LLMs

OpenRouter is a service that provides access to many LLMs through a web app and an API. Some are paid, others are free up to a certain number of queries. Set up an account at OpenRouter using your GitHub account and create an API key[1]. I suggest setting your usage limit to $0 and using only the free models. Copy and paste your API key into a file someone on your computer.

1. Click on the "Models" tab to see the available models. How many total models are there? How many are free to use?[2]

2. Search for and then click on a model called "Google: Gemma 3 27B (free)" and read the documentation page. What are its capabilities?

At the bottom of the page, it shows example API calls in several programming languages. Copy the `curl` example paste it into a JSON file in Positron (create a new file and give it a .JSON file extension). You can select all of the contents of that file, right click it, and select "Format Document" to make it more readable.

3. What is the endpoint URL for the API call?

4. "Headers" (indicated with `-H`) are similar to parameters included in the URL of a GET request. What are the names of the header that you will pass and what are their values?

5. The data that your computer sends with the request is flagged with `-d` and is sent as a JSON object. What are the names of the keys in this object?

6. Replace `$OPENROUTER_API_KEY` with your API key and then run the `curl` command in your terminal. Copy and paste the response in your JSON file and format the document again to make it easier to

[1] If it doesn't prompt you to create an API key right away, go to your account settings (top right corner) and find the "Keys" section.

[2] Use the filters available in the left sidebar. Note that some models are free up to a certain number of queries per day, so be judicious.

read. What is the name of the key in the response that contains the text generated by the model?

7. List 5 other keys and values that are present in the response besides the text generated by the model.

8. In the space below, write a new `curl` command that only sends the prompt "The capital of Canada is".

9. Send the `curl` command. Beyond the content of the text generated by the model, how does the response compare to the previous one?

10. Return to OpenRouter and click on the "Chat" tab. Select the "Google: Gemma 3 27B (free)" model and enter the same prompt ("The capital of Canada is") in the text box. How does the response compare to the previous one?

11. Create an R script and load `ellmer`. Create a chat object using `chat_openrouter()` save it to `or1`. You will need to pass your API key and the model name as arguments (the model name is the same as in the `curl` request). Write your code below then run it.

12. Use `or1` to send the prompt "The capital of Canada is". What is the response?

13. Use `or1` to send a second prompt: "What is the most popular sport?". Write the gist of the response below.

14. Create a second chat object called `or2` using the same model. Send the same prompt: "What is the most popular sport?. How does the response compare to the one from `or1`?

15. Print `or1` and `or2` at the console. How does this explain the difference in responses?

(Optional but cool): Most models have a standard set of other parameters that you can send that will determin how the model generates its text response. You can explore these by reading `?chat_openrouter()`. Max tokens is the maximum length of the response (in tokens, where 1 token is roughly 4 characters of text). Temperature controls how "creative" the model is allowed to be: a temperature of 0 means it will always pick the most likely next token, while values closer to 2 will make it more likely to pick less probable tokens, resulting in more varied and creative responses. You pass these parameters as a named list to the `params` argument of `chat_openrouter()`.

16. Create a new chat object called `or3` that uses the "google/gemma-3-27b-it:free" model but sets the `temperature` parameter to 2 and the `max_tokens` parameter to 500. Use `or3` to send the prompt "Describe new types of punctuation and the emotions they evoke". Resend initialize `or3` and send the same prompt several times to get a sense of how the responses vary. Change the temperate of `or3` to 0 and send the same query several times. Write your observations below.

LLMs in IDEs

The notions of code-completion, contextual chat, and agents are key features of the manner in which LLMs are being integrated into IDEs. This PS is optional but encouraged for those with access to Positron (though note that you can set up code-completion in RStudio as well).

Follow the instructions posted on Ed to activate code-completion, Positron Assistant and DataBot. Experiment with all three features in the IDE. In particular, try the following tasks:

1. Use code-completion to write a function that takes a numeric vector and returns its mean and standard deviation as a named list.
2. Use Positron Assistant to help you answer the questions to one of the early problem sets in this course. Experiment with toggling between Ask, Edit, and Agent mode.
3. Task DataBot with doing one of projects, such as Energy or Gerrymandering. Experiment with loading data files, summarizing them, and creating visualizations. Carefully inspect the output and see how many errors you can detect.

Record your thoughts and experiences below. What worked well? What didn't? How do you see these tools fitting into your workflow as a data scientist?

A World of Abstraction

In lecture we looked through the log of the exchange of text data between me on my laptop and the LLM (i.e. DataBot).

- Conversation HTML file
- Conversation JSON file

The first substantive (although poorly spelled) request that I made was: "I'm working on my gerrmadering prooject. could you please load the g24 sv precinct data and tell me a bit about it?". We saw how the result was impressive (it found the issue with `***`) but also faulty: it didn't catch that some of the rows were totals of other rows, so it greatly overstated the number of votes.

In the space below, rewrite this prompt in a way that you think will give the model a better chance of finding the issue with the row totals. Your rewrite shouldn't show that you already know the answer (e.g. "some rows are totals of other rows"), but it should guide the model to think more carefully about the data.

If you have access to DataBot, you can start it in your project directory (or a new clone of the project repository) and try your prompt there. Is the response fully correct? Are there any other issues that occurred?