Deep Finders - Progress report

CS 410 Course Project | Theme: Intelligent Browsing

Team Members

Diego Carreno (lead)	diegoac3	diegoac3@illinois.edu
Jason Ho	jasonho4	jasonho4@illinois.edu
Chris Kabat	cjkabat2	cjkabat2@illinois.edu
Robbie Li	robbiel2	robbiel2@illinois.edu

Progress made thus far

The team has done progress in a few areas for the project:

- **Research**. We have analyzed several ways to parse HTML documents so that we can tokenize text content as text portions of different lengths (ie. paragraphs, individual sentences, group of sentences) in a robust manner for websites of multiple formats. This tokenization is needed to divide text into sub pieces to create a pseudo collection of documents from which we'll retrieve the most relevant pieces for a given user query. This research has pointed us to the use of Python-Goose as a way to reliably retrieve text data from websites of any format as long as the library can find article-looking text on the site.
  We have also done research to find best Javascript libraries to use to highlight passages as we show the user the most relevant results for their input query. This has pointed us to the use of mark.js as a highlighting library we are planning to use in the project.
- **Frontend: MVP of chrome extension.** We have created the first version of the Chrome Extension for the project. The chrome extension is capable of searching for passages of text from a user query, and can also pass raw HTML data to a backend parsing and ranking system that has been deployed in an Azure function. It can also perform initial highlighting tasks using mark.js for test passages.
- **Backend: System and API design.** We have designed the first high-level version of the system, including API schemas for the Chrome extension (FE) to communicate with the Azure function (BE), having the Chrome extension send raw HTML data to the parsing and ranking Azure function, and having the Azure function reply back to the Chrome extension with ranked text passages for a given user query. Query results are stored in an Azure Cosmos DB database to allow parameter tuning given feedback.
- **Backend: Parsing.** We have implemented an Azure function in Python that utilizes Python-Goose to parse HTML text information into text passages of different length (eg. sentences, paragraphs, and groups of sentences).
- **Backend: Ranking.** We have implemented an Azure function in Python that utilizes the rank-bm25 library to produce ranked passages of text for a given user query.

- **Backend: Parameter tuning.** We have implemented a parameter tuning script to vary k1 and b parameters given the limited feedback data we have.

Remaining tasks

- **Frontend: Highlight top ranked passages in websites.** We are able to highlight text passages now on websites, but we still need to highlight the top ranked passages that are returned from our Azure BM25 ranking function.
- **Frontend / Backend: Testing in sites of different formats.** We still need to systematically test our end to end solution across different websites to assess how robust it is and how well it behaves with different page formats.
- **Backend: Optimize BM25 parameters.** We need to further optimize current BM25 parameters using the parameter tuning script in our backend and through executing a feedback collection workflow we plan to run internally within our 4 team members.
- **[Stretch] Backend / Frontend: Capture users' feedback on results to fine tune parameters and/or develop a different model.** As a stretch goal we want to capture general feedback on results with every potential user to fine tune parameters. This might require modification of our parameter tuning script as well as implementing a mechanism in the UI to allow users to intuitively input their feedback to run Cranfield evaluations (or a similar method) to assess and fine tune results.

Any challenges/issues being faced

- **Websites of different formats.** Although more testing across websites of different formats is still needed, from initial tests we can see that being able to handle websites of different formats might pose a challenge since python goose might not always work well, in particular for websites that do not contain article-looking text information.
- **API connection.** We had some challenges in integrating the Frontend (Chrome extension) with the Backend (Azure parsing and ranking functions) as calls to the Azure function would return errors due to an API schema mismatch. Upon performing tests we have found the issues and managed to solve them and our Frontend and Backend can now connect to each other.