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

Budi Yulianto[1], Widodo Budiharto[2], Iman H. Kartowisastro[3]
Doctor of Computer Science, Bina Nusantara University, Jakarta 11480, Indonesia
Emails: [1]laboratory@binus.ac.id, [2]wbudiharto@binus.edu, [3]imanhk@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].

Received: Apr. 2, 2017; received 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 appearring 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].

TABLE I
Advantages and Disadvantages of the BR and VSM Methods.

	BR	VSM
Advantages	Faster search Easy to understand and implement	Ranked document Easy to implement; BR is easier
Disadvantages	Unranked document	Hard to understand Slower search

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

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 that 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
Create Index	Python Soucecode File Name: "`createIndex.py`" 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 dan its name.
Query Index	Python Sourcecode File Name: "`queryIndex.py`" Function: Display search results. Input: keywords, file "`testIndex.dat`" and "`titleIndex.dat`" Output: List of search results that are not-ranked
	Vector Space Model
Create Index	Python Sourcecode File Name: "`createIndex_tfidf.py`" 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 dan its name.
Query Index	Python Sourcecode File Name: "`queryIndex_tfidf.py`" 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).

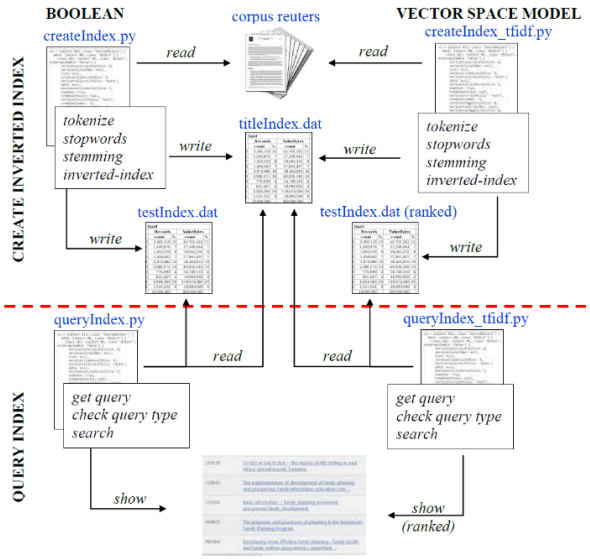

Fig. 1. The Steps of Experiment.

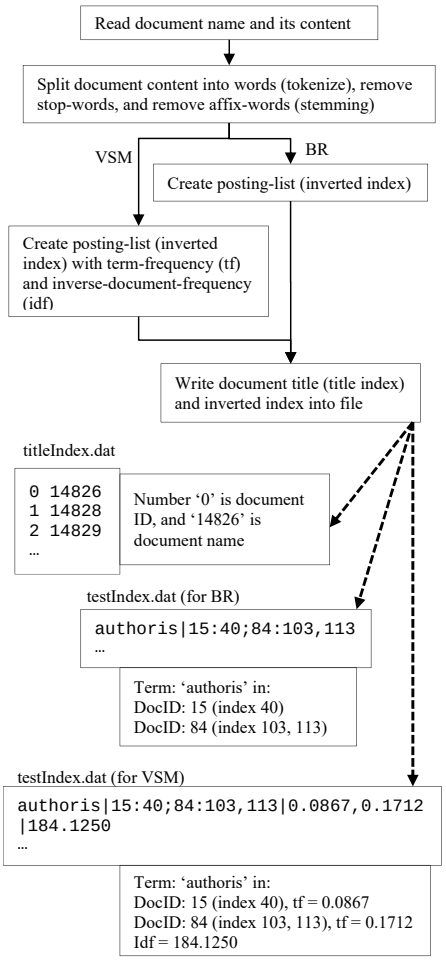

Fig. 2. Flowchart of Creating Inverted Index.

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:

$$\mathrm{tf}_{\mathrm{term,doc}} = N_{\mathrm{term,doc}}, \tag{1}$$

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

$$\hat{\mathrm{tf}}_{\mathrm{term,doc}} = \frac{N_{\mathrm{term,doc}}}{E_{\mathrm{doc}}}, \tag{2}$$

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

$$E_{\mathrm{doc}} = \sqrt{\sum_{\mathrm{term}} \mathrm{tf}^2_{\mathrm{term,doc}}}. \tag{3}$$

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

$$\mathrm{df}_{\mathrm{term}} = N_{\mathrm{term}}. \tag{4}$$

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:

$$\mathrm{idf}_{\mathrm{term}} = \frac{N_{\mathrm{doc}}}{\mathrm{df}_{\mathrm{term}}} \tag{5}$$

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

$$\mathrm{idf}_{\mathrm{term}} = \log\left(\frac{N_{\mathrm{doc}}}{\mathrm{df}_{\mathrm{term}}}\right) \tag{6}$$

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.

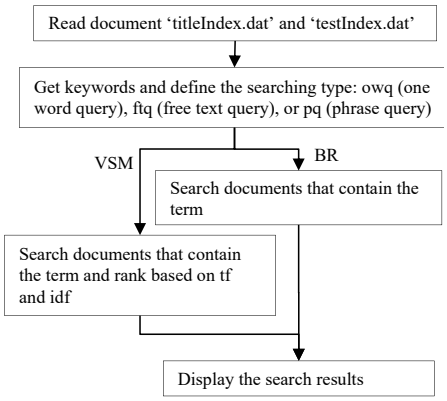

Fig. 3. The flowchart for querying an index.

For example, we search a sentence of digit beye controversi (searching type: ftq), then based on file testIndex.dat is obtained posting-list of each terms:

```
digit|224:90;1090:46|0.0392,0.0556|736.5000
controversi|46:337,475;984:28;1090:24|0.0420,0.0340,0.0556|491.0000
```

In VSM method, there is ranking process of relevant documents. Assume term-index 0 for term digit, term-index 1 for term beye, and term-index 2 for term controversi, then the query vector for each term is idf value: queryVector [0] = 736.5000, queryVector [1] = 0 (since no idf for term beye), and queryVector [2] = 491.0000, so queryVector is [736.5000, 0, 491.0000].

This is implemented in file "`queryIndex_tfidf.py`" as follows:

```
for termIndex, term in enumerate(terms):
    ...
    queryVector[termIndex]=self.idf[term]
```

Both posting-list of term digit or controversi contain DocID [46, 224, 984, 1090]. Then, we generate document vector based on term-frequency of each terms DocID. For term-index 0 (term digit), we obtain DocID 224, and 1090, so the document vector is: docVector [224][0] = 0.0392, and docVector [1090][0] = 0.0556.

For term-index 2 (term controversi), we obtain DocID 46, 984 and 1090, so the document vector is: docVector [46][2] = 0.0420, docVector [984][2] = 0.0340 and docVector [1090][2] = 0.0556. This is implemented in file "`queryIndex_tfidf.py`" as follows:

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

The ranking of the relevant documents is the results of a dot product of the document vector and the query vector:

$$\text{rank} = \text{tf}_{\text{term,doc}} \cdot \text{idf}_{\text{term}}. \quad (7)$$

For examples, we consider four documents with the doc-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:

```
docScores=[[self.dotProduct(curDocVec, \
    queryVector), doc] for doc, curDocVec in \
    docVectors.iteritems()]
docScores.sort(reverse=True)
```

The dot-product function is implemented in file "`queryIndex_tfidf.py`" as follows:

```
def dotProduct(self, vec1, vec2):
    ...
    return sum([x*y for x,y in zip(vec1,vec2)])
```

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.

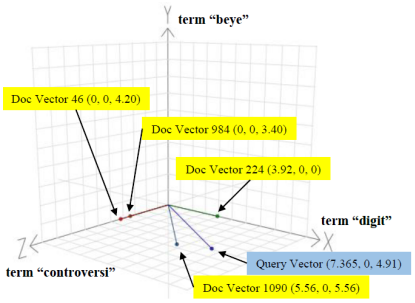

Fig. 4. Comparison of DocVector and QueryVector.

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.

TABLE III
ADJUSTING VALUES OF VECTOR.

Vector	Value Before Adjustment			Multiplication Factor	Value After Adjustment		
	digit	beye	controversi		digit	beye	controversi
Query Vector	$7.365 \times 10^{+2}$	0.0	$4.91 \times 10^{+2}$	1.0×10^{-1}	7.365	0.0	4.91
Doc Vector (46)	0.0	0.0	4.20×10^{-2}	$1.0 \times 10^{+2}$	0.0	0.0	4.20
Doc Vector (224)	3.92×10^{-2}	0.0	0.0	$1.0 \times 10^{+2}$	3.92	0.0	0.0
Doc Vector (984)	0.0	0.0	3.40×10^{-2}	$1.0 \times 10^{+2}$	0.0	0.0	3.40
Doc Vector (1090)	5.56×10^{-2}	0.0	5.56×10^{-2}	$1.0 \times 10^{+2}$	5.56	0.0	5.56

TABLE IV
THE SETS OF THE COLLECTION OF FILES OF CORPUS REUTERS FOR TESTING.

Set Number	File Size (MB)	Number of Docs	Number of Words
1	0.5	615	82725
2	1.0	1300	167143
3	1.5	1870	250254
4	2.0	2610	335012

TABLE V
THE TEST OF CREATING INVERTED INDEX.

Set	Duration (s)		Number of Index		Portion of Index (%)	
	BR	VSM	BR	VSM	BR	VSM
1	1.745	1.708	6399	6399	7.74	7.74
2	3.542	3.443	9549	9549	5.71	5.71
3	5.209	5.131	11947	11947	4.77	4.77
4	7.001	6.879	14391	14391	4.30	4.30

TABLE VI
THE SIZE OF THE GENERATED INDEX FILE IN KB.

Set	BR	VSM	Source Files
1	388	698	500
2	773	1383	1000
3	1156	2040	1500
4	1566	2753	2000

TABLE VII
THE TEST OF READING AND SEARCHING INDEX.

Set	Reading (s)		Searching (s)		# Results Found	
	BR	VSM	BR	VSM	BR	VSM
1	0.219	0.256	0.000	0.016	151	151
2	0.464	0.516	0.016	0.032	324	324
3	0.656	0.755	0.031	0.047	504	504
4	0.906	0.995	0.031	0.062	698	698

III. RESULTS AND DISCUSSION: PERFORMANCE TEST

In addition to the comparison of the concepts and the algorithm above, the performance test is also conducted using the Reuters corpus. The data contains 2610 files, divided into 4 sets of collection based on size scale of 0.5 MB (Table IV). The set 1 is a subset of the sets 2, 3, and 4. Each performance test is conducted 12 times with the highest and lowest results are removed. The remaining 10 results are averaged.The results of the performance test are in the following tables.

The results of creating the inverted index are shown in Table V. The results show that the required time to create the inverted index increases with increasing the file collection size. The time required by VSM method is just slightly lower than the BR method.

The result of the generate file `testIndex.dat` shows that the larger the size of collection, the more indexes are generated. But, the percentage of the generated index to the total number of words in the collection is decreased. This happens because the words (terms) that appear on the following documents have been indexed in the previous documents. The use of 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

TABLE VIII
TERM APPERANCE IN DOCUMENTS.

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

$N_{term,doc}$ = The Number of Term Appears
$tf = N_{term,doc}/E_{doc}$ = Term-Frequency

TABLE IX
INVERSE DOCUMENT FREQUENCY.

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

"`testIndex.dat`" generated as follows.

```
digit|224:90;1090:46|0.0392,0.0556|736.5000
controversi|46:337,475;984:28;1090:24|0.0420,0.0340,0.0556|491.0000
```

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.

```
type word(s) : digit beye controversi
Doc ID: 1090 / Doc Title: 16747
Doc ID: 224 / Doc Title: 15263
Doc ID: 46 / Doc Title: 14921
Doc ID: 984 / Doc Title: 16564
Total Doc(s): 4
```

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].

REFERENCES

[1] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.

[2] A. Gomathi, J. Jayapriya, G. Nishanthi, K. Pranav, and G. Praveen Kumar, "Ontology based semantic information retrieval using particle swarm optimization," *International Journal of Applied Information Communication Engineering*, vol. 1, no. 4, pp. 5–8, 2015.

[3] M. O. Nassar, F. A. Mashagba, and E. A. Mashagba, "Improving the user query for the boolean model using genetic algorithms," *International Journal of Computer Science Issues*, vol. 8, no. 1, pp. 66–70, 2011.

[4] E. Al Mashagba, F. Al Mashagba, and M. O. Nassar, "Query optimization using genetic algorithms in the vector space model," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 5, 2011.

[5] A. Depeursinge, S. Duc, I. Eggel, and H. Muller, "Mobile medical visual information retrieval," *IEEE Transactions on information technology in biomedicine*, vol. 16, no. 1, pp. 53–61, 2012.

[6] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys (Csur)*, vol. 40, no. 2, p. 5, 2008.

[7] X. Peng, D. Ke, Z. Chen, and B. Xu, "Automated chinese essay scoring using vector space models," in *Universal Communication Symposium (IUCS), 2010 4th International*. IEEE, 2010, pp. 149–153.

[8] X. Peng, D. Ke, and B. Xu, "Automated essay scoring based on finite state transducer: towards asr transcription of oral english speech," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 2012, pp. 50–59.

[9] W. Maitah, M. Al-Rababaa, and G. Kannan, "Improving the effectiveness of information retrieval system using adaptive genetic algorithm," *International Journal of Computer Science & Information Technology*, vol. 5, no. 5, p. 91, 2013.

[10] S. Sarkar, A. Roy, and B. S. Purkayastha, "Clustering of documents using particle swarm optimization and semantics information," *Int. J. of Comput. Sci. & Inform. Technologies*, vol. 5, no. 3, 2014.

[11] M. Sharma and R. Patel, "A survey on information retrieval models, techniques and applica-

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

Vector	Term			Dot Product			Sum	Rank
	digit	beye	controversi	digit	beye	controversi		
Query Vector	736.5	0	491.0					
Doc Vector	0	0	0.042	0	0	20.62	20.62	3
Doc Vector	0.0392	0	0	28.82	0	0	28.82	2
Doc Vector	0	0	0.034	0	0	16.69	16.69	4
Doc Vector	0.0556	0	0.056	40.95	0	27.30	68.25	1

tions," *International Journal of Emerging Technology and Advanced Engineering, ISSN*, vol. 3, no. 11, pp. 542–545, 2013.

[12] S. Hassan, M. Rafi, and M. S. Shaikh, "Comparing svm and naive bayes classifiers for text categorization with wikitology as knowledge enrichment," in *Multitopic Conference (INMIC), 2011 IEEE 14th International*. IEEE, 2011, pp. 31–34.

[13] K. Žubrinić, M. Miličević, and I. Zakarija, "Comparison of naive bayes and svm classifiers in categorization of concept maps," *International journal of computers*, vol. 7, no. 3, pp. 109–116, 2013.

[14] 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.