Esteban NERAUDAU January 2023

Text Mining Project Report

Comparing TF-IDF and BM25 method on query based document dataset

Linköping universitet – TDDE16

Contents

1. Introduction

1.1 Context & goal

For this project, I wanted to rely on my passion for Text Mining.

At first, I had to think about a field that interested me and in which I could find data easily. I thought about working on data from a twitch chat, maybe analyzing viewers' emotions. Unfortunately, my research was not conclusive because almost all the topics have been covered (at least all the topics that are affordable for my level of knowledge).

So I changed my strategy and started my research again this time based on what I had the level to do and not what I was interested in. Following this idea, I found the site kaggle which offers a lot of data that we can use as we wish.

The main idea of this project is to implement a search engine to search a set of documents. To do so, we'll use a dataset from CISI (Centre for Inventions and Scientific Information) and two basic approaches TF-IDF and BM25. To evaluate and compare these two approaches, the performances will be measured, using standard metrics: Precision@10, Recall@10, and F-score@10.

1.2 Theory

I chose to work with 2 methods. The first is TF-IDF and the second one is BM25. I chose those 2 methods because BM25 and TF-IDF are two commonly used techniques for information retrieval and text mining, both are designed to rank a set of documents based on a query which is exactly our case. My idea is to compare the performance of these two methods.

We saw TF-IDF during the course so I won't talk about it here. On the other hand, BM25 (Best Match 25) is a ranking function that is commonly used in information retrieval and text mining to rank a set of documents based on a query. It is designed to improve upon the performance of the traditional TF-IDF approach. The main advantage of BM25 over the traditional TF-IDF approach is that it takes into account the length of the document, which can have a big impact on the relevance of a document to a query.

1.3 Data

The data was collected by the Centre for Inventions and Scientific Information ("CISI") and is made up of text data from about 1,460 documents and 112 related queries. Its purpose is to be used to create information retrieval models, where a given query will return a list of document IDs relevant to the query. Here's the Kaggle page I found the dataset.

2. Explanation

2.1 Method

2.1.1 Execution

To run our program, it is necessary to install the **nltk** package with the command :

pip install nltk

Then, you have to execute the file "main.py" which will launch the program.

In the file "Preprocessing.py", the nltk package will automatically download the packages "stopwords", "punkt", "wordnet" and "omw-1.4" during the first execution. For the following executions, you can comment out lines 4 to 8 included.

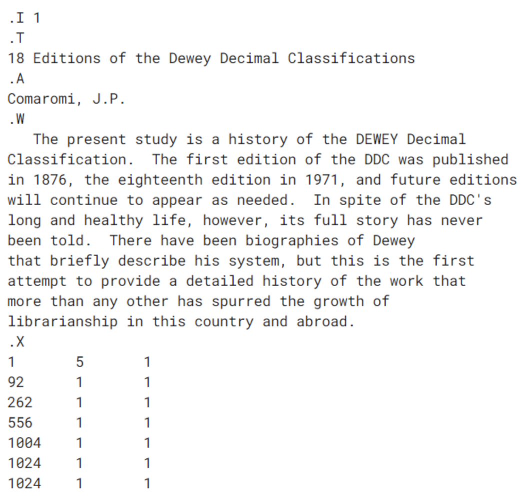
2.1.2 Project architecture

Le projet est composé des fichiers suivants :

- Main.py : calls the functions of each script and runs the main program
- Parsing.py : takes care of parsing the data files
- Preprocessing.py : takes care of processing the data such as stopwords removal and lemmatization
- Compute.py : contains the functions describing the algorithms for the calculation of the vocabulary, the TF/IDF and BM25
- Helper.py : allows to display of the results of the queries according to the algorithm used

I have modified the CISI.QRY file to format it so that the queries from index 58 onwards only have one tag ".I" and ".W" each. It is therefore important to use my own data file (you'll find it following the GitHub link) and not the data file you can find on the website.

2.1.3 Preprocessing

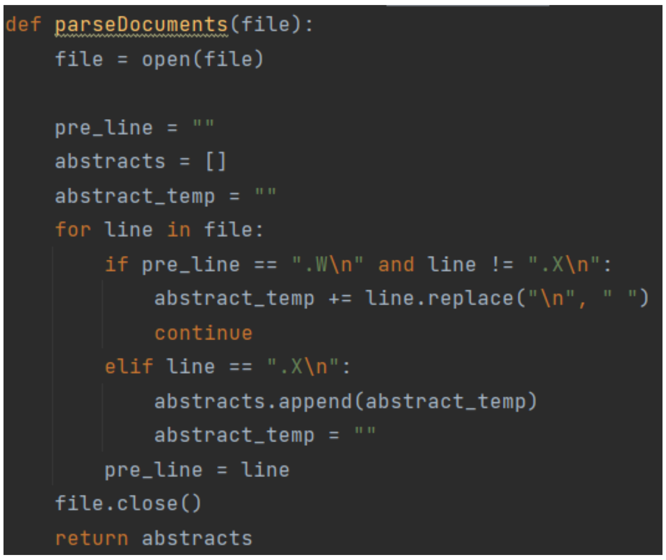


```
.I 1
.T
18 Editions of the Dewey Decimal Classifications
.A
Comaromi, J.P.
.W
   The present study is a history of the DEWEY Decimal
Classification.  The first edition of the DDC was published
in 1876, the eighteenth edition in 1971, and future editions
will continue to appear as needed.  In spite of the DDC's
long and healthy life, however, its full story has never
been told.  There have been biographies of Dewey
that briefly describe his system, but this is the first
attempt to provide a detailed history of the work that
more than any other has spurred the growth of
librarianship in this country and abroad.
.X
1       5       1
92      1       1
262     1       1
556     1       1
1004    1       1
1024    1       1
1024    1       1
```

Here is the general structure of the data, with irregularities, indeed some data have several author fields, or not at all.

The data is not totally homogeneous and therefore requires pre-processing.

Parsing



```python
def parseDocuments(file):
    file = open(file)

    pre_line = ""
    abstracts = []
    abstract_temp = ""
    for line in file:
        if pre_line == ".W\n" and line != ".X\n":
            abstract_temp += line.replace("\n", " ")
            continue
        elif line == ".X\n":
            abstracts.append(abstract_temp)
            abstract_temp = ""
        pre_line = line
    file.close()
    return abstracts
```

We only want to retrieve the abstracts of the documents. We have assumed that we only keep the correctly constituted data as in the example above.

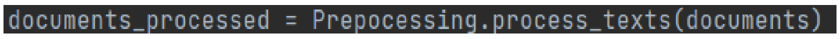
The parseDocuments function relies on tag detection with pre_line, we look at the tag we just passed and according to it, if there was :

- ".W": then the following is an abstract. An abstract is on several lines, so we need to accumulate them in a buffer before adding them

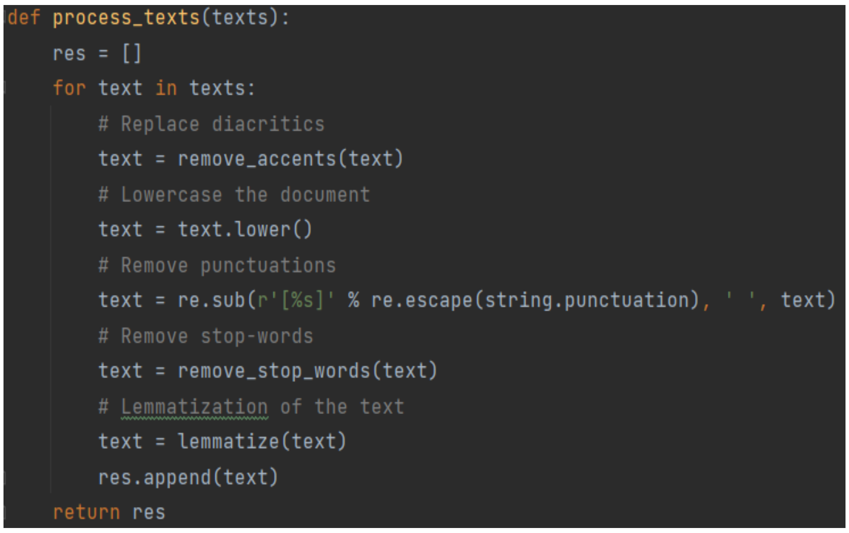
- We are only interested in abstracts so we ignore the other tags.

Then we have another function to parse the queries and the results.

Vocabulary



```
documents_processed = Prepocessing.process_texts(documents)
```



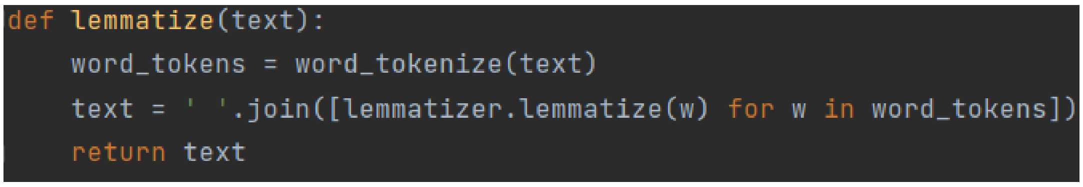
```
def process_texts(texts):
    res = []
    for text in texts:
        # Replace diacritics
        text = remove_accents(text)
        # Lowercase the document
        text = text.lower()
        # Remove punctuations
        text = re.sub(r'[%s]' % re.escape(string.punctuation), ' ', text)
        # Remove stop-words
        text = remove_stop_words(text)
        # Lemmatization of the text
        text = lemmatize(text)
        res.append(text)
    return res
```

Once the text is ready we can build the vocabulary

The first step is Tokenization, we transform the text into a list of individual words, remove the duplicates, and stop words from all the words that are so common that it is useless to index in a search.

Then comes the lemmatization step to have better results and correspondence with a common lexical entry.

Lemmatization consists in extracting the basic form of a word. Lemmatization, like tokenization, is a fundamental step in every NLP operation. Given the different existing linguistic structures, lemmatization is different in each language.



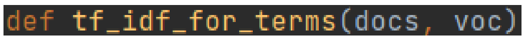
```
def lemmatize(text):
    word_tokens = word_tokenize(text)
    text = ' '.join([lemmatizer.lemmatize(w) for w in word_tokens])
    return text
```

2.2 Computing

2.2.1 TF-IDF

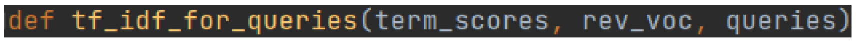
We will browse all the words in the vocabulary, and for each word, we will then browse each document to calculate their IDF and TF that we will store in dictionaries to facilitate their access.

We have two different functions to calculate TF / IDF. The first one allows us to compute the TF IDF of each term of the vocabulary.



```
def tf_idf_for_terms(docs, voc)
```

The second one allows the computation of the TF IDF of each query by adding the TF/IDF of the terms that compose the query.

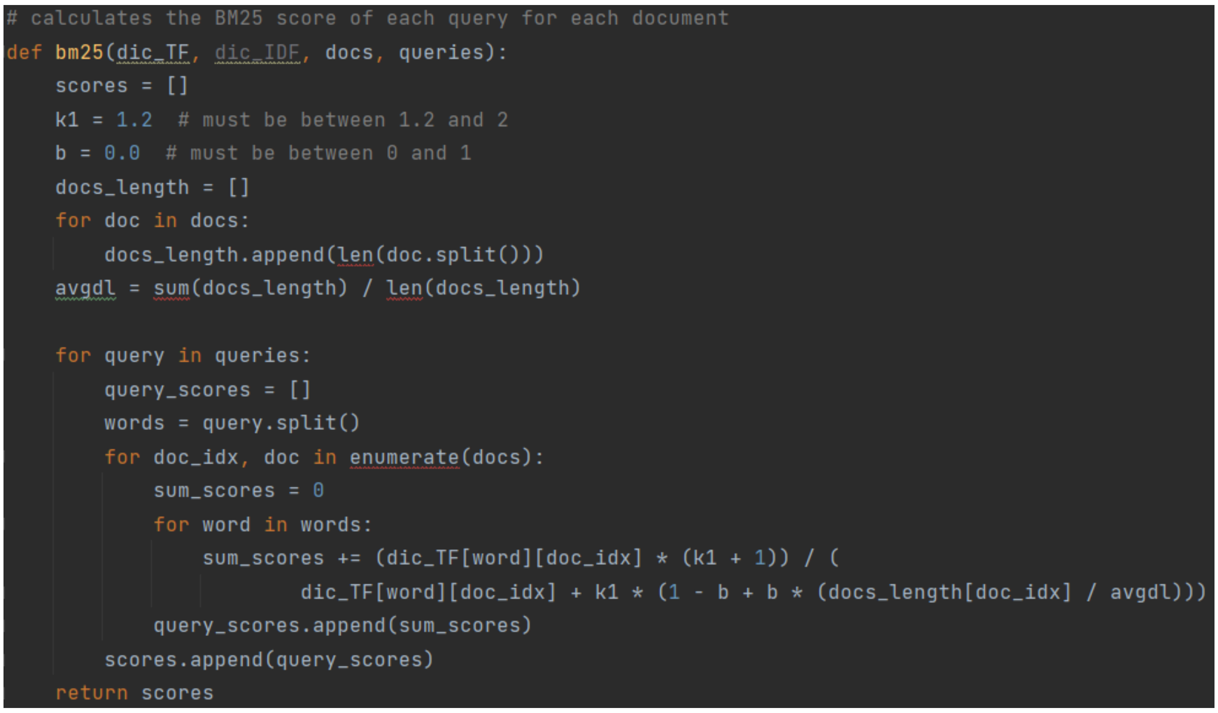


```
def tf_idf_for_queries(term_scores, rev_voc, queries)
```

The output is a query_scores(Q, D) matrix where "Q" is the index of a query and "D" is the index of a document.

2.2.2 BM25

The other method we'll use to calculate the score is BM25: it's a weighting method that uses a probabilistic model of relevance.



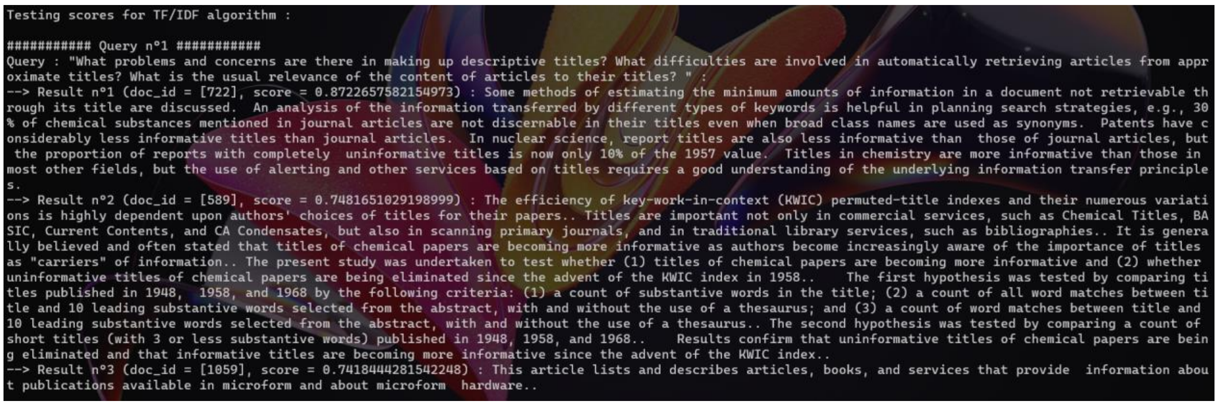
```
# calculates the BM25 score of each query for each document
def bm25(dic_TF, dic_IDF, docs, queries):
    scores = []
    k1 = 1.2  # must be between 1.2 and 2
    b = 0.0  # must be between 0 and 1
    docs_length = []
    for doc in docs:
        docs_length.append(len(doc.split()))
    avgdl = sum(docs_length) / len(docs_length)

    for query in queries:
        query_scores = []
        words = query.split()
        for doc_idx, doc in enumerate(docs):
            sum_scores = 0
            for word in words:
                sum_scores += (dic_TF[word][doc_idx] * (k1 + 1)) / (
                        dic_TF[word][doc_idx] + k1 * (1 - b + b * (docs_length[doc_idx] / avgdl)))
            query_scores.append(sum_scores)
        scores.append(query_scores)
    return scores
```

As for the calculation of the TF/IDF, we obtain in output a matrix scores(Q, D) where "Q" corresponds to the index of a request and "D" to the index of a document.

2.3 Results

Executing the main.py file we get some results. Here's an example of the result we get :



```
Testing scores for TF/IDF algorithm :

########### Query n°1 ###########
Query : "What problems and concerns are there in making up descriptive titles? What difficulties are involved in automatically retrieving articles from appr
oximate titles? What is the usual relevance of the content of articles to their titles? " :
--> Result n°1 (doc_id = [722], score = 0.8722657582154973) : Some methods of estimating the minimum amounts of information in a document not retrievable th
rough its title are discussed.  An analysis of the information transferred by different types of keywords is helpful in planning search strategies, e.g., 30
% of chemical substances mentioned in journal articles are not discernable in their titles even when broad class names are used as synonyms.  Patents have c
onsiderably less informative titles than journal articles.  In nuclear science, report titles are also less informative than  those of journal articles, but
 the proportion of reports with completely  uninformative titles is now only 10% of the 1957 value.  Titles in chemistry are more informative than those in
most other fields, but the use of alerting and other services based on titles requires a good understanding of the underlying information transfer principle
s.
--> Result n°2 (doc_id = [589], score = 0.7481651029198999) : The efficiency of key-work-in-context (KWIC) permuted-title indexes and their numerous variati
ons is highly dependent upon authors' choices of titles for their papers.. Titles are important not only in commercial services, such as Chemical Titles, BA
SIC, Current Contents, and CA Condensates, but also in scanning primary journals, and in traditional library services, such as bibliographies.. It is genera
lly believed and often stated that titles of chemical papers are becoming more informative as authors become increasingly aware of the importance of titles
as "carriers" of information.. The present study was undertaken to test whether (1) titles of chemical papers are becoming more informative and (2) whether
uninformative titles of chemical papers are being eliminated since the advent of the KWIC index in 1958..    The first hypothesis was tested by comparing ti
tles published in 1948,  1958, and 1968 by the following criteria: (1) a count of substantive words in the title; (2) a count of all word matches between ti
tle and 10 leading substantive words selected from the abstract, with and without the use of a thesaurus; and (3) a count of word matches between title and
10 leading substantive words selected from the abstract, with and without the use of a thesaurus.. The second hypothesis was tested by comparing a count of
short titles (with 3 or less substantive words) published in 1948, 1958, and 1968..    Results confirm that uninformative titles of chemical papers are bein
g eliminated and that informative titles are becoming more informative since the advent of the KWIC index..
--> Result n°3 (doc_id = [1059], score = 0.7418444281542248) : This article lists and describes articles, books, and services that provide  information abou
t publications available in microform and about microform  hardware..
```

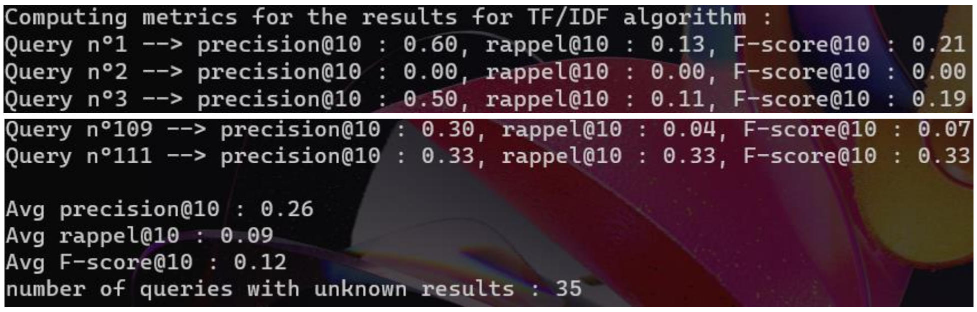
For each query, we calculate the proximity and return the first 3.

3. Discussion

I used 3 metrics to compare these 2 methods :

- Precision is the number of relevant documents retrieved relative to the total number of documents proposed for a given query.
- Recall is defined as the number of relevant documents retrieved relative to the number of relevant documents in the entire database.
- The F-score is a measure that is computed from precision and recall. The highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if precision or recall is zero.

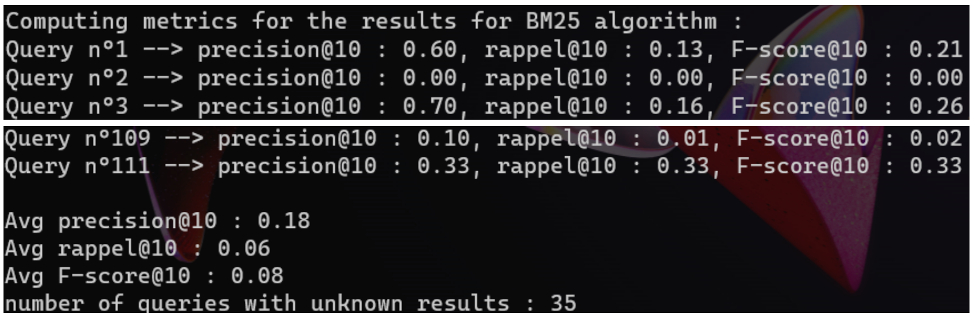
For TF/IDF :



```
Computing metrics for the results for TF/IDF algorithm :
Query n°1 --> precision@10 : 0.60, rappel@10 : 0.13, F-score@10 : 0.21
Query n°2 --> precision@10 : 0.00, rappel@10 : 0.00, F-score@10 : 0.00
Query n°3 --> precision@10 : 0.50, rappel@10 : 0.11, F-score@10 : 0.19
Query n°109 --> precision@10 : 0.30, rappel@10 : 0.04, F-score@10 : 0.07
Query n°111 --> precision@10 : 0.33, rappel@10 : 0.33, F-score@10 : 0.33

Avg precision@10 : 0.26
Avg rappel@10 : 0.09
Avg F-score@10 : 0.12
number of queries with unknown results : 35
```

For BM25 :



```
Computing metrics for the results for BM25 algorithm :
Query n°1 --> precision@10 : 0.60, rappel@10 : 0.13, F-score@10 : 0.21
Query n°2 --> precision@10 : 0.00, rappel@10 : 0.00, F-score@10 : 0.00
Query n°3 --> precision@10 : 0.70, rappel@10 : 0.16, F-score@10 : 0.26
Query n°109 --> precision@10 : 0.10, rappel@10 : 0.01, F-score@10 : 0.02
Query n°111 --> precision@10 : 0.33, rappel@10 : 0.33, F-score@10 : 0.33

Avg precision@10 : 0.18
Avg rappel@10 : 0.06
Avg F-score@10 : 0.08
number of queries with unknown results : 35
```

Comparison

	TF/IDF	BM25
Average precision@10	0.26	0.18
Average rappel@10	0.09	0.06
Average F-score@10	0.12	0.08

From the results, we notice that TF/IDF has better precision, recall, and F-Score results than the BM25 method.

TF-IDF and BM25 are used to rank a set of documents based on a query, TF-IDF is a statistical measure that assigns importance to individual words in a document, while BM25 is a ranking function that takes into account the length of the document and has adjustable parameters that can be fine-tuned for a specific task.

BM25 takes into account the length of the document, which can have a big impact on the relevance of a document to a query. BM25 also allows for the use of free parameters (k1 and b), which can be fine-tuned to optimize the performance of the ranking function for a specific task.

The clear limitation of my work is my results which are not that good regarding performance. Maybe it's my dataset that is not that good, or my method which is not optimized but when I compare my results to the result of this study on Term Weighting for Feature Extraction on Twitter.

4. Conclusion

In a conclusion, we can say that we used the CISI dataset to build a model of information retrieval where a given query will return a list of documents. To do so we used and compared two différents techniques: TF/IDF and BM25.

BM25 is more than a term scoring method, but rather a method for scoring documents against a query. Tf-idf is a term scoring method, which can be integrated into a document scoring method using a similarity measure (e.g. cosine). The theoretical advantage of the BM25 method is that it offers a better probabilistic interpretation. The practical advantage of the BM25 method is that it gives better relevance for short documents.

As a personal conclusion, I struggled all the course and it’s not the subject that I love, I know I only did basic things in this project but doing this project helped me to understand better some concepts.