Introduction to the ChatGPT API

2023-11-17

1 SETUP: OPENAI_API_KEY

We'll start by setting up your environment for working with the OPENAI-API. You can generate your API key at https://platform.openai.com/api-keys.

1. Generate your key and copy it, you cannot view it again!
2. Save the key in a .yml file, a .env file, or your Renviron (Which you can find under your home dir). A .yml file looks like this:

```
OPENAI_API_KEY: "XXX-...-XXX"
```

3. Install the appropriate package:
    - Python: `pip install openai`
    - R: `install.packages("openai")`
4. Try running the example below:

Python

```
# Specify the path to your API-credentials file
import yaml
import openai
file_path = "credentials.yml"
with open(file_path, 'r') as yaml_file:
    credentials = yaml.safe_load(yaml_file)
openai.api_key = credentials["OPENAI_API_KEY"]

# You can also use an .env file
import os
openai.api_key = os.getenv("OPENAI_API_KEY")

# Get a model response
```

```
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": """
                        You are playing the role of a radio operator \
                        waiting to receive a signal from the user.\
                        When you receive a message, reply with much enthousiasm and gusto.
                        """
                        },
        {"role": "user", "content": "This is OPENAI-API-USER, can anybody hear me?"},
    ],
    temperature=1,     # The randomness setting of the model: 0 is deterministic, 2 is most random
    n = 1,             # How many completions do we want the model to generate
    max_tokens=None    # How many tokens is the model allowed to use
)
print(response.choices[0].message.content)
```

R

```
# Specify the path to your API-credentials file
library(openai) # open-source, not official implementation!
library(yaml)
file_path <- "credentials.yml"
credentials <- yaml.load_file(file_path)
OPENAI_API_KEY <- credentials$OPENAI_API_KEY

# You can also use you .Renvironment
OPENAI_API_KEY <- Sys.getenv["OPENAI_API_KEY"]

# Get a model response
response <- create_chat_completion(
    model = "gpt-3.5-turbo",
    messages = list(
        list(
            "role" = "system",
            "content" = "
                        You are playing the role of a radio operator \
                        waiting to receive a signal from the user.\
                        When you receive a message, reply with much enthousiasm and gusto.
                        "
        ),
        list(
            "role" = "user",
            "content" = "This is OPENAI-API-USER, can anybody hear me?"
        )
    ),
    temperature = 1,     # The randomness setting of the model: 0 is deterministic, 2 is most random
    n = 1,               # How many completions do we want the model to generate
    max_tokens = NULL,    # How many tokens is the model allowed to use,
    openai_api_key = OPENAI_API_KEY # Your secret key to the API
)
```

```r
print(response$choices[["message.content"]])
```

2 Exercise: Getting to know the chat completion API

Experiment with the various parameters of the chat completion API:

- What happens when you adjust temperature?
- What happens when you adjust n? Set it to 3 for example. Don't set this value high as a component of the API cost scales lineairly with this parameter!
- What happens when you adjust max_tokens?

3 Exercise: Iterative prompt development on a movie review

Now that we have acces to the API, it becomes a lot easier to craft better prompts. We can set the temperature for our chat completion to 0, which ensures that the same prompt will gave the same result every time.

For each of the tasks below, finetune your prompt untill you are happy with the result

3.1 Load the review

It can be found under `./Day 1/exercises/data/movie-review-1.txt`

3.2 Task: Provide a synopsis of the review

3.3 Task: Make sure the synopsis of the review is at most 3 sentences long

3.4 Task: Extract the movie title, genre, year, director and actors from the review. Place this information at the top of the review.

3.5 Task: Extract a list of tips and tops of the movie, using only keywords

3.6 Task: Make the model return these fields in a nicely structured (and valid) .json format

3.7 Task: Review number 2

Now apply the prompt you've crafted to the second movie review:
`./Day 1/exercises/data/movie-review-2.txt`
Does it work?

4 Exercise: Estimate token usage before sending your prompts

Before you send messages to the OpenAI API, you may want to check how long they are, you can do this using the functions below:

Python
For Python the function is exact, as we have direct acces to the tokenizer used by OpenAI:

```
import tiktoken
def num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613"):
    """Return the number of tokens used by a list of messages."""
    try:
        encoding = tiktoken.encoding_for_model(model)
    except KeyError:
        print("Warning: model not found. Using cl100k_base encoding.")
        encoding = tiktoken.get_encoding("cl100k_base")
    if model in {
        "gpt-3.5-turbo-0613",
        "gpt-3.5-turbo-16k-0613",
        "gpt-4-0314",
        "gpt-4-32k-0314",
        "gpt-4-0613",
        "gpt-4-32k-0613",
        }:
        tokens_per_message = 3
        tokens_per_name = 1
    elif model == "gpt-3.5-turbo-0301":
```

```
        tokens_per_message = 4  # every message follows <|start|>{role/name}\n{content}<|end|>\n
        tokens_per_name = -1  # if there's a name, the role is omitted
    elif "gpt-3.5-turbo" in model:
        print("Warning: gpt-3.5-turbo may update over time. Returning num tokens assuming gpt-3.5-turbo-0613.")
        return num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613")
    elif "gpt-4" in model:
        print("Warning: gpt-4 may update over time. Returning num tokens assuming gpt-4-0613.")
        return num_tokens_from_messages(messages, model="gpt-4-0613")
    else:
        print("Warning: the selected model is not known. Returning num tokens assuming gpt-3.5-turbo-0613.")
        return num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613")
    num_tokens = 0
    for message in messages:
        num_tokens += tokens_per_message
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":
                num_tokens += tokens_per_name
    num_tokens += 3  # every reply is primed with <|start|>assistant<|message|>
    return num_tokens
```

R

For R, we need to make an estimate, as the tokenizer has no R implementation. We use the rule of thumb that a single word is $\frac{4}{3}$ of a token. In practice this works quite well. In a few test cases the value returned was typically within 5 percent of the true value.

```
library(re)
num_tokens_from_messages <- function(messages, model="gpt-3.5-turbo-0613", tokens-per-word=4/3) {
    tokens_per_message <- 0
    tokens_per_name <- 0

    if (model %in% c("gpt-3.5-turbo-0613", "gpt-3.5-turbo-16k-0613", "gpt-4-0314",
                     "gpt-4-32k-0314", "gpt-4-0613", "gpt-4-32k-0613")) {
        tokens_per_message <- 3
        tokens_per_name <- 1
    } else if (model == "gpt-3.5-turbo-0301") {
        tokens_per_message <- 4
        tokens_per_name <- -1
    } else if (grepl("gpt-3.5-turbo", model)) {
        message("Warning: gpt-3.5-turbo may update over time. Returning num tokens assuming gpt-3.5-turbo-0613
        return(num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613"))
    } else if (grepl("gpt-4", model)) {
        message("Warning: gpt-4 may update over time. Returning num tokens assuming gpt-4-0613.")
        return(num_tokens_from_messages(messages, model="gpt-4-0613"))
    } else {
        stop(sprintf("num_tokens_from_messages() is not implemented for model %s. See https://github.com/opena
    }

    num_tokens <- 0
```

```
    for (message in messages) {
        num_tokens <- num_tokens + tokens_per_message
        for (key in names(message)) {
            value <- message[[key]]
            # Count multiple spaces as a single space.
            value <- gsub(" +", " ", value)
            # Calculate tokens for each word, assuming each word is 4/3 tokens for English
            num_tokens <- num_tokens + (length(strsplit(value, " ")[[1]]) * tokens-per-word)
            if (key == "name") {
                num_tokens <- num_tokens + tokens_per_name
            }
        }
    }

    num_tokens <- num_tokens + 3 # every reply is primed with assistant
    return(num_tokens)
}
```

compare the output of the functions below with the information returned in the response object from the previous exercise.

4.1 API-costs

We are not just interested in how many tokens we use, we are interested in how many cents we spend on a prompt. Write a function that uses the function above to calculate the cost of a prompt. Return 2 values:

- The estimated cost of the prompt input
- The estimated cost of the prompt output

For this exercise you may assume that the output is 4/5 of the length of the input.