Package ‘diceR’

January 22, 2024

Type Package

Title Diverse Cluster Ensemble in R

Version 2.2.0

Description Performs cluster analysis using an ensemble clustering framework, Chiu & Talhouk (2018) <doi:10.1186/s12859-017-1996-y>. Results from a diverse set of algorithms are pooled together using methods such as majority voting, K-Modes, LinkCluE, and CSPA. There are options to compare cluster assignments across algorithms using internal and external indices, visualizations such as heatmaps, and significance testing for the existence of clusters.

License MIT + file LICENSE

URL `https://github.com/AlineTalhouk/diceR/`, `https://alinetalhouk.github.io/diceR/`

BugReports `https://github.com/AlineTalhouk/diceR/issues`

Depends R (>= 3.5)

Imports abind, assertthat, class, clue, clusterSim, clv, clValid, dplyr (>= 0.7.5), ggplot2, infotheo, klaR, magrittr, mclust, methods, NMF, purrr (>= 0.2.3), RankAggreg, Rcpp, stringr, tidyr, yardstick

Suggests apcluster, blockcluster, cluster, covr, dbscan, e1071, kernlab, knitr, kohonen, pander, poLCA, progress, RColorBrewer, rlang, rmarkdown, Rtsne, sigclust, testthat

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.0

NeedsCompilation yes

Author Derek Chiu [aut, cre],
Aline Talhouk [aut],
Johnson Liu [ctb, com]

Maintainer Derek Chiu <dchiu@bccrc.ca>

Repository CRAN

Date/Publication 2024-01-22 21:22:46 UTC

R topics documented:

`compactness` *Compactness Measure*

Description

Compute the compactness validity index for a clustering result.

Usage

```
compactness(data, labels)
```

Arguments

`data`	a dataset with rows as observations, columns as variables
`labels`	a vector of cluster labels from a clustering result

Details

This index is agnostic to any reference clustering results, calculating cluster performance on the basis of compactness and separability. Smaller values indicate a better clustering structure.

Value

the compactness score

Author(s)

Derek Chiu

References

MATLAB function `valid_compactness` by Simon Garrett in LinkCluE

Examples

```
set.seed(1)
E <- matrix(rep(sample(1:4, 1000, replace = TRUE)), nrow = 100, byrow =
              FALSE)
set.seed(1)
dat <- as.data.frame(matrix(runif(1000, -10, 10), nrow = 100, byrow = FALSE))
compactness(dat, E[, 1])
```

`consensus_cluster` *Consensus clustering*

Description

Runs consensus clustering across subsamples of the data, clustering algorithms, and cluster sizes.

Usage

```
consensus_cluster(
  data,
  nk = 2:4,
  p.item = 0.8,
  reps = 1000,
  algorithms = NULL,
  nmf.method = c("brunet", "lee"),
  hc.method = "average",
  xdim = NULL,
  ydim = NULL,
  rlen = 200,
  alpha = c(0.05, 0.01),
  minPts = 5,
  distance = "euclidean",
  abs = TRUE,
  prep.data = c("none", "full", "sampled"),
  scale = TRUE,
  type = c("conventional", "robust", "tsne"),
  min.var = 1,
  progress = TRUE,
  seed.nmf = 123456,
  seed.data = 1,
  file.name = NULL,
  time.saved = FALSE
)
```

Arguments

`data` data matrix with rows as samples and columns as variables

`nk` number of clusters (k) requested; can specify a single integer or a range of integers to compute multiple k

`p.item` proportion of items to be used in subsampling within an algorithm

`reps` number of subsamples

`algorithms` vector of clustering algorithms for performing consensus clustering. Must be any number of the following: "nmf", "hc", "diana", "km", "pam", "ap", "sc", "gmm", "block", "som", "cmeans", "hdbscan". A custom clustering algorithm can be used.

`nmf.method` specify NMF-based algorithms to run. By default the "brunet" and "lee" algorithms are called. See `NMF::nmf()` for details.

`hc.method` agglomeration method for hierarchical clustering. The the "average" method is used by default. See`stats::hclust()` for details.

`xdim` x dimension of the SOM grid

`ydim` y dimension of the SOM grid

`rlen` the number of times the complete data set will be presented to the SOM network.

`alpha` SOM learning rate, a vector of two numbers indicating the amount of change. Default is to decline linearly from 0.05 to 0.01 over `rlen` updates. Not used for the batch algorithm.

`minPts` minimum size of clusters for HDBSCAN. Default is 5.

`distance` a vector of distance functions. Defaults to "euclidean". Other options are given in `stats::dist()`. A custom distance function can be used.

`abs` only used for `distance = c("spearman", "pearson")`. If `TRUE`, the absolute value is first applied to the distance before subtracting from 1, e.g., we use 1 - |SCD| instead of 1 - SCD for the spearman correlation distance.

`prep.data` Prepare the data on the "full" dataset, the "sampled" dataset, or "none" (default).

`scale` logical; should the data be centered and scaled?

`type` if we use "conventional" measures (default), then the mean and standard deviation are used for centering and scaling, respectively. If "robust" measures are specified, the median and median absolute deviation (MAD) are used. Alternatively, we can apply "tsne" for dimension reduction.

`min.var` minimum variability measure threshold used to filter the feature space for only highly variable features. Only features with a minimum variability measure across all samples greater than `min.var` will be used. If `type = "conventional"`, the standard deviation is the measure used, and if `type = "robust"`, the MAD is the measure used.

`progress` logical; should a progress bar be displayed?

`seed.nmf` random seed to use for NMF-based algorithms

`seed.data` seed to use to ensure each algorithm operates on the same set of subsamples

`file.name` if not `NULL`, the returned array will be saved at each iteration as well as at the end of the function call to an `rds` object with `file.name` as the file name.

`time.saved` logical; if `TRUE`, the date saved is appended to `file.name`. Only applicable when `file.name` is not `NULL`.

Details

See examples for how to use custom algorithms and distance functions. The default clustering algorithms provided are:

- "nmf": Nonnegative Matrix Factorization (using Kullback-Leibler Divergence or Euclidean distance; See Note for specifications.)
- "hc": Hierarchical Clustering
- "diana": DIvisive ANAlysis Clustering
- "km": K-Means Clustering
- "pam": Partition Around Medoids
- "ap": Affinity Propagation
- "sc": Spectral Clustering using Radial-Basis kernel function
- "gmm": Gaussian Mixture Model using Bayesian Information Criterion on EM algorithm
- "block": Biclustering using a latent block model
- "som": Self-Organizing Map (SOM) with Hierarchical Clustering
- "cmeans": Fuzzy C-Means Clustering
- "hdbscan": Hierarchical Density-based Spatial Clustering of Applications with Noise (HDBSCAN)

The progress bar increments on every unit of `reps`.

Value

An array of dimension `nrow(x)` by `reps` by `length(algorithms)` by `length(nk)`. Each cube of the array represents a different k. Each slice of a cube is a matrix showing consensus clustering results for algorithms. The matrices have a row for each sample, and a column for each subsample. Each entry represents a class membership.

When "hdbscan" is part of `algorithms`, we do not include its clustering array in the consensus result. Instead, we report two summary statistics as attributes: the proportion of outliers and the number of clusters.

Note

The nmf.method options are "brunet" (Kullback-Leibler Divergence) and "lee" (Euclidean distance). When "hdbscan" is chosen as an algorithm to use, its results are excluded from the rest of the consensus clusters. This is because there is no guarantee that the cluster assignment will have every sample clustered; more often than not there will be noise points or outliers. In addition, the number of distinct clusters may not even be equal to nk.

Author(s)

Derek Chiu, Aline Talhouk

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]

# Custom distance function
manh <- function(x) {
  stats::dist(x, method = "manhattan")
}

# Custom clustering algorithm
agnes <- function(d, k) {
  return(as.integer(stats::cutree(cluster::agnes(d, diss = TRUE), k)))
}

assign("agnes", agnes, 1)

cc <- consensus_cluster(dat, reps = 6, algorithms = c("pam", "agnes"),
distance = c("euclidean", "manh"), progress = FALSE)
str(cc)
```

---

consensus_combine *Combine algorithms*

Description

Combines results for multiple objects from consensus_cluster() and outputs either the consensus matrices or consensus classes for all algorithms.

Usage

```
consensus_combine(..., element = c("matrix", "class"))
```

Arguments

... any number of objects outputted from consensus_cluster()

element either "matrix" or "class" to extract the consensus matrix or consensus class, respectively.

Details

This function is useful for collecting summaries because the original results from `consensus_cluster` were combined to a single object. For example, setting `element = "class"` returns a matrix of consensus cluster assignments, which can be visualized as a consensus matrix heatmap.

Value

`consensus_combine` returns either a list of all consensus matrices or a data frame showing all the consensus classes

Author(s)

Derek Chiu

Examples

```
# Consensus clustering for multiple algorithms
set.seed(911)
x <- matrix(rnorm(500), ncol = 10)
CC1 <- consensus_cluster(x, nk = 3:4, reps = 10, algorithms = "ap",
progress = FALSE)
CC2 <- consensus_cluster(x, nk = 3:4, reps = 10, algorithms = "km",
progress = FALSE)

# Combine and return either matrices or classes
y1 <- consensus_combine(CC1, CC2, element = "matrix")
str(y1)
y2 <- consensus_combine(CC1, CC2, element = "class")
str(y2)
```

`consensus_evaluate` *Evaluate, trim, and reweigh algorithms*

Description

Evaluates algorithms on internal/external validation indices. Poor performing algorithms can be trimmed from the ensemble. The remaining algorithms can be given weights before use in consensus functions.

Usage

```
consensus_evaluate(
  data,
  ...,
  cons.cl = NULL,
  ref.cl = NULL,
  k.method = NULL,
  plot = FALSE,
  trim = FALSE,
  reweigh = FALSE,
  n = 5,
  lower = 0,
  upper = 1
)
```

Arguments

`data`	data matrix with rows as samples and columns as variables
`...`	any number of objects outputted from `consensus_cluster()`
`cons.cl`	matrix of cluster assignments from consensus functions such as `kmodes` and `majority_voting`
`ref.cl`	reference class
`k.method`	determines the method to choose k when no reference class is given. When `ref.cl` is not `NULL`, k is the number of distinct classes of `ref.cl`. Otherwise the input from `k.method` chooses k. The default is to use the PAC to choose the best k(s). Specifying an integer as a user-desired k will override the best k chosen by PAC. Finally, specifying "all" will produce consensus results for all k. The "all" method is implicitly performed when there is only one k used.
`plot`	logical; if `TRUE`, `graph_all` is called
`trim`	logical; if `TRUE`, algorithms that score low on internal indices will be trimmed out
`reweigh`	logical; if `TRUE`, after trimming out poor performing algorithms, each algorithm is reweighed depending on its internal indices.
`n`	an integer specifying the top `n` algorithms to keep after trimming off the poor performing ones using Rank Aggregation. If the total number of algorithms is less than `n` no trimming is done.
`lower`	the lower bound that determines what is ambiguous
`upper`	the upper bound that determines what is ambiguous

Details

This function always returns internal indices. If `ref.cl` is not `NULL`, external indices are additionally shown. Relevant graphical displays are also outputted. Algorithms are ranked across internal indices using Rank Aggregation. Only the top `n` algorithms are kept, the rest are trimmed.

Value

`consensus_evaluate` returns a list with the following elements

- `k`: if `ref.cl` is not `NULL`, this is the number of distinct classes in the reference; otherwise the chosen `k` is determined by the one giving the largest mean PAC across algorithms
- `pac`: a data frame showing the PAC for each combination of algorithm and cluster size

- `ii`: a list of data frames for all k showing internal evaluation indices
- `ei`: a data frame showing external evaluation indices for k
- `trim.obj`: A list with 4 elements
  - `alg.keep`: algorithms kept
  - `alg.remove`: algorithms removed
  - `rank.matrix`: a matrix of ranked algorithms for every internal evaluation index
  - `top.list`: final order of ranked algorithms
  - `E.new`: A new version of a `consensus_cluster` data object

Examples

```
# Consensus clustering for multiple algorithms
set.seed(911)
x <- matrix(rnorm(500), ncol = 10)
CC <- consensus_cluster(x, nk = 3:4, reps = 10, algorithms = c("ap", "km"),
progress = FALSE)

# Evaluate algorithms on internal/external indices and trim algorithms:
# remove those ranking low on internal indices
set.seed(1)
ref.cl <- sample(1:4, 50, replace = TRUE)
z <- consensus_evaluate(x, CC, ref.cl = ref.cl, n = 1, trim = TRUE)
str(z, max.level = 2)
```

`consensus_matrix` *Consensus matrix*

Description

Returns the (weighted) consensus matrix given a data matrix

Usage

```
consensus_matrix(data, weights = NULL)
```

Arguments

`data` data matrix has rows as samples, columns as replicates

`weights` a vector of weights for each algorithm used in meta-consensus clustering. Must have `length(weights)` equal to `ncol(data)`.

Details

Given a vector of cluster assignments, we first calculate the connectivity matrix and indicator matrix. A connectivity matrix has a 1 if both samples are in the same cluster, and 0 otherwise. An indicator matrix has a 1 if both samples were selected to be used in a subsample of a consensus clustering algorithm, and 0 otherwise. Summation of connectivity matrices and indicator matrices is performed over different subsamples of the data. The consensus matrix is calculated by dividing the aggregated connectivity matrices by the aggregated indicator matrices.

If a meta-consensus matrix is desired, where consensus classes of different clustering algorithms are aggregated, we can construct a weighted meta-consensus matrix using `weights`.

Value

a consensus matrix

Note

When consensus is calculated over bootstrap samples, not every sample is used in each replication. Thus, there will be scenarios where two samples are never chosen together in any bootstrap samples. This typically happens when the number of replications is small. The coordinate in the consensus matrix for such pairs of samples is `NaN` from a 0 / 0 computation. These entries are coerced to 0.

Author(s)

Derek Chiu

Examples

```
set.seed(2)
x <- replicate(100, rbinom(100, 4, 0.2))
w <- rexp(100)
w <- w / sum(w)
cm1 <- consensus_matrix(x)
cm2 <- consensus_matrix(x, weights = w)
```

CSPA *Cluster-based Similarity Partitioning Algorithm (CSPA)*

Description

Performs hierarchical clustering on a stack of consensus matrices to obtain consensus class labels.

Usage

```
CSPA(E, k)
```

Arguments

`E`	is an array of clustering results.
`k`	number of clusters

Value

cluster assignments for the consensus class

Author(s)

Derek Chiu

References

Strehl, A., & Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. Journal of machine learning research, 3(Dec), 583-617.

See Also

Other consensus functions: `LCA()`, `LCE()`, `k_modes()`, `majority_voting()`

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
x <- consensus_cluster(dat, nk = 4, reps = 4, algorithms = c("hc", "diana"),
progress = FALSE)
CSPA(x, k = 4)
```

`dice` *Diverse Clustering Ensemble*

Description

Runs consensus clustering across subsamples, algorithms, and number of clusters (k).

Usage

```
dice(
  data,
  nk,
  p.item = 0.8,
  reps = 10,
  algorithms = NULL,
  k.method = NULL,
  nmf.method = c("brunet", "lee"),
  hc.method = "average",
  distance = "euclidean",
  cons.funs = c("kmodes", "majority", "CSPA", "LCE", "LCA"),
  sim.mat = c("cts", "srs", "asrs"),
  prep.data = c("none", "full", "sampled"),
  min.var = 1,
  seed = 1,
  seed.data = 1,
  trim = FALSE,
  reweigh = FALSE,
  n = 5,
  evaluate = TRUE,
  plot = FALSE,
  ref.cl = NULL,
  progress = TRUE
)
```

Arguments

`data` data matrix with rows as samples and columns as variables

`nk` number of clusters (k) requested; can specify a single integer or a range of integers to compute multiple k

`p.item` proportion of items to be used in subsampling within an algorithm

`reps` number of subsamples

`algorithms` vector of clustering algorithms for performing consensus clustering. Must be any number of the following: "nmf", "hc", "diana", "km", "pam", "ap", "sc", "gmm", "block", "som", "cmeans", "hdbscan". A custom clustering algorithm can be used.

`k.method` determines the method to choose k when no reference class is given. When `ref.cl` is not `NULL`, k is the number of distinct classes of `ref.cl`. Otherwise the input from `k.method` chooses k. The default is to use the PAC to choose the best k(s). Specifying an integer as a user-desired k will override the best k chosen by PAC. Finally, specifying "all" will produce consensus results for all k. The "all" method is implicitly performed when there is only one k used.

`nmf.method` specify NMF-based algorithms to run. By default the "brunet" and "lee" algorithms are called. See `NMF::nmf()` for details.

`hc.method` agglomeration method for hierarchical clustering. The the "average" method is used by default. See`stats::hclust()` for details.

`distance` a vector of distance functions. Defaults to "euclidean". Other options are given in `stats::dist()`. A custom distance function can be used.

`cons.funs` consensus functions to use. Current options are "kmodes" (k-modes), "majority" (majority voting), "CSPA" (Cluster-based Similarity Partitioning Algorithm), "LCE" (linkage clustering ensemble), "LCA" (latent class analysis)

`sim.mat` similarity matrix; choices are "cts", "srs", "asrs".

`prep.data` Prepare the data on the "full" dataset, the "sampled" dataset, or "none" (default).

`min.var` minimum variability measure threshold used to filter the feature space for only highly variable features. Only features with a minimum variability measure across all samples greater than `min.var` will be used. If `type = "conventional"`, the standard deviation is the measure used, and if `type = "robust"`, the MAD is the measure used.

`seed` random seed for knn imputation reproducibility

`seed.data` seed to use to ensure each algorithm operates on the same set of subsamples

`trim` logical; if `TRUE`, algorithms that score low on internal indices will be trimmed out

`reweigh` logical; if `TRUE`, after trimming out poor performing algorithms, each algorithm is reweighed depending on its internal indices.

`n` an integer specifying the top `n` algorithms to keep after trimming off the poor performing ones using Rank Aggregation. If the total number of algorithms is less than `n` no trimming is done.

`evaluate` logical; if `TRUE` (default), validity indices are returned. Internal validity indices are always computed. If `ref.cl` is not `NULL`, then external validity indices will also be computed.

`plot` logical; if `TRUE`, `graph_all` is called and a summary evaluation heatmap of ranked algorithms vs. internal validity indices is plotted as well.

`ref.cl` reference class

`progress` logical; should a progress bar be displayed?

Details

There are three ways to handle the input data before clustering via argument `prep.data`. The default is to use the raw data as-is ("none"). Or, we can enact `prepare_data()` on the full dataset ("full"), or the bootstrap sampled datasets ("sampled").

Value

A list with the following elements

`E` raw clustering ensemble object

`Eknn` clustering ensemble object with knn imputation used on `E`

`Ecomp` flattened ensemble object with remaining missing entries imputed by majority voting

`clusters` final clustering assignment from the diverse clustering ensemble method

`indices` if `evaluate = TRUE`, shows cluster evaluation indices; otherwise `NULL`

Author(s)

Aline Talhouk, Derek Chiu

Examples

```
library(dplyr)
data(hgsc)
dat <- hgsc[1:100, 1:50]
ref.cl <- strsplit(rownames(dat), "_") %>%
  purrr::map_chr(2) %>%
  factor() %>%
  as.integer()
dice.obj <- dice(dat, nk = 4, reps = 5, algorithms = "hc", cons.funs =
"kmodes", ref.cl = ref.cl, progress = FALSE)
str(dice.obj, max.level = 2)
```

`external_validity` *External validity indices*

Description

External **v**alidity indices compare a predicted clustering result with a reference class or gold standard.

Usage

```
ev_nmi(pred.lab, ref.lab, method = "emp")

ev_confmat(pred.lab, ref.lab)
```

Arguments

`pred.lab`	predicted labels generated by classifier
`ref.lab`	reference labels for the observations
`method`	method of computing the entropy. Can be any one of "emp", "mm", "shrink", or "sg".

Details

`ev_nmi` calculates the normalized mutual information

`ev_confmat` calculates a variety of statistics associated with confusion matrices. Accuracy, Cohen's kappa, and Matthews correlation coefficient have direct multiclass definitions, whereas all other metrics use macro-averaging.

Value

`ev_nmi` returns the normalized mutual information.

`ev_confmat` returns a tibble of the following summary statistics using `yardstick::summary.conf_mat()`:

- `accuracy`: Accuracy
- `kap`: Cohen's kappa

- sens: Sensitivity
- spec: Specificity
- ppv: Positive predictive value
- npv: Negative predictive value
- mcc: Matthews correlation coefficient
- j_index: Youden's J statistic
- bal_accuracy: Balanced accuracy
- detection_prevalence: Detection prevalence
- precision: alias for ppv
- recall: alias for sens
- f_meas: F Measure

Note

ev_nmi is adapted from infotheo::mutinformation()

Author(s)

Johnson Liu, Derek Chiu

References

Strehl A, Ghosh J. Cluster ensembles: a knowledge reuse framework for combining multiple partitions. J. Mach. Learn. Res. 2002;3:583-617.

Examples

```
set.seed(1)
E <- matrix(rep(sample(1:4, 1000, replace = TRUE)), nrow = 100, byrow =
              FALSE)
x <- sample(1:4, 100, replace = TRUE)
y <- sample(1:4, 100, replace = TRUE)
ev_nmi(x, y)
ev_confmat(x, y)
```

graphs *Graphical Displays*

Description

Graph cumulative distribution function (CDF) graphs, relative change in area under CDF curves, heatmaps, and cluster assignment tracking plots.

Usage

```
graph_cdf(mat)

graph_delta_area(mat)

graph_heatmap(mat, main = NULL)

graph_tracking(cl)

graph_all(x)
```

Arguments

mat	same as x, or a list of consensus matrices computed from x for faster results
main	heatmap title. If NULL (default), the titles will be taken from names in mat
cl	same as x, or a matrix of consensus classes computed from x for faster results
x	an object from consensus_cluster()

Details

graph_cdf plots the CDF for consensus matrices from different algorithms. graph_delta_area calculates the relative change in area under CDF curve between algorithms. graph_heatmap generates consensus matrix heatmaps for each algorithm in x. graph_tracking tracks how cluster assignments change between algorithms. graph_all is a wrapper that runs all graphing functions.

Value

Various plots from graph_*{} functions. All plots are generated using ggplot, except for graph_heatmap, which uses NMF::aheatmap(). Colours used in graph_heatmap and graph_tracking utilize RColorBrewer::brewer.pal() palettes.

Author(s)

Derek Chiu

References

https://stackoverflow.com/questions/4954507/calculate-the-area-under-a-curve

Examples

```
# Consensus clustering for 3 algorithms
library(ggplot2)
set.seed(911)
x <- matrix(rnorm(80), ncol = 10)
CC1 <- consensus_cluster(x, nk = 2:4, reps = 3,
algorithms = c("hc", "pam", "km"), progress = FALSE)

# Plot CDF
```

```
p <- graph_cdf(CC1)

# Change y label and add colours
p + labs(y = "Probability") + stat_ecdf(aes(colour = k)) +
scale_color_brewer(palette = "Set2")

# Delta Area
p <- graph_delta_area(CC1)

# Heatmaps with column side colours corresponding to clusters
CC2 <- consensus_cluster(x, nk = 3, reps = 3, algorithms = "hc", progress =
FALSE)
graph_heatmap(CC2)

# Track how cluster assignments change between algorithms
p <- graph_tracking(CC1)
```

`hgsc` *Gene expression data for High Grade Serous Carcinoma from TCGA*

Description

There are 489 samples measured on 321 genes. Sample IDs are in the row names and gene names are in the column names. This data set is used for clustering HGSC into subtypes with prognostic significance. The cluster assignments obtained by TCGA are indicated by the last six characters of each row name in hgsc: `MES.C1`, `IMM.C2`, `DIF.C4`, and `PRO.C5`

Usage

```
hgsc
```

Format

A data frame with 489 rows and 321 columns.

`impute_knn` *K-Nearest Neighbours imputation*

Description

The non-missing cases indicate the training set, and missing cases indicate the test set.

Usage

```
impute_knn(x, data, seed = 123456)
```

Arguments

`x`	clustering object
`data`	data matrix
`seed`	random seed for knn imputation reproducibility

Value

An object with (potentially not all) missing values imputed with K-Nearest Neighbours.

Note

We consider 5 nearest neighbours and the minimum vote for definite decision is 3.

Author(s)

Aline Talhouk

See Also

Other imputation functions: `impute_missing()`

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
x <- consensus_cluster(dat, nk = 4, reps = 4, algorithms = c("km", "hc",
"diana"), progress = FALSE)
x <- apply(x, 2:4, impute_knn, data = dat, seed = 1)
```

`impute_missing` *Impute missing values*

Description

Impute missing values from bootstrapped subsampling

Usage

```
impute_missing(E, data, nk)
```

Arguments

`E`	4D array of clusterings from `consensus_cluster`. The number of rows is equal to the number of cases to be clustered, number of columns is equal to the clusterings obtained by different resamplings of the data, the third dimension are the different algorithms and the fourth dimension are cluster sizes.
`data`	data matrix with samples as rows and genes/features as columns
`nk`	cluster size to extract data for (single value)

Details

The default output from consensus_cluster will undoubtedly contain NA entries because each replicate chooses a random subset (with replacement) of all samples. Missing values should first be imputed using impute_knn(). Not all missing values are guaranteed to be imputed by KNN. See class::knn() for details. Thus, any remaining missing values are imputed using majority voting.

Value

If flattened matrix consists of more than one repetition, i.e. it isn't a column vector, then the function returns a matrix of clusterings with complete cases imputed using majority voting, and relabelled, for chosen k.

Author(s)

Aline Talhouk

See Also

Other imputation functions: impute_knn()

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
E <- consensus_cluster(dat, nk = 3:4, reps = 10, algorithms = c("hc", "km",
"sc"), progress = FALSE)
sum(is.na(E))
E_imputed <- impute_missing(E, dat, 4)
sum(is.na(E_imputed))
```

k_modes *K-modes*

Description

Combine clustering results using K-modes.

Usage

```
k_modes(E, is.relabelled = TRUE, seed = 1)
```

Arguments

`E`	a matrix of clusterings with number of rows equal to the number of cases to be clustered, number of columns equal to the clustering obtained by different resampling of the data, and the third dimension are the different algorithms. Matrix may already be two-dimensional.
`is.relabelled`	logical; if `FALSE` the data will be relabelled using the first clustering as the reference.
`seed`	random seed for reproducibility

Details

Combine clustering results generated using different algorithms and different data perturbations by k-modes. This method is the categorical data analog of k-means clustering. Complete cases are needed: i.e. no `NA`s. If the matrix contains `NA`s those are imputed by majority voting (after class relabeling).

Value

a vector of cluster assignments based on k-modes

Author(s)

Aline Talhouk

References

Luo, H., Kong, F., & Li, Y. (2006, August). Combining multiple clusterings via k-modes algorithm. In International Conference on Advanced Data Mining and Applications (pp. 308-315). Springer, Berlin, Heidelberg.

See Also

Other consensus functions: `CSPA()`, `LCA()`, `LCE()`, `majority_voting()`

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
cc <- consensus_cluster(dat, nk = 4, reps = 6, algorithms = "pam", progress =
FALSE)
table(k_modes(cc[, , 1, 1, drop = FALSE], is.relabelled = FALSE))
```

LCA *Latent Class Analysis*

Description

Combine clustering results using latent class analysis.

Usage

```
LCA(E, is.relabelled = TRUE, seed = 1)
```

Arguments

`E` a matrix of clusterings with number of rows equal to the number of cases to be clustered, number of columns equal to the clustering obtained by different resampling of the data, and the third dimension are the different algorithms. Matrix may already be two-dimensional.

`is.relabelled` logical; if `FALSE` the data will be relabelled using the first clustering as the reference.

`seed` random seed for reproducibility

Value

a vector of cluster assignments based on LCA

Author(s)

Derek Chiu

See Also

Other consensus functions: `CSPA()`, `LCE()`, `k_modes()`, `majority_voting()`

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
cc <- consensus_cluster(dat, nk = 4, reps = 6, algorithms = "pam", progress =
FALSE)
table(LCA(cc[, , 1, 1, drop = FALSE], is.relabelled = FALSE))
```

LCE *Linkage Clustering Ensemble*

Description

Generate a cluster assignment from a CTS, SRS, or ASRS similarity matrix.

Usage

```
LCE(E, k, dc = 0.8, R = 10, sim.mat = c("cts", "srs", "asrs"))
```

Arguments

`E`	is an array of clustering results. An error is thrown if there are missing values. `impute_missing()` can be used beforehand.
`k`	requested number of clusters
`dc`	decay constant for CTS, SRS, or ASRS matrix
`R`	number of repetitions for SRS matrix
`sim.mat`	similarity matrix; choices are "cts", "srs", "asrs".

Value

a vector containing the cluster assignment from either the CTS, SRS, or ASRS similarity matrices

Author(s)

Johnson Liu

See Also

Other consensus functions: `CSPA()`, `LCA()`, `k_modes()`, `majority_voting()`

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
x <- consensus_cluster(dat, nk = 4, reps = 4, algorithms = c("km", "hc"),
progress = FALSE)
## Not run:
LCE(E = x, k = 4, sim.mat = "asrs")

## End(Not run)

x <- apply(x, 2:4, impute_knn, data = dat, seed = 1)
x_imputed <- impute_missing(x, dat, nk = 4)
LCE(E = x_imputed, k = 4, sim.mat = "cts")
```

majority_voting *Majority voting*

Description

Combine clustering results using majority voting.

Usage

```
majority_voting(E, is.relabelled = TRUE)
```

Arguments

E	a matrix of clusterings with number of rows equal to the number of cases to be clustered, number of columns equal to the clustering obtained by different resampling of the data, and the third dimension are the different algorithms. Matrix may already be two-dimensional.
is.relabelled	logical; if FALSE the data will be relabelled using the first clustering as the reference.

Details

Combine clustering results generated using different algorithms and different data perturbations by majority voting. The class of a sample is the cluster label which was selected most often across algorithms and subsamples.

Value

a vector of cluster assignments based on majority voting

Author(s)

Aline Talhouk

See Also

Other consensus functions: CSPA(), LCA(), LCE(), k_modes()

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
cc <- consensus_cluster(dat, nk = 4, reps = 6, algorithms = "pam", progress =
FALSE)
table(majority_voting(cc[, , 1, 1, drop = FALSE], is.relabelled = FALSE))
```

`min_fnorm` *Minimize Frobenius norm for between two matrices*

Description

Finds a permutation of a matrix such that its Frobenius norm with another matrix is minimized.

Usage

```
min_fnorm(A, B = diag(nrow(A)))
```

Arguments

`A`	data matrix we want to permute
`B`	matrix whose distance with the permuted A we want to minimize. By default, `B <- diag(nrow(A))`, so the permutation maximizes the trace of A.

Details

Finds the permutation P of A such that `||PA - B||` is minimum in Frobenius norm. Uses the linear-sum assignment problem (LSAP) solver in the package `clue`. The default B is the identity matrix of same dimension, so that the permutation of A maximizes its trace. This procedure is useful for constructing a confusion matrix when we don't know the true class labels of a predicted class and want to compare to a reference class.

Value

Permuted matrix such that it is the permutation of A closest to B

Author(s)

Ravi Varadhan: https://stat.ethz.ch/pipermail/r-help/2010-April/236664.html

Examples

```
set.seed(1)
A <- matrix(sample(1:25, size = 25, rep = FALSE), 5, 5)
min_fnorm(A)
```

PAC *Proportion of Ambiguous Clustering*

Description

Given a consensus matrix, returns the proportion of ambiguous clusters (PAC). This is a robust way to assess clustering performance.

Usage

```
PAC(cm, lower = 0, upper = 1)
```

Arguments

`cm`	consensus matrix. Should be symmetric and values between 0 and 1.
`lower`	the lower bound that determines what is ambiguous
`upper`	the upper bound that determines what is ambiguous

Details

Since a consensus matrix is symmetric, we only look at its lower (or upper) triangular matrix. The proportion of entries strictly between `lower` and `upper` is the PAC. In a perfect clustering, the consensus matrix would consist of only 0s and 1s, and the PAC assessed on the (0, 1) interval would have a perfect score of 0. Using a (0.1, 0.9) interval for defining ambiguity is common as well.

The PAC is not, strictly speaking, an internal validity index. Originally used to choose the optimal number of clusters, here we use it to assess cluster stability. However, PAC is still agnostic any gold standard clustering result so we use it like an internal validity index.

Value

the PAC is a score used in clustering performance. The lower it is the better, because we want minimal ambiguity amongst the consensus.

Author(s)

Derek Chiu

References

Senbabaoglu, Y., Michailidis, G., & Li, J. Z. (2014). Critical limitations of consensus clustering in class discovery. Scientific reports, 4.

Examples

```
set.seed(1)
x <- replicate(100, rbinom(100, 4, 0.2))
y <- consensus_matrix(x)
PAC(y, lower = 0.05, upper = 0.95)
```

pcn *Simulate and select null distributions on empirical gene-gene correlations*

Description

Using a principal component constructed from the sample space, we simulate null distributions with univariate Normal distributions using `pcn_simulate`. Then a subset of these distributions is chosen using `pcn_select`.

Usage

```
pcn_simulate(data, n.sim = 50)

pcn_select(data.sim, cl, type = c("rep", "range"), int = 5)
```

Arguments

`data`	data matrix with rows as samples, columns as features
`n.sim`	The number of simulated datasets to simulate
`data.sim`	an object from `pcn_simulate`
`cl`	vector of cluster memberships
`type`	select either the representative dataset ("rep") or a range of datasets ("range")
`int`	every `int` data sets from median-ranked `data.sim` are taken. Defaults to 5.

Value

`pcn_simulate` returns a list of length `n.sim`. Each element is a simulated matrix using this "Principal Component Normal" (pcn) procedure.

`pcn_select` returns a list with elements

- `ranks`: When `type = "range"`, ranks of each extracted dataset shown
- `ind`: index of representative simulation
- `dat`: simulation data representation of all in pcNormal

Author(s)

Derek Chiu

Examples

```
set.seed(9)
A <- matrix(rnorm(300), nrow = 20)
pc.dat <- pcn_simulate(A, n.sim = 50)
cl <- sample(1:4, 20, replace = TRUE)
pc.select <- pcn_select(pc.dat, cl, "rep")
```

`prepare_data` *Prepare data for consensus clustering*

Description

Perform feature selection or dimension reduction to remove noise variables.

Usage

```
prepare_data(
  data,
  scale = TRUE,
  type = c("conventional", "robust", "tsne"),
  min.var = 1
)
```

Arguments

`data`	data matrix with rows as samples and columns as variables
`scale`	logical; should the data be centered and scaled?
`type`	if we use "conventional" measures (default), then the mean and standard deviation are used for centering and scaling, respectively. If "robust" measures are specified, the median and median absolute deviation (MAD) are used. Alternatively, we can apply "tsne" for dimension reduction.
`min.var`	minimum variability measure threshold used to filter the feature space for only highly variable features. Only features with a minimum variability measure across all samples greater than `min.var` will be used. If `type = "conventional"`, the standard deviation is the measure used, and if `type = "robust"`, the MAD is the measure used.

Details

We can apply a basic filtering method of feature selection that removes variables with low signal and (optionally) scales before consensus clustering. Or, we can use t-SNE dimension reduction to transform the data to just two variables. This lower-dimensional embedding allows algorithms such as hierarchical clustering to achieve greater performance.

Value

dataset prepared for usage in `consensus_cluster`

Author(s)

Derek Chiu

Examples

```
set.seed(2)
x <- replicate(10, rnorm(100))
x.prep <- prepare_data(x)
dim(x)
dim(x.prep)
```

---

relabel_class *Relabel classes to a standard*

Description

Relabel clustering categories to match to a standard by minimizing the Frobenius norm between the two labels.

Usage

```
relabel_class(pred.cl, ref.cl)
```

Arguments

pred.cl	vector of predicted cluster assignments
ref.cl	vector of reference labels to match to

Value

A vector of relabeled cluster assignments

Author(s)

Aline Talhouk

Examples

```
set.seed(2)
pred <- sample(1:4, 100, replace = TRUE)
true <- sample(1:4, 100, replace = TRUE)
relabel_class(pred, true)
```

sigclust *Significant Testing of Clustering Results*

Description

Uses the SigClust K-Means algorithm to assess significance of clustering results.

Usage

```
sigclust(x, k, nsim, nrep = 1, labflag = 0, label = 0, icovest = 2)
```

Arguments

x	data matrix, samples are rows and features are columns
k	cluster size to test against
nsim	number of simulations
nrep	See sigclust::sigclust() for details.
labflag	See sigclust::sigclust() for details.
label	true class label. See sigclust::sigclust() for details.
icovest	type of covariance matrix estimation

Details

This function is a wrapper for the original sigclust::sigclust(), except that an additional parameter k is allows testing against any number of clusters. In addition, the default type of covariance estimation is also different.

Value

An object of class sigclust. See sigclust::sigclust() for details.

Author(s)

Hanwen Huang: <hanwenh@email.unc.edu>; Yufeng Liu: <yfliu@email.unc.edu>; J. S. Marron: <marron@email.unc.edu>

References

Liu, Yufeng, Hayes, David Neil, Nobel, Andrew and Marron, J. S, 2008, *Statistical Significance of Clustering for High-Dimension, Low-Sample Size Data, Journal of the American Statistical Association* **103**(483) 1281–1293.

Examples

```
data(hgsc)
dat <- hgsc[1:100, 1:50]
nk <- 4
cc <- consensus_cluster(dat, nk = nk, reps = 5, algorithms = "pam",
progress = FALSE)
cl.mat <- consensus_combine(cc, element = "class")
lab <- cl.mat$`4`[, 1]
set.seed(1)
str(sigclust(x = dat, k = nk, nsim = 50, labflag = 1, label = lab))
```

---

similarity *Similarity Matrices*

Description

cts computes the connected triple based similarity matrix, srs computes the simrank based similarity matrix, and asrs computes the approximated simrank based similarity matrix.

Usage

```
cts(E, dc)

srs(E, dc, R)

asrs(E, dc)
```

Arguments

E	an N by M matrix of cluster ensembles
dc	decay factor, ranges from 0 to 1 inclusive
R	number of iterations for srs

Value

an N by N CTS, SRS, or ASRS matrix

Author(s)

Johnson Liu, Derek Chiu

References

MATLAB functions cts, srs, asrs in package LinkCluE by Simon Garrett

Examples

```
set.seed(1)
E <- matrix(rep(sample(1:4, 800, replace = TRUE)), nrow = 100)
CTS <- cts(E = E, dc = 0.8)
SRS <- srs(E = E, dc = 0.8, R = 3)
ASRS <- asrs(E = E, dc = 0.8)
purrr::walk(list(CTS, SRS, ASRS), str)
```

Index