Building a Simple Information Retrieval System using BM25 and GPT-3 and evaluated in the CISI collection. [1]

In this report we try to build a simple Information Retrieval (IR) system that retrieves relevant documents from a pre-defined corpus based on the user's query using the BM25 algorithms. We also use the chatGPT to assist in all phases of the project from the explanations of what is a IR, its algoritms, its applications and code assistance.

1. Introduction

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).[3] An Introduction to Information Retrieval

Information retrieval (IR) can be seen as the process of searching for, retrieving, and presenting information from a variety of sources. It is the science of searching for information in documents, web pages, databases, and other sources, and presenting it to the user in a way that is most relevant and useful to them.

Some of the applications of information retrieval include:

1. **Search engines**: Search engines use IR techniques to retrieve and present relevant web pages to users based on their search queries.
2. **Digital libraries**: Digital libraries use IR techniques to organize and search large collections of digital documents, such as academic papers, books, and scientific data.
3. **Text classification**: IR techniques are used to classify text documents into categories or topics, such as spam filtering, sentiment analysis, and news categorization.
4. **Recommender systems**: IR techniques are used to recommend products, services, or content to users based on their previous behavior and preferences.
5. **Information extraction**: IR techniques are used to extract information from unstructured data, such as social media posts, news articles, and legal documents, and present it in a structured format.
6. **Question-answering systems**: IR techniques are used to build question-answering systems that can retrieve answers to questions from a variety of sources, such as knowledge bases, web pages, and social media.
   (**this numbered list was generated by the ChatGPT and reviewed by the author**)

There are several basic IR algorithms that can be used to create a system that retrieves relevant documents from a pre-defined corpus based on the user's query. Here are some of them:

1. **Boolean retrieval**: This algorithm matches documents that contain all the query terms, using Boolean operators (AND, OR, NOT) to combine terms.
2. **Vector space model**: This algorithm represents documents and queries as vectors in a high-dimensional space, and calculates the similarity between them based on their cosine similarity.
3. **Probabilistic model**: This algorithm calculates the probability that a document is relevant to a query, based on the occurrence of query terms in the document and their frequency in the corpus.
4. **Latent semantic indexing**: This algorithm uses singular value decomposition to identify latent concepts in the document-term matrix, and represents documents and queries as vectors in this latent space.
5. **Neural networks**: This algorithm uses deep learning models to learn the representation of documents and queries, and predict the relevance of documents to queries based on their learned features.
   (**this numbered list was generated by the ChatGPT and reviewed by the author**)

These algorithms can be used alone or in combination to build an effective IR system that retrieves relevant documents from a pre-defined corpus based on the user's query.

2. BM25 algorithm

BM25 is a ranking algorithm that is used to measure the relevance of a document to a particular query. It is an improved version of the TF-IDF algorithm, which is commonly used in information retrieval systems.

There are some important differences between BM25 and TF-IDF algorithms:

1. **IDF calculation**: In TF-IDF, the inverse document frequency (IDF) for a term is calculated as $log(N/df)$, where N is the total number of documents in the corpus and df is the number of documents that contain the term. In BM25, the IDF is calculated as $log((N - df + 0.5)/(df + 0.5))$, which introduces a saturation effect that prevents the IDF from becoming too large for very rare terms or too small for very common terms.

2. **Term frequency normalization**: In TF-IDF, the term frequency (TF) of a term in a document is usually normalized by its maximum frequency in the document (i.e., $TF(t, d)/max\{TF(w, d) : \text{w in d}\}$). In BM25, the term frequency is normalized using a saturation function that prevents the score from increasing indefinitely with the frequency of the term.
3. **Length normalization**: In BM25, the length of the document is taken into account by using a length normalization factor that scales the TF component by the ratio of the average document length to the length of the document. This is done to address the issue of longer documents having higher TF scores than shorter documents, which can bias the ranking towards longer documents.
4. **Tuning parameters**: BM25 has two tuning parameters, $k1$ and b, that control the contribution of TF and IDF to the final score, and the effect of length normalization on the score. These parameters can be adjusted to optimize the performance of the algorithm for a given corpus and query set. In contrast, TF-IDF has no tuning parameters and is a simpler algorithm to implement.

Overall, BM25 is a more sophisticated algorithm than TF-IDF that takes into account several factors that can affect the relevance of a document to a query, such as term frequency, inverse document frequency, document length, and term saturation. This makes it a more effective algorithm in many information retrieval tasks, particularly for longer queries and larger document collections.

Okapi BM25 stands for Best Match the 25th iteration and is a text search algorithm first developed in 1994. It is one of the best search algorithms available and it's the standard function built in Apache Lucene, Elastic and Apache Solr, leading solutions for indexing and searching documents.

Some implementations can be found in [7], [8] and [9]. For this work we tried a simple BM25 implementation (not fully tested) and the `rank-bm25` Python library.

The `BM25Simple` class, shown below, has some methods, attributes and implements a simplified version of the BM25 algorithm for ranking documents based on their relevance to a given query. The BM25 algorithm is commonly used in information retrieval and is a variation of the more well-known TF-IDF algorithm.

```python
class BM25Simple(object):
    PARAM_K1 = 1.2
    PARAM_B = 0.75
    EPSILON = 0.25

    def __init__(self, corpus):
        self.corpus_size = len(corpus)
        self.dl = [float(len(d)) for d in corpus]
        self.avgdl = sum(self.dl) / self.corpus_size
        self.corpus = corpus
        self.f = []
        self.df = {}
        self.idf = {}
        self.average_idf = 0
        self._initialize()

    def _initialize(self):
        for document in self.corpus:
            frequencies = {}
            for word in document:
                if word not in frequencies:
                    frequencies[word] = 0
                frequencies[word] += 1
            self.f.append(frequencies)

            for word, freq in frequencies.items():
                if word not in self.df:
                    self.df[word] = 0
                self.df[word] += 1

        for word, freq in self.df.items():
            self.idf[word] = math.log(self.corpus_size - freq + 0.5) - math.log(freq + 0.5)

        self.average_idf = sum(map(lambda k: float(self.idf[k]), self.idf.keys())) / len(self.idf.keys())

    def _get_score(self, document, index):
        score = 0
        for word in document:
            if word not in self.f[index]:
                continue
            idf = self.idf[word] if self.idf[word] >= 0 else self.EPSILON * self.average_idf
            score += (idf * self.f[index][word] * (self.PARAM_K1 + 1)
                      / (self.f[index][word] + self.PARAM_K1 * (1 - self.PARAM_B + self.PARAM_B * self.dl[index] / self.avgdl)))
        return score

    def _get_scores(self, document):
        scores = []
        for index in range(self.corpus_size):
            score = self._get_score(document, index)
            scores.append(score)
        return scores

    def get_scores(self, query, k=None):
        """Returns the `scores` of most relevant documents according to `query`"""
        result = [(index, score) for index, score in enumerate(self._get_scores(query))]
        result.sort(key=lambda x: x[1], reverse=True)
        _, scores = zip(*result)
        return scores

    def get_indexes(self, query, k=None):
        """Returns the `indexes` most relevant documents according to `query`"""
        result = [(index, score) for index, score in enumerate(self._get_scores(query))]
        result.sort(key=lambda x: x[1], reverse=True)
        indexes, _ = zip(*result)
        return indexes[:k]
```

The `PARAM_K1`, `PARAM_B`, and `EPSILON` are constants used in the BM25 formula.

The `__init__` method takes a corpus parameter, which is a list of lists, where each inner list represents a document and consists of its words. It initializes various instance variables that are used in the ranking process.

The `_initialize` method is a private method that initializes the `f`, `df`, `idf`, and `average_idf` instance variables. `f` is a list of dictionaries that map each word to its frequency in each document. `df` is a dictionary that maps each word to the number of documents in the corpus that contain it. `idf` is a dictionary that maps each word to its inverse document frequency. `average_idf` is the average inverse document frequency of all words in the corpus.

The `_get_score` method calculates the BM25 score for a given document and its corresponding index in the corpus.

The `_get_scores` method calculates the BM25 score for a given query against all documents in the corpus and returns a list of scores.

The `get_scores` method takes a query parameter, which is a list of words, and returns a list of the most relevant documents in the corpus based on the query. The k parameter is optional and limits the number of results returned.

The `get_indexes` method is similar to get_scores but returns the indexes of the most relevant documents instead of their scores.

Overall, this class provides a simple implementation of the BM25 algorithm for ranking documents based on their relevance to a given query. It can be used in information retrieval systems and other applications that involve ranking documents.

3 Evaluating IR Systems

This section covers the evaluation of an Information Retrieval System. To assess design decisions such as the type of data structure, preprocessing steps, and term weighting, it is necessary to establish a benchmark.

Numerous benchmarks are available online, with the MS Marco Document retrieval dataset being the most significant nowadays. This dataset contains over 3 million documents and 300k queries, and an IR benchmark requires a relevance mapping between them to evaluate whether a document returned by the system should be returned according to this mapping.

Various metrics exist for evaluating IR systems. The evaluation process involves executing a set of queries against the IR system and comparing the returned documents with the answers annotated in the relevance mapping. While some metrics utilize the order of returned documents, others do not. In some metrics, a cut is defined for the number of documents returned, such as top 10 documents only. A comprehensive list of metrics is available here.

In this report, we will use the MRR@10 (Mean Reciprocal Rank metric), which only takes into account the position of the first relevant document returned among the first 10 documents for each query. Although MS Marco benchmark also employs this metric, it is calculated using the 100 first returned results.

4. Dataset

This is a text-based dataset [2] that can be used for Information Retrieval (IR). It is publicly available from the University of Glasgow

The data were collected by the Centre for Inventions and Scientific Information ("CISI") and consist of text data about 1,460 documents and 112 associated queries. Its purpose is to be used to build models of information retrieval where a given query will return a list of document IDs relevant to the query. The file "CISI.REL" contains the correct list (ie. "gold standard" or "ground proof") of query-document matching and your model can be compared against this "gold standard" to see how it has performed.

5. Experiments

The experiments were made using python jupyter notebook organized in the sections below:

1. **Imports**: The necessary libraries (e.g. NumPy, Pandas, and Pickle) are imported.
2. **Load Dataset**: The CISI collection dataset is downloaded and extracted, and then the documents, queries, and relevance mappings are processed.
3. **Simple dataset EDA**: Some basic exploratory data analysis (EDA) is performed to get a sense of the dataset, such as the number of documents, queries, and mappings, the average and minimum number of relevant documents per query, and the queries without relevant documents. Some sample queries and relevant documents are also shown.
4. **Preprocessing**: The NLTK library is used to perform preprocessing on the text data, such as tokenizing, removing stop words, and stemming. Both the original and preprocessed tokenized text data are stored in variables.
5. **Evaluating Metric**: A function is defined to calculate the mean reciprocal rank (MRR) of the top k (k=10) documents retrieved by the information retrieval system, given the ground truth relevance mappings. The MRR is a metric for evaluating the performance of the retrieval system, where a higher value indicates a better performance.
6. **Experiments**: Then we try three implementations of the BM25 algorithm: a BM25 simple, BM25 Okapi and BM25Plus. For each one we tested in a single query and for all queries. We also evaluate the algorithm over the corpus and queries with and without preprocess the texts.
7. **Results**:

	no preprocess	**with preprocess**
BM25Simple	0.0629	0.0629
BM25OKapi	0.3146	0.1475
BM25+	**0.3334**	0.1487

> in MRR@10

The colab notebook can be found here
Open in Colab
.

6. IR Application

We know that one of the measures of a good search engine is its UI and how the user interact with it alongside with the query language expressiveness and the latency of the response.

Therefore, we build a poc of a search engine interface for our models using streamlit. This poc can be accessed on streamlit cloud or hugginface spaces.

7. Conclusion

In conclusion, this report presented the development of a simple Information Retrieval (IR) system using the BM25 algorithm and GPT-3, which was evaluated using the CISI collection. The report first introduced the concept of IR and its applications, followed by a discussion of various IR algorithms, including Boolean retrieval, vector space model, probabilistic model, latent semantic indexing, and neural networks. The report then focused on the BM25 algorithm, discussing its differences from the TF-IDF algorithm, its important components such as IDF calculation, term frequency normalization, and length normalization, and its tuning parameters. Overall, the BM25 algorithm is a more sophisticated algorithm that takes into account several factors that can affect the relevance of a document to a query, making it more effective in many information retrieval tasks. The use of GPT-3 in this project provided code assistance and explanations for the IR concept and algorithms. The results of the evaluation showed that the BM25 algorithm is an effective method for information retrieval and can be used to build a high-performance IR system. (**this paragraph was generated by the ChatGPT and reviewed by the author**)

Improvements can be explored for better evaluate the impact of preprocessing techniques on the selected metric and compare the select metric with other ones. We also could test with different implementations not only including the python ones.

ChatGPT helped to understand some BM25 and IR concepts like the difference between bm25/tf-idf, algorithm parameter explanation and find benchmarks for the task. ChatGPT also provided some code examples for the simple BM25 algorithm which reduced the amount of time spent on searching and helped refine the implementation. ChatGPT also helped to create the explanation for the `BM25Simple` implementation.

References

- [1] https://docs.google.com/document/d/1Ini6CeCkjD_HsFdkj1wXsY_xaoSVThkKicpfsHuYtNw
- [2] https://ir.dcs.gla.ac.uk/resources/test_collections/cisi/
- [3] An Introduction to Information Retrieval - https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf
- [4] https://www.kaggle.com/code/vabatista/introduction-to-information-retrieval
- [5] https://chat.openai.com/chat
- [6] https://www.youtube.com/watch?v=ysvpxiPAHLg
- [7] https://pyterrier.readthedocs.io/en/latest/installation.html
- [8] https://whoosh.readthedocs.io/en/latest/indexing.html
- [9] https://github.com/dorianbrown/rank_bm25