Elijah Chou

CS 584: Human AI Interaction

Dr. Chinmay Kulkarni

10/15/2023

Chain of Density Summarization Report

Approach/Explanation:

My approach to the "base_summary(text)" function was to instruct the GPT-3.5 model that it was a "helpful assistant that summarizes text." Since this was the first time that I worked with OpenAI API in Python, I relied on ChatGPT to write the API call to the gpt-3.5-turbo model and print out the result. I then moved on to adjust the prompt by instructing the model that it is "a helpful assistant that generates entity-sparse summaries after being given a text." This way, I would more likely be able to get a summary that is more "entity-sparse" than the original.

Moving on to the "extract_entities(text)" function, I reused the previous OpenAI API model call and instructed the model with the prompt that it is "a helpful assistant that extracts and ranks entities from the given text." I also specified in the system prompt that it should be giving a response in a Python list format where the entities are ordered by their rankings. The reason why I specified this was to simplify the processing of the model's output into a Python list of entities that I would need to return for the output of this specific function.

Realizing that I needed to call the LLM model in multiple functions in the assignment, I created a new helper function called "call_LLM_model(msgs)" that would take the messages input needed for a ChatCompletion.create call and make an API call to the gpt-3.5-turbo model using the predefined model and temperature (set to 0 for reproducibility). I then simplified and cleaned up my base_summary and extract_entitites functions with the new self-defined function.

For increase_density, I instructed the LLM with a prompt telling it that it was an expert in using abstraction, fusion, and compression techniques to increase entity densities of summaries given to it. In the user message, I provided it with the summary, a list of entities from extract_entities, and the total number of characters in the original summary. In the prompt, I asked the model to rewrite the given summary to incorporate any entities in the list that are not present in the given summary, and that it should write the new summary so that it is at most the total number of characters in the original summary.

In order to evaluate the final summary, I implemented the evaluate_summaries(summaries) function so that it take a dictionary of dictionaries, with each dictionary containing a "text" key and a "summary" key containing values of the original text and the generated summary, respectively. For each dictionary, it will call extract_entities on the original text, and then count how many of returned entities are found in the summary. The result returned by my evaluate_summaries is also a dictionary of dictionaries, where each dictionary contains the number of entities in the text, the number of entities found in the summary, and the "score" calculated by dividing the summary entity count by the text entity count.

Observations/Challenges:

Out of curiosity, I passed all my summaries, including ones in between increase_density calls, into evaluate_summaries. I noticed that, even when instructing the model to increase the number of

entities in the summary from a list without reducing the existing ones, the number of entities wouldn't go up after it increased from the first increase_density call. I thought this was because the model incorporated as many new entities as possible given the summary length limit, and so it couldn't add more past a certain point. Regardless, I still was able to increase the number of present entities from the base summary to after calling increase_density multiple times. Another interesting observation I made was that, even though I called the same extract_entities function on the exact same text using the exact same prompt, the number of entities I get for the original text can vary from time to time. It can usually differ by 1-2 entities within and between every run. This is interesting because I thought that I should be getting the same results with the temperature set to 0.

In my initial attempt to produce a dense summary in a single step, I passed a prompt asking the model to first identify entities in the original text, then implement all of them in a summary for the text. When I evaluated this with the other base and final summaries, I noticed that this prompt would return a summary with less entities than even the base entity-sparse summary. I then altered the prompt to ask it to simply create a summary that included as many entities from the original text as possible, but it still performed worse. After reading through a few summaries for different texts, the final summaries obtained from increase_density seemed to have more information overall, including more details and facts than my dense_summary summaries. This helps to explain why my dense_summary summaries have less entities than the final summaries. A potential reason for this is that the entities extracted from the extract_entities function do not overlap or just are not as much as those identified in dense_summary prompt. I confirmed this by asking the model in dense_summary to also return the entities it identified, and it did indeed return a smaller total entity count than the extract_entities function.

One more adjustment I tested was to specify in both the extract_entities function and the dense_summary function what entities might be. I specified that entities could include people, places, etc., based on the example list that was provided to us on Discord, and I observed that the number of extracted entities went down, but the total number of included entities in the final summary went up relative to the total entities extracted from the original text. Doing this also helped my dense_summary function to produce a summary that included slightly more (1-2 more) entities than the base_summary function, so it did introduce an improvement but also highlighted how challenging it is to incorporate more entities in a single prompt.