IR PROJECT

Shikhar Arya | S20180010162 | UG-3

Introduction:

Quasar is a cosmic search engine. The main idea is to build a search engine specifically for the field of astronomy and astrophysics. The user can search from a wide range of available topics like Quarks to the Multiverse itself. Also various theories can also be searched for along with some popular fictional entities like a Wormhole. It has data crawled from Wikipedia and publicly available NASA & ISRO datasets and Kaggle. The final task is to retrieve all the documents relevant according to the query entered.

Approach:

The data is crawled using the python library Scrapy. After the large amount of data being collected, roughly 1 GB, I started working on the searching algorithm. I chose the vector space model, as it is quite easy to implement. And then the tdf-id is found for every vector and stored in the matrix. Finally the relevant documents are returned based on the descending order of similarity scores.

The whole project is divided into three parts:

1) **Web Crawling and gathering the data. (Task 1)**
2) **Creating the Tf-idf model for ranking. (Task 2)**
3) **Building the standalone django application.**

Crawling (Task1):

The data is crawled using the python library Scrapy . Scrapy in an intelligent crawler that has most of the implicit constraints. One can add explicit constraints if required. Here I have not added any. The other half of the data is simply collected from NASA's open source database and Kaggle. These datasets are in CSV format for easy extraction of data using python.

Preprocessing:

The crawled data is not in the required form. So the scraped documents go through series of preprocessing like:

1. Lower casing
2. Removing punctuations and newlines
3. Removing unicode characters

The preprocessed data is then used for creating the vector space model.

Ranking Algorithm (Task 2):

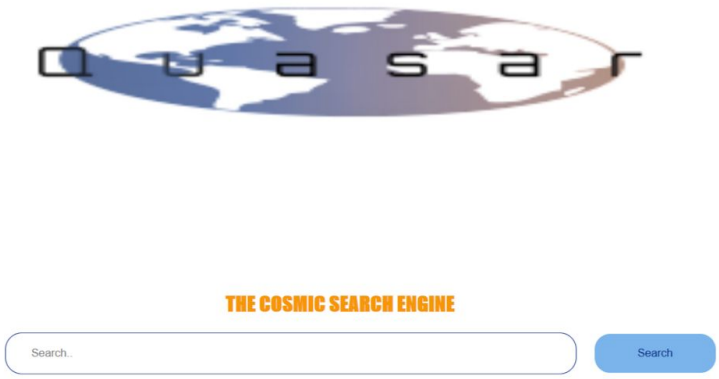
For the ranking algorithm, I have used the tf-idf model for ranking and the cosine similarity model for finding the similarity between the incident query vector and the feature vector model. The tf-idf model promotes the rare and relevant terms and suppresses the common terms. Therefore the common words will be taken care of automatically.Now the similarity scores are calculated based on the Tf-idf values from the loaded model. After the similarity scores are calculated, the scores are sorted and the top twenty documents are retrieved. **Time Complexity: O(mn); Space Complexity: O(M)** , where m: no.of documents, n:no.of terms in documents & M:size of Vocabulary.

Standalone Django Application:

The tf-idf model is saved and directly loaded into the django application. So that the retrieval takes minimal time. The search page takes

the query from the user then redirects to the result page which contains links to the specific documents.

The Search page:



The Result page:

Conclusion:

After finishing building up the project, I tested on all types of queries, one word, two words or even phrasal queries. The results are quite as expected. The documents which have the best similarity score are displayed . In the worst case, if there are no documents returned or there is a spelling mistake in the query, there will be simply void on the result page with the time taken for going through the whole database for search of that query, to check for the best and worst performance measure.