COS 470: Introduction to Information Retrieval
Assignment 4: LLM Reranking

Author: Abigail Pitcairn
Version: November 12, 2021
Github: https://github.com/abbyPitcairn/470assignment4.git

Technique:

The base search for this assignment was tf-idf search from assignment two. This was one of my only functional searches that I had available to implement. Tf-idf search returned a dictionary of dictionaries containing document IDs and tf-idf scores for each query. Unfortunately, this base search was not very precise; when evaluated with the Ranx library, tf-idf search returned P@1 of about 1%. This should be taken into consideration when evaluating reranked results.

I used HuggingFaceTB/SmolLM2-1.7B-Instruct decoder-only LLM to rerank the top 100 results from tf-idf search. I chose this model because it is compact and lightweight, hopefully allowing for a faster execution time than a larger model, and it has improved instruction following and reasoning from its predecessor.

Prompt 1:

```
    prompt = (f"Query: {query_text}\nDocument: {doc_text}\nRate the relevance of
this document on a scale from 1 (least relevant) to 5 (most relevant). Return
only the number.")
```

Prompt 2:

```
    prompt = (f"For this query about travel, {query_text}, how would you rate
this document, {doc_text}, in terms of relevance on a scale from 1 to 5, with 5
being the most relevant? Return only the number.")
```

I did not use a system prompt because it was too complicated to implement with the debugging issues I was already handling.

The first prompt was vague and did not provide any context, examples, or explanation. It simply provided the query and document text and asked to rank their relevance from one to five. The second prompt did not provide examples or explanation, either, but it did include context by informing the LLM that these queries are related to travel. With this additional information, we might expect prompt two to perform better than prompt one.

For the model's tokenizer, I used a left padding-side, compatible with the decoder-only model, and set `return_attention_mask` to True; these changes helped clarify the difference between important and unimportant information for the model and eliminated some console warning messages, although the attention mask could be further implemented for better results.

If I had more time and processing power, I would experiment with a few-shot prompt like this using a relevant query-document pair from the qrel file:

```
    prompt = (f"For this query about travel, {relevant_query_example},
how would you rate this document, {relevant_document_example}, in
terms of relevance on a scale from 1 (least relevant) to 5 (most
relevant). Return only the number. Response: 5.
For this query about travel, {query_text}, how would you rate this
document, {doc_text}, in terms of relevance on a scale from 1 (least
relevant) to 5 (most relevant). Return only the number.")
```

Results by Ranx:

Model Type	P@1	P@5	nDCG@5	MRR	MAP
Prompt 1 (no context)					
Prompt 2 (some context)					

Unfortunately, due to time restraints, results were not fully generated for this assignment. However, in theory, we would expect the prompt 2 model to outperform the prompt 1 model. Although both prompts are zero-shot, meaning they do not provide examples or chain-of-thought reasoning, prompt 2 provides more context about the nature of the task than prompt 1 does.

Challenges:
The most difficult parts of this assignment were tracing data through many method calls and the section of the LLM reranking method where the scores were generated and saved:

```python
# Initialize dictionary of scores by document.
scores = {}
for i in range(0, len(prompts), batch_size):
   batch_prompts = prompts[i:i + batch_size]
   inputs = tokenizer(batch_prompts, return_tensors="pt", padding=True,
truncation=True)

   with torch.no_grad():
       outputs = model.generate(inputs["input_ids"], max_new_tokens=10)

   for j, output in enumerate(outputs):
       response = tokenizer.decode(output, skip_special_tokens=True).strip()
       # Set the score to an integer representation of the model's ranking
response.
       try:
           score = int(response)
       # If no score is given, default to 0.
       except ValueError:
           score = 0
       scores[doc_ids[i + j]] = score
```

In this section, it was difficult to trace what exactly was happening and the error messages were long and not written clearly enough for someone like myself with only a vague understanding of what was supposed to be happening in the code.

Many error messages were for trivial mistakes, such as not calling the query from the queries file with the correct syntax or loading the json files twice. For example, in the `rerank_documents_with_llm` method, `queries[query]` were called instead of `queries[query_id]`, which was not discovered until ten minutes into running the program. With the slow speed of the system, these small errors ate up a large chunk of the assignment timeline.

To avoid creating unnecessary errors in saving and loading results, tf-idf search was rerun each time the system was tested.

References

HUGGINGFACETB/Smollm2-1.7B-instruct · hugging face. HuggingFaceTB/SmolLM2-1.7B-Instruct · Hugging Face. (n.d.). https://huggingface.co/HuggingFaceTB/SmolLM2-1.7B-Instruct

Zhu, Y., Yuan, H., Wang, S., Liu, J., Liu, W., Deng, C., Chen, H., Liu, Z., Dou, Z., & Wen, J.-R. (2024, September 4). *Large language models for Information Retrieval: A survey*. arXiv.org. https://doi.org/10.48550/arXiv.2308.07107

Wolfe, C. R. (2024, September 26). *Masked self-attention: How llms learn relationships between tokens*. Stack Overflow. https://stackoverflow.blog/2024/09/26/masked-self-attention-how-llms-learn-relationships-between-tokens/