

# Package: conversim (via r-universe)

November 20, 2024

**Title** Conversation Similarity Analysis Package

**Version** 0.1.0

**Description** A comprehensive toolkit for analyzing and comparing conversations. This package provides functions to calculate various similarity measures between conversations, including topic, lexical, semantic, structural, stylistic, sentiment, participant, and timing similarities. It supports both pairwise conversation comparisons and analysis of multiple dyads.

**License** GPL (>= 3)

**Imports** tm (>= 0.7-8), lsa (>= 0.73.2), topicmodels (>= 0.2-12), sentimentr (>= 2.7.1), word2vec, lme4, slam, ggplot2, stats

**LazyData** true

**RoxygenNote** 7.3.2

**Encoding** UTF-8

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** <https://github.com/chaoliu-cl/conversim>,  
<http://liu-chao.site/conversim/>

**BugReports** <https://github.com/chaoliu-cl/conversim/issues>

**Config/pak/sysreqs** cmake make libgs10-dev libicu-dev libxml2-dev

**Repository** <https://chaoliu-cl.r-universe.dev>

**RemoteUrl** <https://github.com/chaoliu-cl/conversim>

**RemoteRef** HEAD

**RemoteSha** 3e6d9591c5fb3f6f998686f156706d8f767a8650

## Contents

agg_seq . . . . .	3
calc_sim_cor . . . . .	3
calc_sim_seq . . . . .	4
calc_sum_stats . . . . .	5
combine_sims . . . . .	5
combine_sim_seq . . . . .	6
compare_sim_meas . . . . .	7
compare_style . . . . .	7
cor_sim_seq . . . . .	8
create_windows . . . . .	9
gen_sim_report . . . . .	9
heatmap_sim . . . . .	10
lexical_similarity . . . . .	11
lexical_sim_dyads . . . . .	12
lex_sim_seq . . . . .	12
norm_sim . . . . .	13
participant_sim_dyads . . . . .	14
plot_cor_heatmap . . . . .	14
plot_sims . . . . .	15
plot_sim_comp . . . . .	16
plot_sim_cor_heatmap . . . . .	16
plot_sim_multi . . . . .	17
plot_sim_seq . . . . .	18
plot_sim_time . . . . .	18
plot_sum_stats . . . . .	19
preprocess_dyads . . . . .	20
preprocess_text . . . . .	20
print_sim_report . . . . .	21
radar_sim . . . . .	22
run_example . . . . .	22
semantic_similarity . . . . .	23
semantic_sim_dyads . . . . .	24
sem_sim_seq . . . . .	25
sentiment_similarity . . . . .	25
sentiment_sim_dyads . . . . .	26
sent_sim_seq . . . . .	27
structural_similarity . . . . .	27
structural_sim_dyads . . . . .	28
style_sim_seq . . . . .	29
stylistic_similarity . . . . .	29
stylistic_sim_dyads . . . . .	30
timing_sim_dyads . . . . .	31
topic_similarity . . . . .	31
topic_sim_dyads . . . . .	32
topic_sim_seq . . . . .	33

---

agg_seq	<i>Aggregate Similarity Sequence</i>
---------	--------------------------------------

---

**Description**

Aggregate similarity sequence for a single dyad

**Usage**

```
agg_seq(sequence, num_segments)
```

**Arguments**

sequence	A numeric vector of similarity scores for a single dyad
num_segments	The number of segments to aggregate into

**Details**

This function aggregates a similarity sequence into a specified number of segments for a single dyad.

**Value**

A numeric vector of aggregated similarity scores

**Examples**

```
seq <- c(0.5, 0.6, 0.7, 0.6, 0.8, 0.7, 0.9, 0.8, 0.7, 0.8)
# Aggregate the sequence into 3 segments
agg_3 <- agg_seq(seq, 3)
print(agg_3)

# Aggregate the sequence into 5 segments
agg_5 <- agg_seq(seq, 5)
print(agg_5)
```

---

calc_sim_cor	<i>Calculate correlation between similarity measures</i>
--------------	--

---

**Description**

This function calculates the correlation between different similarity measures.

**Usage**

```
calc_sim_cor(comparison_df)
```

**Arguments**

comparison\_df A data frame output from compare\_sim\_meas()

**Value**

A correlation matrix

**Examples**

```
topic_similarities <- list("1" = c(0.5, 0.6, 0.7), "2" = c(0.4, 0.5, 0.6))
lexical_similarities <- list("1" = c(0.6, 0.7, 0.8), "2" = c(0.5, 0.6, 0.7))
comparison_df <- compare_sim_meas(
  list(topic_similarities, lexical_similarities),
  c("Topic", "Lexical")
)
calc_sim_cor(comparison_df)
```

---

calc\_sim\_seq

*Calculate similarity sequence*

---

**Description**

This function calculates a sequence of similarities between consecutive windows in a conversation.

**Usage**

```
calc_sim_seq(conversation, window_size, similarity_func)
```

**Arguments**

conversation A dataframe containing the conversation, with a column named 'processed\_text'.  
 window\_size An integer specifying the size of each window.  
 similarity\_func A function that calculates similarity between two text strings.

**Value**

A list containing two elements:

sequence A numeric vector of similarity scores between consecutive windows  
 average The mean of the similarity scores

**Examples**

```
conversation <- data.frame(processed_text = c("hello", "world", "how", "are", "you"))
result <- calc_sim_seq(conversation, 2, function(x, y) sum(x == y) / max(length(x), length(y)))
```

---

calc_sum_stats	<i>Calculate summary statistics for similarities</i>
----------------	--

---

**Description**

This function calculates summary statistics for the similarities of multiple dyads.

**Usage**

```
calc_sum_stats(similarities)
```

**Arguments**

`similarities` A list of similarity sequences for each dyad

**Value**

A matrix with summary statistics for each dyad

**Examples**

```
similarities <- list(  
  "1" = c(0.5, 0.6, 0.7),  
  "2" = c(0.4, 0.5, 0.6)  
)  
calc_sum_stats(similarities)
```

---

combine_sims	<i>Utility and visualization functions for speech similarity analysis</i>
--------------	---

---

**Description**

This file contains utility functions and visualization tools to complement the main similarity calculation functions for comparing two speeches. Combine multiple similarity measures

**Usage**

```
combine_sims(similarities, weights = NULL)
```

**Arguments**

`similarities` A named list of similarity scores  
`weights` A named list of weights for each similarity measure (optional)

**Details**

This function combines multiple similarity measures into a single score.

**Value**

A single combined similarity score

**Examples**

```
sims <- list(topic = 0.8, lexical = 0.6, semantic = 0.7, structural = 0.9)
combine_sims(sims)
combine_sims(sims, weights = list(topic = 2, lexical = 1, semantic = 1.5, structural = 1))
```

---

combine\_sim\_seq

*Combine Similarity Measures*

---

**Description**

Combine similarity measures for a single dyad

**Usage**

```
combine_sim_seq(similarities, weights = NULL)
```

**Arguments**

`similarities` A list of similarity measures for a single dyad  
`weights` A numeric vector of weights for each similarity measure (default is equal weights)

**Details**

This function combines multiple similarity measures into a single overall similarity score for a single dyad.

**Value**

A list containing the combined sequence and average similarity

**Examples**

```
sim1 <- list(sequence = c(0.8, 0.7, 0.9), average = 0.8)
sim2 <- list(sequence = c(0.6, 0.8, 0.7), average = 0.7)
combine_sim_seq(list(sim1, sim2))
```

---

compare_sim_meas	<i>Compare multiple similarity measures</i>
------------------	---

---

**Description**

This function compares multiple similarity measures for the same set of dyads.

**Usage**

```
compare_sim_meas(similarity_list, measure_names)
```

**Arguments**

`similarity_list` A list of lists, where each inner list contains similarities for each dyad

`measure_names` A vector of names for each similarity measure

**Value**

A data frame with all similarity measures for each dyad

**Examples**

```
topic_similarities <- list("1" = c(0.5, 0.6, 0.7), "2" = c(0.4, 0.5, 0.6))
lexical_similarities <- list("1" = c(0.6, 0.7, 0.8), "2" = c(0.5, 0.6, 0.7))
compare_sim_meas(
  list(topic_similarities, lexical_similarities),
  c("Topic", "Lexical")
)
```

---

compare_style	<i>Compare stylistic features</i>
---------------	-----------------------------------

---

**Description**

This function visualizes the comparison of stylistic features between two speeches.

**Usage**

```
compare_style(stylistic_result)
```

**Arguments**

`stylistic_result` The result from `stylistic_similarity` function

**Value**

A ggplot object

**Examples**

```
text1 <- "The quick brown fox jumps over the lazy dog. It's a sunny day."  
text2 <- "A lazy cat sleeps on the warm windowsill. Birds chirp outside."  
result <- stylistic_similarity(text1, text2)  
compare_style(result)
```

---

cor\_sim\_seq

*Calculate Correlation Between Similarity Measures for a Single Dyad*

---

**Description**

Calculate Correlation Between Similarity Measures for a Single Dyad

**Usage**

```
cor_sim_seq(similarities, method = "pearson")
```

**Arguments**

`similarities` A list of similarity measures for a single dyad  
`method` The correlation method to use (default is "pearson")

**Details**

This function calculates the correlation between different similarity measures for a single dyad.

**Value**

A correlation matrix

**Examples**

```
sim1 <- list(sequence = c(0.8, 0.7, 0.9), average = 0.8)  
sim2 <- list(sequence = c(0.6, 0.8, 0.7), average = 0.7)  
cor_sim_seq(list(sim1, sim2))
```



---

create_windows	<i>Create windows from a conversation</i>
----------------	---

---

**Description**

This function creates a list of windows from a conversation dataframe.

**Usage**

```
create_windows(conversation, window_size)
```

**Arguments**

conversation    A dataframe containing the conversation, with a column named 'processed\_text'.  
window\_size    An integer specifying the size of each window.

**Value**

A list of character vectors, where each vector represents a window of text.

**Examples**

```
conversation <- data.frame(processed_text = c("hello", "world", "how", "are", "you"))  
windows <- create_windows(conversation, 3)
```

---

gen_sim_report	<i>Generate similarity report</i>
----------------	-----------------------------------

---

**Description**

This function generates a comprehensive report of all similarity measures.

**Usage**

```
gen_sim_report(  
  speech1,  
  speech2,  
  topic_method = "lda",  
  semantic_method = "tfidf",  
  glove_path = NULL  
)
```

**Arguments**

speech1	A character string representing the first speech
speech2	A character string representing the second speech
topic_method	Method for topic similarity calculation ("lda" or "lsa")
semantic_method	Method for semantic similarity calculation ("tfidf", "word2vec", or "glove")
glove_path	Path to pre-trained GloVe file (if using "glove" method)

**Value**

A list containing all similarity measures and visualizations

**Examples**

```
speech1 <- "This is the first speech. It talks about important topics."  
speech2 <- "This is the second speech. It covers similar subjects."  
report <- gen_sim_report(speech1, speech2)
```

---

heatmap\_sim

*Create Similarity Heatmap*

---

**Description**

Create a heatmap of similarity measures for a single dyad

**Usage**

```
heatmap_sim(similarities, titles)
```

**Arguments**

similarities	A list of similarity measures for a single dyad
titles	A character vector of titles for each similarity measure

**Details**

This function creates a heatmap of multiple similarity measures for a single dyad.

**Value**

A ggplot object

**Examples**

```
sim1 <- list(sequence = c(0.5, 0.6, 0.7, 0.6, 0.8), average = 0.64)
sim2 <- list(sequence = c(0.4, 0.5, 0.6, 0.7, 0.7), average = 0.58)
similarities <- list(sim1, sim2)
titles <- c("Measure 1", "Measure 2")

# Plot multiple similarity measures
plot <- plot_sim_multi(similarities, titles)
print(plot)
```

---

lexical_similarity	<i>Calculate lexical similarity between two conversations</i>
--------------------	---

---

**Description**

This function calculates the lexical similarity between two conversations based on the overlap of unique words.

**Usage**

```
lexical_similarity(conv1, conv2)
```

**Arguments**

conv1	A character string representing the first conversation
conv2	A character string representing the second conversation

**Value**

A numeric value representing the lexical similarity

**Examples**

```
conv1 <- "The quick brown fox jumps over the lazy dog"
conv2 <- "The lazy dog sleeps under the quick brown fox"
lexical_similarity(conv1, conv2)
```

---

lexical\_sim\_dyads      *Calculate lexical similarity for multiple dyads*

---

**Description**

This function calculates lexical similarity over a sequence of conversation exchanges for multiple dyads.

**Usage**

```
lexical_sim_dyads(conversations, window_size = 3)
```

**Arguments**

conversations    A data frame with columns 'dyad\_id', 'speaker', and 'processed\_text'  
window\_size      An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities for each dyad and the overall average similarity

**Examples**

```
library(lme4)
convs <- data.frame(
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),
  processed_text = c("i love pizza", "me too favorite food",
                    "whats your favorite topping", "enjoy pepperoni mushrooms",
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",
                    "ever tried making home", "yes quite easy make")
)
lexical_sim_dyads(convs, window_size = 2)
```

---

lex\_sim\_seq      *Calculate lexical similarity sequence for a single dyad*

---

**Description**

This function calculates lexical similarity over a sequence of conversation exchanges within a single dyad.

**Usage**

```
lex_sim_seq(conversation, window_size = 3)
```

**Arguments**

conversation    A data frame representing the conversation  
window\_size    An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities and the average similarity

**Examples**

```
conversation <- data.frame(  
  processed_text = c("Hello world", "World of programming",  
                    "Programming is fun", "Fun world of coding")  
)  
result <- lex_sim_seq(conversation, window_size = 2)  
print(result)
```

---

norm\_sim                      *Normalize Similarity Scores*

---

**Description**

Normalize similarity scores

**Usage**

```
norm_sim(similarities)
```

**Arguments**

similarities    A numeric vector of similarity scores

**Details**

This function normalizes similarity scores to a 0-1 range.

**Value**

A numeric vector of normalized similarity scores

**Examples**

```
similarities <- c(0.2, 0.5, 0.8, 1.0, 0.3)  
norm_sim(similarities)
```

---

participant\_sim\_dyads *Calculate participant similarity for multiple dyads*

---

**Description**

This function calculates an extended measure of participant similarity for multiple dyads.

**Usage**

```
participant_sim_dyads(conversations)
```

**Arguments**

conversations A data frame with columns 'dyad\_id', 'speaker', and 'processed\_text'

**Value**

A list containing participant similarity for each dyad and the overall average similarity

**Examples**

```
convs <- data.frame(  
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),  
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),  
  processed_text = c("i love pizza", "me too favorite food",  
                    "whats your favorite topping", "enjoy pepperoni mushrooms",  
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",  
                    "ever tried making home", "yes quite easy make")  
)  
participant_sim_dyads(convs)
```

---

plot\_cor\_heatmap *Plot Correlation Heatmap for a Single Dyad*

---

**Description**

Plot Correlation Heatmap for a Single Dyad

**Usage**

```
plot_cor_heatmap(cor_matrix, titles)
```

**Arguments**

cor\_matrix A correlation matrix for a single dyad  
titles A character vector of titles for each similarity measure

**Details**

This function creates a heatmap of correlations between similarity measures for a single dyad.

**Value**

A ggplot object

**Examples**

```
sim1 <- list(sequence = c(0.8, 0.7, 0.9), average = 0.8)
sim2 <- list(sequence = c(0.6, 0.8, 0.7), average = 0.7)
cor_matrix <- cor_sim_seq(list(sim1, sim2))
plot_cor_heatmap(cor_matrix, c("Topic", "Lexical"))
```

---

plot\_sims

*Visualize similarity scores*

---

**Description**

This function creates a bar plot of similarity scores.

**Usage**

```
plot_sims(similarities)
```

**Arguments**

similarities    A named list of similarity scores

**Value**

A ggplot object

**Examples**

```
sims <- list(topic = 0.8, lexical = 0.6, semantic = 0.7, structural = 0.9)
plot_sims(sims)
```

---

plot\_sim\_comp      *Plot comparison of multiple similarity measures*

---

**Description**

This function creates a ggplot object comparing multiple similarity measures for the same set of dyads.

**Usage**

```
plot_sim_comp(comparison_df, title)
```

**Arguments**

comparison\_df    A data frame output from compare\_sim\_meas()  
 title            A string specifying the plot title

**Value**

A ggplot object

**Examples**

```
topic_similarities <- list("1" = c(0.5, 0.6, 0.7), "2" = c(0.4, 0.5, 0.6))
lexical_similarities <- list("1" = c(0.6, 0.7, 0.8), "2" = c(0.5, 0.6, 0.7))
comparison_df <- compare_sim_meas(
  list(topic_similarities, lexical_similarities),
  c("Topic", "Lexical")
)
plot_sim_comp(comparison_df, "Comparison of Similarity Measures")
```

---

plot\_sim\_cor\_heatmap    *Plot heatmap of similarity measure correlations*

---

**Description**

This function creates a ggplot object showing a heatmap of correlations between similarity measures.

**Usage**

```
plot_sim_cor_heatmap(cor_matrix, title)
```

**Arguments**

cor\_matrix        A correlation matrix output from calc\_sim\_cor()  
 title            A string specifying the plot title



**Value**

A ggplot object

**Examples**

```
topic_similarities <- list("1" = c(0.5, 0.6, 0.7), "2" = c(0.4, 0.5, 0.6))
lexical_similarities <- list("1" = c(0.6, 0.7, 0.8), "2" = c(0.5, 0.6, 0.7))
comparison_df <- compare_sim_meas(
  list(topic_similarities, lexical_similarities),
  c("Topic", "Lexical")
)
cor_matrix <- calc_sim_cor(comparison_df)
plot_sim_cor_heatmap(cor_matrix, "Correlation of Similarity Measures")
```

---

plot\_sim\_multi

*Plot Multiple Similarity Measures*

---

**Description**

Plot multiple similarity measures for a single dyad

**Usage**

```
plot_sim_multi(similarities, titles)
```

**Arguments**

`similarities` A list of similarity measures for a single dyad  
`titles` A character vector of titles for each similarity measure

**Details**

This function creates a faceted plot of multiple similarity measures for a single dyad.

**Value**

A ggplot object

**Examples**

```
sim1 <- list(sequence = c(0.5, 0.6, 0.7, 0.6, 0.8), average = 0.64)
sim2 <- list(sequence = c(0.4, 0.5, 0.6, 0.7, 0.7), average = 0.58)
similarities <- list(sim1, sim2)
titles <- c("Measure 1", "Measure 2")

# Plot multiple similarity measures
plot <- plot_sim_multi(similarities, titles)
print(plot)
```

---

plot_sim_seq	<i>Plot Similarity Sequence</i>
--------------	---------------------------------

---

**Description**

Plot similarity sequence for a single dyad

**Usage**

```
plot_sim_seq(similarity, title)
```

**Arguments**

similarity	A list containing the sequence of similarities and the average similarity
title	A character string for the plot title

**Details**

This function creates a line plot of the similarity sequence for a single dyad.

**Value**

A ggplot object

**Examples**

```
sim_list <- list(  
  sequence = c(0.5, 0.6, 0.7, 0.6, 0.8),  
  average = 0.64  
)  
  
# Plot the similarity sequence  
plot <- plot_sim_seq(sim_list, "Dyad Similarity Sequence")  
print(plot)
```

---

plot_sim_time	<i>Plot similarity over time for multiple dyads</i>
---------------	---

---

**Description**

This function creates a ggplot object showing the similarity over time for multiple dyads.

**Usage**

```
plot_sim_time(similarities, title, y_label)
```

**Arguments**

similarities    A list of similarity sequences for each dyad  
title            A string specifying the plot title  
y\_label         A string specifying the y-axis label

**Value**

A ggplot object

**Examples**

```
similarities <- list(  
  "1" = c(0.5, 0.6, 0.7),  
  "2" = c(0.4, 0.5, 0.6)  
)  
plot_sim_time(similarities, "Topic Similarity", "Similarity Score")
```

---

plot_sum_stats	<i>Plot summary statistics for similarities</i>
----------------	---

---

**Description**

This function creates a ggplot object showing summary statistics for similarities of multiple dyads.

**Usage**

```
plot_sum_stats(summary_stats, title)
```

**Arguments**

summary\_stats    A data frame with summary statistics for each dyad  
title            A string specifying the plot title

**Value**

A ggplot object

**Examples**

```
similarities <- list(  
  "1" = c(0.5, 0.6, 0.7),  
  "2" = c(0.4, 0.5, 0.6)  
)  
stats <- calc_sum_stats(similarities)  
plot_sum_stats(stats, "Summary Statistics of Similarities")
```

---

```
preprocess_dyads      Preprocess multiple dyad conversations
```

---

**Description**

This function preprocesses conversations from multiple dyads by applying text cleaning to each utterance.

**Usage**

```
preprocess_dyads(conversations)
```

**Arguments**

`conversations` A data frame with columns 'dyad\_id', 'speaker', and 'text'

**Value**

A data frame with an additional 'processed\_text' column, removing any rows with empty processed text

**Examples**

```
convs <- data.frame(
  dyad_id = c(1, 1, 2, 2),
  speaker = c("A", "B", "C", "D"),
  text = c("Hello!", "Hi there!", "How are you?", "I'm fine, thanks!")
)
preprocess_dyads(convs)
```

---

```
preprocess_text      This file contains core similarity calculation functions such as topic
similarity, lexical similarity, semantic similarity, structural similarity,
stylistic similarity, sentiment similarity, participant similarity, and tim-
ing similarity.
```

---

**Description**

Preprocess text for analysis

**Usage**

```
preprocess_text(text)
```

**Arguments**

`text` A character string to be preprocessed

**Details**

This function preprocesses the input text by converting to lowercase, removing punctuation and digits, and trimming whitespace.

**Value**

A preprocessed character string

**Examples**

```
text <- "Hello, World! This is an example text (with 123 numbers)."  
preprocess_text(text)
```

---

`print_sim_report`      *Print similarity report*

---

**Description**

This function prints a formatted summary of the similarity report.

**Usage**

```
print_sim_report(report)
```

**Arguments**

`report`              A similarity report generated by `gen_sim_report` function

**Value**

NULL (invisibly). This function is called for its side effect of printing to the console.

**Examples**

```
speech1 <- "This is the first speech. It talks about important topics."  
speech2 <- "This is the second speech. It covers similar subjects."  
report <- gen_sim_report(speech1, speech2)  
print_sim_report(report)
```

---

radar_sim	<i>Create Radar Chart of Average Similarities</i>
-----------	---

---

**Description**

Create a radar chart of average similarities for a single dyad

**Usage**

```
radar_sim(similarities, titles)
```

**Arguments**

`similarities` A list of similarity measures for a single dyad  
`titles` A character vector of titles for each similarity measure

**Details**

This function creates a radar chart of average similarities for multiple measures of a single dyad.

**Value**

A ggplot object

**Examples**

```
sim1 <- list(sequence = c(0.5, 0.6, 0.7, 0.6, 0.8), average = 0.64)
sim2 <- list(sequence = c(0.4, 0.5, 0.6, 0.7, 0.7), average = 0.58)
sim3 <- list(sequence = c(0.6, 0.7, 0.8, 0.7, 0.9), average = 0.74)
sim4 <- list(sequence = c(0.3, 0.4, 0.5, 0.6, 0.6), average = 0.48)
similarities <- list(sim1, sim2, sim3, sim4)
titles <- c("Measure 1", "Measure 2", "Measure 3", "Measure 4")

# Create radar chart
radar <- radar_sim(similarities, titles)
print(radar)
```

---

run_example	<i>Run package examples</i>
-------------	-----------------------------

---

**Description**

Run package examples

**Usage**

```
run_example(example_name)
```

**Arguments**

example\_name    Name of the example file to run

**Value**

No return value, called for side effects.

**Examples**

```
## Not run:
run_example("sequence_multidyads_examples.R")
run_example("main_functions_examples.R")

## End(Not run)
```

---

semantic\_similarity    *Calculate semantic similarity between two conversations*

---

**Description**

This function calculates the semantic similarity between two conversations using either TF-IDF, Word2Vec, or GloVe embeddings approach.

**Usage**

```
semantic_similarity(
  conversation1,
  conversation2,
  method = "tfidf",
  model_path = NULL,
  dim = 100,
  window = 5,
  iter = 5
)
```

**Arguments**

conversation1    A character string representing the first conversation

conversation2    A character string representing the second conversation

method            A character string specifying the method to use: "tfidf", "word2vec", or "glove"

model\_path        A character string specifying the path to pre-trained GloVe file (required for "glove" method)

dim                An integer specifying the dimensionality for Word2Vec embeddings (default: 100)

window            An integer specifying the window size for Word2Vec (default: 5)

iter               An integer specifying the number of iterations for Word2Vec (default: 5)

**Value**

A numeric value representing the semantic similarity (between 0 and 1)

**Examples**

```
conv1 <- "The quick brown fox jumps over the lazy dog"
conv2 <- "A fast auburn canine leaps above an idle hound"
semantic_similarity(conv1, conv2, method = "tfidf")
```

---

semantic\_sim\_dyads      *Calculate semantic similarity for multiple dyads*

---

**Description**

This function calculates semantic similarity over a sequence of conversation exchanges for multiple dyads.

**Usage**

```
semantic_sim_dyads(conversations, method = "tfidf", window_size = 3, ...)
```

**Arguments**

conversations	A data frame with columns 'dyad_id', 'speaker', and 'processed_text'
method	A character string specifying the method to use: "tfidf", "word2vec", or "glove"
window_size	An integer specifying the size of the sliding window
...	Additional arguments passed to semantic_similarity

**Value**

A list containing the sequence of similarities for each dyad and the overall average similarity

**Examples**

```
library(lme4)
convs <- data.frame(
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),
  processed_text = c("i love pizza", "me too favorite food",
                    "whats your favorite topping", "enjoy pepperoni mushrooms",
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",
                    "ever tried making home", "yes quite easy make")
)
semantic_sim_dyads(convs, method = "tfidf", window_size = 2)
```



---

sem_sim_seq	<i>Calculate semantic similarity sequence for a single dyad</i>
-------------	---

---

**Description**

This function calculates semantic similarity over a sequence of conversation exchanges within a single dyad.

**Usage**

```
sem_sim_seq(conversation, method = "tfidf", window_size = 3, ...)
```

**Arguments**

conversation	A data frame representing the conversation
method	A character string specifying the method to use: "tfidf", "word2vec", or "glove"
window_size	An integer specifying the size of the sliding window
...	Additional arguments passed to semantic_similarity

**Value**

A list containing the sequence of similarities and the average similarity

**Examples**

```
conversation <- data.frame(  
  processed_text = c("The weather is nice", "It's a beautiful day",  
                    "The sun is shining", "Perfect day for a picnic")  
)  
result <- sem_sim_seq(conversation, method = "tfidf", window_size = 2)  
print(result)
```

---

sentiment_similarity	<i>Calculate sentiment similarity between two conversations</i>
----------------------	---

---

**Description**

This function calculates the sentiment similarity between two conversations using the sentimentr package.

**Usage**

```
sentiment_similarity(conv1, conv2)
```

**Arguments**

conv1            A character string representing the first conversation  
conv2            A character string representing the second conversation

**Value**

A numeric value representing the sentiment similarity

**Examples**

```
conv1 <- "I love this product! It's amazing and works great."
conv2 <- "This item is okay. It does the job but could be better."
sentiment_similarity(conv1, conv2)
```

---

sentiment\_sim\_dyads    *Calculate sentiment similarity for multiple dyads*

---

**Description**

This function calculates sentiment similarity over a sequence of conversation exchanges for multiple dyads.

**Usage**

```
sentiment_sim_dyads(conversations, window_size = 3)
```

**Arguments**

conversations    A data frame with columns 'dyad\_id', 'speaker', and 'processed\_text'  
window\_size      An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities for each dyad and the overall average similarity

**Examples**

```
library(lme4)
convs <- data.frame(
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),
  processed_text = c("i love pizza", "me too favorite food",
                    "whats your favorite topping", "enjoy pepperoni mushrooms",
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",
                    "ever tried making home", "yes quite easy make")
)
sentiment_sim_dyads(convs, window_size = 2)
```

---

sent_sim_seq	<i>Calculate sentiment similarity sequence for a single dyad</i>
--------------	--

---

**Description**

This function calculates sentiment similarity over a sequence of conversation exchanges within a single dyad.

**Usage**

```
sent_sim_seq(conversation, window_size = 3)
```

**Arguments**

conversation	A data frame representing the conversation
window_size	An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities and the average similarity

**Examples**

```
conversation <- data.frame(  
  processed_text = c("I love this movie!", "It's really amazing.",  
                    "The acting is superb.", "I couldn't agree more.")  
)  
result <- sent_sim_seq(conversation, window_size = 2)  
print(result)
```

---

structural_similarity	<i>Calculate structural similarity between two conversations</i>
-----------------------	--

---

**Description**

This function calculates the structural similarity between two conversations based on their length and average turn length.

**Usage**

```
structural_similarity(conv1, conv2)
```

**Arguments**

conv1	A character vector representing the first conversation
conv2	A character vector representing the second conversation

**Value**

A numeric value representing the structural similarity

**Examples**

```
conv1 <- c("Hello", "Hi there", "How are you?", "I'm fine, thanks")
conv2 <- c("Good morning", "Hello", "Nice day, isn't it?", "Yes, indeed")
structural_similarity(conv1, conv2)
```

---

structural\_sim\_dyads *Calculate structural similarity for multiple dyads*

---

**Description**

This function calculates an extended measure of structural similarity for multiple dyads.

**Usage**

```
structural_sim_dyads(conversations)
```

**Arguments**

conversations A data frame with columns 'dyad\_id', 'speaker', and 'processed\_text'

**Value**

A list containing structural similarity for each dyad and the overall average similarity

**Examples**

```
convs <- data.frame(
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),
  processed_text = c("i love pizza", "me too favorite food",
                    "whats your favorite topping", "enjoy pepperoni mushrooms",
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",
                    "ever tried making home", "yes quite easy make")
)
structural_sim_dyads(convs)
```

---

style_sim_seq	<i>Calculate stylistic similarity sequence for a single dyad</i>
---------------	--

---

**Description**

This function calculates stylistic similarity over a sequence of conversation exchanges within a single dyad.

**Usage**

```
style_sim_seq(conversation, window_size = 3)
```

**Arguments**

conversation    A data frame representing the conversation  
window\_size    An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities and the average similarity

**Examples**

```
conversation <- data.frame(  
  processed_text = c("How are you doing?", "I'm doing great, thanks!",  
                    "That's wonderful to hear.", "I'm glad you're doing well.")  
)  
result <- style_sim_seq(conversation, window_size = 2)  
print(result)
```

---

stylistic_similarity	<i>Calculate stylistic similarity between two conversations</i>
----------------------	---

---

**Description**

This function calculates various stylistic features and their similarity between two conversations.

**Usage**

```
stylistic_similarity(text1, text2)
```

**Arguments**

text1            A character string representing the first conversation  
text2            A character string representing the second conversation

**Value**

A list containing stylistic features and similarity measures

**Examples**

```
text1 <- "The quick brown fox jumps over the lazy dog. It's a sunny day."  
text2 <- "A lazy cat sleeps on the warm windowsill. Birds chirp outside."  
stylistic_similarity(text1, text2)
```

---

stylistic\_sim\_dyads     *Calculate stylistic similarity for multiple dyads*

---

**Description**

This function calculates stylistic similarity over a sequence of conversation exchanges for multiple dyads.

**Usage**

```
stylistic_sim_dyads(conversations, window_size = 3)
```

**Arguments**

conversations     A data frame with columns 'dyad\_id', 'speaker', and 'processed\_text'  
window\_size       An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities for each dyad and the overall average similarity

**Examples**

```
convs <- data.frame(  
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),  
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),  
  processed_text = c("i love pizza", "me too favorite food",  
                    "whats your favorite topping", "enjoy pepperoni mushrooms",  
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",  
                    "ever tried making home", "yes quite easy make")  
)  
stylistic_sim_dyads(convs, window_size = 2)
```

---

timing\_sim\_dyads      *Calculate timing similarity for multiple dyads*

---

**Description**

This function calculates an extended measure of timing similarity for multiple dyads.

**Usage**

```
timing_sim_dyads(conversations)
```

**Arguments**

conversations    A data frame with columns 'dyad\_id', 'speaker', and 'processed\_text'

**Value**

A list containing timing similarity for each dyad and the overall average similarity

**Examples**

```
convs <- data.frame(
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),
  processed_text = c("i love pizza", "me too favorite food",
                    "whats your favorite topping", "enjoy pepperoni mushrooms",
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",
                    "ever tried making home", "yes quite easy make")
)
timing_sim_dyads(convs)
```

---

topic\_similarity      *Calculate topic similarity between two conversations*

---

**Description**

This function calculates the topic similarity between two conversations using either Latent Dirichlet Allocation (LDA) or Latent Semantic Analysis (LSA).

**Usage**

```
topic_similarity(conv1, conv2, method = "lda", num_topics = 2)
```

**Arguments**

conv1	A character vector representing the first conversation
conv2	A character vector representing the second conversation
method	A character string specifying the method to use: "lda" or "lsa"
num_topics	An integer specifying the number of topics to use in the model

**Value**

A numeric value representing the topic similarity

**Examples**

```
conv1 <- c("I love pizza", "Pizza is my favorite food")
conv2 <- c("I prefer pasta", "Pasta is delicious")
topic_similarity(conv1, conv2, method = "lda", num_topics = 2)
topic_similarity(conv1, conv2, method = "lsa", num_topics = 2)
```

---

topic_sim_dyads	<i>Calculate topic similarity for multiple dyads</i>
-----------------	--

---

**Description**

This function calculates topic similarity over a sequence of conversation exchanges for multiple dyads. It uses the Latent Dirichlet Allocation (LDA) method for topic modeling and the "slam" package for efficient handling of sparse matrices.

**Usage**

```
topic_sim_dyads(conversations, method = "lda", num_topics = 2, window_size = 3)
```

**Arguments**

conversations	A data frame with columns 'dyad_id', 'speaker', and 'processed_text'
method	A character string specifying the method to use: currently only "lda" is supported
num_topics	An integer specifying the number of topics to use in the LDA model
window_size	An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities for each dyad and the overall average similarity



**Examples**

```

convs <- data.frame(
  dyad_id = c(1, 1, 1, 1, 2, 2, 2, 2),
  speaker = c("A", "B", "A", "B", "C", "D", "C", "D"),
  processed_text = c("i love pizza", "me too favorite food",
                    "whats your favorite topping", "enjoy pepperoni mushrooms",
                    "i prefer pasta", "pasta delicious like spaghetti carbonara",
                    "ever tried making home", "yes quite easy make")
)
topic_sim_dyads(convs, method = "lda", num_topics = 2, window_size = 2)

```

---

topic\_sim\_seq

*Calculate topic similarity sequence for a single dyad*


---

**Description**

This function calculates topic similarity over a sequence of conversation exchanges within a single dyad.

**Usage**

```
topic_sim_seq(conversation, method = "lda", num_topics = 2, window_size = 3)
```

**Arguments**

conversation	A data frame representing the conversation
method	A character string specifying the method to use: "lda" or "lsa"
num_topics	An integer specifying the number of topics to use in the model
window_size	An integer specifying the size of the sliding window

**Value**

A list containing the sequence of similarities and the average similarity

**Examples**

```

conversation <- data.frame(
  processed_text = c("The cat sat on the mat", "The dog chased the cat",
                    "The mat was comfortable", "The cat liked the mat")
)
result <- topic_sim_seq(conversation, method = "lda", num_topics = 2, window_size = 2)
print(result)

```

# Index

agg\_seq, 3

calc\_sim\_cor, 3  
calc\_sim\_seq, 4  
calc\_sum\_stats, 5  
combine\_sim\_seq, 6  
combine\_sims, 5  
compare\_sim\_meas, 7  
compare\_style, 7  
cor\_sim\_seq, 8  
create\_windows, 9

gen\_sim\_report, 9

heatmap\_sim, 10

lex\_sim\_seq, 12  
lexical\_sim\_dyads, 12  
lexical\_similarity, 11

norm\_sim, 13

participant\_sim\_dyads, 14  
plot\_cor\_heatmap, 14  
plot\_sim\_comp, 16  
plot\_sim\_cor\_heatmap, 16  
plot\_sim\_multi, 17  
plot\_sim\_seq, 18  
plot\_sim\_time, 18  
plot\_sims, 15  
plot\_sum\_stats, 19  
preprocess\_dyads, 20  
preprocess\_text, 20  
print\_sim\_report, 21

radar\_sim, 22  
run\_example, 22

sem\_sim\_seq, 25  
semantic\_sim\_dyads, 24  
semantic\_similarity, 23

sent\_sim\_seq, 27  
sentiment\_sim\_dyads, 26  
sentiment\_similarity, 25  
structural\_sim\_dyads, 28  
structural\_similarity, 27  
style\_sim\_seq, 29  
stylistic\_sim\_dyads, 30  
stylistic\_similarity, 29

timing\_sim\_dyads, 31  
topic\_sim\_dyads, 32  
topic\_sim\_seq, 33  
topic\_similarity, 31