Package 'SignacX'

January 20, 2025

Version 2.2.5

Date 2021-11-17

Title Cell Type Identification and Discovery from Single Cell Gene Expression Data

Maintainer Mathew Chamberlain <chamberlainphd@gmail.com>

URL https://github.com/mathewchamberlain/SignacX

BugReports https://github.com/mathewchamberlain/SignacX/issues

Imports neuralnet, lme4, methods, Matrix, pbmcapply, Seurat(>= 3.2.0), RJSONIO, igraph (>= 1.2.1), jsonlite (>= 1.5), RColorBrewer (>= 1.1.2), stats

Suggests hdf5r, rhdf5, knitr, rmarkdown, formatR

Description An implementation of neural networks trained with flow-sorted gene expression data to classify cellular phenotypes in single cell RNA-sequencing data. See Chamberlain M et al. (2021) <doi:10.1101/2021.02.01.429207> for more details.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Mathew Chamberlain [aut, cre],
Virginia Savova [aut],
Richa Hanamsagar [aut],
Frank Nestle [aut],
Emanuele de Rinaldis [aut],
Sanofi US [fnd]

Repository CRAN

Date/Publication 2021-11-18 16:20:03 UTC

Contents

`CID.entropy` *Normalized Shannon entropy-based "unclassified" assignment*

Description

`CID.entropy` calculates the normalized Shannon entropy of labels for each cell among k-nearest neighbors less than four-degrees apart, and then sets cells with statistically significant large Shannon entropy to be "Unclassified."

Usage

```
CID.entropy(ac, distM)
```

Arguments

`ac`	a character vector of cell type labels
`distM`	the distance matrix, see ?CID.GetDistMat

Value

A character vector like 'ac' but with cells type labels set to "Unclassified" if there was high normalized Shannon entropy.

Examples

```
## Not run:
# load data classified previously (see \code{SignacFast})
P <- readRDS("celltypes.rds")
S <- readRDS("pbmcs.rds")

# get edges from default assay from Seurat object
default.assay <- Seurat::DefaultAssay(S)
edges = S@graphs[[which(grepl(paste0(default.assay, "_nn"), names(S@graphs)))]]

# get distance matrix
D = CID.GetDistMat(edges)

# entropy-based unclassified labels labels
entropy = CID.entropy(ac = P$L2, distM = D)

## End(Not run)
```

`CID.GetDistMat` *Computes distance matrix from edge list*

Description

`CID.GetDistMat` returns the distance matrix (i.e., adjacency matrix, second-degree adjacency matrix, ..., etc.) from an edge list. Here, edges is the edge list, and n is the order of the connetions.

Usage

```
CID.GetDistMat(edges, n = 4)
```

Arguments

`edges`	a data frame with two columns; V1 and V2, specifying the cell-cell edge list for the network
`n`	maximum network distance to subtend (n neighbors)

Value

adjacency matrices for distances < n

Examples

```
## Not run:
# Loads edges
file.dir = "https://kleintools.hms.harvard.edu/tools/client_datasets/"
file = "CITESEQ_EXPLORATORY_CITESEQ_5K_PBMCS/FullDataset_v1_protein/edges.csv"
download.file(paste0(file.dir, file, "?raw=true"), destfile = "edges.csv")

# data.dir is your path to the "edges.csv" file
```

```
edges = CID.LoadEdges(data.dir = ".")

# get distance matrix (adjacency matrices with up to fourth-order connections) from edge list
distance_matrices = CID.GetDistMat(edges)

## End(Not run)
```

`CID.IsUnique` *Extracts unique elements*

Description

`CID.IsUnique` returns a Boolean for the unique elements of a vector.

Usage

```
CID.IsUnique(x)
```

Arguments

`x`	A character vector

Value

boolean, unique elements are TRUE

Examples

```
# generate a dummy variable
dummy = c("A", "A", "B", "C")

# get only unique elements
logik = CID.IsUnique(dummy)
dummy[logik]
```

`CID.LoadData` *Load data file from directory*

Description

Loads 'matrix.mtx' and 'genes.txt' files from a directory.

Usage

```
CID.LoadData(data.dir, mfn = "matrix.mtx")
```

Arguments

data.dir	Directory containing matrix.mtx and genes.txt.
mfn	file name; default is 'matrix.mtx'

Value

A sparse matrix with rownames equivalent to the names in genes.txt

Examples

```
## Not run:
# Loads data from SPRING

# dir is your path to the "categorical_coloring_data.json" file
dir = "./FullDataset_v1"

# load expression data
E = CID.LoadData(data.dir = dir)

## End(Not run)
```

CID.LoadEdges *Load edges from edge list for single cell network*

Description

CID.LoadEdges loads edges, typically after running the SPRING pipeline.

Usage

```
CID.LoadEdges(data.dir)
```

Arguments

data.dir	A directory where "edges.csv" file is located

Value

The edge list in data frame format

Examples

```
## Not run:
# Loads edges
file.dir = "https://kleintools.hms.harvard.edu/tools/client_datasets/"
file = "CITESEQ_EXPLORATORY_CITESEQ_5K_PBMCS/FullDataset_v1_protein/edges.csv"
download.file(paste0(file.dir, file, "?raw=true"), destfile = "edges.csv")

# data.dir is your path to the "edges.csv" file
```

```
edges = CID.LoadEdges(data.dir = ".")

## End(Not run)
```

CID.Louvain *Detects community substructure by Louvain community detection*

Description

CID.Louvain determines Louvain clusters from an edge list.

Usage

```
CID.Louvain(edges)
```

Arguments

edges	A data frame or matrix with edges

Value

A character vector with the community substructures of the graph corresponding to Louvain clusters.

Examples

```
## Not run:
# Loads edges
file.dir = "https://kleintools.hms.harvard.edu/tools/client_datasets/"
file = "CITESEQ_EXPLORATORY_CITESEQ_5K_PBMCS/FullDataset_v1_protein/edges.csv"
download.file(paste0(file.dir, file, "?raw=true"), destfile = "edges.csv")

# data.dir is your path to the "edges.csv" file
edges = CID.LoadEdges(data.dir = ".")

# get louvain clusters from edge list
clusters = CID.Louvain(edges)

## End(Not run)
```

CID.Normalize *Library size normalize*

Description

CID.Normalize normalizes the expression matrix to the mean library size.

Usage

```
CID.Normalize(E)
```

Arguments

E	Expression matrix

Value

Normalized expression matrix where each cell sums to the mean total counts

Examples

```
## Not run:
# download single cell data for classification
file.dir = "https://cf.10xgenomics.com/samples/cell-exp/3.0.0/pbmc_1k_v3/"
file = "pbmc_1k_v3_filtered_feature_bc_matrix.h5"
download.file(paste0(file.dir, file), "Ex.h5")

# load data, process with Seurat
library(Seurat)
E = Read10X_h5(filename = "Ex.h5")

# observe average total counts
mean(Matrix::colSums(E))

# run normalization
E_norm = CID.Normalize(E)

# check normalization
kmu = mean(Matrix::colSums(E_norm))
head(kmu)

## End(Not run)
```

CID.smooth *Smoothing function*

Description

CID.smooth uses k-nearest neighbors to identify cells which correspond to a different label than the majority of their first-degree neighbors. If so, those annotations are "smoothed."

Usage

```
CID.smooth(ac, dM)
```

Arguments

ac	list containing a character vector where each element is a cell type or cell state assignment.
dM	distance matrix (see ?CID.GetDistMat).

Value

A character vector with smoothed labels

Examples

```
## Not run:
# load data classified previously (see SignacFast)
P <- readRDS("celltypes.rds")
S <- readRDS("pbmcs.rds")

# get edges from default assay from Seurat object
default.assay <- Seurat::DefaultAssay(S)
edges = S@graphs[[which(grepl(paste0(default.assay, "_nn"), names(S@graphs)))]]

# get distance matrix
D = CID.GetDistMat(edges)

# smooth labels
smoothed = CID.smooth(ac = P$CellTypes, dM = D[[1]])

## End(Not run)
```

CID.writeJSON *Writes JSON file for SPRING integration*

Description

CID.writeJSON is a SPRING-integrated function for writing a JSON file.

Usage

```
CID.writeJSON(
  cr,
  json_new = "categorical_coloring_data.json",
  spring.dir,
  new_populations = NULL,
  new_colors = NULL
)
```

Arguments

cr	output from GenerateLabels
json_new	Filename where annotations will be saved for new SPRING color tracks. Default is "categorical_coloring_data_new.json".
spring.dir	Directory where file 'categorical_coloring_data.json' is located. If set, will add Signac annotations to the file.
new_populations	Character vector specifying any new cell types that Signac has learned. Default is NULL.
new_colors	Character vector specifying the HEX color codes for new cell types. Default is NULL.

Value

A categorical_coloring_data.json file with Signac annotations and Louvain clusters added.

GenerateLabels *Generates cellular phenotype labels*

Description

GenerateLabels returns a list of cell type and cell state labels, as well as novel cellular phenotypes and unclassified cells.

Usage

```
GenerateLabels(
  cr,
  E = NULL,
  smooth = TRUE,
  new_populations = NULL,
  new_categories = NULL,
  min.cells = 10,
  spring.dir = NULL,
  graph.used = "nn"
)
```

Arguments

cr	list returned by Signac or by SignacFast.
E	a sparse gene (rows) by cell (column) matrix, or a Seurat object. Rows are HUGO symbols.
smooth	if TRUE, smooths the cell type classifications. Default is TRUE.
new_populations	Character vector specifying any new cell types that were learned by Signac. Default is NULL.
new_categories	If new_populations are set to a cell type, new_category is a corresponding character vector indicating the population that the new population belongs to. Default is NULL.
min.cells	If desired, any cell population with equal to or less than N cells is set to "Unclassified." Default is 10 cells.
spring.dir	If using SPRING, directory to categorical_coloring_data.json. Default is NULL.
graph.used	If using Seurat object by default, Signac uses the nearest neighbor graph in the graphs field of the Seurat object. Other options are "wnn" to use weighted nearest neighbors, as well as "snn" to use shared nearest neighbors.

Value

A list of cell type labels for cell types, cell states and novel populations.

Examples

```
## Not run:
# download single cell data for classification
file.dir = "https://cf.10xgenomics.com/samples/cell-exp/3.0.0/pbmc_1k_v3/"
file = "pbmc_1k_v3_filtered_feature_bc_matrix.h5"
download.file(paste0(file.dir, file), "Ex.h5")

# load data, process with Seurat
library(Seurat)
E = Read10X_h5(filename = "Ex.h5")
pbmc <- CreateSeuratObject(counts = E, project = "pbmc")
```

```
# run Seurat pipeline
pbmc <- SCTransform(pbmc, verbose = FALSE)
pbmc <- RunPCA(pbmc, verbose = FALSE)
pbmc <- FindNeighbors(pbmc, dims = 1:30, verbose = FALSE)

# classify cells
labels = SignacFast(E = pbmc)
celltypes = GenerateLabels(labels, E = pbmc)

## End(Not run)
```

`Genes_Of_Interest` *Genes of interest for drug discovery / disease biology research*

Description

3,304 genes curated from external sources

Usage

```
Genes_Of_Interest
```

Format

A data frame with five columns

Genes genes, identified by HUGO symbols

CellPhoneDB 0 if not in CellPhoneDB, 1 if gene is listed as a receptor in CellPhoneDB

GWAS 0 if not GWAS, 1 if GWAS in GWAS catalog

PI_drugs 0 if not in Priority Index paper, 1 if gene is listed as a drug target in priority index paper

PI_GWAS 0 if not in Priority Index GWAS list, 1 if gene is listed as GWAS in priority index paper

...

`GetModels_HPCA` *Loads neural network models from GitHub*

Description

`GetModels_HPCA` returns a list of neural network models that were trained with the HPCA training data. The HPCA training data has 18 classification tasks with bootstrapped training data. The training data are split into two distinct cellular phenotypes (i.e., immune and nonimmune). `GetModels_HPCA` downloads pre-computed neural networks trained to classify cells for each task (n = 100 models trained for each tasks).

Usage

```
GetModels_HPCA()
```

Value

list of 1,800 neural network models stacked to solve 18 classification problems.

See Also

SignacFast, ModelGenerator

Examples

```
## Not run:
P = GetModels()

## End(Not run)
```

GetTrainingData_HPCA *Loads bootstrapped HPCA training data from GitHub*

Description

GetTrainingData_HPCA returns a list of bootstrapped HPCA training data. The HPCA training data has 18 classification tasks, each split into two distinct cellular phenotypes (i.e., immune and nonimmune). GetTrainingData_HPCA downloads bootstrapped training data to use for model-building.

Usage

```
GetTrainingData_HPCA()
```

Value

A list with two elements; 'Reference' are the training data, and 'genes' – the union of all features present in the training data.

Examples

```
## Not run:
P = GetTrainingData_HPCA()

## End(Not run)
```

KSoftImpute *KNN-based imputation*

Description

KSoftImpute is an ultra-fast method for imputing missing gene expression values in single cell data. KSoftImpute uses k-nearest neighbors to impute the expression of each gene by the weighted average of itself and it's first-degree neighbors. Weights for imputation are determined by the number of detected genes. This method works for large data sets (>100,000 cells) in under a minute.

Usage

```
KSoftImpute(E, dM = NULL, genes.to.use = NULL, verbose = FALSE)
```

Arguments

E	A gene-by-sample count matrix (sparse matrix or matrix) with genes identified by their HUGO symbols.
dM	see ?CID.GetDistMat
genes.to.use	a character vector of genes to impute. Default is NULL.
verbose	If TRUE, code reports outputs. Default is FALSE.

Value

An expression matrix (sparse matrix) with imputed values.

See Also

Signac and SignacFast

Examples

```
## Not run:
# download single cell data for classification
file.dir = "https://cf.10xgenomics.com/samples/cell-exp/3.0.0/pbmc_1k_v3/"
file = "pbmc_1k_v3_filtered_feature_bc_matrix.h5"
download.file(paste0(file.dir, file), "Ex.h5")

# load data, process with Seurat
library(Seurat)
E = Read10X_h5(filename = "Ex.h5")
pbmc <- CreateSeuratObject(counts = E, project = "pbmc")

# run Seurat pipeline
pbmc <- SCTransform(pbmc, verbose = FALSE)
pbmc <- RunPCA(pbmc, verbose = FALSE)
pbmc <- RunUMAP(pbmc, dims = 1:30, verbose = FALSE)
pbmc <- FindNeighbors(pbmc, dims = 1:30, verbose = FALSE)
```

```
# get edges from default assay from Seurat object
default.assay <- Seurat::DefaultAssay(pbmc)
edges = pbmc@graphs[[which(grepl(paste0(default.assay, "_nn"), names(pbmc@graphs)))]]

# get distance matrix
dM = CID.GetDistMat(edges)

# run imputation
Z = KSoftImpute(E = E, dM = dM, verbose = TRUE)

## End(Not run)
```

MASC *Mixed effect modeling*

Description

MASC was imported from https://github.com/immunogenomics/masc. Performs mixed-effect modeling.

Usage

```
MASC(
  dataset,
  cluster,
  contrast,
  random_effects = NULL,
  fixed_effects = NULL,
  verbose = FALSE
)
```

Arguments

dataset	data frame of covariate, cell type, clustering or disease information
cluster	celltypes returned by Signac or cluster identities
contrast	Typically disease
random_effects	User specified random effect variables in dataset
fixed_effects	User specific fixed effects in dataset
verbose	If TRUE, algorithm reports outputs

Value

mixed effect model results

Examples

```
## Not run:
# Load metadata
file.dir = "https://kleintools.hms.harvard.edu/tools/client_datasets/"
file = "AMP_Phase1_SLE_Apr2019/FullDataset_v1/categorical_coloring_data.json"
download.file(paste0(file.dir, file, "?raw=true"), destfile = "categorical_coloring_data.json")
d = rjson::fromJSON(file='categorical_coloring_data.json')
d = data.frame(sapply(d, function(x) x$label_list))

# run MASC
x = d$CellStates # optionally use clusters or cell types
d$Disease = factor(d$Disease) # the contrast term must be encoded as a factor
Q = MASC(d, cluster = x, contrast = 'Disease', random_effects = c( "Tissue", "Plate", "Sample"))

## End(Not run)
```

`ModelGenerator` *Generates an ensemble of neural network models.*

Description

`ModelGenerator` generates an ensemble of neural network models each trained to classify cellular phenotypes using the reference data set.

Usage

```
ModelGenerator(
  R,
  N = 1,
  num.cores = 1,
  verbose = TRUE,
  hidden = 1,
  set.seed = TRUE,
  seed = "42"
)
```

Arguments

`R`	Reference data set returned by `GetTrainingData_HPCA`
`N`	Number of neural networks to train. Default is 1.
`num.cores`	Number of cores to use for parallel computing. Default is 1.
`verbose`	if TRUE, code will report outputs. Default is TRUE.
`hidden`	Number of hidden layers in the neural network. Default is 1.
`set.seed`	If TRUE, seed is set to ensure reproducibility of these results. Default is TRUE.
`seed`	if set.seed is TRUE, the seed can be set. Default is 42.

Value

A list, each containing N neural network models

See Also

[SignacFast()] for a function that uses the models generated by this function.

Examples

```
## Not run:
# download training data set from GitHub
Ref = GetTrainingData_HPCA()

# train a stack of 1,800 neural network models
Models = ModelGenerator(R = Ref, N = 100, num.cores = 4)

# save models
save(Models, file = "models.rda")

## End(Not run)
```

SaveCountsToH5 *Save count_matrix.h5 files for SPRING integration*

Description

To integrate with SPRING, SaveCountsToH5 saves expression matrices in a sparse ".h5" format to be read with SPRING notebooks in Jupyter.

Usage

```
SaveCountsToH5(D, data.dir, genome = "GRCh38")
```

Arguments

D	A list of count matrices
data.dir	directory (will be created if it does not exist) where results are saved
genome	default is GRCh38.

Value

matrix.h5 file, where each is background corrected

Examples

```
## Not run:
# download single cell data for classification
file.dir = "https://cf.10xgenomics.com/samples/cell-exp/3.0.0/pbmc_1k_v3/"
file = "pbmc_1k_v3_filtered_feature_bc_matrix.h5"
download.file(paste0(file.dir, file), "Ex.h5")

# load data
library(Seurat)
E = Read10X_h5(filename = "Ex.h5")

# save counts to h5 files
SaveCountsToH5(E, data.dir = "counts_h5")


## End(Not run)
```

Signac *Classification of cellular phenotypes in single cell data*

Description

Signac trains and then uses an ensemble of neural networks to classify cellular phenotypes using an expression matrix or Seurat object. The neural networks are trained with the HPCA training data using only features that are present in both the single cell and HPCA training data set. Signac returns annotations at each level of the classification hierarchy, which are then converted into cell type labels using GenerateLabels. For a faster alternative, try SignacFast, which uses pre-computed neural network models.

Usage

```
Signac(
  E,
  R = "default",
  spring.dir = NULL,
  N = 100,
  num.cores = 1,
  threshold = 0,
  smooth = TRUE,
  impute = TRUE,
  verbose = TRUE,
  do.normalize = TRUE,
  return.probability = FALSE,
  hidden = 1,
  set.seed = TRUE,
  seed = "42",
  graph.used = "nn"
)
```

Arguments

E	a sparse gene (rows) by cell (column) matrix, or a Seurat object. Rows are HUGO symbols.
R	Reference data. If 'default', R is set to GetTrainingData_HPCA().
spring.dir	If using SPRING, directory to categorical_coloring_data.json. Default is NULL.
N	Number of machine learning models to train (for nn and svm). Default is 100.
num.cores	Number of cores to use. Default is 1.
threshold	Probability threshold for assigning cells to "Unclassified." Default is 0.
smooth	if TRUE, smooths the cell type classifications. Default is TRUE.
impute	if TRUE, gene expression values are imputed prior to cell type classification (see KSoftImpute). Default is TRUE.
verbose	if TRUE, code will report outputs. Default is TRUE.
do.normalize	if TRUE, cells are normalized to the mean library size. Default is TRUE.
return.probability	if TRUE, returns the probability associated with each cell type label. Default is TRUE.
hidden	Number of hidden layers in the neural network. Default is 1.
set.seed	If true, seed is set to ensure reproducibility of these results. Default is TRUE.
seed	if set.seed is TRUE, seed is set to 42.
graph.used	If using Seurat object by default, Signac uses the nearest neighbor graph in the graphs field of the Seurat object. Other options are "wnn" to use weighted nearest neighbors, as well as "snn" to use shared nearest neighbors.

Value

A list of character vectors: cell type annotations (L1, L2, ...) at each level of the hierarchy as well as 'clusters' for the Louvain clustering results.

See Also

SignacFast, a faster alternative that only differs from Signac in nuanced T cell phenotypes.

Examples

```
## Not run:
# download single cell data for classification
file.dir = "https://cf.10xgenomics.com/samples/cell-exp/3.0.0/pbmc_1k_v3/"
file = "pbmc_1k_v3_filtered_feature_bc_matrix.h5"
download.file(paste0(file.dir, file), "Ex.h5")

# load data, process with Seurat
library(Seurat)
E = Read10X_h5(filename = "Ex.h5")
pbmc <- CreateSeuratObject(counts = E, project = "pbmc")
```

```
# run Seurat pipeline
pbmc <- SCTransform(pbmc, verbose = FALSE)
pbmc <- RunPCA(pbmc, verbose = FALSE)
pbmc <- RunUMAP(pbmc, dims = 1:30, verbose = FALSE)
pbmc <- FindNeighbors(pbmc, dims = 1:30, verbose = FALSE)

# classify cells
labels = Signac(E = pbmc)
celltypes = GenerateLabels(labels, E = pbmc)

# add labels to Seurat object, visualize
pbmc <- Seurat::AddMetaData(pbmc, metadata=celltypes$CellTypes_novel, col.name = "immmune")
pbmc <- Seurat::SetIdent(pbmc, value='immmune')
DimPlot(pbmc)

# save results
saveRDS(pbmc, "example_pbmcs.rds")

## End(Not run)
```

SignacBoot *Generates bootstrapped single cell data*

Description

SignacBoot uses a Seurat object or an expression matrix and performs feature selection, normalization and bootstrapping to generate a training data set to be used for cell type or cluster classification.

Usage

```
SignacBoot(
  E,
  L,
  labels,
  size = 1000,
  impute = TRUE,
  spring.dir = NULL,
  logfc.threshold = 0.25,
  p.val.adj = 0.05,
  verbose = TRUE
)
```

Arguments

E	a gene (rows) by cell (column) matrix, sparse matrix or a Seurat object. Rows are HUGO symbols.
L	cell type categories for learning.
labels	cell type labels corresponding to the columns of E.

size	Number of bootstrapped samples for machine learning. Default is 1,000.
impute	if TRUE, performs imputation prior to bootstrapping (see KSoftImpute). Default is TRUE.
spring.dir	if using SPRING, directory to categorical_coloring_data.json. Default is NULL.
logfc.threshold	Cutoff for feature selection. Default is 0.25.
p.val.adj	Cutoff for feature selection. Default is 0.05.
verbose	if TRUE, code speaks. Default is TRUE.

Value

Training data set (data.frame) to be used for building new models=.

See Also

ModelGenerator

Examples

```
## Not run:
# load Seurat object from SignacFast example
P <- readRDS("pbmcs.rds")

# run feature selection + bootstrapping to generate 2,000 bootstrapped cells
x = P@meta.data$celltypes
R_learned = SignacBoot(P, L = c("B.naive", "B.memory"), labels = x)

## End(Not run)
```

SignacFast *Fast classification of cellular phenotypes*

Description

SignacFast uses pre-computed neural network models to classify cellular phenotypes in single cell data: these models were pre-trained with the HPCA training data. Any features that are present in the training data and absent in the single cell data are set to zero. This is a factor of ~5-10 speed improvement over Signac.

Usage

```
SignacFast(
  E,
  Models = "default",
  spring.dir = NULL,
  num.cores = 1,
  threshold = 0,
```

```
  smooth = TRUE,
  impute = TRUE,
  verbose = TRUE,
  do.normalize = TRUE,
  return.probability = FALSE,
  graph.used = "nn"
)
```

Arguments

E	a gene (rows) by cell (column) matrix, sparse matrix or a Seurat object. Rows are HUGO symbols.
Models	if 'default', as returned by GetModels_HPCA. An ensemble of 1,800 neural network models.
spring.dir	If using SPRING, directory to categorical_coloring_data.json. Default is NULL.
num.cores	number of cores to use for parallel computation. Default is 1.
threshold	Probability threshold for assigning cells to "Unclassified." Default is 0.
smooth	if TRUE, smooths the cell type classifications. Default is TRUE.
impute	if TRUE, gene expression values are imputed prior to cell type classification (see KSoftImpute). Default is TRUE.
verbose	if TRUE, code will report outputs. Default is TRUE.
do.normalize	if TRUE, cells are normalized to the mean library size. Default is TRUE.
return.probability	if TRUE, returns the probability associated with each cell type label. Default is TRUE.
graph.used	If using Seurat object by default, Signac uses the nearest neighbor graph in the graphs field of the Seurat object. Other options are "wnn" to use weighted nearest neighbors, as well as "snn" to use shared nearest neighbors.

Value

A list of character vectors: cell type annotations (L1, L2, ...) at each level of the hierarchy as well as 'clusters' for the Louvain clustering results.

See Also

Signac for another classification function.

Signac

Examples

```
## Not run:
# download single cell data for classification
file.dir = "https://cf.10xgenomics.com/samples/cell-exp/3.0.0/pbmc_1k_v3/"
file = "pbmc_1k_v3_filtered_feature_bc_matrix.h5"
download.file(paste0(file.dir, file), "Ex.h5")
```

```
# load data, process with Seurat
library(Seurat)
E = Read10X_h5(filename = "Ex.h5")
pbmc <- CreateSeuratObject(counts = E, project = "pbmc")
pbmc <- SCTransform(pbmc)
pbmc <- RunPCA(pbmc, verbose = FALSE)
pbmc <- RunUMAP(pbmc, dims = 1:30, verbose = FALSE)
pbmc <- FindNeighbors(pbmc, dims = 1:30, verbose = FALSE)

# classify cells
labels = SignacFast(E = pbmc)
celltypes = GenerateLabels(labels, E = pbmc)

# add labels to Seurat object, visualize
lbls <- factor(celltypes$CellStates)
levels(lbls) <- sort(unique(lbls))
pbmc <- AddMetaData(pbmc, metadata=celltypes$CellStates, col.name = "celltypes")
pbmc <- SetIdent(pbmc, value='celltypes')
DimPlot(pbmc, label = T)

# save results
saveRDS(pbmc, "pbmcs.rds")
saveRDS(celltypes, "celltypes.rds")

## End(Not run)
```

Index