Supplemental Report

More casual supplement to main final report

Data Mining

For data mining we used selenium to crawl through the pages and collect links then beautifulsoup and requests to go through the links and grab the pages. If you guys end up having to scrape it like we did, the script will take 17 hours if following Census web scraping guidelines but if you want to ignore them just change the wait time from 15 seconds to 0 seconds. There's no bot protection from what I can tell, but sometimes pages won't load in time so to fix that, set the "retry" value to >3 and most pages will be scraped properly.

Datastore

Because we used a vector based model for similarity scores, we have to use a vector store. The options we explored were Chroma, Faiss and PGvector. We went with Faiss because the files were the smallest and easiest to work with, a .npy file for the embeddings and a .bin file for the metadata. Chroma was similar but the metadata is stored in a .sqlite file. PGvector is an extension for postgreSQL which would be good if the data is already in a SQL database, but for our purposes it was too much overhead. For scalable real world use Faiss wouldn't be as reliable, and there is no way to easily add articles constantly. For actual implementation PGvector on a postgreSQL server would likely be the most realistic, and easiest to implement a pipeline of adding an article and it being automatically embedded.

NLP Methods

We compared four methods. The metrics are in the NLP section of results in the final report, but in summary Word2Vec and Doc2Vec didn't produce meaningful results and TF-IDF was consistent for exact wording but wasn't good for research or discovering pages not already discoverable by a keyword search. The SBERT model we used performed more than enough to meet out expectations so we didn't do testing for methods more computationally intensive, but some considerations if needed could be a SBERT model with more dimensions(784 as opposed to 384) or OpenAI embeddings.

For auditioning different NLP methods there's a Python notebook in the EDA section of the github that provides a framework to compare models on the results they output. We found that for comparing the results the only best way was to manually score them with Precision@k or a similar metric. We did see a paper from 2024 on using an LLM to automatically rate the relevance of a paper to the search result which seemed very promising, but that would cost some money to run.

We tried using the whole articles for similarity, but with the models we were using it just confused the results a lot more. It could be possible with a more comprehensive model or using something like openAI embeddings, but that would be more on the scale of LLMs than just semantic search.

Application

We tried to keep the application more usable than aesthetically pleasing. It runs a Python backend to handle operations when called by the frontend. The current implementation works well and all the filters seem to work. Current problematic points is the current logic for searching is find similar, rank then apply filters. Could be made more efficient if its apply filters then rank all that apply. Another issue is that as it stands when doing the more like this search, the top result is just the page the search was done off of. This can be fixed by adding a boolean that is passed through in the json request for redo, but we discovered this problem a little late and don't want to risk breaking the current code for a visual improvement.

Running the Code - Errors

There is a guide to set up the environment in the github under app, but these are some potential issues code was run on Python 3.12

- Errors installing faiss, pip can't find suitable version
  - Install faiss-cpu instead
- Issue with cached download of huggingface hub in imports
  - Use huggingface_hub version 0.25.0 or lower
- Errors with multithreading?
  - Briefly had errors when my compiler was on Python 3.9(Unintentionally, was set for a different project) where on embed or search the program would crash. Crash logs would say something about allocating memory. Was fixed with switching to Python 3.12