Text based information retrieval system

CORPUS:-

Lyrics of songs scrapped from site http://www.mldb.org/

Procedure for scrapping:-

1. First we downloaded all the site map files

2. Then we used beautiful soup to parse all these xml files to get links containing song lyrics and stored links in file "links.txt"(code in getlinks.py)

3. Using beautiful-soup we parsed all links and stored songs in "english language" in folder "songs" with filename as title of song(code in getlyrics.py)

Data Preprocessing:-

-Tokenized the data, did data stemming and data lemmatizing and all necessary steps to ease process of retrieving results(code in" lemmatizng.py")

-stored the processed songs in folder "processed_songs"

Data structures and methodology used:-

- first We formed inverted index for every term in form of list(term-list)

- Every term in list had dictionary that maps document ids 1 or 0 depending on if document cntains term or not

- Similarly we have created list for storing every document

- For every document, there is a dictionary that maps terms with their term frequency

(above thing is implemented by function construct_tf_idf(tokens, doc_id) in tfidfdict.py)

- Then we have formed inverted index for every term in form of list(term-list)

- Every term in list had dictionary that maps document ids with its normalized idf

- Similarly we have created list for storing every document

- For every document, there is a dictionary that maps terms with their normalized tf

(above thing is implemented by function normalize_tf_idf(norm_file) in tfidfdict.py)

Query Preprocessing:-

- Then for the query taken as input we have processed it using function tokenize_query(input_query) that processes query by lemmatizing it, stemming it and removing stop words from

Getting search results:-

- Then we first vectorized query , i.e., we created term-dictionary for query that maps terms with their normalized tf (using function get_search_results() present in queryprocess.py)

- Then for every document in document-list structure, we calculated its cosine similarity(dot product) with query vectorized (using function cosine_similarity() present in queryprocess.py)

- Then we stored all document ids with their cosine similarity in max heap which returns us top 10 results