Package ‘chatRater’

March 3, 2025

Type Package

Title Rating and Evaluating Texts Using Large Language Models

Version 1.1.0

Date 2025-02-18

Maintainer Shiyang Zheng <Shiyang.Zheng@nottingham.ac.uk>

Description
Generates ratings and psycholinguistic metrics for textual stimuli using large language models.
It enables users to evaluate idioms and other language materials by combining con-
text, prompts, and stimulus inputs.
It supports multiple LLM APIs (such as 'OpenAI', 'DeepSeek', 'Anthropic', 'Co-
here', 'Google PaLM', and 'Ollama')
by allowing users to switch models with a single parameter. In addition to generating nu-
meric ratings,
'chatRater' provides functions for obtaining detailed psycholinguistic metrics including word fre-
quency (with optional corpus input),
lexical coverage (with customizable vocabulary size and test basis), Zipf metric, Levenshtein dis-
tance, and semantic transparency.

License MIT + file LICENSE

Encoding UTF-8

Imports openai, httr, jsonlite

Suggests testthat

NeedsCompilation no

Author Shiyang Zheng [aut, cre]

RoxygenNote 7.2.3

Repository CRAN

Date/Publication 2025-03-03 20:00:07 UTC

Contents

`chatRater-package` *Multi-Model LLM API Wrapper and Cognitive Experiment Utilities*

Description

This package provides functions to interact with multiple LLM APIs (e.g., 'OpenAI', 'DeepSeek', 'Anthropic', 'Cohere', 'Google PaLM', 'Ollama').

Additionally, several functions are provided that encapsulate LLM prompts to obtain various psycholinguistic metrics:

- **Word Frequency**: Number of occurrences (often log-transformed) of a word in a corpus.
- **Lexical Coverage**: Proportion of words in the stimulus that are found in a target vocabulary.
- **Zipf Metric**: The slope of the log-log frequency vs. rank distribution, per Zipf's law.
- **Levenshtein Distance (D)**: Minimum number of single-character edits required to transform one stimulus into another.
- **Semantic Transparency**: The degree to which the meaning of a compound or phrase is inferable from its parts.

Details

The LLM API functions allow users to generate ratings using various large language models. The cognitive experiment functions.

`generate_ratings` *Generate Ratings for a Stimulus Using LLM APIs*

Description

Generates ratings for a given stimulus by calling one of several LLM APIs.

Usage

```
generate_ratings(
  model = "gpt-3.5-turbo",
  stim = "kick the bucket",
  prompt = "You are a native English speaker.",
  question = "Please rate the following stim:",
  top_p = 1,
  temp = 0,
  n_iterations = 30,
  api_key = "",
  debug = FALSE
)
```

Arguments

`model`	A character string specifying the LLM model (e.g., "gpt-3.5-turbo", "deepseek-chat").
`stim`	A character string representing the stimulus (e.g., an idiom).
`prompt`	A character string for the system prompt (e.g., "You are a native English speaker.").
`question`	A character string for the user prompt (e.g., "Please rate the following stim:").
`top_p`	Numeric value for the probability mass (default 1).
`temp`	Numeric value for the temperature (default 0).
`n_iterations`	An integer indicating how many times to query the API.
`api_key`	API key as a character string.
`debug`	Logical; if TRUE, debug information is printed.

Details

This function supports multiple APIs. Branching is based on the `model` parameter.

Value

A data frame containing the stimulus, the rating, and the iteration number.

Examples

```
## Not run:
  ratings <- generate_ratings(
    model = "gpt-3.5-turbo",
    stim = "kick the bucket",
    prompt = "You are a native English speaker.",
    question = "Please rate the following stim:",
    n_iterations = 5,
    api_key = "your_api_key"
  )
  print(ratings)

## End(Not run)
```

generate_ratings_for_all — *Generate Ratings for Multiple Stimuli*

Description

Applies the `generate_ratings` function to a list of stimuli.

Usage

```
generate_ratings_for_all(
  model = "gpt-3.5-turbo",
  stim_list,
  prompt = "You are a native English speaker.",
  question = "Please rate the following stim:",
  top_p = 1,
  temp = 0,
  n_iterations = 30,
  api_key = "",
  debug = FALSE
)
```

Arguments

`model`	A character string specifying the LLM model.
`stim_list`	A character vector of stimuli.
`prompt`	A character string for the system prompt.
`question`	A character string for the user prompt.
`top_p`	Numeric value for the probability mass.
`temp`	Numeric value for the temperature.
`n_iterations`	An integer indicating how many iterations per stimulus.
`api_key`	API key as a character string.
`debug`	Logical; if TRUE, debug information is printed.

Value

A data frame with stimuli, ratings, and iteration numbers.

Examples

```
## Not run:
  all_ratings <- generate_ratings_for_all(
    model = "gpt-3.5-turbo",
    stim_list = c("kick the bucket", "spill the beans"),
    prompt = "You are a native English speaker.",
    question = "Please rate the following stim:",
    n_iterations = 5,
    api_key = "your_api_key"
  )
  print(all_ratings)

## End(Not run)
```

`get_levenshtein_d` *Get Levenshtein Distance (D)*

Description

Uses an LLM to compute the Levenshtein distance (D) between two linguistic stimuli.

Usage

```
get_levenshtein_d(
  stimulus1,
  stimulus2,
  model = "gpt-3.5-turbo",
  api_key = "",
  top_p = 1,
  temp = 0
)
```

Arguments

`stimulus1`	A character string representing the first text.
`stimulus2`	A character string representing the second text.
`model`	A character string specifying the LLM model (default "gpt-3.5-turbo").
`api_key`	API key as a character string.
`top_p`	Numeric value (default 1).
`temp`	Numeric value (default 0).

Details

Default definition: "Levenshtein distance is defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another."

Value

A numeric value representing the Levenshtein distance.

Examples

```
## Not run:
  lev_dist <- get_levenshtein_d("kitten", "sitting",
                                model = "gpt-3.5-turbo",
                                api_key = "your_api_key")
  cat("Levenshtein Distance:", lev_dist, "\n")

## End(Not run)
```

`get_lexical_coverage` *Get Lexical Coverage with Specified Vocabulary*

Description

Uses an LLM to obtain the lexical coverage (percentage) of a given text, taking into account a specified vocabulary size and the vocabulary test basis.

Usage

```
get_lexical_coverage(
  stimulus,
  vocab_size = 2000,
  vocab_test = "Vocabulary Levels Test",
  model = "gpt-3.5-turbo",
  api_key = "",
  top_p = 1,
  temp = 0
)
```

Arguments

`stimulus`	A character string representing the language material.
`vocab_size`	A numeric value indicating the size of the target vocabulary (e.g., 1000, 2000, 3000).
`vocab_test`	A character string specifying the vocabulary test used (e.g., "Vocabulary Levels Test", "LexTALE"). Users may provide any test name.
`model`	A character string specifying the LLM model (default "gpt-3.5-turbo").
`api_key`	API key as a character string.
`top_p`	Numeric value for probability mass (default 1).
`temp`	Numeric value for temperature (default 0).

Details

Default definition: "Lexical coverage is the proportion of words in a text that are included in a given vocabulary list. For this evaluation, assume a target vocabulary size of `vocab_size` words based on the `vocab_test`."

Value

A numeric value indicating the lexical coverage percentage.

Examples

```
## Not run:
  coverage <- get_lexical_coverage("The quick brown fox jumps over the lazy dog",
                                  vocab_size = 2000,
                                  vocab_test = "Vocabulary Levels Test",
                                  model = "gpt-3.5-turbo",
                                  api_key = "your_api_key")
  cat("Lexical Coverage (%):", coverage, "\n")

## End(Not run)
```

---

get_semantic_transparency   *Get Semantic Transparency Rating*

Description

Uses an LLM to obtain a semantic transparency rating for the given linguistic stimulus.

Usage

```
get_semantic_transparency(
  stimulus,
  model = "gpt-3.5-turbo",
  api_key = "",
  top_p = 1,
  temp = 0
)
```

Arguments

`stimulus`	A character string representing the language material.
`model`	A character string specifying the LLM model (default "gpt-3.5-turbo").
`api_key`	API key as a character string.
`top_p`	Numeric value (default 1).
`temp`	Numeric value (default 0).

Details

Default definition: "Semantic transparency is the degree to which the meaning of a compound or phrase can be inferred from its constituent parts."

Value

An integer rating (1-7) indicating the semantic transparency.

Examples

```
## Not run:
  sem_trans <- get_semantic_transparency("blackbird",
                                         model = "gpt-3.5-turbo",
                                         api_key = "your_api_key")
  cat("Semantic Transparency Rating:", sem_trans, "\n")

## End(Not run)
```

`get_word_frequency` *Get Word Frequency Information*

Description

Uses an LLM to obtain frequency information for a specified word position in the stimulus. The user can specify a corpus; if none is provided and corpus_source is "llm", the LLM will generate or assume a representative corpus.

Usage

```
get_word_frequency(
  stimulus,
  position = "first",
  corpus = "",
  corpus_source = ifelse(corpus != "", "provided", "llm"),
  model = "gpt-3.5-turbo",
  api_key = "",
  top_p = 1,
  temp = 0
)
```

Arguments

`stimulus`	A character string representing the language material.
`position`	A character string indicating which word to analyze ("first", "last", "each", or "total").
`corpus`	An optional character string representing the corpus to use for frequency analysis.
`corpus_source`	A character string, either "provided" or "llm". Default is "provided" if corpus is given, otherwise "llm".
`model`	A character string specifying the LLM model (default "gpt-3.5-turbo").
`api_key`	API key as a character string.
`top_p`	Numeric value for probability mass (default 1).
`temp`	Numeric value for temperature (default 0).

Details

Default definition: "Word frequency is defined as the number of times a word appears in a corpus (often log-transformed)."

Value

A numeric value representing the frequency (or a JSON string if "each" is specified).

Examples

```
## Not run:
  freq_first <- get_word_frequency("The quick brown fox jumps over the lazy dog",
                                   position = "first",
                                 corpus = "A sample corpus text with everyday language.",
                                   corpus_source = "provided",
                                   model = "gpt-3.5-turbo",
                                   api_key = "your_api_key")
  cat("Frequency (first word):", freq_first, "\n")

## End(Not run)
```

`get_zipf_metric` *Get Zipf Metric*

Description

Uses an LLM to estimate a Zipf-based metric (slope) for the given stimulus.

Usage

```
get_zipf_metric(
  stimulus,
  model = "gpt-3.5-turbo",
  api_key = "",
  top_p = 1,
  temp = 0
)
```

Arguments

`stimulus`	A character string representing the language material.
`model`	A character string specifying the LLM model (default "gpt-3.5-turbo").
`api_key`	API key as a character string.
`top_p`	Numeric value (default 1).
`temp`	Numeric value (default 0).

Details

Default definition: "Zipf's law states that word frequency is inversely proportional to its rank; the Zipf metric is the slope of the log-log frequency vs. rank plot."

Value

A numeric value representing the Zipf metric.

Examples

```
## Not run:
  zipf_metric <- get_zipf_metric("The quick brown fox jumps over the lazy dog",
                                 model = "gpt-3.5-turbo",
                                 api_key = "your_api_key")
  cat("Zipf Metric:", zipf_metric, "\n")

## End(Not run)
```

`llm_api_call` *Base LLM API Call Wrapper*

Description

Sends a prompt (with background academic definitions) to an LLM API (defaulting to 'OpenAI') and returns the LLM response.

Usage

```
llm_api_call(
  prompt_text,
  model = "gpt-3.5-turbo",
  api_key = "",
  top_p = 1,
  temp = 0
)
```

Arguments

`prompt_text`	A character string containing the user prompt.
`model`	A character string specifying the LLM model (default "gpt-3.5-turbo").
`api_key`	API key as a character string.
`top_p`	Numeric value for the probability mass (default 1).
`temp`	Numeric value for the sampling temperature (default 0).

Details

The system prompt includes academic definitions for word frequency, lexical coverage, Zipf's law, Levenshtein distance, and semantic transparency.

Value

A character string containing the LLM's response.

Examples

```
## Not run:
  response <- llm_api_call("Please provide a rating for the stimulus 'apple'.")
  cat(response)

## End(Not run)
```

Index