ChatGPT for of Political Document Annotation

Haohan Chen

The University of Hong Kong

Prepared for lecture at Tsinghua University. December 16, 2023

Large Language Models

- Chat partner
- Consultant: e.g., Financial planning
- Teacher: e.g., R Programming
- Counselor
- Designer

Large Language Models for Political Data Analytics

What you probably have tried...

- Proof-reading, copy-editing for papers
- Debugging for code
- Inspiring research ideas

Today: Document annotation

How to extract information you need from documents about using ChatGPT?

Document Annotation

Laborious, tedious, BUT ESSENTIAL part of political social data analytics!

What may need annotation for research:

- Social media posts
- Newspapers
- Politicians' pronouncement (e.g., speeches, campaign leaflets)
- Laws and regulations
- Historical archives

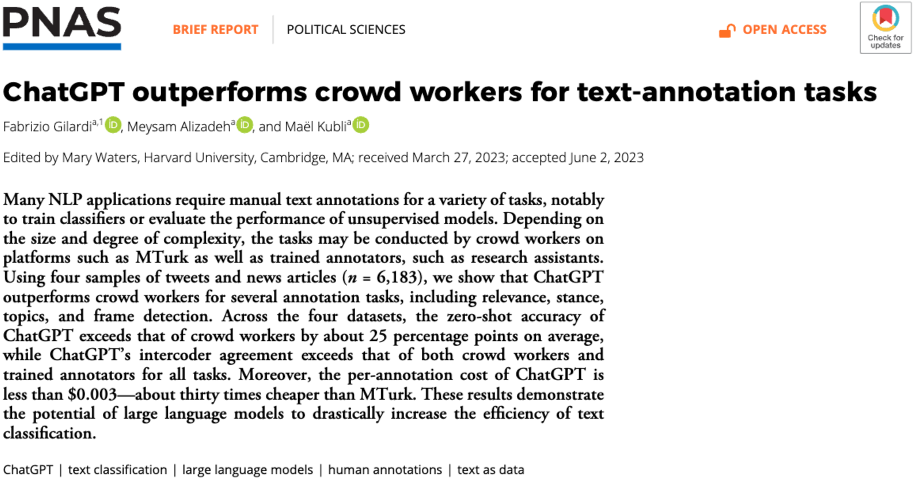
Pre-ChatGPT Approaches

- **Manual annotation**: Hand-code all the documents following rules
- Machine annotation
  - Keyword matching
  - Topic modeling
  - Dictionary-based sentiment analysis
- Manual + Machine annotation
  - Hand-code a portion of documents
  - Train machine learning models
  - Use machine learning models to code the remainder of documents

ChatGPT as a Research Assistant

Instead of hiring human assistants, hire ChatGPT to annotate documents!

ChatGPT performs well in political text annotation



PNAS BRIEF REPORT | POLITICAL SCIENCES OPEN ACCESS

ChatGPT outperforms crowd workers for text-annotation tasks

Fabrizio Gilardi[a,1], Meysam Alizadeh[a], and Maël Kubli[a]

Edited by Mary Waters, Harvard University, Cambridge, MA; received March 27, 2023; accepted June 2, 2023

Many NLP applications require manual text annotations for a variety of tasks, notably to train classifiers or evaluate the performance of unsupervised models. Depending on the size and degree of complexity, the tasks may be conducted by crowd workers on platforms such as MTurk as well as trained annotators, such as research assistants. Using four samples of tweets and news articles (n = 6,183), we show that ChatGPT outperforms crowd workers for several annotation tasks, including relevance, stance, topics, and frame detection. Across the four datasets, the zero-shot accuracy of ChatGPT exceeds that of crowd workers by about 25 percentage points on average, while ChatGPT's intercoder agreement exceeds that of both crowd workers and trained annotators for all tasks. Moreover, the per-annotation cost of ChatGPT is less than $0.003—about thirty times cheaper than MTurk. These results demonstrate the potential of large language models to drastically increase the efficiency of text classification.

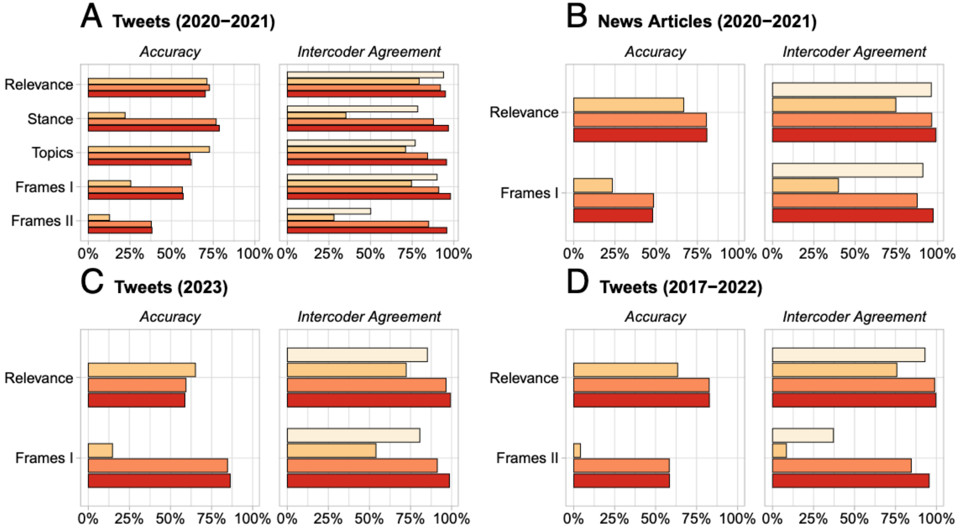
ChatGPT | text classification | large language models | human annotations | text as data

Gilardi et al. (2023)

A systematic evalation of GPT's performance in political text annotation

- Two Benchmarks
  - Trained annotators
  - Crowdsourced annotators
- Two Metrics
  - Accuracy
  - Intercoder agreement
- Various tasks
  - Vary in terms of complexity
  - Vary in terms of novelty (being part of GTP's training data or not)

Gilardi et al. (2023)



Why I recommend this piece of work

- Clearly defined metrics
- My favorite replication materials
  https://doi.org/10.7910/DVN/PQYF6M

Other Selected Works Using GPT for Political Science Research

- Wu, P. Y., Nagler, J., Tucker, J. A., & Messing, S. (2023). Large language models can be used to estimate the latent positions of politicians.
- Argyle, L. P., Bail, C. A., Busby, E. C., Gubler, J. R., Howe, T., Rytting, C., ... & Wingate, D. (2023). Leveraging AI for democratic discourse: Chat interventions can improve online political conversations at scale. Proceedings of the National Academy of Sciences, 120(41).
- Argyle, L. P., Busby, E. C., Fulda, N., Gubler, J. R., Rytting, C., & Wingate, D. (2023). Out of one, many: Using language models to simulate human samples. Political Analysis, 31(3), 337-351.

An emerging literature!

How We Want ChatGPT's Document Annotation Look Like

- Inputs
  - Transparent
  - Interpretable
- Process
  - Follow rules
  - Be sophisticated
- Outputs
  - Structured
  - Reproducible

What you need to learn

Getting the input, process, and output right:

- Give helpful **inputs** to guide GPT through the **process**
- Scientifically evaluate GPT's **outputs**

A 4-Step Workflow

- **Step 1**: Create instructions and a “training” set
- **Step 2**: Apply ChatGPT to the “training” set
- **Step 3**: Evaluate ChatGPT’s performance with the training set
- **Step 4**: Apply ChatGPT to annotate documents outside the “training” set

Step 1: To start with...

Before an AI assistant starts working, you should:

- Figure what you want
- Tell AI precisely what you want

However, AI can help you:

- Clarify what you want
- Better articulate what you want

Example: Content Moderation

Example: Using ChatGPT to label a set of tweets' relevance to "content moderation" (Gilardi et al. 2023)

"Content moderation" refers to the practice of screening and monitoring content posted by users on social media sites to determine if the content should be published or not, based on specific rules and guidelines."

We will use the authors' replication materials:

- Instructions in `data/instructions.txt`
- A "training" annotated set in `data/training_set.csv`

Draft Instructions

Write instructions as if you are writing them for a human assistant.

- Be clear and specific about:
  - AI's role and tasks
  - Inputs
  - Desired outputs
- Be organized. But do not underestimate AI's ability to understand.

Ultimately, your instructions should make sense to both humans and AIs.

Take a Look at the Example Instructions

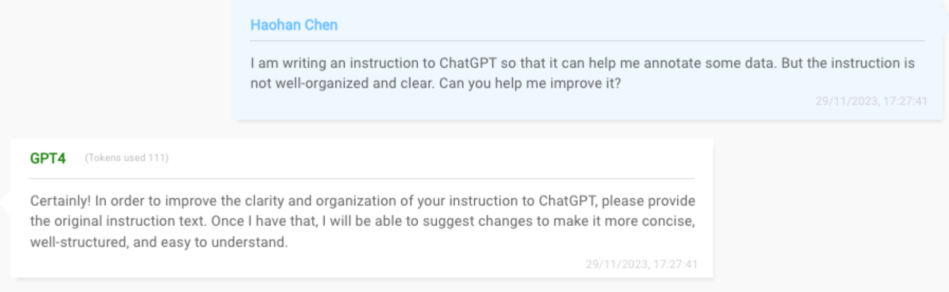
```
instructions <- readLines("data/input/instructions.txt")
instructions <- str_c(instructions, collapse = "\n")

cat(instructions) # Print the instruction (not shown on slides)
```

Can we improve it? ChatGPT can help!

ChatGPT's graphic user interface suffices (no programming is required).

Improve Instructions using ChatGPT



Haohan Chen

I am writing an instruction to ChatGPT so that it can help me annotate some data. But the instruction is not well-organized and clear. Can you help me improve it?

29/11/2023, 17:27:41

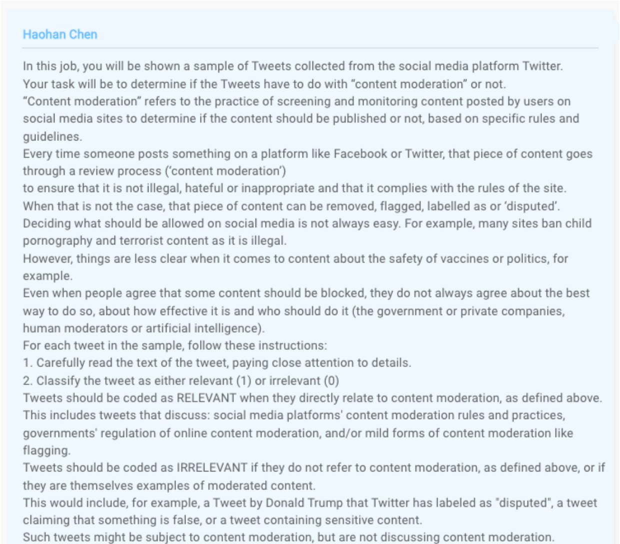
GPT4 (Tokens used 111)

Certainly! In order to improve the clarity and organization of your instruction to ChatGPT, please provide the original instruction text. Once I have that, I will be able to suggest changes to make it more concise, well-structured, and easy to understand.

29/11/2023, 17:27:41

- The comment on the original instruction is exageratively harsh. I just wanted to get GPT's attention...
- Excuse me for the grammatical mistake in my prompt.

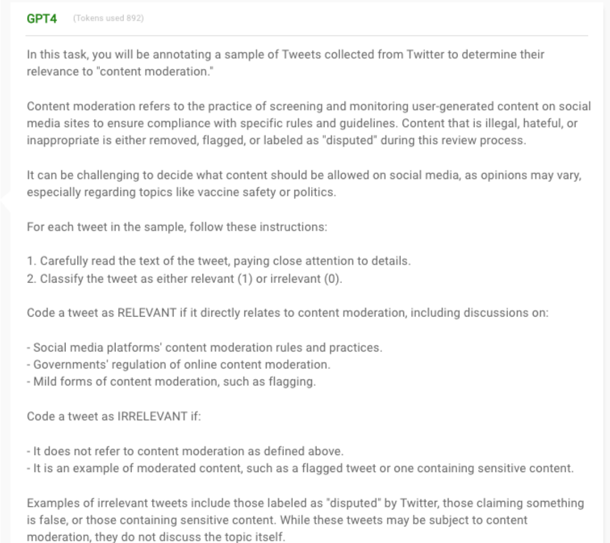
Improve Instructions using ChatGPT



Haohan Chen

In this job, you will be shown a sample of Tweets collected from the social media platform Twitter.
Your task will be to determine if the Tweets have to do with "content moderation" or not.
"Content moderation" refers to the practice of screening and monitoring content posted by users on social media sites to determine if the content should be published or not, based on specific rules and guidelines.
Every time someone posts something on a platform like Facebook or Twitter, that piece of content goes through a review process ('content moderation')
to ensure that it is not illegal, hateful or inappropriate and that it complies with the rules of the site.
When that is not the case, that piece of content can be removed, flagged, labelled as or 'disputed'.
Deciding what should be allowed on social media is not always easy. For example, many sites ban child pornography and terrorist content as it is illegal.
However, things are less clear when it comes to content about the safety of vaccines or politics, for example.
Even when people agree that some content should be blocked, they do not always agree about the best way to do so, about how effective it is and who should do it (the government or private companies, human moderators or artificial intelligence).
For each tweet in the sample, follow these instructions:
1. Carefully read the text of the tweet, paying close attention to details.
2. Classify the tweet as either relevant (1) or irrelevant (0)
Tweets should be coded as RELEVANT when they directly relate to content moderation, as defined above.
This includes tweets that discuss: social media platforms' content moderation rules and practices, governments' regulation of online content moderation, and/or mild forms of content moderation like flagging.
Tweets should be coded as IRRELEVANT if they do not refer to content moderation, as defined above, or if they are themselves examples of moderated content.
This would include, for example, a Tweet by Donald Trump that Twitter has labeled as "disputed", a tweet claiming that something is false, or a tweet containing sensitive content.
Such tweets might be subject to content moderation, but are not discussing content moderation.

Improve Instructions using ChatGPT



In this task, you will be annotating a sample of Tweets collected from Twitter to determine their relevance to "content moderation."

Content moderation refers to the practice of screening and monitoring user-generated content on social media sites to ensure compliance with specific rules and guidelines. Content that is illegal, hateful, or inappropriate is either removed, flagged, or labeled as "disputed" during this review process.

It can be challenging to decide what content should be allowed on social media, as opinions may vary, especially regarding topics like vaccine safety or politics.

For each tweet in the sample, follow these instructions:

1. Carefully read the text of the tweet, paying close attention to details.
2. Classify the tweet as either relevant (1) or irrelevant (0).

Code a tweet as RELEVANT if it directly relates to content moderation, including discussions on:

- Social media platforms' content moderation rules and practices.
- Governments' regulation of online content moderation.
- Mild forms of content moderation, such as flagging.

Code a tweet as IRRELEVANT if:

- It does not refer to content moderation as defined above.
- It is an example of moderated content, such as a flagged tweet or one containing sensitive content.

Examples of irrelevant tweets include those labeled as "disputed" by Twitter, those claiming something is false, or those containing sensitive content. While these tweets may be subject to content moderation, they do not discuss the topic itself.

Finalize the Instructions

I made a few more changes to the instructions. The finalized instructions is saved in `data/instructions_improved_2.txt`.

```r
instructions <- readLines("data/input/instructions_improved_2.txt") |>
  str_c(collapse = "\n")
cat(instructions) # Print the improved instruction, not shown on slides
```

How Else Can ChatGPT Help with Instructions?

- Clarify concepts
- Expand elaborations (e.g., possible scenarios)

You can start with half-cooked instructions and ask ChatGPT to help you build it up. You will be pleasantly surprised how well ChatGPT does.

ChatGPT can inspire, but it cannot replace thinking and decision-making. You should know what you want.

Create a “Training” Set

A straightforward but tedious step

- Annotate **a small set of** documents using the instructions you have developed. This can be called a “training” set or a “gold standard” set.
- Usually, each document should be annotated by two persons.
- Conflicts between two coders? It happens a lot!
  - Discard those with disagreement
  - Discuss and reach an agreement
  - Give them to a third “tie-breaker”
- Adjust the instructions based on tricky cases/ disagreement.

Case: One of the Gilardi et al. (2023) Training Sets

```
d <- read_csv("data/input/tweets_training.csv",
              col_types = cols(status_id = col_character()))
# A small text cleaning to replace line breaks with spaces
d <- d |> mutate(text = str_replace_all(text, "(\n|\r)", " "))
```

Q: How do they deal with conflicts?

For this in-class demonstration, let's make a smaller "training" set.

```
set.seed(12)
d_train <- d |>
  select(status_id, text, relevant_ra) |>
  filter(!is.na(relevant_ra)) |>
  sample_n(40) |>
  arrange(status_id)

table(d_train$relevant_ra)
```

```
##
##  0  1
## 20 20
```

Step 2: Apply ChatGPT to your “Training” Set

- Now, it is time to delegate the work to the AI!
- Do not rush into this step. But there is no need to do too much marginal improvement on the instructions and the training set.
- To discuss:
    - How to annotate one example document?
    - How to annotate many documents in batch?

How to Annotate one document?

First, I will demonstrate how to use ChatGPT to annotate one document.

```r
to_annotate_id <- d_train$status_id[1]
to_annotate_text <- d_train$text[1]
to_annotate_raLabel <- d_train$relevant_ra[1]

cat(to_annotate_id, "\n\n", to_annotate_text, "\n\n", to_annotate_raLabel)
```

```
1212265682634604544

ONCE ! Update : the police are still investigating so please dont report his account anymore Tweets are needed
for his further actions Just unfollow and block

1
```

Annotate one Doc: Communicate with the ChatGPT API

```r
# Required packages
library(httr) # Package to communicate with the ChatGPT API
library(jsonlite) # Package to parse results JSON-format results returned from ChatGPT
api_key <- readLines("data/input/api/api_key.txt")
api_url <- readLines("data/input/api/api_url_gpt3.5.txt")
```

```r
# Make an API call
response <- POST(
  url = api_url,
  add_headers(`Content-Type` = "application/json", `api-key` = api_key),
  encode = "json",
  body = list(
    temperature = 0,
    # 0 to 1. How "creative" you want ChatGPT to be. Smaller = less creative.
    messages = list(
      list(role = "system", content = instructions),
      list(role = "user", content = to_annotate_text))
    )
  )
dir.create("data/output_gpt3.5")
write_rds(response, str_c("data/output_gpt3.5/", to_annotate_id, ".rds"))
# Save ChatGPT's outputs
```

Annotate one Doc: Interpret ChatGPT's Responses

```
response <- read_rds(str_c("data/output_gpt3.5/", to_annotate_id, ".rds"))
content(response)
```

```
## $id
## [1] "chatcmpl-8W48pOVM7shiMPXXqUYI4aHcWg25z"
##
## $object
## [1] "chat.completion"
##
## $created
## [1] 1702653107
##
## $model
## [1] "gpt-35-turbo"
##
## $choices
## $choices[[1]]
## $choices[[1]]$finish_reason
## [1] "stop"
##
## $choices[[1]]$index
## [1] 0
##
## $choices[[1]]$message
## $choices[[1]]$message$role
## [1] "assistant"
##
## $choices[[1]]$message$content
## [1] "0"
```

Annotate one Doc: Interpret ChatGPT's Responses (2)

```r
# This gets ChatGPT's response
content(response)$choices[[1]]$message$content
```

```
## [1] "0"
```

```r
# Compare with RA label
to_annotate_raLabel
```

```
## [1] 1
```

Annotate Many Documents: Loop through All Documents

```r
# Note: packages have been loaded, and api_key and api_url have been defined.

for (i in 1:5){ # Only loop through 5 for demo purpose
  to_annotate_id <- d_train$status_id[i]
  to_annotate_text <- d_train$text[i]
  # Above I do a small string operation -- replacing line breaks by spaces
  to_annotate_raLabel <- d_train$relevant_ra[i]

  response <- POST(
    url = api_url,
    add_headers(`Content-Type` = "application/json", `api-key` = api_key),
    encode = "json",
    body = list(
      temperature = 0,
      # 0 to 1. How "creative" you want ChatGPT to be. Smaller = less creative.
      messages = list(
        list(role = "system", content = instructions),
        list(role = "user", content = to_annotate_text))
      )
    )
  to_annotate_gptLabel <- content(response)$choices[[1]]$message$content

  write_rds(response, str_c("data/output_gpt3.5/", to_annotate_id, ".rds"))
  Sys.sleep(0.5) # Sleep for 0.5 seconds after finishing each doc.
  message(i, " of ", nrow(d_train))
  # Optional below: Print results to get a "live update"
  message("status_id: ", to_annotate_id, "\n", "text: ", to_annotate_text)
  message("Human: ", to_annotate_raLabel, "\t", "ChatGPT: ", to_annotate_gptLabel, "\n")
}
```

Collect and Clean ChatGPT's responses

```
# Get file names of ChatGPT's responses. Note that file names are the status_id
# This helps us identify which file is from which tweet
file_names <- list.files("data/output_gpt3.5")
gpt_labels <- rep(NA, length(file_names))

for (i in seq_along(file_names)){
  response <- read_rds(file.path("data/output_gpt3.5", file_names[i]))
  gpt_labels[i] <-
    ifelse(
      is.null(content(response)$choices[[1]]$message$content),
      NA, content(response)$choices[[1]]$message$content)
  # The above ifelse() function handles the situation when the output is "content-mderated" by Microsoft!
}

d_gptLabel <- tibble(
  status_id = str_remove(file_names, "\\.rds$"),
  relevant_gpt = gpt_labels)

d_gptLabel |> print(n = 3)
```

```
## # A tibble: 40 x 2
##   status_id           relevant_gpt
##   <chr>               <chr>
## 1 1212265682634604544 0
## 2 1212425759421292546 1
## 3 1217297045108723713 0
## # i 37 more rows
```

Merge with the Human-Annotated Data

```r
d_train_merge <- d_train |> inner_join(d_gptLabel, by = "status_id")

d_train_merge |> print(n = 5)
```

```
## # A tibble: 40 x 4
##   status_id           text                                  relevant_ra relevant_gpt
##   <chr>               <chr>                                       <dbl> <chr>
## 1 1212265682634604544 ONCE !     Update : the police a~           1 "0"
## 2 1212425759421292546 The release of these updated ter~           1 "1"
## 3 1217297045108723713 My account would get suspended i~           1 "0"
## 4 1221020543417233408 - Huawei Prepared for Any Furthe~           0 "1\n1\n0\n0"
## 5 1223437508014493696 Here in Ontario there’s many peo~           0 "0"
## # i 35 more rows
```

Compare Humans' and ChatGPT's Annotation

Make a Confusion Matrix to compare GPT's and humans' results.

```
confusion_mat <- with(d_train_merge, table(relevant_ra, relevant_gpt, useNA = "ifany"))
confusion_mat
```

```
##            relevant_gpt
## relevant_ra  0  1 1\n1\n0\n0
##           0 15  4          1
##           1  5 15          0
```

How good do you think this result is? It agrees with the human annotator in 3/4 of the tasks.

Compare Humans' and ChatGPT's Annotation (2)

Check anomalies

```
# Check the tweet with wrong output
d_train_merge |> filter(relevant_gpt == "1\n1\n0\n0") %>% .$text |> print()
```

[1] "- Huawei Prepared for Any Further U.S. "Attacks" - Google and IBM have both called for global regulations on AI - Social media use not linked to poor mental health - Spotify is testing letting influencers post stories to introduce their own playlists https://t.co/BOv2kYNIjf"

```
# Recode manually
d_train_merge <- d_train_merge |>
  mutate(relevant_gpt =
           case_match(relevant_gpt, "1\n1\n0\n0" ~ "1", .default = relevant_gpt))
# Regenerate the confusion matrix
confusion_mat <- with(d_train_merge, table(relevant_ra, relevant_gpt, useNA = "ifany"))
confusion_mat
```

```
##              relevant_gpt
## relevant_ra   0  1
##            0 15  5
##            1  5 15
```

Compare Humans' and ChatGPT's Annotation (3)

```
library(caret)
confusionMatrix(confusion_mat)
```

```
## Confusion Matrix and Statistics
##
##            relevant_gpt
## relevant_ra  0  1
##           0 15  5
##           1  5 15
##
##                Accuracy : 0.75
##                  95% CI : (0.588, 0.8731)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.001111
##
##                   Kappa : 0.5
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.750
##             Specificity : 0.750
##          Pos Pred Value : 0.750
##          Neg Pred Value : 0.750
##              Prevalence : 0.500
##          Detection Rate : 0.375
##    Detection Prevalence : 0.500
##       Balanced Accuracy : 0.750
##
##        'Positive' Class : 0
##
```

How can you improve it?

- Manually examine the tweets where the human annotator disagree with ChatGPT
- Think: How can you improve the instructions?
- Do not rule out the possibility that human annotator make mistakes.

After you are done with Steps 1-3...

- You are confident about your instructions
- Apply the model to documents outside the “training” set.

What to pay attention to

- Estimate the cost
- Watch out for rate limit (add enough pauses between iterations)
- Do it in batch
- **Add error-handling modules**
  - Save all raw outputs as individual files with unique identifiers
  - You can start over without redoing what has been done
  - Check for rate limit error code

Example Code with Error-Handling Modules

```r
for (i in 1:nrow(d_to_annotate)) {
  to_annotate_id <- d_to_annotate$status_id[i]
  to_annotate_text <- d_to_annotate$text[i]
  file_path <- file.path(dir_output, str_c(to_annotate_id, ".rds"))
  # Check if a document has been processed, if so, skipped
  if (file.exists(file_path)){
    response <- read_rds(file_path)
    gpt_label <- content(response)$choices[[1]]$message$content
    error_msg <- ifelse(is.null(gpt_label), content(response)$error$code, "")
    if (error_msg %in% c("", "content_filter")){
      message(i, " of ", nrow(data_batch), " label: ", gpt_label, error_msg, " SKIPPED!")
      next
    }
  }
  response <- POST(
    url = api_url,
    add_headers(`Content-Type` = "application/json", `api-key` = api_key),
    encode = "json",
    body = list(
      temperature = 0,
      messages = list(
        list(role = "system", content = instructions),
        list(role = "user", content = to_annotate_text))))
  gpt_label <- content(response)$choices[[1]]$message$content
  error_msg <- ifelse(is.null(gpt_label), content(response)$error$code, "")
  write_rds(response, file_path)
  Sys.sleep(1)
  message(i, " of ", nrow(data_batch), " label: ", gpt_label, error_msg)
}
```

Does GPT 4 perform better?

```r
api_url_gpt4 <- readLines("data/input/api/api_url_gpt4.txt")
dir.create("data/api/output_gpt4")

for (i in 1:5){ # Just do first 5 for demo purpose
  to_annotate_id <- d_train$status_id[i]
  to_annotate_text <- d_train$text[i]
  to_annotate_raLabel <- d_train$relevant_ra[i]

  response <- POST(
    url = api_url_gpt4,
    add_headers(`Content-Type` = "application/json", `api-key` = api_key),
    encode = "json",
    body = list(
      temperature = 0,
      messages = list(
        list(role = "system", content = instructions),
        list(role = "user", content = to_annotate_text))
      )
    )

  to_annotate_gptLabel <- content(response)$choices[[1]]$message$content

  write_rds(response, str_c("data/output_gpt4/", to_annotate_id, ".rds"))
  Sys.sleep(0.5) # Sleep for 0.5 seconds after finishing each doc.
  message(i, " of ", nrow(d_train))
  # Optional below: Print results to get a "live update"
  message("status_id: ", to_annotate_id, "\n", "text: ", to_annotate_text)
  message("Human: ", to_annotate_raLabel, "\t", "ChatGPT: ", to_annotate_gptLabel, "\n")
}
```

Collect and Clean GPT-4's responses

```r
# Get file names of ChatGPT's responses. Note that file names are the status_id
# This helps us identify which file is from which tweet
file_names <- list.files("data/output_gpt4")
gpt_labels <- rep(NA, length(file_names))

for (i in seq_along(file_names)){
  response <- read_rds(file.path("data/output_gpt4", file_names[i]))
  gpt_labels[i] <-
    ifelse(
      is.null(content(response)$choices[[1]]$message$content),
      NA, content(response)$choices[[1]]$message$content)
  # The above ifelse() function handles the situation when the output is "content-mderated" by Microsoft!
}

d_gptLabel <- tibble(
  status_id = str_remove(file_names, "\\.rds$"),
  relevant_gpt4 = gpt_labels)

d_gptLabel |> print(n = 5)
```

```
## # A tibble: 40 x 2
##   status_id           relevant_gpt4
##   <chr>               <chr>
## 1 1212265682634604544 1
## 2 1212425759421292546 1
## 3 1217297045108723713 1
## 4 1221020543417233408 0
## 5 1223437508014493696 0
## # i 35 more rows
```

Merge GPT-4 Results

```
d_train_merge_2 <- d_train_merge |> inner_join(d_gptLabel, by = "status_id")
d_train_merge_2 |> print(n = 5)
```

```
## # A tibble: 40 x 5
##   status_id           text                 relevant_ra relevant_gpt relevant_gpt4
##   <chr>               <chr>                      <dbl> <chr>        <chr>
## 1 1212265682634604544 ONCE !     Update ~            1 0            1
## 2 1212425759421292546 The release of the~            1 1            1
## 3 1217297045108723713 My account would g~            1 0            1
## 4 1221020543417233408 - Huawei Prepared ~            0 1            0
## 5 1223437508014493696 Here in Ontario th~            0 0            0
## # i 35 more rows
```

```
write_csv(d_train_merge_2, "data/output_clean/labeled.csv")
```

Evaluate GPT-4 Results

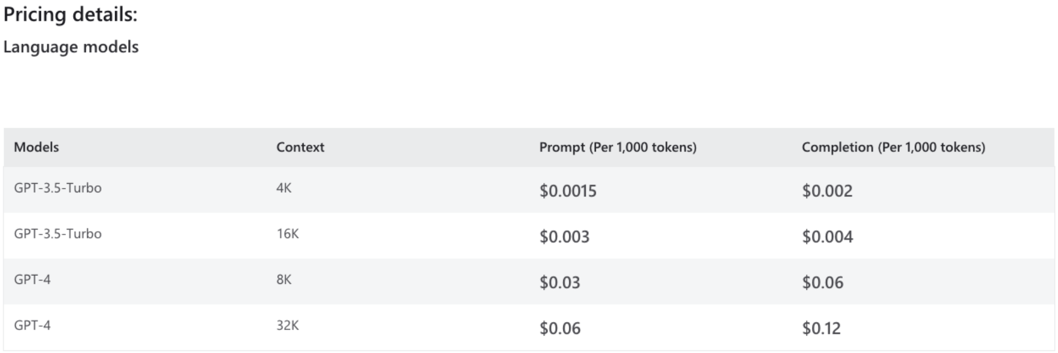
```r
with(d_train_merge_2, table(relevant_ra, relevant_gpt4, useNA = "ifany"))
```

```
##            relevant_gpt4
## relevant_ra  0  1
##           0 20  0
##           1  0 20
```

100% CORRECT!

Takeaway

ChatGPT 4 is much more expensive than GPT 3.5. But it is probably worth the money.



Pricing details:

Language models

Models	Context	Prompt (Per 1,000 tokens)	Completion (Per 1,000 tokens)
GPT-3.5-Turbo	4K	$0.0015	$0.002
GPT-3.5-Turbo	16K	$0.003	$0.004
GPT-4	8K	$0.03	$0.06
GPT-4	32K	$0.06	$0.12

Source: Microsoft Azure

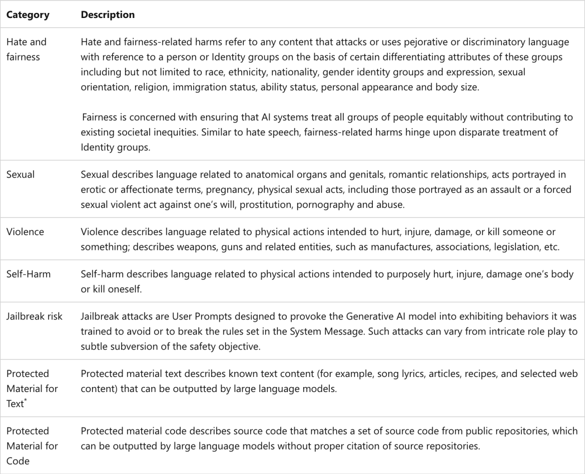
Content moderation

ChatGPT may refuse to give a response if the input violates its content moderation policy.

Below is the error message you get if it happens (for the Azure version of ChatGPT)

> *The response was filtered due to the prompt triggering Azure OpenAI's content management policy. Please modify your prompt and retry. To learn more about our content filtering policies please read our documentation: https://go.microsoft.com/fwlink/?linkid=2198766*

What input can be content-moderated



Category	Description
Hate and fairness	Hate and fairness-related harms refer to any content that attacks or uses pejorative or discriminatory language with reference to a person or Identity groups on the basis of certain differentiating attributes of these groups including but not limited to race, ethnicity, nationality, gender identity groups and expression, sexual orientation, religion, immigration status, ability status, personal appearance and body size. Fairness is concerned with ensuring that AI systems treat all groups of people equitably without contributing to existing societal inequities. Similar to hate speech, fairness-related harms hinge upon disparate treatment of Identity groups.
Sexual	Sexual describes language related to anatomical organs and genitals, romantic relationships, acts portrayed in erotic or affectionate terms, pregnancy, physical sexual acts, including those portrayed as an assault or a forced sexual violent act against one's will, prostitution, pornography and abuse.
Violence	Violence describes language related to physical actions intended to hurt, injure, damage, or kill someone or something; describes weapons, guns and related entities, such as manufactures, associations, legislation, etc.
Self-Harm	Self-harm describes language related to physical actions intended to purposely hurt, injure, damage one's body or kill oneself.
Jailbreak risk	Jailbreak attacks are User Prompts designed to provoke the Generative AI model into exhibiting behaviors it was trained to avoid or to break the rules set in the System Message. Such attacks can vary from intricate role play to subtle subversion of the safety objective.
Protected Material for Text*	Protected material text describes known text content (for example, song lyrics, articles, recipes, and selected web content) that can be outputted by large language models.
Protected Material for Code	Protected material code describes source code that matches a set of source code from public repositories, which can be outputted by large language models without proper citation of source repositories.

Source: https://go.microsoft.com/fwlink/?linkid=2198766

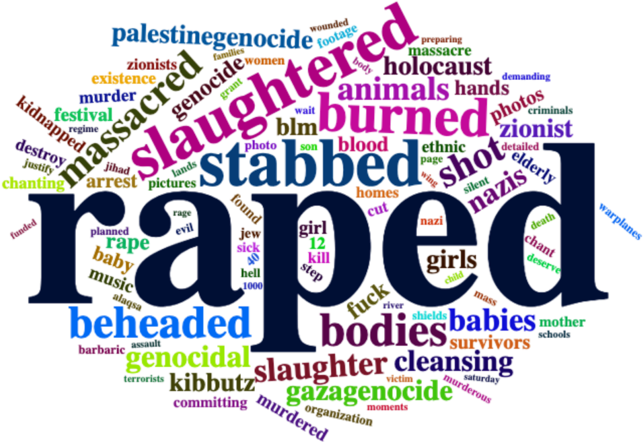
Case

A recent advisee's project on the Israel-Palestine conflict

- Used ChatGPT to label tweets containing keywords related to the recent Israel-Palestine conflict
- 15% of the tweets submitted for classification gets **NULL** results because of content moderation

Case (con'd)

What words distinguishes content-moderated tweets from the others? (Odds Ratio)



Reflection

The content moderation policy may limit our ability to use ChatGPT on studies about discourse related to **wars, conflicts, and violence**!

Recap: A 4-Step Workflow

- **Step 1**: Create instructions and a “training” set
  - Write instructions
  - Create a “training” annotated set
- **Step 2**: Apply ChatGPT to the “training” set
  - Give ChatGPT the instruction
  - Use ChatGPT to annotate the “training” set
  - Clean ChatGPT’s outputs
- **Step 3**: Evaluate ChatGPT’s performance
  - Examine how well ChatGPT’s annotation agrees with the “training” set
  - If satisfactory, go to Step 4
  - If not satisfactory:
    - Examine cases where human coders disagree with the machine
    - Modify instructions
    - Check the “training” set
    - Go back to **Step 2**
- **Step 4**: Apply ChatGPT to annotate documents outside the “training” set.

Other General Tips

- Usually, the most difficult part is figuring out what **YOU** want the machine to do for you.
- Test on small samples to avoid unnecessary costs.
- Be open to being persuaded by the machine. Ask the machine to explain reasons if you are stuck.

Frequently occurred technical errors

- Rate limit
- Empty strings (usually after text pre-processing)
- Content moderation

Generalizability of the Skills Learned about ChatGPT

- Tools come and go
- Long live Generative AIs
- Skills with ChatGPT can be generalizable to other Generative AIs
- You sacrifice things for simpler operation
  - Hand over more control to machines (and the companies selling them)
  - Less interpretability
  - Less reproducibility
- If possible, use smaller, open-sourced, and locally-executable models