The Performance of Boolean Retrieval and Vector Space Model in Textual Information Retrieval

Budi Yulianto¹, Widodo Budiharto², Iman H. Kartowisastro³
Doctor of Computer Science, Bina Nusantara University, Jakarta 11480, Indonesia
Emails: ¹laboratory@binus.ac.id, ²wbudiharto@binus.edu, ³imanhk@binus.edu

Abstract—Boolean Retrieval (BR) and Vector Space Model (VSM) are very popular methods in information retrieval for creating an inverted index and querying terms. BR method searches the exact results of the textual information retrieval without ranking the results. VSM method searches and ranks the results. This study empirically compares the two methods. The research utilizes a sample of the corpus data obtained from Reuters. The experimental results show that the required times to produce an inverted index by the two methods are nearly the same. However, a difference exists on the querying index. The results also show that the number of generated indexes, the sizes of the generated files, and the duration of reading and searching an index are proportional with the file number in the corpus and the file size.

Index Terms—Boolean Retrieval, Vector Space Model, Information Retrieval, Inverted Index, Querying Index, Corpus

I. INTRODUCTION

INTERNET users usually use the World Wide Web to retrieve data/information from current large-scale sources [1]. Unfortunately, the presented information are sometimes less relevant. In the field of information retrieval, the users expect to obtain very accurate results. The facts are that several existing approaches may provide less accurate queries [2].

Many researchers of information retrieval use Boolean Retrieval (BR) [3] and Vector Space Model (VSM) [4] in creating an inverted index and querying terms. The previous studies have done a search engine for collection of English or Arabic and shown the BR and VSM methods were optimal [3, 4]. Not only for text searching, information retrieval also can query multimedia elements, such as pictures [5, 6], or sounds [7, 8]. Information retrieval methods can be optimized with another algorithm such as Genetic Algorithm (GA) [3, 4, 9], or Particle Swarm Optimization (PSO) [2, 10].

Revised: Apr. 2, 2017; received in revised form: July 9, 2017; accepted: July 9, 2017; available online: July 11, 2017.

Due to the popularity of BR and VSM methods, there is a need to understand the relative performance between the two methods. The previous work had compared the Naïve Bayes (NB) method and the SVM method and found that the former method was better for the case of an external knowledge base [12]. The NB method correctly classified 79.44% of instances compared to SVM method [13]. Reference [4] found that SVM method with GA was better than SVM only for similarity measure. The SVM method with Finite State Transducer was better than SVM with Latent Semantic Analysis for automatic speech recognition [8]. Some comparisons by using Arabic data collection

| TABLE I
| ADVANTAGES AND DISADVANTAGES OF THE BR AND VSM METHODS. |
|-----------------|-----------------|-----------------|
| BR              | VSM             |
| Advantages      | Faster search   | Ranked document |
|                 | Easy to understand and implement | Easy to implement; BR is easier |
| Disadvantages   | Unranked document | Hard to understand Slower search |

Received: Apr. 2, 2017; revised in revised form: July 9, 2017; accepted: July 9, 2017; available online: July 11, 2017.

The BR method searches the exact results [11]. This method does not rank the number of terms appearing in a document because it only finds the term whether it exists or not (boolean) in documents. The result of the search method is a list of documents containing the unranked terms. In the other side, the VSM method searches the exact results with ranking [11]. This method ranks the number of terms appearing in a document, and count how many documents contain that term. The calculation process uses vector and the search results in the form of an ordered list. However, it is possible for two search engine methods to retrieve highly different documents, or to rank similar documents in a very different order [8]. The advantages and disadvantages of BR and VSM methods are described on Table I [11].
Cite this article as: B. Yulianto, W. Budiharto, and I. H. Kartowisastro, “The Performance of Boolean Retrieval and Vector Space Model in Textual Information Retrieval”, CommIT (Communication & Information Technology) Journal 11(1), 33–39, 2017.

showed that BR with GA was better than BR only [3], BR with adaptive GA was better (55.1%) than BR with traditional GA, and SVM with adaptive GA was better (42.1%) than SVM with traditional GA [9].

In this study, the researchers compare the performance of BR and VSM methods in textual information retrieval by applying a search engine application written in Python. The application is needed to do some experiment such as to get keywords input from user and then display the search results in the form of documents ID and name. Along this study, researchers also explain the steps of experiments in simple and clear ways. Researchers will use tables, graphics, arithmetic equations, and representative sourcecode to make clearer explanation for readers or other researchers in re-testing this experiment for studying, teaching, validation, or further works. At the end of this research, the performance of both methods is compared in any fields to be concluded.

Some terminologies used in this study are explained in the following on the basis of Ref. [14]. Corpus is a collection of documents, for example the articles on Wikipedia. Some examples of the popular corpus are Gutenberg, Brown, and Reuters. Term is a unique word contained in a document. Term derived from a tokenize process of a document. Generally, the document has been cleared from stop-words to obtain more specific term, and then do stemming to obtain clear terms of affixes. Tokenize is the process of converting a sentence into words (terms). Generally, the results of tokenize are stored in an array, set, or list. Sentence of “Mr. Widodo is a professor who teaches the course Information Retrieval” is tokenized to terms of “Mr.”, “Widodo”, “is”, “a”, “professor”, “who”, “teaches”, “the”, “course”, “Information”, “Retrieval”. Stop-word is common word that is less meaningful for the search process, such as “the”, “a”, “an”, “with”, and others. In the tokenize example above, the clearance of stop-words will become (“Widodo”, “professor”, “teaches”, “course”, “Information”, “Retrieval”). The words “Mr.”, “is”, “a”, “who”, and “the” are discarded because just as conjunctions are less meaningful in the search process. Stemming is the process of simplifying the word to its basic word by removing the suffix. For example, the word “teaches” becomes “teach”. Some often used stemming method are Porter Stemmer and Lancaster Stemmer (the differences are not discussed in this study). Inverted Index is a mapping of terms to any document containing the terms and the position (index) of terms in the document. Example of inverted index is “budi—1:17,30,63;3:1,4,8”, means that the term ‘budi’ in the document ID ‘1’ with position (index) of ‘17, 30, and 63’, and in the document ‘3’ with position (index) of ‘1, 4, and 8’.

Posting-List is the value of the mapped term. In the above example, the posting-list for the term ‘budi’ is ‘1:17,30,63;3:1,4,8’. Term Frequency (tf) is the number a term appears in a document. Document Frequency (df) is the number of documents that contain a term. Inverse Document Frequency (idf) is the result of total documents divided by document frequency (df).

II. RESEARCH METHOD

This study uses experimental method for getting and analyzing quantitative data. The researchers use Python for creating index and querying index for both BR and VSM methods. The comparison of used application for creating and querying the inverted index is described simply in Table II.

After writing the source code, the researchers execute it to create an inverted index of Reuters corpus. The application does process of tokenization, stopwords removal, stemming, inverted index and generating the files “titleIndex.dat” and “testIndex.dat.” Both the files will be read in query process and show the result. The process is described in Fig. 1.

A. Creating Inverted Index

The first thing is creating an inverted index. The application reads the corpus and process the content

| Table II | A Comparison of Creating and Querying Index |
|----------|--------------------------------------------|
| **Boolean Retrieval** | **Index Function:** Create mapping of the terms and its posting-list (inverted index) from available file collection. | **Input:** Files / documents in a corpus. **Output:** File “testIndex.dat” contains terms and its posting-list (document ID and its index position). **File “titleIndex.dat” contains document ID and its name.** |
| **Query** | **Function:** Display search results. | **Input:** keywords, file “testIndex.dat” and “titleIndex.dat” **Output:** List of search results that are not-ranked |

| **Vector Space Model** | **Index Function:** Create mapping of the terms of its posting-list (inverted index) from available file collection, and provide term frequency (tf) and inverse document-frequency (idf). | **Input:** Files / documents in a corpus. **Output:** File “testIndex.dat” contains terms and its posting-list (document ID and its index position), term-frequency (tf) and inverse document-frequency (idf). **File “titleIndex.dat” contains document ID and its name.** |
| **Query** | **Function:** Display search results. | **Input:** keywords, file “testIndex.dat” and “titleIndex.dat” **Output:** List of search results that are ranked (based on the most relevant documents). |
into words (tokenize), remove stop-words, and remove affix-words (stemming). After that, it will create posting-list (inverted index). For VSM, the posting-list will be completed with term-frequency (tf), and inverse document-frequency (idf). Last, it will write all the document titles and inverted-index into files. The process is described in Fig. 2.

In the VSM method, term frequency (tf) is obtained by the number of terms appearing in a document using the formula:

\[ tf_{term,doc} = N_{term,doc} \]  \hspace{1cm} (1)

where \( N_{term,doc} \) is the frequency occurrence of a word in a document. Then, the term frequency is normalized by

\[ \hat{tf}_{term,doc} = \frac{N_{term,doc}}{E_{doc}} \]  \hspace{1cm} (2)

where \( E_{doc} \) denotes the Euclidian norm and is defined by

\[ E_{doc} = \sqrt{\sum_{term} tf_{term,doc}^2} \]  \hspace{1cm} (3)

It has been implemented by Ref. [15].

In searching two or more terms, such as ‘authoris buckey’, it is necessary to rank these two terms. By doing ranking based on term-frequency is not sufficient. There is possibility of generating the same term-frequency. It is necessary to calculate document-frequency that is the number of documents that contain that term or

\[ df_{term} = N_{term} \]  \hspace{1cm} (4)

The more documents contain the term, the more unimportant the document because the term is too common. In addition, there should be an inverse document-frequency according to the formula:

\[ idf_{term} = \frac{N_{doc}}{df_{term}} \]  \hspace{1cm} (5)

where \( N_{doc} \) is the number of the documents containing the term. The statistic \( idf_{term} \) reflects how important a word is to a document in a corpus. The statistic is often expressed in the log-scale by:

\[ idf_{term} = \log \left( \frac{N_{doc}}{df_{term}} \right) \]  \hspace{1cm} (6)

B. Querying Index

The querying index is by reading document that contains document title and inverted index. Query can be from one (single) word, multiple words (free text), or exact (phrase) words. The process of query is described in Fig. 3.
The ranking of the relevant documents is the results of a dot product of the document vector and the query vector:

$$\text{rank} = \text{tf} \cdot \text{idf}$$  \hspace{1cm} (7)

For examples, we consider four documents with the dot-product values of the following: docVector[46] · queryVector = 20.622, docVector[224] · queryVector = 28.8708, docVector[984] · queryVector = 16.694, and docVector[1090] · queryVector = 68.249.

Using the dot-product results in the distance of document vector (DocID: 46, 224, 984, and 1090) to the query vector. DocVector that is closest to the queryVector is the most relevant document. This is indicated by smaller cosine angle towards queryVector (Fig. 4), or greater summed value of dot product. To obtain ranking of relevant documents, then sort in descending the whole summed values of the dot product. From the results, we obtain that DocID 1090 (68.249) is the most relevant document, following by DocID 224 (28.87), DocID 46 (20.622), and DocID 984 (16.694). This is implemented in file “queryIndex_tfidf.py” as follows:

```python
for docIndex, (doc, postings) in enumerate(docVectors):
    if doc in docs:
        docVectors[doc][termIndex] = self.tf[term][docIndex]
```

To display in 3D Cartesian, we often need to adjust docVector values (if too large) by using same multiplication factor. Value adjustment will not change the angle of vector, it’s only pull the vertex near to the coordinate center. Adjusted values are displayed in Table III and drawn in Fig. 4.

Fig. 4. Comparison of DocVector and QueryVector.
been indexed in the previous documents. The use of (terms) that appear on the following documents have selection is decreased. This happens because the words generated index to the total number of words in the collection are generated. But, the percentage of the indexes are generated. But, the percentage of the collection size. The time required by VSM method create the inverted index increases with increasing the file collection size. The time required by VSM method to create the inverted index increases with increasing the size of the collection.

The results of creating the inverted index are shown in Table V. The results show that the larger the size of collection, the more reading time is required. The VSM method requires less time than the BR method. The BR and VSM methods show similar results, proving the consistency of the algorithms. They also show that the larger the size of collection is, the greater the size of generated index file. The use of BR method generates an index file size smaller than the collection of source files, while the VSM method generates a larger index file than the collection of source files (see Table VI). The test results of reading/querying index are shown in Table VII. They suggest that the larger the size of the collection is, the longer the time it takes to read the index file that has been generated previously, the longer the time required to perform the query, and the more search results found. The use of BR method requires less time than the VSM method, and show similar results to prove the consistency of the algorithm. Results on VSM method already ranked by using tf-idf formula (term frequency with inverse document frequency).

In summary, the process of creating index and querying term digit beye controversi can be demonstrated through the results presented in Tables VIII and IX from a study of 1473 documents.

The results are also same with the file
The rankings for the document relevance is calculated through the dot-product. The results are displayed in Table X. The results are also same with the output of executed application as follows.

| Doc ID | N_{term,doc} | E_{doc} | tf |
|--------|--------------|---------|----|
| 46     | 0 0 2       | 47.62   | 0 0 0.0420 |
| 224    | 1 0 0       | 25.51   | 0 0 0.0392 |
| 984    | 0 0 1       | 29.41   | 0 0 0.0340 |
| 1090   | 1 0 1       | 17.98   | 0 0 0.0556 |

N_{term,doc} = The Number of Term Appears

tf = N_{term,doc} / E_{doc} = Term-Frequency

| Doc ID | N_{term,doc} | E_{doc} | tf |
|--------|--------------|---------|----|
| 46     | 0 0 2       | 47.62   | 0 0 0.0420 |
| 224    | 1 0 0       | 25.51   | 0 0 0.0392 |
| 984    | 0 0 1       | 29.41   | 0 0 0.0340 |
| 1090   | 1 0 1       | 17.98   | 0 0 0.0556 |

TABLE VIII
TERM APPEARANCE IN DOCUMENTS.

| Doc ID | N_{term,doc} | E_{doc} | tf |
|--------|--------------|---------|----|
| 46     | 0 0 2       | 47.62   | 0 0 0.0420 |
| 224    | 1 0 0       | 25.51   | 0 0 0.0392 |
| 984    | 0 0 1       | 29.41   | 0 0 0.0340 |
| 1090   | 1 0 1       | 17.98   | 0 0 0.0556 |

TABLE IX
INVERSE DOCUMENT FREQUENCY.

|       | digit | beye | controversi |
|-------|------|------|-------------|
| dfterm (n doc contain term) | 2    | 0    | 3           |
| idfterm (1473 / dfterm)    | 736.5| 0    | 491.0       |

The rankings for the document relevance is calculated through the dot-product. The results are displayed in Table X. The results are also same with the output of executed application as follows.

IV. CONCLUSIONS AND FUTURE WORK

Results of applied algorithm in Python for both BR and VSM are working properly when those are compared to manual calculation. By using corpus Reuters (2610 docs, 2 MB), we find there is no significant time difference for creating inverted index for both method. The differences come from querying index. Number of generated indexes, generated file size, duration of reading and searching index, and results found are growing inline with corpus file number and file size. However, choosing the appropriate method of BR or VSM is basically depending on the need of document ranking availability.

Comparing the implementation of BR and VSM can be applied in many other fields. Further researches may try non-Latin language, such as Chinese or Arab [3, 9]. There are also other bigger corpus to be used for next experiments such as Reuters Corpora (RCV1, RCV2, TRC2) [16], and can be combined with MapReduce (Hadoop) [17, 18]. Validation for recall, precision, accuracy, f-measure (f-score), and error rate are also interesting topics for future works [3, 11].
Cite this article as: B. Yulianto, W. Budiarto, and I. H. Kartowisastro, “The Performance of Boolean Retrieval and Vector Space Model in Textual Information Retrieval”, CommIT (Communication & Information Technology) Journal 11(1), 33–39, 2017.

| Vector        | Term       | Dot Product | Sum    | Rank |
|---------------|------------|-------------|--------|------|
| Query Vector  | 736.5      | 0           | 491.0  |      |
| Doc Vector    | 0.0392     | 0           | 0      | 20.62| 3    |
| Doc Vector    | 0.0556     | 0           | 0.056  | 16.69| 4    |
| Doc Vector    | 0.0392     | 0           | 0.056  | 16.69| 1    |

TABLE X
THE INVERSE DOCUMENT-FREQUENCY AND RANKED RELEVANT DOCUMENTS.

[8] C. D. Manning, P. Raghavan, and H. Schütze, An Introduction to Information Retrieval. Cambridge University Press, 2009.
[15] D. Dertat. (2012) Accessed on March 30, 2015. [Online]. Available: http://www.ardendertat.com/2012/01/11/implementing-search-engines/
[16] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, “Rcv1: A new benchmark collection for text categorization research,” Journal of machine learning research, vol. 5, no. Apr, pp. 361–397, 2004.
[17] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
[18] D. Hiemstra and C. Hauff, “Mirex: Mapreduce information retrieval experiments,” CTIT, Tech. Rep., 2010.