Multimedia Search and Retrieval: Task 1 - Group A

TOBIAS HINUM, Johannes Kepler University, Austria
BENJAMIN LUCHICI, Johannes Kepler University, Austria
CORINA PÖTSCHER, Johannes Kepler University, Austria
MANUEL VUJAKOVIC, Johannes Kepler University, Austria

This document describes the findings of Group A for the first assignment of the course "Multimedia Search and Retrieval", which consisted of creating a reusable framework for text-based retrieval and similarity of music pieces.

1 INTRODUCTION

The goal of this work was to create a text-based retrieval system for the similarity of music pieces based on their lyrics using algorithms and approaches (i.e., cosine similarity computed on TF-IDF vectors and / or combination of song representation and similarity metric) discussed in the lecture. Our results were thereafter evaluated using genre as relevance criterion ("a song in the list of retrieved songs (results list) is relevant iff any of its genres matches any of the query song's genres"). The used performance metrics were the following:

- Precision
- MRR
- NDCG (all @10 and @100)

In order to be able to test our text-retrieval engine, a data set ("*Music4All-Onion[1]*") was provided to us, containing 76,115 tracks, labeled by an alphanumerical ID. Said data set was provided to us as .tsv files, which are explained in the following:

- **id_information_mmsr.tsv**: contains the metadata of the music pieces (the song's ID, its artist, the song title and the album it appeared on) for every entry,
- **id_genres_mmsr.tsv**: contains the song's ID and the genres of each entry. Each genre has ben preprocessed by lowercasing and removing special characters and double spaces,
- **id_lyrics_tf-idf_mmsr.tsv**: contains the song's ID and the relevance of lyrics in every entry in the form of *TF-IDF*,
- **id_lyrics_word2vec_mmsr.tsv**: contains the song's ID and the contextual relevance of lyrics in every entry in the form of *word2vec*,
- **id_bert_mmsr.tsv**: contains the song's ID and its values generated via *BERT*, which is a transformer-based machine learning technique for natural language processing.

Authors' addresses: Tobias Hinum, Johannes Kepler University, Linz, Austria, t.hinum@gmx.at; Benjamin Luchici, Johannes Kepler University, Linz, Austria, benjaminluchici@gmail.com; Corina Pötscher, Johannes Kepler University, Linz, Austria, corina03.poetscher@gmail.com; Manuel Vujakovic, Johannes Kepler University, Linz, Austria, manuelvujak@gmail.com.

2 APPROACH

In general, we chose to implement the whole project in Python 3.9. For that, the needed packages are:

- **csv**: was needed to be able to extract data from the .tsv files,
- **numpy**: was needed for the useful array manipulation as well as array handling,
- **ast**: was needed to have string lines from the .tsv file to be able to be transformed into numpy arrays,
- **sklearn.matrics.pariwise**: was needed for the cosine as well as the eucleadian_distances which we well discuss further later in the documentation.
- **pandas**: needed to read in the .tsv files as a data-frame.

The mentioned packages allowed us to start the work on the first task, which had a clear description on how to use cosine similarity for our similarity metric. Therefore, we retrieved the TF-IDF coefficient from the corresponding file and calculated the cosine similarity with the cosine_similarity function from the sklearn package, which uses the formula

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|}$$

As another similarity metric we chose to use the euclidean distance, which was applied on the BERT data. The BERT data includes a vector space model of word embeddings, which not only considers precise words, like TF-IDF, but also contextual information, which is highly useful when it comes to lyrics similarities. We then applied the euclidean distance to get a similarity value between the vector of two songs.

$$dist(x, y) = \sqrt{x \cdot x - 2 * x \cdot y + y \cdot y}$$

To combine it with our other similarity metric, we normalized this distance to get a value between 0 and 1 with 1 meaning that both songs have the same position in this high dimensional vector space. For the normalization, we simply calculated:

$$\frac{1}{1 + distance}$$

Both similarities are then weighted with 0.5 and summed up to provide us with an overall similarity score of our search engine. Our system then returns the 100 most similar songs ordered by similarity score. We chose this approach over a certain threshold to guarantee that the user gets a certain selection of songs that are the most similar.

3 EXPERIMENTAL SETUP

Starting off, we first thought it would be very memory intensive if we loaded all the song metadata (multiple .tsv files) needed in our approach into our memory. Therefore, we decided to go with a way that would not need as much memory space, which led to us doing it with line-wise reading of the files into our application, accepting the downside of longer run-time.

However, during the implementation process as we proceeded with the task, we found that this approach could not suffice in computing all the similarities as the run-time would exceed our amount of time resources available (similarity

calculation for one song, comparing it to the other 76,114 songs, took around five minutes).

Hence, we used panda to create a data frame containing all the data within the file(s). Although this approach takes up a lot more memory, it is by far magnitudes faster than our first implementation concept (100 similarity calculations for one song, comparing them to the other 76,114 songs, only took slightly more that a minute).

To offset the amount of times the memory is allocated, we also implemented all the functionalities to receive the data frame(s) as a parameter to be able to reuse them within the function calls, which means that each data frame has only been initialized once.

4 RESULTS/FINDINGS

Another part of the task was to compute Precision, MRR, and nDCG (all @10 and @100) for our search engine. By default, we always retrieved the top 100 (most similar) results were retrieved. However, this number is adjustable via parameter within the code.

After running 10,000 songs through our application, we found that the **precision** was 0,5169. This value shows that around 50 % of our retrieved songs meet our query criteria. The value of the **MRR** (Mean Reciprocal Rank) was 0,6152, showcasing that, in average, the first relevant entry was found at between the second and the third position in the result. For the **nDCG**[1] value, we chose the alphanumeric order of the IDs, as the normal **DCG** and our similarity as the **IDCG**. As the **ICDG** is impossible to be acquired, we decided to take our ranking order as the perfect approximation.

REFERENCES

[1] 2022. Music4All-Onion. https://doi.org/10.5281/zenodo.6609677

[1] https://en.wikipedia.org/wiki/Discounted_cumulative_gain last accessed 10.11.2022.