

# Podcast Search

Damian Valle      Anton Chen  
damianvs@kth.se      antonche@kth.se

David Happel      Gilles Van De Vyer  
happel@kth.se      givdv@kth.se

## 1 Introduction

Wouldn't it be great to find the exact part of the podcast that discusses your topic of interest? The average podcast length is 43 minutes and 24 seconds. Spotify offers a total of 2.2 million podcasts in total [[Jov21](#)] [[Swe](#)]. Most podcasts talk about a large variety of topics in each episode. It can be hard to find what you are looking for without wasting a lot of time. Existing search engines offer the functionality to find relevant podcasts, based on the description. Imagine however, you are short on time and only want to listen to a fragment of the podcast where they talk about a particular interest or you want to listen to a specific part of your favourite podcast again. This project describes a search engine that finds fragments of podcasts similar to user specified queries to address this problem.

## 2 Related Work

The goal of the retrieval system described in [[MOG08](#)] is to find podcasts similar to a given podcasts. It uses a database of results from speech recognition on the podcasts, just like in this project. The idea is to base the similarity not on the full podcast, but on keywords extracted from each episode. The goal and usage of the system is fundamentally different from the goal in this project as the algorithm in this project finds specific, small fragments of podcasts similar to a user query.

[Fan] describes the technology behind the popular podcast search engine *listennotes*. The focus is on the practical aspect and combining the engine with a user interface. The actual information retrieval technology in this system is elasticsearch, just like in this project. The engine bases the results of the query only on the podcast description. This makes the engine a lot more lightweight, but doesn't use the full content information available. This project however, uses the full content information to find more specific results.

Automatic speech recognition enables traditional information retrieval methods to be applied to spoken material. While this opens up for many possibilities, it is common for automatic transcriptions to contain mistakes. This study [SP99] shows that a loss of effectiveness in the retrieval algorithms can be counteracted by utilizing document expansion. In the paper, the authors show that poor transcriptions tend to have a lower average precision due to words that are not recognized. Similarly, words that should not be there contributed to a further loss of precision, most noticeable in shorter queries.

This paper [Cli+20] describes the Spotify podcast dataset that we are using. It describes the structure of the data. The paper also provides a number of benchmarks relating to passage retrieval and summarization. Additionally, it presents limitations as a result of the noisy nature of the data. This data-set, and particularly its size, provides many possible areas of research, relating to linguistics research, and even sociology, since there are many different natural conversations, opinions, and topics present in the dataset.

## 3 Methodology

### 3.1 Search Engine

The search engine we built consists of several parts. The ElasticSearch API, Scripts for indexing and querying, A GUI application, and a connection to the Spotify API. An overview of this can be seen in Figure 1.

#### 3.1.1 Indexing

Firstly, there is the task of indexing the data set into ElasticSearch. This is done using a Python script. The script takes care of creating the ElasticSearch indexes, defining the data model, transforming the data, and inputting the data into the ElasticSearch indexes. The script defines the data model of the index managed by ElasticSearch, Figure 2 shows a UML diagram of this data model. The script then reads the data-set, and transforms the data as it appears in the data-set into the structure seen in Figure 2. The nested transcript data is stored in a separate index from the episode metadata. It also appears separately in the data-set.

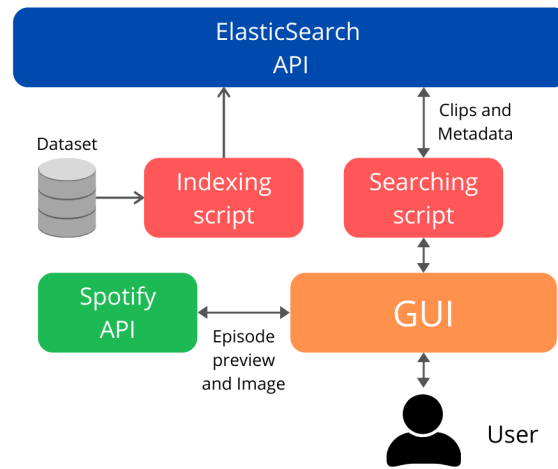


Figure 1: Structural overview of the search engine system.

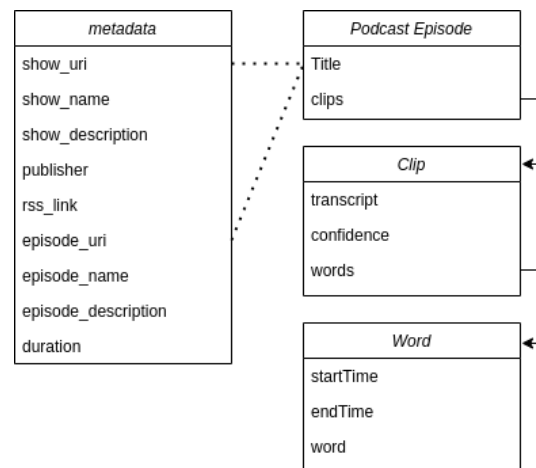


Figure 2: UML Diagram of data as it is stored in ElasticSearch.

### 3.1.2 Searching

Searching is done through another Python script. This script can either be run by itself, or called by the GUI, which will be described in the next section. The searching, or querying, script first connects to ElasticSearch, and builds up a JSON object representing the query. After receiving the results, for each returned episode, it constructs another simple query to retrieve the metadata of the returned podcast episodes. Then all the data is processed into an object that is easy to work with in the GUI.

Some of the options that can be given to this script is the returned number of results, the interval size, and the score combining function. By default the interval size is 1 segment of 30 seconds. When selecting higher intervals, it will also return the segments around this segment that has been found to be relevant. Adding segments behind, and before the relevant segment one-by-one.

In appendix A, the constructed query for a union search can be seen, this is the query used in the evaluation. We can see this is a nested query, where we find the episodes, by matching one or more of the transcripts within an episode. The number of results "size" can be dynamically set, and of course the query term. Additionally, the resulting score is multiplied by a second factor, the confidence of a transcript. As described before, this confidence value corresponds to the confidence of the speech-to-text AI when creating the transcription. Most transcripts are within the same narrow range of confidence. However, it means that "bad" transcriptions are less likely to score high or at least penalized slightly, which we believe is a useful feature in this case. In practice, as far as we have observed, this did not influence the ordering of the evaluation queries.

### 3.1.3 GUI

The GUI is also written in Python using the PySimpleGUI library. Figure 3 shows that the GUI has three columns:

- **Left column:** displays the search box, the list of ranked results and three dropdown menus:
  - Number of results to retrieve.
  - Type of ranking to use.
  - Number of 30 second segments around each result.
- **Middle column:** for a selected result, displays the podcast and episode names as well as their descriptions, the release date and episode length. For every matching clip it shows its transcription along with the start and end times.
- **Right column** displays the podcast cover image, provides an audio preview button for the podcast summary and a button that links to the podcast's Spotify URL.

The dataset did not include the audio preview and cover image so in order to display them we had to fetch them from the spotify database. We use the episode URI provided in the dataset to query the Spotify Web API by using the Spotipy library [Lam].

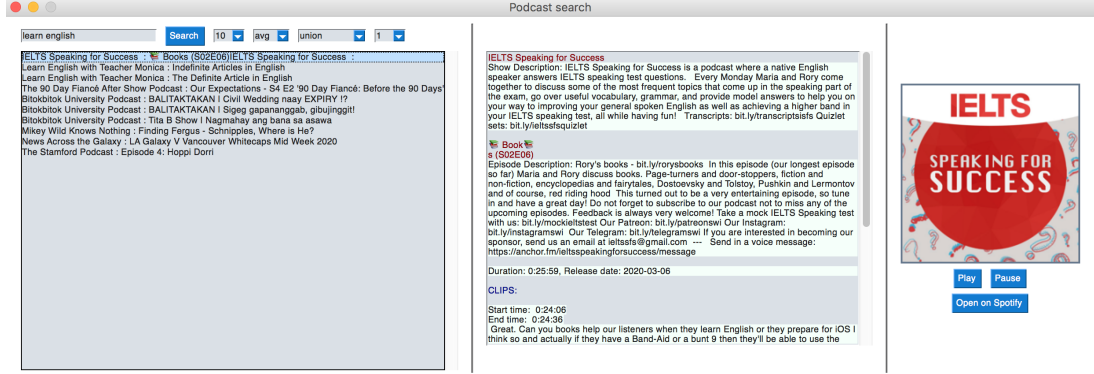


Figure 3: Graphical User Interface

### 3.2 Experiments and Evaluation

When defining the relevance of the clips returned by the search engine, we used the same assessment scale as the one used in TREC 2020 [Jon+21] as follows:

3. Excellent: segment is fully on topic, an ideal entry point for a listener, and the content is highly relevant.
2. Good: segment is mostly or almost fully on topic, a good entry point for a listener, and the content is either very relevant or somewhat relevant.
1. Fair: segment might not be fully on topic, is a poor entry point for a listener, and the content is somewhat relevant.
0. Bad: segment is not relevant.

Each member of the group was asked to evaluate the same search results from a set of predefined queries. The queries are split into two groups: the first is the same set of queries as the ones used in the dataset paper [Cli+20], while the second one is a set defined by us just to test how the search engine performs on topics we thought would be popular and also some more rare ones. The evaluation was performed with the search engine parameters that were set by default (average score and 10 results retrieved).

The main measures used to evaluate the search engine are the *normalized discounted cumulative gain* ( $nDCG$ ) at  $k$  and *precision at  $k$* . The former takes into account the rankings of the results and penalizes the value of relevant results that are placed lower down on the rankings. The latter is a simple measure of how many search results are deemed relevant or not at different positions  $k$  of all results. The formulas can be found below:

$$DCG_k = \sum_{i=1}^k \frac{rel_i}{\log_2(i+1)}$$

$$nDCG_k = \sum_{i=1}^k \frac{DCG_k}{IDCG_k}$$

$$Precision_k = \frac{1}{k} \sum_{i=1}^k \mathbb{1}_{(rel_i > 0)}$$

where  $IDCG_k$  is the ideal discounted cumulative gain at  $k$ , found by calculating  $DCG_k$  if the results were ordered in descending order from most relevant to least relevant, and  $rel_i$  is the relevance label in  $[0, 3]$ .

Group 1	Group 2
coronavirus spread	zombie attack
greta thunberg cross atlantic	vegan bakery
black hole image	learn swedish
story about riding a bird	best movies
daniel ek interview	
michelle obama becoming	
anna delvey	
facebook stock prediction	

Table 1: Queries used for evaluation. Group 1 contains the queries from the dataset paper. Group 2 are the queries that we have defined ourselves.

To take into account that we are evaluating the relevance of the search results individually, we use the Fleiss' Kappa as a measure of the degree of agreement in the classification. The kappa,  $\kappa$ , can take on values in the range of  $(-\infty, 1]$  and is defined by the following equation:

$$\kappa = \frac{P - P_e}{1 - P_e}$$

where the numerator represents the degree of agreement achieved above chance, while the denominator represents the degree of agreement that is possible above chance. A kappa of 1 means that the evaluating parties are in complete agreement, while scores below 0 mean that there is no agreement other than what one would expect by chance.

To calculate these values, we use the equations below. Let  $N$  be the number of search results,  $n$  be the number of ratings per search result,  $k$  be the number of labels on the assessment scale, and  $n_{ij}$  be the number of raters who labelled the  $i$ -th search result with the  $j$ -th category. In our case,  $N = 12 * 10 = 120$  as we have 12 queries returning 10 results each,  $n = 4$ , the number of students in our group, and  $k = 4$ .

First, we calculate the proportions of all assignments made to each category:

$$p_j = \frac{1}{Nn} \sum_{i=1}^N n_{ij}$$

We can then calculate the degree of agreement, that is how many rater pairs of the total ones that agree on their labelling, for the  $i$ -th subject:

$$P_i = \frac{1}{n(n-1)} \sum_{j=1}^k n_{ij}(n_{ij} - 1)$$

$P_e$  and  $P$  can then be calculated with the following expressions:

$$P = \frac{1}{N} \sum_{i=1}^N P_i$$

$$P_e = \sum_{j=1}^N p_j^2$$

The joint labels from the entire group of evaluators was chosen to be the most common label for each search result across the group. In cases where there was a tie, the lower-relevance label was chosen to avoid too optimistic estimates of the relevance.

To reduce the possibility of bias affecting the kappa, each of us were asked to rate the retrieved results individually without discussing anything during the evaluation process.

## 4 Results and discussion

### 4.1 Evaluation

$\kappa$	0,274
$P$	0,508
$P_e$	0,643

Table 2: Results from computing the agreement between raters

The calculated  $\kappa$  was higher than zero, showing that there was some form of agreement between the ratings. It is not entirely trivial to quantify whether the score is poor or not, as the magnitude of  $\kappa$  is affected by the number of categories. However, since a score of one indicates perfect agreement, the score we achieved seems to point towards a somewhat lower degree of agreement.

Looking at the labels in detail, all raters were generally in perfect agreement regarding several of the non-relevant results. For results that were a bit more ambiguous, labels were much more spread out however. While there was no result that got assigned all four different labels, there were several ones that had a 2:1:1 or 2:2 split in labelling. There were also a few instances of 2:2 splits where the labels differed by more than one step on the evaluation scale.

## 4.2 Search results

Table 3 and 4 compares the nDCG's of our search with the results in the dataset paper with five and ten results respectively. Both experiments give similar results on average. However, the problem in the dataset paper is different because full episodes are returned instead of small fragments as in our experiment. This affects the results as well. For example it is easier to find a small fragment about the book 'becoming' by Michelle Obama than a full episode about this book. The comparison also shows there are easy and harder queries. For example, searching for 'Anna Delvey' gave no relevant results in both experiments. On the other hand 'coronavirus spread' gave good results for both. Table 5 show the precision with five and ten results for every test query.

Query	nDCG own search @5	nDCG paper @5
coronavirus spread	0.524	0.6655
Greta Thunberg cross Atlantic	0	0.5801
black hole image	0.237	0.8721
story about riding a bird	1	0
daniel ek interview	0	0
michelle obama becoming	0.697	0.0838
anna delvey	0	0
zombie attack	0.76	-
vegan bakery	0	-
learn Swedish	0.386	-
best movies	0.56	-

Table 3: nDCG scores for test queries with five results

Query	nDCG own search @10	nDCG paper @10
coronavirus spread	0.712	0.6717
Greta Thunberg cross Atlantic	0.289	0.4742
black hole image	0.442	0.7921
story about riding a bird	1	0
daniel ek interview	0	0
michelle obama becoming	0.861	0.643
anna delvey	0	0
facebook stock prediction	0.19	0.6005
zombie attack	0.887	-
vegan bakery	0.301	-
learn Swedish	0.591	-
best movies	0.693	-

Table 4: nDCG scores for test queries with ten results



Query	Precision @5	Precision @10
coronavirus spread	0.6	0.5
Greta Thunberg cross Atlantic	0	0.1
black hole image	0.2	0.2
story about riding a bird	0.2	0.1
daniel ek interview	0	0
michelle obama becoming	0.6	0.5
anna delvey	0	0
facebook stock prediction	0.166	0.143
zombie attack	0.2	0.2
vegan bakery	0	0.1
learn Swedish	0.2	0.2
best movies	0.6	0.4

Table 5: Precision scores for test queries with five and 10 results

## 5 Conclusions

We created a functional search engine for a large scale dataset of transcribed podcasts. Compared to other podcast search engines, we implement the novel feature of relevant segment retrieval instead of retrieving whole episodes.

Podcasts usually last for around an hour and they cover a broad variety of topics, our fine grained capability allows our engine to find more relevant sections and display them with timestamps so the user is able to jump straight into the interesting part.

The dataset transcripts are automatically generated from a speech recognizer, although human labeled data would make for better results that is infeasible due to the scale of the dataset. It is also shown that using these automatically generated transcriptions the engine is still able to retrieve relevant segments.

There are some noticeable drawbacks of relying on the automatic speech recognizer however. One of our ideas was to boost the result scores by the confidence scores given by the speech recognizer to the clips. In reality, it did not seem to work in the intended way as we were able to identify cases where clips in other languages had been transcribed to English words with a high confidence score. Furthermore, queries such as "Daniel Ek interview" and "Anna Delvey" returned a lot of results containing "EK" and "Anna" respectively, but as was observed in the dataset paper [Jon+21], some names were just not transcribed properly by the speech recognizer.

While the results we obtained were similar to the results in the dataset paper, it is also important to note that the nDCG scores are fairly sensitive to changes. First of all, the assessment is entirely qualitative and with only four raters, just one rater could potentially change the scores by a lot. An example of this is the "story about riding a bird" which achieved a nDCG of 1 in our evaluation which would indicate a very good result. Looking at the actual labels reveals that this

was only due to the fact that none of the results were deemed relevant except the first one, and even that result would have been labelled as irrelevant had one more rater given it a 0. Since the kappa also shows that ratings were not entirely in agreement, it is possible that raters had slightly different interpretations of the queries and the actual information need embedded in those queries. If we were to do the same experiments again, it could be wise to also include a short description of the information need for every query.

## References

- [Cli+20] Ann Clifton et al. “100,000 Podcasts: A Spoken English Document Corpus”. In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 5903–5917. DOI: 10.18653/v1/2020.coling-main.519. URL: <https://www.aclweb.org/anthology/2020.coling-main.519>.
- [Fan] Wenbin Fang. *The boring technology behind a one-person Internet company*.
- [Jon+21] Rosie Jones et al. “TREC 2020 Podcasts Track Overview”. In: *arXiv preprint arXiv:2103.15953* (2021).
- [Jov21] Danica Jovic. *40 Powerful Podcast Statistics to Tune Into*. <https://www.smallbizgenius.net/by-the-numbers/podcast-statistics/#gref>. 2021.
- [Lam] Paul Lamere. *Spotipy: A python Spotify Web API*.
- [MOG08] Junta Mizuno, Jun Ogata, and Masataka Goto. “A similar content retrieval method for podcast episodes”. In: *2008 IEEE Spoken Language Technology Workshop*. IEEE. 2008, pp. 297–300.
- [SP99] Amit Singhal and Fernando Pereira. “Document expansion for speech retrieval”. In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. 1999, pp. 34–41.
- [Swe] Mark Sweney. *Spotify credits podcast popularity for 24% growth in subscribers*.

## A Search Object

```
search_object = { "from" : 0, "size" : k,
  'query': {
    "nested": {
      "path": "clips",
      "query": {
        "function_score": {
          "field_value_factor": {
            "field": "confidence",
            "factor": 1.0,
            "modifier": "none",
            "missing": 1
          },
          },
        "query": {
          "bool": {
            "should": [
              {
                "match": {
                  "clips.transcript": word
                }
              }
            ],
            "minimum_should_match": 1
          }
        }
      }
    }
  },
  "inner_hits": {},
  "score_mode": score_mode
}}
```