

Package ‘rgpt3’

March 5, 2023

Title Making requests from R to the GPT-3 API and ChatGPT

Version 0.4

Description

With this package you can interact with the powerful GPT-3/GPT-3.5 models in two ways: making requests for completions (e.g., ask GPT-3 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). You can also interact with ChatGPT. You need to authenticate with your own Open AI API key and all requests you make count towards your token quota. For completion requests and embeddings requests, two functions each allow you to send either single requests (`gpt3_single_request()` and `gpt3_single_embedding()`) or send bunch requests where the vectorised structure is used (`gpt3_requests()` and `gpt3_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:

<code>chatgpt</code>	2
<code>chatgpt_single</code>	4
<code>gpt3_authenticate</code>	7
<code>gpt3_completions</code>	8
<code>gpt3_embeddings</code>	11
<code>gpt3_single_completion</code>	12
<code>gpt3_single_embedding</code>	15
<code>gpt3_test_completion</code>	16
<code>to_numeric</code>	16
<code>url.completions</code>	17

Index	18
--------------	-----------

chatgpt

Makes bunch chat completion requests to the ChatGPT API

Description

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

Usage

```
chatgpt(
  prompt_role_var,
  prompt_content_var,
  id_var,
  param_output_type = "complete",
  param_model = "gpt-3.5-turbo",
  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
)
```

Arguments

prompt_role_var	character vector that contains the role prompts to the ChatGPT 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 ChatGPT request. This is the key instruction that ChatGPT 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 ChatGPT model to use; one of "gpt-3.5-turbo" (default), "gpt-3.5-turbo-0301"
param_max_tokens	numeric (default: 100) indicating the maximum number of tokens that the completion request should return (from the official API documentation: <i>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).</i>)
param_temperature	numeric (default: 1.0) specifying the sampling strategy of the possible completions (from the official API documentation: <i>What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random,</i>

	<i>while lower values like 0.2 will make it more focused and deterministic. We generally recommend altering this or top_p but not both.)</i>
param_top_p	numeric (default: 1) specifying sampling strategy as an alternative to the temperature sampling (from the official API documentation: <i>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.</i>)
param_n	numeric (default: 1) specifying the number of completions per request (from the official API documentation: <i>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.</i>)
param_stop	character or character vector (default: NULL) that specifies after which character value when the completion should end (from the official API documentation: <i>Up to 4 sequences where the API will stop generating further tokens.</i>)
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: <i>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.</i>). 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: <i>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.</i>). See also: https://beta.openai.com/docs/api-reference/parameter-details

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 ChatGPT 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 an 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 no. of n responses requested), prompt_role (= the role that was set for the prompt), prompt_content (= the content that was set for the prompt), chatgpt_role

(= the role that ChatGPT assumed in the chat completion) and `chatgpt_content` (= the content that ChatGPT 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 id (= the provided `id_var` or its default alternative).

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.

Examples

```
# First authenticate with your API key via `gpt3_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', 3)
  , 'prompt_id' = c(LETTERS[1:3]))
chatgpt(prompt_role_var = dt_prompts$prompts_role
  , prompt_content_var = dt_prompts$prompts_content
  , id_var = dt_prompts$prompt_id)

## With more controls
chatgpt(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 (deterministic approach)
chatgpt(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
  , param_n = 3)
```

chatgpt_single

Makes a single chat completion request to the ChatGPT API

Description

`chatgpt_single()` sends a single **chat completion request** to the Open AI GPT API. Doing so, makes this equivalent to the sending single completion requests with `gpt3_single_completion()`. You can see the notes on chat vs completion requests here: <https://platform.openai.com/docs/guides/chat/chat-vs-completions>. This function allows you to specify the role and content for your API call.

Usage

```
chatgpt_single(
  prompt_role = "user",
  prompt_content,
  model = "gpt-3.5-turbo",
  output_type = "complete",
  max_tokens = 100,
  temperature = 1,
  top_p = 1,
  n = 1,
  stop = NULL,
  presence_penalty = 0,
  frequency_penalty = 0
)
```

Arguments

prompt_role	character (default: 'user') that contains the role for the prompt message in the ChatGPT 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 ChatGPT message format, see https://platform.openai.com/docs/guides/chat . This is the key instruction that ChatGPT receives.
model	a character vector that indicates the ChatGPT model to use; one of "gpt-3.5-turbo" (default), "gpt-3.5-turbo-0301"
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: <i>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).</i>)
temperature	numeric (default: 1.0) specifying the sampling strategy of the possible completions (from the official API documentation: <i>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.</i>)
top_p	numeric (default: 1) specifying sampling strategy as an alternative to the temperature sampling (from the official API documentation: <i>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.</i>)
n	numeric (default: 1) specifying the number of completions per request (from the official API documentation: <i>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.</i>)
stop	character or character vector (default: NULL) that specifies after which character value when the completion should end (from the official API documentation: <i>Up to 4 sequences where the API will stop generating further tokens.</i>)

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>

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 two data tables (if output_type is the default "complete"): [1] contains the data table with the columns n (= the no. of n responses requested), prompt_role (= the role that was set for the prompt), prompt_content (= the content that was set for the prompt), chatgpt_role (= the role that ChatGPT assumed in the chat completion) and chatgpt_content (= the content that ChatGPT 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) and the total usage (tok_usage_total).

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.

Examples

```
# First authenticate with your API key via `gpt3_authenticate('pathtoken')`
```

```
# Once authenticated:
```

```
## Simple request with defaults:
```

```
chatgpt_single(prompt_content = 'You are a teacher: explain to me what science is')
```

```
## Instruct ChatGPT to write ten research ideas of max. 150 tokens with some controls:
```

```
chatgpt_single(prompt_role = 'user', prompt_content = 'Write a research idea about using text data to understand',
  , temperature = 0.8
  , n = 10)
```

```
, max_tokens = 150)

## For fully reproducible results, we need `temperature = 0`, e.g.:
chatgpt_single(prompt_content = 'Finish this sentence:\n There is no easier way to learn R than'
, temperature = 0.0
, max_tokens = 50)
```

gpt3_authenticate

Set up the authentication with your API key

Description

Access to GPT-3's functions requires an API key that you obtain from <https://openai.com/api/>. gpt3_authenticate() looks for your API key in a file that you provide the path to and ensures you can connect to the models. gpt3_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

```
gpt3_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). gpt3_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:
gpt3_authenticate(path = './YOURPATH/access_key.txt')
# After you are finished:
gpt3_endsession()
```

gpt3_completions

Makes bunch completion requests to the GPT-3 API

Description

`gpt3_completions()` is the package's main function for requests and takes as input a vector of prompts and processes each prompt as per the defined parameters. It extends the `gpt3_single_completion()` function to allow for bunch processing of requests to the Open AI GPT-3 API.

Usage

```
gpt3_completions(
  prompt_var,
  id_var,
  param_output_type = "complete",
  param_model = "text-davinci-003",
  param_suffix = NULL,
  param_max_tokens = 100,
  param_temperature = 0.9,
  param_top_p = 1,
  param_n = 1,
  param_logprobs = NULL,
  param_stop = NULL,
  param_presence_penalty = 0,
  param_frequency_penalty = 0,
  param_best_of = 1
)
```

Arguments

<code>prompt_var</code>	character vector that contains the prompts to the GPT-3 request
<code>id_var</code>	(optional) character vector that contains the user-defined ids of the prompts. See details.
<code>param_output_type</code>	character determining the output provided: "complete" (default), "text" or "meta"
<code>param_model</code>	a character vector that indicates the model to use; one of "text-davinci-003" (default), "text-davinci-002", "text-davinci-001", "text-curie-001", "text-babbage-001" or "text-ada-001"
<code>param_suffix</code>	character (default: NULL) (from the official API documentation: <i>The suffix that comes after a completion of inserted text</i>)
<code>param_max_tokens</code>	numeric (default: 100) indicating the maximum number of tokens that the completion request should return (from the official API documentation: <i>The maximum number of tokens to generate in the completion. The token count of your prompt plus max_tokens cannot exceed the model's context length. Most models have a context length of 2048 tokens (except for the newest models, which support 4096)</i>)
<code>param_temperature</code>	numeric (default: 0.9) specifying the sampling strategy of the possible completions (from the official API documentation: <i>What sampling temperature to use.</i>

Higher values means the model will take more risks. Try 0.9 for more creative applications, and 0 (argmax sampling) for ones with a well-defined answer. 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: <i>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.</i>)
param_n	numeric (default: 1) specifying the number of completions per request (from the official API documentation: <i>How many completions to generate for each prompt. 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.</i>)
param_logprobs	numeric (default: NULL) (from the official API documentation: <i>Include the log probabilities on the logprobs most likely tokens, as well the chosen tokens. For example, if logprobs is 5, the API will return a list of the 5 most likely tokens. The API will always return the logprob of the sampled token, so there may be up to logprobs+1 elements in the response. The maximum value for logprobs is 5. If you need more than this, please go to https://help.openai.com/en/ and describe your use case.</i>)
param_stop	character or character vector (default: NULL) that specifies after which character value when the completion should end (from the official API documentation: <i>Up to 4 sequences where the API will stop generating further tokens. The returned text will not contain the stop sequence.</i>)
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: <i>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.</i>). 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: <i>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.</i>). See also: https://beta.openai.com/docs/api-reference/parameter-details
param_best_of	numeric (default: 1) that determines the space of possibilities from which to select the completion with the highest probability (from the official API documentation: <i>Generates best_of completions server-side and returns the "best" (the one with the highest log probability per token).</i>). See details.

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 GPT-3 and a prompt id (see examples below). For a general guide on the completion requests, see <https://beta.openai.com/docs/guides/completion>. 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.

For the `best_of` parameter: The `gpt3_single_completion()` (which is used here in a vectorised manner) handles the issue that `best_of` must be greater than `n` by setting `if(best_of <= n){ best_of = n}`.

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

Parameters not included/supported:

- `logit_bias`: <https://beta.openai.com/docs/api-reference/completions/create#completions/create-logit-bias>
- `echo`: <https://beta.openai.com/docs/api-reference/completions/create#completions/create-echo>
- `stream`: <https://beta.openai.com/docs/api-reference/completions/create#completions/create-stream>

Value

A list with two data tables (if `param_output_type` is the default "complete"): [1] contains the data table with the columns `n` (= the no. of `n` responses requested), `prompt` (= the prompt that was sent), `gpt3` (= the completion as returned from the GPT-3 model) and `id` (= the provided `id_var` or its default alternative). [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 `id` (= the provided `id_var` or its default alternative).

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.

Examples

```
# First authenticate with your API key via `gpt3_authenticate('pathtokey')`

# Once authenticated:
# Assuming you have a data.table with 3 different prompts:
dt_prompts = data.table::data.table('prompts' = c('What is the meaning if life?', 'Write a tweet about London:'))
gpt3_completions(prompt_var = dt_prompts$prompts
  , id_var = dt_prompts$prompt_id)

## With more controls
gpt3_completions(prompt_var = dt_prompts$prompts
  , id_var = dt_prompts$prompt_id
  , param_max_tokens = 50
  , param_temperature = 0.5
  , param_n = 5)

## Reproducible example (deterministic approach)
gpt3_completions(prompt_var = dt_prompts$prompts
  , id_var = dt_prompts$prompt_id
  , param_max_tokens = 50
  , param_temperature = 0.0)

## Changing the GPT-3 model
gpt3_completions(prompt_var = dt_prompts$prompts
  , id_var = dt_prompts$prompt_id
```

```
, param_model = 'text-babbage-001'
, param_max_tokens = 50
, param_temperature = 0.4)
```

gpt3_embeddings	<i>Retrieves text embeddings for character input from a vector from the GPT-3 API</i>
-----------------	---

Description

gpt3_embeddings() extends the single embeddings function gpt3_single_embedding() to allow for the processing of a whole vector

Usage

```
gpt3_embeddings(input_var, id_var, param_model = "text-embedding-ada-002")
```

Arguments

input_var	character vector that contains the texts for which you want to obtain text embeddings from the GPT-3 model #' @param 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-ada-002" (default), "text-similarity-ada-001", "text-similarity-curie-001", "text-similarity-babbage-001", "text-similarity-davinci-001"

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 "Ada 2nd gen." model, these are 1536 dimensions (i.e., dim_1... dim_1536).

The function supports the text similarity embeddings for the **five GPT-3 embeddings models** as specified in the parameter list. It is strongly advised to use the second generation model "text-embedding-ada-002". The main difference between the five models is the size of the embedding representation as indicated by the vector embedding size and the pricing. The newest model (default) is the fastest, cheapest and highest quality one.

- Ada 2nd generation text-embedding-ada-002 (1536 dimensions)
- Ada (1024 dimensions)
- Babbage (2048 dimensions)
- Curie (4096 dimensions)
- Davinci (12288 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 `gpt3_authenticate('pathtokey')`

# Use example data:
## The data below were generated with the `gpt3_single_request()` function as follows:
##### DO NOT RUN #####
# travel_blog_data = gpt3_single_request(prompt_input = "Write a travel blog about a dog's journey through the l
##### 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 = gpt3_embeddings(input_var = travel_blog_data$gpt3)
dim(emb_travelblogs)
```

gpt3_single_completion

Makes a single completion request to the GPT-3 API

Description

gpt3_single_completion() sends a single **completion request** to the Open AI GPT-3 API.

Usage

```
gpt3_single_completion(
  prompt_input,
  model = "text-davinci-003",
  output_type = "complete",
  suffix = NULL,
  max_tokens = 100,
  temperature = 0.9,
  top_p = 1,
  n = 1,
  logprobs = NULL,
  stop = NULL,
  presence_penalty = 0,
  frequency_penalty = 0,
  best_of = 1
)
```

Arguments

prompt_input	character that contains the prompt to the GPT-3 request
model	a character vector that indicates the model to use; one of "text-davinci-003" (default), "text-davinci-002", "text-davinci-001", "text-curie-001", "text-babbage-001" or "text-ada-001"
output_type	character determining the output provided: "complete" (default), "text" or "meta"
suffix	character (default: NULL) (from the official API documentation: <i>The suffix that comes after a completion of inserted text</i>)

max_tokens	numeric (default: 100) indicating the maximum number of tokens that the completion request should return (from the official API documentation: <i>The maximum number of tokens to generate in the completion. The token count of your prompt plus max_tokens cannot exceed the model's context length. Most models have a context length of 2048 tokens (except for the newest models, which support 4096)</i>)
temperature	numeric (default: 0.9) specifying the sampling strategy of the possible completions (from the official API documentation: <i>What sampling temperature to use. Higher values means the model will take more risks. Try 0.9 for more creative applications, and 0 (argmax sampling) for ones with a well-defined answer. We generally recommend altering this or top_p but not both.</i>)
top_p	numeric (default: 1) specifying sampling strategy as an alternative to the temperature sampling (from the official API documentation: <i>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.</i>)
n	numeric (default: 1) specifying the number of completions per request (from the official API documentation: <i>How many completions to generate for each prompt. 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.</i>)
logprobs	numeric (default: NULL) (from the official API documentation: <i>Include the log probabilities on the logprobs most likely tokens, as well the chosen tokens. For example, if logprobs is 5, the API will return a list of the 5 most likely tokens. The API will always return the logprob of the sampled token, so there may be up to logprobs+1 elements in the response. The maximum value for logprobs is 5. If you need more than this, please go to https://help.openai.com/en/ and describe your use case.</i>)
stop	character or character vector (default: NULL) that specifies after which character value when the completion should end (from the official API documentation: <i>Up to 4 sequences where the API will stop generating further tokens. The returned text will not contain the stop sequence.</i>)
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: <i>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.</i>). 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: <i>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.</i>). See also: https://beta.openai.com/docs/api-reference/parameter-details
best_of	numeric (default: 1) that determines the space of possibilities from which to select the completion with the highest probability (from the official API documentation: <i>Generates best_of completions server-side and returns the "best" (the one with the highest log probability per token).</i>). See details.

Details

For a general guide on the completion requests, see <https://beta.openai.com/docs/guides/completion>. 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.

For the `best_of` parameter: When used with `n`, `best_of` controls the number of candidate completions and `n` specifies how many to return – `best_of` must be greater than `n`. Note that this is handled by the wrapper automatically if (`best_of <= n`) `best_of = n`.

Parameters not included/supported:

- `logit_bias`: <https://beta.openai.com/docs/api-reference/completions/create#completions/create-logit-bias>
- `echo`: <https://beta.openai.com/docs/api-reference/completions/create#completions/create-echo>
- `stream`: <https://beta.openai.com/docs/api-reference/completions/create#completions/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 no. of `n` responses requested), `prompt` (= the prompt that was sent), and `gpt3` (= the completion as returned from the GPT-3 model). [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`) and the total usage (`tok_usage_total`).

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.

Examples

```
# First authenticate with your API key via `gpt3_authenticate('pathtokey')`

# Once authenticated:

## Simple request with defaults:
gpt3_single_completion(prompt_input = 'How old are you?')

## Instruct GPT-3 to write ten research ideas of max. 150 tokens with some controls:
gpt3_single_completion(prompt_input = 'Write a research idea about using text data to understand human behaviour',
  , temperature = 0.8
  , n = 10
  , max_tokens = 150)

## For fully reproducible results, we need `temperature = 0`, e.g.:
gpt3_single_completion(prompt_input = 'Finish this sentence:\n There is no easier way to learn R than'
  , temperature = 0.0
  , max_tokens = 50)

## The same example with a different GPT-3 model:
gpt3_single_completion(prompt_input = 'Finish this sentence:\n There is no easier way to learn R than'
  , model = 'text-babbage-001'
  , temperature = 0.0
  , max_tokens = 50)
```

`gpt3_single_embedding` *Obtains text embeddings for a single character (string) from the GPT-3 API*

Description

`gpt3_single_embedding()` sends a single **embedding request** to the Open AI GPT-3 API.

Usage

```
gpt3_single_embedding(input, model = "text-embedding-ada-002")
```

Arguments

<code>input</code>	character that contains the text for which you want to obtain text embeddings from the GPT-3 model
<code>model</code>	a character vector that indicates the similarity embedding model ; one of "text-embedding-ada-002" (default), "text-similarity-ada-001", "text-similarity-curie-001", "text-similarity-babbage-001", "text-similarity-davinci-001". Note: it is strongly recommend to use the faster, cheaper and higher quality second generation embeddings model "text-embedding-ada-002".

Details

The function supports the text similarity embeddings for the four GPT-3 models as specified in the parameter list. The main difference between the four models is the sophistication of the embedding representation as indicated by the vector embedding size.

- Second-generation embeddings model text-embedding-ada-002 (1536 dimensions)
- Ada (1024 dimensions)
- Babbage (2048 dimensions)
- Curie (4096 dimensions)
- Davinci (12288 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 `gpt3_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 people"
gpt3_single_embedding(input = sample_string)
```

```
## Change the model:
#' gpt3_single_embedding(input = sample_string
  , model = 'text-similarity-curie-001')
```

`gpt3_test_completion` *Make a test request to the GPT-3 API*

Description

`gpt3_test_completion()` sends a basic **completion request** to the Open AI GPT-3 API.

Usage

```
gpt3_test_completion(verbose = T)
```

Arguments

`verbose` (boolean) if TRUE prints the actual prompt and GPT-3 completion of the test request (default: TRUE).

Value

A message of success or failure of the connection.

Examples

```
gpt3_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	<i>Contains the package's base URLs</i>
-----------------	---

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

* datasets

`url.completions`, [17](#)

[1](#), [3](#), [4](#), [6](#), [10](#), [14](#)

[2](#), [4](#), [6](#), [10](#), [14](#)

`chatgpt`, [2](#)

`chatgpt_single`, [4](#)

`gpt3_authenticate`, [7](#)

`gpt3_completions`, [8](#)

`gpt3_embeddings`, [11](#)

`gpt3_single_completion`, [12](#)

`gpt3_single_embedding`, [15](#)

`gpt3_test_completion`, [16](#)

`to_numeric`, [16](#)

`url.completions`, [17](#)