Project report

Search algorithms for spoken passage retrieval on the Spotify dataset.

Tim Widmayer [1], Tchatchawin Leelawat [2] and Vincent Lefeuve [3]

[1] University College London, tim.widmayer.20@ucl.ac.uk
[2] University College London, thatchawin.leelawat.19@ucl.ac.uk
[3] University College London, vincent.lefeuve.20@ucl.ac.uk

Abstract: The X5Learn platform hosts thousands of learning resources in different formats. Most resources are in video, audio or PDF formats which cannot be searched directly. Therefore, search engines have to rely on computer-generated, inherently noisy transcripts; i.e., they contain transcription errors. We tested different search engine implementations on a relevance-labelled dataset of podcasts from Spotify. This project aims to demonstrate the feasibility of implementing such a search engine in Elasticsearch and if some "classic" algorithms perform better than others while also comparing this with some state-of-the-art Learn-To-Rank models.

Keywords: Search, Noisy Data, Software Engineering, Machine Learning, Information Retrieval

Introduction

This project aims to build a search engine to retrieve relevant documents for noisy data. This "noise" comes from the fact that some documents from the X5Learn database are in PDF or video format. As they need to be transcribed into text, some errors can appear due to the imperfection of the transcription algorithms. The noisy nature of these documents creates several problems for searching, especially for names and domain-specific words which are often wrongly transcribed. To find a search algorithm, we compare different techniques using nDCG on a dataset with judgement lists from Spotify. How will different algorithms perform with this type of data? We tried to compare the efficiency of BM25, Dirichlet Smoothing, and LambdaMart with nDCG@5 and nDCG@10. We also try to get an idea of the efficiency of stemming by comparing these algorithms with and without it.

Citation: Widmayer, T.; Leelawat, T.; Lefeuve, V. Search algorithms for spoken passage retrieval on the Spotify dataset, 2022

I – Background and dataset

II.1. Background

Search algorithms are a family of algorithms that involve a search problem, usually to retrieve information saved in a data structure. That is to say; given a query q and a set of documents d, a search algorithm returns a list of documents from d ordered by relevance to q. In this paper, we are interested in search algorithms for web search engines. With the increase of internet usage, search engines like Google or Bing are used by millions of people worldwide each day.

Our project aimed to build a search engine for the X5Learn project, an open educational ressources (OER) database that comprises content from different sites in one web application [2]. Some documents are in non-textual forms, like video lectures and podcasts. This makes search more difficult as these documents will have to be transcribed before searching, thus containing transcription errors called noise.

In this report, we are interested in building a search engine that retrieves content from *The Spotify Podcast Dataset* [7]. As this podcast contains transcripts from podcasts, it also contains noise. Thus, this implementation results can help us build the search engine for X5GON's OER platform.

II.2. Related work

Related works include *100,000 Podcasts: A spoken English Document Corpus*, which uses the same dataset [6]. The method they use is to implement standard retrieval models (BM25 and Query Likelihood (QL, a Language Model) with Dirichlet Smoothing), with the RM3 relevance model using Pyserini [10]. This is built with the Lucene search Library [4], and stemming is added with the Porter Stemmer. They then compare the accuracy of the different algorithms using nDCG@10 and nDCG@5.
This project tries to follow a similar methodology to the aforementionned paper, while using new technologies and building libraries to allow developpers to test search algorithms on their datasets. Our approach is pretty similar. However, we use newer technologies like ElasticSearch in order to make the implementation easier. We also build on top of that and try Learn-to-rank (LTR) models for this task. LTR models are retrieval models which make use of machine learning.

II.3. Dataset

The dataset we use is *The Spotify Podcast Dataset* [7]. It is a sample of 105,360 podcasts episodes from the platform Spotify, coupled with relevant metadata. These episodes are transcribed using Google's cloud Text to Speech API [1]. The dataset includes 2 judgement lists: one called training, which included 8 queries, and one for testing. We decided to save the training judgement list for creating the LTR models.

We compiled the dataset into one CSV file with all relevant information. Due to performance and time constraints, we decided to only use a random subset of the dataset which includes the documents included in the training and testing judgement lists. The size of this subset is around 166,000, out of the total ~3M documents. A document, in our case, is a two-minute extract of a transcript with one minute overlap.

II – Implementation of the algorithms

This project is built using Python and ElasticSearch 7.11.2. To complement this paper, we created a python package[1] which allows to implement these algorithms and test them. In this paper, we build and compare the following algorithms: BM25, Language Model, and LambdaMart. We also implemented stemming to the search algorithms.

II.1. Classic algorithms

The classic search algorithms we decided to implement are Okapi BM25 [12] and a Language model with Dirichlet Smoothing [15]. These two algorithms are natively built-in in ElasticSearch.

We used the following parameters for the algorithms:

- K1 = 0.9, b=0.4 for BM25
- μ = 1000 for Dirichlet Smoothing

These parameters are the same as in the paper "100,000 Podcasts: A Spoken English Document Corpus" [6] in order to make results as comparable as possible.

II.2. LambdaMart and LTR

In order to build a LambdaMart model, we used Ranklib and a plugin for ElasticSearch called ElasticSearch LTR.

We built the models using the following features:

Table 1. Feature list for the LTR models. [9][13]

Name	Description
BM25_Score	Okapi BM25 score for the document on the query. (with and without stemming)
sum_df	the direct document frequency for a term
sum_idf	1 divided by the number of documents containing the query term summed for each term.
sum_ttf	the total term frequency for the term across all documents
Sum_tf	Sum of counts of each query term in the document
min_tf	Minimum of counts of each query term in the document
max_tf	Maximum of counts of each query term in the document
Min_idf	1 divided by the number of documents containing the query term (minimum)
Max_idf	1 divided by the number of documents containing the query term (maximum)
Avg_tf	Average of counts of each query term in the document
Avg_idf	1 divided by the number of documents containing the query term (averaged for each term)
Stddev_tf	Standard deviation of counts of each query term in the document
Stddev_idf	1 divided by the number of documents containing the query term (Stddev of each term idf)

We then constructed the training file using the judgement list and adding the feature score for each (query, document) pair. In order to add more documents and improve the model, we added the document relevant to the other queries to each query, judging them as having 0 relevance.

We then trained two LambdaMart models with the default RankLib parameters: one with, and one without stemming in the BM25_Score feature. The model trained with stemming will be used as a comparison and with stemming on top, as using stemming on a model trained without it could decrease performance. We then incorporated both models into Elasticsearch.

To help with training and uploading models to Elasticsearch, we created a Python module[1]

II.3. Stemming

Stemming is "a computational procedure which reduces all words with the same root (or, if prefixes are left untouched, the same stem) to a common form, usually by stripping each word of its derivational and inflectional suffixes."[11] We decided to compare the efficiency of these different algorithms with and without it.

In order to add stemming to our search engine, we used Elasticsearch's functionality [3]. As all of the queries in the dataset are in English, we used ElasticSearch's English stemmer. It is added each time the engine is loaded with data.

[1] The module is available on github : https://github.com/COMP0016-IR-Noisy-documents/IR_RankingAlgorithms

III – Testing & results

III.1. Testing methodology

In order to test the efficiency of these algorithms, we used the judgement list given in the dataset. We chose to use nDCG@10 and nDCG@5 as metrics for accuracy, which is given by:

$$\mathrm{nDCG}_k = \frac{DCG_k}{IDCG_k} \text{ ,where:}$$

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

$$IDCG_k = \sum_{i=1}^{|rel_k|} \frac{rel_i}{log_2(i+1)}$$

Where k= 5 or 10, rel_i is the relevance score of document I on the judgement list, and rel_k represents the list of relevant documents (ordered by their relevance) in the corpus up to position k.

To perform the testing, we used Elasticsearch's ranking evaluation module [16], which allows getting the *nDCG@k* scores for a set of queries given a list of (query, document, score) triples. We used that method to get nDCG@10 and nDCG@5 scores for all of the mentioned algorithms with and without stemming, for all of the queries in the judgement list.

III.2. Results

Using the methodology described above, we ran the tests and compiled the results for all queries in the testing judgement list.

Table 2. nDCG scores for different search algorithms.

Algorithm	nDCG@5	nDCG@10
BM25	0.290	0.302
BM25 + Stemming	0.316	0.326
QL	0.289	0.312
QL + Stemming	0.304	0.311
LambdaMart	0.295	0.324
LambdaMart + Stemming	0.320	0.327

III.3. Analysis

As we can see, algorithms with stemming consistently perform better than their counterparts. It also seems like the best performing retrieval algorithm is LambdaMart with stemming. However, using Wilcoxon rank-signed tests [14], we found that the algorithms are not distinguishable, as none of the results is statistically significant. This lack of difference could be due to the limitations of the bag-of-words strategy [6]. We cannot reject the hypothesis that they all follow the same distribution.

Some queries suffer a lot from the noise created by automatic speech transcription. For example, the query *Fauci interview* consistently gets an nDCG score of 0 for all of the algorithms.

Using a larger sample of the dataset could make the results more precise.

Conclusions and discussion

We built a search engine using ElasticSearch. Using this technology proved practical and has many valuable functionalities for information retrieval built-in.

Unfortunately, our test results are not significant and cannot help us to draw any definite conclusion. Thus, after reading more literature on search for information retrieval [5], we decided to implement LM with Dirichlet Smoothing to the X5Learn web application.

Future work could include improving the LambdaMart models. Indeed, some terms are often wrongly transcribed (usually names and complicated words), and adding the BM25 score with fuzziness to the list of features could help tackle that. Furthermore, using a larger dataset like the TREC datasets [5] would help gather significant knowledge about which retrieval technique is more accurate.

References

[1] "Google Cloud Speech-to-T." [Online]. Available: https://cloud.google.com/speech-to-text.

[2] "x5learn.org." [Online]. Available: https://x5learn.org/.

[3] "stemming @ www.elastic.co." [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/stemming.html.

[4] Apache, "Lucene." [Online]. Available: http://lucene.apache.org/.

[5] G. Bennett, F. Scholer, and A. Uitdenbogerd, *A Comparative Study of Probabalistic and Language Models for Information Retrieval.* 2008.

[6] A. Clifton, M. Eskevich, G. J. F. Jones, B. Carterette, and R. Jones, "100 , 000 Podcasts : A Spoken English Document Corpus," pp. 5903–5917, 2020.

[7] A. Clifton *et al.*, "The Spotify Podcast Dataset," 2020, [Online]. Available: http://arxiv.org/abs/2004.04270.

[8] D. Davies, "How Search Engine Algorithms Work: Everything You Need to Know," *Search Engine J.*, 2020, [Online]. Available: https://www.searchenginejournal.com/search-engines/algorithms/#whysearc.

[9] X. Han and S. Lei, "Feature Selection and Model Comparison on Microsoft Learning-to-Rank Data Sets," vol. 231, no. Fall, 2018, [Online]. Available: http://arxiv.org/abs/1803.05127.

[10] lintool, "PySerini @ pypi.org." https://pypi.org/project/pyserini/.

[11] J. B. Lovins, "Development of a stemming algorithm," *Mech. Transl. Comput. Linguist.*, vol. 11, pp. 22–31, 1968.

[12] S. E. Robertson and K. Sparck Jones, "Relevance Weighting of Search Terms," in *Document Retrieval Systems*, GBR: Taylor Graham Publishing, 1988, pp. 143–160.

[13] D. Turnbull, E. Berhardson, D. Causse, and D. Worley, "Elasticsearch Learning to Rank Documentation," 2019. [Online]. Available: https://media.readthedocs.org/pdf/elasticsearch-learning-to-rank/latest/elasticsearch-learning-to-rank.pdf.

[14] W. J. Conover, "Practical Nonparametric Statistics, 3rd Edition."

[15] C. Zhai and J. Lafferty, "A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval," *SIGIR Forum*, vol. 51, no. 2, pp. 268–276, 2017, doi: 10.1145/3130348.3130377.

[16] "Google Cloud Speech-to-T." [Online]. Available: https://cloud.google.com/speech-to-text.