Package 'MOFSR'

November 28, 2024

Type Package

Title Multimodality Fusion Subtyping

Author Zaoqu Liu [aut, cre] (<https://orcid.org/0000-0002-0452-742X>)

Maintainer Zaoqu Liu <liuzaoqu@163.com>

Description

MOFSR is capable of integrating multimodal biological data. It can discover potential disease subtypes and biological mechanisms through a variety of clustering algorithms and feature selection methods. Additionally, it enables cluster quality assessment, classification prediction, and functional enrichment analysis, facilitating in-depth analysis and interpretation of biological data and promoting the development of precision medicine and related research.

Version 1.0.0

License GPL (>= 3)

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

RoxygenNote 7.3.1

Imports dplyr

Contents

`CalCHI` *Calinski-Harabasz Index Calculation*

Description

This function calculates the Calinski-Harabasz index for evaluating the quality of clustering solutions. It is used to determine the optimal number of clusters in a hierarchical clustering scenario.

Usage

```
CalCHI(
  hclust_result,
  dist_matrix = NULL,
  max_clusters = round(1 + 3.3 * log(length(hclust_result$order), 10))
)
```

Arguments

hclust_result	A hierarchical clustering object (result of hclust function).
dist_matrix	An optional distance matrix. If not provided, the cophenetic matrix is used. The matrix should be of type "euclidean".
max_clusters	Integer. The maximum number of clusters to evaluate (default: round(1 + 3.3 * log(length(hclust_result$order), 10))).

Details

The function calculates the Calinski-Harabasz index for different numbers of clusters to help identify the optimal number of clusters. The index measures the ratio of between-cluster dispersion to within-cluster dispersion.

Value

A vector containing the Calinski-Harabasz index values for each number of clusters.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data <- mtcars
dist_matrix <- dist(data)
hclust_result <- hclust(dist_matrix)
calinski_values <- CalCHI(hclust_result, dist_matrix)
```

CalPAC *Calculate Proportion of Ambiguous Clustering (PAC)*

Description

This function calculates the Proportion of Ambiguous Clustering (PAC) to help evaluate the optimal number of clusters in a consensus clustering analysis.

Usage

```
CalPAC(consensus_result, range_clusters = 2:6, x1 = 0.1, x2 = 0.9)
```

Arguments

consensus_result	A list containing consensus clustering results from ConsensusClusterPlus.
range_clusters	Integer vector. The range of cluster numbers evaluated during consensus clustering.
x1	Numeric. Lower bound for defining the PAC (default: 0.1).
x2	Numeric. Upper bound for defining the PAC (default: 0.9).

Details

The PAC is calculated as the difference between the cumulative distribution function values at two thresholds, x2 and x1, on the consensus matrix values. A lower PAC indicates a more stable clustering solution.

Value

A data frame containing the PAC values for each number of clusters.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
data <- mtcars
cc_res <- RunCC(data)
pac_values <- CalPAC(cc_res)
pac_values
```

Classifier.Adaboost *AdaBoost Classifier for Cluster Prediction*

Description

This function performs classification using AdaBoost to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.Adaboost(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains an AdaBoost model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.Adaboost(
```

```
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.DT` *Decision Tree Classifier for Cluster Prediction*

Description

This function performs classification using Decision Tree to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.DT(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a Decision Tree model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.DT(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.Enet *Elastic Net Classifier for Cluster Prediction*

Description

This function performs classification using Elastic Net to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.Enet(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains an Elastic Net model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.Enet(
```

```
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.Enrichment` *Enrichment-Based Neural Network Classifier for Cluster Prediction*

Description

This function performs classification using pathway enrichment analysis and a neural network to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.Enrichment(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE,
  nCores = 5
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names, 'OR' as odds ratio, and 'AUC' as area under curve.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.
`nCores`	An integer indicating the number of cores to use for pathway enrichment analysis. Default is 5.

Details

The function operates as follows: 1. Ensures that the ‘cluster.data‘ has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Filters out markers with low odds ratio. 6. Performs pathway enrichment analysis using the ssMwwGST method.

Value

A data frame with: - ID: The sample identifier. - Cluster: The predicted cluster label for each sample. - NES: Normalized Enrichment Score (NES) for each cluster assignment.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.Enrichment(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.GBDT *Gradient Boosted Decision Trees (GBDT) Classifier for Cluster Prediction*

Description

This function performs classification using Gradient Boosted Decision Trees (GBDT) to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.GBDT(
  data.test,
  data.train,
  cluster.data,
```

```
  cluster.markers,
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a Gradient Boosted Decision Trees model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
```

```
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.GBDT(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.kNN` *k-Nearest Neighbors (kNN) Classifier for Cluster Prediction*

Description

This function performs classification using k-Nearest Neighbors to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.kNN(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the ‘cluster.data‘ has the correct column names and matches training data. 2. Scales the test data for prediction if ‘scale‘ is TRUE. 3. Selects genes that are common between the test and training datasets. 4. Uses glmnet to identify the important markers for each cluster and trains a k-Nearest Neighbors (kNN) model for classification. 5. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Cluster: The predicted cluster label for each sample. - Probabilities: The probabilities for each cluster assignment.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.kNN(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.LASSO *LASSO Classifier for Cluster Prediction*

Description

This function performs classification using LASSO (Least Absolute Shrinkage and Selection Operator) to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.LASSO(
  data.test,
  data.train,
  cluster.data,
```

```
  cluster.markers,
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a LASSO model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
```

```
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.LASSO(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.LDA` *Linear Discriminant Analysis (LDA) Classifier for Cluster Prediction*

Description

This function performs classification using Linear Discriminant Analysis (LDA) to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.LDA(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains an LDA model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.LDA(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.NBayes *Naive Bayes Classifier for Cluster Prediction*

Description

This function performs classification using Naive Bayes to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.NBayes(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
```

```
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a Naive Bayes model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
```

```
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.NBayes(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.NNet` *Neural Network Classifier for Cluster Prediction*

Description

This function performs classification using a neural network to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.NNet(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the ‘cluster.data‘ has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a neural network model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.NNet(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.PCA *PCA-Based Neural Network Classifier for Cluster Prediction*

Description

This function performs classification using PCA and a neural network to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.PCA(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
```

```
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and performs PCA to reduce dimensionality. 6. Trains a neural network model for classification using the top PCs. 7. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
```

```
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.PCA(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.RF` *Random Forest Classifier for Cluster Prediction*

Description

This function performs classification using Random Forest to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.RF(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the ‘cluster.data‘ has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction if ‘scale‘ is TRUE. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a Random Forest model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Cluster: The predicted cluster label for each sample. - Probabilities: The probabilities for each cluster assignment.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.RF(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.Ridge *Ridge Classifier for Cluster Prediction*

Description

This function performs classification using Ridge Regression to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.Ridge(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
```

```
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains a Ridge model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
```

```
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.Ridge(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.StepLR` *Stepwise Logistic Regression Classifier for Cluster Prediction*

Description

This function performs classification using Stepwise Logistic Regression to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.StepLR(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the ‘cluster.data‘ has the correct column names. 2. Scales the test data for prediction. 3. Selects genes that are common between the test and training datasets. 4. Uses glmnet to identify the important markers for each cluster and trains a multinomial logistic regression model for classification. 5. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.StepLR(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.SVD *SVD-Based Neural Network Classifier for Cluster Prediction*

Description

This function performs classification using SVD and a neural network to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.SVD(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
```

```
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and performs SVD to reduce dimensionality. 6. Trains a neural network model for classification using the top singular vectors. 7. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
```

```
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.SVD(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

`Classifier.SVM` *Support Vector Machine (SVM) Classifier for Cluster Prediction*

Description

This function performs classification using Support Vector Machine (SVM) to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.SVM(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the ‘cluster.data‘ has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains an SVM model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.SVM(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

Classifier.XGBoost *XGBoost Classifier for Cluster Prediction*

Description

This function performs classification using XGBoost to predict cluster assignments for test data based on trained models from training data and cluster markers.

Usage

```
Classifier.XGBoost(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
```

```
  scale = TRUE
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
scale	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function operates as follows: 1. Ensures that the 'cluster.data' has the correct column names. 2. Adds a one-hot encoded matrix for cluster assignments. 3. Scales the test data for prediction. 4. Selects genes that are common between the test and training datasets. 5. Uses glmnet to identify the important markers for each cluster and trains an XGBoost model for classification. 6. Predicts the cluster for test samples and provides probabilities for each cluster.

Value

A data frame with: - ID: The sample identifier. - Probabilities: The probabilities for each cluster assignment. - Predict: The predicted cluster label for each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
cluster.data <- data.frame(
  Sample = paste0("Sample", 1:60),
  Cluster = rep(paste0("C", 1:3), each = 20)
)
data.train <- matrix(rnorm(6000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), cluster.data$Sample)
)
data.test <- matrix(rnorm(5000),
  nrow = 100,
  dimnames = list(paste0("Gene", 1:100), paste0("P", 1:50))
)
cluster.markers <- setNames(
  lapply(
    unique(cluster.data$Cluster),
    function(cluster) {
      data.frame(Gene = sample(rownames(data.train), 10))
    }
```

```
  ),
  unique(cluster.data$Cluster)
)
result <- Classifier.XGBoost(
  data.test = data.test, data.train = data.train,
  cluster.data, cluster.markers
)
head(result)
```

CV *Calculate Coefficient of Variation for a Numeric Vector*

Description

This function calculates the coefficient of variation (CV) for a given numeric vector.

Usage

```
CV(V)
```

Arguments

V	A numeric vector.

Value

The coefficient of variation (CV) for the input vector.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

CV.df *Calculate Coefficient of Variation for a Data Frame*

Description

This function calculates the coefficient of variation (CV) for each column in a data frame.

Usage

```
CV.df(df)
```

Arguments

df	A data frame with row samples and column features.

Value

A sorted vector of CV values for each column in the data frame, in decreasing order.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

`Find.OptClusterFeatures` *Optimal Feature Combination for Multi-Modality Clustering*

Description

This function selects the optimal combination of features for multi-modality clustering analysis by integrating various modalities (e.g., mutation, CNV, RNA, protein, pathology, radiology) to explore the optimal number of clusters. It uses Nonnegative Matrix Factorization (NMF) for cluster consistency analysis and Multi-block Principal Component Analysis (mbPCA) for assessing cluster separation. The combination of these methods helps identify distinct biological subgroups and evaluate the stability and biological relevance of clustering.

Usage

```
Find.OptClusterFeatures(
  data_layers,
  feature_subset_sizes,
  try_num_clusters = 2:6,
  n_runs = 5,
  n_fold = 5
)
```

Arguments

`data_layers`	A named list of matrices representing different modalities. Each matrix should have features as rows and samples as columns. A minimum of two datasets is required.
`feature_subset_sizes`	A list of sequences representing possible feature subset sizes for each modality. The names of this list must match the names in data_layers.
`try_num_clusters`	Integer vector. The range of cluster numbers to be tested (default: 2:6).
`n_runs`	Integer. Number of iterations for NMF for each cluster number (default: 5).
`n_fold`	Integer. Number of folds for cross-validation in NMF (default: 5).

Value

A list containing: - optimal_combination: A dataframe with the optimal feature combination and associated clustering score. - all_results: A dataframe containing CPI and GAP scores for all tested feature combinations and cluster numbers.

Author(s)

Zaoqu Liu

gene_sets *Functional Gene Sets*

Description

The gene_sets data object contains a unified collection of gene sets derived from multiple sources, including curated pathways (c2.cp.v2022.1.Hs), Gene Ontology terms (c5.go.v2022.1.Hs), and hallmark gene sets (h.all.v2022.1.Hs). These gene sets are based on version 2022.1 and represent key biological processes, pathways, and well-defined biological states. By integrating these sources, the gene_sets object provides a comprehensive dataset that can be utilized for enrichment analysis and functional exploration, offering valuable insights into underlying biological mechanisms.

Usage

```
gene_sets
```

Format

data.frame

get.binary.clusters *Get Binary Clusters from Clustering Results*

Description

This function extracts binary cluster assignments from multiple clustering results.

Usage

```
get.binary.clusters(res)
```

Arguments

res	A list containing clustering results

Value

A data frame where each row represents a binary encoding of cluster assignments across different methods.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

get.class *Get Cluster Assignments*

Description

Extract cluster assignments from Consensus Clustering results for a specific number of clusters.

Usage

```
get.class(cc.res, k)
```

Arguments

cc.res	Consensus clustering results from ConsensusClusterPlus.
k	Integer. The number of clusters to extract.

Value

A data frame with the following columns: - ID: The sample identifier. - Cluster: The assigned cluster label, prefixed by 'C'.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
data <- mtcars
cc_res <- RunCC(data)
clu <- get.class(cc_res, 2)
clu
```

get.Jaccard.Distance *Jaccard Distance Calculation for Binary Matrix*

Description

This function calculates the Jaccard distance or similarity for a binary matrix. It is typically used to evaluate the similarity or dissimilarity between columns of a binary matrix.

Usage

```
get.Jaccard.Distance(data, dissimilarity = TRUE)
```

Arguments

data	A binary matrix where rows represent features and columns represent samples.
dissimilarity	Logical. If TRUE, returns the Jaccard distance; if FALSE, returns the Jaccard similarity (default: TRUE).

Details

The function computes the Jaccard distance (or similarity) between each pair of columns in the input binary matrix. The Jaccard distance is calculated as 1 minus the Jaccard similarity.

Value

A matrix containing the Jaccard distance or similarity between each pair of columns in the input matrix.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
data <- matrix(sample(0:1, 1500, replace = TRUE), nrow = 30, ncol = 50)
jaccard_dist <- get.Jaccard.Distance(as.data.frame(data), dissimilarity = TRUE)
jaccard_dist
```

MAD.df *Calculate Median Absolute Deviation for a Data Frame*

Description

This function calculates the median absolute deviation (MAD) for each column in a data frame.

Usage

```
MAD.df(df)
```

Arguments

df	A data frame with row samples and column features.

Value

A sorted vector of MAD values for each column in the data frame, in decreasing order.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

Mean.df *Calculate Mean Value for a Data Frame*

Description

This function calculates the mean value for each column in a data frame.

Usage

```
Mean.df(df)
```

Arguments

df	A data frame with row samples and column features.

Value

A vector of mean values for each column in the data frame.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

Median.df *Calculate Median Value for a Data Frame*

Description

This function calculates the median value for each column in a data frame.

Usage

```
Median.df(df)
```

Arguments

df	A data frame with row samples and column features.

Value

A vector of median values for each column in the data frame.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

minmax *Minmax Normalization for a Numeric Vector*

Description

This function performs min-max normalization on a numeric vector.

Usage

```
minmax(x)
```

Arguments

x	A numeric vector.

Value

A normalized numeric vector with values between 0 and 1.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

minmax.df *Minmax Normalization for a Data Frame*

Description

This function performs min-max normalization on each column in a data frame.

Usage

```
minmax.df(data)
```

Arguments

data	A data frame with row samples and column features.

Value

A data frame with normalized values between 0 and 1 for each column.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

PathDEA *Pathway Differential Expression Analysis (PathDEA)*

Description

This function performs pathway differential expression analysis based on clustering results and pathway activity scores derived from ssMwwGST.

Usage

```
PathDEA(
  Cluster_data,
  ssMwwGST_results,
  dea_FDR_threshold = 0.001,
  dea_gap_threshold = 1.5
)
```

Arguments

`Cluster_data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments.
`ssMwwGST_results`	A list of results from ssMwwGST, including NES (Normalized Enrichment Scores).
`dea_FDR_threshold`	Numeric. The FDR threshold to use for filtering significant pathways. Default is 0.001.
`dea_gap_threshold`	Numeric. The median gap threshold to use for filtering significant pathways. Default is 1.5.

Details

The function operates as follows: 1. Extracts Normalized Enrichment Scores (NES) from the ssMwwGST results. 2. Performs Wilcoxon rank-sum tests to compare pathway activity between clusters for each pathway. 3. Calculates median and mean differences in pathway activity between clusters. 4. Adjusts p-values using the Benjamini-Hochberg method to control the false discovery rate (FDR).

Value

A list containing: - dea_path: A list of data frames, each containing the differential expression analysis results for each cluster. - dea_path2: A list of data frames containing the filtered differential expression analysis results for each cluster based on FDR and median gap thresholds. - NES: A data frame of Normalized Enrichment Scores for each gene set and each sample. - Cluster: A data frame of sample IDs and their corresponding cluster assignments.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
Cluster_data <- data.frame(Sample = paste0("Sample", 1:10), Cluster = rep(1:2, each = 5))
ssMwwGST_results <- list(NES = matrix(rnorm(100),
  nrow = 10, ncol = 10,
  dimnames = list(paste0("Pathway", 1:10), paste0("Sample", 1:10))
))
result <- PathDEA(Cluster_data, ssMwwGST_results)
```

RunBCC *Run Bayesian Consensus Clustering (BCC) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Bayesian Consensus Clustering (BCC) to integrate multiple omics datasets. BCC is a Bayesian method that helps capture shared patterns across different datasets by finding a consensus clustering solution.

Usage

```
RunBCC(data = NULL, N.clust = NULL, max.iterations = 10)
```

Arguments

`data`	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
`N.clust`	Integer. Number of clusters to create from the BCC components (optional but recommended).
`max.iterations`	Integer. Maximum number of iterations for the Bayesian algorithm (default: 10).

Details

This function uses BCC to integrate multiple data matrices and assign clusters to the samples. BCC uses a Bayesian approach to identify shared patterns across multiple datasets, providing a robust clustering solution.

The function operates as follows: 1. Each matrix in the input list is converted to a matrix to ensure compatibility. 2. BCC is used to identify shared patterns across different modalities. 3. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'BCC' to indicate that the clustering was performed using BCC.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Lock EF, Dunson DB. Bayesian consensus clustering. Bioinformatics. 2013;29(20):2610-2616. doi:10.1093/bioinformatics/btt425.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run BCC clustering
result <- RunBCC(data = data_list, N.clust = 3)
```

RunCC *Run Consensus Clustering using ConsensusClusterPlus*

Description

This function performs consensus clustering on the given data using the ConsensusClusterPlus package.

Usage

```
RunCC(
  data,
  maxK = 6,
  reps = 1000,
  pItem = 0.8,
  pFeature = 1,
  clusterAlg = "hc",
  distance = "euclidean",
  title = "Consensus Clustering",
  plot = TRUE
)
```

Arguments

data	A numeric matrix or data frame where rows represent features and columns represent samples.
maxK	The maximum number of clusters to evaluate (default: 6).
reps	Number of subsamples (default: 1000).
pItem	Proportion of items to sample (default: 0.8).
pFeature	Proportion of features to sample (default: 1).

clusterAlg	The clustering algorithm to use, either "hc" for hierarchical or "km" for k-means (default: "hc").
distance	The distance metric to use, "pearson", "spearman", "euclidean", etc. (default: "euclidean").
title	Optional title for the results (default: "Consensus Clustering").
plot	Whether to plot the consensus matrix and dendrogram (default: TRUE).

Details

This function leverages the ConsensusClusterPlus package to perform consensus clustering, which is useful for identifying robust clusters in genomic data or other high-dimensional data.

Value

A list containing the consensus clustering results.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data <- matrix(rnorm(1000), nrow = 100, ncol = 10)
result <- RunCC(data, maxK = 4)
```

RunCIMLR *Run Consensus Iterative Multi-view Learning (CIMLR) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Consensus Iterative Multi-view Learning (CIMLR) to integrate multiple omics datasets. CIMLR is useful for discovering shared patterns across multiple datasets and identifying distinct subtypes.

Usage

```
RunCIMLR(
  data,
  N.clust = NULL,
  num.dimensions = NA,
  tuning.parameter = 10,
  cores.ratio = 1
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to create (optional but recommended).
num.dimensions	Integer. Number of dimensions for CIMLR (default: NA).
tuning.parameter	Integer. Tuning parameter for CIMLR (default: 10).
cores.ratio	Numeric. Ratio of the number of cores to be used when computing the multi-kernel (default: 1).

Details

This function uses CIMLR to integrate multiple data matrices and assign clusters to the samples. CIMLR is particularly effective for multi-view learning and discovering common patterns among different data types.

The function operates as follows: 1. CIMLR is performed using the CIMLR package to extract components that summarize the shared variation across different modalities. 2. Feature ranking is applied to identify the most important features across all data. 3. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'CIMLR' to indicate that the clustering was performed using CIMLR.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Ramazzotti D, Lal A, Wang B, Batzoglou S, Sidow A. Multi-omic tumor data reveal diversity of molecular mechanisms that correlate with survival. Nat Commun. 2018;9(1):4453. doi:10.1038/s41467-018-06921-8.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run CIMLR clustering
result <- RunCIMLR(data = data_list, N.clust = 3)
```

`RunClassifier` *Run Classifiers for Cluster Prediction*

Description

This function runs different classification models based on user input to predict cluster assignments for test data.

Usage

```
RunClassifier(
  algorithm,
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  scale = TRUE
)
```

Arguments

`algorithm`	A character string indicating the classifier to use. Supported algorithms include: "Adaboost", "DT", "Enet", "Enrichment", "GBDT", "LASSO", "LDA", "NBayes", "NNet", "PCA", "Ridge", "StepLR", "SVD", "SVM", "XGBoost", "kNN", "RF".
`data.test`	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
`data.train`	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
`cluster.data`	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
`cluster.markers`	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
`scale`	A logical value indicating whether to scale the test data. Default is TRUE.

Details

The function dynamically selects and runs a classification model based on user input. The supported classifiers include a range of machine learning models such as Random Forest, kNN, PCA, SVM, LASSO, Ridge, and others.

Value

A data frame containing the prediction results based on the selected algorithm.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data.test <- matrix(rnorm(1000), nrow = 100, ncol = 10)
data.train <- matrix(rnorm(1000), nrow = 100, ncol = 10)
cluster.data <- data.frame(Sample = paste0("Sample", 1:10), Cluster = rep(1:2, each = 5))
cluster.markers <- setNames(lapply(unique(cluster.data$Cluster), function(c) data.frame(Gene = paste0("Gene"
result <- RunClassifier(algorithm = "RF", data.test, data.train, cluster.data, cluster.markers, scale = TRUE)
```

RunCOCA *Run Consensus Clustering Analysis (COCA)*

Description

This function performs Consensus Clustering Analysis (COCA) using the ConsensusClusterPlus package to identify stable clusters in the input data.

Usage

```
RunCOCA(
  jaccard.matrix,
  max.clusters = 6,
  optimal.clusters = 3,
  linkage.method = "ward.D2",
  clustering.algorithm = "hc",
  distance.metric = "euclidean",
  resampling.iterations = 10000,
  resample.proportion = 0.7
)
```

Arguments

`jaccard.matrix`	A Jaccard distance matrix, typically obtained from binary data.
`max.clusters`	Integer. The maximum number of clusters to evaluate (default: 6).
`optimal.clusters`	Integer. The optimal number of clusters to select (default: 3).
`linkage.method`	Character. The linkage method for hierarchical clustering (default: "ward.D2").
`clustering.algorithm`	Character. The clustering algorithm to use (default: 'hc').
`distance.metric`	Character. The distance metric to use (default: "euclidean").
`resampling.iterations`	Integer. The number of resampling iterations (default: 10000).
`resample.proportion`	Numeric. Proportion of items to resample in each iteration (default: 0.7).

Details

This function uses ConsensusClusterPlus to perform consensus clustering on the input Jaccard distance matrix, evaluates the stability of different clustering solutions using PAC, and returns the clustering assignments.

Value

A list containing the consensus clustering results, optimal cluster solution, PAC values, and final cluster assignments.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
jaccard_matrix <- CalJaccardDistance(data)
coca_result <- RunCOCA(jaccard.matrix = jaccard_matrix, max.clusters = 6, optimal.clusters = 3)
```

RunCPCA *Run Consensus Principal Component Analysis (CPCA) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Consensus Principal Component Analysis (CPCA) to integrate multiple omics data. CPCA helps capture the shared variation across different data modalities and provides insights into the common structure of the data.

Usage

```
RunCPCA(
  data = NULL,
  N.clust = NULL,
  num.components = 2,
  integration.algorithm = "globalScore",
  nonzero.coeff.k = "all",
  center.data = FALSE,
  scale.data = FALSE,
  normalization.option = "uniform",
  max.iterations = 1000,
  return.moa.object = TRUE,
  show.verbose = FALSE,
  svd.solver.method = "fast.svd",
  nonzero.coeff.obs = "all",
  weight.variables = NA,
  weight.observations = NA,
  unit.length.variables = FALSE,
  unit.length.observations = FALSE,
  retain.nonnegative = FALSE,
  clustering.algorithm = "ward.D2"
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to create from the hierarchical clustering of the CPCA components (optional but recommended).
num.components	Integer. Number of components to retain for each modality (default: 2).
integration.algorithm	Character. Algorithm to use for CPCA, options include "globalScore" (default), "blockScore", or "blockLoading".
nonzero.coeff.k	Numeric or Character. The number (if >= 1) or proportion (if 0 < k < 1) of non-zero coefficients for variable loadings (default: "all").
center.data	Logical. Whether to center each block to zero mean (default: FALSE).
scale.data	Logical. Whether to scale each block to unit variance (default: FALSE).
normalization.option	Character. Normalization option, one of "lambda1", "inertia", or "uniform" (default: "uniform").
max.iterations	Integer. Maximum number of iterations (default: 1000).
return.moa.object	Logical. Whether to return an object of class ‘moa-class‘ (default: TRUE).
show.verbose	Logical. Whether to print process information (default: FALSE).
svd.solver.method	Character. SVD solver to use, one of "svd", "fast.svd", or "propack" (default: "fast.svd").
nonzero.coeff.obs	Numeric or Character. Number or proportion of non-zero coefficients for observation scores (default: "all").
weight.variables	Numeric, Vector, or List. Weights for variables (default: NA).
weight.observations	Numeric, Vector, or List. Weights for observations (default: NA).
unit.length.variables	Logical. Whether the loading vectors for each block should have unit length (default: FALSE).
unit.length.observations	Logical. Whether the score vectors for each block should have unit length (default: FALSE).
retain.nonnegative	Logical. Whether to retain only non-negative coefficients in loadings and scores (default: FALSE).
clustering.algorithm	Character. The clustering algorithm to use for hierarchical clustering (default: "ward.D2").

Details

This function uses CPCA to integrate multiple data matrices and then performs hierarchical clustering on the resulting components to identify clusters.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'CPCA' to indicate that the clustering was performed using CPCA.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Meng C, Basunia A, Peters B, Gholami AM, Kuster B, Culhane AC. MOGSA: Integrative Single Sample Gene-set Analysis of Multiple Omics Data. Mol Cell Proteomics. 2019;18(8 suppl 1):S153-S168. doi:10.1074/mcp.TIR118.001251.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run CPCA clustering
result <- RunCPCA(data = data_list, N.clust = 3)
```

RunEnsemble *Run Ensemble of Multiple Classifiers for Cluster Prediction*

Description

This function runs an ensemble of different classification models to predict cluster assignments for test data, considering consensus among the models.

Usage

```
RunEnsemble(
  data.test,
  data.train,
  cluster.data,
  cluster.markers,
  surdata = NULL,
  time = "time",
  event = "event",
  methods = NULL,
  sur.trend.rank = NULL,
  cutoff.P = 0.05
)
```

Arguments

data.test	A numeric matrix or data frame of test data. Rows represent genes, and columns represent samples.
data.train	A numeric matrix or data frame of training data. Rows represent genes, and columns represent samples.
cluster.data	A data frame where the first column must be the sample IDs and the second column must be the cluster assignments. The sample IDs must match the column names of the training data.
cluster.markers	A list of data frames, each containing markers for a specific cluster, with columns 'Gene' indicating gene names.
surdata	A data frame containing survival information for the samples. The first column must be sample IDs.
time	A character string specifying the column name in 'surdata' representing survival time (default: "time").
event	A character string specifying the column name in 'surdata' representing the event status (default: "event").
methods	A character vector specifying which classifiers to use in the ensemble. If NULL, all available methods will be used (default: NULL).
sur.trend.rank	A character vector specifying the desired order of survival trends (e.g., c("C2", "C3", "C1")) to filter the models (default: NULL). Note: the order should be from risk to protective.
cutoff.P	Numeric value for the p-value cutoff for survival analysis (default: 0.05).

Details

This function runs an ensemble of classifiers, checks the consistency among classifiers, and optionally performs survival analysis to filter models based on trends in clinical outcomes.

Value

A list containing the ensemble prediction results and optional survival analysis.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data.test <- matrix(rnorm(1000), nrow = 100, ncol = 10)
data.train <- matrix(rnorm(1000), nrow = 100, ncol = 10)
cluster.data <- data.frame(Sample = paste0("Sample", 1:10), Cluster = rep(1:2, each = 5))
cluster.markers <- setNames(lapply(unique(cluster.data$Cluster), function(c) data.frame(Gene = paste0("Gene",
surdata <- data.frame(ID = paste0("Sample", 1:10), time = runif(10, 1, 1000), event = sample(0:1, 10, replace =
result <- RunEnsemble(data.test, data.train, cluster.data, cluster.markers, surdata, time = "time", event = "ev
```

RunGSVA *Generate Single-Sample Gene-Set Enrichment Score*

Description

This function estimates gene-set enrichment scores across all samples using various methods.

Usage

```
RunGSVA(
  exp,
  gene.list,
  min.size = 3,
  max.size = 1000,
  method = "ssgsea",
  ssgsea.normalize = TRUE,
  ssgsea.alpha = 0.25,
  gsva.kcdf = "Gaussian",
  gsva.tau = 1,
  gsva.maxDiff = TRUE,
  gsva.absRanking = FALSE,
  verbose = TRUE,
  nCores = parallel::detectCores() - 3
)
```

Arguments

exp	Numeric matrix containing the expression data or gene expression signatures, with samples in columns and genes in rows.
gene.list	Gene sets provided either as a list object or as a GeneSetCollection object.
min.size	Minimum size of the gene sets to be considered in the analysis. Default is 3.
max.size	Maximum size of the gene sets to be considered in the analysis. Default is 1000.
method	Method to employ in the estimation of gene-set enrichment scores per sample. Options are "gsva" (default), "ssgsea", "zscore", or "plage".
ssgsea.normalize	Logical vector of length 1; if TRUE runs the ssGSEA method from Barbie et al. (2009) normalizing the scores by the absolute difference between the minimum and the maximum, as described in their paper. Otherwise this last normalization step is skipped.
ssgsea.alpha	Numeric vector of length 1. The exponent defining the weight of the tail in the random walk performed by the ssGSEA (Barbie et al., 2009) method. The default value is 0.25 as described in the paper.
gsva.kcdf	Character vector of length 1 denoting the kernel to use during the non-parametric estimation of the cumulative distribution function of expression levels across samples. By default, kcdf="Gaussian" which is suitable when input expression values are continuous, such as microarray fluorescent units in logarithmic scale, RNA-seq log-CPMs, log-RPKMs or log-TPMs. When input expression values are integer counts, such as those derived from RNA-seq experiments, then this argument should be set to kcdf="Poisson".

`gsva.tau` Numeric vector of length 1. The exponent defining the weight of the tail in the random walk performed by the GSVA (Hänzelmann et al., 2013) method. The default value is 1 as described in the paper.

`gsva.maxDiff` Logical vector of length 1 which offers two approaches to calculate the enrichment statistic (ES) from the KS random walk statistic. FALSE: ES is calculated as the maximum distance of the random walk from 0. TRUE (the default): ES is calculated as the magnitude difference between the largest positive and negative random walk deviations.

`gsva.absRanking` Logical vector of length 1 used only when maxDiff=TRUE. When absRanking=FALSE (default) a modified Kuiper statistic is used to calculate enrichment scores, taking the magnitude difference between the largest positive and negative random walk deviations. When absRanking=TRUE the original Kuiper statistic that sums the largest positive and negative random walk deviations, is used. In this latter case, gene sets with genes enriched on either extreme (high or low) will be regarded as 'highly' activated.

`verbose` Logical indicating whether to print progress messages. Default is TRUE.

`nCores` The number of cores to use for parallel computation. Default is 'parallel::detectCores() - 2', which detects the number of cores available on the system and reserves 2 cores for other tasks.

Details

This function supports multiple methods for estimating gene-set enrichment scores, including ssGSEA, GSVA, zscore, and plage. The scores are calculated for each gene set across all samples. The 'ssGSES' function is flexible and allows for customization of the minimum and maximum size of gene sets considered in the analysis. By providing different methods, the function can adapt to various types of gene-set enrichment analysis, each having its own strengths and suitable applications.

- `"gsva"`: Gene Set Variation Analysis, suitable for detecting subtle changes in pathway activity.
- `"ssgsea"`: Single-Sample Gene Set Enrichment Analysis, useful for individual sample analysis.
- `"zscore"`: Z-score transformation, a simpler approach to standardize expression values.
- `"plage"`: Pathway Level Analysis of Gene Expression, which focuses on correlating pathway components.

Value

A gene-set by sample matrix of gene-set enrichment scores.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

`RuniClusterBayes` *Run Bayesian iCluster (iClusterBayes) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Bayesian iCluster (iClusterBayes) to integrate multiple omics datasets. iClusterBayes is an extension of iCluster that allows for Bayesian inference, which is useful for identifying shared patterns across multiple datasets with different distributions.

Usage

```
RuniClusterBayes(
  data = NULL,
  N.clust = NULL,
 data.type = c("binomial", "gaussian", "gaussian", "gaussian", "gaussian", "gaussian"),
  num.burnin = 20,
  num.draws = 10,
  prior.probabilities = rep(0.5, length(data)),
  proposal.sdev = 0.05,
  thinning.interval = 3
)
```

Arguments

`data`	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
`N.clust`	Integer. Number of clusters to create (optional but recommended).
`data.type`	A character vector specifying the type of data for each modality. Options include "binomial", "gaussian", etc. Default is c("binomial", "gaussian", "gaussian", "gaussian", "gaussian", "gaussian").
`num.burnin`	Integer. Number of burn-in iterations for the Bayesian algorithm (default: 18000).
`num.draws`	Integer. Number of MCMC draws after burn-in (default: 12000).
`prior.probabilities`	Numeric vector. Prior values for the inclusion probabilities for each modality (default: 0.5 for each modality).
`proposal.sdev`	Numeric. Standard deviation of the proposal distribution (default: 0.05).
`thinning.interval`	Integer. Thinning interval for MCMC sampling (default: 3).

Details

This function uses iClusterBayes to integrate multiple data matrices and assign clusters to the samples. iClusterBayes performs Bayesian inference, which is useful for modeling different data distributions across multiple datasets.

The function operates as follows: 1. Each matrix in the input list is transposed so that rows represent samples and columns represent features. 2. iClusterBayes is used to identify shared patterns across the different modalities. 3. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'iClusterBayes' to indicate that the clustering was performed using iClusterBayes.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Mo Q, Shen R, et al. (2013) Pattern discovery and cancer gene identification in integrated cancer genomic data. PNAS, 110 (11) 4245-4250. Shen R, Olshen AB, et al. (2012) Integrative Subtype Discovery in Glioblastoma Using iCluster, PLOS ONE 7(4):e35236. Shen R, Olshen AB, et al. (2009) Integrative clustering of multiple genomic data types using a joint latent variable model with application to breast and lung cancer subtype analysis. Bioinformatics, 25(22):2906-12.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run iClusterBayes clustering
result <- RuniClusterBayes(
  data = data_list, N.clust = 3,
  data.type = c("gaussian", "gaussian")
)
```

RunIF *Run Intermediate Fusion (IF) for Multi-Modality Data Integration*

Description

This function runs intermediate fusion (IF) analysis using a specified multi-modality clustering algorithm. Users can choose from a variety of clustering algorithms and adjust their respective parameters to perform data integration on multiple modalities, such as RNA, protein, and methylation.

Usage

```
RunIF(
  data,
  algorithm,
  N.clust,
 data.types = c("binomial", "gaussian", "gaussian", "gaussian", "gaussian", "gaussian"),
  BCC.max.iterations = 10,
  CIMLR.num.dimensions = NA,
  CIMLR.tuning.parameter = 10,
  CIMLR.cores.ratio = 1,
```

```
CPCA.num.components = 2,
CPCA.integration.algorithm = "globalScore",
CPCA.nonzero.coeff.k = "all",
CPCA.center.data = FALSE,
CPCA.scale.data = FALSE,
CPCA.normalization.option = "uniform",
CPCA.max.iterations = 1000,
CPCA.return.moa.object = TRUE,
CPCA.show.verbose = FALSE,
CPCA.svd.solver.method = "fast.svd",
CPCA.nonzero.coeff.obs = "all",
CPCA.weight.variables = NA,
CPCA.weight.observations = NA,
CPCA.unit.length.variables = FALSE,
CPCA.unit.length.observations = FALSE,
CPCA.retain.nonnegative = FALSE,
CPCA.clustering.algorithm = "ward.D2",
iClusterBayes.num.burnin = 20,
iClusterBayes.num.draws = 10,
iClusterBayes.prior.probabilities = rep(0.5, length(data)),
iClusterBayes.proposal.sdev = 0.05,
iClusterBayes.thinning.interval = 3,
IntNMF.max.iterations = 5,
IntNMF.stability.count = 20,
IntNMF.num.initializations = 30,
IntNMF.use.nndsvd = TRUE,
IntNMF.random.seed = TRUE,
IntNMF.weight = NULL,
LRAcluster.cluster.algorithm = "ward.D2",
MCIA.n.components = 10,
MCIA.clustering.algorithm = "ward.D2",
MCIA.scan.eigenvalues = FALSE,
MCIA.use.nsc = TRUE,
MCIA.use.svd = TRUE,
PINSPlus.agreement.cutoff = 0.5,
PINSPlus.num.cores = 10,
PINSPlus.sampled.set.size = 2000,
PINSPlus.knn.k = NULL,
RGCCA.connection.matrix = 1 - diag(length(data)),
RGCCA.num.components = rep(1, length(data)),
RGCCA.scheme = "centroid",
RGCCA.regularization = "optimal",
RGCCA.scale = TRUE,
RGCCA.initialization = "svd",
RGCCA.bias = TRUE,
RGCCA.tolerance = 1e-08,
RGCCA.verbose = FALSE,
RGCCA.clustering.algorithm = "ward.D2",
SGCCA.connection.matrix = 1 - diag(length(data)),
SGCCA.num.components.per.modality = rep(1, length(data)),
SGCCA.integration.scheme = "centroid",
SGCCA.sparsity.level = rep(0.5, length(data)),
```

```
  SGCCA.scale.data = FALSE,
  SGCCA.initialization.method = "svd",
  SGCCA.use.biased.variance = TRUE,
  SGCCA.convergence.tolerance = .Machine$double.eps,
  SGCCA.show.progress = FALSE,
  SGCCA.cluster.algorithm = "ward.D2",
  SNF.num.neighbors = 20,
  SNF.variance = 0.5,
  SNF.num.iterations = 20,
  ...
)
```

Arguments

`data`	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
`algorithm`	Character. The integration algorithm to use. Options include "cpca", "iclusterbayes", "intnmf", "lracluster", "mcia", "nemo", "pinsplus", "rgcca", "sgcca", "snf", "cimlr", "bcc".
`N.clust`	Integer. Number of clusters to create from the hierarchical clustering of the integrated components (optional but recommended).
`data.types`	Character vector. Specifies the type of data for each modality (e.g., "binomial", "gaussian"). Default is a mixture of "binomial" and "gaussian".
`BCC.max.iterations`	Integer. Maximum number of iterations for the BCC algorithm. Default is 10.
`CIMLR.num.dimensions`	Integer. Number of dimensions for CIMLR. Default is NA.
`CIMLR.tuning.parameter`	Numeric. Tuning parameter for CIMLR. Default is 10.
`CIMLR.cores.ratio`	Numeric. Ratio of cores to use for CIMLR. Default is 1.
`CPCA.num.components`	Integer. Number of components for CPCA. Default is 2.
`CPCA.integration.algorithm`	Character. Integration algorithm for CPCA. Default is "globalScore".
`CPCA.nonzero.coeff.k`	Character or numeric. Nonzero coefficient for CPCA. Default is "all".
`CPCA.center.data`	Logical. Whether to center the data for CPCA. Default is FALSE.
`CPCA.scale.data`	Logical. Whether to scale the data for CPCA. Default is FALSE.
`CPCA.normalization.option`	Character. Normalization option for CPCA. Default is "uniform".
`CPCA.max.iterations`	Integer. Maximum number of iterations for CPCA. Default is 1000.
`CPCA.return.moa.object`	Logical. Whether to return MOA object for CPCA. Default is TRUE.
`CPCA.show.verbose`	Logical. Whether to show verbose output for CPCA. Default is FALSE.

`CPCA.svd.solver.method`
: Character. SVD solver method for CPCA. Default is "fast.svd".

`CPCA.nonzero.coeff.obs`
: Character or numeric. Nonzero coefficient for observations in CPCA. Default is "all".

`CPCA.weight.variables`
: Numeric. Weight for variables in CPCA. Default is NA.

`CPCA.weight.observations`
: Numeric. Weight for observations in CPCA. Default is NA.

`CPCA.unit.length.variables`
: Logical. Whether to set unit length for variables in CPCA. Default is FALSE.

`CPCA.unit.length.observations`
: Logical. Whether to set unit length for observations in CPCA. Default is FALSE.

`CPCA.retain.nonnegative`
: Logical. Whether to retain nonnegative values in CPCA. Default is FALSE.

`CPCA.clustering.algorithm`
: Character. Clustering algorithm for CPCA. Default is "ward.D2".

`iClusterBayes.num.burnin`
: Integer. Number of burn-in iterations for iClusterBayes. Default is 20.

`iClusterBayes.num.draws`
: Integer. Number of draws for iClusterBayes. Default is 10.

`iClusterBayes.prior.probabilities`
: Numeric vector. Prior probabilities for iClusterBayes. Default is rep(0.5, length(data)).

`iClusterBayes.proposal.sdev`
: Numeric. Proposal standard deviation for iClusterBayes. Default is 0.05.

`iClusterBayes.thinning.interval`
: Integer. Thinning interval for iClusterBayes. Default is 3.

`IntNMF.max.iterations`
: Integer. Maximum number of iterations for IntNMF. Default is 5.

`IntNMF.stability.count`
: Integer. Stability count for IntNMF. Default is 20.

`IntNMF.num.initializations`
: Integer. Number of initializations for IntNMF. Default is 30.

`IntNMF.use.nndsvd`
: Logical. Whether to use NNDSVD for IntNMF. Default is TRUE.

`IntNMF.random.seed`
: Logical. Whether to use a random seed for IntNMF. Default is TRUE.

`IntNMF.weight` Numeric. Weight for IntNMF. Default is NULL.

`LRAcluster.cluster.algorithm`
: Character. Clustering algorithm for LRAcluster. Default is "ward.D2".

`MCIA.n.components`
: Integer. Number of components for MCIA. Default is 10.

`MCIA.clustering.algorithm`
: Character. Clustering algorithm for MCIA. Default is "ward.D2".

`MCIA.scan.eigenvalues`
: Logical. Whether to scan eigenvalues for MCIA. Default is FALSE.

`MCIA.use.nsc` Logical. Whether to use NSC for MCIA. Default is TRUE.

`MCIA.use.svd` Logical. Whether to use SVD for MCIA. Default is TRUE.

`PINSPlus.agreement.cutoff`
: Numeric. Agreement cutoff for PINSPlus. Default is 0.5.

`PINSPlus.num.cores`
: Integer. Number of cores to use for PINSPlus. Default is 10.

`PINSPlus.sampled.set.size`
: Integer. Sampled set size for PINSPlus. Default is 2000.

`PINSPlus.knn.k`
: Integer. K for k-NN in PINSPlus. Default is NULL.

`RGCCA.connection.matrix`
: Matrix. Connection matrix specifying the relationships between blocks for RGCCA. Default is 1 - diag(length(data)).

`RGCCA.num.components`
: Integer vector. Number of components for each block in RGCCA. Default is rep(1, length(data)).

`RGCCA.scheme`
: Character. Scheme for RGCCA. Default is "centroid".

`RGCCA.regularization`
: Character or numeric vector. Regularization parameter for RGCCA. Default is "optimal".

`RGCCA.scale`
: Logical. Whether to scale data for RGCCA. Default is TRUE.

`RGCCA.initialization`
: Character. Initialization method for RGCCA. Default is "svd".

`RGCCA.bias`
: Logical. Whether to use a biased estimator for RGCCA. Default is TRUE.

`RGCCA.tolerance`
: Numeric. Convergence tolerance for RGCCA. Default is 1e-08.

`RGCCA.verbose`
: Logical. Whether to show progress messages for RGCCA. Default is FALSE.

`RGCCA.clustering.algorithm`
: Character. Clustering algorithm for RGCCA. Default is "ward.D2".

`SGCCA.connection.matrix`
: Matrix. Connection matrix specifying the relationships between blocks for SGCCA. Default is 1 - diag(length(data)).

`SGCCA.num.components.per.modality`
: Integer vector. Number of components per modality for SGCCA. Default is rep(1, length(data)).

`SGCCA.integration.scheme`
: Character. Integration scheme for SGCCA. Default is "centroid".

`SGCCA.sparsity.level`
: Numeric vector. Sparsity level for each block in SGCCA. Default is rep(0.5, length(data)).

`SGCCA.scale.data`
: Logical. Whether to scale data for SGCCA. Default is FALSE.

`SGCCA.initialization.method`
: Character. Initialization method for SGCCA. Default is "svd".

`SGCCA.use.biased.variance`
: Logical. Whether to use a biased estimator for SGCCA. Default is TRUE.

`SGCCA.convergence.tolerance`
: Numeric. Convergence tolerance for SGCCA. Default is .Machine$double.eps.

`SGCCA.show.progress`
: Logical. Whether to show progress messages for SGCCA. Default is FALSE.

`SGCCA.cluster.algorithm`
: Character. Clustering algorithm for SGCCA. Default is "ward.D2".

`SNF.num.neighbors`
: Integer. Number of neighbors for SNF. Default is 20.

`SNF.variance` Numeric. Variance for SNF. Default is 0.5.

`SNF.num.iterations`
: Integer. Number of iterations for SNF. Default is 20.

Details

This function allows the user to integrate multiple data modalities using a variety of different algorithms. Each algorithm has its own parameters that can be adjusted to fit the data and the research question. The function returns clustering results for each sample based on the selected algorithm.

Value

A list containing the clustering results based on the selected algorithm.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run integration clustering using CPCA
result <- RunIF(data = data_list, algorithm = "cpca", N.clust = 3)
```

RunIntNMF *Run Integrative Non-negative Matrix Factorization (IntNMF) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Integrative Non-negative Matrix Factorization (IntNMF) to integrate multiple omics datasets. IntNMF is a powerful tool for capturing shared patterns across multiple datasets by decomposing them into components that reflect shared and individual structures.

Usage

```
RunIntNMF(
  data = NULL,
  N.clust = NULL,
  max.iterations = 5,
  stability.count = 20,
  num.initializations = 30,
```

```
  use.nndsvd = TRUE,
  random.seed = TRUE,
  weight = NULL
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to create from the IntNMF components (optional but recommended).
max.iterations	Integer. Maximum number of iterations for the NMF algorithm (default: 200).
stability.count	Integer. Count for stability in connectivity matrix (default: 20).
num.initializations	Integer. Number of initializations of the random matrices (default: 30).
use.nndsvd	Logical. Whether to use non-negative double singular value decomposition (NNDSVD) for initialization (default: TRUE).
random.seed	Logical. Whether to use a random seed for initialization of the algorithm (default: TRUE).
weight	Numeric vector. Weight for each data matrix in the list (default: equal weights).

Details

This function uses IntNMF to integrate multiple data matrices and then assigns clusters to the samples based on the resulting components. IntNMF is particularly useful for capturing shared and individual variation in multi-omics data.

The function operates as follows: 1. Each data matrix is normalized to ensure non-negative values and scaled to unit variance. 2. IntNMF is performed using the IntNMF package to extract components that summarize the shared variation across different modalities. 3. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'IntNMF' to indicate that the clustering was performed using IntNMF.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Chalise P, Fridley BL. Integrative clustering of multi-level 'omic data based on non-negative matrix factorization algorithm. PLoS One. 2017;12(5):e0176278. doi:10.1371/journal.pone.0176278. Chalise P, Raghavan R and Fridley B (2016). InterSIM: Simulation tool for multiple integrative 'omic datasets. Computer Methods and Programs in Biomedicine, 128:69-74.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run IntNMF clustering
result <- RunIntNMF(data = data_list, N.clust = 3)
```

RunLRAcluster *Run Low-Rank Approximation Clustering (LRAcluster) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Low-Rank Approximation Clustering (LRAcluster) to integrate multiple omics datasets. LRAcluster helps reduce the dimensionality of multiple data types while retaining key structures, facilitating effective clustering.

Usage

```
RunLRAcluster(
  data = NULL,
  N.clust = NULL,
 data.types = list("binary", "gaussian", "gaussian", "gaussian", "gaussian", "gaussian"),
  data.names = NULL,
  cluster.algorithm = "ward.D2"
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to create from the hierarchical clustering of the LRAcluster components (optional but recommended).
data.types	A list specifying the type of data for each modality. Options include "binary", "gaussian", or "poisson". Default is c("binary", "gaussian").
data.names	Character vector. Names of the datasets (optional).
cluster.algorithm	Character. The clustering algorithm to use for hierarchical clustering (default: "ward.D2").

Details

This function uses LRAcluster to integrate multiple data matrices and then performs hierarchical clustering on the resulting components to identify clusters. LRAcluster is particularly useful for reducing the dimensionality of high-dimensional omics data while retaining the most informative features.

The function operates as follows: 1. Each matrix in the input list is converted to a matrix to ensure compatibility. 2. LRAcluster is performed to extract components that summarize the shared variation across different modalities. 3. Hierarchical clustering is applied to the concatenated components to assign each sample to a cluster. 4. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'LRAcluster' to indicate that the clustering was performed using LRAcluster.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Wu D, Wang D, Zhang MQ, Gu J. Fast dimension reduction and integrative clustering of multi-omics data using low-rank approximation: application to cancer molecular classification. BMC Genomics. 2015;16:1022. doi:10.1186/s12864-015-2223-8.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run LRAcluster clustering
result <- RunLRAcluster(
  data = data_list, N.clust = 3,
  data.types = c("gaussian", "gaussian")
)
```

`RunMCIA` *Run Multiple Co-Inertia Analysis (MCIA) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Multiple Co-Inertia Analysis (MCIA) to integrate multiple omics datasets. MCIA helps capture shared variation across different data modalities and provides insights into the common structures of the data.

Usage

```
RunMCIA(
  data = NULL,
  N.clust = NULL,
  n.components = 10,
  clustering.algorithm = "ward.D2",
  scan.eigenvalues = FALSE,
  use.nsc = TRUE,
  use.svd = TRUE
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to create from the hierarchical clustering of the MCIA components (optional but recommended).
n.components	Integer. Number of components to retain for each modality (default: 10).
clustering.algorithm	Character. The clustering algorithm to use for hierarchical clustering (default: "ward.D2").
scan.eigenvalues	Logical. Whether to show the co-inertia analysis eigenvalue plot to help select the number of axes (default: FALSE).
use.nsc	Logical. Whether to perform multiple non-symmetric correspondence analyses. Recommended to keep TRUE (default: TRUE).
use.svd	Logical. Whether to use singular value decomposition to perform the analysis (default: TRUE).

Details

This function uses MCIA to integrate multiple data matrices and then performs hierarchical clustering on the resulting components to identify clusters. MCIA is useful for identifying shared structures across multiple data modalities.

The function operates as follows: 1. Each matrix in the input list is transposed so that rows represent samples and columns represent features. 2. MCIA is performed using the omicade4 package to extract components that summarize the shared variation across different modalities. 3. Hierarchical clustering is applied to the concatenated components to assign each sample to a cluster. 4. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'MCIA' to indicate that the clustering was performed using MCIA.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Meng C, Kuster B, Culhane A, Gholami AM. A multivariate approach to the integration of multi-omics datasets. BMC Bioinformatics. 2013.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run MCIA clustering
result <- RunMCIA(data = data_list, N.clust = 3)
```

RunMOFS *Run MultiModality Fusion Subtyping (MOFS) for Multi-Modality Data Integration*

Description

This function performs MultiModality Fusion Subtyping (MOFS) analysis by utilizing multiple clustering algorithms for multi-modality data integration. The user can flexibly select the desired clustering algorithms and adjust relevant parameters.

Usage

```
RunMOFS(
  data,
  methods,
  max.clusters = 6,
  optimal.clusters = 3,
  linkage.method = "ward.D2",
  clustering.algorithm = "hc",
  distance.metric = "euclidean",
  resampling.iterations = 10000,
  resample.proportion = 0.7,
  silhouette.cutoff = 0.4,
  ...
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
methods	Character vector. The clustering algorithms to use. Options are: "CPCA", "iClusterBayes", "IntNMF", "LRAcluster", "MCIA", "NEMO", "PINSPlus", "RGCCA", "SGCCA", "SNF", "CIMLR", "BCC". At least two methods must be specified.

`max.clusters` Integer. The maximum number of clusters to evaluate during consensus clustering analysis (default: 6).

`optimal.clusters` Integer. The optimal number of clusters to select from the consensus clustering analysis (default: 3).

`linkage.method` Character. The linkage method to use for hierarchical clustering (default: "ward.D2").

`clustering.algorithm` Character. The clustering algorithm to use during consensus clustering (default: 'hc').

`distance.metric` Character. The distance metric to use for clustering (default: "euclidean").

`resampling.iterations` Integer. The number of resampling iterations for consensus clustering (default: 10000).

`resample.proportion` Numeric. The proportion of items to resample in each iteration for consensus clustering (default: 0.7).

`silhouette.cutoff` Numeric. Silhouette coefficient cutoff value for selecting core set samples (default: 0.4).

`...` Additional parameters specific to the chosen clustering algorithms.

Details

The function performs MultiModality Fusion Subtyping (MOFS) by running multiple clustering algorithms on the input multi-modality data. The results of each clustering algorithm are stored for further analysis, including binary cluster assignments, Jaccard distance calculation, consensus clustering analysis, Calinski-Harabasz index calculation, silhouette analysis, and PCA.

The steps involved are: 1. Running the specified clustering algorithms on the input data. 2. Extracting binary cluster assignments from the clustering results. 3. Calculating Jaccard distance between clusters. 4. Performing consensus clustering analysis to identify stable clusters. 5. Calculating the Calinski-Harabasz index to assess clustering quality. 6. Performing silhouette analysis to evaluate cluster cohesion and separation. 7. Identifying a core set of samples based on the silhouette coefficient cutoff. 8. Performing PCA on the core set for dimensionality reduction and visualization.

Value

A list containing the results for each specified clustering algorithm, as well as the results of further analysis including consensus clustering, silhouette scores, core set identification, and PCA.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)
```

```
# Run MultiModality Fusion Subtyping using CPCA and CIMLR
result <- RunMOFS(data = data_list, methods = c("CPCA", "CIMLR"), max.clusters = 6, optimal.clusters = 3, linkag
```

RunNEMO *Run NEMO for Multi-Modality Data Clustering*

Description

This function performs clustering analysis using the NEMO method, which is suitable for identifying subtypes in multi-omics data. NEMO allows integration of multiple modalities to discover coherent subtypes by partial data integration.

Usage

```
RunNEMO(data, N.clust = NULL)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to evaluate during subtyping. If NULL, NEMO will determine the optimal number of clusters automatically. Default is NULL.

Details

The function operates as follows: 1. Uses the NEMO package to identify subtypes across the provided omics modalities. 2. Returns a data frame containing the cluster assignment for each sample.

Value

A data frame with the following columns: - Sample: The sample identifier, taken from the column names of the input data matrices. - Cluster: The assigned cluster number for each sample, indicating the subtype classification. - Cluster2: The assigned cluster label, prefixed by 'NEMO' to indicate that the clustering was performed using NEMO.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Rappoport N, Shamir R. NEMO: cancer subtyping by integration of partial multi-omic data. Bioinformatics. 2019;35(18):3348-3356. doi:10.1093/bioinformatics/btz058

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run NEMO clustering
result <- RunNEMO(data = data_list, N.clust = 3)
```

RunPCA *Run PCA on Data*

Description

This function performs Principal Component Analysis (PCA) on the given data using the FactoMineR package.

Usage

```
RunPCA(data)
```

Arguments

data	A distance matrix or data frame on which to perform PCA.

Details

This function uses the FactoMineR package to perform PCA, which can be used to visualize the relationships between samples in reduced dimensional space.

Value

The PCA result as produced by the FactoMineR package.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

Examples

```
# Example usage:
data <- matrix(rnorm(100), nrow = 10, ncol = 10)
pca_result <- RunPCA(data)
```

RunPINSPlus *Run PINSPlus for Multi-Modality Data Clustering*

Description

This function performs clustering analysis using the PINSPlus method, which is suitable for identifying subtypes in multi-omics data. PINSPlus allows integration of multiple modalities to discover coherent subtypes.

Usage

```
RunPINSPlus(
  data,
  N.clust = NULL,
  agreement.cutoff = 0.5,
  num.cores = 10,
  sampled.set.size = 2000,
  knn.k = NULL
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples. All matrices should have the same number of samples.
N.clust	Integer. Maximum number of clusters to evaluate during subtyping. If NULL, PINSPlus will determine the optimal number of clusters automatically. Default is NULL.
agreement.cutoff	Numeric. Agreement threshold to be considered consistent (default: 0.5).
num.cores	Integer. Number of cores for parallel processing (default: 10). Setting a higher value can speed up the computation.
sampled.set.size	Integer. The number of sample size used for the sampling process when the dataset is large (default: 2000).
knn.k	Integer. The value of k for the k-nearest neighbors algorithm. If not set, elbow method will be used to calculate k (default: NULL).

Details

The function operates as follows: 1. Transposes each data matrix so that rows represent samples and columns represent features. 2. Uses the PINSPlus package to identify subtypes across the modalities. 3. Returns a data frame containing the cluster assignment for each sample.

Value

A data frame with the following columns: - Sample: The sample identifier, taken from the column names of the input data matrices. - Cluster: The assigned cluster number for each sample, indicating the subtype classification. - Cluster2: The assigned cluster label, prefixed by 'PINSPlus' to indicate that the clustering was performed using PINSPlus.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Nguyen H, Shrestha S, Draghici S, Nguyen T. PINSPlus: a tool for tumor subtype discovery in integrated genomic data. Bioinformatics. 2019;35(16):2843-2846. doi:10.1093/bioinformatics/bty1049 Nguyen T, Tagett R, Diaz D, Draghici S. A novel method for data integration and disease subtyping. Genome Research. 2017;27(12):2025-2039. Nguyen T. Horizontal and vertical integration of bio-molecular data. PhD thesis, Wayne State University. 2017. Nguyen H, Tran D, Tran B, Roy M, Cassell A, Dascalu S, Draghici S, Nguyen T. SMRT: Randomized Data Transformation for Cancer Subtyping and Big Data Analysis. Frontiers in Oncology. 2021.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run PINSPlus clustering
result <- RunPINSPlus(data = data_list, N.clust = 3)
```

RunRGCCA *Run Regularized Generalized Canonical Correlation Analysis (RGCCA) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Regularized Generalized Canonical Correlation Analysis (RGCCA), which integrates multiple data modalities (e.g., RNA, protein, methylation) to provide a unified clustering result. RGCCA helps in capturing shared information between multiple data types and provides insights into the relationships between different modalities.

Usage

```
RunRGCCA(
  data,
  N.clust = NULL,
  connection.matrix = 1 - diag(length(data)),
  num.components = rep(1, length(data)),
  scheme = "centroid",
  regularization = "optimal",
  scale = TRUE,
  initialization = "svd",
  bias = TRUE,
  tolerance = 1e-08,
  verbose = FALSE,
  clustering.algorithm = "ward.D2"
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters to create from the hierarchical clustering of the RGCCA components (optional but recommended).
connection.matrix	Matrix. Connection matrix specifying the relationships between blocks (default: 1 - diagonal matrix of block length).
num.components	Integer vector. Number of components to compute for each block (default: 1 component per block).
scheme	Character. The RGCCA scheme to use, can be one of "centroid", "factorial", or "horst" (default: "centroid").
regularization	Character or numeric vector. Regularization parameter for each block, can be "optimal" or a numeric value (default: "optimal").
scale	Logical. Whether to scale each block to zero mean and unit variance (default: TRUE).
initialization	Character. Initialization method for the RGCCA algorithm, either "svd" or "random" (default: "svd").
bias	Logical. Whether to use biased or unbiased estimator of the variance/covariance (default: TRUE).
tolerance	Numeric. Convergence tolerance (default: 1e-08).
verbose	Logical. Whether to show progress messages (default: FALSE).
clustering.algorithm	Character. The clustering algorithm to use for hierarchical clustering (default: "ward.D2").
max.iterations	Integer. Maximum number of iterations for the RGCCA algorithm (default: 1000).

Details

This function uses RGCCA to integrate multiple data matrices and then performs hierarchical clustering on the resulting components to identify clusters. RGCCA is particularly useful for identifying shared structures across multiple modalities, providing a comprehensive view of the relationships between different data types.

The function operates as follows: 1. Each matrix in the input list is transposed so that rows represent samples and columns represent features. 2. RGCCA is performed to extract components that summarize the shared variation across different modalities. 3. Hierarchical clustering is applied to the concatenated components to assign each sample to a cluster. 4. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'RGCCA' to indicate that the clustering was performed using RGCCA.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Tenenhaus, M., Tenenhaus, A., & Groenen, P. J. (2017). Regularized generalized canonical correlation analysis: a framework for sequential multiblock component methods. Psychometrika, 82(3), 737-777. Tenenhaus, A., Philippe, C., & Frouin, V. (2015). Kernel generalized canonical correlation analysis. Computational Statistics & Data Analysis, 90, 114-131.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run RGCCA clustering
result <- RunRGCCA(data = data.list, N.clust = 3)
```

RunSGCCA *Run Sparse Generalized Canonical Correlation Analysis (SGCCA) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Sparse Generalized Canonical Correlation Analysis (SGCCA). SGCCA is an extension of Generalized Canonical Correlation Analysis that allows for sparse estimation, which is beneficial when working with high-dimensional datasets, making it effective for integrating multiple modalities (e.g., RNA, protein).

Usage

```
RunSGCCA(
  data,
  N.clust = NULL,
  connection.matrix = 1 - diag(length(data)),
  num.components.per.modality = rep(1, length(data)),
  integration.scheme = "centroid",
  sparsity.level = rep(0.5, length(data)),
  scale.data = FALSE,
  initialization.method = "svd",
  use.biased.variance = TRUE,
  convergence.tolerance = .Machine$double.eps,
  show.progress = FALSE,
  cluster.algorithm = "ward.D2"
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.
N.clust	Integer. Number of clusters for hierarchical clustering (optional but recommended).
connection.matrix	Matrix. A matrix describing the relationships between different modalities (default: complete design matrix).
num.components.per.modality	Integer vector specifying the number of components to compute for each modality (default: 1 component per modality).
integration.scheme	Character. The method used for integrating different data modalities. Options are "centroid", "horst", or "factorial" (default: "centroid").
sparsity.level	Numeric vector specifying the regularization (sparsity) parameters for each modality, controlling the sparsity level (default: 0.5 for each modality).
scale.data	Logical. Whether to scale each block to zero mean and unit variance (default: FALSE).
initialization.method	Character. Method for initializing the SGCCA algorithm, either "svd" or "random" (default: "svd").
use.biased.variance	Logical. Whether to use a biased estimator for the variance/covariance (default: TRUE).
convergence.tolerance	Numeric. The convergence tolerance for the algorithm (default: .Machine$double.eps).
show.progress	Logical. Whether to display progress messages during the execution (default: FALSE).
cluster.algorithm	Character. The method used for hierarchical clustering (default: "ward.D2").

Details

The function proceeds as follows: 1. Each matrix in the input list is transposed so that rows represent samples and columns represent features. 2. SGCCA is applied to find the canonical components for each modality. 3. Combines the canonical components and applies hierarchical clustering to identify clusters. 4. Returns a data frame with the assigned clusters for each sample.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'SGCCA' to indicate that the clustering was performed using SGCCA.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Tenenhaus, M., Tenenhaus, A., & Groenen, P. J. (2017). Regularized generalized canonical correlation analysis: a framework for sequential multiblock component methods. Psychometrika, 82(3), 737-777. Tenenhaus, A., Philippe, C., & Frouin, V. (2015). Kernel generalized canonical correlation analysis. Computational Statistics & Data Analysis, 90, 114-131. Tenenhaus, A., Philippe, C., Guillemot, V., Le Cao, K. A., Grill, J., & Frouin, V. (2014). Variable selection for generalized canonical correlation analysis. Biostatistics, 15(3), 569-583. Tenenhaus, A., & Tenenhaus, M. (2011). Regularized generalized canonical correlation analysis. Psychometrika, 76(2), 257. Van de Geer, J. P. (1984). Linear relations among K sets of variables. Psychometrika, 49(1), 79-94. Schafer J. and Strimmer K. (2005). A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. Statistical Applications in Genetics and Molecular Biology 4:32. Tenenhaus et al. Variable Selection For Generalized Canonical Correlation Analysis. 2013. Submitted to Biostatistics.

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run SGCCA clustering
result <- RunSGCCA(data = data_list, N.clust = 3)
```

RunSNF *Run Similarity Network Fusion (SNF) for Multi-Modality Data Integration*

Description

This function performs clustering analysis using Similarity Network Fusion (SNF), which integrates multiple data types (e.g., different modalities such as RNA, protein, methylation) to provide a unified clustering result. SNF is an effective technique for capturing complementary information from multiple modalities and determining common subtypes.

Usage

```
RunSNF(
  data,
  N.clust = NULL,
  num.neighbors = 20,
  variance = 0.5,
  num.iterations = 20
)
```

Arguments

data	A list of matrices where each element represents a different modality (e.g., RNA, protein, methylation). Each matrix should have rows as features and columns as samples.

N.clust	Integer. Number of clusters for spectral clustering. This is the desired number of groups to partition the samples into (optional but recommended).
num.neighbors	Integer. Number of nearest neighbors to consider in building the affinity matrix (default: 20). This parameter controls the local neighborhood size. It typically takes values between 10 and 30.
variance	Numeric. The variance for the local model in building the affinity matrix (default: 0.5). This value affects the distance weighting between samples. It typically ranges from 0.3 to 0.8.
num.iterations	Integer. Number of iterations for the similarity network fusion process (default: 20). This parameter controls how the affinity matrices from different data types are fused. Typically, 10 to 20 iterations are used.

Details

This function uses the Similarity Network Fusion (SNF) approach to integrate multiple data matrices and then performs spectral clustering on the fused network to identify clusters. SNF is particularly useful when dealing with multi-modality datasets, as it takes into account the complementary nature of different data types to improve clustering robustness and biological interpretability.

The function operates as follows: 1. Each matrix in the input list is transposed so that rows represent samples and columns represent features. 2. For each data type, an affinity matrix is computed using the distance between samples and then transformed using an exponential kernel with parameters 'num.neighbors' and 'variance'. 3. The affinity matrices are fused using the SNF approach over 'num.iterations' iterations to create a consensus similarity network. 4. Spectral clustering is then applied to the fused affinity matrix to assign each sample to a cluster. 5. The function returns a data frame containing the cluster assignment for each sample, along with additional information about the clustering process.

Value

A data frame with the following columns: - Sample: The sample identifier. - Cluster: The assigned cluster number for each sample. - Cluster2: The assigned cluster label, prefixed by 'SNF' to indicate that the clustering was performed using SNF.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Wang B, Mezlini AM, Demir F, Fiume M, Tu Z, Brudno M, Haibe-Kains B, Goldenberg A. Similarity Network Fusion for Aggregating Data Types on a Genomic Scale. Nat Methods. 2014;11(3):333-337. doi:10.1038/nmeth.2810

Examples

```
# Example usage:
data1 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
data2 <- matrix(rnorm(10000), nrow = 100, ncol = 100)
colnames(data1) <- colnames(data2) <- paste0("Sample", 1:100)
data_list <- list(data1, data2)

# Run SNF clustering
result <- RunSNF(data = data_list, N.clust = 3)
```

SD.df *Calculate Standard Deviation for a Data Frame*

Description

This function calculates the standard deviation (SD) for each column in a data frame.

Usage

```
SD.df(df)
```

Arguments

df	A data frame with row samples and column features.

Value

A sorted vector of SD values for each column in the data frame, in decreasing order.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

Select.Features *Select Hypervariable Features*

Description

This function selects hypervariable features from a data frame based on specified variance calculation methods.

Usage

```
Select.Features(
  data,
  method = "mad",
  top.percent = NULL,
  top.number = 1000,
  custom.features = NULL
)
```

Arguments

data	A data frame with row features and column samples.
method	Method of calculating variance ("sd", "mad", "cv").
top.percent	The top percent of hypervariable features based on variance.
top.number	The top number of hypervariable features based on variance.
custom.features	A vector of custom features to select.

Value

A vector of selected hypervariable features.

Author(s)

Zaoqu Liu; E-mail: liuzaoqu@163.com

Examples

```
## Not run:
df <- data.frame(matrix(rnorm(1000), nrow = 100, ncol = 10))
selected_features <- Select.Features(df, method = "mad", top.percent = 10)

## End(Not run)
```

---

ssMwwGST *Single-Sample Pathway Activity Analysis Using ssMwwGST*

Description

This function performs single-sample pathway activity analysis using the MWW-GST (Mann-Whitney-Wilcoxon Gene Set Test) method. It assesses pathway activity for each individual sample, leveraging the yaGST package to estimate pathway scores for each gene set.

Usage

```
ssMwwGST(geData, geneSet, nCores = 8)
```

Arguments

geData	A numeric matrix of gene expression data. Rows represent genes, and columns represent samples.
geneSet	A list of gene sets where each element is a vector of gene names included in the set. Each gene set should have at least 15 genes.
nCores	Integer. The number of cores to use for parallel processing. Default is 8.

Details

The function operates as follows: 1. Calculates the mean and standard deviation for each gene across samples. 2. For each sample, the gene expression is standardized (z-score normalization). 3. Uses the Mann-Whitney-Wilcoxon Gene Set Test (MWW-GST) to assess pathway enrichment for each gene set in each sample. 4. Performs multiple testing correction (FDR) for p-values.

Value

A list containing: - NES: A numeric matrix of Normalized Enrichment Scores (NES) for each gene set and each sample. - pValue: A numeric matrix of p-values for the enrichment of each gene set in each sample. - FDR: A numeric matrix of False Discovery Rate (FDR) adjusted p-values for each gene set in each sample.

Author(s)

Zaoqu Liu; Email: liuzaoqu@163.com

References

Frattini V, Pagnotta SM, Tala, et al. A metabolic function of FGFR3-TACC3 gene fusions in cancer. Nature. 2018;553(7687):222-227. doi:10.1038/nature25171

Examples

```
# Example usage:
geData <- matrix(rnorm(1000), nrow = 100, ncol = 10) # Generate random gene expression data
geneSet <- list(set1 = sample(rownames(geData), 20), set2 = sample(rownames(geData), 15))
# Run ssMwwGST pathway activity analysis
result <- ssMwwGST(geData, geneSet, nCores = 4)
```

Index