Package: cmAnalysis (via r-universe)

March 14, 2025

Title Process and Visualise Concept Mapping Data

Version 1.0.0

Author Jos Hageman [aut, cre], Jarl Kampen [aut]

Maintainer Jos Hageman <jos.hageman@wur.nl>

Description Processing and visualizing concept mapping data. Concept maps are versatile tools used across disciplines to enhance understanding, teaching, brainstorming, and information organization. The analysis of concept mapping data involves the sequential use of cluster analysis (for sorting participants and statements), multidimensional scaling (for positioning statements in a conceptual space), and visualization techniques, including point cluster maps and dendrograms.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Imports cluster, ggplot2, factoextra, clue, igraph, stringr, pheatmap

NeedsCompilation no

Date/Publication 2025-03-13 12:50:05 UTC

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libxml2-dev zlib1g-dev

Repository https://joshageman.r-universe.dev

RemoteUrl https://github.com/cran/cmAnalysis

RemoteRef HEAD

RemoteSha 13c198f353d1876336a51ad75ce6948b944fb353

Contents

`abbreviateStatements` *Abbreviate Statements in Concept Map Data*

Description

This function processes the "statement" column of a data frame containing concept map data by converting each statement to lowercase, removing stopwords, and truncating the statement to a specified maximum length. It allows for the abbreviation of long statements while maintaining their core meaning by removing unnecessary words.

Usage

```
abbreviateStatements(CMData, max_length = 30)
```

Arguments

`CMData`	A data frame containing concept map data. The data should have at least the following column: • `statement`: The text of the statement to be abbreviated.
`max_length`	An integer specifying the maximum number of characters for the abbreviated statement. Default is 30.

Details

This function performs several preprocessing steps on the "statement" column:

- Converts statements to lowercase.
- Removes punctuation and stopwords from the statements.
- Truncates statements to a specified maximum length.
- Removes any rows with empty statements after processing.

Stopwords are predefined and include common English words (e.g., "the", "and", "is", "a", etc.) that do not contribute much meaning to the core idea of the statement.

Value

A data frame with the same structure as the input, but with an updated "statement" column containing the abbreviated statements.

Examples

```
# Create a sample data frame with concept map data
conceptMapData <- data.frame(
   id = c(1, 2, 3),
    statement = c(
       "The quick brown fox jumps over the lazy dog",
       "This is a simple concept map example",
       "Data science involves analyzing datasets"
    )
)

# Apply the abbreviateStatements function with a maximum length of 20
result <- abbreviateStatements(conceptMapData, max_length = 20)

print(result)
```

---

checkConceptMapData *Validate Concept Map Data*

Description

Checks whether the provided data frame meets the requirements for concept map data.

Usage

```
checkConceptMapData(CMData)
```

Arguments

CMData	A data frame containing concept map data. This data frame must include the required columns: "sorterID", "statement", and "stackID".

Details

This function performs the following checks on the input data:

- Verifies that `CMData` is a data frame.
- Ensures the presence of required columns: `"sorterID"`, `"statement"`, and `"stackID"`.
- Confirms that there are at least 2 unique values in the `"stackID"` column.
- Confirms that there are at least 2 unique values in the `"sorterID"` column.
- Confirms that there are at least 2 unique values in the `"statement"` column.

Value

Returns TRUE if all checks pass. If any check fails, an error is raised with a descriptive message.

Examples

```
# Example of valid data
validData <- data.frame(
  sorterID = c("resp1", "resp1", "resp1", "resp2",
   "resp2", "resp2", "resp3", "resp3", "resp3"),
  statement = c("London", "Frankfurt", "Berlin", "London",
  "Frankfurt", "Berlin", "London", "Frankfurt", "Berlin"),
  stackID = c("capital city", "city", "capital city", 1, 2, 2, "A", "B", "A")
)
checkConceptMapData(validData) # Should return TRUE

# Example of invalid data (missing columns)
invalidData <- data.frame(
  sorterID = c(1, 2),
  stackID = c(1, 2)
)
# checkConceptMapData(invalidData) # Will return False as an error
```

---

conceptMapping *Perform Concept Mapping Analysis*

Description

This function conducts a concept mapping analysis on sorter data, producing a concept map based on one of three clustering methods: k-means, network analysis, or classical multidimensional scaling (CMDS).

Usage

```
conceptMapping(
  CMData,
  method = "network",
  numberOfClusters = "auto",
  verbose = TRUE,
  rangeNumberOfClusters = 2:15,
  graph = FALSE,
  numberOfKmeansRestarts = 100,
  backgroundColor = "black",
  main = NULL,
  ...
)
```

Arguments

CMData	A data frame containing concept mapping data. It must include the columns: "sorterID", "statement", and "stackID".
method	A string specifying the clustering method to use. Options are: "kmeans", "network", or "cmds". Default is "network".
numberOfClusters	Either a character string ("auto") to determine the optimal number of clusters or an integer specifying the desired number of clusters. Default is "auto".
verbose	Logical, if TRUE, additional information about the processing steps is printed to the console.
rangeNumberOfClusters	A vector of integers specifying the range of clusters to evaluate when numberOfClusters = "auto". Default is 2:15.
graph	Logical. If TRUE, visualizes clustering results, including heatmaps and cluster-specific plots. Default is FALSE.
numberOfKmeansRestarts	An integer specifying the number of restarts for k-means clustering. Only relevant if method = "kmeans". Default is 100.
backgroundColor	A string specifying the background color of network plots. Default is "black".
main	A string specifying the title for plots. Default is NULL.
...	Additional arguments, such as resolution, which may be passed to specific clustering methods.

Details

The function supports three methods for clustering:

- "kmeans": Uses k-means clustering with an optional silhouette-based determination of cluster count.
- "network": Generates a network plot using modularity-based clustering.
- "cmds": Applies classical multidimensional scaling (CMDS) and clusters the results.

Heatmaps are created for all methods, while additional visualizations depend on the chosen method and graph parameter.

Value

An object of class conceptMap, containing:

allStatements A data frame with statement numbers and text.

CMData The original concept mapping data.

method The clustering method used.

numberOfClusters The number of clusters identified.

clusterResults A vector indicating cluster assignments for each statement.

`heatmapPlot` A heatmap visualizing co-occurrence patterns.

`silhouettePlot` (If applicable) A silhouette plot for `"kmeans"` or `"cmds"`.

`networkPlot` (If applicable) A network plot for `"network"`.

`cmdsPlot` (If applicable) A CMDS plot for `"cmds"`.

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Concept mapping on sorter cluster 3 using default "network" method
myCMAnalysis3 <- conceptMapping(myCMDataBySorters[[3]])

# Concept mapping using default network method using 3 clusters
myCMAnalysis3b <- conceptMapping(myCMDataBySorters[[3]], numberOfCluster = 3)

# Concept mapping using kmeans clustering and 3 clusters
myCMAnalysis3c <- conceptMapping(myCMDataBySorters[[3]], method = "kmeans",
  numberOfCluster = 3)
```

---

createStatementOverview

Create an Overview of Statements in Concept Map Data

Description

This function generates an overview of all unique statements in a given concept mapping dataset. The function assigns a unique statement number to each distinct statement and returns a summary data frame.

Usage

```
createStatementOverview(CMData)
```

Arguments

`CMData`	A data frame containing concept map data. This must include a column named `"statement"`.

Details

The function first checks if the provided dataset is suitable for concept mapping using the `checkConceptMapData` function. If the data is valid, it assigns a numeric statement number to each distinct statement, ordered by their appearance in the dataset. The function then generates an overview where each statement is paired with its corresponding statement number.

Value

A data frame with two columns: `StatementNumber` and `Statement`. Each row represents a unique statement with its corresponding number.

Examples

```
# Example of valid data
CMData <- data.frame(
    sorterID = c("resp1", "resp1", "resp1", "resp2",
     "resp2", "resp2", "resp3", "resp3", "resp3"),
   statement = c("London", "Frankfurt", "Berlin", "London",
    "Frankfurt", "Berlin", "London", "Frankfurt", "Berlin"),
    stackID = c("capital city", "city", "capital city", 1, 2, 2, "A", "B", "A")
)

# Create an overview of the statements
createStatementOverview(CMData)
```

---

crossClusterMap *Cross-Cluster Mapping Between Concept Maps*

Description

This function compares two concept maps by aligning their clustering results and visualizing the correspondence between clusters. It identifies matches between clusters from the two maps and highlights differences visually.

Usage

```
crossClusterMap(conceptMap1, conceptMap2)
```

Arguments

`conceptMap1`	An object of class `"conceptMap"` representing the first concept map.
`conceptMap2`	An object of class `"conceptMap"` representing the second concept map.

Details

The function aligns clusters between two concept maps using an optimal matching algorithm. It first creates a matching matrix based on the overlap between clusters in the two maps. Then, it uses the Hungarian algorithm (via the `solve_LSAP` function from the `clue` package) to find an optimal alignment of clusters.

The output is a plot that shows the alignment of clusters from the two concept maps, with connecting lines colored to indicate matches or mismatches. Statements not clustered in both maps are highlighted in grey.

Value

The function does not return a value but generates a `ggplot2` visualization.

See Also

`solve_LSAP`, `ggplot`

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Subject sorter cluster 1 to concept mapping using default "network" method
myCMAnalysis1 <- conceptMapping(myCMDataBySorters[[1]])

# Subject sorter cluster 3 to concept mapping using default "network" method
myCMAnalysis3 <- conceptMapping(myCMDataBySorters[[3]])

# Visualise comparison of results of two sorter clusters
crossClusterMap(myCMAnalysis1, myCMAnalysis3)
```

---

numberOfSorters *Count the Number of Sorters in Concept Map Data*

Description

This function calculates the number of unique sorters (users) in a given concept mapping dataset.

Usage

```
numberOfSorters(CMData, verbose = TRUE)
```

Arguments

CMData	A data frame containing concept map data. This must include a column named "sorterID".
verbose	A logical, if TRUE, the function will print the number of sorters.

Details

The function first checks if the provided dataset is suitable for concept mapping using the checkConceptMapData function. If the data is valid, it calculates and returns the number of unique sorterIDs.

Value

An integer representing the number of unique sorters in the dataset.

Examples

```
# Example of valid data
CMData <- data.frame(
    sorterID = c("resp1", "resp1", "resp1", "resp2",
     "resp2", "resp2", "resp3", "resp3", "resp3"),
   statement = c("London", "Frankfurt", "Berlin", "London",
    "Frankfurt", "Berlin", "London", "Frankfurt", "Berlin"),
    stackID = c("capital city", "city", "capital city", 1, 2, 2, "A", "B", "A")
)

# Count the number of sorters silently
numberOfSorters(CMData, verbose = FALSE)

# Count the number of sorters with message
numberOfSorters(CMData)
```

---

numberOfStatements *Count the Number of Statements in Concept Map Data*

Description

This function calculates the number of unique statements in a given concept mapping dataset.

Usage

```
numberOfStatements(CMData, verbose = TRUE)
```

Arguments

CMData	A data frame containing concept map data. This must include a column named "statement".
verbose	A logical, if TRUE, the function will print the number of statements the console.

Details

The function first checks if the provided dataset is suitable for concept mapping using the `checkConceptMapData` function. If the data is valid, it calculates and returns the number of unique `statements`.

Value

An integer representing the number of unique statements in the dataset.

Examples

```
# Example of valid data
CMData <- data.frame(
  sorterID = c("resp1", "resp1", "resp1", "resp2",
   "resp2", "resp2", "resp3", "resp3", "resp3"),
  statement = c("London", "Frankfurt", "Berlin", "London",
   "Frankfurt", "Berlin", "London", "Frankfurt", "Berlin"),
  stackID = c("capital city", "city", "capital city", 1, 2, 2, "A", "B", "A")
)

# Count the number of statements silently
numberOfStatements(CMData, verbose = FALSE)

# Count the number of statements with message
numberOfStatements(CMData)
```

`plot.conceptMap` *Plot a Concept Map*

Description

This function generates visualizations for an object of class `conceptMap`. Depending on the method used to create the concept map, different types of plots (e.g., heatmap, silhouette, network, or CMDS plots) are available.

Usage

```
## S3 method for class 'conceptMap'
plot(x, whichPlot = "all", ...)
```

Arguments

`x`	An object of class `conceptMap`. This object must contain plotting data and attributes specific to the method used for creating the concept map.
`whichPlot`	A character string specifying which plot to display. Options depend on the method used to create the concept map: `"all"` Displays all available plots for the given method.

"heatmap" Displays the heatmap plot (if available).
"silhouette" Displays the silhouette plot (if available).
"network" Displays the network plot (if available).
"cmds" Displays the CMDs plot (if available).
The default is "all".

... arguments to be passed to methods

Details

The function behaves differently depending on the method used to create the concept map. The conceptMap object must include attributes such as method (e.g., "kmeans", "network", or "cmds") and the corresponding plot objects (e.g., heatmapPlot, silhouettePlot, etc.).

The following methods are supported:

- "kmeans": Supports "heatmap" and "silhouette" plots.
- "network": Supports "heatmap" and "network" plots.
- "cmds": Supports "heatmap", "silhouette", and "cmds" plots.

Value

This function displays the specified plot(s) in the current graphical device.

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Subject sorter cluster 3 to concept mapping using default "network" method
myCMAnalysis3 <- conceptMapping(myCMDataBySorters[[3]])

# Visualise the concept map
plot(myCMAnalysis3)
```

---

print.conceptMap *Print a Concept Map Object*

Description

This function prints a detailed summary of a concept map object, including information about statements, sorters, clusters, and the specific statements within selected clusters.

Usage

```
## S3 method for class 'conceptMap'
print(x, whichCluster = NULL, ...)
```

Arguments

x	An object of class "conceptMap" containing the results of concept mapping analysis creating using the function conceptMapping.
whichCluster	A vector of cluster numbers to display. If NULL (default), all clusters are displayed.
...	arguments to be passed to methods

Details

The function first checks if the input object is of class "conceptMap" and validates the requested clusters (if specified). It provides an overview of the number of statements, sorters, and clusters. For each requested cluster, the function lists the statements included in that cluster.

Value

This function does not return a value; it prints the details of the concept map to the console.

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Subject sorter cluster 3 to concept mapping using default "network" method
myCMAnalysis3 <- conceptMapping(myCMDataBySorters[[3]])

# Print content of concept map of sorter cluster 3
print(myCMAnalysis3)
```

---

printSorters *Print and Return Unique Sorters in Concept Map Data*

Description

This function retrieves and optionally prints the unique sorters (users) in a given concept mapping dataset.

Usage

```
printSorters(CMData, verbose = TRUE)
```

Arguments

CMData	A data frame containing concept map data. This must include a column named "sorterID".
verbose	A logical, if TRUE, the function will print the list of unique sorters to the console.

Details

The function first checks if the provided dataset is suitable for concept mapping using the checkConceptMapData function. If the data is valid, it retrieves the unique sorter IDs from the sorterID column. If verbose = TRUE, the function prints the sorter IDs.

Value

A vector of unique sorter IDs.

Examples

```
# Example of valid data
CMData <- data.frame(
    sorterID = c("resp1", "resp1", "resp1", "resp2",
     "resp2", "resp2", "resp3", "resp3", "resp3"),
   statement = c("London", "Frankfurt", "Berlin", "London",
    "Frankfurt", "Berlin", "London", "Frankfurt", "Berlin"),
    stackID = c("capital city", "city", "capital city", 1, 2, 2, "A", "B", "A")
)

# Retrieve unique sorters without printing
printSorters(CMData, verbose = FALSE)

# Retrieve and print unique sorters to console
printSorters(CMData)
```

---

selectStatements *Select Significant Statements from Concept Map Data*

Description

This function selects statements from a concept map dataset based on their significance in terms of co-occurrence. It applies a chi-squared test on the co-occurrence matrix of the statements to identify those that are statistically significant (i.e., those that co-occur more frequently than would be expected by chance).

Usage

```
selectStatements(CMData, significanceThreshold = 0.05, verbose = TRUE, ...)
```

Arguments

CMData	A data frame containing concept map data. The data should have at least the following columns: • statement: The text of the statement. • sorterID: The identifier for the sorter. • stackID: The identifier for the stack.
significanceThreshold	A numeric value representing the significance threshold for the chi-squared test. Statements with p-values less than this threshold are considered significant. Default is 0.05.
verbose	Logical, if TRUE, additional information about the processing steps is printed to the console.
...	Additional arguments to be passed to the chi-squared test (optional).

Value

A data frame with the same structure as the input, but with non-significant statements removed (if any). If all statements are significant, the original data frame is returned unchanged.

See Also

chisq.test for chi-squared test functionality.

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.70, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Select significant statements
mySelectedStatementsSorterCluster3 <- selectStatements(myCMDataBySorters[[1]])
```

`simulateCardData` *Simulate Card Sorting Data*

Description

This function simulates card sorting data based on user-specified parameters, such as the number of sorters, the probability of correct sorting, and the weights for different card attributes.

Usage

```
simulateCardData(
  nSorters = 40,
  pCorrect = 0.95,
  attributeWeights = c(1, 1, 1, 1),
  verbose = TRUE
)
```

Arguments

`nSorters`	An integer specifying the number of sorters to simulate. Default is 40.
`pCorrect`	A numeric value between 0 and 1 specifying the probability that a card is sorted correctly. Default is 0.95.
`attributeWeights`	A numeric vector of length 4 specifying the weights for the card attributes (e.g., `color`, `suit`, `rank_picture`, and `odd_even_picture`). Default is `c(1, 1, 1, 1)`.
`verbose`	Logical, if `TRUE`, additional information about the processing steps is printed to the console.

Details

The function simulates a card sorting experiment where cards are sorted by multiple sorters based on one of four attributes. The probability of sorting a card correctly is determined by `pCorrect`, and errors are introduced randomly for each sorter. The attribute weights determine how many sorters focus on each attribute, and a warning is issued if the weights do not align with the total number of sorters.

The function returns a data frame containing simulated card sorting data for all sorters, including sorter IDs, card IDs, and assigned stacks.

Value

A data frame with columns:

`sorterID` A unique identifier for each sorter.

`statement` The ID of the card being sorted.

`stackID` The stack assigned to the card by the sorter.

See Also

rbinom, sample

Examples

```
# Simulate data with default parameters:
set.seed(1)
myCMData <- simulateCardData()

# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))
```

---

sorterMapping *Cluster Sorters in Concept Mapping Data*

Description

This function performs clustering of sorters in concept mapping data based on their sorting behavior. It uses hierarchical clustering and allows the automatic determination of the optimal number of clusters or a user-defined number.

Usage

```
sorterMapping(
  CMData,
  numberOfSorterClusters = "auto",
  verbose = TRUE,
  rangeNumberOfClusters = 2:15,
  graph = TRUE
)
```

Arguments

CMData	A data frame containing concept mapping data. It must include the columns: "sorterID", "statement", and "stackID".
numberOfSorterClusters	Either a character string ("auto") to automatically determine the optimal number of clusters or an integer specifying the desired number of clusters.
verbose	Logical, if TRUE, additional information about the processing steps is printed to the console.
rangeNumberOfClusters	A vector of integers specifying the range of clusters to evaluate when numberOfSorterClusters = "auto". Default is 2:15.
graph	Logical. If TRUE, plots the dendrogram and silhouette method results. Default is TRUE.

Details

This function clusters sorters based on their sorting behavior using hierarchical clustering with Ward's method. If `numberOfSorterClusters = "auto"`, the silhouette method is used to determine the optimal number of clusters within the range specified by `rangeNumberOfClusters`.

Each cluster's data is validated for its suitability for concept mapping, and cluster-specific data is returned as a list of data frames. Graphical output includes a dendrogram and silhouette plot if `graph = TRUE`.

Value

A list of data frames, each representing the concept mapping data for a cluster of sorters. If only one cluster is found, the original `CMData` is returned.

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Subject the data to sorter cluster analysis with a predefined number of sorter clusters
myCMDataBySorters <- sorterMapping(myCMData, numberOfSorterClusters=2)
```

---

summary.conceptMap *Summary of a Concept Map Object*

Description

This function provides a summary of a concept map object, including the number of statements, sorters, clusters, and the distribution of statements across clusters.

Usage

```
## S3 method for class 'conceptMap'
summary(object, ...)
```

Arguments

`object`	An object of class `"conceptMap"` containing the results of concept mapping analysis creating using the function `"conceptMapping"`.
`...`	arguments to be passed to methods

Details

The function verifies that the input object is of class "conceptMap" and extracts key information from it. It summarizes the number of statements, sorters, and clusters, and details the distribution of statements across the identified clusters.

Value

This function does not return a value; it prints a summary of the concept map object to the console.

Examples

```
# Simulate data with custom parameters:
set.seed(1)
myCMData <- simulateCardData(nSorters=40, pCorrect=.90, attributeWeights=c(1,1,1,1))

# Subject the data to sorter cluster analysis
myCMDataBySorters <- sorterMapping(myCMData)

# Subject sorter cluster 3 to concept mapping using default "network" method
myCMAnalysis3 <- conceptMapping(myCMDataBySorters[[3]])

# Generate summary of concept map of sorter cluster 3
summary(myCMAnalysis3)
```

Index