CS 410 Project Progress Report

Quote Finder

- Team Name | **Lunar-Tsai**
    - Manuel Suarez Lunar | manuel6@illinois.edu
    - Wei-Lun (Will) Tsai | wltsai2@illinois.edu --> team captain
- University of Illinois Urbana-Champaign | Fall 2023
- Code Repo | github.com/.../quote-finder

Summary

Our team has finalized the overall design for our Quote Finder application. It consists of four main backend components: (1) web crawler, (2) preprocessor, (3) sentiment analyzer, and (4) indexer and ranker. The web crawler collects raw quotes data from the Goodreads quotes website, which is then processed by the preprocessor to extract, normalize, and tokenize the quotes along with their corresponding metadata. The sentiment analyzer then assigns a sentiment to each quote and user input, which is used by the indexer and ranker to build an inverted index and rank the quotes based on their relevance to a given query. A web application (API) is built on top of these components, which is then fronted by a user interface for users to interact with the application. The overall architecture of the application is shown in Figure 1.

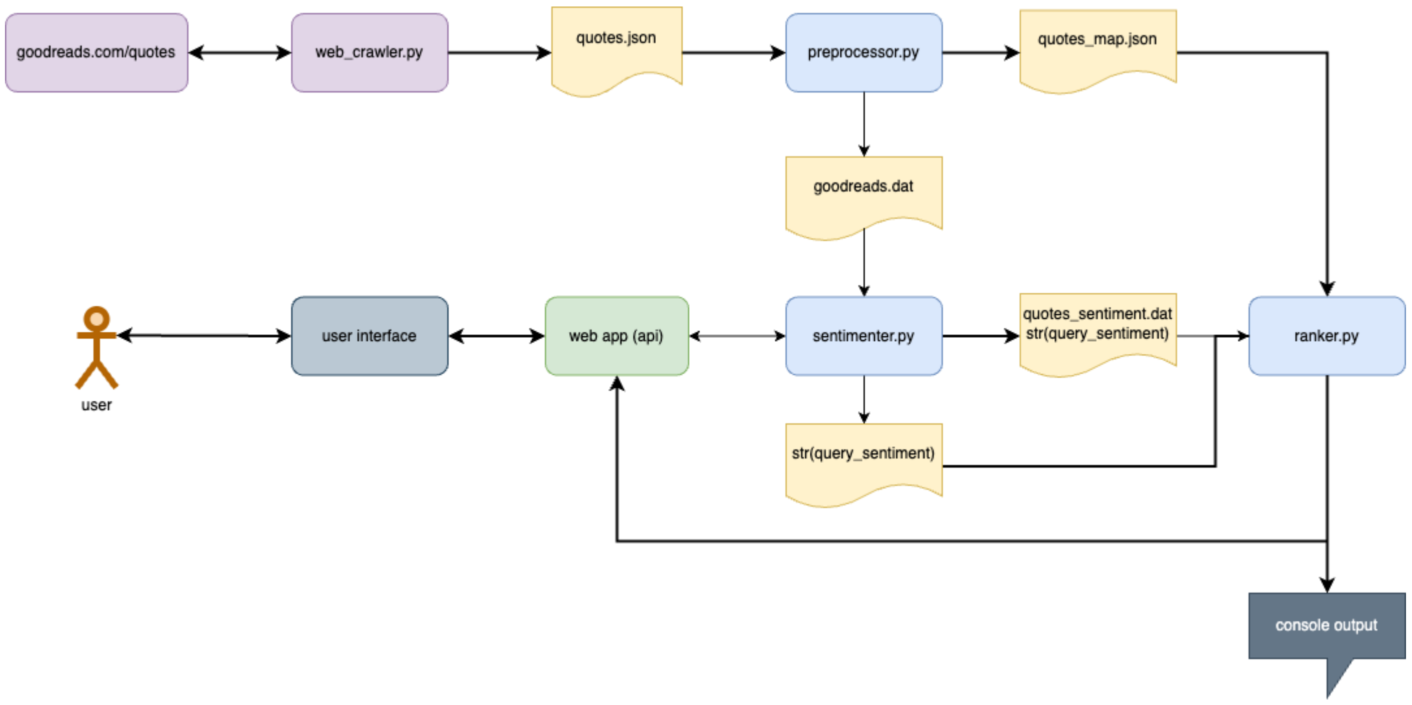

Figure 1: Overall architecture diagram for Quote Finder

Progress Update

So far, we have built a web crawler that uses the Scrapy python library to crawl Goodreads.com and extract all the quotes available in the website onto a JSON file, a tokenizer to create a bag of words representation of the quotes, an index to store our bag of words data and a ranker to provide a list of results in response to a user query.

Web crawler🔗

For our web crawler, we used the Scrapy library. We wrote a script that creates a Scrapy spider that will start at the Goodreads quotes page, extract quotes, the author's name, and tags for each quote, and then follow the "next" link to crawl subsequent pages. Lastly, it outputs the quotes to a JSON file that's picked up by the preprocessor script.

Preprocessor🔗

Implementation for the preprocessor has been completed, including functions for parsing, extracting, normalizing, and tokenizing the raw quotes data collected by the web crawler. It is implemented in Python, and the code can be found in the `preprocessor.py` file. The preprocessor takes a raw quotes data file (in json format) as input, and outputs two data files: (1) a quotes data file (in .dat format) containing the processed quotes data, and (2) a metadata file (in .json format) containing the metadata for each quote. The quotes data file is used by the indexer and ranker to build the inverted index, while the metadata file is used to display the metadata for each quote in the user interface. Two more tasks remain for the preprocessor: (1) tweaking the normalization and tokenization functions to improve the quality of the processed quotes data (e.g. removing punctuation) and (2) determining whether the preprocessor should be its own class object or a standolone process/script.

Indexer and Ranker🔗

Both the indexer and ranker have been implemented and can be found in the `ranker.py` file. The indexer builds an inverted index from the quotes data file output by the preprocessor, and the ranker uses the inverted index to rank the quotes based on their relevance to a given query. Currently, the ranker is a standalone process that takes a user query from the terminal and outputs the top 5 quotes that are most relevant to the query. The next steps are to (1) experiment with different ranking algorithms (e.g. BM25, Absolute Discount, etc.) to try improve the accuracy and (2) integrate the ranker with the web application (API) and user interface.

Upcoming Work🔗

In the following weeks we'll be focusing on creating the sentiment analyzer script and integrating its output into our index so that we can store sentiments tags to each quote. Then we'll work on creating the web application that will communicate between our python scripts and the web interface (website). Then we'll create the web interface that will be responsible for allowing the user to submit their desired input and then view a ranked list of quotes and authors based on their sentiment. Finally, we'll move our application to a public cloud provider (e.g., Azure) to host our application in a 24/7 and seamless manner.

Sentiment Analyzer🔗

We plan to leverage the Scikit-Learn Python library to implement the sentiment analyzer, which will be used to extract sentiment from the quotes as well as the user input. The idea is that our Quote

Finder application will take a descriptive query from the user (e.g. "sense of excitement and wonder"), extract the sentiment from the input, and then search the indexed sentiment of quotes. After implementing the sentiment analyzer, we will integrate it with the indexer and ranker to build the inverted index and rank the quotes based on their relevance to a given query.

Web Application (API)

The web application will be built using the Python library Flask, which will be used to create the API endpoints for the user interface to call in order to invoke the sentiment analyzer, indexer, and ranker. We plan to host the web application on a server so that it may service queries from the user interface.

User Interface

For the web interface, we’ll use standard HTML code to configure the visuals of the website and connect that to the our web app (Flask).