Package ‘rgpt3’

May 21, 2024

Title Making requests from R to the GPT model API

Version 1.0.1

Description With this package you can interact with the family of GPT models in two ways: making requests for completions (e.g., ask GPT-4 to write a novel, classify text, answer questions, etc.) and retrieving text embeddings representations (i.e., obtain a low-dimensional vector representation that allows for downstream analyses). The model also wraps the ChatGPT API. You need to authenticate with your own Open AI API key and all requests you make count towards you token quota. For completion requests and embeddings requests, two functions each allow you to send either sinlge requests (`rgpt_single()` and `rgpt_single_embedding()`) or send bunch requests where the vectorised structure is used (`rgpt()` and `rgpt_embeddings()`).

URL https://github.com/ben-aaron188/rgpt3

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports data.table,
httr

R topics documented:

rgpt *Makes bunch chat completion requests to the OpenAI API for all chat models*

Description

rgpt() is the package's (new) main function for all chat completion functionality and takes as input a vector of prompts and processes each prompt as per the defined parameters. It extends the rgpt_single() function to allow for bunch processing of requests to the Open AI GPT API.

Usage

```
rgpt(
  prompt_role_var,
  prompt_content_var,
  param_seed = NULL,
  id_var,
  param_output_type = "complete",
  param_model = "gpt-4o",
  param_max_tokens = 100,
  param_temperature = 1,
  param_top_p = 1,
  param_n = 1,
  param_stop = NULL,
  param_presence_penalty = 0,
  param_frequency_penalty = 0,
  param_logprobs = T
)
```

Arguments

prompt_role_var
: character vector that contains the role prompts to the GPT request. Must be one of 'system', 'assistant', 'user' (default), see https://platform.openai.com/docs/guides/chat

prompt_content_var
: character vector that contains the content prompts to the GPT request. This is the key instruction that the GPT model receives.

id_var
: (optional) character vector that contains the user-defined ids of the prompts. See details.

param_output_type
: character determining the output provided: "complete" (default), "text" or "meta"

param_model
: a character vector that indicates the GPT model to use; currently supported are: "gpt-3.5-turbo-0125', 'gpt-3.5-turbo', 'gpt-3.5-turbo-1106', 'gpt-3.5-turbo-16k', 'gpt-3.5-turbo-0613', 'gpt-3.5-turbo-16k-0613', 'gpt-4', 'gpt-4-0125-preview', 'gpt-4-turbo-preview', 'gpt-4-turbo-2024-04-09', 'gpt-4-turbo', 'gpt-4o' (default)

param_max_tokens
: numeric (default: 100) indicating the maximum number of tokens that the completion request should return (from the official API documentation: *The maximum number of tokens allowed for the generated answer. By default, the number of tokens the model can return will be (4096 - prompt tokens).*)

`param_temperature`
: numeric (default: 1.0) specifying the sampling strategy of the possible completions (from the official API documentation: *What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. We generally recommend altering this or* `top_p` *but not both.*)

`param_top_p`
: numeric (default: 1) specifying sampling strategy as an alternative to the temperature sampling (from the official API documentation: *An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. We generally recommend altering this or* `temperature` *but not both.*)

`param_n`
: numeric (default: 1) specifying the number of completions per request (from the official API documentation: *How many chat completion choices to generate for each input message.* **Note: Because this parameter generates many completions, it can quickly consume your token quota.** *Use carefully and ensure that you have reasonable settings for max_tokens and stop.*)

`param_stop`
: character or character vector (default: NULL) that specifies after which character value when the completion should end (from the official API documentation: *Up to 4 sequences where the API will stop generating further tokens.*)

`param_presence_penalty`
: numeric (default: 0) between -2.00 and +2.00 to determine the penalisation of repetitiveness if a token already exists (from the official API documentation: *Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics.*). See also: https://beta.openai.com/docs/api-reference/parameter-details

`param_frequency_penalty`
: numeric (default: 0) between -2.00 and +2.00 to determine the penalisation of repetitiveness based on the frequency of a token in the text already (from the official API documentation: *Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim.*). See also: https://beta.openai.com/docs/api-reference/parameter-details #' @param param_logprobs boolean (default: TRUE) from the official API documentation: : *whether to return log probabilities of the output tokens or not. If true, returns the log probabilities of each output token returned in the content of message.* Will be returned in the output list at slot 3.

`seed`
: numeric (optional) the seed to control reproducibility of the completions. If NULL, no seed will be used and results may differ at each completion. See: https://platform.openai.com/docs/api-reference/chat/create#chat-create-seed

Details

The easiest (and intended) use case for this function is to create a data.frame or data.table with variables that contain the prompts to be requested from the GPT models and a prompt id (see examples below). For a general guide on the chat completion requests, see https://platform.openai.com/docs/guides/chat/chat-completions-beta. This function provides you with a general R wrapper to send requests with the full range of request parameters as detailed on https://platform.openai.com/docs/api-reference/chat/create and reproduced below.

If `id_var` is not provided, the function will use `prompt_1` ... `prompt_n` as id variable.

Parameters not included/supported:

- logit_bias: https://platform.openai.com/docs/api-reference/chat/create#chat/create-logit_bias
- stream: https://platform.openai.com/docs/api-reference/chat/create#chat/create-stream

Value

A list with two data tables (if output_type is the default "complete"): [1] contains the data table with the columns n (= the mo. of n responses requested), prompt_role (= the role that was set for the prompt), prompt_content (= the content that was set for the prompt), gpt_role (= the role that the GPT assumed in the chat completion) and gpt_content (= the content that the GPT model provided with its assumed role in the chat completion). [2] contains the meta information of the request, including the request id, the parameters of the request and the token usage of the prompt (tok_usage_prompt), the completion (tok_usage_completion), the total usage (tok_usage_total), the id (= the provided id_var or its default alternative), and the system fingerprint (system_fingerprint) (for reproducibility related to the seed). [3] contains the tokens of the completion (per n requests if applicable) and the corresponding log probabilities.

If output_type is "text", only the data table in slot [1] is returned.

If output_type is "meta", only the data table in slot [2] is returned.

If output_type is "logprobs", only the data table in slot [3] is returned.

Examples

```
# First authenticate with your API key via `rgpt_authenticate('pathtokey')`

# Once authenticated:
# Assuming you have a data.table with 3 different prompts:
dt_prompts = data.table::data.table('prompts_content' = c('What is the meaning if life?', 'Write a tweet about
    , 'prompts_role' = rep('user', 4)
    , 'prompt_id' = c(LETTERS[1:4]))
rgpt(prompt_role_var = dt_prompts$prompts_role
   , prompt_content_var = dt_prompts$prompts_content
   , id_var = dt_prompts$prompt_id)

## With more controls
rgpt(prompt_role_var = dt_prompts$prompts_role
    , prompt_content_var = dt_prompts$prompts_content
    , id_var = dt_prompts$prompt_id
    , param_max_tokens = 50
    , param_temperature = 0.5
    , param_n = 5)

## Reproducible example (with seed)
rgpt(prompt_role_var = dt_prompts$prompts_role
    , prompt_content_var = dt_prompts$prompts_content
    , param_seed = 42
    , id_var = dt_prompts$prompt_id
    , param_max_tokens = 50
    , param_temperature = 0
    , param_n = 3)
```

rgpt_authenticate *Set up the authentication with your API key*

Description

Access to GPT's functions requires an API key that you obtain from https://openai.com/api/. rgpt_authenticate() looks for your API key in a file that you provide the path to and ensures you can connect to the models. rgpt_endsession() overwrites your API key *for this session* (it is recommended that you run this when you are done). check_apikey_form() is a simple check if any information has been provided at all.

Usage

```
rgpt_authenticate(path)
```

Arguments

path	The file path to the API key

Details

The easiest way to store you API key is in a .txt file with *only* the API key in it (without quotation marks or other common string indicators). rgpt_authenticate() reads the single file you point it to and retrieves the content as authentication key for all requests.

Value

A confirmation message

Examples

```
# Starting a session:
rgpt_authenticate(path = './YOURPATH/access_key.txt')
# After you are finished:
rgpt_endsession()
```

rgpt_embeddings *Retrieves text embeddings for character input from a vector from OpenAI's GPT API*

Description

rgpt_embeddings() extends the single embeddings function rgpt_single_embedding() to allow for the processing of a whole vector

Usage

```
rgpt_embeddings(input_var, id_var, param_model = "text-embedding-3-large")
```

Arguments

`input_var` character vector that contains the texts for which you want to obtain text embeddings from the specified GPT model

`id_var` (optional) character vector that contains the user-defined ids of the prompts. See details.

`param_model` a character vector that indicates the embedding model; one of "text-embedding-3-large" (default), "text-embedding-3-small", "text-embedding-ada-002"

Details

The returned data.table contains the column `id` which indicates the text id (or its generic alternative if not specified) and the columns `dim_1` ... `dim_{max}`, where `max` is the length of the text embeddings vector that the different models (see below) return. For the default "Embedding V3 large" model, these are 3072 dimensions (i.e., `dim_1`... `dim_3072`).

The function supports the text similarity embeddings for the three GPT embeddings models as specified in the parameter list.

- Embedding V3 large `text-embedding-3-large` (3072 dimensions)
- Embedding V3 small `text-embedding-3-small` (1536 dimensions)
- Ada 2nd generation `text-embedding-ada-002` (1536 dimensions)

Note that the dimension size (= vector length), speed and associated costs differ considerably.

These vectors can be used for downstream tasks such as (vector) similarity calculations.

Value

A data.table with the embeddings as separate columns; one row represents one input text. See details.

Examples

```
# First authenticate with your API key via `rgpt_authenticate('pathtokey')`

# Use example data:
## The data below were generated with the `rgpt_single()` function as follows:
##### DO NOT RUN #####
# travel_blog_data = rgpt_single(prompt_content = "Write a travel blog about a dog's journey through the UK:", t
##### END DO NOT RUN #####

# You can load these data with:
data("travel_blog_data") # the dataset contains 10 completions for the above request

## Obtain text embeddings for the completion texts:
emb_travelblogs = rgpt_embeddings(input_var = travel_blog_data$gpt_content)
dim(emb_travelblogs)
```

rgpt_single *Makes a single chat completion request to the GPT API for all chat models*

Description

rgpt_single() sends a single chat completion request to the Open AI GPT API. This function allows you to specify the role and content for your API call.

Usage

```
rgpt_single(
  prompt_role = "user",
  prompt_content,
  seed = NULL,
  model = "gpt-4o",
  output_type = "complete",
  max_tokens = 100,
  temperature = 1,
  top_p = 1,
  n = 1,
  stop = NULL,
  presence_penalty = 0,
  frequency_penalty = 0,
  logprobs = T
)
```

Arguments

prompt_role	character (default: 'user') that contains the role for the prompt message in the GPT (chat) message format. Must be one of 'system', 'assistant', 'user' (default), see https://platform.openai.com/docs/guides/chat
prompt_content	character that contains the content for the prompt message in the GPT (chat) message format, see https://platform.openai.com/docs/guides/chat. This is the key instruction that the GPT model receives.
seed	numeric (optional) the seed to control reproducibility of the completions. If NULL, no seed will be used and results may differ at each completion. See: https://platform.openai.com/docs/api-reference/chat/create#chat-create-seed
model	a character vector that indicates the GPT model to use; currently supported are: 'gpt-3.5-turbo-0125', 'gpt-3.5-turbo', 'gpt-3.5-turbo-1106', 'gpt-3.5-turbo-16k', 'gpt-3.5-turbo-0613', 'gpt-3.5-turbo-16k-0613', 'gpt-4', 'gpt-4-0125-preview', 'gpt-4-turbo-preview', 'gpt-4-turbo-2024-04-09', 'gpt-4-turbo', 'gpt-4o' (default)
output_type	character determining the output provided: "complete" (default), "text" or "meta"
max_tokens	numeric (default: 100) indicating the maximum number of tokens that the completion request should return (from the official API documentation: *The maximum number of tokens allowed for the generated answer. By default, the number of tokens the model can return will be (4096 - prompt tokens).*)
temperature	numeric (default: 1.0) specifying the sampling strategy of the possible completions (from the official API documentation: *What sampling temperature to use,*

between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. We generally recommend altering this or `top_p` *but not both.*)

`top_p` numeric (default: 1) specifying sampling strategy as an alternative to the temperature sampling (from the official API documentation: *An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. We generally recommend altering this or* `temperature` *but not both.*)

`n` numeric (default: 1) specifying the number of completions per request (from the official API documentation: *How many chat completion choices to generate for each input message.* **Note: Because this parameter generates many completions, it can quickly consume your token quota.** *Use carefully and ensure that you have reasonable settings for max_tokens and stop.*)

`stop` character or character vector (default: NULL) that specifies after which character value when the completion should end (from the official API documentation: *Up to 4 sequences where the API will stop generating further tokens.*)

`presence_penalty`

numeric (default: 0) between -2.00 and +2.00 to determine the penalisation of repetitiveness if a token already exists (from the official API documentation: *Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics.*). See also: https://beta.openai.com/docs/api-reference/parameter-details

`frequency_penalty`

numeric (default: 0) between -2.00 and +2.00 to determine the penalisation of repetitiveness based on the frequency of a token in the text already (from the official API documentation: *Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim.*). See also: https://beta.openai.com/docs/api-reference/parameter-details

`logprobs` boolean (default: TRUE) from the official API documentation: : *whether to return log probabilities of the output tokens or not. If true, returns the log probabilities of each output token returned in the content of message.* Will be returned in the output list at slot 3.

Details

For a general guide on the completion requests, see https://platform.openai.com/docs/api-reference/chat. This function provides you with an R wrapper to send requests with the full range of request parameters as detailed on https://beta.openai.com/docs/api-reference/completions and reproduced below.

Parameters not included/supported:

- `logit_bias`: https://platform.openai.com/docs/api-reference/chat/create#chat/create-logit_bias
- `stream`: https://platform.openai.com/docs/api-reference/chat/create#chat/create-stream

Value

A list with three data tables (if `output_type` is the default "complete"): [1] contains the data table with the columns `n` (= the mo. of `n` responses requested), `prompt_role` (= the role that was set for the prompt), prompt_content (= the content that was set for the prompt), rgpt_role (= the role that the GPT model assumed in the chat completion) and rgpt_content (= the content that the GPT model provided with its assumed role in the chat completion). [2] contains the meta information of the request, including the request id, the parameters of the request and the token usage of the prompt (tok_usage_prompt), the completion (tok_usage_completion), the total usage (tok_usage_total), and the system fingerprint (system_fingerprint) (for reproducibility related to the seed). [3] contains the tokens of the completion (per n requests if applicable) and the corresponding log probabilities.

If output_type is "text", only the data table in slot [1] is returned.

If output_type is "meta", only the data table in slot [2] is returned.

If output_type is "logprobs", only the data table in slot [3] is returned.

Examples

```
# First authenticate with your API key via `rgpt_authenticate('pathtokey')`

# Once authenticated:

## Simple request with defaults:
rgpt_single(prompt_content = 'You are a teacher: explain to me what science is')

## Instruct a GPT model to write ten research ideas of max. 150 tokens with some controls:
rgpt_single(prompt_role = 'user', prompt_content = 'Write a research idea about using text data to understand hu
   , temperature = 0.8
   , n = 10
   , max_tokens = 150)

## For fully reproducible results, we need to set a seed, e.g., `seed = 42`, e.g.:
rgpt_single(prompt_content = 'Finish this sentence:/n There is no easier way to learn R than'
    , temperature = 0.7
    , seed = 42
    , max_tokens = 50)
```

rgpt_single_embedding *Obtains text embeddings for a single character (string) from OpenAI's GPT API*

Description

rgpt_single_embedding() sends a single embedding request to the Open AI GPT API.

Usage

```
rgpt_single_embedding(input, model = "text-embedding-3-large")
```

Arguments

input — character that contains the text for which you want to obtain text embeddings from the specified GPT model

model — a character vector that indicates the embedding model; one of "text-embedding-3-large" (default), "text-embedding-3-small", "text-embedding-ada-002"

Details

The returned data.table contains the column `id` which indicates the text id (or its generic alternative if not specified) and the columns `dim_1` ... `dim_{max}`, where `max` is the length of the text embeddings vector that the different models (see below) return. For the default "Embedding V3 large" model, these are 3072 dimensions (i.e., `dim_1`... `dim_3072`).

The function supports the text similarity embeddings for the three GPT embeddings models as specified in the parameter list.

- Embedding V3 large `text-embedding-3-large` (3072 dimensions)
- Embedding V3 small `text-embedding-3-small` (1536 dimensions)
- Ada 2nd generation `text-embedding-ada-002` (1536 dimensions)

Note that the dimension size (= vector length), speed and associated costs differ considerably.

These vectors can be used for downstream tasks such as (vector) similarity calculations.

Value

A numeric vector (= the embedding vector)

Examples

```
# First authenticate with your API key via `rgpt_authenticate('pathtokey')`

# Once authenticated:

## Simple request with defaults:
sample_string = "London is one of the most liveable cities in the world. The city is always full of energy and peo
rgpt_single_embedding(input = sample_string)

## Change the model:
rgpt_single_embedding(input = sample_string
  , model = 'text-embedding-ada-002')
```

`rgpt_test_completion` *Make a test request to the GPT API*

Description

`rgpt_test_completion()` sends a basic completion request to the Open AI GPT API.

Usage

```
rgpt_test_completion(verbose = T)
```

Arguments

`verbose`	(boolean) if TRUE prints the actual prompt and GPT completion of the test request (default: TRUE).

Value

A message of success or failure of the connection.

Examples

```
rgpt_test_completion()
```

`to_numeric` *Convert character vector of numeric values into a numeric vector*

Description

Converts a character vector of numeric values into a numeric vector

Usage

```
to_numeric(x)
```

Arguments

`x`	a character vector of numeric values

Value

A numeric vector

Examples

```
to_numeric('12312')
```

`url.completions` *Contains the package's base URLs*

Description

These are the base URLs for the `rgpt3` package. Do not change these!

Usage

```
url.completions
```

Format

An object of class `character` of length 1.

Index