Final Project Writeup

Jeffrey Bailey

Natural Language Processing with Professor Tucker

5/16/2023

Creating an Academic Paper Browsing Tool using Natural Language Processing Techniques

Abstract

Introduction

The idea for this project came from a conversation with my research professor, who once told me that to stay up to date on his research field he casually reads through newly published chemistry papers every morning. With this habit, he stays updated on what research his peers are doing, and he is always bringing new ideas into the lab. Impressed with his academic thoroughness, I decided to try reading through published papers one morning and found it extremely difficult and uninteresting. Therefore, the goal for this project was to make a tool for the less-rigorous academic to easily browse published papers and stay up to date in their field.

Methods

First, I had to source a dataset that could be used to develop this paper-browsing tool. After combing through numerous academic literature datasets, I found that the Spacy scientific papers database would be the easiest to import and a good fit for my project. It can be imported with the 'datasets' library, and has nearly 400,000 published papers, roughly split between a PubMed dataset and an arXiv dataset. Due to computational limitations, I chose to randomly sample 2,500 articles from the arXiv set, which gave me a dataframe with raw text for each article's body and abstract.

Before working on the functions and process for the actual paper-browsing app, I had to design a database of relevant features created from my arXiv articles. I knew that I wanted the program to have a search engine function, a topic tagging and exploration function, and a browsing engine that allows a user to peruse suggested articles and be given fast recommendations for future articles to read. To implement this, I needed to create a topic-identifying model to tag and categorize the articles, I needed to generate document vector embeddings for effective searching and suggesting, and I had to generate condensed summaries of each article to be displayed to the user for fast and casual browsing.

First, the text was preprocessed with standard steps, which are detailed in the code walkthrough. The preprocessed text was used to generate document vectors using the GenSim Doc2Vec model, which was then used in conjunction with the preprocessed text to train a Latent-Dirichlet-Allocation model to identify topic clusters in the database and tag our documents. From these topic clusters and topic-ids, I then generated topic titles by stringing together the most influential words in the documents present in each topic. Finally, I imported a pretrained large BART transformer model from the huggingface transformers library. This was used to generate 1-2 sentence summaries, condensed from the raw abstract text.

After these steps, I had a full database of the 2,500 articles. It included preprocessed text, document vectors, transformer-generated summaries, and LDA-tagged topics.

Results

Once the database and article features were successfully created and organized, I created a simple interface to demonstrate how an NLP-based academic article browsing app could work. Using functions and processes detailed in the walkthrough notebook, I successfully put together a simple python program that allows a user to search for academic papers from an inputted text query and to browse through articles and be given suggestions based on what they choose to read.

While the program could be considered crude and lacking some possible features, the focus of this project was on the NLP feature engineering and modeling work that went into making the program's database, and not on designing a robust app. It does, however, successfully demonstrate the capabilities of an NLP-based browsing application, so I believe the prototype was effective.

Focusing on the outcomes of the actual natural-language-processing work I performed, I found mixed results in the performance of some of my models. The search engine and article suggestion feature (which both use the same function) produced decently similar results to the query or original article, but also often produced articles which were only marginally similar to the query. I was, however, satisfied to see that the program seemed to output a good amount of articles with the same topic-tag as the original (showing the LDA model groupings aligned with the vector similarities). I believe that the output of the search/suggestion function would be greatly improved if I had implemented the full 200,000 article set into the database, but I simply did not have the computing power or time for this.

I was also satisfied with the performance of the BART-based summarization model. While many of the summaries it generated were simply pared down versions of the given article abstracts, I found that it did an impressive job of integrating sentences together, preserving key words in the summary, and producing short digestible summaries that are actually coherent to read. When implemented in the browsing interface, I found it was very easy to get the general topic and main idea from these short summaries, which allowed me (acting as a user of this program) to easily browse through the articles and pick which ones I wanted to read more of. The summaries generally represented their overall article well, but many had strange cut-offs where the transformer seemed to reach its word limit and stop in the middle of its sentence. While this was an obvious shortcoming, I was impressed that the model was able to condense deep technical language into 1 or 2 sentences.

Conclusion

At the end of my project, I had a working prototype program to demonstrate the capabilities of an NLP-driven academic article browser. I did find that computational limits did diminish the quality of some of the models' outputs, but overall the program was usable and produced outputs that made sense. Given more data and time to train more complex models, I believe the program would be greatly improved and could be a promising product. Increasing the size of the feature-engineered database would allow for more accurate search/suggestion results, and more computing time would allow future iterations of this project to fine-tune the pretrained models on the dense technical language present in this dataset. I chose a dataset with incredibly complex and niche language that I noticed some of the models struggled to process, but the end result was a promising prototype!