A *Domain Specific Information Retrieval System*

Design Document

16-September-2018

Index

1.Overview

2.Dataset

3.Programming language

4.Libraries used

5.Input Format

6.Data Structures

7.Features of the system

8.System Flow

9.The Front End

10.Programming Details

11.Execution Steps

12.Results

1. Overview:

The purpose of this project is to design and develop one's own text-based information retrieval system catering to a specific domain. In this case the domain chosen is searching for songs satisfying the criteria provided by the user. The system aims to provide different options to the user to convey their information need in order to provide them with relevant search results

2. Dataset

Obtained the dataset from Kaggle. The data set consists of 380000+ songs

Dataset url :-

https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics/home

3. Programming language: Python3.6.5

4. Libraries used:

- NLTK – Tokenization and stemming
- Sqlite3 – SQLite database support for indexing
- Pickle – Serialization and deserialization of python objects
- Tkinter – For GUI

5. Input Format:

The input to the program is a "**lyrics.csv**" file containing the following details of the songs in order

index, song, year, artist, genre, lyrics

6. Data structures used :

Python list and dictionaries -list are they are dynamic arrays with exponential over-allocation and python dictionaries are implemented as hash tables internally. Additionally, indexes are saved offline in SQLite db. to decrease query access time

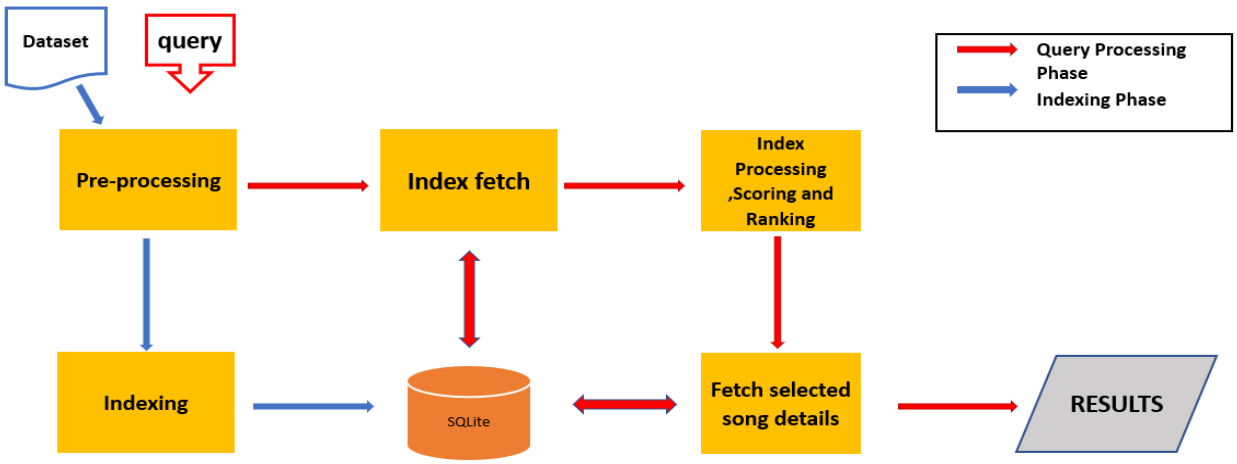
7. Features of the system

The major features of the system are –

- Provide an intuitive UI to the user to query and display the results
- Provides basic, lucky and advanced search features to cater to variety of information need of the user
- Processing the dataset and building different indexes for appropriate need. Supported indexes are – term index, positional index, permuterm index to support different types of user queries
- Tf-idf scoring is used for scoring and ranking the documents
- A relational database to save the computed indexes offline for faster and space efficient query processing.
- Appropriate query processing based on the search feature chosen

8. **System Flow:** There are two phase in the fucntioning of the system

    1. Indexing done priorly(represented by blue arrows in the figure below)
    2. Query the system(represented by red arrows below)



High Level System flow

8.1 Indexing Phase:

This phase involves the following steps

1. Pre-processing :
   - Tokenization – the process of extracting tokens and eliminating unwanted data from the file. NLTK's **word_tokenize()** is used for this purpose
   - Stemming – NLTK's implementation of porter stemmer is used to stem the parsed tokens. The token statistics (frequency, positions, … etc) are accounted.
2. Indexing :
   - The Back End – Database tables are created to update the dataset, term statistics and score based on the model, the indexes built during the processes. Following tables are created to hold the inverted indexes

     - songs(id, name ,year ,artist ,genre ,lyrics)

     - **terms**(term, cfreq ,dfreq)
       # collection and document frequency

     - **termdoc**(term ,docid , tfreq ,dscore ,posList)
       Dscore – tfidf score
       Poslist – pickled list of postions index
     - **permArtist**(key ,docid)
       Permuterm keys and their posting list (pickled)
     - **permName**(key,docid)
       Permuterm keys and their posting list (pickled)
     - **genreDoc**(genre ,docid)
       Parametric keys and their posting list (pickled)
     - **yearDoc**(year ,docid)
       Parametric keys and their posting list (pickled)

   - Index generation – Generated indexes are updated in the database. The following types of inverted indexes are created and updated for different types of content in the document
     - **Lyrics** : inverted term indexes are generated for each term in the lyrics and their statistics and tf-idf scores are accounted in the terms and termdoc tables

- **Song name and Artist name**: permuterm indexes are generated for these and saved in permartist and permname
- **Year and genre** – Parametric indexes are generated
- **Positional index -** In order to support the exact / approximate match of query positional indexes are generated and updated

8.2 <u>Query Processing Phase :</u>

Query processing involves – Pre-processing the query, Fetching the relevant indexes, processing the query indexes and fetching details of the selected results

1. Pre-processing – similar as done during indexing for the query terms
2. Fetching indexes – Based on the type of search chosen the relevant indexes are fetched from the database. The types of search are
   - Basic and lucky search: Here inverted index of the terms under consideration are fetched from the database file into python lists and dictionaries. Output is the Union of posting list of each term.
   - Advanced Search: Similar as before the respective indexes are loaded in memory in dictionaries and lists appropriately. If additional inputs are given then the appropriate corresponding indexes are fetched.

3. Processing indexes:

   The score (q, d) of query with respect to document is calculated and documents are scored in **document at a time** fashion. The results are ranked based on the scored and displayed in decreasing order of their score. Score of each query term in the doc is summed, tf-idf is calculated as follows:

   Score(q,d) = term frequency *log(N/document frequency)

   N = corpus size

In case of advanced search, If the query text is not given the score is calculated accordingly. Score of query with respect to document is computed only case of when query terms exist. Also, additional indexes are extracted based on the input provided and processed to filter the result. Scoring is a done by multiplying the zone weight of each serach criteria.Indexes based on the input shown in figure 4.

- If checkbox 1 is checked the posting lists are merging would be an intersection of documents containing two lists
- If checkbox 2 is checked the positional indexes are fetched and processed
- If input of type 3 is provided then the corresponding permuterm index is fetched
- If input type 5 is provided then the corresponding paramatric index is fetched

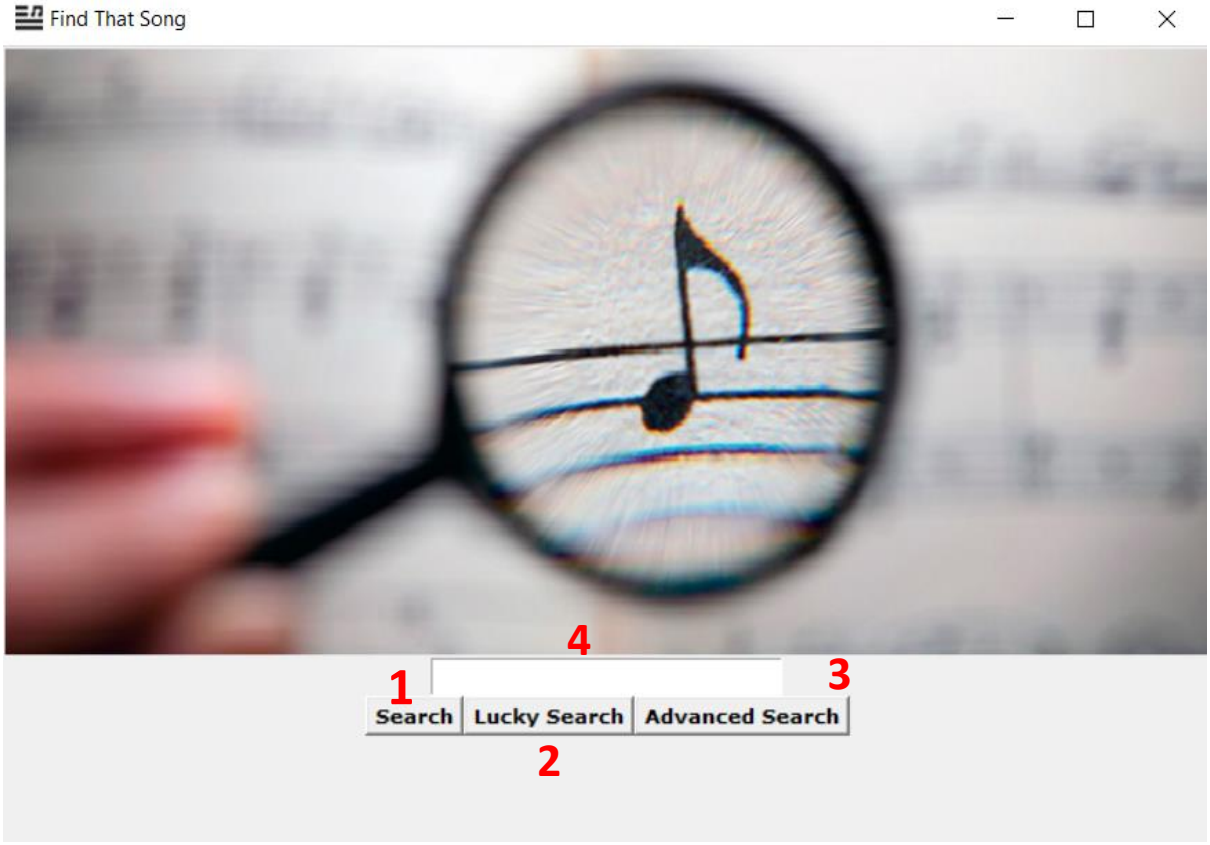
4. Fetching details of output: the details of the results are fetched from the song db.

9. <u>The Front End</u>

9.1 Features of Homepage: This page is shown to user upon start up.The Number on the screen are explained below

1. Search Button: On click leads to query results page
2. Lucky search: leads directly to song result page of the highest ranked document
3. Advanced Search: leads to the advanced search page to take additional inputs from user
4. Text box for query – should contain term from the song's lyrics, Results would be songs containing any of the terms

Figure1: Homepage



9.2 Features of Query Result Page:

This page is shown to user to the display results (Search/Advanced Search)

1. All search features of the homepage
2. The song names are hyperlinks -selecting it leads to it respective song details page
3. Navigates to the next or previous listing of the output
4. Tf-idf score, this is displayed for evaluation purposes only

Figure 2 : Query Result Page

9.3 Song lyrics page:

This page is reached from the lucky search or by clicking on the hyperlink in figure 2. Displays all details of the song

Figure 3 : Song lyrics page

9.4 Features of Advanced search page:

Selecting Advanced search opens up the page in figure 4

1. This check box helps use ensure that all the input query terms are present in the lyrics of the song
2. This check box helps user ensure exact or approximate match. The range of approximation is set in configuration and not taken from user
3. Song and Artist name provides support for wildcard queries the start and end of the query are matched with the permuterm index created apriori
4. Year and Genre are parametric indexes helping to future narrow down selection. Meaning of different combination of (from, to) values mean

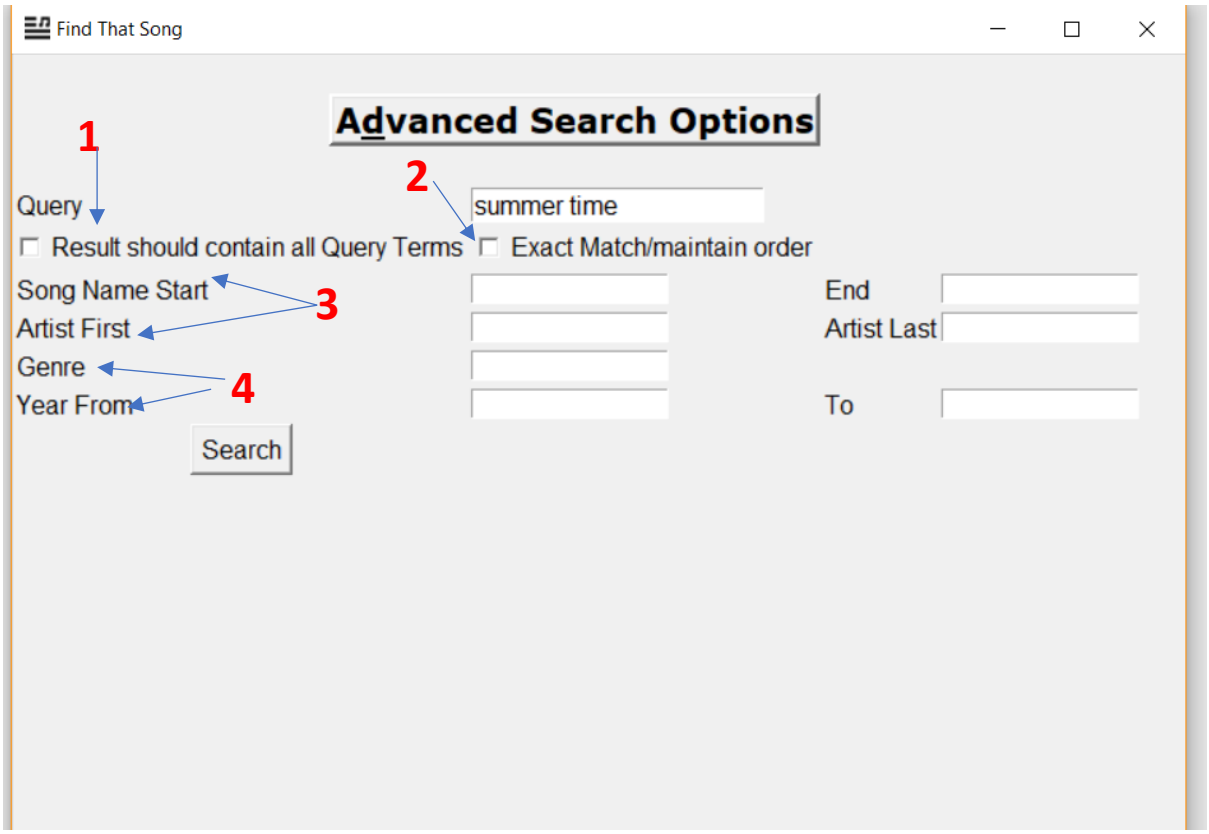
   (Year1==year2) -> select for that year

   (Year1, blank) -> include all entries after or in year1

   (blank, year2) -> include all entries before on in year2

   (blank, blank) -> Not processed

Figure 4 : Advance search window



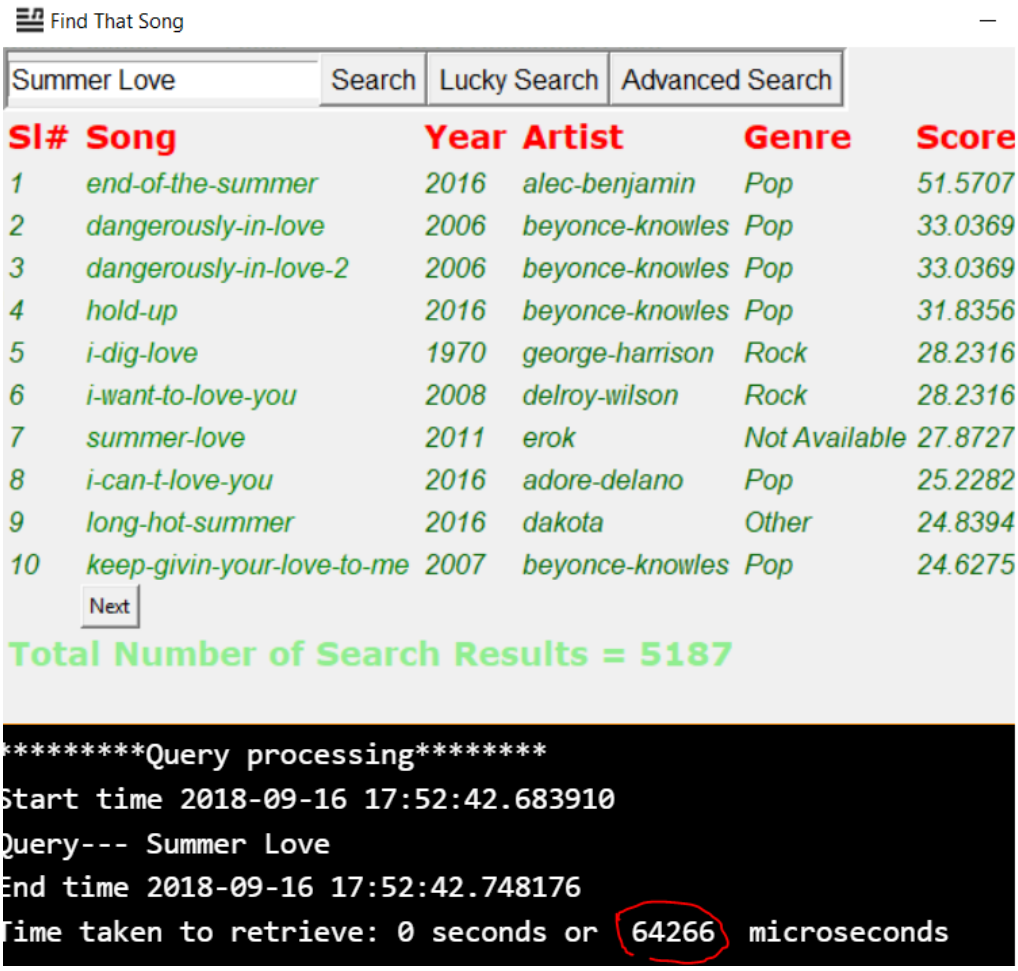
10. Programming Details:

Program names:

1. **Indexing.py** - Contains methods and functions for indexing logic
2. **SearchEngine.py** – starts up the application using class **searchEngine**, contains methods for UI, its handlers and communicates with Query Class methods to fetch the result
3. **query.py** – **Class Query** contains processing methods
4. **Preprocessing.py** – helper functions

The detailed documentation of the classes and methods can be seen in the program comments.

11. **Execution steps**: The Detailed execution steps are outlined in the README file
12. **Results**:
    1. The Execution time for indexing on a dataset consisting of 20000 songs is around 10 minutes. For the whole dataset it took about 4 hours
    2. The query processing for different types of queries on a 20k dataset is as seen in the example screenshots below. The time of query depends on the type or frequency of query terms in the corpus
        - **Basic Search – 64266 microseconds**



- **Advanced Search – 106718 microseconds**