Performance Comparison of Information Retrieval Models Against a Cranfield Document Collection

CA6005I – Assignment 1

Andrey Totev
School of Computing
Dublin City University
Dublin, Ireland
andrey.totev2@mail.dcu.ie

ABSTRACT

This work presents TIRS - a textual information retrieval system. The system supports three different information retrieval (IR) models for processing user queries: a vector-space model, the BM25 probabilistic model, and a language model with Jelinek-Mercer Smoothing. We compare the accuracy of the individual IR models using a well-known evaluation dataset. Our experiments show that the probabilistic and language models outperform the vector space model significantly while the difference between BM25 and the language model is marginal and could be influenced by the model parameters as other works in this area have demonstrated.

KEYWORDS

Information Retrieval, vector space model, BM25

1 Introduction

(Source code: https://gitlab.computing.dcu.ie/toteva2/ca6005i)

1.1 Functionality

TIRS is a REST-based search service accessible via a command line client. The system can index documents, execute queries against the index, and report back a ranked list of relevant results.

1.2 Operation

Once the service component is up and ready to serve requests, the user can issue the following commands via the client application:

- Select the active IR model
- Configure the active IR model
- Execute a query using the currently configured IR model

1.3 Architecture

The application consists of a REST service based on the Flask framework [1]. The service hosts the Index data structure and three IR components implementing each of the models. It also features a Selector component with ephemeral configuration holding the currently active model and its parameter values. There are two REST resources available (/, /config) and their respective commands in the CLI client (<nil>, set.) The current solution assumes that the "raw" documents are stored in a shared file system location accessible to both the service application and the users responsible for indexing new content.

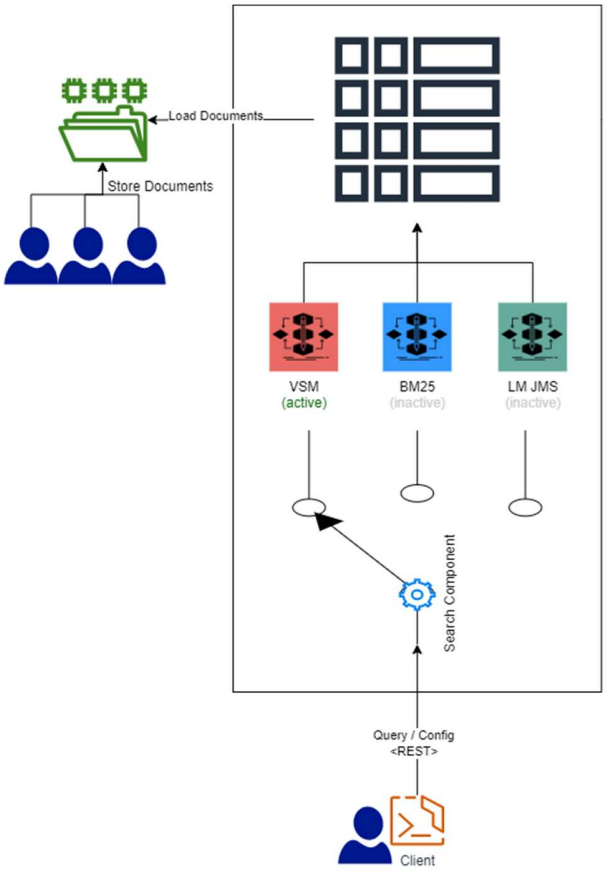

Figure 1: TIRS Architecture

1.4 REST API

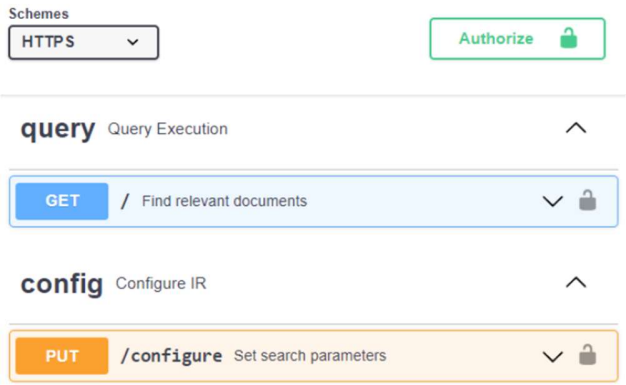

Figure 2: TIRS REST API

1.4 Command line interface

The following commands are supported by the application:

- tirs search "query terms" # searches for relevant documents
- tirs configure active <vsm,bm25,lm> # set active IR model
- tirs configure <param> <value> # configure parameter for active model

2 Indexing

2.1 Document analysis and pre-processing

Once the document text extraction and preprocessing is complete, the occurrences of each term are counted using the Python Counter utility. The term counts and the document ID are used to create postings and append them to the corresponding posting list in the index structure.

2.2 Indexing construction

The Index component supports ingestion of documents from the CLEF ad-hoc robust task collection. The following actions are performed upon indexing a new document:

- Load the document DOM using the lxlm [1] library
- Extract the DOCID and TEXT elements
- Remove stopwords using the NLTK facility [2]
- Remove 1-symbol terms and non-alphabetical terms
- Lowercase
- Lemmatize using NLTK WordNet Lemmatizer [2]

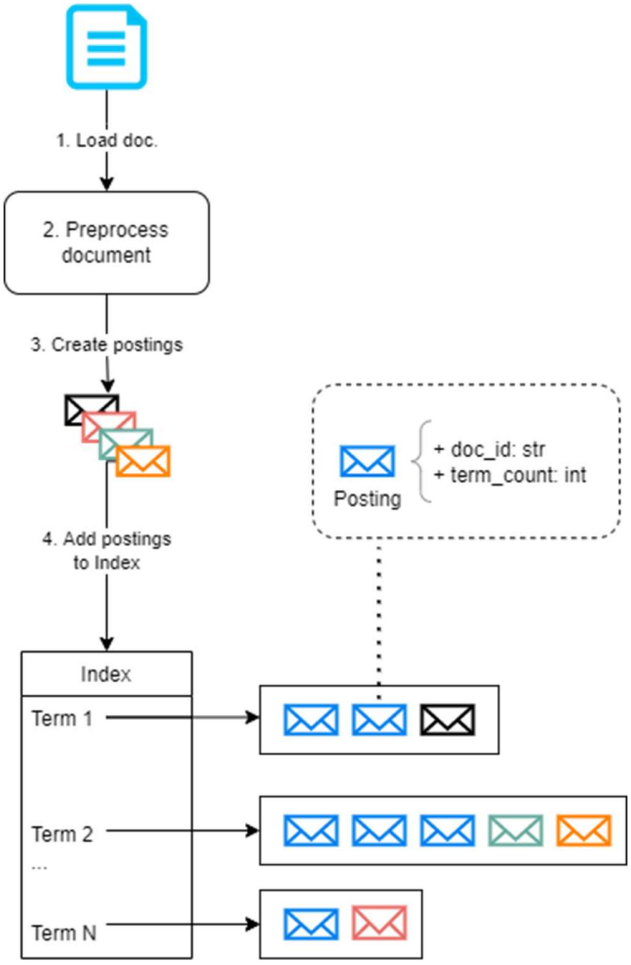

Figure 3: Indexing construction

2.3 Data structures

The index is implemented using a dictionary mapping the term to its postings list. Each posting contains the document identifier and the count of occurrences in the document of the mapped term.

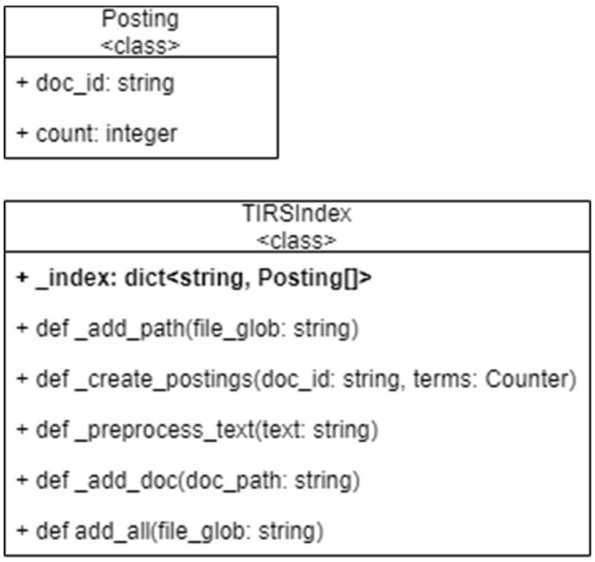

Figure 4: Index class diagram

3 Search and Ranking

3.1 Component structure

The search and ranking implementation leverages the Strategy design pattern [3] where each IR model class extends a RankingStrategy abstract base class. This allows for a pluggable design where new IR model variants can be seamlessly integrated within the application. The search component provides two basic operations - execute and configure bound to the "/" and "/configure" API resource paths respectively. The *execute* operation processes a text query and returns a ranked list of relevant documents, while *configure* is used to 1) select the currently active IR model and 2) set parameter values for the currently active model.

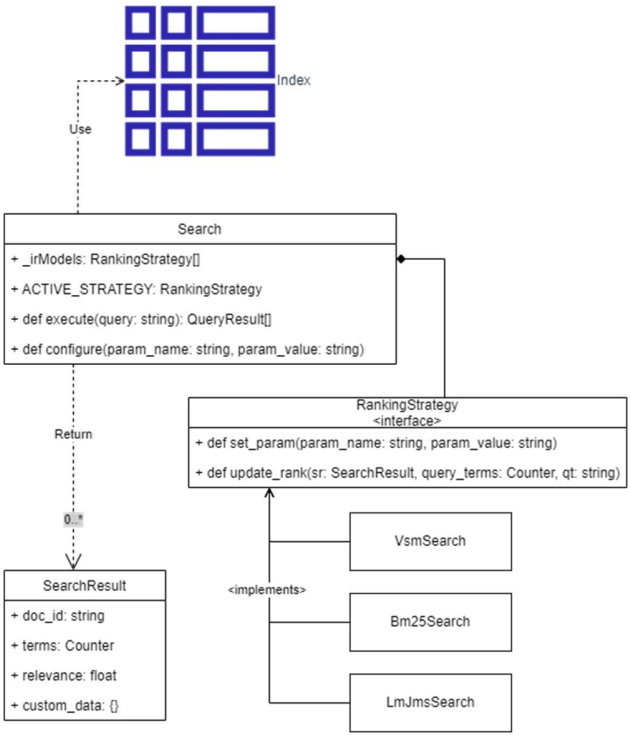

Figure 5: Search component

3.2 Search and ranking workflow

For each term in the query, the search function iterates over the posting list associated with the term in the inverted index. If the document from the current posting is visited for the first time, it is loaded from the file system, preprocessed, and its term counts are added to a newly created SearchResult object. The SearchResult object for the current posting is passed to the active ranking function along with the term and the counts of all query terms. The ranking function calculates the document-term relevance and updates the cumulative value in SearchResult. At the end, the search results are sorted by relevance in descending order and clipped to the 1000th element before returning.

It is important to note that unlike VSM and BM25, the language model's ranking for a document has a component which depends on the term frequency across all documents, therefore every "candidate" document is passed to the ranking function for every query term (even if the term is missing from the document.)

```
  search_results = {}
    query_terms = count_terms(query_text)
    query_terms = remove_unkown(query_terms)
    for qt in query_terms:
        posting_list =
Index.get_posting_list(qt)
        for p in posting_list:
            if p.docid not in search_results:
                sr = SearchResult(p.docid)
                sr.filepath =
get_document_file(p.docid)
```

```
                _, sr.headline, text =
load_document(sr.filepath)
                sr.terms = count_terms(text)
                search_results[p.docid] = sr
    for qt in query_terms:
        for docid in search_results:

_ACTIVE_RANKING.update_rank(search_results[doci
d], query_terms, qt)
    result = list(search_results.values())
    result.sort(reverse=True, key=lambda sr:
sr.relevance)
    return result[:Search.RESULT_LIST_SIZE]
```

Listing 1: Search and ranking workflow

3.3 Information retrieval models

The information retrieval models are abstracted by the RankingStrategy base class. TIRS supports three different RankingStrategy implementations: vector space, BM25 probabilistic, and language model with Jelinek-Mercer smoothing.

The vector space model relies on a sparse vector representation of each document/query where the individual weights are calculated using the following TF*IDF scheme [4]:

$$(1 + log f_{i,j}) \times log(1 + \frac{n}{df_j})$$

The relevance of each document to the query is determined based on the cosine similarity between their respective vector representations.

The BM25 model is based on the probability ranking principle which orders the search results according to their probability of being relevant to the query. The probability of each document term being relevant to the query is calculated using the following expression with two parameters (k1, b where typically 1.2 < k1 < 2 and 0.5 < b < 0.8) [5]

$$w^{RSJ} = log \frac{N - n + 0.5}{n + 0.5}$$

$$w^{BM25} = \frac{tf}{k_1 \left((1 - b) + b \frac{dl}{avdl}\right) + tf} w^{RSJ}$$

The language model with Jelinek-Mercer smoothing uses a query likelihood model which orders the documents based on a maximum likelihood estimation of each query term being generated by the language model of the document. The model uses a parameter in the range [0, 1] [6].

$$\hat{P}_\lambda(t \mid M_d) = (1 - \lambda) \frac{C_d(t)}{\sum_{t'} C_d(t')} + \lambda \frac{C_\mathcal{D}(t)}{\sum_{t'} C_\mathcal{D}(t')}$$

3 Evaluation

A dedicated evaluation script is created using the TIRS inverted index and search classes. The TEXT elements from all documents in the collection are indexed and the TITLE of each test topic is used as a search query. The BM25 model is configured using B=0.75, k1=1.25 and the language model is using lambda=0.25. The evaluation is performed using the Text REtrieval Conference (TREC) standard methodology, using the MAP, P@5, and NDCG metrics. Like [7], our findings show comparable performance for BM25 and LM which both exceed the VSM by significant 50% in this experiment. Unlike the results in [7], our BM25 variant exceeds the LM by 4% in all three metrics - possibly due to using a different value for the lambda parameter. The computation efficiency of all three methods is similar, with VSM taking approximately 10% longer to complete than BM25 and LM.

	VSM	BM25	LM-JMS
MAP	0.1848	**0.2746**	0.2598
P@5	0.1712	**0.2500**	0.2375
NDCG	0.3484	**0.4270**	0.4162
Duration	1:01:26	55:39	55:36

Table 1: Evaluation results

3 Conclusions

In this report we presented TIRS – a desktop IR system with a client-server architecture allowing pluggable ranking functions. We also evaluated three IR models against a Cranfield document collection – a vector space model, BM25 probabilistic model, and a language model with Jelinek-Mercer Smoothing. Our findings show that BM25 and LM-JMS significantly exceed the VSM model performance and, for the selected parameters, BM25 is superior to the language model by 4%. In future experiments, we could evaluate the model performance against a "grid" of parameter values in order to select the most performant model and configuration against the test data set. Other possible improvements include the adoption of more intelligent text preprocessing techniques such as part-of-speech tagging or named entity recognition. Additional improvements include better use of the available document/query data by leveraging the remaining fields: DESC, NARR, and HEADLINE.

REFERENCES

[1] Anon, Welcome to flask¶. Welcome to Flask - Flask Documentation (2.0.x). Available at: https://flask.palletsprojects.com/en/2.0.x/ [Accessed February 23, 2022].

[2] Lxml.de. 2022. lxml - Processing XML and HTML with Python. [online] Available at: <https://lxml.de/> [Accessed 22 February 2022].

[3] Anon, NLTK. Available at: https://www.nltk.org/ [Accessed February 22, 2022].

[4] Anon, 2021. Strategy pattern. Wikipedia. Available at: https://en.wikipedia.org/wiki/Strategy_pattern [Accessed February 22, 2022].

[5] G. Salton. 1971. The SMART Retrieval System—Experiments in Automatic Document Processing. Prentice-Hall, Inc., USA.

[6] Robertson, Stephen E., et al. “Okapi at TREC-4.” Nist Special Publication Sp (1996): 73-96. Robertson, Stephen, and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. Now Publishers Inc, 2009.

[7] Anon, Language models LM Jelinek-Mercer Smoothing and LM ... Available at: http://ctp.di.fct.unl.pt/~jmag/ir/slides/a05%20Language%20models.pdf [Accessed February 22, 2022].

[8] Bennett, G., Scholer, F. and Uitdenbogerd, A., 2008. A comparative study of probabilistic and language models for information retrieval. In Database Technologies 2008: Proceedings of the Nineteenth Australasian Database Conference (ADC 2008) (pp. 65-74). RMIT University.Conference Name:ACM Woodstock conference