Team RSJB

CS 410 Progress Report

Team: William Skedd (wskedd2), Sewoong (Sam) Lee (samuel27), Ritik Kulkarni (rk30), and Jeremy Bao (jbao8).

1. Our Progress:

Thus far, we have mostly been performing research and experimentation, as this is the first time that many of us have encountered the technologies to be used. Jeremy Bao has done some reading about BeautifulSoup and figured out how to extract the text content from posts on StockTwits. He has also figured out how to extract the date from StockTwits posts. His exploratory code can be found in "src/Try_Beautiful_Soup.ipynb". Sam Lee has developed a simple text retrieval algorithm in the "src/search_algorithm" folder. He chose to use the rank-bm25 package instead of metapy, because it seemed like a better fit for our project. He has also investigated how different search algorithms and combinations of hyperparameters can be compared in order to determine which is the best. William Skedd has been learning about how Firebase and Firestore are used, in order to find the most optimal way to store, manage, retrieve, and update data. His current plan is to retrieve the posts scraped by Jeremy and store them into Firestore, with each post stored as a document. This will give Sam access to the corpus of documents he needs to start testing his search algorithms. William has also begun to develop models for storing documents and caching query results. Ritik Kulkarni has been learning about Flask, UI design, and best practices for web development. He is figuring out how to make the web application interact with our databases so that it can handle queries correctly, and is thinking about how we can create an easy to use, visually appealing user interface, which will likely be similar to those offered by existing search engines.

2. Remaining Tasks:

Jeremy Bao still needs to figure out how to store the obtained posts and implement the crawler program as a whole. Probably, each time it is run, it will obtain all posts up to the current date, not looking at any that have been previously crawled. Old posts do not have to be revisited, as StockTwits posts cannot be edited. The ID number of the last visited post will be recorded so that the crawler can start there the next time it is run. Also, we are still working on deciding how each post should be stored in our database. Currently, we think that storing their text content, date of posting, author, and ID number is the best approach. The text content is required for the search algorithm to work, and will be what is presented to the user, date of posting and author may be useful for filtering results, and post ID numbers are useful for providing our users with links to the retrieved posts. Our inverted index has to be stored as well. Regarding the search engine, we still need to create a test collection of example queries, documents, and relevance judgements, try out different combinations of parameters, and choose the one that yields the best results (perhaps F1-score or mAP). To handle users' queries, we plan to set up a cloud function (perhaps using AWS lambda). If a user inputs a previously unseen query, then the Sam's text retrieval algorithm will be run to find relevant documents, and those documents will be returned and cached. If a user enters a previously seen query, the relevant documents for that query which were stored in the cache will be returned, thus reducing runtime. A flag will be stored on Firestore to tell the cloud function when new posts have been scraped. If another run of the crawler has occurred, then all results cached before that run should be discarded. Finally, the web application must be created using Flask. Our database and cloud function must be implemented for this web application to work, so Ritik will first focus on designing a good UI layout capable of soliciting queries from the user and displaying relevant documents.

3. Issues and Challenges:

One challenge we are facing is that we are not aware of the best way to deal with very rare words, such as usernames. Each of these likely appears in only a handful of posts, and may or may not be relevant to queries, but there may be a massive number of them. Having so many rare words could inflate the size of our inverted index. A massive amount of distinct numbers also appear in our corpus of StockTwits posts, and may present similar problems. Perhaps these rare terms could simply be ignored, but some of our web application's users might want to search for posts mentioning a specific StockTwits user, and rare words are very useful for ranking documents. Additionally, extracting the date from posts is somewhat difficult. While Jeremy was able to extract the date from old posts, more recent posts (that were made a few days ago) have the string "now" as their date, even though we can see an actual date on StockTwits for any post made at least one day ago. Finally, figuring out effective ways in which to store all of our data and connect the different modules created by different group members may prove difficult, as our project consists of many components that are reliant on each other's correctness to function properly.