Exercise Set 1: Prompting

Version 2

Instructor: Daniel Karell (daniel.karell@yale.edu)

January 2026

Overview

The goals of this activity are to gain (1) hands-on familiarity with prompting techniques (some of which we have read about) and (2) experience utilizing large language models (LLMs) to label observations in a dataset and extract information from text, which are common tasks in social science research.

We will be using a dataset provided by Armed Conflict Location and Event Data (ACLED). ACLED is a non-profit organization that collects data on violent conflict and protests around the world. It organizes and publishes these data, along with a codebook, or a guide explaining the data. For the exercises below, we will utilize both a sample of ACELD's data and its codebook.

When working through the example code in the following section(s), you do not need to submit answers to any questions in the text. These questions are meant to help you reflect on what is happening in the example analysis. Try to answer them to yourself to check your understanding. You only need to submit answers to the questions and prompts in the "Exercises" section below.

Resources

This is a list of links to external resources which are mentioned in the following instructions:

- Getting started with RStudio:
  - https://docs.posit.co/ide/user/ide/get-started/
- Using `R Markdown`:
  - https://rmarkdown.rstudio.com/lesson-1.html
- Mistral, the source of the LLM:
  - https://mistral.ai/
- The vignette about how to use the `tidyllm` package:
  - https://edubruell.github.io/tidyllm/articles/tidyllm.html
- Information about `tibbles`, from the `tidyverse`:
  - https://tibble.tidyverse.org/articles/tibble.html
- The data codebook from the Armed Conflict Location and Event Data project
  - https://acleddata.com/knowledge-base/codebook/

The Teaching Fellow and Instructor are also resources! Please feel free to ask for help as needed. Do not wait until the last minute.

Preparing to use a large language model

Most of the state-of-the-art large language models (LLMs) are created and maintained by profit-driven companies. Researchers can access and use these models, but only after registering accounts with the companies and, in some cases, paying fees. In addition, researchers have very little insight into these models' training data and training procedures. These kinds of models are called "closed" models. Some of the most well-known "closed" models are offered by OpenAI.

I will not ask you to use closed models because it would require payment, but, as I explain below, you are free to use them if you have an account that you decided to establish independently of this course.

There are some "open" state-of-the-art models. Several of these models have also been created by private companies, but they are free to use, can be downloaded to a user's computer (and thus used privately), and offer a lot more insight into how they were constructed. Some of the most well-known "open" models belong to Meta's Llama family of models. However, downloading a Llama model to run on a personal computer or laptop requires computing resources that not every student may have access to.

Since I do not want to assume that students have the necessary computing resources, I will ask you to use the application program interface, or API, of one company that offers several open models, Mistral. This means that we will use code to query Mistral's models while they are hosted on its servers. The upside is that we do not have to use any personal computing resources. The downside is that you will have to create and obtain an API key, or a unique alphanumeric sequence that allows Mistral to track your usage of its models. You can use either your own API key or share one with other students. The latter option means that the API key's daily usage limits will potentially be split among the students using the key.

Note that while I will focus on using Mistral's models, you can use other models. I elaborate below.

To obtain your personal API key, you will need to create an account with Mistral. To do so, go to the Mistral website, https://mistral.ai/. Click on "Try the API" in the upper right-hand corner. Create a free account. Then, once on the "La Platforme" dashboard, click on "API Keys" on the left-hand sidebar. Now on the "Your API Keys" page, click on the black button labeled "Create new key" and go through the few steps to create a key. I recommend creating a unique key only for this class's assignments; you can set the expiration date to the end of the semester. Once you have created your key, copy and store it somewhere secure (as Mistral will not show it you again). You will use the key in this assignment, as well as for future assignments. Do not share you key with anyone. You can delete existing keys and create new ones at any time.

It is not a requirement that you create an account with Mistral. If you do not want to create an account, please contact the Instructor and you will be provided with a shared, class-wide API key.

Note that there are usage limits on your using your key. If your key does not successfully call the Mistral API when working on the exercises below, or if you get blank responses, wait a day before trying it again. If the problem persits, contact the Teaching Fellow or Instructor.

Finally, if you want to learn more about the `R` package we will use, called `tidyllm`, you can find the package vignette at this URL: https://edubruell.github.io/tidyllm/articles/tidyllm.html. For details about the package's functions, see: https://cran.r-project.org/web/packages/tidyllm/tidyllm.pdf.

Prepare your workspace

We will now start using code to work with LLMs. Please ask for help if you run into a problem or would need for explanation about what to do. Do not wait until the last minute to ask for help!

First, install the package that we will use for accessing the Mistral API, `tidyllm`. You only need to install this package once (until you update your `R` software). However, at some point in the future, you should re-install the package after it has significant updates.

```
## install `tidyllm`
install.packages("tidyllm")
```

Then, execute the following line of code so that we can use the functions in the `tidyllm` library:

```
library(tidyllm)
```

The next step is to specify the API key for Mistral. Please replace YOUR-MISTRAL-API-KEY in the following line of code with your own API key such that your API key is inside double quotations.

Note that if you want to use models from a different source, you should change the following code, as well as the code in the next code chunk. For details on how to do this for various models, refer to the `tidyllm`

vignette at this URL: https://edubruell.github.io/tidyllm/articles/tidyllm.html.

```
Sys.setenv(MISTRAL_API_KEY = "YOUR-MISTRAL-API-KEY")

# For example:
# Sys.setenv(MISTRAL_API_KEY = "a1b2c3d4e5")
```

We can see all of the Mistral models that are available by running the following line of code:

```
print(n = 67, mistral_list_models())
```

We are going to start a conversation with Mistral using the `llm_message()` function. The text we enter into this function are the *prompts*. We can specify the model name within the parenthesis in the function `mistral()`. The default model that is accessed from Mistral is called `mistral-large-latest`:

```
conversation <- llm_message("Hello, how are you?") |> chat(mistral())
```

We can continue the conversation with the following syntax:

```
conversation <- conversation |> llm_message("Good, how are you?") |> chat(mistral())
```

We can utilize the following syntax to print the reply from Mistral:

```
get_reply(conversation)
```

We can also use the last_reply function:

```
last_reply(conversation)
```

Note that when you make many queries to the API, such as prompting it multiple times in a row, it can be helpful to pause between each request. When developing this activity, I found that adding a pause between each request in a series of requests decreased the number of errors messages I would otherwise sometimes receive (presumably because of a usage limit).

To add a pause in a script, use the following code. In this example, the code is adding in a three second pause.

```
Sys.sleep(3)
```

Finally, we can compile all of the conversation into a dataframe using the `as_tibble()` function from the `tidyverse` package. Note that this format reveals the *system message* prepended to the interaction with the LLM (as the `system` value), and it reports the *prompts* as the `user` value.

```
## first execute the following line if you have not
## previously installed the `tidyverse` package
# install.packages("tidyverse")

library(tidyverse) ## this library enables the `as_tibble()` function
conversation |> as_tibble()
```

Please see the `tidyllm` vignette to learn more about how to use the package: https://edubruell.github.io/tidyllm/articles/tidyllm.html.

Next, access the ACLED dataset from the `Files/exercise_sets/activity1/` folder on Canvas. The file is called `acled_LatinAmerica_sample.csv`. Below is code on how to read `.csv` file into `R` if the coding script file is in the same directory level as the datafile.

```
# library(tidyverse) # the function can be found in the `tidyverse` package
df <- read_csv("file.csv")
```

If the `.csv` file is not in the same directory as the coding script file, you can provide the entire path as an argument to the `read.csv()` function as follows:

```r
# library(tidyverse) # the function can be found in the `tidyverse` package
df <- read_csv("path/to/your/file.csv")
```

Finally, view the dataset and become familiar with what information it contains in which variables. To complete the assignment, you will use information from these columns: `event_date`, `actor1`, and `region`, `country`, `admin1`, `admin2`, and `notes`.

```r
# library(tidyverse) # the function can be found in the `tidyverse` package
glimpse(df)
```

If you need help downloading and reading the dataset into the `R` environment, please ask.

Using different models

As mentioned earlier, I suggest that you use Mistral's models because they are "open" and it is easy to set up a free account. However, you are free to use other LLMs if, for example, you are already using another model for your ongoing projects or if you encounter difficulty with Mistral's models. https://edubruell.github.io/tidyllm/articles/tidyllm.html.

Fortunately, the `tidyllm` package makes it easy to use a variety of models, including those in the Claude, Gemini, OpenAI, and DeepSeek families. For instructions on how to do this, see the package's vignette: https://edubruell.github.io/tidyllm/articles/tidyllm.html.

If you do decide to use a model to respond to a question that differs from the one in the question's instructions, simply note the model that you did use in your response. For example, write "To answer this question, I used models X and Y, provided by Z."

Instructions for submission

Please submit your completed problem set as `PDF` or `HTML` file generated using an `R Markdown` document. For a beginner's guide to creating an `R Markdown` document, see https://rmarkdown.rstudio.com/lesson-1.html.

A completed assignment includes: (a) your code; (b) the output of your code; and (c) a written answer that provides a response to each question by explaining and interpreting the output that your code produced. Upload the completed problem set to the "Assignments" page of the class's Canvas site. This activity will be graded on a 100-point scale.

Please note! When you knit your assignment (either to `HTML` or `PDF`), all your code will run again. This may have important implications: if a new run of your code produces slightly different output, then comparisons to other results and/or your discussion of the results (the text you've written) may no longer be aligned.

To avoid this issue, please adopt the following workflow. First, as you develop the code to address each exercise, write it in a Markdown code chunk:

```r
# ```{r}
# example <- code(about = something)
# ```
```

Then, once you have obtained the results you consider "final", store or save your results as an R object.

```r
# ```{r}
# save(output, file = "classwork\saved_results\output.Rda")
# ```
```

Next, when you are ready to knit, add a command to *each* code chunk that prevents the code from running, or being evaluated. The command is `eval=FALSE`, and it goes it in the curly brackets, like this:

```
# ```{r, eval=FALSE}
# example <- code(about = something)
# ```
```

The above command will ensure that the code chunk will not run, or "evaluate", during knitting. However, you still need to show your output or results when you knit the document for submission. So, after each code chunk that shows the code you used to get your results, include a new code chunk that simply prints the results. Doing this will put into the document: (a) your code (which will not run when you knit) and (b) your output. To show the output, something like the following chunk should work, as long as `eval=TRUE` since you want *this* code chunk to run!.

```
# ```{r, eval=TRUE}
# load(file = "classwork\saved_results\output.Rda")
# print(output)
# ```
```

Please let the teaching team know if you have any questions.

Exercises

Part 1

1. Using your own prompting strategy, try to extract three pieces of information from the text associated with the 50 observations in the ACLED dataset: the event's date, a location associated with the event, and the main actor involved in the event. Specifically, ask the default Mistral LLM (`mistral-large-latest`) to parse through the text that is provided in the `notes` column of the dataset and identify the three facts. You may opt to use a loop to feed the events to the LLM one at a time, or you may try providing all events at once. Whichever strategy you chose, report your code and results (in a table), and provide an explanation of what you did. [10 points]

2. Randomly sample 20 of the observations (potentially using the `sample_n()` function from the `tidyverse` library), and assess how effective the LLM was in identifying the event's date, a location, and the main actor by "manually" comparing the extracted information to the text of the `notes` column, as well as the columns labeled `event_date`, `actor1`, and `region`, `country`, `admin1`, and/or `admin2`. You can decide how to report effectiveness. It can be as simple as a table comparing the LLMs' identifications with the information in the ACLED dataset (with a value indicating the results of each comparison). Note that your prompting strategy might mean that the number of possible dates, locations, and actors that the Mistral language model identified were unbounded. Explain your comparison and assessment of the results. [10 points]

Part 2

At this point, start a new session (or "conversation") with the default LLM (`mistral-large-latest`).

Now, download and peruse the ACLED codebook. You can find it in the `Files/exercise_sets/activity1/` folder on Canvas. The file is called `acled_codebook.pdf`. It can also be found at https://acleddata.com/knowledge-base/codebook/

3. Utilizing the ACLED codebook, design a way to (hopefully) improve your prompting strategy. You may opt to include specific definitions or possible response categories from the codebook or try some other technique. Explain how you will improve your prompting strategy, and why. [10 points]

4. Now, using your updated prompting strategy, repeat the first exercise: use the model to try to extract the event date, a location associated with the event, and the main actor involved in the conflict from each of the 50 observations' `notes` text. Show your code. [10 points]

5. Using a random sample of 20 observations, compare the results of using (1) your original prompting strategy and (2) prompts integrating information from the ACLED codebook to (3) the "ground truth" information in the ACLED dataset. Report your comparison, which can be as simple as constructing a table

showing all the results and values indicating how they do or do not differ. Explain which of your prompting strategies seemed to fare better, and why you arrived at this conclusion. Save this sample of 20 observations and these results; you will use them in Part 4. [10 points]

Part 3

At this point, start a new session (or "conversation") with the the default LLM (`mistral-large-latest`).

6. Update your prompting strategy once again. Select 10 examples of events and provide them to the model through the prompt. Specifically, view the `event_type` column, find examples of different kinds of events, and then (for the different kinds) give both the `event_type` label and the corresponding `notes` text to the LLM. In effect, you will "teach" the model how to identify particular event types from the text. After selecting the examples, incorporate them into a new prompting strategy that asks the model to (1) learn from the examples and then (2) parse through the text that is provided in the `notes` column of the dataset to identify the type of event for each 50 observations. Show and describe the prompting strategy. [10 points]

7. Once again, randomly sample 20 of the observations and evaluate how effective the LLM was in identifying the type of event by "manually" comparing them to the text of the `notes` column, as well as the information in the column labeled `event_type`. As before, you can decide how to report effectiveness. Explain your comparison and assessment of the results. [10 points]

Part 4

At this point, start a new session (or "conversation") with the default LLM (`mistral-large-latest`).

8. Now, practice utilizing "chain-of-thought" prompting, which we read about, and which you can see an example of in the appendices of Stuhler, et al. 2025. Plan how you will implement this technique. Describe and explain your plan. [10 points]

9. Implement your "chain-of-thought" prompting plan to repeat the first exercise. Use the model to try to extract the event date, a location associated with the event, and the main actor involved in the conflict from each of the 50 observations' `notes` text. Show your code. [10 points]

10. Finally, using the sample of observations from Question 5, compare the results of using (1) your "chain-of-thought" prompting strategy, (2) the prompting strategy that fared the best in Question 5, and (3) the "ground truth" information in the ACLED dataset. Report your comparison and explain whether your implementation of "chain-of-thought" prompting performed better, worse, or the same as the best prompting strategy from Question 5. Discuss why you think the best-performing prompting strategy worked better than the others. [10 points]