Assignment 1 Report

GitHub Link -*https://github.com/Neelkanth04/Vector-Space-Model_CSD358*

Team Members

Nilaansh Mathur (2310110537) **Manasvi Vedanta (2210110385)** **Shivam Doriya (2310110598)**

How to Run the Code

1. Make sure the ***corpus*** folder (with all the text files) is in the same folder as the Python files.
2. Open a terminal and install the required library:

```
pip install nltk
```

nltk library – python library for text pre processing (stopword removal and stemming)

3. Run the program with:

```
python main.py
```

The program will first run two test cases and then let you type your own search queries.

About the Code

We divided our program into five Python files, each one having its own job:

- preprocessing.py

- Cleans up the text before we use it.
- split_text_into_words breaks a sentence into a list of words.
- clean_my_words removes common stopwords (like a, the, is) and reduces words to their base form (e.g., running → run).

- soundex.py

- Handles the Soundex algorithm.
- get_soundex_code turns a word into a 4-character code based on how it sounds.
- Useful for matching names that may sound the same but have different spellings.

- indexer.py

- Reads all the .txt files inside the corpus.
- create_search_index builds the main index by storing which words appear in which document and at what positions.
- calculate_doc_lengths calculates the length of document vectors, needed for scoring.

- search.py

- Does the main searching.
- The function search handles both keyword searches and phrase searches (inside quotes).
- calculate_query_weights figures out how important each query term is.
- find_scores_for_docs calculates and ranks the final scores for documents.

- main.py

- The entry point of the project.
- start_program calls the indexer, runs the test cases, and then waits for user queries.

Novelty Feature – Phrase Search

The special feature we added is phrase search.

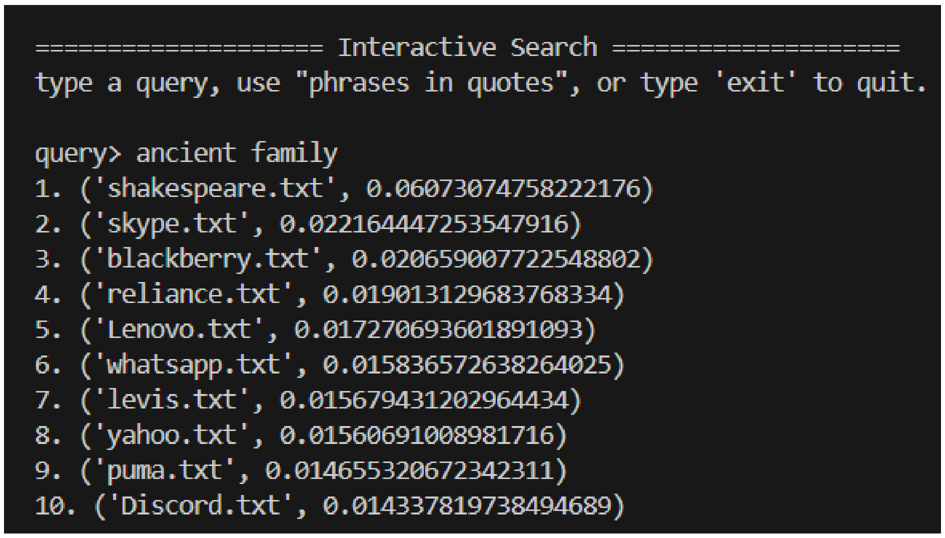
A standard search engine will just look for items that contain both phrases anywhere if you enter in "ancient family."
- In order to store the words and their precise locations, our search engine uses a positional index.
- In this manner, when you search for "ancient family," it only shows results where, in the proper order, family comes after ancient. As with Google, this improves the accuracy of the results.

EXAMPLE- After running the program (python main.py), the program will ask you to enter a search query.

➔ If you type a normal query, for example:

Ancient Family

It will return all documents that contain both ancient and family, even if they are not next to each other .

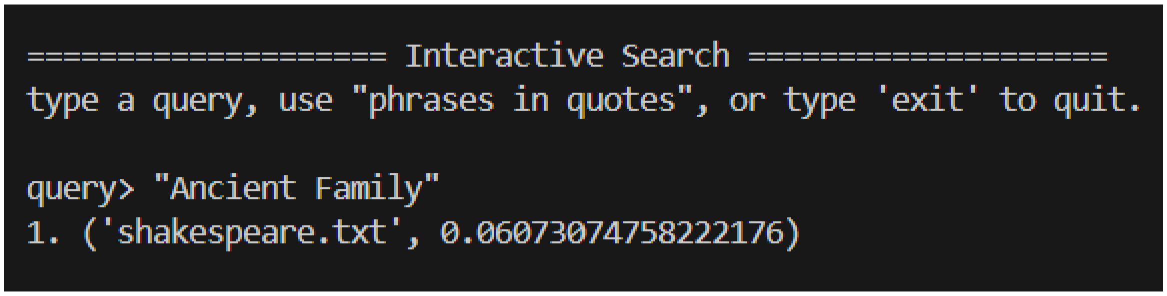


```
==================== Interactive Search ====================
type a query, use "phrases in quotes", or type 'exit' to quit.

query> ancient family
1. ('shakespeare.txt', 0.06073074758222176)
2. ('skype.txt', 0.022164447253547916)
3. ('blackberry.txt', 0.020659007722548802)
4. ('reliance.txt', 0.019013129683768334)
5. ('Lenovo.txt', 0.017270693601891093)
6. ('whatsapp.txt', 0.015836572638264025)
7. ('levis.txt', 0.015679431202964434)
8. ('yahoo.txt', 0.01560691008981716)
9. ('puma.txt', 0.014655320672342311)
10. ('Discord.txt', 0.014337819738494689)
```

➔ If you type the same query inside double quotes, "Ancient Family"

The program will only return documents where the two words appear next to each other in the correct order using positional index.

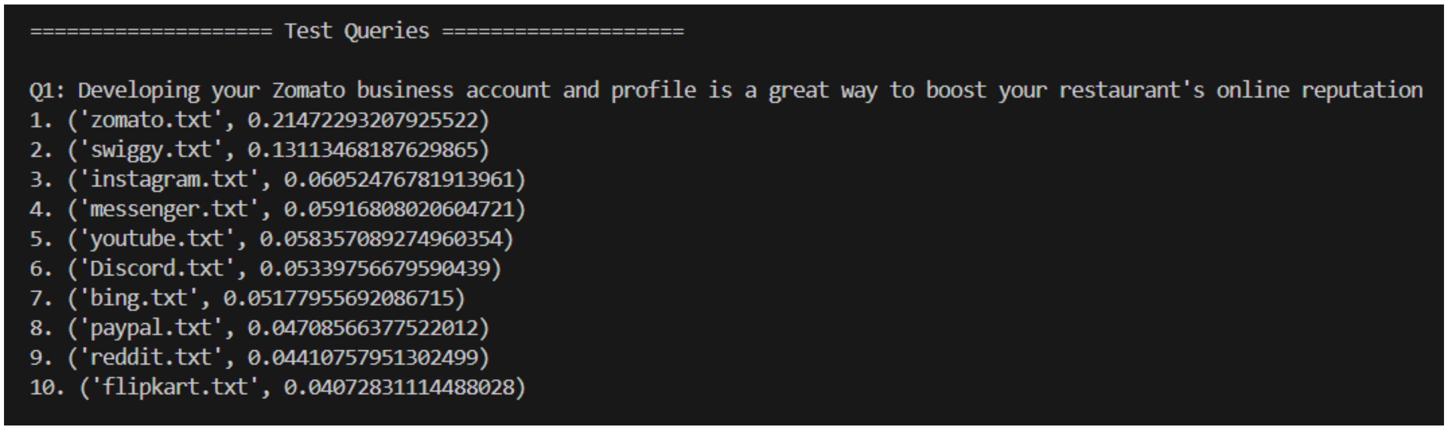


```
==================== Interactive Search ====================
type a query, use "phrases in quotes", or type 'exit' to quit.

query> "Ancient Family"
1. ('shakespeare.txt', 0.06073074758222176)
```

Output Screenshots

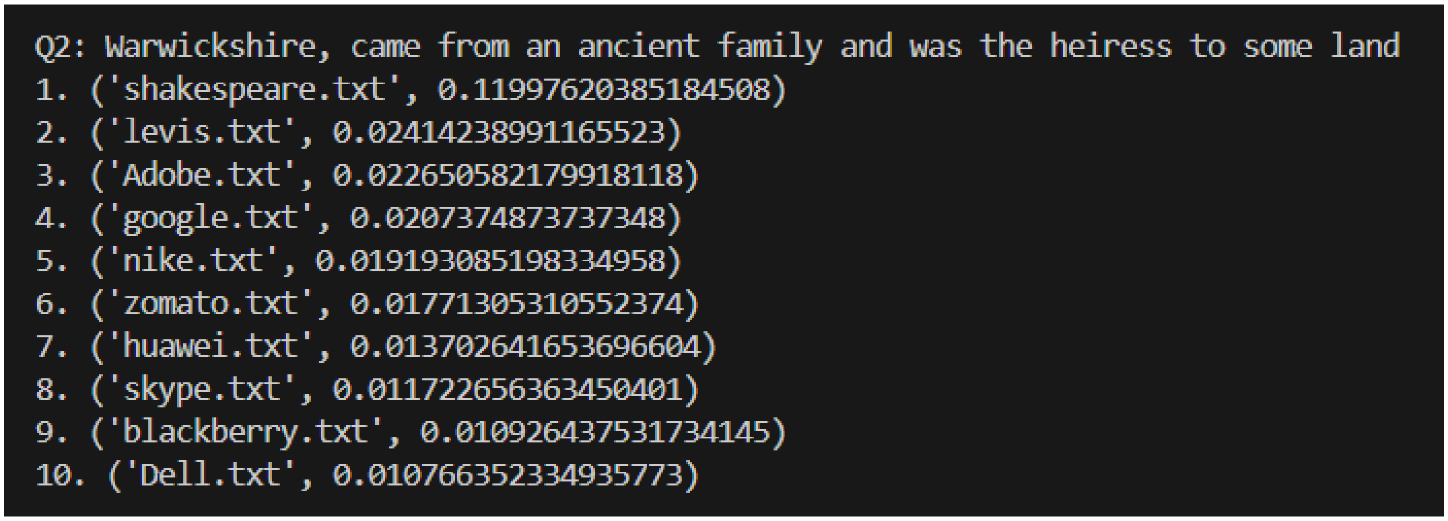
Test Case 1 Output:



```
==================== Test Queries ====================

Q1: Developing your Zomato business account and profile is a great way to boost your restaurant's online reputation
1. ('zomato.txt', 0.21472293207925522)
2. ('swiggy.txt', 0.13113468187629865)
3. ('instagram.txt', 0.06052476781913961)
4. ('messenger.txt', 0.05916808020604721)
5. ('youtube.txt', 0.058357089274960354)
6. ('Discord.txt', 0.05339756679590439)
7. ('bing.txt', 0.05177955692086715)
8. ('paypal.txt', 0.04708566377522012)
9. ('reddit.txt', 0.04410757951302499)
10. ('flipkart.txt', 0.04072831114488028)
```

Test Case 2 Output:



```
Q2: Warwickshire, came from an ancient family and was the heiress to some land
1. ('shakespeare.txt', 0.11997620385184508)
2. ('levis.txt', 0.02414238991165523)
3. ('Adobe.txt', 0.022650582179918118)
4. ('google.txt', 0.02073748737373348)
5. ('nike.txt', 0.019193085198334958)
6. ('zomato.txt', 0.01771305310552374)
7. ('huawei.txt', 0.013702641653696604)
8. ('skype.txt', 0.011722656363450401)
9. ('blackberry.txt', 0.010926437531734145)
10. ('Dell.txt', 0.010766352334935773)
```