CS 6301.004

PROJECT 2: SEARCH ENGINE

TEAM MEMBERS:

Prathyusha Kanmanth Reddy(pxk166230)

Teja Kiran Chunduri(txc163430)

Shree Charan Varma Enugandla(sxe170530)

Sneha Nandigala(sxn171430)

Srinivasa Rajesh Ayachitula(sxa172830)

1. INTRODUCTION:

SEARCH ENGINE ON MOVIE PLOT SUMMARIES:

In this project, we are building a search engine on movie plot summaries from the Carnegie Movie Summary Corpus (site: http://www.cs.cmu.edu/~ark/personas/.) This dataset consists of plot summaries of 42,306 movies extracted from the November 2, 2012 dump of English-language Wikipedia and stored in the file plot_summaries.txt. Each line contains the Wikipedia movie ID (which indexes into movie.metadata.tsv) followed by the summary. The movie names can be fetched from movie.metadata.tsv which has all the metadata for 81,741 movies, extracted from the November 4, 2012 dump of Freebase.

2. IMPLEMENTATION:

The program asks the user for a query term and then displays the top 10 movies details, followed by a prompt that asks user to press enter key to continue with another query or “q” key to quit. The program terminates when the “q” key is pressed.

2.1 PREPROCESSING:

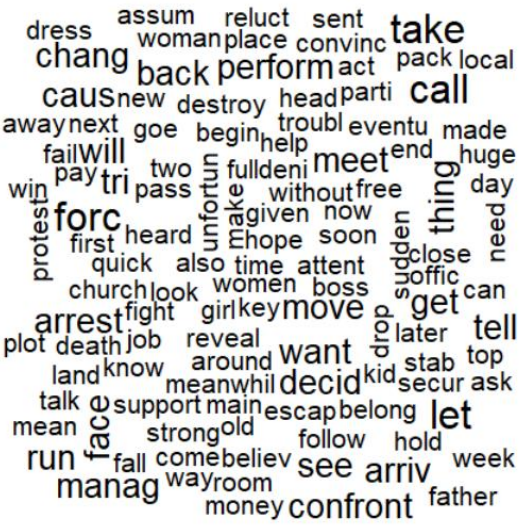
The first step in building the search engine is to preprocess the dataset. The following methods are used for preprocessing:

1. Each plot summary is converted to a Corpus object and is converted to UTF8 format to get rid of any non ascii characters.
2. tm library is used to perform the following text preprocessing:
    a. Replaced hyphens, colons and single quotes with space
    b. Removed punctuations
    c. Converted the entire corpus to lowercase letters
    d. Removed all numbers and stop words
    e. Stripped extra white spaces
    f. Stemmed the entire document
    g. Removed words with length less than or equal to 2.
3. Now the preprocessed plot summaries can be used to compute the TF-IDF values for each document-term pair. The query can be either a single term or multiple terms. If it’s a single term, the top 10 documents with the highest TF-IDF values are returned to the user. If the query consists of multiple terms, cosine similarity is computed between the query and each of the documents and the top 10 documents with highest scores are returned.

2.2 THE TF-IDF TECHNIQUE FOR SINGLE TERM QUERY:

Document similarity can be calculated by implementing the vector space model approach where the similarity between two documents is a function of the angle between their vectors in the term vector space. The following steps are required to implement it in R:

1. The first step here is to get the query and preprocess it using the same technique we've used for the plot summaries.
2. Create a corpus object of plot summaries and query and then the *term document matrix*, with rows corresponding to the words within documents and the columns corresponding to the document IDs. As with any matrix, we may think of a term document matrix as a collection of column vectors existing in a space defined by the rows. The query lives in this space as well.
3. Remove the sparse terms in the term document matrix and convert it to matrix data type to further analyze the data we're working with i.e. we can view the top 50 most frequent words and generate a word cloud to visualize it.



4. For both the document and query, we choose tf-idf weights of $(1+\log_2(tf))\times\log_2(N/df)$, which are defined to be 0 if tf=0. Note that whenever a term does not occur in a specific document, or when it appears in every document, its weight is zero.
5. Using *apply*, we run the tf-idf weighting function on every row of the term document matrix. The document frequency is easily derived from each row by the counting the non-zero entries (not including the query).
6. We normalize each column vector in our tfidf matrix so that its norm is one.
7. Split up the obtained matrix to query vector and document vectors. Performing matrix multiplication of these two matrices will result in the cosθ values for each

document vector and the query vector. These are simple dot products as our vectors have been normalized to unit length.

8. With scores in hand, merge the doc IDs with their respective movie names, rank the documents by their scores with the query vector and return the top 10 documents.

2.3 THE COSINE SIMILARITY TECHNIQUE FOR MULTIPLE TERM QUERY:

Cosine Similarity algorithm parses a document into a vector where each entry is the number of occurrences of a word in this document, then to compare the similarity of two documents we just need to calculate the cosine of the two vectors. The text2vec package provides various functions for measuring various distances/similarity in a unified way. The following steps are required to implement it in R:

1. The first step here is to get the query and preprocess it using the same technique we've used for the plot summaries.
2. We will compare documents in a vector space. So, we need to define common space and project documents to it. We will use vocabulary-based vectorization for better interpretability.
3. Create 2 DTMs, one for the documents and one for the query using the vectorizer created above.
4. Text2vec package has sim2(x, y, method) function which calculates similarity between each row of matrix x and each row of matrix y using given method. We use this on the above 2 DTMs with method = "cosine". The resulting matrix has all the scores we need.
5. With scores in hand, merge the doc IDs with their respective movie names, rank the documents by their scores with the query vector and return the top 10 documents.

2.4 ERROR HANDLING:

1. When the user enters a query with terms which do not exist in the documents, the similarity score is NaN for all the documents. This has been handled in the application by placing proper checks and displaying an error message to the user that says, "The query you entered does not match with any of the documents!".
2. When the user does not provide any query and just hits enter, the application throws an error saying "Please provide a valid search term"

2.5 RESULTS:

Case 1: Single term query

Query - Earth

Doc ID	Score	Movie Title
6177791	0.54	Invaders from Space
6173832	0.53	Attack from Space
6163787	0.48	Atomic Rulers of the World
5856565	0.44	Illegal Aliens
29198000	0.43	Chandra Mukhi
11020786	0.43	Lighter Than Hare
6042486	0.41	Sci-fighters
24627900	0.41	Lucinda's Spell
6172892	0.40	Evil Brain from Outer Space
34548840	0.40	Prey

Case 2: Multi term query

Query - Alien invasion

Doc ID	Score	Movie Title
26213151	0.32	Remote Control
24355525	0.31	Zone Troopers
25644434	0.29	Peacemaker
18079673	0.27	Alien Agent
4088706	0.26	Alienator
17377846	0.25	Alien invasion arizona
22877388	0.25	Gladiatress
30465048	0.25	Mathrubhoomi
12566479	0.24	Xtro 3: Watch the Skies

20752962 0.24 Invasion of the Star Creatures

Case 3: Query with no hits

Query - qwerty

The query you entered does not match with any of the documents!

Important note: This error also shows up when the query term has document frequency < total number of documents * (1 - 0.99) i.e. it has to be present in at least 42306*(1 - 0.99) = 423 documents. If it is less than that, it is removed during the removeSparseTerms function call and hence will not have any hits during the score calculation.

Case 4: Empty Query

Query -

Please provide a valid search term

3. LIMITATIONS, PROBLEMS FACED:

1. The dataset is too huge to work with on databricks, so we had to do it locally using plain R.
2. Initially, we had size restrictions when working with the term document matrix as it was too huge. We had to remove unnecessary elements as the matrix was too sparse by setting the sparse argument to 0.98. This resulted in giving no results for few words with low document frequency. So, we had to increase the sparse argument to 0.99 to include a few more words in the term document matrix but this caused the application to run a little slower due to size of the matrix. The execution speed can be improved if it could run on databricks.