Package 'promptr'

May 9, 2026

Title Format and Complete Few-Shot LLM Prompts

Version 1.0.0

Description Format and submit few-shot prompts to OpenAI's Large Language Models (LLMs). Designed to be particularly useful for text classification problems in the social sciences. Methods are described in Ornstein, Blasingame, and Truscott (2024) <https://joeornstein.github.io/publications/ornstein-blasingame-truscott.pdf>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports curl, dplyr, glue, httr2, jsonlite, stringr

Depends R (>= 2.10)

LazyData true

URL https://github.com/joeornstein/promptr

BugReports https://github.com/joeornstein/promptr/issues

NeedsCompilation no

Author Joe Ornstein [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-5704-2098>)

Maintainer Joe Ornstein <jornstein@uga.edu>

Repository CRAN

Date/Publication 2024-08-23 11:30:02 UTC

Contents

`complete_chat` *Complete an LLM Chat*

Description

Submits a prompt to OpenAI's "Chat" API endpoint and formats the response into a string or tidy dataframe.

Usage

```
complete_chat(
  prompt,
  model = "gpt-3.5-turbo",
  openai_api_key = Sys.getenv("OPENAI_API_KEY"),
  max_tokens = 1,
  temperature = 0,
  seed = NULL,
  parallel = FALSE
)
```

Arguments

`prompt`	The prompt
`model`	Which OpenAI model to use. Defaults to 'gpt-3.5-turbo'
`openai_api_key`	Your API key. By default, looks for a system environment variable called "OPENAI_API_KEY" (recommended option). Otherwise, it will prompt you to enter the API key as an argument.
`max_tokens`	How many tokens (roughly 4 characters of text) should the model return? Defaults to a single token (next word prediction).
`temperature`	A numeric between 0 and 2 When set to zero, the model will always return the most probable next token. For values greater than zero, the model selects the next word probabilistically.
`seed`	An integer. If specified, the OpenAI API will "make a best effort to sample deterministically".
`parallel`	TRUE to submit API requests in parallel. Setting to FALSE can reduce rate limit errors at the expense of longer runtime.

Value

If max_tokens = 1, returns a dataframe with the 5 most likely next-word responses and their probabilities. If max_tokens > 1, returns a single string of text generated by the model.

Examples

```
## Not run:
format_chat('Are frogs sentient? Yes or No.') |> complete_chat()
format_chat('Write a haiku about frogs.') |> complete_chat(max_tokens = 100)

## End(Not run)
```

`complete_prompt` *Complete an LLM Prompt*

Description

Submits a text prompt to OpenAI's "Completion" API endpoint and formats the response into a string or tidy dataframe. (Note that, as of 2024, this endpoint is considered "Legacy" by OpenAI and is likely to be deprecated.)

Usage

```
complete_prompt(
  prompt,
  model = "gpt-3.5-turbo-instruct",
  openai_api_key = Sys.getenv("OPENAI_API_KEY"),
  max_tokens = 1,
  temperature = 0,
  seed = NULL,
  parallel = FALSE
)
```

Arguments

`prompt`	The prompt
`model`	Which OpenAI model to use. Defaults to 'gpt-3.5-turbo-instruct'
`openai_api_key`	Your API key. By default, looks for a system environment variable called "OPENAI_API_KEY" (recommended option). Otherwise, it will prompt you to enter the API key as an argument.
`max_tokens`	How many tokens (roughly 4 characters of text) should the model return? Defaults to a single token (next word prediction).
`temperature`	A numeric between 0 and 2 When set to zero, the model will always return the most probable next token. For values greater than zero, the model selects the next word probabilistically.
`seed`	An integer. If specified, the OpenAI API will "make a best effort to sample deterministically".
`parallel`	TRUE to submit API requests in parallel. Setting to FALSE can reduce rate limit errors at the expense of longer runtime.

Value

If max_tokens = 1, returns a dataframe with the 5 most likely next words and their probabilities. If max_tokens > 1, returns a single string of text generated by the model.

Examples

```
## Not run:
complete_prompt('I feel like a')
complete_prompt('Here is my haiku about frogs:',
                max_tokens = 100)

## End(Not run)
```

`format_chat` *Format a Chat Prompt*

Description

Format a chat prompt to submit to OpenAI's ChatGPT or GPT-4 (particularly useful for classification tasks).

Usage

```
format_chat(
  text,
  instructions = NA,
  examples = data.frame(),
  system_message = NA
)
```

Arguments

`text`	The text to be classified.
`instructions`	Instructions to be included at the beginning of the prompt (format them like you would format instructions to a human research assistant).
`examples`	A dataframe of "few-shot" examples. Must include one column called 'text' with the example text(s) and another column called "label" with the correct label(s).
`system_message`	An optional "system message" with high-level instructions (e.g. "You are a helpful research assistant.")

Value

Returns a series of messages formatted as a list object, which can be used as an input for promptr::complete_chat() or openai::create_chat_completion().

Examples

```
data(scotus_tweets_examples)

format_chat(text = "I am disappointed with this ruling.",
                   instructions = "Decide if the statement is Positive or Negative.",
                   examples = scotus_tweets_examples)
```

`format_prompt` *Format an LLM prompt*

Description

Format a text prompt for a Large Language Model. Particularly useful for few-shot text classification tasks. Note that if you are planning to use one of OpenAI's chat models, like ChatGPT or GPT-4, you will want to use the `format_chat()` function instead.

Usage

```
format_prompt(
  text,
  instructions = "",
  examples = data.frame(),
  template = "Text: {text}\nClassification: {label}",
  prompt_template = "{instructions}{examples}{input}",
  separator = "\n\n"
)
```

Arguments

`text`	The text to be classified. Can be a character vector or a single string.
`instructions`	Instructions to be included in the prompt (format them like you would format instructions to a human research assistant).
`examples`	A dataframe of "few-shot" examples. Must include one column called 'text' with the example text(s) and another column called "label" with the correct label(s).
`template`	The template for how examples and completions should be formatted, in `glue` syntax. If you are including few-shot examples in the prompt, this must contain the {text} and {label} placeholders.
`prompt_template`	The template for the entire prompt. Defaults to instructions, followed by few-shot examples, followed by the input to be classified.
`separator`	A character that separates examples. Defaults to two carriage returns.

Value

Returns a formatted prompt that can be used as input for `complete_prompt()` or `openai::create_completion()`.

Examples

```
data(scotus_tweets_examples)

format_prompt(text = "I am disappointed with this ruling.",
         instructions = "Decide if the sentiment of this statement is Positive or Negative.",
              examples = scotus_tweets_examples,
              template = "Statement: {text}\nSentiment: {label}")

format_prompt(text = 'I am sad about the Supreme Court',
              examples = scotus_tweets_examples,
              template = '"{text}" is a {label} statement',
              separator = '\n')
```

---

occupations *Occupations*

Description

This dataset contains 3,948 ballot designations from municipal elections in California. A random subset are hand-labeled as either "Working Class" or "Not Working Class" occupations.

Usage

```
data(occupations)
```

Format

A data frame with 3948 rows and 2 columns:

baldesig Ballot designation as it appears in the CEDA dataset

hand_coded A hand-coded occupation classification (for a random subset)

References

California Elections Data Archive (CEDA). https://hdl.handle.net/10211.3/210187

---

occupations_examples *Labelled Occupations*

Description

This dataset contains 9 example occupations along with a classification. These can be used as few-shot examples for classifying occupations in the `occupations` dataset.

Usage

```
data(occupations_examples)
```

Format

A data frame with 9 rows and 2 columns:

text The text of the ballot designation

label The hand-coded label (Working Class, Not Working Class, NA)

References

California Elections Data Archive (CEDA). https://hdl.handle.net/10211.3/210187

---

openai_api_key *Install an OPENAI API KEY in Your* .Renviron *File for Repeated Use*

Description

This function will add your OpenAI API key to your .Renviron file so it can be called securely without being stored in your code. After you have installed your key, it can be called any time by typing Sys.getenv("OPENAI_API_KEY") and will be automatically called in package functions. If you do not have an .Renviron file, the function will create one for you. If you already have an .Renviron file, the function will append the key to your existing file, while making a backup of your original file for disaster recovery purposes.

Usage

```
openai_api_key(key, overwrite = FALSE, install = FALSE)
```

Arguments

key	The API key provided to you from the OpenAI formated in quotes.
overwrite	If this is set to TRUE, it will overwrite an existing OPENAI_API_KEY that you already have in your .Renviron file.
install	if TRUE, will install the key in your .Renviron file for use in future sessions. Defaults to FALSE.

Value

No return value, called for side effects

Examples

```
## Not run:
openai_api_key("111111abc", install = TRUE)
# First time, reload your environment so you can use the key without restarting R.
readRenviron("~/.Renviron")
# You can check it with:
Sys.getenv("OPENAI_API_KEY")
```

```
## End(Not run)

## Not run:
# If you need to overwrite an existing key:
openai_api_key("111111abc", overwrite = TRUE, install = TRUE)
# First time, reload your environment so you can use the key without restarting R.
readRenviron("~/.Renviron")
# You can check it with:
Sys.getenv("OPENAI_API_KEY")

## End(Not run)
```

`scotus_tweets` *Tweets About The Supreme Court of the United States*

Description

This dataset contains 945 tweets referencing the US Supreme Court. Roughly half were collected on June 4, 2018 following the *Masterpiece Cakeshop* ruling, and the other half were collected on July 9, 2020 following the Court's concurrently released opinions in *Trump v. Mazars* and *Trump v. Vance*. Each tweet includes three independent human-coded sentiment scores (-1 to +1).

Usage

```
data(scotus_tweets)
```

Format

A data frame with 945 rows and 5 columns:

tweet_id A unique ID

text The text of the tweet

case An identifier denoting which Supreme Court ruling the tweet was collected after.

expert1, expert2, expert3 Hand-coded sentiment score (-1 = negative, 0 = neutral, 1 = positive)

Details

CONTENT WARNING: These texts come from social media, and many contain explicit or offensive language.

References

Ornstein et al. (2024). "How To Train Your Stochastic Parrot"

`scotus_tweets_examples` *Labelled Example Tweets About The Supreme Court of the United States*

Description

This dataset contains 12 example tweets referencing the Supreme Court along with a sentiment label. These can be used as few-shot prompt examples for classifying tweets in the `scotus_tweets` dataset.

Usage

```
data(scotus_tweets_examples)
```

Format

A data frame with 12 rows and 4 columns:

tweet_id A unique ID for each tweet

text The text of the tweet

case The case referenced in the tweet (Masterpiece Cakeshop or Trump v. Mazars)

label The "true" label (Positive, Negative, or Neutral)

References

Ornstein et al. (2023). "How To Train Your Stochastic Parrot"

Index