# Efficient Vector Space Model

**Yin-Feng Li, Roman Salzwedel, Rahul Taneja**
Information Retrieval Project Course (IE681)
Faculty of Business Informatics and Mathematics
University of Mannheim
{yinli, rsalzwed, rtaneja}@mail.uni-mannheim.de

**Abstract**

The vector space model is one of the widely applied information retrieval (IR) models to rank the documents based on similarity values. The retrieval operations consist of cosine similarity function to compute the similarity score between a given query and the set of documents retrieved and then rank the documents according to the relevance. In this report, we are presenting different approaches of vector space model to compute similarity scores between a given set of queries and documents from the NFCorpus. Hereby, we investigate the trade-off between efficiency and performance often faced in IR. In order to achieve a higher efficiency, we test certain speed-up techniques, i.e., inverted index, posting merge, pre-clustering and tiered index. To evaluate the different vector space retrieval systems, we compare retrieval time, MAP and nDCG.

## 1 Introduction

The goal of this project is to implement an efficient vector space retrieval application for a given document-query collection. It is successfully built, using the data which is extracted and processed by the University of Heidelberg from **NutritionFacts.org** (NF)[1]. Besides the "vanilla" vector space retrieval model, we achieve efficiency improvements by various speed-up methods and evaluate their performance.

In general, the workflow of vector space models for information retrieval can be divided into three steps. The first step is to preprocess the information resource, that is, each document. Preprocessing includes e.g. parsing the document corpus, tokenizing the text, potentially stopwords removal and stemming.

Secondly, construct the vector representation for each document, that this project uses TF-IDF. To facilitate the computation of TF-IDF, an inverted index of the document corpus is usually constructed first. This inverted index structure is serialized for efficiency purpose.

The last step involves calculating the similarity between user-provided queries and each of the candidate documents. Queries are also preprocessed in the same fashion. We use cosine score as the similarity measure in this project.

At the later stages, we implement speed up methods, i.e., pre-clustering, posting merge and tiered index. The implementation mentioned in this report refers to "**own implementation**" as the requirement of this project. We have implemented most of the functions from scratch and only imported some very basic python packages, i.e., pandas, numpy, math etc[2]. Besides, our code is also published on Github[3], making it publicly available.

This report is structured as follows: Section 2 describes the dataset, its source and the overall number of documents and queries. Section 3 briefly describes the prepocessing steps we have applied. In Section 4 the vector space model and the similarity measure used in this project are explained. Section 5 discusses the speed-up methods, which we have implemented for the efficient retrieval. This is also the major focus of this project. In Section 6, several tables are presented, which provide an overview of the results for different speed-up methods and preprocessing strategies. We use retrieval time, MAP and nDCG as our evaluation metrics. Section 7 concludes this project by discussing how various speed-up methods effect the performance of the model.

[1]https://nutritionfacts.org/

[2]For the complete list of packages, please refer to the jupyter notebook

[3]https://github.com/rahultaneja111/IRProject

## 2 Dataset

There are three major contents on NF website - blogs, videos and Q&A posts, from which NFCorpus was extracted by (Boteva et al., 2016). NF-Corpus is a full-text English retrieval dataset for Medical Information Retrieval that contains two raw text corpora, nfDump and docDump. The former contains full-text queries annotated with automatically extracted relevance links at several levels from the NF website. The latter one contains abstracts of medical documents extracted from PubMed[4] and PMC[5] websites.

Boteva et al. assigned 80% of the corpus to be the training set and 10% to be the dev and test set each. We use the queries and documents of the test set to be compatible with Boteva et al., and to compare the evaluation results at the end.

In NFCorpus, queries are categorized by different combinations of the field on NF website. We use *titles of non-topic pages*, which is the title of NF pages, excluding topic pages[6]. The reason is that ordinary users are very likely to express their information needs in this way as well. (e.g., ”do vegetarians get enough protein ?” or ”is caffeinated tea really dehydrating ?” )

There are three possible values for the relevance, (2) refers to that, given a query **q**, a scientific document **d** is cited in the sources section on the page. (1) is used when there exists another query **q’** that links to **d** and **q’s** text contains a link to **q**. The lowest level, (0) is for marginally relevant **q’** and **d**.(Boteva et al., 2016)

In total, we have 3162 document in our collections as information resource, and 144 queries as information needs.

## 3 Preprocessing

Parsing the document collection is the first step. The next step is to extract meaningful information from the parsed document collection, that is, each document is tokenized with separators. We use two types of tokenizers based on different regular expressions. The simple tokenizer is with regular expression $[a-z\backslash-]+$. The advanced tokenizer uses a more sophisticated regular expression $(?u)\backslash\backslash b\backslash\backslash w\backslash\backslash w+\backslash\backslash b$. Then we remove stopwords and apply stemming methods to each token. The stopwords used are from a list of common words given by NFCorpus. In this project, we test Porter and Snowball stemmer as well as lemmatization. Stemming recognizes words with the same root as the same one, and can deal with morphological changes to some extents.

[4]https://www.ncbi.nlm.nih.gov/pubmed/

[5]https://www.ncbi.nlm.nih.gov/pmc/

[6]Topic pages lists tags of topics in each blog

## 4 Vector Space Model

The Vector Space Model (VSM) is an IR model that represents documents and queries as vectors in a common vector space. The VSM model builds on the bag of words (BOW) approach, in which there is one axis for each vocabulary term. Following the initial preprocessing of the text, the VSM model represents queries and documents as vectors, where the components of the vectors are TF-IDF weights of vocabulary terms[7]. The document vectors are represented in the term-document matrix. Each row of the matrix represents a term in the vocabulary, and each column represents a document in our collection. The value in a cell $M_{ij}$ represents the TF-IDF weight for term $t_i$ in document $d_j$.

In this project, the TF-IDF weighting scheme is applied as a measure of the relative importance of $t_i$ in document $d_j$. TF-IDF has two components. The term frequency $tf(t,d)$ (that is, local), measures how frequently the term t appears in the document d. In this project, we use the normalized term frequency. The definition is as below:

$$tf(t,d) = \frac{(1+\log_{10}(f_{t,d}))}{(1+\log_{10}(max\{f_{t',d} : t' \in d\}))} \tag{1}$$

The second component is the inverse document frequency (that is, global), that accounts for the fact that rare terms are more informative than frequent terms. Assuming that informativeness of term t is inversely proportional to the number of documents in the collection in which t appears in, IDF is defined as below:

$$idf(t,D) = \log_{10}\left(\frac{|D|}{|\{d' \in D : t \in d\}|}\right) \tag{2}$$

The TF-IDF weight of a term is computed as the product of the term frequency and the inverse document frequency.

[7]Although, theoretically alternative weighting schemes such as raw term-frequency or binary term-occurrence exist, TF-IDF is the best known weighting scheme in IR (Manning et al., 2008)

$$tfidf_{t,d} = tf(t_i, d_j) * idf(t_i) \tag{3}$$

### 4.1 Cosine Similarity & Retrieval

For document retrieval we view a query as a short document and represent it in the same vector space as our document collection $q = (w_{1q}, w_{2q}, ....w_{Nq})$ . Hence, the similarity between the query and a document is computed as the distance between the two vectors. In IR, the most popular similarity measure to compute vector distances is cosine similarity[8]. Thus, in this project we rely on cosine similarity. The similarity between query x and document y is defined as:

$$cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||} \tag{4}$$

Note, that in the naive or vanilla approach using our term-document matrix and the notion of cosine similarity as defined above we traverse through the whole document collection and compute the scores of one document at a time (this is known as document-at-a-time scoring). In the accompanying code, the cosine computation is integrated into our top K function. With this function we rank the documents in the collection by decreasing score and return only the top K documents with the highest similarity to the user. The top K function is used with all VSM schemes presented in this paper. We choose a value of K = 10 to mirror the behavior of many popular search engines on the Web.

## 5 Speed-up Methods

The ”vanilla” approach outlined above requires to compute the cosine between the query vector and all document vectors, to sort the resulting scores, and to select the top K documents. Hereby, even a single similarity computation can demand thousands or tens of thousands of arithmetic operations depending on the size of the vocabulary (Manning et al., 2008). Obviously, this is computationally very expensive and infeasible for most real-time querying scenarios.

Thus, we now turn to methods of efficient VSM retrieval that try to dramatically lower the cost of finding the top K documents. In general, all of the methods follow a common idea. They aim to reduce the computation cost by selecting a subset S of the total documents N for which we compute similarity scores. Theoretically, it holds that $K < |S| << N$. While efficient vector space models reduce the computational effort, it is not guaranteed that the set S contains the true K top-scoring documents for a given query. That is why, in efficient VSM retrieval we often face the trade-off between efficiency and performance.

The following sections will further investigate this trade-off by considering four schemes of efficient vector space retrieval: InvertedIndex, PostingMerge, Preclustering, and TieredIndex. The subsections will briefly introduce the theoretical underpinnings of each method. The evaluation of all models in terms of efficiency (i.e. retrieval time) and performance (MAP and nDCG) is shown in the tables in section 6.

### 5.1 Inverted Index

The first approach we use to speed up the retrieval is the implementation of an inverted index in combination with index-based elimination. In contrast to the term-document matrix, the inverted-index is a data structure that stores for each term t in the vocabulary the posting list. A posting list is a sorted list of all documents that contain t with the respective TF-IDF weight of term t in document d. The inverted index is stored as a dictionary of the format:

$\{"term" : [(docID1, tf - idf), (docID2, tf - idf), \ldots]\}$

This requires much less storage and allows for computationally more efficient retrieval (see Table 1).

Furthermore, for retrieval we traverse over each query term at a time (term-at-a-time scoring) and fetch only the documents that contain at least one of the query terms. Documents that do not contain any of the query terms will have a cosine similarity of 0. This significantly reduces the number of arithmetic operations compared to the vanilla approach. In addition, in contrast to Formula (4) we normalize the dot product only with the length of the document vector [9] The performance of this approach is shown in Table 2 and 3.

[8]Compared to Euclidean distance, the advantage of cosine similarity is that it is not affected by the length of the input vectors.

[9]This approach follows the basic algorithm for computing vector space scores given by (Manning et al., 2008). Since the length of the query vector is a constant, it will not effect the resulting ranking of the documents.

### 5.2 Posting Merge(Intersection)

This algorithm is efficient for merging of conjunctive queries with multiple terms. In this speed up method, we take the intersection of the documents appearing for each query term. This reduces the number of documents which are used for cosine similarity computation, and hence reducing the retrieval time. This method performs well in terms of efficiency, but it performs badly in terms of the evaluation metrics such as MAP and nDCG as it retrieves only documents which contain all the terms of the query (See table 1, 2 and 3). This retrieval method might be better suited for Boolean queries.

### 5.3 Pre-clustering

Pre-clustering is implemented in the way that, firstly, we randomly choose square root of N documents as cluster leaders. Secondly, each document, which is not selected as a leader in the first step, is treated as a query and fired against the leader collection. Hereby, we use the same TF-IDF approach and cosine computation as defined in Section 5.1. Lastly, we assign each document to the cluster of the leader with the highest similarity score.

For retrieval, we fire each query against the leader collection. Then, we compute the similarity between the query and the documents in the cluster of the cluster leader with the highest similarity score. If there are not sufficient results, we then look into the cluster whose leader has the second highest similarity with the query and repeat until we get a sufficient number of relevant documents that is required by the user[10].

### 5.4 Tiered Index

Tiered index is a concept in which the postings in the inverted index are broken down hierarchically into several lists. Tiers are ordered in decreasing order of importance. For a term $t_i$, break down of documents is usually done according to the TF-IDF scores. We first look into the top tier, i.e., merge the term postings of the first tier. If the merges over the top-tier postings result in too few hits, we continue to merge lists of the lower tiers. In our project, we create a two-level tiered index from the inverted index. We divide the posting list for each term into two sub-lists(i.e. tiers) based on the TF-IDF score. To calculate the cosine similarity score, we first look into the first tier and retrieve the documents. If there are few hits, we look into the second tier and repeat. Tiered index is an efficient and fast way of retrieving the documents based on the query.

[10]For the results presented in this paper we define the minimum number of retrieved documents to be at least 30. This value is the same for the Tiered Index introduced below.

## 6 Evaluation

There are different aspects through which we can evaluate IR systems such as retrieval effectiveness and retrieval time. To measure the evaluation of our different IR systems with different speed-up methods, we use different evaluation methods such as Mean Average Precision (MAP), and normalized Discounted Cumulative Gain (nDCG). Also, we measure the time the system takes for the retrieval. We use the same queries and documents that were assigned to test set in the original paper (Boteva et al., 2016). Recall, that our document collection is of size 3,612 with 144 queries for which we have complete relevant judgments. The retrieval speed is estimated on an Ubuntu Machine with 8 GB RAM, Intel Core i5-4200U CPU @ 1.60GHz, and with GeForce GT 730M 2 GB graphics card. We run 1440 iterations (i.e. 10 iteration over the whole set of queries) to measure the average retrieval time. To measure the MAP and nDCG we iterate once over the 144 queries in our test collection.

Table 1 and 2 show the results with advanced tokenizer and PorterStemmer used in preprocessing.

| **Method** | **time in sec** |
|---|---|
| vanilla | 0.4852 |
| invertedIndex | 0.0029 |
| posting Merge(Intersection) | 0.0025 |
| pre-Clustering | 0.0288 |
| tieredIndex(t =0.5) | 0.0028 |
| tieredIndex(t =0.8) | 0.0027 |

Table 1: Average retrieval time in seconds with Advanced Tokenizer and Porter Stemmer

Table 3 shows the results with simple tokenizer, no stemming, no lemmatization used in preprocessing. We see that posting Merge Intersection performs best in terms of time but worst in terms of MAP and nDCG metrics. Simple Vanilla model performs the best in terms of MAP and nDCG but is the slowest retrieval method. To make a com-

| Method | MAP | nDCG |
|---|---|---|
| vanilla | 0.1447 | 0.4749 |
| invertedIndex | 0.1447 | 0.4749 |
| posting Merge(Intersection) | 0.0329 | 0.2993 |
| pre-Clustering | 0.0549 | 0.3397 |
| tieredIndex(t =0.5) | 0.1432 | 0.4711 |
| tieredIndex(t =0.8) | 0.1369 | 0.4626 |

Table 2: Results with Advanced Tokenizer and Porter Stemmer

promise, inverted index and tiered index are the best performing retrieval methods. Taking the results in the paper by Boteva et al. as the baseline for our results, we see an improvement in MAP and nDCG (Boteva et al. report a MAP of 9.72, nDCG of 28.51 percent for titles of non-topic pages).

| Method | MAP | nDCG |
|---|---|---|
| vanilla | 0.1264 | 0.4513 |
| invertedIndex | 0.1264 | 0.4513 |
| posting Merge(Intersection) | 0.0286 | 0.2916 |
| pre-Clustering | 0.0536 | 0.3401 |
| tieredIndex(t =0.5) | 0.126 | 0.4489 |
| tieredIndex(t =0.8) | 0.1221 | 0.4412 |

Table 3: Results with Simple Tokenizer, No Stemming and No Lemmatization

## 7 Conclusion

In this project we implemented a vector space model for IR on a small medical document collection. Hereby, we compared several schemes for efficient VSM retrieval to investigate the trade-off between efficiency and performance often faced in IR. We tested all schemes in terms of retrieval speed, as well as performance in MAP and nDCG. As shown in section 6, we find that the basic implementation of inverted index as well as the speed-up using a two-level tiered index provide the best results considering both efficiency and performance. In terms of retrieval speed both methods are ca. 190 times faster than the naive vanilla approach. According to MAP and nDCG, the inverted index achieves the exact same performance as the vanilla approach. The tiered index is only slightly worse (depending on the threshold 0.1 to 0.7 percentage points in MAP) while offering another slight increase in retrieval speed. Given that in this project we deal with a small document collection (N = 3612), the impact of these marginal differences will become more profoundly visible in real-world scenarios dealing with larger collections of millions of documents. Regarding the other speed-up methods we find that posting merge intersection offers the fastest document retrieval, but provides very poor performance. It might be more suited for Boolean retrieval than free-text queries. Finally, the results of pre-clustering are not exactly impressive. It is ca. 10 times slower compared to the basic inverted index approach and scores a MAP of just about 5.5 percent. This might indicate that during clustering too many relevant documents end up in clusters that are later not considered in the document retrieval phase. Last but not least, we find that our results for VSM retrieval outperform the results presented in the original paper (Boteva et al., 2016). This is most likely due to differences in document preprocessing. As we have shown in the previous section we find that tokenization and more importantly stemming substantially improve the performance of our VSM systems.

Overall, this project was a beneficial learning experience. It fostered our understanding of vector space retrieval systems and provided valuable hands-on experience. A further improvement for the future will be the implementation of random projections as another method for efficient VSM retrieval.

## References

Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. 2016. A full-text learning to rank dataset for medical information retrieval. In *Proceedings of the European Conference on Information Retrieval (ECIR)*. Springer.

Christopher D. Manning, Prabhakar Raghavan, and Heinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.