BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(October 25, 2020)

Design Document

Domain Specific Information Retrieval System

SUBMITTED BY

ANUSHA AGARWAL 2018A3PS0032H

JUI PRADHAN 2018B3A70984H

KRITI JETHLIA 2018A7PS0223H

Under the supervision of

Prof. Aruna Malapati

1. Problem Statement

The task is to build a search engine which will cater to the needs of a particular domain. You have to feed your IR model with documents containing information about the chosen domain. It will then process the data and build indexes. Once this is done, the user will give a query as an input. You are supposed to return top 10 relevant documents as the output.

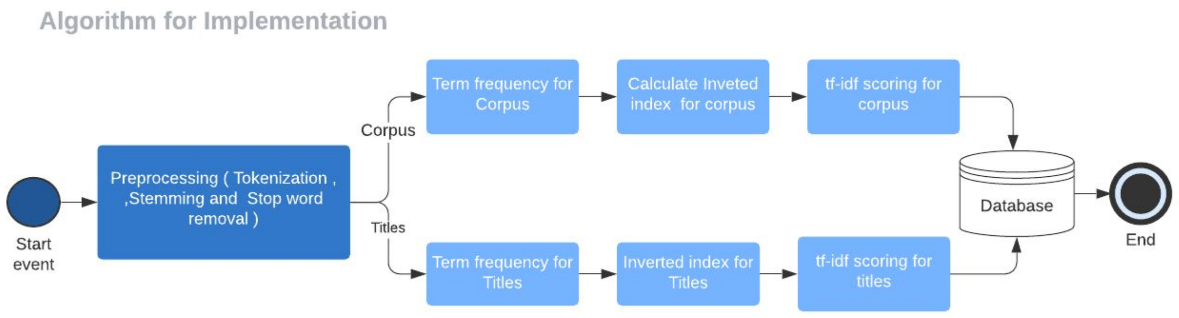
2. Dataset used

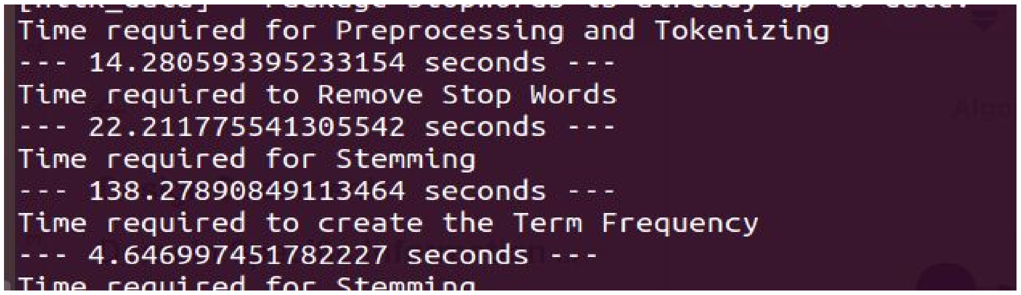
https://www.kaggle.com/jrobischon/wikipedia-movie-plots
It returns a list of 10 movie titles based on an input plot description, cast, director or a few keywords from the title.

3. Algorithm

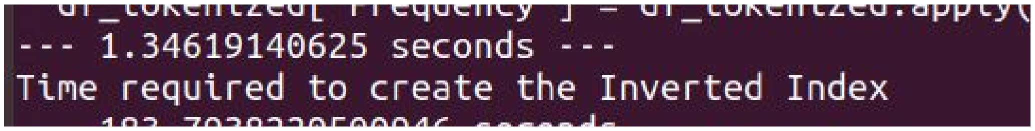
- The first step is **preprocessing** of the dataset. All the empty entries NA are replaced with a space and all the text is converted into lowercase letters. Punctuation and escape sequences are then removed and then text is tokenized using nltk’s tokenize function.
  Tokenization refers to splitting a sentence into its component words called Tokens.
  Stop words are removed from the tokens obtained from above. **Stop words** are the words with low significance as far as query search is concerned. These are words like is, are,the,etc. These are therefore filtered out from the search process.
  Next is stemming. **Stemming** is the process of reducing words to their root form. We have used Porter’s Algorithm for stemming.
  We have used **stemming instead of lemmatization** for normalization purposes because stemming is a faster algorithm. Lemmatization scans the whole corpus and is therefore a slower technique. Also obtaining a lemma requires defining a set of parts of speech. Thus lemmatization can be preferred when we need a model where language is important.
  Example -
  Caresses -> caress   bunnies -> bunni   connection -> connect
  The advantages of Porter’s Stemmer are:
  - It is easy to understand.
  - The recall is fast.

- We then create a Dictionary with Keys as words and Values as the word-frequency in the document. This is our **bag of words**. **Term frequency** tf$_{t,d}$ is the number of times term t is present in document d. We created a Vocabulary list for all the documents. i.e storing all the unique tokens.
  Then the **Inverted Index** is generated. For each term t, we store a list of documents that contain t. This is known as Postings list or the Inverted Index.


Algorithm for Implementation
Start event
Preprocessing (Tokenization , ,Stemming and Stop word removal)
Corpus
Titles
Term frequency for Corpus
Calculate Inveted index for corpus
tf-idf scoring for corpus
Term frequency for Titles
Inverted index for Titles
tf-idf scoring for titles
Database
End



```
Time required for Preprocessing and Tokenizing
--- 14.280593395233154 seconds ---
Time required to Remove Stop Words
--- 22.211775541305542 seconds ---
Time required for Stemming
--- 138.27890849113464 seconds ---
Time required to create the Term Frequency
--- 4.646997451782227 seconds ---
Time required for Stemming
```



```
--- 1.34619140625 seconds ---
Time required to create the Inverted Index
```

This all forms the part of preprocessing the corpus.

- Next step is **query processing**. We first preprocess the query in a similar manner by converting into lower case, tokenizing and stemming. Calculate tf-idf for query with documents and title as corpus.
  Document frequency df$_t$ is the number documents that content term t and an inverse measure of informativeness of term t.
  The i**nverse of doc frequency** can represent how rare a term is in all the documents.

  Inverse document frequency idf$_t$ = log (N/df$_t$)

  Where N is the number of documents in the corpus.

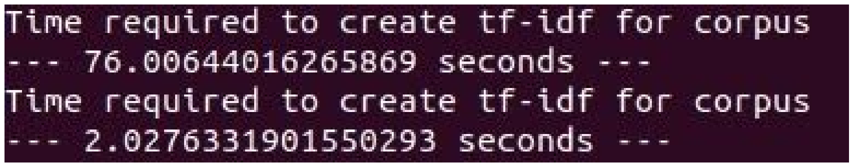
Tf-idf score is generally calculated as

$$\Sigma_{t\epsilon q \cap d} = tf.\ Idf_t$$

But, we have used a modified version of it called **Best Match 25**.
The formula for the same is :

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)},$$

Where k and b are constant parameters.
This technique is further discussed in the innovations part of our report.

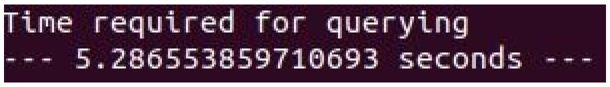


```
Time required to create tf-idf for corpus
--- 76.00644016265869 seconds ---
Time required to create tf-idf for corpus
--- 2.0276331901550293 seconds ---
```

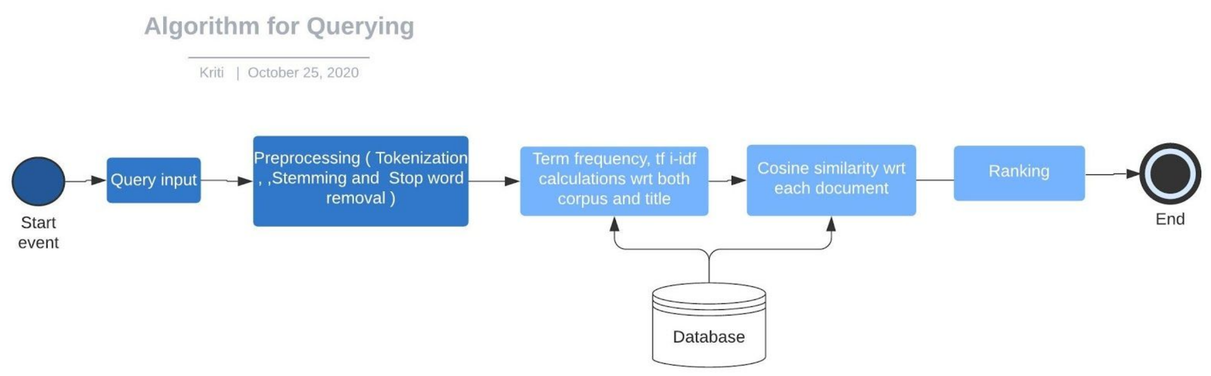
- We find the **cosine similarity** between the documents and query and rank the same using a heap. Cosine similarity helps us find out how similar a query and document are. The advantages are that it is independent of the size of the document i.e distance and measures proximity in terms of angle. Smaller is the angle, larger is the similarity. Each word represents a dimension when represented in a multi-dimensional space.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where A_i is the tf-idf weight of term i in query and B_i is the tf-idf weight of term i in document. We then push these cosine values into a **heap** for ranking. Larger is the value of cosine, larger is the proximity.



```
Time required for querying
--- 5.286553859710693 seconds ---
```


Algorithm for Querying
Kriti | October 25, 2020
Start event
Query input
Preprocessing (Tokenization , ,Stemming and Stop word removal)
Term frequency, tf i-idf calculations wrt both corpus and title
Cosine similarity wrt each document
Ranking
End
Database

4. Data Structures Used

- **Dictionary -** Dictionaries in python use keys for indexing which can be of any data types. The dictionary is stored in a set of curly brackets {} as a set of key-value pairs. We can store and access these values with the help of these keys. We have used a dictionary for storing the bag of words, tf-idf scores. They have helped in improving the query time by providing O(1) best case access time. Though dictionaries can be used to handle unstructured data and fast with small amounts of data, they are slow with big data and therefore we have used dataframes to store our dataset.

- **Dataframes -** A dataframe is a 2-D data structure with columns representing different features and values.All our preprocessing computations are carried out on a dataframe only. A data frame is a faster substitute to a dictionary if the data involved is huge. Also, accessing an element in a dataframe is an easy and quick process using .loc function.

- **Heaps-** Heaps are binary trees where every parent node is less than or equal to the children nodes. We push the elements into a heap and they reorder themselves into the specific heap property. We have used heap to rank our documents on the basis of values of cosine similarity. heapq.nlargest(n, iterable) returns a list of n largest elements from the dataset defined by iterable.

- **List-** List in Python is used to group data. It is highly versatile. A list can be indexed, concatenated, mutated, sliced and copied. We have used lists in intermediate steps while implementing the algorithm

5. Innovation

- **Best Match 25: an extension of regular TF-IDF**

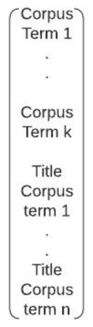
Tf-idf approximates the relative concentration of a given term in a set of documents. A certain term is more likely to be "concentrated" in the shorter document, therefore it is much more likely to be about the searched-for term and thus gets a higher score.
Best Match 25 (BM25) is based on the probabilistic retrieval framework

BM25 is an extension of the normal tf-idf score which combines idf-like ranking term frequency normalized by ratio of document length and average length and query term frequency. It takes a midway between full normalization of the document and not normalization at all by considering the tuning parameter b. Here k is a tuning parameter which is controlling scaling of term frequency.
Overall BM25 scores in such a manner that repetition of query words and more words in common with query increases the score. Repetition of terms in documents are given importance only if the length of document is longer than the average.

We tried both tf-idf and BM25 and BM 25 gave better results for queries.

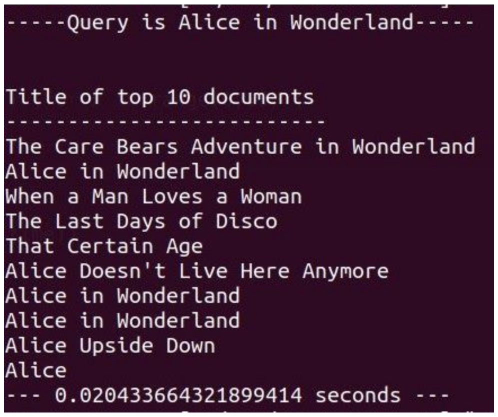
- **Title of Movie having more weight in Query**

Another innovation added is giving more weightage to the title of the movie. Since a user might just want to search for a particular movie title only. It is implemented by separately calculating the tf-idf scores for title and query and considering them as parameters of the document vector in the vector space model and checking the ranking accordingly. We checked queries with and without the consideration of titles as parameters and found the result of consideration of titles as parameters quite better.



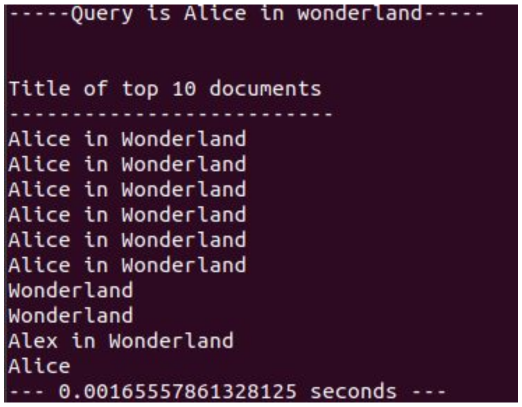
Vector for documents and query

Fig a) Without considering title as parameter b) consider title parameter



```
-----Query is Alice in Wonderland-----


Title of top 10 documents
-------------------------
The Care Bears Adventure in Wonderland
Alice in Wonderland
When a Man Loves a Woman
The Last Days of Disco
That Certain Age
Alice Doesn't Live Here Anymore
Alice in Wonderland
Alice in Wonderland
Alice Upside Down
Alice
--- 0.02043364321899414 seconds ---
```

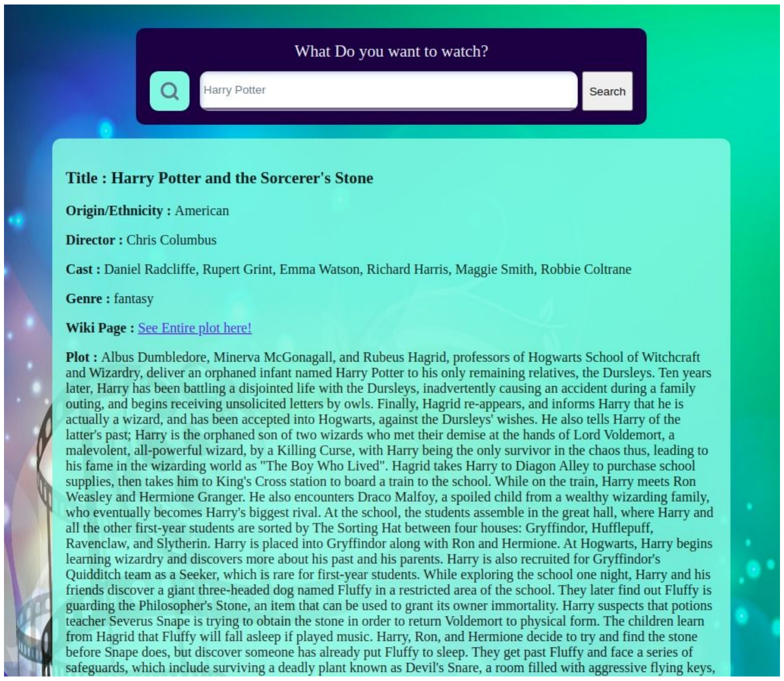


```
-----Query is Alice in wonderland-----


Title of top 10 documents
--------------------------
Alice in Wonderland
Alice in Wonderland
Alice in Wonderland
Alice in Wonderland
Alice in Wonderland
Alice in Wonderland
Wonderland
Wonderland
Alex in Wonderland
Alice
--- 0.00165557861328125 seconds ---
```

6. User Interface and Analysis

Overall the output of the query seems fine. The time taken for the querying is also low. We are able to get the appropriate documents for the required queries. Since we are working on very small document length ,BM25 works, however it won't work on large documents. Exact precision and recall can't be calculated since search results are subjective. But on an individual level, the search results can be justified as we have also used the title parameter. We have also tried to keep the user interface uncluttered which is also an important feature for real world search engines.



Title : Harry Potter and the Sorcerer's Stone

Origin/Ethnicity : American

Director : Chris Columbus

Cast : Daniel Radcliffe, Rupert Grint, Emma Watson, Richard Harris, Maggie Smith, Robbie Coltrane

Genre : fantasy

Wiki Page : See Entire plot here!

Plot : Albus Dumbledore, Minerva McGonagall, and Rubeus Hagrid, professors of Hogwarts School of Witchcraft and Wizardry, deliver an orphaned infant named Harry Potter to his only remaining relatives, the Dursleys. Ten years later, Harry has been battling a disjointed life with the Dursleys, inadvertently causing an accident during a family outing, and begins receiving unsolicited letters by owls. Finally, Hagrid re-appears, and informs Harry that he is actually a wizard, and has been accepted into Hogwarts, against the Dursleys' wishes. He also tells Harry of the latter's past; Harry is the orphaned son of two wizards who met their demise at the hands of Lord Voldemort, a malevolent, all-powerful wizard, by a Killing Curse, with Harry being the only survivor in the chaos thus, leading to his fame in the wizarding world as "The Boy Who Lived". Hagrid takes Harry to Diagon Alley to purchase school supplies, then takes him to King's Cross station to board a train to the school. While on the train, Harry meets Ron Weasley and Hermione Granger. He also encounters Draco Malfoy, a spoiled child from a wealthy wizarding family, who eventually becomes Harry's biggest rival. At the school, the students assemble in the great hall, where Harry and all the other first-year students are sorted by The Sorting Hat between four houses: Gryffindor, Hufflepuff, Ravenclaw, and Slytherin. Harry is placed into Gryffindor along with Ron and Hermione. At Hogwarts, Harry begins learning wizardry and discovers more about his past and his parents. Harry is also recruited for Gryffindor's Quidditch team as a Seeker, which is rare for first-year students. While exploring the school one night, Harry and his friends discover a giant three-headed dog named Fluffy in a restricted area of the school. They later find out Fluffy is guarding the Philosopher's Stone, an item that can be used to grant its owner immortality. Harry suspects that potions teacher Severus Snape is trying to obtain the stone in order to return Voldemort to physical form. The children learn from Hagrid that Fluffy will fall asleep if played music. Harry, Ron, and Hermione decide to try and find the stone before Snape does, but discover someone has already put Fluffy to sleep. They get past Fluffy and face a series of safeguards, which include surviving a deadly plant known as Devil's Snare, a room filled with aggressive flying keys,