GitHub Link

https://github.com/zezosarg/Soongle

Introduction

The goal of the project is a complete web application that provides an information retrieval system for searching information about songs or suggesting relevant songs to the user using semantic search. The main technologies used are the Lucene library, the Spring Boot framework and Eclipse’s DeepLearing4j for the word2vec model.

Corpus

The corpus is a subset of the Spotify million song dataset provided by Kaggle [1]. Using a python script, the data was loaded and some of the songs were randomly chosen. The extracted dataset uses a csv format as does the original. It consists of 10.000 random songs.

Text analyzer and index creation

For the analysis of text, the standard analyzer will be used. This will remove stop words and lowercase the tokens generated. The fields of the documents are: Artist, Title, Lyrics. We also keep the `songID` from the csv to use it later in our semantic search. Moreover, we keep one more field for the artist, `SortedDocValuesField`, that we can use later during grouping search to group the results by artist. The indexes to be created will be of type `FSDirectory` and will be persistently stored to disk. There are two types of indexes, one for regular search/grouping search and another for the realization of the semantic search. The first contains all the documents with the fields mentioned and it’s named `luceneIndex`. The second one, named `modelIndex`, contains the document ID which is indexable and a vector which will be the vector of the document created by the model.

Search

The system will support keyword-based search. Regular search performs search on any of the fields that are artist, title, or lyrics. Types of searching include regular, grouping and semantic search. Regular search accepts normal queries as well as queries that follow the standard Lucene syntax. For example, a normal query could be “Eminem no love” and a query which utilizes Lucene’s query syntax “artist:(Eminem) AND title:(no love)”. Meanwhile grouping search will search inside the lyrics field and group the results by artist. An important thing to notice is that for sorting purposes, each group is "represented" by the highest-sorted document according to the `groupSort` within it. This means that the first group is the one containing the highest scoring Document based on the field lyrics [2]. Therefore, the search will still produce the most relevant Documents to the user while grouping by artist.

Implementations of the abstract class `SoongleSearcher` are responsible for the different search methods. Searcher classes perform the actual search taking as input the search query and keeping an offset for the view more results button. An array of ranked objects is retrieved from TopDocs result. Then each result is highlighted and processed to a suitable format to be fed back to the application. Lastly a record of the queries will be preserved to suggest alternative results to the user as he types, similar to how most browsers autocomplete the user's input.

Result representation

Results will be ranked based on affinity with the query. They will be fetched, 10 at a time upon user request. Search keywords will be highlighted in results. A grouping feature upon the artist field is available as well as a semantic search.

Usage

Upon visiting the home page, the regular and word2vec indexes are built enabling all types of search. There is an option to rebuild the regular and model lucene indexes in case a problem occurs. The user can select between the three types of searches (regular, group by artist, semantic) via a dropdown list. Autocompletion is supported based on search history. Regular Search supports standard Lucene query syntax, the user can select the fields that the search will utilize by typing 'field:(query)' separated with AND/OR operators. For obvious reasons this form of query is not supported in semantic search, we will analyze this more in the next section of our report. Finally, when viewing results, the user can request additional results or return to the home page.

Semantic Search engine implementation using Word2Vec

We chose to use Eclipse Deeplearning4j which offers a suite of tools for running deep learning programs on the JVM. Dl4j (Deeplearning4j) will allow us to load a pretrained model in memory which offers us fast vector operations to compare word similarities. The reason we load the model in memory is because unlike our document index the amount of space it consumes will likely never increase in the future. In addition, most recent personal computers and servers can afford to spare a few gigabytes of memory as a tradeoff for more speed. The model was trained on all of English Wikipedia [3] and contains around 368,999 words and 300 dimensions. Ideally the vectors would be loaded in memory separately from the main application using detached threads. However, for the sake of simplicity this process happens every time we run the main tomcat server.

Using the word2vec model dl4j offers we built a simple semantic search into our application that takes in a string of text and outputs a list of documents which are similar to the text inputted by the user. The documents in the resulted list may contain a keyword from the user's query or have a similar context. Therefore, the search is not limited to matching the exact word tokens but can also output results for more abstract questions. To achieve this we generate once the vectors of all our documents so that we can compare their similarities with future queries and return the ones that have the closest contextual meaning. The vectors of the documents are generated by summing the vectors of all individual words inside their fields. However, unlike our model those vectors are stored in a brand new Lucene index that we can search using a document's unique identifier. We can now compare those vectors with a query and keep a list of indexes and scores of the documents that pass a certain threshold. After sorting the list we can take a slice from it and give it to another class on our service layer to search the main lucene index and retrieve all the fields of those documents to be outputted on the screen (10 at a time). Then when the user presses the view more results button, we take a new slice from that list and so on. For our implementation we found that generating the document vectors only using the artist and title field will give the model enough context to produce relevant results. Finally, even though we are not using any approximations for the cosine similarity or any multithreading (which could be easily implemented, at least for calculating the similarity of words) the results for 10000 documents which are way above the required 500, is near instant.

Using the semantic search, the user can for example search for a particular artist and get results for not only that artist but also artists who create similar music. The user can also search for a song title and get recommendations for similar titles from potentially different artists. This search could also be used to find songs in the same music genre however it may give priority to names or titles who include the genre's name in them. Fortunately this can be easily tweaked in the future so rather than create a vector for the query ex. "pop","rock" etc. to instead create a vector for the topN - 1 similar keywords and then compare that with the document's vectors.

Bibliography

[1]:

https://www.kaggle.com/datasets/notshrirang/spotify-million-song-dataset

[2]:

https://docs.atlassian.com/DAC/javadoc/lucene/3.2.0/reference/org/apache/lucene/search/grouping/package-summary.html

[3]:

https://github.com/alexandres/lexvec