CODING EVENT DATA WITH GPT

THE PROBLEM

The New York Times

TRILOBITES

Leave It to Beavers? Not if You're a Wolf.

This is what happens when an apex predator collides with an ecosystem engineer.

Share full article

Solution to save trees

Export Citation

The Advertiser (Australia)

November 8, 2023 Wednesday
Advertiser Edition

Section: SPORT; Pg. 46

Length: 504 words

Byline: Matt Turner

Body

Encroaching into land being acquired for the Torrens to Darlington road project may provide a win-win solution for the Crows and residents who want" 10 **trees** saved on the club's proposed second oval at Thebarton.

It is understood an alternative being considered is to move Adelaide's training oval on Kings Reserve north towards Ashwin Pde and east into a Department for Infrastructure and Transport (DIT) temporary works zone near South Rd.

That option would protect the 10 **trees** in the middle of the reserve, near a rotunda, but would be expected to require the razing of up to nine others along the northern boundary. Netting would likely need to be erected to stop footballs being kicked onto Ashwin Pde.

DEFINE YOUR BROAD CORPUS

Find a set of keywords that appear in your corpus.

This could include:

- Event terms, for example: “protest”, “vote”, “bombing”
- Actor names or identifiers, for example: “Biden”, “LTTE”, “Prime Minister”

You should use **regular expressions** to capture different versions of your key words.

COLLECT YOUR BROAD CORPUS

Collect all articles or text sources that include your keywords.

Sources could include:

- APIs
- Web scraping

Getting access to a large collection of news articles is very expensive. This can be the

MY BROAD CORPUS

```
 1 library(tidyverse)
 2
 3 all_articles <- read_csv(here::here("reports", "data", "all_articles.csv"))
 4 all_articles
```

```
# A tibble: 49,343 × 7
   document_document_id  source_name title publication_date    text
probability
   <chr>                 <chr>       <chr> <dttm>              <chr>
<dbl>
 1 7V2B-8WR1-2PBV-B4HN-… The Associ… Sovi… 1991-04-02 19:00:00 "Whe…
0.517
 2 3TDD-S850-0031-V23G-… Agence Fra… Firs… 1991-09-08 20:00:00 "The…
0.883
 3 3SJ4-DY20-0007-W197-… RusData Di… BALT… 1991-09-17 20:00:00 "WAS…
0.929
 4 49R6-55B0-01VR-924M-… CNN.com     Afgh… 1991-11-10 19:00:00 "A d…
0.966
 5 3SJ4-D9W0-0008-C28H-… The Associ… Afgh… 1991-11-14 19:00:00 "The…
0.904
```

WHITTLE THIS BROAD CORPUS DOWN

You now need to work out which of these articles actually include information on your events and actors.

We are going to ask GPT to tell us the following:

> Identify with ‘yes’ or ‘no’ whether the following article mentions a meeting or event involving representatives of at least two countries, or of at least one country and organization, in which they discuss a conflict involving at least one of those countries.

Building a good prompt is critical to your success.

BUILDING YOUR PROMPT

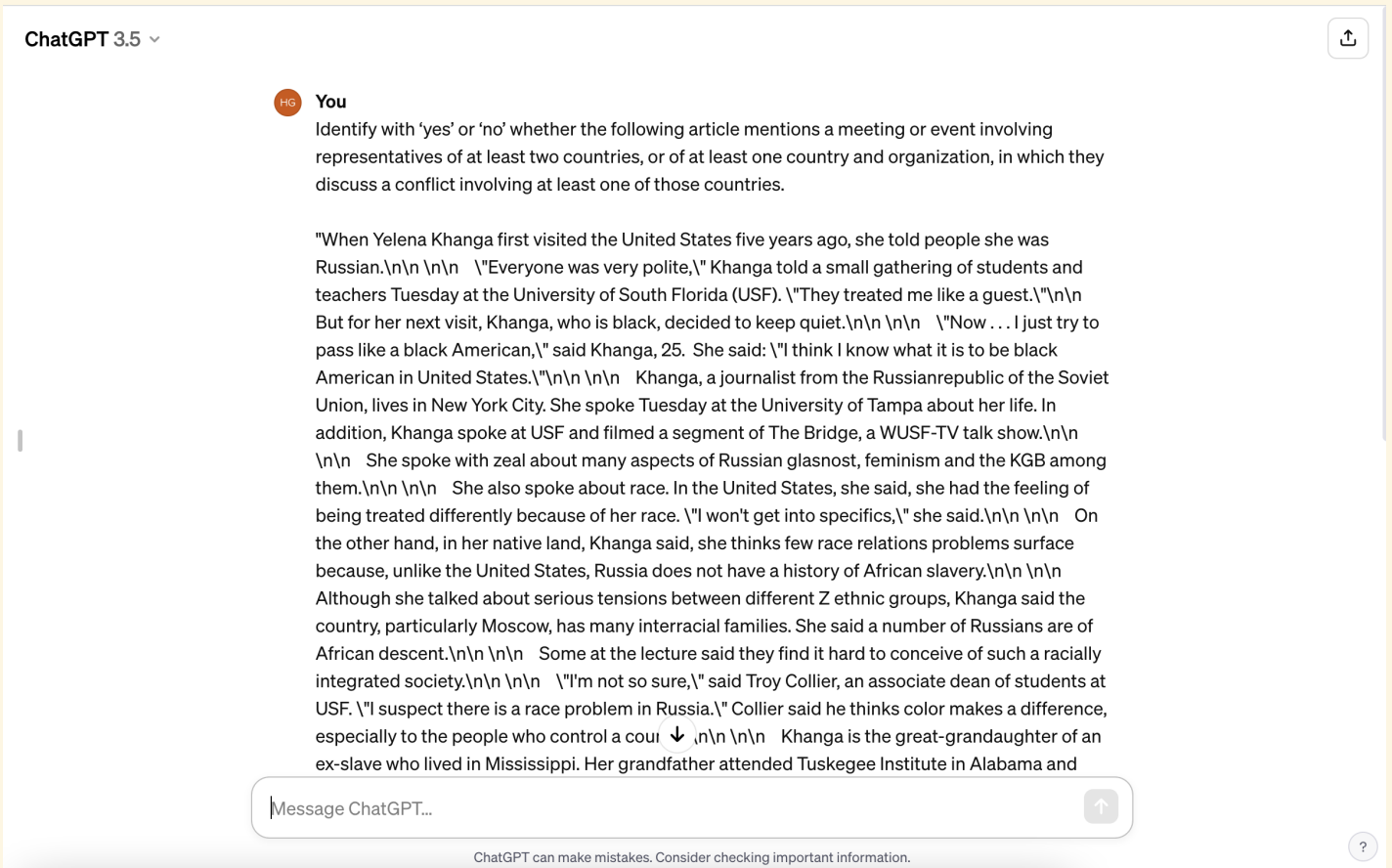
Your prompt needs to be:

- Specific,
- Concise,
- Simple.

You will probably want to experiment with several different prompts.

WORKING WITH A SINGLE ARTICLE

Let’s head over to ChatGPT and see this classification task in action:



HOW DID IT GO?

ChatGPT

No, the provided article does not mention a meeting or event involving representatives of at least two countries, or of at least one country and organization, discussing a conflict involving at least one of those countries.

WORKING WITH MANY ARTICLES

This is great, but what if you have many, *many* articles that you need to code?

Let's use R to help us out!

For each article, we want to:

1. Build a prompt,
2. Run a GPT model of our choice against that prompt,
3. Record its response in a data frame.

BUILDING YOUR PROMPT

First, we need to start with our base prompt:

```
base_prompt <- "Identify with 'yes' or 'no' whether the following article (
base_prompt
```

```
[1] "Identify with 'yes' or 'no' whether the following article (delimited in
XML tags) mentions a meeting or event involving representatives of at least
two countries, or of at least one country and organization, in which they
discuss a conflict involving at least one of those countries: <article>
{article_body}</article>"
```

Notice the more precise markers (XML tags). These delineate the article for the GPT

BUILDING YOUR PROMPT

Next, we need to include our article text:

```
article_body <- all_articles |>
  # Select the first article
  slice(1) |>
  # Pull out the text
  pull(text)
article_body
```

```
[1] "When Yelena Khanga first visited the United States five years ago, she
told people she was Russian.\n\n \n\n    \"Everyone was very polite,\" Khanga
told a small gathering of students and teachers Tuesday at the University of
South Florida (USF). \"They treated me like a guest.\"\n\n   But for her next
visit, Khanga, who is black, decided to keep quiet.\n\n \n\n    \"Now . . . I
just try to pass like a black American,\" said Khanga, 25.  She said: \"I
think I know what it is to be black American in United States.\"\n\n \n\n
   Khanga, a journalist from the Russianrepublic of the Soviet Union, lives in
New York City. She spoke Tuesday at the University of Tampa about her life. In
addition, Khanga spoke at USF and filmed a segment of The Bridge, a WUSF-TV
talk show.\n\n \n\n    She spoke with zeal about many aspects of Russian
glasnost, feminism and the KGB among them.\n\n \n\n    She also spoke about
race. In the United States, she said, she had the feeling of being treated
differently because of her race. \"I won't get into specifics,\" she
said.\n\n \n\n    On the other hand, in her native land, Khanga said, she
```

BUILDING YOUR PROMPT

Finally, we need to add this article body into our prompt:

```
1 article_prompt <- glue::glue(base_prompt)
2 article_prompt
```

```
Identify with 'yes' or 'no' whether the following article (delimited in XML
tags) mentions a meeting or event involving representatives of at least two
countries, or of at least one country and organization, in which they discuss
a conflict involving at least one of those countries: <article>When Yelena
Khanga first visited the United States five years ago, she told people she was
Russian.



    "Everyone was very polite," Khanga told a small gathering of students and
teachers Tuesday at the University of South Florida (USF). "They treated me
like a guest."

   But for her next visit, Khanga, who is black, decided to keep quiet.
```

Check out the `glue()` function from the `glue` package for easy ways to include R

RUN A GPT MODEL AGAINST THIS PROMPT

The end goal:

```
library(httr2)

req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer", Sys.getenv("OPENAI_API_KEY_
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo-0613",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
```

BUILDING YOUR API REQUEST

We are going to take advantage of the fantastic `httr2` R package to work with the OpenAI API.

First, we need to build our request to the API. You need:

- The API endpoint,
- The content type with which you would like to work,
- Your authorization to use the API,
- Your chosen GPT model,
- Your model parameters.

THE API ENDPOINT

The endpoint depends on the type of GPT model you want to use.

For classification tasks, we have two options:

- **Chat completion models**,
- **Completion models**.

THE API ENDPOINT

The base URL for chat completion models is:

> https://api.openai.com/v1/chat/completions

The base URL for completion models is:

> https://api.openai.com/v1/completions

We will focus on chat completion models, which tend to perform better.

THE API ENDPOINT

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer", Sys.getenv("OPENAI_API_KEY_
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo-0613",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
      "max_tokens" = 1
```

`httr2::request()` takes that URL as its first argument.

CONTENT TYPE

Most modern APIs are stored using **JSON**, which is a very light-weight way of sharing data.

Source: Stack Overflow

CONTENT TYPE

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer", Sys.getenv("OPENAI_API_KEY_
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo-0613",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
      "max_tokens" = 1
```

We specify that we want to work with this structure by appending

AUTHORIZATION

It is not free to use GPT.

You will need:

- A subscription,
- An API key.

Head over to https://platform.openai.com/api-keys to set up your API key.

AUTHORIZATION

You should *never* hard code an API key into an R script.

Instead, save it in your R environment:

```
1  Sys.setenv("OPENAI_API_DEMO" = "XXXXXXXXXXXXXXXXXXXXXX")
```

Now you can use the API key without writing it out directly:

```
1  Sys.getenv("OPENAI_API_DEMO")
```

```
[1] "XXXXXXXXXXXXXXXXXXXXXX"
```

AUTHORIZATION

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo-0613",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

We can append our API key to our request using `httr2::req_headers()`.

SELECTING YOUR GPT MODEL

There are many different families of GPT models:

Overview

The OpenAI API is powered by a diverse set of models with different capabilities and price points. You can also make customizations to our models for your specific use case with fine-tuning.

MODEL	DESCRIPTION
GPT-4 and GPT-4 Turbo	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
GPT-3.5	A set of models that improve on GPT-3 and can understand as well as generate natural language or code
DALL·E	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
GPT base	A set of models without instruction following that can understand as well as generate natural language or code
GPT-3 Legacy	A set of models that can understand and generate natural language
Deprecated	A full list of models that have been deprecated along with the suggested replacement

Source: OpenAI API documentation

SELECTING YOUR GPT MODEL

We are working with chat completion models:

- `gpt-4`
- `gpt-4 turbo`
- `gpt-3.5-turbo`

You should always start with the most simple (and cheap) model. See if it performs well

SELECTING YOUR GPT MODEL

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

We specify our selected model in the request body using

SPECIFYING THE ROLE WE WANT GPT TO PLAY

GPT can play many different roles. You can be very creative here and specify exactly what role you would like GPT to play.

For example, you can ask GPT to respond like:

- An academic colleague,
- A reviewer,
- Shakespeare.

SPECIFYING THE ROLE WE WANT GPT TO PLAY

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

INCLUDE YOUR PROMPT

```r
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

SPECIFY YOUR MODEL PARAMETERS: `temperature`

The `temperature` parameter controls how random the model output will be.

- It takes a value between 0 and 2, where 2 is very random and 0 is not random.

For classification tasks, we want a straightforward “Yes” or “No” answer. In other words, we want no randomness.

It is useful to think of randomness like creativity: the higher the temperature, the more

SPECIFY YOUR MODEL PARAMETERS: `temperature`

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

SPECIFY YOUR MODEL PARAMETERS: `max_tokens`

A **token** represents a group of characters (sometimes whole words) that is meaningful to the GPT model.

Source: OpenAI API documentation

SPECIFY YOUR MODEL PARAMETERS: `max_tokens`

The `max_tokens` parameter sets the maximum number of tokens the output can produce.

- We want “Yes” or “No”. These are represented by one token.
- You can check out how many tokens your output will be using the OpenAI Tokenizer.

SPECIFY YOUR MODEL PARAMETERS: max_tokens

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

SPECIFY YOUR MODEL PARAMETERS

There are many different parameters you can control when using chat completion models.

Check out the full list in the OpenAI documentation.

MAKE YOUR REQUESTS MORE ROBUST

Sometimes, your request will fail. You can ask R to retry your request using `httr2::req_retry()`.

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = prompt
        )
      ),
      "temperature" = 0,
```

RUN THE GPT MODEL FOR ONE ARTICLE

We are ready to make that API request!

- We have our prompt:

```
1 article_prompt
```

```
Identify with 'yes' or 'no' whether the following article (delimited in XML
tags) mentions a meeting or event involving representatives of at least two
countries, or of at least one country and organization, in which they discuss
a conflict involving at least one of those countries: <article>When Yelena
Khanga first visited the United States five years ago, she told people she was
Russian.



    "Everyone was very polite," Khanga told a small gathering of students and
teachers Tuesday at the University of South Florida (USF). "They treated me
like a guest."

   But for her next visit, Khanga, who is black, decided to keep quiet.
```

RUN THE GPT MODEL FOR ONE ARTICLE

We are ready to make that API request!

- Which we can insert into our fully fleshed out request:

```
req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
        list(
          "role" = "user",
          "content" = article_prompt
        )
      ),
```

RUN THE GPT MODEL FOR ONE ARTICLE

Now we just need to perform that request:

```
1 resp <- req_perform(req)
```

And see the response:

```
1 resp
```

```
$method
[1] "POST"

$url
[1] "https://api.openai.com/v1/chat/completions"

$status_code
[1] 200

$headers
$date
[1] "Sat, 18 Nov 2023 18:26:01 GMT"

$`content-type`
[1] "application/json"
```

GPT'S RESPONSE

Let's break down this response:

```
1 resp
```

```
$method
[1] "POST"

$url
[1] "https://api.openai.com/v1/chat/completions"

$status_code
[1] 200

$headers
$date
[1] "Sat, 18 Nov 2023 18:26:01 GMT"

$`content-type`
[1] "application/json"
```

BUT WHERE IS THE PREDICTION?

Welcome to the JSON rabbit hole…

```
1 resp_body_json(resp)
```

```
$id
[1] "chatcmpl-8MKIzTqhYhQNtudApua7dMQMMpD0g"

$object
[1] "chat.completion"

$created
[1] 1700331961

$model
[1] "gpt-3.5-turbo-0613"

$choices
$choices[[1]]
$choices[[1]]$index
```

BUT WHERE IS THE PREDICTION?

Welcome to the JSON rabbit hole…

```
1 resp_body_json(resp)$choices
```

```
[[1]]
[[1]]$index
[1] 0

[[1]]$message
[[1]]$message$role
[1] "assistant"

[[1]]$message$content
[1] "No"


[[1]]$finish_reason
[1] "length"
```

BUT WHERE IS THE PREDICTION?

Welcome to the JSON rabbit hole…

```
1 resp_body_json(resp)$choices[[1]]$message
```

```
$role
[1] "assistant"

$content
[1] "No"
```

BUT WHERE IS THE PREDICTION?

Welcome to the JSON rabbit hole…

```
1 resp_body_json(resp)$choices[[1]]$message$content
```

```
[1] "No"
```

SAVE THE PREDICTION

```r
pred <- resp_body_json(resp)$choices[[1]]$message$content

df <- tibble(
    body = article_body,
    gpt_pred = pred
  )

df
```

```
# A tibble: 1 × 2
  body
gpt_pred
  <chr>                                                                      <chr>
1 "When Yelena Khanga first visited the United States five years ago, … No
```

CONGRATULATIONS!

You have now used an advanced large language model to identify whether an event is referenced in a news article.

Let’s set you up to do that across all 49,343 articles.

BUILDING YOUR ARTICLE READER FUNCTION

End goal:

```
article_classification <- function(article_body) {

  # Create your prompt
  article_prompt <- glue::glue("Identify with 'yes' or 'no' whether the fol

  # Build your request
  req <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers("Content-Type" = "application/json",
              "Authorization" = paste("Bearer",
                                      Sys.getenv("OPENAI_API_PERSONAL"))) |
  req_body_json(
    list(
      "model" = "gpt-3.5-turbo",
      "messages" = list(
        list(
          "role" = "system",
          "content" = "You are a helpful assistant."
        ),
```

RUNNING OUR FUNCTION ACROSS OUR ARTICLES

```
1 all_articles |>
2   slice(1) |>
3   pull(text)
```

```
[1] "When Yelena Khanga first visited the United States five years ago, she
told people she was Russian.\n\n \n\n    \"Everyone was very polite,\" Khanga
told a small gathering of students and teachers Tuesday at the University of
South Florida (USF). \"They treated me like a guest.\"\n\n   But for her next
visit, Khanga, who is black, decided to keep quiet.\n\n \n\n    \"Now . . . I
just try to pass like a black American,\" said Khanga, 25.  She said: \"I
think I know what it is to be black American in United States.\"\n\n \n\n
   Khanga, a journalist from the Russianrepublic of the Soviet Union, lives in
New York City. She spoke Tuesday at the University of Tampa about her life. In
addition, Khanga spoke at USF and filmed a segment of The Bridge, a WUSF-TV
talk show.\n\n \n\n    She spoke with zeal about many aspects of Russian
glasnost, feminism and the KGB among them.\n\n \n\n    She also spoke about
race. In the United States, she said, she had the feeling of being treated
differently because of her race. \"I won't get into specifics,\" she
said.\n\n \n\n    On the other hand, in her native land, Khanga said, she
```

RUNNING OUR FUNCTION ACROSS OUR ARTICLES

In base R:

```
1 all_articles$text[[1]]
```

```
[1] "When Yelena Khanga first visited the United States five years ago, she
told people she was Russian.\n\n \n\n    \"Everyone was very polite,\" Khanga
told a small gathering of students and teachers Tuesday at the University of
South Florida (USF). \"They treated me like a guest.\"\n\n   But for her next
visit, Khanga, who is black, decided to keep quiet.\n\n \n\n    \"Now . . . I
just try to pass like a black American,\" said Khanga, 25.  She said: \"I
think I know what it is to be black American in United States.\"\n\n \n\n
   Khanga, a journalist from the Russianrepublic of the Soviet Union, lives in
New York City. She spoke Tuesday at the University of Tampa about her life. In
addition, Khanga spoke at USF and filmed a segment of The Bridge, a WUSF-TV
talk show.\n\n \n\n    She spoke with zeal about many aspects of Russian
glasnost, feminism and the KGB among them.\n\n \n\n    She also spoke about
race. In the United States, she said, she had the feeling of being treated
differently because of her race. \"I won't get into specifics,\" she
said.\n\n \n\n    On the other hand, in her native land, Khanga said, she
```

RUNNING OUR FUNCTION ACROSS OUR ARTICLES

```
labelled_articles <- map(
  1:5,
  ~ all_articles |>
      slice(.x) |>
      pull(text) |>
      article_classification()
) |>
  bind_rows()
```

`purrr::map()` is a tidy method for looping. `dplyr::bind_rows()` appends the

THE RESULT

```
1 labelled_articles
```

```
# A tibble: 5 × 2
  body                                                                  gpt_pred
  <chr>                                                                 <chr>
1 "When Yelena Khanga first visited the United States five years ago, … No
2 "The first Soviet delegation to visit Afghanistan since the failed h… Yes
3 "WASHINGTON - President George Bush met Tuesday with the presidents … Yes
4 "A delegation of Afghan mujahedeen rebels met Russian Vice-President… Yes
5 "The chief guerrilla delegate at talks on ending Afghanistan's civil… Yes
```

SOME TIPS: TOKEN LIMITS

There are token limits for your prompts.

- For `gpt-3.5-turbo`, it is 4,096 tokens.

A token is roughly four characters (in English).

SOME TIPS: TOKEN LIMITS

To make sure your prompts do not go over this limit, add this:

```
article_classification <- function(article_body) {

  # Create your prompt
  article_prompt <- glue::glue("Identify with 'yes' or 'no' whether the fol

  if (nchar(article_prompt) / 4 < 4096) {

    # Build your request
    req <- request("https://api.openai.com/v1/chat/completions") |>
    req_headers("Content-Type" = "application/json",
                "Authorization" = paste("Bearer",
                                        Sys.getenv("OPENAI_API_PERSONAL")))
    req_body_json(
      list(
        "model" = "gpt-3.5-turbo",
        "messages" = list(
          list(
            "role" = "system",
```

SOME TIPS: CLEAN YOUR INPUT

You will often have junk in your text (for example, paragraph delimiters or news agency bylines). Removing this:

- Increases the likelihood that you won't reach the token limit,
- Reduces your use costs,
- Can produce more accurate results.

SOME TIPS: EVALUATING YOUR MODEL

You should check whether or not the model is correctly identifying relevant articles.

- Select some labelled articles at random and hand code them.
- Evaluate how accurate your model is performing.

Extension: you can fine-tune these models using labelled data to produce a GPT model