The goal of this ongoing project is to leverage NLP and Machine Learning techniques to build a robust yet efficient Semantic Search Engine for Discover O-Chem, an online Chemistry Textbook built in PHP by Clark University chemistry professor Charles Jakobsche. Below will be details regarding the process, results, and takeaways from our first semester of work.

Brainstorm and Planning

There was an initial idea to improve the search feature for the website, but little idea or direction on how to actually go about completing this task. We had a dataset containing all the information from the textbook, structured by sentence and page number, as well as the current search engine that utilized a keyword feature for reference. This keyword system had some notable issues, primarily that the results it returned were based on the presence of certain words in a user's query, rather than what the query actually meant. This limited the scope of what user's could effectively look up, as overly specific queries were too sensitive for the system when they did not contain any of the notable keywords. Our goal was to improve the effectiveness of the search engine such that students had a tool that they actively wanted to engage with to improve the Organic Chemistry experience.

After some initial research, it became clear that a Semantic Search model would be an effective way to tackle this problem. The process for doing so was relatively simple: pre-compute embeddings for every sentence in the textbook, take the user's search query and send it off to some backend where an embedding is generated for it, then use some similarity metric to return the K most relevant sentences to the user's query. Of course during implementation there are a lot more things to consider, but that was the general blueprint for our system.

In regards to implementation the idea was to try some different methods for embedding generation, upping the complexity with each iteration through the process.

Implementation Process

The first method we used for embedding generation was a simple word frequency vector. The process for this was very simple. We identified all the unique words in the textbook, saved them to a dictionary, and ran the user's query through a simple method that tokenized and cleaned the text before giving it a numeric vector representation. The precomputed frequency embeddings were saved in a vector database for efficient querying over the cosine similarity metric. This method was by far the simplest, and also performed the worst. This method still identified similar sentences based on only the words present, which offered minimal improvement to the original search feature.

The next method was to use some kind of pre-computed embedding dictionary, such as GloVe. There were many issues when attempting to use GloVe based embeddings for this use case. For one the GloVe dictionary is massive, and we had a hard time dealing with its size in local memory. On top of that many of the technical chemistry terms were not present in the precomputed embeddings, which made it hard to generate meaningful embeddings for a lot of the text that relied heavily on chemistry based jargon. Ultimately this embedding method gave me the hardest time, and is one that I will have to go back and revisit.

The next embedding method was via a LLM. For this use case LLM based embeddings performed the best by far in regards to sheer accuracy, but also presented some challenges in regards to making efficient computations. To utilize the LLM I incorporated HuggingFaces Transformers library, which made things very streamlined and easy.

The final embedding method was via AWS using a cloud-deployed LLM. This performed slightly worse than the HuggingFace in regards to accuracy, but offered a huge upside for efficient computation and embedding generation.

Key Takeaways

While this project is not done yet, I have some interesting takeaways from my first semester of work. I think my biggest one is the challenge of balancing accuracy with efficiency.

…