KV Multimedia Search and Retrieval

Exercise 1 Group E

|  |  |  |  |
| --- | --- | --- | --- |
| Ali Ayadi  Johannes Kepler University  K12042692@students.jku.at | Luca Della Mura  Johannes Kepler University  k12241884@students.jku.at | Ruo Li  Johannes Kepler University  k12247854@students.jku.at | Sara Scheucher  Johannes Kepler University  k1648069@students.jku.at |

ABSTRACT

This paper outlines the development of a rudimentary music retrieval system. The goal of this paper is to assess the impact of varied representations of data and similarity measures within a retrieval system. Subsequently the results retrieved by the developed systems will be evaluated through multiple metrics. The report explains the different representations used in text-based, audio-based, and video-based music retrieval systems, accompanied by an explanation of two different data fusion techniques. The implementation details are also briefly explained.

1. Introduction

The main goal of a music retrieval system is to find similar songs based on a specific query song provided by the user. Developing a retrieval system for music can be a challenging task due to the diverse nature of musical elements. Using text-based features as a representation, may lead to matches based on similar lyrics, the retrieved songs however can belong to a completely different genre. Moreover, there are also many songs that don´t have lyrics. If audio-based representations are used instead, the question arises as to which features are used to measure whether a song sounds similar to a specific query song, because the similarity of music is highly subjective.

This paper describes the approach to implementing various music-retrieval systems and subsequently evaluates the systems Within the scope of this paper 11 distinct Music retrieval systems will be implemented using different representations. The Dataset as well as the representations will be explained in chapter 2. The Similarity of Songs will be compared by calculating the cosine similarity of different text, audio-, and video-based features and furthermore a combination of some of those features. The retrieved results will be evaluated qualitatively according to the similarity to the queried song. The data set used for testing the 11 retrieval systems is a subset of the Music4All-Onion dataset which was kindly provided by the university.

1. Methodology

In this chapter the dataset as well as the different features we chose for the retrieval systems and the methods to calculate the similarities between the query songs and the retrieved songs will be explained.

* 1. The dataset

Music4All-Onion is a large-scale, multi-modal music data set, which expands the Music4All-dataset with additional features and meta-data. The dataset provides many audio, video and metadata features of 109,269 music pieces. The data was extracted from the platform last.fm [1]

For the implementation of the music retrieval systems, the university provided us with a subset containing 10k songs. The provided data is presented in various TSV-Files. One for each representation as well as one which contains the song name, artist, and album info, and one file which contains the ids and the corresponding genres of the song, which we will need for different evaluations of our retrieval systems. To read these files we use the Pandas library which interprets the files as data frames.

* 1. Text-based features

To compare songs based on the cosine-similarity of their lyrics we chose the following 3 representations which we used as input for our retrieval systems:

BERT is a language representation model. The name stands for Bidirectional Encoder Representations from Transformers.

trained on a collection of words that can capture rich contextual information in natural language. This allows BERT to understand the meanings of words in a given phrase. [2]

TF-IDF: this method is the mix of two components TF (Term Frequency) and IDF (Inverse Document Frequency), this combination yields a vector capturing the relevance of words in a document relative to a corpus. TF calculates the occurrence of a word for a given document on the other hand IDF calculates a word's importance for a given collection of documents the product of IDF by TF yields the relative importance of a word to a document within the overall corpus. [3]

Word2Vec is a popular technique to construct word embeddings in the field of Natural language processing. A word embedding is a vector representation of a word. There are two methods that can be used to create such an embedding with the Word2Vec technique. The first one is the CBOW Model (Common Bag of Words) which takes the context of every word as input to predict the word. The second method is the Skip Gram model which takes the target word and predicts the context of the word and generates the representation of the target word in the process. Both techniques use shallow neural networks. [4]

* 1. Audio-based features

To calculate the similarities between songs with audio-based features we chose the following four representations:

Blf-Correlation: This is

Mfcc:

Musicnn:

Ivec256:

* 1. Video-based features

We only implemented one retrieval system with video-based features. We decided to use the Vgg19 representation.

* 1. Cosine-Similarity

In our retrieval systems we used the cosine similarity to find the songs that are most similar to the query song.

In information retrieval the cosine similarity is a widely used metric especially in text-based retrieval. It calculates the similarity between two terms which are modeled as a vector. [5]

Ein Bild, das Schrift, Text, Handschrift, weiß enthält.

Automatisch generierte Beschreibung

* 1. Data Fusion

1. Implementation

For implementing the assignment, the programming language Python is used as it is most suitable for data analysis and data science purposes. The coding environment used is Jupyter Notebook as it supports the programming language Python. We also used the libraries NumPy and pandas as well as Scikit-learn which provide us with different similarity functions. The code repository is hosted on the platform GitHub. The coordination and integration of code contributions of each team member is therefore ensured using Git.

To ensure that new functionality as well as new algorithms can easily be added to the music retrieval system in the future, a large focus is set on making the code modular.

The input of the system is a string which contains the name and the artist of the query track. The retrieval system should output a list of k similar songs with the title and the artist. To be able to make better use of the results in further calculations this output is saved in a Python dictionary.

To keep the code modular and make it reusable we first implemented some basic functions in a separate Python file. First, we defined a function to get information about the artist and song name from the IDs. The function takes a list of ids as input and returns a list of tuples of strings containing the name and artist of tracks as the output. Then we implemented a function, which can be used for all text-based analysis. The function itself is called “text-based” and takes as input parameters “id”, the id of the query song, “repr” the string which represents the word embedding used in the data, for example, the tf-idf, “N”, the number of tracks retrieved as well as the similarity function for example Cosine similarity or Euclidean similarity. These measures help increase the flexibility and reusability of the code. Then we imported the functions into the main file where they can be called with different input parameters.

3.1 Random baseline

For the random baseline, we first shuffled the songs in a random order, so we get a different result each time the function is called. Then we excluded the query song from the data frame, so it does not appear in the result list. Afterwards, we retrieve the top N random songs and stored them in the result list.

3.2 Cos-sim based on tf-idf

To calculate the cosine-similarity of the tf-idf representation of the lyrics, we created a wrapper function called “cos\_sim” that takes two Numpy-arrays as input and reshapes them to 2d arrays so they can be used in the cosine similarity function which is provided by the Scikit-learn library. The result of the cos\_sim function is the similarity score of the two arrays. The similarity function cos\_sim is then passed to the “text\_based” function, as well as the query song-id and the dataset containing the tf-idf values. The first thing the test\_based function does is search for the query song in the tf-idf dataset and extract the row vector representing the queried song. Then we create an array which is called similarities to store the similarity scores.

Afterwards, the “text\_based” function iterates through all the rows in the tf-idf dataset. The similarities between the query-vector and track-vector are then calculated using the cos\_sim function. The song-id as well as the similarity-score are then saved in the similarities list. Afterwards, we sort the list in decreasing order of the similarity score and retrieve the ids of the 10 most similar songs.

2.3 Cos-sim based on word2vec

The next retrieval system uses the word2vec-embedding with the cos-sim-similarity measure. The flow of the code is the same as tf-idf. So only one parameter has to be changed when calling the function.

2.4 Cos-sim based on BERT

As the last retriever that we have implemented for this project, we have chosen cosine similarity as its similarity function and lyric representation generated by BERT. As mentioned above, we have adopted a modularized scheme when implementing the text-based function. Thus, for this step, we just set the repr parameter of the text\_base function to “bert”, and the sim\_func parameter to “cos-sim.” The flow of execution is the same as when using the other two representations.

1. Evaluation

To evaluate all the 11 music retrieval system we calculated different evaluation metrics that are described in detail in this chapter.

5.1 Precision & Recall

Recall and Precision are measures that show how well a retrieval system retrieves relevant information. The precision is the ratio of true positives and total retrieved results. The recall is the ratio of true positives and the actual number of positives. [6]

For the calculation of precision and recall we first obtained the genres of our retrieved results and put them into a list which consists of ids and genres of retrieved songs. This list is one of the parameters for the precision\_at\_k function. We also need k as a parameter as well as the id and genre of the query track. Afterwards we store the top k results into a variable and compare the genres of the top k results with the query genre and count how many of the retrieved results are relevant (a result is relevant if it has at least one common genre with the query track). Finally, we divide the relevant retrieved results by k.

For the calculation of the recall, we need one more parameter which is the whole genres dataset. The method calculates the number of retrieved relevant songs as well as the number of relevant songs in the whole genres dataset and then divides the relevant retrieved songs by all relevant songs.

**5.2 Genre diversity**

Diversity:

Diversity is the opposite of similarity, in music recommendation systems (RS), diverse genres are needed to give the user a better and broader recommendation outcome so he can choose flexibly according to his preferences. To ensure large diversity a wide range of genres have to be presented in the retrieved list by the RS. Different methods have been proposed by researchers to calculate the diversity such as calculating the distance between two elements i and j in the recommended list. Cosine similarity can be also used as a distance function to calculate the diversity. [7]

Diversity as an evaluation metric:

This method measures the diversity of genres of a top k retrieved tracks given a query. It calculates how the genres are evenly distributed over k retrieved tracks.

We can break down the formula into two parts:

Genre distribution:

We initialize a vector of zeros with a length equal to the number of all genres existing in the dataset, then for every genre found in each retrieved track, we add one to the corresponding genre position in the zeros vector divided by the number of genres of the retrieved track.

So, this could be considered as the normalized attribution of each genre within the retrieved track genres to the overall genres in the dataset.

Normalize of the distribution:

We divide the resulting vector by the number of the retrieved tracks. To get the genre diversity we calculate Shannon’s entropy of the resulting vector.

This is calculated by taking the negative sum of all the items of the resulting vector multiplied by its logarithm (base 2).

Formula in python:

For this formula, we defined a function called diversity which takes genres\_retrived, all\_genres and N as input parameters.

Genre\_retrived: it is a list of sets containing the genres of the retrieved tracks.

All\_genres: list of all unique genres in the whole dataset

N: is the number of retrieved tracks. The formula should return the genre diversity@k. We first define the zeros vector (zeros\_vec) with the length of all\_genres, and then we run through all the retrieved genres of every retrieved track.

Position: takes the index of the retrieved genre in the whole dataset's unique genres.

We calculate each retrieved genre contribution by dividing one by the length of the retrieved set of genres and we store it in a variable called g\_i\_contribution. Afterwards, we accumulate the attribution to the zeros\_vec in the giving position. After running through all the genres\_retrieved we divide the zeros\_vec by N and we assign the result to the variable: result\_vec. Next, we move on to the second part of the formula (Shannon’s Entropy) that returns the genre diversity@k. We initialize the diversity variable to zero.

We run through all the items in the result\_verc and if it is different than zero then multiply it by its logarithm (base 2) and the result should be accumulated into the diversity variable.

The function will then return the negative diversity which represents the genre diversity@k.

5.3 Genre coverage

This metric is similar to the Genre Diversity described in the last chapter a way to measure the quality of the audiobased-retrieval-system beyond using accuracy. For this research project the Genre Coverage is defined as the proportion of the number of unique genres present within at least one the top k retrieved tracks and the number of unique genres within the dataset itself.

5.4 nDCG

Normalized discounted cumulative gain (later referred in the current study as nDCG) evaluates the results based on graded relevance, i.e. nDCG assumes the users prefer the elements in the list of retrieved results to be presented in the descending order of their degree of relevance. Its calculation can be summarized in the following four steps:

1) Calculate the degree of relevance of each element (later referred in the current study as gain) in the list of retrieved results.

2) Assign the weight to the gain obtained from each element respectively according to its position in the list of retrieved results. Calculate the weighted sum of the gains. (later referred in the current study as DCG, i.e. Discounted Cumulative Gain)

3) Generate an ideal list of the retrieved results by reordering the elements in the list in the descending order of their gains. Calculated the DCG score for the ideal list. (later referred in the current study as iDCG, i.e. ideal Discounted Cumulative Gain)

4) Divide the DCG score of the list of the retrieved results by the iDCG score. [8]

The setting of the current study is as follows:

The function written to calculate the nDCG score for currently study only consider the top k elements in the list of retrieved results. When evaluation the results of this particular study, k is set to 10. (The metric is thus referred in the result section as nDCG@10).

The Sørensen–Dice coefficient of the genres is used to compute the gain, which adopts the following formula:



Gquery refers to the set of genres of the query track. Gi refers to the set of genres of the track used to calculate the gain. The genre information is obtained from the id\_genres\_mmsr.tsv dataset.

The weight to the gain was calculated with inverse logarithm of 2. The formulae adopted by the current study to calculate the nDCG are the following:

图示, 文本

描述已自动生成

6. Evaluation

we distinguish from the table of results for the query track 3: Every Christmas by Kelly Clarkson that a based approach using cosine similarity as a measure of similarity and Bert as data features have the highest precision and recall values this means that this method is the best recommender system among all the tested system while the diversity is the smallest one this means that the recommender succeeded to derive 10 tracks very similar to the query track which explains the high precision and recall and this limits the list of retrieved track genres as they are so close to the query track genres and therefore the genre diversity is small.

6.1 Accuracy

The first query Track which is “Love Me” by “The 1975” is assigned to the genres: pop, rock, indie pop, electro pop, indie rock, funk, and funk rock. We achieved a high precision of 90 % with all the audio-based representations except for the ivec256 where we only achieved a precision of 50 %. We achieved the highest recall of 0.125 % with the “MFCC Stats”, the “Blf Correlation” and the “musicnn” features. The recall is relatively low for all retrieval systems because the dataset consists of a large number of tracks, and we only retrieve 10 tracks with each system. Also, the genres pop and rock are very common genres so there might be many songs in the dataset assigned to these genres, therefore we have many relevant documents in the dataset which are not retrieved with our systems.

The second query Track “One” by “U2” is assigned to the genres rock, classic rock, pop, alternative rock, soft rock, easy listening and Irish rock. We achieved a precision of 80 % with the “blf-correlation”, “ivec256” and “musicnn” representations. With these three we also obtained the highest recall of 0.114 % With the mfcc stats we only get a precision of 60 %.

The third query Track “Every Christmas” by “Kelly Clarkson” is only assigned to one genre which is pop. With the audio-based retrieval system we achieved the highest precision of 60 % with the “blf-correlation” representation. With the text-based retrieval system using we were able to achieve a precision of 90 % using the Bert representation. The reason for this overall lower precision for query track 3 could be that it is only assigned to one genre, whereas the other two query tracks belong to several genres. With the audio-based retrieval systems we obtained the highest recall for query track 3 with the “blf-correlation” feature with 0.143 %.

In the precision-recall curves we plotted for all three query tracks we can see how the number of k affects the precision and recall. We note that, as the value of k increases, also the recall increases.

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Figure 1: Precision-Recall Curve Track1

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Figure 2: Precision-Recall Curve Track2

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Figure 3: Precision-Recall Curve Track3

6.3 Genre Coverage@10

The next result section was concerned with the genre coverage@10 score. As mentioned in the methodology section, genre coverage assesses the proportion of unique genres covered in the retrieved list. Therefore, a higher genre coverage@10 score indicates a more diverse retrieved list. As can be seen from table 1, genre coverage@10 score obtained with query track 1 spread out within the range between 0.02878 and 0.06205. The results from the four audio-based retrieval systems show two clusters. The one using ivec 256 feature embedding with a higher genre coverage@10 score of 0.3957 forms a cluster of its own. The rest three with genre coverage@10 scores around 0.3 forms another cluster. The results from the three text-based retrieval system also show two clusters. The two using BERT and word2vec feature embeddings achieved a higher genre coverage@10 score around 0.6, whereas the one using tfidf achieved a lower genre coverage@10 score of 0.02878. When comparing the results between those obtained from audio-based and text-based retrieval systems, genre coverage@10 scores of text-based retrieval systems are high, indicating for the results generated with query track 1, text-based retrieval systems tend to return lists that are more diverse in genre.

Moving on to the results obtained with query track 2, as shown in table 2, genre coverage@10 score obtained with query track 2 spread out within the range between 0.02518 and 0.04856. The results from the four audio-based retrieval systems show three clusters. The retrieval system using ivec256 feature embedding returns the highest genre coverage@10 score of 0.04856. The one using blf-correlation feature embedding returns the second highest genre coverage@10 score of 0.03957. The rest two return the genre coverage@10 scores around 0.03. The results from the three text-based retrieval systems show two clusters. The retrieval systems using tfidf and bert feature embeddings form one cluster with higher genre coverage@10 scores around 0.04. The one using word2vec feature embedding achieved a lower score of 0.02518. When comparing the results between those obtained from audio-based and text-based retrieval systems, the genre coverage@10 scores are distributed evenly, indicating for the results generated with query track 2, neither text-based retrieval systems nor audio-based retrieval system tends to return lists that are more diverse in genre.

Finally, let us examine the results obtained with query track 3. A closer inspection of table 3 reveals that genre coverage@10 score obtained with query track 3 spread out within the range between 0.03507 and 0.05845. The results from the four audio-based retrieval systems are evenly distributed around 0.05. The results from the three text-based retrieval systems show two clusters. The one using word2vec feature embedding achieved a slightly higher genre coverage@10 score than the other two at 0.04676. The other two achieved scores around 0.035. When comparing the results between those obtained from audio-based and text-based retrieval systems, genre coverage@10 scores of text-based retrieval systems are high, indicating for the results generated with query track 3, audio-based retrieval systems tend to return lists that are more diverse in genre overall than text-based retrieval systems.

6.4 nDCG@10

Now let us shift our focus to the results concerning the nDCG@10 score. As mentioned in the methodology section, a larger nDCG@10 score indicates a better performance of the retrieval system as the lists obtained from the results are more similar to their ideal counterparts when ranking is taken into account. Closer inspection of table 1 shows that all seven retrieval systems achieved better performance than the random base line with query track 1. Also, all seven retrieval systems managed to achieve an nDCG@10 score larger than 0.7. Among all four audio-based retrieval systems, the results show two clusters, with the two retrieval systems using musicnn and ivec 256 feature embeddings forming one cluster which achieved an nDCG@score larger than 0.9, and the rest two retrieval systems forming a slight underperformed cluster. Among all three text-based retrieval systems, the one using tf-idf feature embedding achieved the best performance. No apparent cluster can be observed from the results. When comparing the results between those obtained from audio-based and text-based retrieval systems, audio-based retrieval systems achieved a better performance regarding query track 1.

Let us then move on to discuss the results concerning the nDCG@10 scores obtained with query track 2. As shown in table 2, same as the results obtained with query track 1, all seven retrieval systems achieved an nDCG@10 score larger than 0.7 and better performance than the random baseline. Among all four audio-based retrieval systems, the results again show two cluster, with the retrieval system using ivec 256 feature embeddings in one cluster which achieved an nDCG@10 larger than 0.8 and the rest three in another cluster with an nDCG@10 larger than 0.7. The nDCG@10 from the text-based retrieval systems all exceed the threshold of 0.8 and exhibits a close interval from each other. When comparing the results between those obtained from audio-based and text-based retrieval systems, text-based retrieval systems achieved a better performance regarding query track 2.

Finally, turning now to the results concerning the nDCG@10 scores obtained with query track 3. As can be seen in table 3, the retrieval systems achieved overall a worse performance than with the other two query tracks with the exception of the text-based retrieval system using the tf-idf feature embedding. No apparent cluster can be observed from the results obtained from the audio-based retrieval systems. The nDCG@10 scores are evenly distributed around 0.6. As mentioned above, the results obtained from the text-based retrieval systems show two clusters. The retrieval system employing tf-idf feature embedding achieved an nDCG@10 score larger than 0.9. The rest two form another cluster with nDCG@10 scores around 0.7. When comparing the results between those obtained from audio-based and text-based retrieval systems, text-based retrieval systems achieved a better performance regarding query track 3.

Results (rounded to 5 digits)

Query Track 1: Love Me by The 1975

Table 1: Evaluation Results Track 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Precision@10 | Recall@10 | nDCG@10 | Coverage@10 | Diversity@10 |
| Audio-based(cosine, mfcc\_stats) | 0.9 | 0.00125 | 0.76718 | 0.02878 | 4.51273 |
| Audio-based(cosine,  Blf-correlation) | 0.9 | 0.00125 | 0.82250 | 0.02878 | 4.43499 |
| Audio-based(cosine, ivec 256) | 0.5 | 0.00070 | 0.93006 | 0.03957 | 4.69441 |
| Audio-based(cosine,  musicnn) | 0.9 | 0.00125 | 0.96787 | 0.03058 | 4.32781 |
| Text-based(cosine, tf-idf) | 0.7 | 0.00097 | 0.85336 | 0.02878 | 4.30428 |
| Text-based(cosine,  word2vec) | 0.9 | 0.00125 | 0.73128 | 0.05486 | 5.26926 |
| Text-based(cosine, Bert) | 0.9 | 0.00125 | 0.77874 | 0.06205 | 4.98983 |
| Random-Baseline | 0.6 | 0.00083 | 0.67518 | 0.04317 | 5.24167 |

Query Track 2: One by U2

Table 2: Evaluation Results Track 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Precision@10 | Recall@10 | nDCG@10 | Coverage@10 | Diversity@10 |
| Audio-based(cosine, mfcc\_stats) | 0.6 | 0.00085 | 0.75426 | 0.02968 | 4.46326 |
| Audio-based(cosine,  Blf-correlation) | 0.8 | 0.00114 | 0.70208 | 0.03957 | 4.94064 |
| Audio-based(cosine, ivec 256) | 0.8 | 0.00114 | 0.83261 | 0.04856 | 5.02604 |
| Audio-based(cosine,  musicnn) | 0.8 | 0.00114 | 0.74983 | 0.03237 | 4.48012 |
| Text-based(cosine, tf-idf) | 0.6 | 0.00085 | 0.85755 | 0.03956 | 4.98059 |
| Text-based(cosine,  word2vec) | 0.4 | 0.00057 | 0.84163 | 0.02518 | 4.41128 |
| Text-based(cosine,  Bert) | 0.8 | 0.00114 | 0.88648 | 0.03866 | 4.73847 |
| Random-Baseline | 0.5 | 0.00071 | 0.65161 | 0.03327 | 4.88552 |

Query Track 3 : Every Christmas by Kelly Clarkson

Table 3: Evaluation Results Track 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Precision@10 | Recall@10 | nDCG@10 | Coverage@10 | Diversity@10 |
| Audio-based(cosine, mfcc\_stats) | 0.5 | 0.00119 | 0.57058 | 0.05845 | 5.33913 |
| Audio-based(cosine,  Blf-correlation) | 0.6 | 0.00143 | 0.61340 | 0.04946 | 4.84336 |
| Audio-based(cosine, ivec 256) | 0.5 | 0.00119 | 0.58630 | 0.05486 | 5.37249 |
| Audio-based(cosine,  musicnn) | 0.4 | 0.00167 | 0.59099 | 0.04856 | 5.14491 |
| Text-based(cosine, tf-idf) | 0.6 | 0.00143 | 0.95267 | 0.03507 | 4.30369 |
| Text-based(cosine,  word2vec) | 0.7 | 0.00167 | 0.73712 | 0.04676 | 5.33174 |
| Text-based(cosine,  Bert) | 0.9 | 0.00214 | 0.76511 | 0.03776 | 4.24776 |
| Random-Baseline | 0.3 | 0.00071 | 0.49639 | 0.03417 | 4.52973 |

7. Tables

[Table 1: Evaluation Results Track 1 1](#_Toc153289103)

[Table 2: Evaluation Results Track 2 2](#_Toc153289104)

[Table 3: Evaluation Results Track 3 3](#_Toc153289105)

8. Figures

[Figure 1: Precision-Recall Curve Track1 7](#_Toc153289111)

[Figure 2: Precision-Recall Curve Track2 7](#_Toc153289112)

[Figure 3: Precision-Recall Curve Track3 7](#_Toc153289113)

9. REFERENCES

[1] Moscati, M., Parada-Cabaleiro, E., Deldjoo, Y., Zangerle, E., & Schedl, M., „Music4All-Onion“, doi: 10.5281/zenodo.6609677.

[2] J. Devlin, M.-W. Chang, K. Lee, und K. Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. arXiv, 24. Mai 2019. Zugegriffen: 16. Januar 2024. [Online]. Verfügbar unter: http://arxiv.org/abs/1810.04805

[3] V. Jayaswal, „Text Vectorization: Term Frequency — Inverse Document Frequency (TFIDF)“, Medium. Zugegriffen: 16. Januar 2024. [Online]. Verfügbar unter: https://towardsdatascience.com/text-vectorization-term-frequency-inverse-document-frequency-tfidf-5a3f9604da6d

[4] D. Karani, „Introduction to Word Embedding and Word2Vec“, Medium. Zugegriffen: 16. Januar 2024. [Online]. Verfügbar unter: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

[5] F. Rahutomo, T. Kitasuka, und M. Aritsugi, „Semantic cosine similarity“, in *The 7th international student conference on advanced science and technology ICAST*, 2012, S. 1.

[6] K. M. Ting, „Precision and Recall“, in *Encyclopedia of Machine Learning*, C. Sammut und G. I. Webb, Hrsg., Boston, MA: Springer US, 2010, S. 781–781. doi: 10.1007/978-0-387-30164-8\_652.

[7] M. Kuanr und P. Mohapatra, „Assessment Methods for Evaluation of Recommender Systems: A Survey“, *Found. Comput. Decis. Sci.*, Bd. 46, Nr. 4, S. 393–421, Dez. 2021, doi: 10.2478/fcds-2021-0023.

[8] S. Büttcher, C. L. A. Clarke, und G. V. Cormack, *Information retrieval: implementing and evaluating search engines*. Cambridge, Mass: MIT Press, 2010.