Advanced Methods in Text Analytics
Exercise 8: Large Language Models - Part 2

Daniel Ruffinelli

FSS 2025

In this exercise, we use code to explore some basic uses of pre-trained LLMs. The code runs well on CPU in a laptop with 16GB of RAM, where the most expensive tasks take about 10 seconds. Still, feel free to modify the code to better suit your resources.

1 Making Predictions with LLMs

In this task, we look at how to make predictions with pre-trained LLMs.

(a) We first load two pre-trained LLMs: GPT Neo of 1.3B parameters and released in 2021 by EleutherAI, and Llama-3.2-1B, a model released in late 2024. We also load their corresponding tokenizers, so we can encode and decode text to interact with the models. Run the given code to load the models. Then give a basic description of their architecture as described by the output in the code, e.g. number of transformer layers, size of hidden states, size of the vocabulary.

(b) Next, we write a simple prompt and tokenize it with the tokenizer of any of the models we loaded. Run the give code and inspect the object given as output. What type of object is it? What components does it have? What do they represent? Go over some of HuggingFace's documentation and add some code to explore the output in order to find the answer. How many tokens are required to encode your prompt?

(c) We now use the model to predict the next token given your prompt from the previous subtask. Run the given code and inspect the object returned by the model. What type of object is it? What components does it have? What do they represent? As before, use HuggingFace's documentation and run some code to find the answer. Where is the distribution over vocabulary that we should use to predict the next token?

(d) Finally, we obtain the top k tokens predicted by a model. For that, fill in the function `get_top_k_tokens` in the given code. To decode tokens, use the function `batch_decode`. You can learn about it here. Do the top k tokens make sense given the prompt you tested? Are there any differences in the output of the two different models? What could this function `get_top_k_tokens` be used for?

2 Prompting

In this task, we test the impact that different prompts have on the predictions made by LLMs.

(a) Let's inspect the top k tokens predicted by the models from the previous task. Run the code and test the output of the model when given the different prompts in the code. Try different prompts of your own as well. What do the top k tokens look like? Do they contain the answer to the questions? Can we use them to make predictions? If so, how? If not, can you find a prompt that results in top tokens we can actually use for predictions?

(b) Let's now try a more specific type of prompt: in-context learning (ICL) prompts. Look at the tasks defined in the code, run it and see whether the format of the prompts matches what we know about ICL. What are the different components of an ICL prompt?

(c) We now try our ICL prompts with our loaded pre-trained models. Let's start with zero-shot prompts and no instruction. Do the models understand some tasks? Which ones? Then add a few demonstrations and compare with the zero-shot setting. Do some tasks benefit from having demonstrations more than others? Do both models perform equally well? Does the prompt template make a difference?

(d) Try adding natural language instructions to the task. Does it help? If so, on which tasks? And on which models?

(e) We now add a new model, an instruction-tuned variant of the same Llama we have been testing. Try all prompts so far starting with the natural questions and all the way to the ICL prompts with natural language instruction. Do you see any difference compared to the other two models?

3 Generating Longer Responses

In this task, we explore how to generate longer responses with LLMs and what these responses look like under different settings.

(a) Complete the function `generate` given in the code using everything we've learned so far. Make sure to use greedy sampling.

(b) Try the `generate` function with different prompts. Do the models produce reasonable responses? Is there a difference between models? Does the output of some models make more sense now that we are decoding more tokens?

(c) HuggingFace models already include a `generate` function similar to ours, which we will use from now for simplicity. Read about that function and try using it to generate responses using different sampling methods. What differences do you observe between them? Note that the output of this functio already includes the given prompt, i.e. it isn't generated by the model.

(d) We now contruct a prompt that simulates a dialogue system as a way to see if this type of prompts make a difference with model outputs. Inspect the function `construct_chat_prompt` and run the code to test it. Try different system prompts and generate responses of different lengths. Use HuggingFace's `generate` function, as it is more efficient. Are the responses as you expected? If not, what issues do you see? And how could we solve them?