Package ‘AnimalSequences’

September 23, 2024

Type Package

Title Analyse Animal Sequential Behaviour and Communication

Version 0.2.0

Description All animal behaviour occurs sequentially. The package has a number of functions to format sequence data from different sources, to analyse sequential behaviour and communication in animals. It also has functions to plot the data and to calculate the entropy of sequences.

License Apache License (>= 2.0)

Encoding UTF-8

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Depends R (>= 4.0.0)

Imports stringr, dplyr, tidytext, ggplot2, fpc, mclust, kernlab, dbscan, apcluster, tidyr, tibble, stats, rlang, igraph, ggraph, magrittr, naivebayes, ranger

RoxygenNote 7.3.1

NeedsCompilation yes

ByteCompile true

Config/testthat/edition 3

Author Alex Mielke [aut, cre]

Maintainer Alex Mielke <a.mielke@qmul.ac.uk>

Repository CRAN

Date/Publication 2024-09-23 13:02:06 UTC

Contents

`association_metrics` *Calculate Association Metrics for Sequences*

Description

This function calculates various association metrics for elements in a sequence, such as Pointwise Mutual Information (PMI), normalized PMI, attraction, reliance, Delta P, z-score, t-score, Chi-squared, Jaccard coefficient, Dice coefficient, log odds ratio, and geometric mean.

Usage

```
association_metrics(sequences)
```

Arguments

sequences A character vector of sequences to analyze.

Value

A data frame with the calculated association metrics for each dyad (pair of elements).

Examples

```
# Example usage:
sequences <- c("A B C", "A B", "A C", "B C", "A B C D")
result <- association_metrics(sequences)
print(result)
```

`average_seq_length` *Calculate the Average Length of Sequences*

Description

This function calculates the average length of a sequence of elements, where each sequence is split by spaces.

Usage

```
average_seq_length(sequences)
```

Arguments

sequences A character vector where each element is a sequence of elements separated by spaces.

Value

A numeric value representing the average length of the sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
average_seq_length(sequences)
```

`calculate_conditional_entropy`
Calculate Conditional Entropy of B given A in Bits

Description

This function calculates the conditional entropy of B given A in bits between two categorical vectors

Usage

```
calculate_conditional_entropy(vectorA, vectorB)
```

Arguments

`vectorA`	A categorical vector representing the conditioning variable A
`vectorB`	A categorical vector representing the conditioned variable B

Value

The conditional entropy of B given A in bits

Examples

```
vectorA <- c("A", "B", "A", "A", "B", "C", "C", "C", "A", "B")
vectorB <- c("X", "Y", "X", "Y", "X", "Y", "Y", "X", "X", "Y")
calculate_conditional_entropy(vectorA, vectorB)
```

`calculate_distance_matrix`
Calculate Distance Matrix from Co-occurrence Matrix

Description

This function calculates a distance matrix from a given co-occurrence matrix.

Usage

```
calculate_distance_matrix(cooccurrence_matrix)
```

Arguments

`cooccurrence_matrix`
A matrix representing the co-occurrence counts of elements.

Value

A distance matrix computed from the normalized co-occurrence matrix.

Examples

```
# Example usage:
cooccurrence_matrix <- matrix(c(3, 2, 1, 2, 5, 0, 1, 0, 4), nrow = 3, byrow = TRUE)
result <- calculate_distance_matrix(cooccurrence_matrix)
print(result)
```

---

calculate_transition_counts

Calculate Transition Counts from Sequences

Description

This function calculates the transition counts between elements in a set of sequences. It creates a matrix where each element represents the number of times a transition occurs from one element to another.

Usage

```
calculate_transition_counts(sequences)
```

Arguments

`sequences`	A vector of character strings, where each string represents a sequence of elements separated by spaces. Elements should be labeled with prefixes (e.g., "e1", "e2").

Details

The function assumes that elements in the sequences are labeled with prefixes (e.g., "e1", "e2"), which are stripped to extract the integer labels for counting. The matrix is initialized to be of size `num_elements` x `num_elements`, where `num_elements` should be defined in your script or session. Ensure that `num_elements` is set to the correct number of unique elements before running this function.

Value

A matrix where the entry at `[i, j]` represents the number of times an element labeled `i` is followed by an element labeled `j` across all sequences.

Examples

```
sequences <- c("e1 e2 e3", "e2 e3 e1", "e1 e3")
num_elements <- 3
calculate_transition_counts(sequences)
```

calculate_transition_probs
Calculate Transition Probabilities from Sequences

Description

This function calculates the transition probabilities between elements in a set of sequences. It computes the probability of transitioning from one element to another based on the frequency of transitions observed in the input sequences.

Usage

```
calculate_transition_probs(sequences)
```

Arguments

`sequences`	A vector of character strings, where each string represents a sequence of elements separated by spaces.

Details

The function uses the ‘unnest_tokens‘ function from the ‘tidytext‘ package to split sequences into individual elements. It then calculates transition counts and probabilities for each pair of consecutive elements in the sequences. The resulting data frame shows the transition probabilities for each possible element pair.

Value

A data frame with the following columns:

`previous_element`	The element that transitions to the next element.
`element`	The element that follows the previous element.
`count`	The number of times the transition from the previous element to the current element occurs.
`probability`	The probability of transitioning from the previous element to the current element.

Examples

```
library(tidytext)
sequences <- c("A B C", "A B", "B C A")
calculate_transition_probs(sequences)
```

cluster_elements *Cluster Elements Using Hierarchical Clustering*

Description

This function performs hierarchical clustering on a distance matrix and optionally plots the dendrogram. It uses the specified method for clustering and can visualize the results.

Usage

```
cluster_elements(distance_matrix, method = "complete", plot = TRUE)
```

Arguments

distance_matrix	A matrix of distances between elements. Should be a symmetric matrix with row and column names representing elements.
method	A character string specifying the method for hierarchical clustering. Options include "complete", "average", "single", etc. Default is "complete".
plot	A logical value indicating whether to plot the dendrogram. Default is TRUE.

Details

Hierarchical clustering is performed using the specified method. If `plot` is TRUE, the function will generate a dendrogram to visualize the clustering.

Value

An object of class `"hclust"` representing the hierarchical clustering result.

Examples

```
# Create a distance matrix
distance_matrix <- dist(matrix(rnorm(20), nrow = 5))

# Perform hierarchical clustering and plot the dendrogram
cluster_elements(distance_matrix, method = "complete", plot = TRUE)
```

---

compare_distinct_elements_per_list_item
Compare True and Shuffled Distinct Elements per List Item

Description

This function compares the true number of distinct elements per list item in a list of sequences to the number of distinct elements per list item in shuffled sequences. The comparison is done by calculating p-values from shuffled sequences.

Usage

```
compare_distinct_elements_per_list_item(sequences, iterations = 100)
```

Arguments

`sequences`	A list of character vectors, where each vector contains sequences of elements separated by spaces.
`iterations`	An integer specifying the number of shuffling iterations.

Value

A data frame with columns:

`true_distinct_elements`	The number of distinct elements per list item in the original sequences.
`shuffled_distinct_elements`	The average number of distinct elements per list item in shuffled sequences.
`p_value`	The p-value representing the proportion of shuffled sequences where the number of distinct elements is less than or equal to the true number.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world', 'hello world')
compare_distinct_elements_per_list_item(sequences, iterations = 100)
```

---

cooccurrence_matrix *Calculate Co-occurrence Matrix for Sequences*

Description

This function calculates a co-occurrence matrix for elements in sequences.

Usage

```
cooccurrence_matrix(sequences)
```

Arguments

`sequences` A character vector of sequences to analyze.

Value

A matrix representing the co-occurrence counts of elements.

Examples

```
# Example usage:
sequences <- c("e1 e2 e3", "e2 e3 e4", "e1 e4", "e1 e2 e4")
result <- cooccurrence_matrix(sequences)
print(result)
```

`count_distinct_elements` *Count Distinct Elements in Sequences*

Description

This function counts the number of distinct elements across all sequences, where each sequence is split by spaces.

Usage

```
count_distinct_elements(sequences)
```

Arguments

`sequences` A character vector where each element is a sequence of elements separated by spaces.

Value

An integer representing the number of distinct elements across all sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
count_distinct_elements(sequences)
```

count_distinct_elements_per_list_item

Count Distinct Elements per List Item

Description

This function calculates the average number of distinct elements per item in a list of sequences, where each sequence is split by spaces.

Usage

```
count_distinct_elements_per_list_item(sequences)
```

Arguments

sequences	A list of character vectors, where each vector contains sequences of elements separated by spaces.

Value

A numeric value representing the average number of distinct elements per list item.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
count_distinct_elements_per_list_item(sequences)
```

count_distinct_elements_per_list_item_shuffled

Count Distinct Elements per List Item in Shuffled Sequences

Description

This function calculates the number of distinct elements per list item in a list of sequences shuffled using the 'shuffle_sequences_across' function. The shuffling is performed a specified number of times.

Usage

```
count_distinct_elements_per_list_item_shuffled(sequences, iterations = 100)
```

Arguments

sequences	A list of character vectors, where each vector contains sequences of elements separated by spaces.
iterations	An integer specifying the number of shuffling iterations.

Value

A numeric vector of length ‘iterations‘, each element representing the number of distinct elements per list item in a shuffled sequence.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
count_distinct_elements_per_list_item_shuffled(sequences, iterations = 100)
```

---

element_covariate — *Calculate Element-Covariate Conditional Probabilities*

Description

This function calculates the conditional probability of each element given each covariate and performs permutation tests to compute the expected conditional probabilities and p-values.

Usage

```
element_covariate(
  sequences_long,
  element = "element",
  covariate = "covariate",
  n_permutations = 1000
)
```

Arguments

sequences_long	A data frame containing the sequences, with columns for elements and covariates.
element	A string specifying the column name for elements in the sequences data frame.
covariate	A string specifying the column name for covariates in the sequences data frame.
n_permutations	An integer specifying the number of permutations for the bootstrapping process.

Value

A data frame with the calculated probabilities, expected probabilities, and p-values for each element-covariate pair.

Examples

```
# Example usage:
sequences_long <- data.frame(
  element = rep(letters[1:3], each = 4),
  covariate = rep(letters[4:7], times = 3)
)
result <- element_covariate(sequences_long,
```

```
                              element = 'element',
                              covariate = 'covariate',
                              n_permutations = 50)
print(result)
```

---

element_covariate_network

Plot the network of elements and covariates based on the long format of sequences

---

Description

Plot the network of elements and covariates based on the long format of sequences

Usage

```
element_covariate_network(
  sequences_long,
  cutoff = 3,
  element,
  covariate,
  n_permutations = 1000,
  pvalue = 0.01,
  clusters = FALSE
)
```

Arguments

`sequences_long`	A data frame containing the sequences, with columns for elements and contexts.
`cutoff`	minimum number of occurrences for which element or covariate should be included
`element`	A string specifying the column name for elements in the sequences data frame.
`covariate`	A string specifying the column name for contexts in the sequences data frame.
`n_permutations`	An integer specifying the number of permutations for the bootstrapping process.
`pvalue`	cutoff pvalue to include combination
`clusters`	should clusters be calculated and added?

Value

plot of bimodal network containing the elements and covariates

element_duration *Calculate Individual Element Durations*

Description

This function calculates the individual element durations and compares them to a shuffled distribution.

Usage

```
element_duration(sequences_long, n_permutations = 1000)
```

Arguments

sequences_long	A data frame containing sequences with start and end times for each element.
n_permutations	An integer specifying the number of permutations to perform. Default is 1000.

Value

A data frame with the median duration, standard deviation, expected duration, effect size, and p-value for each element.

Examples

```
# Example usage:
sequences_long <- data.frame(
  element = c("A", "B", "C", "A", "B", "C"),
  start_time = c(0, 5, 10, 15, 20, 25),
  end_time = c(5, 10, 15, 20, 25, 30)
)
result <- element_duration(sequences_long, n_permutations = 100)
print(result)
```

element_position *Calculate Median Position of Each Element in Sequences*

Description

This function calculates the median position of each element across sequences, summarizes the distribution, and compares it to a shuffled distribution.

Usage

```
element_position(sequences, n_permutations = 1000)
```

Arguments

`sequences`	A character vector of sequences to analyze.
`n_permutations`	The number of permutations to use for the null distribution.

Value

A data frame with the median position, standard deviation, expected position, effect size, and p-value for each element.

Examples

```
# Example usage:
sequences <- c("A B C", "A B", "A C", "B C", "A B C D")
result <- element_position(sequences)
print(result)
```

`find_most_similar_columns`

Find Most Similar Columns in a Distance Matrix

Description

This function identifies the most similar columns for each column in a distance matrix. For each column, it finds the columns with the smallest distances (i.e., most similar) based on the given number of similar columns to retrieve.

Usage

```
find_most_similar_columns(distance_matrix, n_similar = 3)
```

Arguments

`distance_matrix`	A numeric matrix where the distance between columns is represented. The rows and columns should correspond to the same set of entities.
`n_similar`	An integer specifying the number of most similar columns to find for each column. Default is 3.

Value

A list of character vectors. Each element of the list corresponds to a column in the distance matrix and contains the column names of the most similar columns.

Examples

```
# Create a sample distance matrix
distance_matrix <- matrix(c(0, 1, 2, 1, 0, 3, 2, 3, 0),
                          nrow = 3,
                          dimnames = list(NULL, c("A", "B", "C")))

# Find the 2 most similar columns for each column
find_most_similar_columns(distance_matrix, n_similar = 2)
```

---

generate_sequence *Generate a Sequence of Elements*

Description

This function generates a sequence of elements based on a given length function, a transition matrix, and probabilities for the first element. The sequence is generated by sampling from the transition matrix and then combining the sampled elements into a single sequence string.

Usage

```
generate_sequence(length_func, transition_matrix, first_element_probs)
```

Arguments

length_func	A function that generates a numeric value representing the length of the sequence. It is typically a random function that defines the length distribution of the sequence.
transition_matrix	A matrix representing the transition probabilities between elements. Each entry in the matrix indicates the probability of transitioning from one element to another.
first_element_probs	A numeric vector of probabilities for selecting the first element in the sequence. The length of the vector should match the number of possible elements.

Value

A character string representing the generated sequence. The sequence elements are prefixed with "e" and separated by spaces.

Examples

```
# Define parameters
num_elements <- 3
average_sequence_length <- 5
sequence_length_sd <- 1
length_func <- function() {
  rnorm(1, mean = average_sequence_length, sd = sequence_length_sd)
```

```
}
transition_matrix <- matrix(c(0.1, 0.6, 0.3,
                              0.2, 0.5, 0.3,
                              0.3, 0.3, 0.4), nrow = 3, byrow = TRUE)
first_element_probs <- c(0.3, 0.4, 0.3)

# Generate a sequence
generate_sequence(length_func, transition_matrix, first_element_probs)
```

`long_to_sequences` *Convert Long Format to Sequences*

Description

This function converts a data frame in long format into sequences by combining all rows with the same sequence identifier. It also aggregates covariates if provided.

Usage

```
long_to_sequences(
  sequences_long,
  elements = "element",
  sequence_identifier = "sequence_identifier",
  start_time = "start_time",
  end_time = "end_time",
  covariates = NULL
)
```

Arguments

`sequences_long`	A data frame in long format containing the sequences.
`elements`	Column name for elements that should be combined into sequences.
`sequence_identifier`	Column name with the sequence identifier.
`start_time`	Column name with the start time.
`end_time`	Column name with the end time.
`covariates`	A vector with column names of the covariates. Defaults to NULL.

Value

A data frame with sequences, start time, end time, and aggregated covariates.

Examples

```
sequences_long <- data.frame(sequence_identifier = c(1, 1, 2, 2, 2),
                             element = c('A', 'B', 'A', 'B', 'C'),
                             start_time = c(1, 2, 1, 2, 3),
                             end_time = c(2, 3, 2, 3, 4),
                             covariate1 = c('X', 'Y', 'X', 'Y', 'Z'),
                             covariate2 = c('M', 'N', 'M', 'N', 'O'))
long_to_sequences(sequences_long,
                  elements = 'element',
                  sequence_identifier = 'sequence_identifier',
                  start_time = 'start_time',
                  covariates = c('covariate1', 'covariate2'))
```

---

median_seq_length *Calculate the Median Length of Sequences*

Description

This function calculates the median length of a sequence of elements, where each sequence is split by spaces.

Usage

```
median_seq_length(sequences)
```

Arguments

sequences	A character vector where each element is a sequence of elements separated by spaces.

Value

A numeric value representing the median length of the sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
median_seq_length(sequences)
```

menzerath_plot *Create a Menzerath-Altmann Plot*

Description

This function generates a Menzerath-Altmann plot from a data frame in long format. The plot visualizes the relationship between the number of elements in sequences and the mean duration of these sequences.

Usage

```
menzerath_plot(sequences_long)
```

Arguments

sequences_long	A data frame in long format. It should include columns for ‘sequence_nr‘, ‘start_time‘, and ‘end_time‘. Each row represents an element in the sequence with its start and end times.

Details

The function calculates the duration of each element as the difference between ‘end_time‘ and ‘start_time‘. It then groups the data by ‘sequence_nr‘ to compute the number of elements and the mean duration of each sequence. The resulting plot helps in understanding the relationship described by the Menzerath-Altmann law, which postulates that larger linguistic units tend to have shorter mean durations.

Value

A ‘ggplot‘ object. The plot shows the number of elements (x-axis) against the mean duration of sequences (y-axis) with a linear regression line.

Examples

```
# Sample data frame
sequences_long <- data.frame(
  sequence_nr = rep(1:5, each = 3),
  start_time = rep(1:3, times = 5),
  end_time = rep(2:4, times = 5)
)
menzerath_plot(sequences_long)
```

min_max_seq_length *Calculate the Minimum and Maximum Length of Sequences*

Description

This function calculates the minimum and maximum length of sequences of elements, where each sequence is split by spaces.

Usage

```
min_max_seq_length(sequences)
```

Arguments

sequences	A character vector where each element is a sequence of elements separated by spaces.

Value

A numeric vector of length 2, with the minimum and maximum lengths of the sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
min_max_seq_length(sequences)
```

perform_clustering *Perform Various Clustering Methods*

Description

This function performs multiple clustering methods on the input data and returns the results. The methods include K-means, hierarchical clustering, DBSCAN, Gaussian Mixture Model (GMM), spectral clustering, and affinity propagation.

Usage

```
perform_clustering(data, n_clusters = 3)
```

Arguments

data	A numeric matrix or data frame where rows represent observations and columns represent features.
n_clusters	An integer specifying the number of clusters for methods that require it (e.g., K-means, hierarchical clustering). Default is 3.

Details

- **K-means**: Performs K-means clustering with the specified number of clusters. - **Hierarchical clustering**: Performs hierarchical clustering and cuts the dendrogram to create the specified number of clusters. - **DBSCAN**: Applies DBSCAN clustering with predefined parameters. - **Gaussian Mixture Model (GMM)**: Uses the Mclust package to perform GMM clustering. - **Spectral clustering**: Uses the kernlab package to perform spectral clustering with a kernel matrix. - **Affinity propagation**: Uses the apcluster package to perform affinity propagation clustering.

Value

A list with clustering results for each method:

`kmeans`	A list containing the results of K-means clustering, including cluster assignments.
`hierarchical`	A vector of cluster assignments from hierarchical clustering.
`dbscan`	A vector of cluster assignments from DBSCAN.
`gmm`	A vector of cluster assignments from Gaussian Mixture Model (GMM).
`spectral`	A vector of cluster assignments from spectral clustering.
`affinity_propagation`	A list of clusters from affinity propagation.

Examples

```
# Generate sample data
data <- matrix(rnorm(100), nrow = 10)

# Perform clustering
clustering_results <- perform_clustering(data, n_clusters = 3)

# Access the results
clustering_results$kmeans
clustering_results$hierarchical
clustering_results$dbscan
clustering_results$gmm
clustering_results$spectral
clustering_results$affinity_propagation
```

`plot_seq_length_distribution`

Plot the Distribution of Sequence Lengths

Description

This function plots the distribution of the lengths of sequences of elements, where each sequence is split by spaces. The plot includes a histogram and a vertical line indicating the mean length.

Usage

```
plot_seq_length_distribution(sequences)
```

Arguments

sequences	A character vector where each element is a sequence of elements separated by spaces.

Value

A ‘ggplot‘ object showing the distribution of sequence lengths.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
plot_seq_length_distribution(sequences)
```

redundancy *Calculate Observed and Expected Redundancy of Sequences*

Description

This function calculates the observed redundancy of sequences and compares it to expected redundancy values obtained from shuffled sequences. The redundancy is defined as the proportion of consecutive identical elements in the sequences.

Usage

```
redundancy(sequences)
```

Arguments

sequences	A vector of character strings, where each string represents a sequence of elements separated by spaces.

Details

The function calculates redundancy as the proportion of consecutive identical elements within each sequence. It then compares this observed redundancy to expected values derived from sequences where elements are shuffled either across sequences or within each sequence. The function relies on auxiliary functions ‘shuffle_sequences_across‘ and ‘shuffle_sequences_within‘ for generating the shuffled sequences.

Value

A data frame with the following columns:

`redundancy` The observed redundancy in the original sequences. This is the mean proportion of consecutive identical elements across all sequences.

`redundancy_expected_across` The expected redundancy obtained from sequences where elements have been shuffled across the sequences.

`redundancy_expected_within` The expected redundancy obtained from sequences where elements have been shuffled within each sequence.

Examples

```
# Example sequences
sequences <- c("A A B C C", "B A A C C", "A B C C C")
# Compute redundancy
redundancy(sequences)
```

---

sd_seq_length — *Calculate the Standard Deviation of Sequence Lengths*

Description

This function calculates the standard deviation of the lengths of sequences of elements, where each sequence is split by spaces.

Usage

```
sd_seq_length(sequences)
```

Arguments

`sequences` A character vector where each element is a sequence of elements separated by spaces.

Value

A numeric value representing the standard deviation of the lengths of the sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
sd_seq_length(sequences)
```

sequences_to_long *Convert Sequences to Long Format*

Description

This function converts a data frame with sequences into long format. It expands each sequence into individual rows, optionally including start and end times and covariates.

Usage

```
sequences_to_long(
  sequences,
  sequence = "sequence",
  start_time = NULL,
  end_time = NULL,
  covariates = NULL
)
```

Arguments

sequences	A data frame containing sequences.
sequence	Column name with the sequences.
start_time	Column name with the start time. Defaults to NULL.
end_time	Column name with the end time. Defaults to NULL.
covariates	A vector with column names of the covariates. Defaults to NULL.

Value

A data frame in long format with sequences, start time, end time, duration, and covariates.

Examples

```
sequences <- data.frame(sequence = c('A B C', 'A B', 'A C', 'B C'),
                        covariate1 = c('X', 'Y', 'X', 'Y'),
                        covariate2 = c('M', 'N', 'M', 'N'))
sequences_to_long(sequences,
                  sequence = 'sequence',
                  covariates = c('covariate1', 'covariate2'))
```

sequence_duration_summary

Summarize Sequence Durations

Description

This function calculates summary statistics for the durations of sequences, where the duration is defined as the difference between ‘end_time‘ and ‘start_time‘. If ‘duration‘ is provided, it will be used directly.

Usage

```
sequence_duration_summary(sequences, start_time, end_time, duration = NULL)
```

Arguments

`sequences`	A character vector where each element is a sequence of elements separated by spaces.
`start_time`	A numeric vector representing the start times of the sequences.
`end_time`	A numeric vector representing the end times of the sequences.
`duration`	(Optional) A numeric vector representing the durations of the sequences. If ‘NULL‘, it will be calculated as ‘end_time - start_time‘.

Value

A data frame with the following columns:

`mean_seq_duration`
: The mean duration of the sequences.

`sd_seq_duration`
: The standard deviation of the sequence durations.

`median_seq_duration`
: The median duration of the sequences.

`min_seq_duration`
: The minimum duration of the sequences.

`max_seq_duration`
: The maximum duration of the sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
start_time <- c(1, 2, 3)
end_time <- c(2, 4, 7)
sequence_duration_summary(sequences, start_time, end_time)
```

sequence_length_summary
Summarize Sequence Lengths

Description

This function calculates summary statistics for the lengths of sequences of elements, including mean, standard deviation, median, minimum, and maximum lengths. It also counts the number of distinct elements and compares this to shuffled sequences.

Usage

```
sequence_length_summary(sequences)
```

Arguments

`sequences` A character vector where each element is a sequence of elements separated by spaces.

Value

A data frame with the following columns:

`mean_seq_elements`
The mean length of the sequences.

`sd_seq_elements`
The standard deviation of the sequence lengths.

`median_seq_elements`
The median length of the sequences.

`min_seq_elements`
The minimum length of the sequences.

`max_seq_elements`
The maximum length of the sequences.

`distinct_elements`
The number of distinct elements across all sequences.

`pvalue_distinct_elements`
The p-value comparing the true number of distinct elements to shuffled sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
sequence_length_summary(sequences)
```

sequence_length_summary_covariate

Summarize Sequence Lengths by Covariate

Description

This function calculates summary statistics for the lengths of sequences of elements, grouped by a specified covariate. It includes mean, standard deviation, median, minimum, and maximum lengths, along with the number of distinct elements and the p-value comparing to shuffled sequences.

Usage

```
sequence_length_summary_covariate(sequences, covariate)
```

Arguments

`sequences` A character vector where each element is a sequence of elements separated by spaces.

`covariate` A vector of covariates with the same length as ‘sequences‘, used to group the sequences.

Value

A data frame with the following columns:

`covariate` The value of the covariate.

`mean_seq_elements` The mean length of sequences for this covariate value.

`sd_seq_elements` The standard deviation of the sequence lengths for this covariate value.

`median_seq_elements` The median length of sequences for this covariate value.

`min_seq_elements` The minimum length of sequences for this covariate value.

`max_seq_elements` The maximum length of sequences for this covariate value.

`distinct_elements` The number of distinct elements for this covariate value.

`pvalue_distinct_elements` The p-value comparing the number of distinct elements to shuffled sequences for this covariate value.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
covariate <- c('A', 'B', 'A')
sequence_length_summary_covariate(sequences, covariate)
```

sequence_length_summary_element
Summarize Sequence Lengths by Element

Description

This function calculates summary statistics for the lengths of sequences containing specific distinct elements. It performs the summary for each distinct element found across the sequences.

Usage

```
sequence_length_summary_element(sequences)
```

Arguments

sequences	A character vector where each element is a sequence of elements separated by spaces.

Value

A data frame with the following columns:

element	The distinct element.
mean_seq_elements	The mean length of sequences containing the element.
sd_seq_elements	The standard deviation of the lengths of sequences containing the element.
median_seq_elements	The median length of sequences containing the element.
min_seq_elements	The minimum length of sequences containing the element.
max_seq_elements	The maximum length of sequences containing the element.
distinct_elements	The number of distinct elements in sequences containing the element.
pvalue_distinct_elements	The p-value comparing the true number of distinct elements to shuffled sequences.

Examples

```
sequences <- c('hello world', 'hello world hello', 'hello world hello world')
sequence_length_summary_element(sequences)
```

shuffle_sequences_across *Shuffle Elements Across All Sequences*

Description

This function shuffles elements across all sequences, preserving the lengths of the original sequences.

Usage

```
shuffle_sequences_across(sequences)
```

Arguments

sequences	A character vector of sequences to shuffle.

Value

A character vector of sequences with elements shuffled across all sequences.

Examples

```
# Example usage:
sequences <- c("A B C", "D E F", "G H I")
result <- shuffle_sequences_across(sequences)
print(result)
```

shuffle_sequences_within *Shuffle Elements Within Each Sequence*

Description

This function shuffles the elements within each sequence independently.

Usage

```
shuffle_sequences_within(sequences)
```

Arguments

sequences	A character vector of sequences to shuffle.

Value

A character vector of sequences with elements shuffled within each sequence.

Examples

```
# Example usage:
sequences <- c("A B C", "D E F", "G H I")
result <- shuffle_sequences_within(sequences)
print(result)
```

`temporal_overlap` *Temporal Overlap*

Description

This function calculates the temporal overlap of elements in sequences. It determines how much each element overlaps with other elements in the same sequence.

Usage

```
temporal_overlap(sequences_long)
```

Arguments

`sequences_long`	A data frame containing sequences with columns: sequence_nr, element, start_time, and end_time.

Value

A data frame summarizing the mean overlap elements and mean overlap proportion for each element.

Examples

```
sequences_long <- data.frame(
  sequence_nr = c(1, 1, 1, 2, 2),
  element = c("A", "B", "C", "A", "B"),
  start_time = c(0, 5, 10, 0, 5),
  end_time = c(5, 10, 15, 5, 10)
)
result <- temporal_overlap(sequences_long)
print(result)
```

transition_chisq *Perform a Chi-Squared Test for Transition Counts*

Description

This function performs a chi-squared test to determine if there are significant differences between observed and expected transition counts in sequences. It calculates the chi-squared statistic and tests the null hypothesis that transitions occur according to the expected frequencies.

Usage

```
transition_chisq(sequences, alpha = 0.05)
```

Arguments

`sequences`	A vector of sequences, where each sequence is a character string with elements separated by spaces.
`alpha`	A numeric value representing the significance level for the chi-squared test. Default is 0.05.

Details

The function calculates observed transition counts from the input sequences, computes expected transition counts based on row and column sums, and performs a chi-squared test to compare observed and expected counts. The test determines if the transitions in the sequences differ significantly from what would be expected by chance.

Value

A list with two elements:

`significant`	A logical value indicating whether the chi-squared test result is significant at the given significance level.
`p_value`	A numeric value representing the p-value of the chi-squared test.

Examples

```
# Define sequences
sequences <- c('e1 e2 e3', 'e2 e1 e3', 'e3 e2 e1')

# Perform chi-squared test
transition_chisq(sequences, alpha = 0.05)
```

transition_entropy *Calculate Transition Entropy for Sequences*

Description

This function calculates the transition entropy for sequences using n-grams. It performs bootstrapping to compute entropy and expected entropy over multiple iterations.

Usage

```
transition_entropy(sequences, ngram = 2, iterations = 20)
```

Arguments

`sequences`	A list of sequences (character vectors) to analyze.
`ngram`	The size of the n-gram (default is 2).
`iterations`	The number of bootstrap iterations (default is 20).

Value

A data frame with calculated entropies, expected entropies, and entropy ratios for each iteration.

Examples

```
sequences <- unlist(list("A B C", "B C A", "C A B"))
transition_entropy(sequences, ngram = 2, iterations = 20)
```

transition_predictions *Transition Predictions*

Description

This function takes sequences of elements and uses a machine learning classifier to predict the next elements in the sequence. It supports n-gram tokenization and k-fold cross-validation. Optionally, it can upsample the training data.

Usage

```
transition_predictions(
  sequences,
  classifier = "nb",
  ngram = 2,
  upsample = TRUE,
  k = 10
)
```

Arguments

sequences	A list of character strings representing sequences of elements.
classifier	A character string specifying the classifier to use. Options are 'nb' for Naive Bayes and 'forest' for random forest.
ngram	An integer specifying the number of elements to consider in the n-gram tokenization. Default is 2.
upsample	A logical value indicating whether to upsample the training data to balance class distribution. Default is TRUE.
k	An integer specifying the number of folds for k-fold cross-validation. Default is 10.

Value

A list containing the mean accuracy, mean null accuracy, and a data frame of prediction errors.

Examples

```
sequences <- list("a b c", "b c d", "c d e")
result <- transition_predictions(sequences, classifier = 'nb', ngram = 2, upsample = TRUE, k = 5)
print(result)
```

---

transition_test *Perform a Statistical Test for Transition Probabilities*

Description

This function performs a permutation test to evaluate the significance of observed transition probabilities in sequences. It compares the observed transition probabilities to those obtained from permuted sequences to determine if the observed probabilities are significantly different from what would be expected by chance.

Usage

```
transition_test(sequences, observed_probs, n_permutations = 1000)
```

Arguments

sequences	A character vector of sequences where each sequence is represented as a string of elements separated by spaces.
observed_probs	A data frame containing observed transition probabilities with columns `previous_element`, `element`, and `probability`.
n_permutations	An integer specifying the number of permutations to perform. Default is 1000.

Details

- **Observed Transition Probabilities**: Calculated from the input sequences. - **Permutations**: The sequences are permuted `n_permutations` times, and transition probabilities are computed for each permutation. - **P-Values**: Calculated as the proportion of permuted transition probabilities that are greater than or equal to the observed transition probabilities.

Value

A data frame with the observed transition probabilities, expected probabilities from permutations, and p-values for each transition. The data frame contains the following columns:

`previous_element`
: The element preceding the transition.

`element`
: The element following the transition.

`probability`
: The observed probability of the transition.

`expected_probability`
: The mean probability of the transition obtained from permuted sequences.

`p_value`
: The p-value indicating the significance of the observed probability compared to the permuted probabilities.

Examples

```
# Example sequences
sequences <- c('e1 e2 e3', 'e2 e3 e4', 'e3 e4 e1')

# Calculate observed transition probabilities
observed_probs <- calculate_transition_probs(sequences)

# Perform the transition test
test_results <- transition_test(sequences, observed_probs, n_permutations = 50)

# View results
head(test_results)
```

---

zipf_plot *Create a Zipf's Law Plot*

Description

This function creates a log-log plot to visualize Zipf's law, which states that the frequency of a word is inversely proportional to its rank in the frequency table. The plot compares the observed frequency distribution of elements with the expected distribution if Zipf's law were true.

Usage

```
zipf_plot(sequences_long)
```

Arguments

sequences_long A data frame containing at least one column named ‘element‘ which represents the elements of sequences. Each element’s frequency is used to create the plot.

Details

- **Observed Frequencies**: Calculated from the provided ‘sequences_long‘ data frame. - **Expected Frequencies**: Calculated using Zipf’s law formula, where the frequency of the element is inversely proportional to its rank. - **Plotting**: Both observed and expected frequencies are plotted on a log-log scale to compare against Zipf’s law.

Value

A ‘ggplot‘ object that visualizes the observed and expected frequencies of elements according to Zipf’s law. The plot includes:

Rank The rank of each element based on its frequency, plotted on a log scale.

Count The observed frequency of each element, plotted on a log scale.

Expected The expected frequency of each element if Zipf’s law were true, shown as a grey dashed line.

Examples

```
# Example data frame
sequences_long <- data.frame(element = c('a', 'b', 'a', 'c', 'b', 'a', 'd', 'c', 'b', 'a'))

# Generate the Zipf's law plot
zipf_plot(sequences_long)
```

Index