of the transcript coordinates when both types of information are available. The default setting is FALSE.

Details

To get a permutation p-value for the correlation between a gene and a cluster, this function will permute the cluster label for each cell randomly, and calculate correlation between the genes and permuted clusters. This process will be repeated for `perm.size` times, and permutation p-value is calculated as the probability of permuted correlations larger than the observation correlation.

Value

An object of class 'cor_mg_result'. To access specific components of the returned object:

- Use `get_cor` to retrieve the matrix of observed correlation coefficients.
- Use `get_perm_p` to access the matrix of raw permutation p-values.
- Use `get_perm_adjp` to obtain the matrix of adjusted permutation p-values.

Examples

```
library(SpatialExperiment)
library(BumpyMatrix)
set.seed(100)
# simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=10, mean=20, sd=5),
                     y = rnorm(n=10, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=10, mean=100, sd=5),
                     y = rnorm(n=10, mean=100, sd=5), cluster="B")
clusters <- rbind(df_clA, df_clB)
clusters$sample="sample1"
# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A1"),
                     cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A2"),
                     cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B1"),
                     cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B2")))
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell =  rep(paste("cell",1:20, sep=""), times=2)
mol <- BumpyMatrix::splitAsBumpyMatrix(
     trans_info[, c("x", "y")],
     row = trans_info$feature_name, col = trans_info$cell )
spe_sample1 <- SpatialExperiment(
```

```
        assays = list(molecules = mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
             floor(min(clusters$x))),
         max(ceiling(max(trans_info$x)),
             ceiling(max(clusters$x))))
w_y <-  c(min(floor(min(trans_info$y)),
             floor(min(clusters$y))),
         max(ceiling(max(trans_info$y)),
             ceiling(max(clusters$y))))
set.seed(100)
corr_res <- compute_permp(x=spe_sample1,
             cluster_info=clusters,
             perm.size=10,
             bin_type="square",
             bin_param=c(2,2),
             test_genes=unique(trans_info$feature_name),
             correlation_method = "pearson",
             n_cores=1,
             correction_method="BH",
             w_x=w_x ,
             w_y=w_y)

# raw permutation p-value
perm_p <- get_perm_p(corr_res)
# adjusted permutation p-value
adjusted_perm_p <- get_perm_adjp(corr_res)
# observed correlation
obs_corr <- get_cor(corr_res)
```

`create_genesets` *Convert the coordinates of set of genes into vectors.*

Description

Convert the coordinates of set of genes into vectors.

Usage

```
create_genesets(
  x,
  name_lst,
  cluster_info,
  sample_names,
  bin_type,
  bin_param,
  w_x,
  w_y,
  use_cm = FALSE,
  n_cores = 1
)
```

Arguments

x	a named list (of transcript detection coordinates) or SingleCellExperiment or SpatialExperiment or SpatialFeatureExperiment object. If a named list is provided, every list element is a dataframe containing the transcript detection coordinates and column names must include "feature_name" (nagative control name), "x" (x coordinate) and "y" (y coordinate). The list names must match samples in cluster_info.
name_lst	A named list of strings giving the name of features that are treated as background.
cluster_info	A dataframe/matrix containing the centroid coordinates, cluster and sample label for each cell.The column names must include "x" (x coordinate), "y" (y coordinate), "cluster" (cluster label) and "sample" (sample).
sample_names	a vector of strings giving the sample names
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the bin_type is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the bin_type is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
w_x	A numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	A numeric vector of length two specifying the y coordinate limits of enclosing box.
use_cm	A boolean value that specifies whether to create spatial vectors for genes using the count matrix and cell coordinates instead of the transcript coordinates when both types of information are available. The default setting is FALSE.
n_cores	A positive number specifying number of cores used for parallelizing permutation testing. Default is one core (sequential processing).

Value

a list of two matrices with the following components

gene_mt	contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene.
cluster_mt	contains the number of cells in a specific cluster in each grid. Each row refers to a grid, and each column refers to a cluster.

The row order of gene_mt matches the row order of cluster_mt.

A matrix contains the sum count in each grid. Each row refers to a grid, each column refers to a set in name_lst. The column name will match the names in name_lst.

Examples

```
library(SpatialExperiment)
set.seed(15)
trans = as.data.frame(rbind(cbind(x = runif(10, min=1, max=10),
                          y = runif(10, min=1, max=10),
                                 feature_name="A"),
                          cbind(x = runif(5, min=10, max=24),
                                y = runif(5, min=1, max=10),
                                feature_name="B"),
                          cbind(x = runif(10, min=10, max=24),
                                y = runif(10, min=10, max=24),
                                feature_name="C")))
trans$x = as.numeric(trans$x)
trans$y = as.numeric(trans$y)
trans$cell = sample(c("cell1","cell2","cell2"),replace=TRUE,
                         size=nrow(trans))
# create SpatialExperiment object
trans_mol <- BumpyMatrix::splitAsBumpyMatrix(
    trans[, c("x", "y")],
    row = trans$feature_name, col = trans$cell )
rep1_spe<- SpatialExperiment(
     assays = list(molecules = trans_mol),sample_id ="sample1" )
geneset_res <- create_genesets(x=rep1_spe, sample=c("sample1"),
                               name_lst=list(dummy_A=c("A","C"),
                                             dummy_B=c("A","B","C")),
                               bin_type="square",
                               bin_param=c(2,2),
                               w_x=c(0,25), w_y=c(0,25), cluster_info=NULL)
```

---

get_cor *Get observed correlation cor_mg_result*

Description

Accessor function to retrieve the observed correlation from an 'cor_mg_result' object.

Usage

```
get_cor(obj)
```

Arguments

obj	An 'cor_mg_result' object.

Value

A matrix contains the observation statistic for every gene and every cluster. Each row refers to a gene, and each column refers to a cluster

Examples

```
library(SpatialExperiment)
library(BumpyMatrix)
set.seed(100)
# simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=10, mean=20, sd=5),
                     y = rnorm(n=10, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=10, mean=100, sd=5),
                     y = rnorm(n=10, mean=100, sd=5), cluster="B")
clusters <- rbind(df_clA, df_clB)
clusters$sample="sample1"
# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A1"),
                     cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A2"),
                     cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B1"),
                     cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B2")))
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell =  rep(paste("cell",1:20, sep=""), times=2)
mol <- BumpyMatrix::splitAsBumpyMatrix(
     trans_info[, c("x", "y")],
     row = trans_info$feature_name, col = trans_info$cell )
spe_sample1 <- SpatialExperiment(
        assays = list(molecules = mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
             floor(min(clusters$x))),
         max(ceiling(max(trans_info$x)),
             ceiling(max(clusters$x))))
w_y <-  c(min(floor(min(trans_info$y)),
             floor(min(clusters$y))),
         max(ceiling(max(trans_info$y)),
             ceiling(max(clusters$y))))
set.seed(100)
corr_res <- compute_permp(x=spe_sample1,
             cluster_info=clusters,
             perm.size=10,
             bin_type="square",
             bin_param=c(2,2),
             test_genes=unique(trans_info$feature_name),
             correlation_method = "pearson",
             n_cores=1,
             correction_method="BH",
             w_x=w_x ,
             w_y=w_y)
```

```
# observed correlation
obs_corr <- get_cor(corr_res)
```

`get_full_mg` *Get full lasso result from glm_mg_result*

Description

Accessor function to retrieve the 'full_result' dataframe from an 'glm_mg_result' object.

Usage

```
get_full_mg(obj, coef_cutoff = 0)
```

Arguments

`obj`	An 'glm_mg_result' object.
`coef_cutoff`	A positive number giving the coefficient cutoff value. Genes whose cluster showing a coefficient value smaller than the cutoff will be removed. Default is 0.

Value

A data frame with detailed information for each gene and the most relevant cluster label.

- `gene` Gene name
- `cluster` The name of the significant cluster after
- `glm_coef` The coefficient of the selected cluster in the generalised linear model.
- `pearson` Pearson correlation between the gene vector and the selected cluster vector.
- `max_gg_corr` A number showing the maximum pearson correlation for this gene vector and all other gene vectors in the input `gene_mt`
- `max_gc_corr` A number showing the maximum pearson correlation for this gene vector and every cluster vectors in the input `cluster_mt`

Examples

```
library(SpatialExperiment)
set.seed(100)
#  simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=100, mean=20, sd=5),
                     y = rnorm(n=100, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=100, mean=100, sd=5),
                    y = rnorm(n=100, mean=100, sd=5), cluster="B")

clusters <- rbind(df_clA, df_clB)
clusters$sample<-"sample1"
```

```
# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=100, mean=20,sd=5),
                                y = rnorm(n=100, mean=20, sd=5),
                                 feature_name="gene_A1"),
                           cbind(x = rnorm(n=100, mean=20, sd=5),
                                 y = rnorm(n=100, mean=20, sd=5),
                                 feature_name="gene_A2"),
                           cbind(x = rnorm(n=100, mean=100, sd=5),
                                 y = rnorm(n=100, mean=100, sd=5),
                                 feature_name="gene_B1"),
                           cbind(x = rnorm(n=100, mean=100, sd=5),
                                 y = rnorm(n=100, mean=100, sd=5),
                                 feature_name="gene_B2")))
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell<-sample(c("cell1","cell2","cell2"),replace=TRUE,
                        size=nrow(trans_info))
trans_mol <- BumpyMatrix::splitAsBumpyMatrix(
    trans_info[, c("x", "y")],
    row = trans_info$feature_name, col = trans_info$cell )
spe<- SpatialExperiment(
     assays = list(molecules = trans_mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
         floor(min(clusters$x))),
      max(ceiling(max(trans_info$x)),
          ceiling(max(clusters$x))))
w_y <- c(min(floor(min(trans_info$y)),
          floor(min(clusters$y))),
      max(ceiling(max(trans_info$y)),
          ceiling(max(clusters$y))))
vecs_lst <- get_vectors(x=spe,sample_names=c("sample1"),
                    cluster_info = clusters,
                    bin_type = "square",
                    bin_param = c(20,20),
                    test_genes =c("gene_A1","gene_A2","gene_B1","gene_B2"),
                    w_x = w_x, w_y=w_y)
lasso_res <- lasso_markers(gene_mt=vecs_lst$gene_mt,
                        cluster_mt = vecs_lst$cluster_mt,
                        sample_names=c("sample1"),
                        keep_positive=TRUE,
                        background=NULL)
# the full result
full_result <- get_full_mg(lasso_res, coef_cutoff=0.05)
```

`get_perm_adjp` *Get permutation adjusted p value from cor_mg_result*

Description

Accessor function to retrieve the permutation adjusted p-value from an 'cor_mg_result' object.

Usage

```
get_perm_adjp(obj)
```

Arguments

obj	An 'cor_mg_result' object.

Value

A matrix contains the adjusted permutation p-value. Each row refers to a gene, and each column refers to a cluster.

Examples

```
library(SpatialExperiment)
library(BumpyMatrix)
set.seed(100)
# simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=10, mean=20, sd=5),
                     y = rnorm(n=10, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=10, mean=100, sd=5),
                     y = rnorm(n=10, mean=100, sd=5), cluster="B")
clusters <- rbind(df_clA, df_clB)
clusters$sample="sample1"
# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A1"),
                     cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A2"),
                     cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B1"),
                     cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B2")))
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell =  rep(paste("cell",1:20, sep=""), times=2)
mol <- BumpyMatrix::splitAsBumpyMatrix(
     trans_info[, c("x", "y")],
     row = trans_info$feature_name, col = trans_info$cell )
spe_sample1 <- SpatialExperiment(
        assays = list(molecules = mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
             floor(min(clusters$x))),
         max(ceiling(max(trans_info$x)),
             ceiling(max(clusters$x))))
w_y <-  c(min(floor(min(trans_info$y)),
             floor(min(clusters$y))),
         max(ceiling(max(trans_info$y)),
```

```
              ceiling(max(clusters$y))))
set.seed(100)
corr_res <- compute_permp(x=spe_sample1,
              cluster_info=clusters,
              perm.size=10,
              bin_type="square",
              bin_param=c(2,2),
              test_genes=unique(trans_info$feature_name),
              correlation_method = "pearson",
              n_cores=1,
              correction_method="BH",
              w_x=w_x ,
              w_y=w_y)
# adjusted permutation p-value
adjusted_perm_p <- get_perm_adjp(corr_res)
```

---

get_perm_p *Get permutation p value from cor_mg_result*

Description

Accessor function to retrieve the raw permutation p-value from an 'cor_mg_result' object.

Usage

```
get_perm_p(obj)
```

Arguments

obj	An 'cor_mg_result' object.

Value

A matrix contains the raw permutation p-value. Each row refers to a gene, and each column refers to a cluster.

Examples

```
library(SpatialExperiment)
library(BumpyMatrix)
set.seed(100)
# simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=10, mean=20, sd=5),
                     y = rnorm(n=10, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=10, mean=100, sd=5),
                     y = rnorm(n=10, mean=100, sd=5), cluster="B")
clusters <- rbind(df_clA, df_clB)
clusters$sample<-"sample1"
# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=10, mean=20, sd=5),
```

```
                                    y = rnorm(n=10, mean=20, sd=5),
                                    feature_name="gene_A1"),
                        cbind(x = rnorm(n=10, mean=20, sd=5),
                                    y = rnorm(n=10, mean=20, sd=5),
                                    feature_name="gene_A2"),
                        cbind(x = rnorm(n=10, mean=100, sd=5),
                                    y = rnorm(n=10, mean=100, sd=5),
                                    feature_name="gene_B1"),
                        cbind(x = rnorm(n=10, mean=100, sd=5),
                                    y = rnorm(n=10, mean=100, sd=5),
                                    feature_name="gene_B2")))
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell =  rep(paste("cell",1:20, sep=""), times=2)
mol <- BumpyMatrix::splitAsBumpyMatrix(
     trans_info[, c("x", "y")],
     row = trans_info$feature_name, col = trans_info$cell )
spe_sample1 <- SpatialExperiment(
        assays = list(molecules = mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
             floor(min(clusters$x))),
         max(ceiling(max(trans_info$x)),
             ceiling(max(clusters$x))))
w_y <-  c(min(floor(min(trans_info$y)),
             floor(min(clusters$y))),
          max(ceiling(max(trans_info$y)),
             ceiling(max(clusters$y))))
set.seed(100)
corr_res <- compute_permp(x=spe_sample1,
             cluster_info=clusters,
             perm.size=10,
             bin_type="square",
             bin_param=c(2,2),
             test_genes=unique(trans_info$feature_name),
             correlation_method = "pearson",
             n_cores=1,
             correction_method="BH",
             w_x=w_x ,
             w_y=w_y)

# raw permutation p-value
perm_p <- get_perm_p(corr_res)
```

---

get_top_mg *Get top lasso result from glm_mg_result*

Description

Accessor function to retrieve the 'top_result' dataframe from an 'glm_mg_result' object.

Usage

```
get_top_mg(obj, coef_cutoff = 0.05)
```

Arguments

obj	An 'glm_mg_result' object.
coef_cutoff	A positive number giving the coefficient cutoff value. Genes whose top cluster showing a coefficient value smaller than the cutoff will be marked as non-marker genes ("NoSig"). Default is 0.05.

Value

A data frame with detailed information for each gene and the most relevant cluster label.

- gene Gene name
- top_cluster The name of the most relevant cluster after thresholding the coefficients.
- glm_coef The coefficient of the selected cluster in the generalised linear model.
- pearson Pearson correlation between the gene vector and the selected cluster vector.
- max_gg_corr A number showing the maximum pearson correlation for this gene vector and all other gene vectors in the input gene_mt
- max_gc_corr A number showing the maximum pearson correlation for this gene vector and every cluster vectors in the input cluster_mt

Examples

```
library(SpatialExperiment)
set.seed(100)
#  simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=100, mean=20, sd=5),
                 y = rnorm(n=100, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=100, mean=100, sd=5),
                y = rnorm(n=100, mean=100, sd=5), cluster="B")

clusters <- rbind(df_clA, df_clB)
clusters$sample<-"sample1"

# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=100, mean=20,sd=5),
                                y = rnorm(n=100, mean=20, sd=5),
                                 feature_name="gene_A1"),
                           cbind(x = rnorm(n=100, mean=20, sd=5),
                                 y = rnorm(n=100, mean=20, sd=5),
                                 feature_name="gene_A2"),
                           cbind(x = rnorm(n=100, mean=100, sd=5),
                                 y = rnorm(n=100, mean=100, sd=5),
                                 feature_name="gene_B1"),
                           cbind(x = rnorm(n=100, mean=100, sd=5),
                                 y = rnorm(n=100, mean=100, sd=5),
                                 feature_name="gene_B2")))
```

```
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell<-sample(c("cell1","cell2","cell2"),replace=TRUE,
                        size=nrow(trans_info))
trans_mol <- BumpyMatrix::splitAsBumpyMatrix(
    trans_info[, c("x", "y")],
    row = trans_info$feature_name, col = trans_info$cell )
spe<- SpatialExperiment(
     assays = list(molecules = trans_mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
         floor(min(clusters$x))),
      max(ceiling(max(trans_info$x)),
          ceiling(max(clusters$x))))
w_y <- c(min(floor(min(trans_info$y)),
          floor(min(clusters$y))),
      max(ceiling(max(trans_info$y)),
          ceiling(max(clusters$y))))
vecs_lst <- get_vectors(x=spe,sample_names=c("sample1"),
                    cluster_info = clusters,
                    bin_type = "square",
                    bin_param = c(20,20),
                    test_genes =c("gene_A1","gene_A2","gene_B1","gene_B2"),
                    w_x = w_x, w_y=w_y)
lasso_res <- lasso_markers(gene_mt=vecs_lst$gene_mt,
                        cluster_mt = vecs_lst$cluster_mt,
                        sample_names=c("sample1"),
                        keep_positive=TRUE,
                        background=NULL)
# the top result
top_result<- get_top_mg(lasso_res, coef_cutoff=0.05)
```

`get_vectors` *Vectorise the spatial coordinates*

Description

This function will convert the coordinates into a numeric vector for genes and clusters.

Usage

```
get_vectors(
  x,
  cluster_info,
  sample_names,
  bin_type,
  bin_param,
  test_genes,
  w_x,
  w_y,
```

```
  use_cm = FALSE,
  n_cores = 1
)
```

Arguments

x	a named list (of transcript detection coordinates) or SingleCellExperiment or SpatialExperiment or SpatialFeatureExperiment object. If a named list is provided, every list element is a dataframe containing the transcript detection coordinates and column names must include "feature_name" (gene name), "x" (x coordinate), "y" (y coordinate). The list names must match samples in cluster_info.
cluster_info	A dataframe/matrix containing the centroid coordinates, cluster label and sample for each cell.The column names must include "x" (x coordinate), "y" (y coordinate), "cluster" (cluster label) and "sample" (sample). It is strongly recommended to use syntactically valid names for columns clusters and samples. If invalid names are detected, the function `make.names` will be employed to generate valid names. A message will also be displayed to indicate this change.
sample_names	a vector of strings giving the sample names. It is strongly recommended to use syntactically valid names for columns clusters and samples. If invalid names are detected, the function `make.names` will be employed to generate valid names. A message will also be displayed to indicate this change.
bin_type	A string indicating which bin shape is to be used for vectorization. One of "square" (default), "rectangle", or "hexagon".
bin_param	A numeric vector indicating the size of the bin. If the `bin_type` is "square" or "rectangle", this will be a vector of length two giving the numbers of rectangular quadrats in the x and y directions. If the `bin_type` is "hexagonal", this will be a number giving the side length of hexagons. Positive numbers only.
test_genes	A vector of strings giving the name of the genes you want to create gene vector. This will be used as column names for one of the result matrix `gene_mt`.
w_x	A numeric vector of length two specifying the x coordinate limits of enclosing box.
w_y	A numeric vector of length two specifying the y coordinate limits of enclosing box.
use_cm	A boolean value that specifies whether to create spatial vectors for genes using the count matrix and cell coordinates instead of the transcript coordinates when both types of information are available. The default setting is FALSE.
n_cores	A positive number specifying number of cores used for parallelizing permutation testing. Default is one core (sequential processing).

Details

This function can be used to generate input for `lasso_markers` by specifying all the parameters.

Suppose the input data contains n genes, c clusters, and k samples, we want to use $a \times a$ square bin to convert the coordinates of genes and clusters into 1d vectors.

If $k = 1$, the returned list will contain one matrix for gene vectors (gene_mt) of dimension $a^2 \times n$ and one matrix for cluster vectors (cluster_mt) of dimension $a^2 \times c$.

If $k > 1$, gene and cluster vectors are constructed for each sample separately and concat together. There will be additional k columns on the returned cluster_mt, which is the one-hot encoding of the sample information.

Moreover, this function can vectorise genes and clusters separately based on the input. If x is NULL, this function will return vectorised clusters based on cluster_info. If cluster_info is NULL, this function will return vectorised genes based on x.

Value

a list of two matrices with the following components

gene_mt	contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene.
cluster_mt	contains the number of cells in a specific cluster in each grid. Each row refers to a grid, and each column refers to a cluster.

The row order of gene_mt matches the row order of cluster_mt.

Examples

```
library(SpatialExperiment)
set.seed(100)
#  simulate coordinates for clusters
df_clA = data.frame(x = rnorm(n=100, mean=20, sd=5),
                    y = rnorm(n=100, mean=20, sd=5), cluster="A")
df_clB = data.frame(x = rnorm(n=100, mean=100, sd=5),
                   y = rnorm(n=100, mean=100, sd=5), cluster="B")

clusters = rbind(df_clA, df_clB)
clusters$sample="sample1"

# simulate coordinates for genes
trans_info = data.frame(rbind(cbind(x = rnorm(n=10, mean=20,sd=5),
                                    y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A1"),
                              cbind(x = rnorm(n=10, mean=20, sd=5),
                                     y = rnorm(n=10, mean=20, sd=5),
                                     feature_name="gene_A2"),
                              cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B1"),
                              cbind(x = rnorm(n=10, mean=100, sd=5),
                                     y = rnorm(n=10, mean=100, sd=5),
                                     feature_name="gene_B2")))
trans_info$x=as.numeric(trans_info$x)
trans_info$y=as.numeric(trans_info$y)
trans_info$cell = sample(c("cell1","cell2","cell2"),replace=TRUE,
                         size=nrow(trans_info))
w_x =  c(min(floor(min(trans_info$x)),
```

```
         floor(min(clusters$x))),
      max(ceiling(max(trans_info$x)),
          ceiling(max(clusters$x))))
w_y =  c(min(floor(min(trans_info$y)),
          floor(min(clusters$y))),
      max(ceiling(max(trans_info$y)),
          ceiling(max(clusters$y))))
# use named list as input
vecs_lst = get_vectors(x= list("sample1" = trans_info),
                    sample_names=c("sample1"),
                    cluster_info = clusters,
                    bin_type = "square",
                    bin_param = c(5,5),
                    test_genes =c("gene_A1","gene_A2","gene_B1","gene_B2"),
                    w_x = w_x, w_y=w_y)
# use SpatialExperiment object as input
trans_mol <- BumpyMatrix::splitAsBumpyMatrix(
    trans_info[, c("x", "y")],
    row = trans_info$feature_name, col = trans_info$cell )
spe<- SpatialExperiment(
     assays = list(molecules = trans_mol),sample_id ="sample1" )
vecs_lst_spe = get_vectors(x=spe,sample_names=c("sample1"),
                    cluster_info = clusters,
                    bin_type = "square",
                    bin_param = c(5,5),
                    test_genes =c("gene_A1","gene_A2","gene_B1","gene_B2"),
                    w_x = w_x, w_y=w_y)
```

`lasso_markers` *Find marker genes with spatial coordinates*

Description

This function will find the most spatially relevant cluster label for each gene.

Usage

```
lasso_markers(
  gene_mt,
  cluster_mt,
  sample_names,
  keep_positive = TRUE,
  background = NULL,
  n_fold = 10
)
```

Arguments

gene_mt	A matrix contains the transcript count in each grid. Each row refers to a grid, and each column refers to a gene. The column names must be specified and refer to the genes. This can be the output from the function get_vectors.
cluster_mt	A matrix contains the number of cells in a specific cluster in each grid. Each row refers to a grid, and each column refers to a cluster. The column names must be specified and refer to the clusters. Please do not assign integers as column names. This can be the output from the function get_vectors.
sample_names	A vector specifying the names for the samples.
keep_positive	A logical flag indicating whether to return positively correlated clusters or not.
background	Optional. A matrix providing the background information. Each row refers to a grid, and each column refers to one category of background information. Number of rows must equal to the number of rows in gene_mt and cluster_mt. Can be obtained by only providing coordinates matrices cluster_info. to function get_vectors.
n_fold	Optional. A positive number giving the number of folds used for cross validation. This parameter will pass to cv.glmnet to calculate a penalty term for every gene.

Details

This function will take the converted gene and cluster vectors from function get_vectors, and return the most relevant cluster label for each gene. If there are multiple samples in the dataset, this function will find shared markers across different samples by including additional sample vectors in the input cluster_mt.

This function treats all input cluster vectors as features, and create a penalized linear model for one gene vector with lasso regularization. Clusters with non-zero coefficient will be selected, and these clusters will be used to formulate a generalised linear model for this gene vector.

- If the input keep_positive is TRUE, the clusters with positive coefficient and significant p-value will be saved in the output matrix lasso_full_result. The cluster with a positive coefficient and the minimum p-value will be regarded as the most relevant cluster to this gene and be saved in the output matrix lasso_result.
- If the input keep_positive is FALSE, the clusters with negative coefficient and significant p-value will be saved in the output matrix lasso_full_result. The cluster with a negative coefficient and the minimum p-value will be regarded as the most relevant cluster to this gene and be saved in the output matrix lasso_result.

If there is no clusters with significant p-value, the a string "NoSig" will be returned for this gene.

The parameter background can be used to capture unwanted noise pattern in the dataset. For example, we can include negative control genes as a background cluster in the model. If the most relevant cluster selected by one gene matches the background "clusters", we will return "NoSig" for this gene.

Value

An object of class 'glm_mg_result' To access specific components of the returned object:

- Use get_top_mg to retrieve the top result data frame
- Use get_full_mg to retrieve full result data frame

See Also

get_vectors

Examples

```
library(SpatialExperiment)
set.seed(100)
#  simulate coordinates for clusters
df_clA <- data.frame(x = rnorm(n=100, mean=20, sd=5),
                y = rnorm(n=100, mean=20, sd=5), cluster="A")
df_clB <- data.frame(x = rnorm(n=100, mean=100, sd=5),
               y = rnorm(n=100, mean=100, sd=5), cluster="B")

clusters <- rbind(df_clA, df_clB)
clusters$sample<-"sample1"

# simulate coordinates for genes
trans_info <- data.frame(rbind(cbind(x = rnorm(n=100, mean=20,sd=5),
                                y = rnorm(n=100, mean=20, sd=5),
                                 feature_name="gene_A1"),
                           cbind(x = rnorm(n=100, mean=20, sd=5),
                                 y = rnorm(n=100, mean=20, sd=5),
                                 feature_name="gene_A2"),
                           cbind(x = rnorm(n=100, mean=100, sd=5),
                                 y = rnorm(n=100, mean=100, sd=5),
                                 feature_name="gene_B1"),
                           cbind(x = rnorm(n=100, mean=100, sd=5),
                                 y = rnorm(n=100, mean=100, sd=5),
                                 feature_name="gene_B2")))
trans_info$x<-as.numeric(trans_info$x)
trans_info$y<-as.numeric(trans_info$y)
trans_info$cell<-sample(c("cell1","cell2","cell2"),replace=TRUE,
                        size=nrow(trans_info))
trans_mol <- BumpyMatrix::splitAsBumpyMatrix(
    trans_info[, c("x", "y")],
    row = trans_info$feature_name, col = trans_info$cell )
spe<- SpatialExperiment(
     assays = list(molecules = trans_mol),sample_id ="sample1" )
w_x <- c(min(floor(min(trans_info$x)),
         floor(min(clusters$x))),
      max(ceiling(max(trans_info$x)),
          ceiling(max(clusters$x))))
w_y <- c(min(floor(min(trans_info$y)),
          floor(min(clusters$y))),
      max(ceiling(max(trans_info$y)),
```

```
          ceiling(max(clusters$y))))
vecs_lst <- get_vectors(x=spe,sample_names=c("sample1"),
                    cluster_info = clusters,
                    bin_type = "square",
                    bin_param = c(20,20),
                    test_genes =c("gene_A1","gene_A2","gene_B1","gene_B2"),
                    w_x = w_x, w_y=w_y)
lasso_res <- lasso_markers(gene_mt=vecs_lst$gene_mt,
                        cluster_mt = vecs_lst$cluster_mt,
                        sample_names=c("sample1"),
                        keep_positive=TRUE,
                        background=NULL)
# the top result
top_result <- get_top_mg(lasso_res, coef_cutoff=0.05)
# the full result
full_result <- get_full_mg(lasso_res, coef_cutoff=0.05)
```

`rep1_clusters` *Rep1 selected cells*

Description

A data frame file containing the coordinates and cluster label for each cell of the selected subset of rep1.

Usage

```
data(rep1_clusters)
```

Format

A data frame with 1705 rows and 6 variables:

anno the provided cell type annotation

cluster cluster label

x x coordinates

y y coordiantes

cells cell id

sample sample id

Value

data frame

Source

<https://cf.10xgenomics.com/samples/xenium/1.0.1/ Xenium_FFPE_Human_Breast_Cancer_Rep1/ Xenium_FFPE_Human_Breast_Cancer_Rep1_outs.zip>

rep1_neg *Rep1 negative control genes within the selected region.*

Description

A SpatialExperiment object containing the coordinates for every negative control detection for rep1_sub

Usage

```
data(rep1_neg)
```

Format

A SpatialExperiment object. The molecules assay slot is a BumpyDataFrameMatrix obejct. Can retrieve DataFrame version by calling 'BumpyMatrix::unsplitAsDataFrame(molecules(rep1_neg))'. The molecules slot contains:

x x coordinates

y y coordiantes

feature_name negative control probe name

category negative control category

Value

SpatialExperiment

Source

<https://cf.10xgenomics.com/samples/xenium/1.0.1/ Xenium_FFPE_Human_Breast_Cancer_Rep1/Xenium_FFPE_Human_Cancer_Rep1_outs.zip>

rep1_sub *A small section of Xenium human breast cancer rep1.*

Description

A SpatialExperiment object containing the coordinates for every transcript

Usage

```
data(rep1_sub)
```

Format

A SpatialExperiment object with 20 genes and 1713 cells. The molecules assay slot is a BumpyDataFrameMatrix obejct. Can retrieve DataFrame version by calling 'BumpyMatrix::unsplitAsDataFrame(molecules(rep2_s
The molecules assay contains 79576 rows and 3 variables:

x x coordinates

y y coordiantes

feature_name transcript name

Value

SpatialExperiment

Source

<https://cf.10xgenomics.com/samples/xenium/1.0.1/ Xenium_FFPE_Human_Breast_Cancer_Rep1/ Xenium_FFPE_Human_Breast_Cancer_Rep1_outs.zip>

`rep2_clusters` *Rep2 selected cells*

Description

A csv file containing the coordinates and cluster label for each cell of the selected subset of rep2.

Usage

```
data(rep2_clusters)
```

Format

A data frame with 1815 rows and and 6 variables:

anno the provided cell type annotation

cluster cluster label

x x coordinates

y y coordiantes

cells cell id

sample sample id

Value

data frame

Source

<https://cf.10xgenomics.com/samples/xenium/1.0.1/ Xenium_FFPE_Human_Breast_Cancer_Rep2/ Xenium_FFPE_Human_Breast_Cancer_Rep2_outs.zip>

rep2_neg *Rep2 negative control genes within the selected region.*

Description

A data frame containing the coordinates for every negative control detection for rep2

Usage

```
data(rep2_neg)
```

Format

A SpatialExperiment object. The molecules assay slot is a BumpyDataFrameMatrix obejct. Can retrieve DataFrame version by calling 'BumpyMatrix::unsplitAsDataFrame(molecules(rep2_neg))'. The molecules slot contains:

x x coordinates

y y coordiantes

feature_name negative control probe name .

category negative control category

Value

SpatialExperiment

Source

<https://cf.10xgenomics.com/samples/xenium/1.0.1/ Xenium_FFPE_Human_Breast_Cancer_Rep2/ Xenium_FFPE_Human_Breast_Cancer_Rep2_outs.zip>

rep2_sub *A small section of Xenium human breast cancer rep2.*

Description

A SpatialExperiment object containing the coordinates for every negative control detection for rep2_sub

Usage

```
data(rep2_sub)
```

Format

A SpatialExperiment object with 20 genes and 1829 cells. The molecules assay slot is a BumpyDataFrameMatrix obejct. Can retrieve DataFrame version by calling 'BumpyMatrix::unsplitAsDataFrame(molecules(rep2_su
The molecules slot contains:

x x coordinates

y y coordiantes

feature_name transcript name

Value

SpatialExperiment

Source

<https://cf.10xgenomics.com/samples/xenium/1.0.1/ Xenium_FFPE_Human_Breast_Cancer_Rep2/ Xenium_FFPE_Human_Breast_Cancer_Rep2_outs.zip>

Index