Semantic Search with Embeddings

DSC 360 F25 Lab 03

A.J. Daniels and Madeleine Arganbright

Overview

The goal of this project was to create a semantic search engine that allows the user to inquire about a specific part from a public-domain book and in return receive the top five paragraphs from the actual book that they are referring to. Hit scores were reported if the intended segment from the book appeared in the top one or top five of our model's suggested matches. Building this involved chunking the book into clear sections, embedding each chunk to numerically compare with our search, and storing all pieces to separate files. We compared models and analyzed results using the book Moby Dick.

Problem

Using a single public-domain text file for Moby Dick, we first needed to determine how to separate the full book into small segments that could be searched for. Separating by paragraphs was not viable as some stretched across multiple pages and some consisted of only one line. Our objective was to build a chunker that split the text into roughly even length passages so we could perform vector embedding and find matches to our user query. The semantic search through our chunks file provides the user with the top five matches in order for their query, with each one containing the ID (number chunk out of 1269 total chunks), the actual text itself, and the sentence number the chunk starts and ends with.

Methods

Creating the chunker proved to be a difficult task but we ultimately found success by filtering for the number of sentences in each paragraph. Paragraphs were formed by concatenating lines in the text which occurred between blank spaces. If the paragraph had more than five sentences, it was split in half. If the paragraph had less than three sentences, those sentences were stored and added to the next chunk.

We had to implement vector embedding to assign numerical values to each chunk. Then we could use the same embedding on our search query and compute the distance between that and all of our chunks in the book with the smallest distance indicating the best match. We used Numpy to create a two-dimensional array, compute the dot products of the vectors being compared, and then return the top five. We used the top 5 indexes returned to find the corresponding chunk from our chunks.jsonl file, and then compared those to our expected chunk. We decided to automate the testing process by reading prompts from a file, and comparing them to the expected ids in the file.

Evaluation Protocol

We built our dev sets by looking through our chunks.jsonl file, finding random chunks, and building a prompt which related to the chunk. To create our supertest, we combined prompts and gold ids from our superteam partners, Chaney and Nathan.

Our Hit@1 was calculated by adding up all the times the gold idea appeared in the very top result for each test prompt, divided by the number of total prompts (so 6 for our original tests and 11 for the

combined superteam tests). Hit@5 was calculated by adding up all the times the gold idea appeared in the top5, divided by the number of total prompts.

Results

On just our tests (6 prompts and "gold ids" total- specifics listed in **tests_mxbaimodel.cvs** (baseline) **and test_nomicmodel.cvs** (variant)):

Model	Hit@1	Hit@5
Baseline*	0.16666666666666666	0.3333333333333333
Variant**	0.16666666666666666	0.16666666666666666

* Chunker without overlap, Model = mxbai-embed-large
** Chunker without overlap, Model = nomic-embed-text:v1.5

On our superteam's combined dev set (11 prompts total- specifics listed in **supertests_mxbaimodel.cvs** (baseline) **and supertest_nomicmodel.cvs** (variant))

Model	Hit@1	Hit@5
Baseline*	0.18181818181818182	0.2727272727272727
Variant**	0.36363636363636365	0.36363636363636365

* Chunker without overlap, Model = mxbai-embed-large
** Chunker without overlap, Model = nomic-embed-text:v1.5

Results Analysis

The above results show us that using the testing set of our super team with eleven prompts produced more accurate results as the model had more chances to perform correctly. As with any experiment, the more observations increases accuracy and reduces random error. With these superteam testing data, our variant embedding model "nomic-embed-text:v1.5" appears to be more consistently accurate than the "mxbai-embed-large" embedding model. It is interesting to note that using the baseline model in both experiments, the results were better with Hit@5. However, with the variant model in both experiments, the results were the same in Hit@1 and Hit@5. The highest hit score in the superteam test data with the variant embed model was approximately .36 meaning on average the intended chunk from the book was given in the top five 36% of the time.

Error Analysis

One of our dev sets was searching for the opening lines of the novel ("Call me Ishmael.."). Our prompt was "Find the passage in the book where Ishmael introduces himself." (We intentionally kept the prompt vague, to see if it would return the line we were looking for with very little information). The opening line never appeared in the top5, but it did return other lines in which Ishmael introduces himself, for example, top1 and top3 were, respectively: I, Ishmael, was one of that crew…." and "I say, Queequeg!

why don't you speak? It's I—Ishmael…" which were both instances in which Ishmael does introduce himself.

Similarly, when using the prompt "find the passage where Ahab offers a reward for the first sighting of the whale" returns "'And did none of ye see it before?' cried Ahab, hailing the perched men all around him… 'no, the doubloon is mine, Fate reserved the doubloon for me'", which is in reference to that scene, but doesn't directly come from that scene. If we elongated to include top 10 hits, it is very likely that the correct lines for both of these prompts would have shown up.

Conclusion

The experiments were only done on between six or eleven testing prompts and although the results we gathered did show us that the "nomic-embed-text:v1.5" embedding model seems to be more accurate, truthfully none of our experiments produced particularly satisfying results. This is likely due to a number of things that we could attempt to fix in the future. We would like to modify our chunker to see if incorporating an overlapping opening sentence in each prompt would improve the model. We also would have liked to have provided more prompts that were perhaps a bit more specific, as our model seemed to struggle with identifying passages which did not contain the specific words in the prompt. For example, one of our prompts asked the model to find the scene where the captain drops his trumpet in the water, but replaced the word trumpet with instrument in the prompt to see if it could extrapolate the meaning, but the model was not able to identify the correct passage. The lack of specificity and even the contents of Moby Dick may lead our model to returning perfectly reasonable estimated matches, just not the one we were intending. It would also be beneficial to try our search engine on a different book, and we could also perhaps be more specific with our prompts if we know the book better.

Acknowledgements and References

We would like to thank Chaney and Nathan for providing us with prompts for our testing. Thank you to Dr. Allen for helping us understand the vector embedding logic that we were then able to implement. We also used some code segments generated by the Google Ai, which are indicated on our code. All models used were pulled from the ollama.com website.