Podcast Search

Marcus Nilszén, Melvin Jakobsson, Shuang Luo

May 11, 2023

Abstract

Podcasts have become an important medium for information dissemination. Based on the podcast data provided by Spotify, this project build a episode-segment search system using Elasticsearch as the search engine. We design an interface for users to input keywords and select the length of the desired snippet. We assemble the small segments returned by Elasticsearch to meet the user's specified length. In addition to this feature, we also use the test data provided by the TREC competition to test the ranking performance of the BM25 algorithm. We experiment with query expansion using synonyms and compare the effects of using and not using synonym search. Finally, we discuss the issues and insights gained during the project.

1 Introduction

In recent years, podcasts have grown to become an increasingly popular medium for people to consume information, stories, and entertainment. With the growing number of podcasts available, it can be overwhelming for listeners to find the right content that aligns with their interests. For this, we implement a podcast search engine that allows users to easily search for podcast segments based on specific keywords.

We build a podcast search engine based on Elasticsearch. This search engine accepts inputs from the users. The inputs include a query, whether to use synonyms and a selectable N. N represents the length of segments the user want to retrieve and the unit is minute. We also try some methods to improve the relevance performance and evaluate them by computing Normalized Discounted Cumulative Gain(nDCG) of the dataset provided by TREC 2020 podcasts track task 1 [1].

2 Related Work

This project is inspired by the podcast track added to the 2020 edition of the Text REtrieval Conference, TREC [2]. The podcast track contained two tasks, of which the first is highly relevant to this project. The relevant task covered searches of 2-minute segments. Each podcast had already been transcribed before the competition. One competitor of the podcast retrieval task was a team from DCU-ADAPT[3]. The competitors experimented with various different techniques, including query expansion using synonyms from WordNet. In their results section, they find that the WordNet-based query expansion is especially useful for some queries. They give the example of a query including the term "career" being expanded to also include "job" and "assistance" giving improved nDCG.

This confirms what is presented in the book *Introduction to information retrieval*[4]. It mentions that a common way of increasing recall is by using a Thesaurus, like mentioned above. This will increase recall by finding documents using another term for the same concept, like in the example. The book mentions multiple ways such a system can be implemented, such as the option of giving the user the option to expand the query, or doing it automatically.

According to [5], a short paper referencing the TREC 2020, successful podcast search engines should index content and metadata separately. This is due to many of the podcasts in the sample corpus used both by this study and by the competitors in the podcast track of TREC 2020 having titles and descriptions not representing the content of the podcast. Additionally, the authors found that successful podcast search engines should be able to generate a summary, tailored to the query, summarising the contents. This in order to help users understand if podcast retrievals match their information needs.

3 Method

This project can be mainly divided into three parts. One part is the interactive interface, another part is data processing based on Elasticsearch, and the third part is receiving input from the interactive interface, processing it, passing it to Elasticsearch, processing the results returned by Elasticsearch, and returning them to the interactive interface for display.

The structure of the project is shown in Figure 1. It shows how different parts interact with each other.

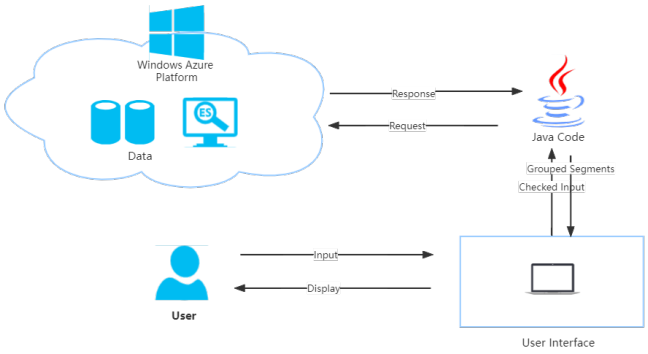


Figure 1: Project Structure for Podcast Search

3.1 Interaction

A graphical user interface (GUI) was constructed in Java with the help of the Swing library. The user may type in their desired query in the text field. Whenever the search button is pressed, the query is performed using the Elasticsearch REST High Level Client for the connection to the database. The names and timestamps of the highest ranking results are displayed in the GUI after the search is complete. Clicking on any of the search results opens a pop-up window displaying the transcript of the matching podcast segment.

The user can manually select the value of N, used for Grouping of segments. N is the length of segments the user desires to search in (in minutes), see the introduction. The

selection is done in the menu bar of the search window. Clicking on the button opens a pop-up in which the user can type in their preferred new value of N. The menu bar also contains a button to toggle whether query expansion using synonyms should be enabled or not.

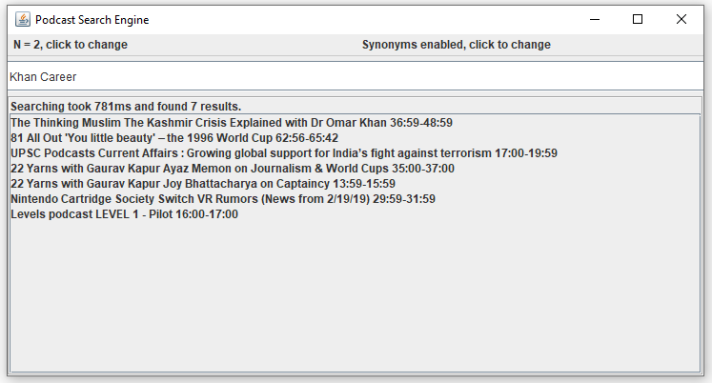


Figure 2: The user interface displaying a list of results and the time the search took.

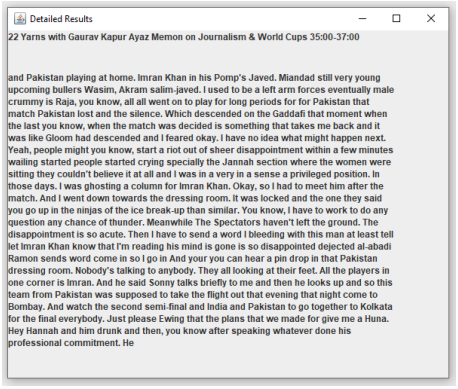


Figure 3: The user interface displaying the transcript of a podcast segment. This window is opened when the user clicks on a result in the results window, see figure 2.

3.2 Indexing

Our project is based on a large dataset of JSON files, each with a unique name, where the transcript has already been divided into sections. To store and manage this data, we decided to use Elasticsearch as our primary database and search engine.

In Elasticsearch, we utilize three different indices: episodes, episodes_2min, and metadata, each with a specific purpose. The term "document" in Elasticsearch refers to an entry in the database, not to be confused with files on a computer.

The episodes index consists of documents that represent each section of an episode. Since an episode has multiple sections, each episode is split into multiple documents. To differentiate between documents belonging to the same episode, we include the unique file name, excluding the .json file extension, followed by an underscore and an integer, e.g. 25ZTSteExFK0N9cNl2E4tb_0 and 25ZTSteExFK0N9cNl2E4tb_1. Each section includes a subset of the transcript and a list of objects containing the start time, end time, and word for each word in the transcript subset.

The second index, episodes_2min, consists of two-minute sections, with one minute overlap between each consecutive section. The length of documents can be written as [0s, 120s] and [60s, 180s] etc. Note however, if a podcast cannot be evenly divided into sections of multiples of 60s, the length will be similar to [0s, 119.5s]. This is very common among the documents and we simple treat those as full minutes rounded to the nearest multiple of 60, e.g. 119.5s would be treated as 120s. The documents in this index follow the same naming format as the episodes index. However, the naming scheme for the documents is slightly different from the evaluation data we use. In our index, we increment the integer following the underscore by one, while in the evaluation files, the integer is incremented by 60. We solve this issue by multiplying our integer by 60 during evaluation, but a better way would be to increment by 60 instead of one during indexing.

The metadata index contains documents that provide information about all the different podcast shows. This information includes podcast name, length, language, and many other attributes. We use this index in the UI/UX part of the application. The documents in this index are named after the unique file name, excluding the .json file extension, and the data used in this index comes from the file metadata.csv included in the data set.

3.3 Ranking and Grouping

We use the search function of Elasticsearch to perform keyword searching. Elasticsearch has already implemented some relevance scoring functions for sorting the retrieved text. We adopt the default model of Elasticsearch BM25 [6] for this purpose. BM25 is a classic text retrieval algorithm, and its formula includes several parameters. Some parameters need to be calculated through the document, such as inverse document frequency(IDF) and document length, while some parameters need to be manually set as hyperparameters. We use default values in Elasticsearch for these hyperparameters, which are $k1 = 1.2$ and $b = 0.75$.

In this project, we need to return the corresponding segment to the user based on the selected segment length. It is not practical to store segments of various lengths in Elasticsearch. Therefore, based on two indexes of different granularity, we group the returned small segments to the length required by the user. When the user selects 1 minute, we use the sentence-level index. When the user selects more than or equal to 2 minutes, we use the two-minute segment-level index. The grouping algorithm we implement is to center on the segment returned by Elasticsearch and to expand its context forwards and backward until the desired time length is reached. The expansion process would be a little different depending on the position of the hit segment as shown in Figure 4. As we extend the returned segments into segments of desired time length, we keep the score returned from Elasticsearch, so the ranking of segments would not change during the grouping process.

3.4 Synonyms

The search engines includes the option to expand search queries using a thesaurus. The data source of synonyms we used is WordNet [7]. Only synonyms are used, not other information from WordNet. The thesaurus consists of a long list of words, each listing some synonyms. When searching for a term and synonym expansion is enabled, an analyser in Elasticsearch is activated. It checks all the query terms for possible synonyms, and if a term has synonyms, they are added to the query. This should increase recall since documents with terms synonymous with the query term will also be retrieved[4].

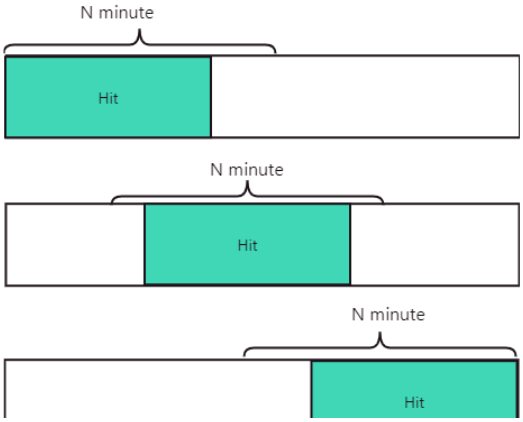


Figure 4: Grouping process

4 Experiments

We use the evaluation dataset provided by TREC 2020 podcasts track task 1 [1]. This dataset includes a training dataset and a testing dataset. We will use both of them to evaluate different methods. The dataset provides three different types of queries and descriptions, as well as a two-minute segments of each query after searching and scoring relevance. The relevance scores of two-minute segments to the query are judged manually by people on a graded scale of Perfect, Excellent, Good, Fair, Bad. There are a total of 58 queries, and we will use this data to calculate Normalized Discounted Cumulative Gain(nDCG) [8], $nDCG@10$ and $nDCG@20$.

Besides using the default BM25 algorithm, we also tried using synonym search to expand the query and retrieve more relevant documents. Here we use BM25+synonym to represent the method with synonym search. With a limit of maximum of 1000 returned segments, we have results listed in Table 1,2 and 3.

Table 1: nDCG results for two-minute episode segments

method	mean nDCG	best nDCG	worst nDCG
BM25	0.4882	0.9234	0.0000
BM25+synonym	0.2642	0.9234	0.0000

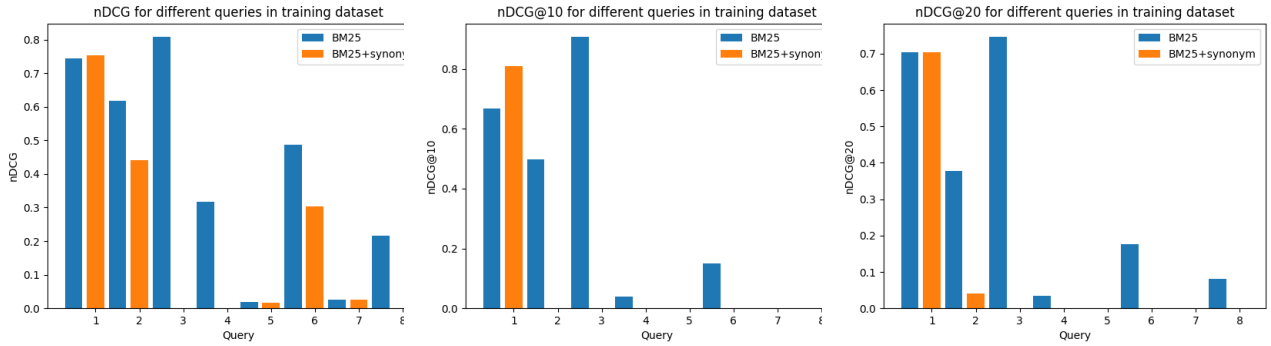
Table 2: $nDCG@10$ results for two-minute episode segments

method	mean $nDCG@10$	best $nDCG@10$	worst $nDCG@10$
BM25	0.3562	0.9066	0.0000
BM25+synonym	0.1541	0.8435	0.0000

Table 3: $nDCG@20$ results for two-minute episode segments

method	mean $nDCG@20$	best $nDCG@20$	worst $nDCG@20$
BM25	0.3496	0.8565	0.0000
BM25+synonym	0.1570	0.8305	0.0000

We also observed nDCG performance on specific query cases in the training dataset. As shown in Figure 5, we can see that adding synonyms could bring improvements in some cases, but it leads to worse results in most cases.We found that for more unfamiliar technical words, adding synonyms is good, while for more common words, adding synonyms makes the results worse.



(a) nDCG comparison on query cases in training dataset

(b) $nDCG@10$ comparison on query cases in training dataset

(c) $nDCG@20$ comparison on query cases in training dataset

Figure 5: nDCG comparison on query cases in training dataset

5 Discussion

5.1 Interaction

As this project is limited in scope to the search function, the user interface is lacking. Some main aspects which could be developed further is the results presentation to the user. According to Cartette et. al. [5], the results displayed should ideally contain a summary of the contents, especially the contents relating to the query in order to show the user what the match was based upon.

5.2 Indexing

It took over 40 hours to index the uncompressed files provided by Spotify [1]. We developed a Python script that processed the JSON data in all the .json files and then sent a PUT request to a local Elasticsearch instance. We had to index the data twice, once for the episodes index and again for the episodes_min index, which took a total of more than 80 hours. This doesn't even include the time we spent indexing some indices that we ultimately didn't use. There are probably some optimizations and improvements we could make to reduce the indexing time, such as printing less to the terminal (i.e. reduce IO operations) or using a faster programming language. However, we chose Python for its simplicity, and it allowed us to develop the script relatively quickly, compared to using a language like C++. Although we spent several days researching how to index large amounts of data into Elasticsearch, we did not find a better or more optimized approach.

There was also a smaller folder containing a test set of data within the Spotify data. We could have used this to experiment with different indexing structures and techniques before settling on the final structure. It would have saved us a lot of time to use it, but unfortunately we didn't do this. Nevertheless, given the project's relatively long

timeframe, the lengthy indexing time may not be a significant issue, especially considering the script's short development time.

5.3 Synonyms

The query expansion using WordNet synonyms is user-controlled by a toggle button. When evaluating the synonym query expansion, the mean nDCG scores were in all cases lower than the unmodified BM25 search. Our result is especially interesting since the competitors from DCU-ADAPT found a benefit from including synonym-based query expansion[3]. One possible reason is that the competitors used a more advanced version of synonym-based query expansion than was implemented in this project. Another possible reason is that the competitors always extracted nouns from the query description and including them in the query. Yet another possible reason is that the team from DCU-ADAPT used a different retrieval method, DPH instead of BM25.

One notable query which yielded a nDCG of 0.8076 without synonyms but 0.0 with was "Black hole image". The query terms are common words with several synonyms each. However, expanding the query will in this case include results not relevant to the information need since the query is not after dark holes or similar, only the astronomical kind.

One example to the contrary is the query "hvac industry environmentalism" which yielded a nDCG of 0.5820 without synonyms but 0.7003 with. The terms in the query are specific and the relevant documents contain technical terms. This seems to indicate that synonym based query expansion is beneficial when the query contains terms too specific for the information need and can be detrimental if the exact words of the query are vital to the need.

5.4 Grouping

We have conceived several methods of grouping, including combining segments of the same episode, so that each episode is displayed only once, as well as aggregating segments within a fixed time period. Because we believe that when users choose the length of the segment, they want to locate the segment with more accurate positions from longer episodes, we ultimately chose the method to expand the context around the segment hit by keywords as the results displayed for the users.

References

[1] A. Clifton, A. Pappu, S. Reddy, Y. Yu, J. Karlgren, B. Carterette, and R. Jones, "The spotify podcast dataset," *arXiv preprint arXiv:2004.04270*, 2020.

[2] R. Jones, B. Carterette, A. Clifton, M. Eskevich, G. J. Jones, J. Karlgren, A. Pappu, S. Reddy, and Y. Yu, "Trec 2020 podcasts track overview," *arXiv preprint arXiv:2103.15953*, 2021.

[3] G. J. F. J. Yasufumi Moriya, "Dcu-adapt at the trec 2020 podcasts track," *TREC*, 2020.

[4] C. D. Manning, *Introduction to information retrieval*. Cambridge: Cambridge University Press, 2008.

[5] B. C. Rosie Jones et. al, "Podcast metadata and content: Episode relevance and attractiveness in ad hoc search," *SIGIR*, 2021.

[6] S. Robertson, H. Zaragoza, *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[7] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[8] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.