# CS-423 Document Retrieval

Vincent Roduit
MSc student, EPFL
Electrical Engineering
Sciper: 325140

Yann Cretton
MSc student, EPFL
Life Sciences
Sciper: 274709

Fabio Palmisano
MSc student, EPFL
Electrical Engineering
Sciper: 296708

**Team name**: SLO-remake
**Link to Kaggle Notebook**

***Abstract*—Document retrieval is an important task in text processing. With the fast expansion of data, it is crucial to have search engines that can efficiently retrieve documents, given a set of queries. Retrieval systems are widely used, and probably the best example is the internet search engine. This report will explore several retrieval techniques, including word embedding methods and TF-IDF coupled with the BM25 ranking algorithm.**

***Keywords*—** Document retrieval, BM25, Word2Vec, Word Embedding, TF-IDF

## I. Introduction

This report documents the work completed for Project 1 of the course *CS-423: Distributed Information Systems*, offered at EPFL during the Fall semester of 2024. The project's objective was to explore various techniques and build a fully functional retrieval system that returns the top ten documents matching a given query. This report outlines two distinct approaches. The first employs Word2Vec as a word embedding method, while the second—proven to be more efficient—leverages a TF-IDF approach in combination with the BM25 ranking algorithm. The complete code for the optimal solution and supporting materials can be found on Github.

## II. Data Exploration

Before diving into the algorithm, it is necessary to understand the data to better tackle the problem. There are different data available:

- *corpus.json*: This file contains a list of documents, that constitute the database for the retrieval system.
- *dev.csv*: This file contains a set of queries. The corresponding positive document and ten negative documents are attached to these queries. The language of the query is also available.
- *train.csv*: This file is similar to *dev.csv*.
- *test.csv*: This file is the one that should be used for testing the retrieval system. Only the query, as well as the language is available in this document.

Each file consists of documents (and queries, respectively) in six different languages: English, French, German, Italian, Spanish, Korean, and Arabic. Additionally, since both proposed methods are unsupervised, information about negative documents is not utilized.

TABLE I
Summary of the different datasets

| Dataset | #en | #fr | #de | #it | #es | #ko | #ar |
|---|---|---|---|---|---|---|---|
| corpus.json | 21e3 | 11e3 | 11e3 | 11e3 | 11e3 | 7893 | 8829 |
| train.csv | 10e3 | 1608 | 1847 | 2151 | 2254 | 2198 | 1817 |
| dev.csv | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| tets.csv | 200 | 200 | 200 | 200 | 200 | 200 | 200 |

The main challenge is to achieve good performance for smaller corpora, such as those in Korean and Arabic. Indeed, word embedding techniques tend to perform better for languages with a larger number of examples. It is more difficult for such methods to build effective representations for corpora with fewer examples. Section IV will present some approaches to address this issue, and Section VI will show the results of these different strategies.

## III. Data Preprocessing

Regardless of the method, the first step is to preprocess the data. This step aims to remove unnecessary words and punctuation. The main objective is to significantly reduce the vocabulary size and eliminate redundant words that are not relevant. In the literature, such words are referred to as *stopwords*. For this project, the NLTK library has been used, which provides useful tools such as *nltk.stopwords*, offering lists of stopwords for various languages. Since stopwords for Korean are not available in NLTK, an external set was used[1]. The complete preprocessing involves:

1) Tokenizing the text: For this step, the **RegexpTokenizer** class from NLTK was used. It removes all characters except letters and numbers.
2) Converting all words to lowercase: This step ensures that "Cat" and "cat" are not treated as two different words.
3) Remove stopwords: Stopwords are common words in a language that do not provide context. For instance, in English, words such as 'I' and 'only' are considered stopwords. Removing them reduces the vocabulary size without affecting the meaning of the documents.
4) Applying stemming: Stemming reduces words to their root form. For example, "changing", "change", and "changed" are all reduced to the root "chang". This helps minimize the vocabulary size and aids in the computation of TF and IDF values.

To remove the stopwords, the library NLTK [2] has been used. Since this library does not provides stopwords for Korean, a specific list has been added for this langage [3]

## IV. Methods

This sections presents the two methods used in this project, namely Word2Vec and TF-IDF (with BM25)

### A. Word2Vec

Word2Vec is a Natural Language Processing method based on word embedding techniques [1]. In this family of models, a word is represented as a high-dimensional vector (typically consisting of several hundred dimensions). Two words that are semantically similar should be close in terms of cosine similarity within the vector space.

There are two different architectures to produce the model: Continuous Bag of Words (CBOW) and the Skip-gram model. In

[1] https://github.com/stopwords-iso/stopwords-ko

this project, the CBOW method has been explored. CBOW takes a group of context words and averages their embedding vectors. This averaged vector is then compared to the embedding of a target (or center) word. The goal is to ensure that the embedding of the context words is similar to that of the target word when they co-occur in a document. In other words, CBOW captures the semantic relationships between words, helping the model learn the meaning of a word based on its neighboring words.

After mapping each word into the vector space, each document needs to be represented as a single vector. Several methods exist for this transformation, with one of the simplest being to take the mean of the word vectors. Thus, a document can be represented as:

$$d = \frac{1}{W} \sum_{i=1}^{W} w_i \tag{1}$$

where $w_i$ represents a word vector in the document. This solution is clearly sub-optimal, as it does not take into account the frequency of a word.

A more advanced solution could be to add the TF-IDF value as a weight to each vector. A document will then be represented as:

$$d = \frac{1}{W} \sum_{i=1}^{W} w_i * TF_i * IDF_i \tag{2}$$

Finally, retrieval is performed using cosine similarity. After converting the query into a vector using the same process described above, cosine similarity is calculated between this query vector and each document vector. The documents with the highest scores are then recommended.

### B. BM25

This sections will focus on the BM25 ranking and will explain the steps that leads to the retrieval system.

*1) Constitutive elements:* TF-IDF is another way to embed words. TF stands for term frequency and is defined as:

$$TF(i,j) = \frac{freq(i,j)}{max_{k \in T} freq(k,j)} \tag{3}$$

Similarily, IDF stands for inverse document frequency and BM25 uses a smoothed version of it:

$$IDF(t) = \log\left(1 + \frac{N - DF(t) + 0.5}{DF(t) + 0.5}\right) \tag{4}$$

where DF (Document Frequency) is defined as:

$$DF(t) = \sum_{d \in D} \mathbb{1}_{t \in D} \tag{5}$$

where t represents a term and D is the set of documents. DF will be necessary for the computation of the BM25 algorithm.

*2) Okapi BM25 ranking:* Now that all the necessary elements have been presented, we can introduce the formula used for ranking. Okapi BM25 is a ranking function based on a probabilistic retrieval framework. It is a bag-of-words retrieval algorithm that ranks documents based on the presence of query terms, without considering the order or proximity of those terms within the document. The scoring function used is:

$$BM25(d,q) = \sum_{t \in q} IDF(t) \cdot \frac{TF(t,d) \cdot (k_1 + 1)}{TF(t,d) + length\_norm} \tag{6}$$

and

$$length\_norm = k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right) \tag{7}$$

where:

- $k_1$ is a constant that controls term frequency scaling.
- $b$ is a constant that controls the degree of length normalization.
- $avgdl$ is the average document length across the corpus.
- $|d|$ is the length of the document.

To retrieve documents, the scores are then sorted in a decreasing order and the k first documents will be retrieved.

## V. Optimization

This section presents optimizations performed on the BM25 algorithm to enhance performance and meet the requirements outlined in the project description. These optimizations address several issues, including time cost, memory constraints, and overall performance.

### A. Performance

To boost the performance of the basic BM25 algorithm, two improvements have been implemented. The first involves filtering the documents of interest by language. It is reasonable to assume that a query in German will retrieve German documents as positive results. Moreover, since information regarding the language of both the query and the documents is available, it is pertinent to leverage this information.

### B. Memory Constraints

Given that the BM25 algorithm requires TF, IDF, and other data that can be substantial due to the size of the corpus, it is essential to store this data efficiently. To address this issue, we use a mapper that converts each word in the vocabulary into a unique integer identifier. Consequently, a document that was initially represented as a list of tokens will be transformed into a list of integers. This allows the TF and IDF dictionaries to store lists of integers instead of strings, drastically reducing the size of the resulting files.

### C. Time Performance

The project description sets a time constraint of ten minutes to embed the queries and retrieve documents. Since the BM25 algorithm needs to compute the score between a query and each document in the relevant set, this task can be very time-consuming due to the size of the corpus. To mitigate this issue, we reduce the set of documents based on matching between the query and the documents. Documents are ordered according to the number of query words they contain. This solution reduces the number of scores that need to be computed and aims to maintain performance.

## VI. Results

This section presents the results obtained for the two methods, with a greater focus on the best model, namely BM25. To assess the performance of each model, tests were conducted using the *dev.csv* file and the recall@10 metric, followed by comparisons with the *test.csv* results on Kaggle.

### A. Word2Vec

This method proved to be inefficient, exhibiting a recall@10 value of 0.28 on the dev set and 0.25 on Kaggle. The analysis of scores separated by language does not reveal a clear trend. The primary reason for the low performance of this solution can be attributed to several factors. As mentioned in IV-A, aggregating all word vectors is not a good choice, as it tends to give more weight to frequent words. Even though stopwords have been removed, giving more weight to recurring words may not be ideal. Furthermore, since the matching documents are retrieved using cosine similarity, this aggregation biases the document representation, making it difficult for cosine similarity to retrieve the best documents. The second proposed method will address the issues identified in this model.

Nevertheless, the strength of this method lies in its computational speed. The retrieval function relies on matrix multiplication, which can be very efficient when utilizing libraries such as NumPy.

### B. Okapi BM25

This section presents the results of the BM25 retrieval system. The initial implementation consists solely of the original BM25 algorithm. Starting from this baseline, optimizations have been implemented, as described in V. Table II summarizes the different scores obtained.

TABLE II
SUMMARY OF THE RECALLS

| method | #en | #fr | #de | #es | #it | #ko | #ar | overall |
|---|---|---|---|---|---|---|---|---|
| basic | 0.76 | 0.9 | 0.69 | 0.93 | 0.80 | 0.63 | 0.74 | 0.78 |
| 10e3 | 0.88 | 0.9 | 0.69 | 0.93 | 0.8 | 0.63 | 0.74 | 0.8 |
| adaptative | 0.87 | 0.9 | 0.7 | 0.94 | 0.81 | 0.62 | 0.74 | 0.81 |

By inspecting this table, it can be observed that reducing the number of relevant documents using the method proposed in V benefits English. The other languages are not significantly affected, as the number of documents is almost equal to the total number of documents in the corpus[2]

### C. Effect of Filtering

This section examines the effects of applying filtering to the relevant documents. Figure 1 displays the recall for each language. It can be observed that for all languages except English, it may be sufficient to consider only the first 5000 documents. However, since the corpus is larger for English, it is necessary to increase this value to $10^4$ to avoid influencing the recall for that language. The optimal value can then be set to $10^4$ documents for english documents and 5000 for others. Table II indeed confirms that this choice leads to the best performance.

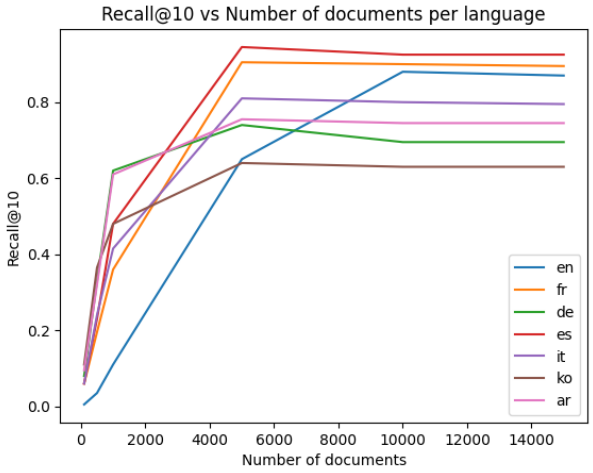


Fig. 1. Recall per language

Moreover, filtering also impacts computational time. Figure 2 illustrates the relationship between the number of relevant documents and the time required to compute the results. From this, it can be seen that time increases with the number of documents. However, the point $k = 10^4$ indicates that the time remains within the required limits.

It is important to note that this plot was generated on a MacBook Pro M3, so performance may differ from that observed on Kaggle. Nevertheless, the performance on Kaggle still meets the requirements, as the solution computes results in less than ten minutes (9 minutes and 24 seconds).

[2]Corpus refers to the collection of documents separated by language.

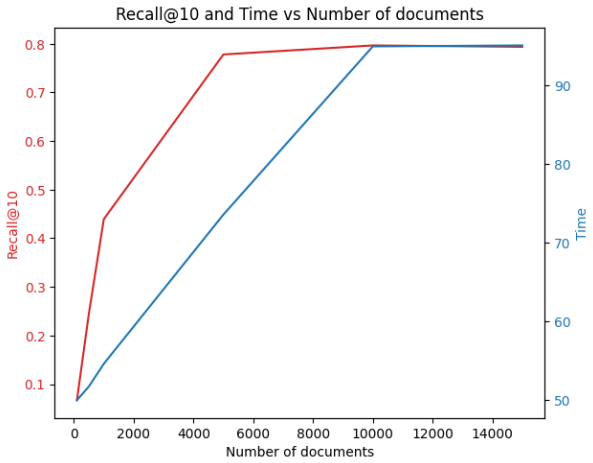


Fig. 2. Effect of filtering on execution time [s]

*1) Fine-tuning $k_1$ and b:* Finally, the last step of the process involves fine-tuning the two parameters $k_1$ and $b$. This fine-tuning was performed using a grid search. Although this method may not identify the absolute optimal combination, the relatively close values of recall indicate that the best solution is likely near the optimum. By inspecting the matrix presented in 3, the optimal parameters are found to be $k_1 = 2$ and $b = 0.9$.

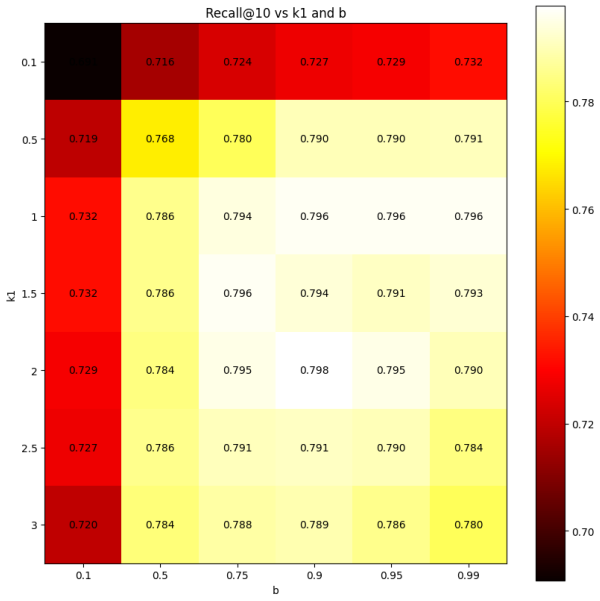


Fig. 3. Scores for the different combinations of $k_1$ and b

Finally, the hyperparameters that lead to best solutions are:

- $k_{en} = 10e4$: number of relevant documents for english
- $k_{others} = 5e3$: number of relevant documents for others
- $k_1 = 2$: term frequency scaling constant
- $b = 0.9$: degree of length normalization constant

This solution provides a recall@10 score of 0.80816 on Kaggle.

## VII. Further Work

Despite the good results obtained with the proposed method, there is still room for improvement. This section presents several ideas for

enhancing the results. First, the tokenization process could be refined by applying advanced techniques, such as handling compound words. For instance, the phrase ”machine learning” may be tokenized as ”machine” and ”learning,” which can have very different meanings and affect the results. Additionally, addressing synonyms could also enhance performance by treating words like ”car” and ”automobile” as equivalent.

Finally, exploring other retrieval methods that may be more efficient than BM25 could yield further improvements.

## REFERENCES


[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL: https://arxiv.org/abs/1301.3781, arXiv:1301.3781.

[2] Edward Loper Steven Bird and Ewan Klein. Natural language processing with python, 2009. https://www.nltk.org/.

[3] Stopwords-ISO. Stopwords for korean (stopwords-ko). https://github.com/stopwords-iso/stopwords-ko, 2024. Accessed: 2024-11-02.