ITC 2/50
Information Technology and Control
Vol. 50 / No. 2 / 2021
pp. 375-389
DOI 10.5755/j01.itc.50.2.25470

Big Data Full-Text Search Index Minimization Using Text Summarization

Received 2020/03/10 | Accepted after revision 2021/05/03

http://dx.doi.org/10.5755/j01.itc.50.2.25470

HOW TO CITE: Iqbal, W., Ilyas, W., Bukhari, F., Almustafa, K. M., Nawaz, Z. (2021). Big Data Full-Text Search Index Minimization Using Text Summarization. *Information Technology and Control*, 50(2), 375-389. https://doi.org/10.5755/j01.itc.50.2.25470

Big Data Full-Text Search Index Minimization Using Text Summarization

Waheed Iqbal, Waqas Ilyas Malik, Faisal Bukhari
Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan

Khaled Mohamad Almustafa
College of Computer and Information Sciences, Prince Sultan University Riyadh, Saudi Arabia

Zubair Nawaz
Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan

Corresponding author: waheed.iqbal@pucit.edu.pk

An efficient full-text search is achieved by indexing the raw data with an additional 20 to 30 percent storage cost. In the context of Big Data, this additional storage space is huge and introduces challenges to entertain full-text search queries with good performance. It also incurs overhead to store, manage, and update the large size index. In this paper, we propose and evaluate a method to minimize the index size to offer full-text search over Big Data using an automatic extractive-based text summarization method. To evaluate the effectiveness of the proposed approach, we used two real-world datasets. We indexed actual and summarized datasets using Apache Lucene and studied average simple overlapping, Spearman's rho correlation, and average ranking score measures of search results obtained using different search queries. Our experimental evaluation shows that automatic text summarization is an effective method to reduce the index size significantly. We obtained a maximum of 82% reduction in index size with 42% higher relevance of the search results using the proposed solution to minimize the full-text index size.

KEYWORDS: Big Data, Indexing, Searching, Index Minimization, Text Summarization.

1. Introduction

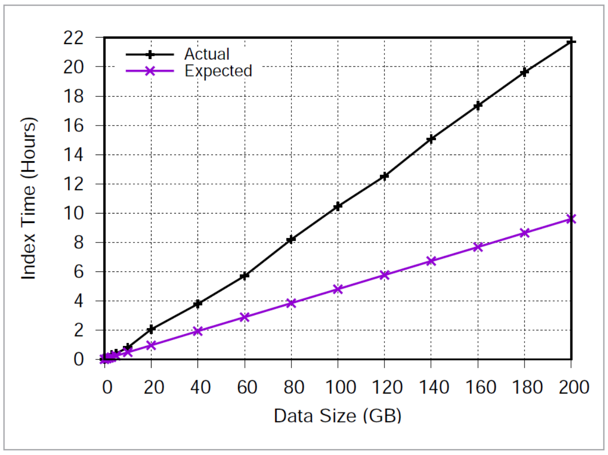
Recent advancements and adaptation of technology are contributing to growing digital data exponentially. For example, nowadays inexpensive and readily available network-enabled electronic devices (smartphones, laptops, personal computers, and tablets), the adaptation of social networks, and electronic healthcare gadgets are widely used and generate enormous data. This increasing growth of data poses special challenges to process, store, and analyze it [27]. To overcome this challenge, recently a new research area namely Big Data has emerged. One of the important research problems in Big Data is to provide efficient full-text search services on a large dataset. A common technique to provide text search is through data indexing [5, 6]. There are many data structures in practice to provide data indexing. One of the widely used data structure is inverted index [6]. This data structure is based on the hash table. Each entry in the inverted index is a key-value pair, where the key is a term and the value is a list of document identifiers containing the term. All of the terms from the specific dataset are also known as term dictionary and the corresponding list of document identifiers are known as posting list. An inverted index after compression is roughly 20-30% of the actual size of the dataset.

Apache Lucene is the most used library to index data [24, 29, 32] for providing full-text search. It uses inverted index data structure to provide efficient data search capabilities. Furthermore, Lucene uses compression techniques to reduce the size of the index. However, still, the index size remains around 20% to 30% of the actual size of the data. To entertain search queries, Lucene loads the inverted index into a physical memory as a hash table containing term dictionary and posting lists. For each query consisting of multiple words, Lucene identifies the corresponding posting lists, merge them, and rank the documents to return as a search query result.

Lucene index generation is a time-consuming task specifically for large datasets [17, 40]. Figure 1 shows profiling of index time for different sizes of datasets using Lucene. The figure shows actual and expected index time for the datasets. The actual line is plotted after profiling index time for different dataset sizes varying from 1 GB to 200 GB. Whereas, the expected line is plotted by fitting the line using small size datasets varying from 1 GB to 10 GB. This shows that on increasing size of datasets, the performance of Lucene decreases significantly. We advocate that a large dataset can be reduced to a smaller representative dataset for indexing to offer full-text search with better performance.

Figure 1

Apache Lucene index generation time profiling using different sizes of datasets



Traditionally data index is minimized using posting list compression techniques [50, 51]. Compression algorithms provide an effective reduction in space but introduce overhead on decompression as it requires to serve the queries which reduce the speed of search queries drastically for a large index size. We advocate to reduce the actual dataset using text automatic summarization method and then posting list compression methods can further be used to reduce the index size. Automatic text summarization [35] is a process to create a summary of a text document by significantly reducing its size. However, it ensures to retain important points of the document. In this paper, we investigate to minimize the index size of Big Data using an automatic text summarization method. To evaluate the effectiveness of this approach, we performed four different experiments using two datasets to study average overlapping, average ranking score, and Spearman's rho correlation measures of search results using different search queries in comparison with actual datasets.

The main contributions of this paper includes:

- We propose an automatic extractive-based text summarization for Big Data index minimization for the full-text search problem.
- We evaluate the effectiveness of the proposed method by studying relevance and overlapping of the search query results with baseline datasets.
- Study the effect of different text summarization threshold levels on data index minimization and search results.

The rest of this paper is organized as follow. Related work is discussed in Section 2. Commonly used Big Data tools for fulltext search are discussed in Section 3. The proposed solution for index generation using automatic text summarization is presented in Section 4. Experimental evaluation setup is discussed in Section 5. Experimental results are presented in Section 6. Finally, the conclusion is drawn and future work is discussed in Section 7.

2. Related Work

There have been excellent efforts to develop tools, methods, and programming models to store, process, and analyze Big Data [3, 31, 41]. The full-text search on the Big Data is a challenging and interesting problem which recently gained attention. Many applications and domains are using full-text search. For example, Cuggia et al. [9] developed a full-text search engine to use clinical notes for identifying different diseases. Garcelon et al. [15] use full-text search to detect the family history of patients from a biomedical data warehouse. Hanauer et al. [19] develop a search query recommendations system which exploits the query semantics using synonym variants of the query text and obtain most relevant data from electronic health record system. Abe et al. [1] present a high-speed full-text search engine for system log files. The solution automatically converts system log files into an efficient searchable index and provide good performance to facilitate full-text search for end-users. Wang et al. [44] use full-text search for large-scale cloud data center monitoring. Their proposed solution is based on tree index structure and correlation methods to index the data and obtain relevant results.

Full-text search on Big Data is commonly achieved using data indexing and hashing methods. A comprehensive survey on Big Data indexing methods is reported by Gani et al. [13]. Zhu et al. [51] introduced sparse hashing for effectively searching high-dimensional data by reducing the dimensionality of data dynamically. He et al. [20] proposed and evaluated deep learning solution to image-text retrieval using two convolution-based networks to offer efficient image-text retrieval. Wang et al. [43] present a survey on learning to hash algorithms and categorize them. The learning to the hash method is used to find data elements from the database on given query so the distance of the selected data elements is minimum with the query text.

There have been several efforts to minimize search index using posting list compression techniques. For example, Zhang et al. [50] discussed inverted index compression for high-performance information retrieval systems by compressing posting list. This work explained various posting list compression algorithms and then proposed a solution to select method to use disk speed, cache size, and memory effectively to improve performance for search engines. Yan et al. [47] proposed to reorder document IDs in the posting list for higher data index compression. They proposed a method to optimize compression for posting list and query processing by optimally reordering the documents. Wang et al. [42] proposes a new inverted list compression method based on multiple techniques including fixed-bit encoding, inblock compression, dynamic data partitioning, and cache-aware optimization to improve the query performance. Another work by Claude et al. [7] introduced a new method to compress inverted indexes for applications required full-text facility on a large repository of repetitive documents like version control systems. Wu et al. [46] reduce the indexing space and time by using indexing blocks instead of individual records for minimizing the query processing time.

The relevance of the full-text search results is important and many studies have been performed to observe user's behavior towards the ranking of the full-text search results. Most of these studies show that only a few tops ranked results are important from user's perspective [22, 23, 37, 38]. Bar-Ilan et al. [4] discussed different techniques used to correlate the rankings of search engine results. This work applied different methods of comparison of top 10 results using a specific set of queries and compared ranked results returned by major search engines. Ghose et al. [16] present a study of ranking results obtained from a search engine based on consumer behavior and revenue of search engine using hierarchical Bayesian model. Williams et al. [45] use pseudo-relevance feedback recursively to improve search results for given text query.

Automatic text summarization is a well-established research area [2, 12] which reduces the size of the text significantly. However, it ensures to retain important sentences and central idea of the given text while reducing its size. Two different methods exist for automatic text summarization namely Extractive and Abstractive. In Extractive method [28], important sentences are picked up from the given text to generate a compact summary. It first ranks sentences according to their importance and then assigns them relevance score and finally selects the sentences with the higher score as a summary of the document. In Abstractive method [14, 25], Natural Language Processing (NLP) methods are used to generate the summary of given text document. This method generates the summary with possibly new vocabulary and sentences similar to a human generated summary of the documents. Recently, advanced techniques are used to improve automatic text summary generation. For example, [48] introduced deep learning for automatic text summarization.

To our knowledge, no work has investigated the use of text summarization methods for Big Data search index minimization. We take the first step to introduce text summarization for reducing search index size significantly while providing higher relevance of the search results.

3. Big Data Indexing Tools

3.1. Apache Lucene

Apache Lucene [32] is a high-performance open-source information retrieval library written in Java programming language. It is primarily used for indexing and searching of text data. Lucene provides fast indexing and fast searching capabilities for very large datasets. It can process roughly 150 GB/hour of data on latest hardware [10] with heap consumption of only 1 MB. The index size generated by Lucene is 20-30% of actual dataset size. Besides simple indexing and searching functionality, Lucene also ranks the search results to show the most relevant results in descending order of relevance. These features make it appealing to use and build Big Data solutions on top of Lucene.

3.2. Apache Solr

Apache Solr [18] is a highly scalable enterprise search engine that uses Lucene for indexing and searching functionality. Solr extends Lucene and provides functionality like rich documents processing (including PDF, XML, HTML etc.), integration with the database, index replication and load balancing for fault tolerance. Solr also provides Distributed Searching by introducing the concept of Shards. Solr provides REST-based XML/JSON APIs that make it integrable with most of the programming languages. Solr exploits the fast-searching capability of Lucene and make sure the availability of documents for searching immediately after they are added for indexing.

3.3. Elasticsearch

Elasticsearch [26] is also an enterprise search engine. Like Solr, it also uses Lucene for indexing and searching. Elasticsearch is much similar to Solr in terms of its functionality. Elasticsearch, like Solr also provides distribution of index by dividing it into different Shards. It maintains replicas of every Shard. Elasticsearch also provides a feature of Gateway that allows recovery in case of any server crash. Elasticsearch supports NoSQL solutions which makes it attractive to use as a database with Big Data applications. However, it doesn't support distributed transactions.

3.4. Cloudera Search

Cloudera Search uses Hadoop Distributed File System (HDFS) for storing data indices to provide near to real-time full-text search facility. It is based on Apache Solr and provides fast individual and batch indexing of text data. It works by indexing events (streamed by Flume) while they are being stored in HDFS. It first maps all events to Solr schema and then uses Lucene for indexing of events. Cloudera Search offers fault tolerance by leveraging the benefits of HDFS. Cloudera Search is easy to integrate with HBase to provide full-text search.

3.5. Sphinx

Sphinx is an open-source search engine written in C++ which uses native protocols to communicate with any Data Base Management System (DBMS).

This allows Sphinx to directly index data of any DBMS. It also works with NoSQL-based database and allows the user to use it with raw text data to index and use it in their applications. Sphinx also allows RDBMS like query style (use of WHERE, GROUP BY etc. clauses). It offers aggregate functions for sum, average, minimum, maximum etc. It also allows the distributed searching and very easy to integrate with any application.

3.6. Xapian

Xapian is an open-source search engine library written in C++. It is fast and highly scalable for searching text documents and can scale up to hundreds of millions of documents. Xapian has a built-in support for Probabilistic IR models for the ranking of results. It also allows the use of Boolean operators like AND, OR etc. Xapian allows transactions with a guarantee of data consistency in case of any failure. Some of the important features available in Xapian are data updates, automatic spell correction, probabilistic ranking algorithms, and intuitive usage of synonyms for the given text query.

Among all of these Apache Lucene is the most powerful, mature, and famous among Big Data application developers to use for offering the full-text search facility. Moreover, the flexibility of Apache Lucene to use its APIs and easy customization of the source code helps greatly to integrate and implement our proposed index minimization method.

4. Proposed Index Minimization Using Text Summarization

Our proposed solution is based on automatic text summarization for Big Data index minimization to offer efficient fulltext search. We explained automatic text summarization method and the proposed system in the following subsections in turn.

4.1. Automatic Text Summarization Methods

Automatic text summarization is a process to create a summary of a text document by significantly reducing its size. However, it ensures to retain important points of the document. Mainly, two different methods exist for automatic text summarization [35] namely Extractive and Abstractive. In Extractive method, important keywords, and sentences from the original text are selected to create the summary. However, in Abstractive method, natural language processing techniques are used to create the summary. This method generates a summary which looks closer to a human generated summary of the document. But this method may not use sentences and keywords from the original document to prepare the summary.

Searching text documents heavily rely on the keywords present in the documents, therefore, in our context Extractive method is appropriate to prepare the summary of the document. Then, we index the summary to significantly reduce the index size. Most of the Extractive methods generate a summary by finding the similarity between sentences and then assigning a similarity score to them. Finally, our method selects sentences having the higher similarity scores to prepare the summary. There are two commonly used approaches to prepare extractive-based text summary known as Textrank-based [30] and centroid-based [33]. Textrank-based algorithm prepares a complete graph of sentences. Where each sentence represents a vertex in the graph and edges represent intersection score between two sentences. In this paper, we chosen centroid-based [33] algorithm to prepare the summary of text documents as this method is better than textrank-based text summarization algorithm [11]. We explained the centroid-based text summarization method in the following subsection.

4.1.1. Centroid-based Text Summarization

The centroid-based algorithm identifies a set of keywords, labels them as centroid and then identify cosine similarity [33, 39] among other keywords to the centroid. To identify set of centroid keywords for a document, many techniques exist. For example, Cohen [8] proposed to use n-gram statistics to identify the set of keywords. Ramos [34] proposes to use term frequency-inverse document frequency (tf-idf) of keywords to prepare a set of important keywords of a document. Once the list of keywords is prepared then cosine similarity score for each sentence is computed with the centroid set of keywords. Finally, sentences with higher cosine similarity are picked. However, the number of selected sentences is defined by the user as a percentage of the text (summary threshold) required to be part of the summary.

More formally, let d_i is a given document to generate a summary, a user defined τ which represents maximum summary size in percentage for the given document, and $C = \{k_1, k_2, ..., k_n\}$ is a set of keywords identified as a list of centroid keywords using tif-idf measure. A vector $\vec{C}$ represent the tf-idf vector of C. The document d_i consists of a set of sentences $S = \{S_1, S_2, S_3, ..., S_m\}$ and we have a tf-idf vector $\vec{S_i}$ identified for each sentence. We use the following formula to identify the consine similarity between each $\vec{S_i}$ with $\vec{C}$:

$$sim(S_i, C) = \frac{\vec{S_i} \cdot \vec{C}}{\|\vec{S_i}\|_2 \|\vec{C}\|_2} \tag{1}$$

Then we get a list of similarity for each sentence in the document:

$$\begin{pmatrix} sim(S_1, C) & sim(S_2, C) & sim(S_3, C) \\ sim(S_4, C) & sim(S_5, C) & sim(S_6, C) \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ sim(S_{m-2}, C) & sim(S_{m-1}, C) & sim(S_m, C) \end{pmatrix}$$

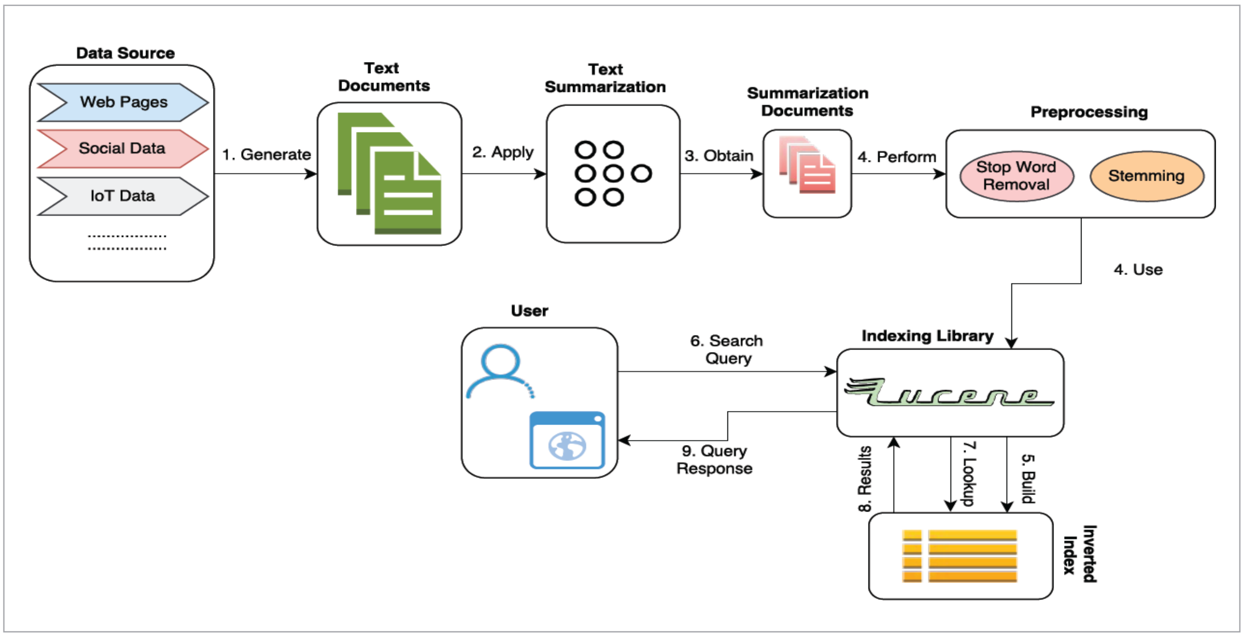
Then we sort the sentences by their similarity scores and identify top sentences that give the summary size less than given τ. We call τ as a summary threshold which defines the user choice of the size of summary required to generate for the given text document.

4.2. Proposed System

Figure 2 shows our proposed system to index data using text summarization and serving user queries. The data indexing process works by aggregating data from different data sources like websites, social networks, server logs, and smart devices. The data from these sources are collected as text documents and passed to the text summarization method which generates summaries for all given documents. Then a preprocessing step is performed which uses NLP methods like stop word removal and stemming to filter insignificant data. Stopword removes all frequent words like a, the, their, we, etc. Stemming reduces the words to their roots which greatly help to minimize the vocabulary of the given documents. Once the pre-processing is done, the important extracted keywords are then passed to the indexing library (Apache Lucene) which prepares an inverted index using the given keywords and document IDs. Once the inverted index is ready, then users can invoke queries using the methods exposed by the indexing library. For the given queries, indexing library

Figure 2

Proposed Big Data index generation and query serving system using text summarization



identifies the related documents, sorts them with ranking scores and returns a list of documents to the users. Generation time and size for both datasets.

5. Experimental Evaluation

In this section, we explain dataset, evaluation criteria, and experiments performed to evaluate the proposed method to index Big Data for full-text search applications. We performed all experiments using a core i7 computer system with 8GB physical memory.

5.1. Datasets and Search Queries

We used two publicly available datasets namely Wikipedia and Project Gutenberg5 to generate their summaries and then indexed them using the proposed system to study the effect of index generation on time and size. We also studied the overlapping and relevance of search results using summarized datasets with the actual dataset. The experimental datasets (actual and summarized) are briefly explained in Table 1. Figure 3 shows the effect of different summary thresholds on index generation time and size for both datasets.

The actual Wikipedia dataset we used is 77 GB in size, consisting of 14.25 million HTML pages and its search index size is 12 GB. We used nine different summarized datasets of Wikipedia datasets with different values of the summary threshold. The Project Gutenberg data is 1.2 GB in size, consisting of 3035 plain text files and its search index size is 301.2 MB. Then we summarized this database using nine different summary thresholds to study the impact on index size, time, overlapping, and relevance.To consider the effect of different search queries for both actual and summarized datasets, we used 200 different queries. For Wikipedia actual and summarized datasets, we used 200 search queries randomly selected from a set of 5000 most frequent search queries of Wikipedia website. For the Project Gutenberg actual and summarized dataset, we used 200 randomly selected nouns from the dataset.

Figure 3

Effect of different summary thresholds on index generation time and size for both datasets

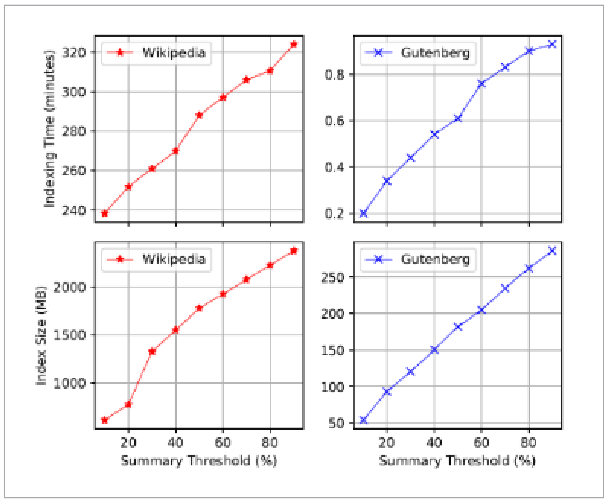


Table 1

Datasets with different summary thresholds. The actual dataset (without summarization) is used as a baseline to compare index size, index generation time, search results overlapping, and relevance with summarized datasets

Data Set	Summary	Dataset Size (GB)	Index Size (GB)	Index Creation Time (mins)
Wikipedia	90	58.98	2.42	323.7
	80	43.89	2.27	310.17
	70	33.88	2.11	305.66
	60	27.72	1.95	296.65
	50	20.79	1.79	287.75
	40	16.17	1.56	269.83
	30	12.32	1.33	261.16
	20	7.7	0.75	251.79
	10	3.85	0.58	238.61
Project Gutenberg	90	1.1	0.28	0.92
	80	0.96	0.26	0.88
	70	0.86	0.23	0.82
	60	0.76	0.2	0.75
	50	0.66	0.18	0.6
	40	0.55	0.15	0.53
	30	0.44	0.12	0.43
	20	0.32	0.09	0.33
	10	0.19	0.05	0.2

5.2. Evaluation Criteria

We used different measures to compare the summarized and the actual datasets for overlapping and relevance of the results. We used simple overlapping, document ranking scores, and Spearman's rho correlation to study the impact of the summarized dataset on search results. In this section, we explain the evaluation measures used in our experimental evaluation.

5.2.1. Simple Overlapping

To compare search results of the summarized dataset with the actual dataset, we use a simple overlapping measure. The simple overlapping measure is a ratio of overlapping search results (documents) for the same queries on both datasets (actual and summarized) for Top 1, Top 5, Top 10, Top 15, and Top 20 search results. We calculate the simple overlapping (S_0) using the following formula:

$$S_o = \frac{R_S}{R}, \quad (2)$$

where R is the number of documents returned by a query from the actual datasets and R_s is the number of intersection documents between results returned for the query from the actual and the summarized dataset.

5.2.2. Ranking Score

To compare the relevance of search results with the queries for both the summarized and the actual datasets, we use ranking score assigned by Apache Lucene to each document in the search result. We compute the average ranking of Top 1, Top 5, Top 10, Top 15, and Top 20 search results on both actual and summarized datasets using same search queries. To assign ranking scores, a combination of Vector Space Model (VSM) and Boolean Model is used.

The Boolean Model identifies the relevant documents on the given queries. Assuming a given set of n documents $D = \{d_1, d_2, ..., d_n\}$, term dictionary of size m is is $T = \{t_1, t_2, ..., t_m\}$, and a boolean expression consisting on k search terms $Q = (q_1 \vee q_2 \vee ... q_k)$. Then Boolean Model identifies lists of documents ∀d and $\forall q$ using $S_l = \{d_i | q_j\}$ and then identifies a list of identical documents $S = \{S_1 \cap S_2 \cap S_3 ... \}$. The list S gives all documents containing the search terms. In the VSM, term frequency-inverse document frequency (tf-idf) is used to identify the ranking of relevant documents given by Boolean Model. For each document d_i in S, a weight vector $v_{di} = [w_{1,di}, w_{2,di}, ..., w_{z,di}]$ is learned. Where:

$$w_{t,d} = (tf_{t,d})(\log \frac{|S|}{|t \in S|}) \quad (3)$$

and $tf_{t,d}$ is term frequency for give term t in document d_i. The remaining part of the Equation 3 is representing inverse document frequency. Then finally ranking score of document d_i is computed using $\Re_{d_i,q}$ for the qiven query q using:

$$\Re_{d_i,q} = \frac{\sum_{i=1}^{z} w_{i,j} . w_{i,q}}{\sqrt{\sum_{i=1}^{z} w^2{}_{i,j}} . \sqrt{\sum_{i=1}^{z} w^2{}_{i,q}}}. \quad (4)$$

Finally, $\Re_{d_i,q}$ is associated with each document d_i and then return to the user in sorted order as a response to the search expression.

5.2.3. Spearman's rho Correlation

We used Spearman's rho [49] to find the correlation between summarized and actual datasets for Top 1, Top 5, Top 10, Top 15, and Top 20 search results. The Spearman's rho works by finding overlapping between two given sets. It ignores non-overlapping members of the set and gives a higher score to higher ranked overlapped results to compute a measure ranging between −1 and 1. The sign of Spearman's rho value shows the direction of overlapping. Since, in our experimental evaluation, we required to identify absolute overlapping between two search results, therefore, we take the absolute value of Spearman's rho measure. The Spearman's rho (S_r) is computed using the following formula:

$$S_r = 1 - \frac{6 \sum {D_i}^2}{N(N^2-1)}, \quad (5)$$

where D_i represents the difference of document ranking between two sets of documents, returned by both actual and summarized datasets for i^{th} query and N is the total number of overlapped documents in both sets.

5.3. Experimental Details

We performed three experiments to evaluate the effectiveness of the summarization method for minimizing index size for full-text search. In all three experiments, we used 200 randomly selected search queries for Wikipedia and Project Gutenberg datasets. In each experiment, we use the actual datasets (without summarization) as a baseline and compare the results with summarized datasets of 200 randomly selected search queries for Top 1, Top 5, Top 10, Top 15, and Top 20 search results.

In Experiment 01, we compare and evaluate average simple overlapping, explained in Section 5.2.1, for the search results obtained on actual and summarized versions for both datasets. In Experiment 02 and Experiment 03, we compare and evaluate the average ranking score, explained in Section 5.2.2, and Spearman's rho measures, described in Section 5.2.3, respectively on the search results.

6. Experimental Results

6.1. Experiment 1: Overlapping Results

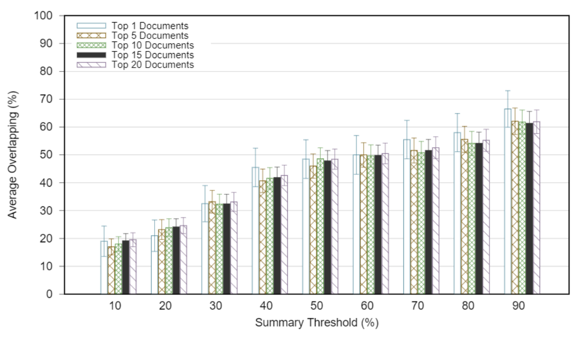
Figure 4(a) shows average overlapping results of summarized datasets using different summary thresholds with baseline Wikipedia dataset for 200 search queries. The overlapping measure shows the summarized version of the Wikipedia dataset provides minimum 18% similar documents returned using 10% summary threshold and maximum 66% similar documents returned using 90% summary threshold. We observe that on increasing value of summary threshold the overlapping results also increased. However, we observe the growth of overlapping results slow down after 40% summary threshold.

Figure 4(b) shows average overlapping results of summarized datasets using different summary thresholds with actual Project Gutenberg datasets for 200 search queries. The overlapping measure shows the summarized version of the Wikipedia dataset provides minimum 52% similar documents returned using 10% summary threshold and maximum 89% similar documents returned using 90% summary threshold. We observe that on increasing value of the summary threshold, the overlapping results also increased. However, we observe the growth of overlapping results slow down after 50% summary threshold.

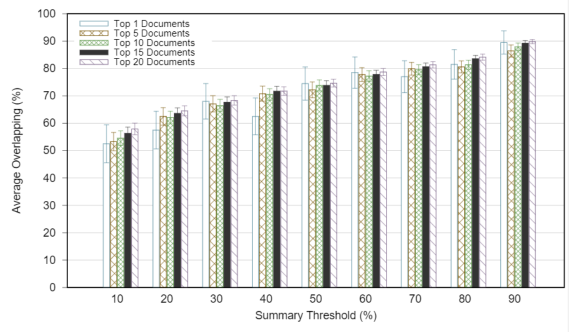
This experimental result shows a correlation between summary threshold and overlapping results.

Figure 4

Experiment 1 results show average overlapping and standard deviation for Wikipedia and Project Gutenberg datasets



(a) Wikipedia



(b) Project Gutenberg

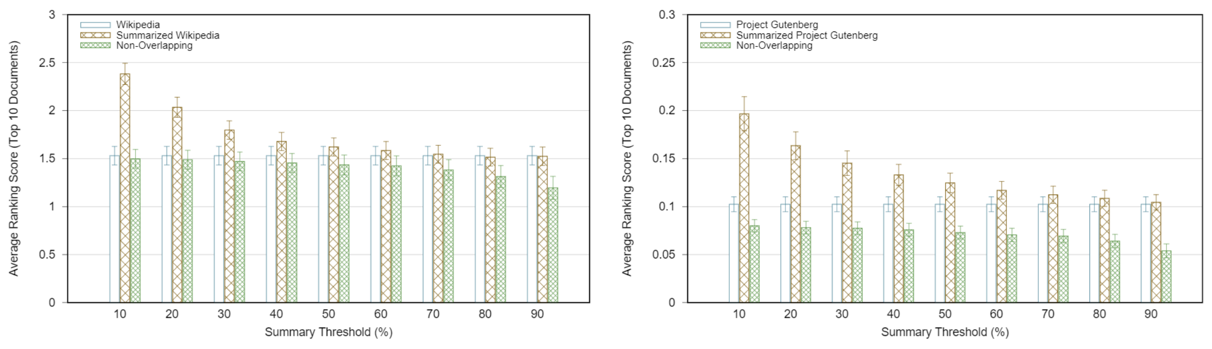
If we use the higher value of summary threshold, we can find more similar results as compared to actual datasets. However, the large value of summary threshold will not help to decrease the index size significantly.

6.2. Experiment 2: Relevance of Search Results

Figure 5(a) shows the average ranking score of summarized datasets using different summary thresholds and actual Wikipedia dataset for 200 search queries. The ranking score for the summarized version of the Wikipedia datasets provides always higher ranking score comparing to actual dataset results. However, the non-overlapping results which are part of actual dataset results but missing from the summarized dataset are low. It shows the non-overlapping results are not highly relevant to the search queries. The maximum score is obtained using 10% summary

Figure 5

Experiment 2 results show average ranking score and standard deviation for Wikipedia and Project Gutenberg datasets



(a) Wikipedia (b) Project Gutenberg

threshold. Overall, the ranking score decreases whenever we increase the value of the summary threshold.

Figure 5(b) shows the average ranking score of summarized datasets using different summary thresholds with actual Project Gutenberg dataset. The ranking score for the summarized version of the Project Gutenberg datasets provides always higher ranking score comparing to actual dataset results. However, the non-overlapping results which are part of actual dataset results but missing from the summarized dataset are low. It shows the non-overlapping results are not highly relevant to the search queries. The maximum score is obtained using 10% summary threshold. Overall, the ranking score decreases whenever we increase the value of the summary threshold.

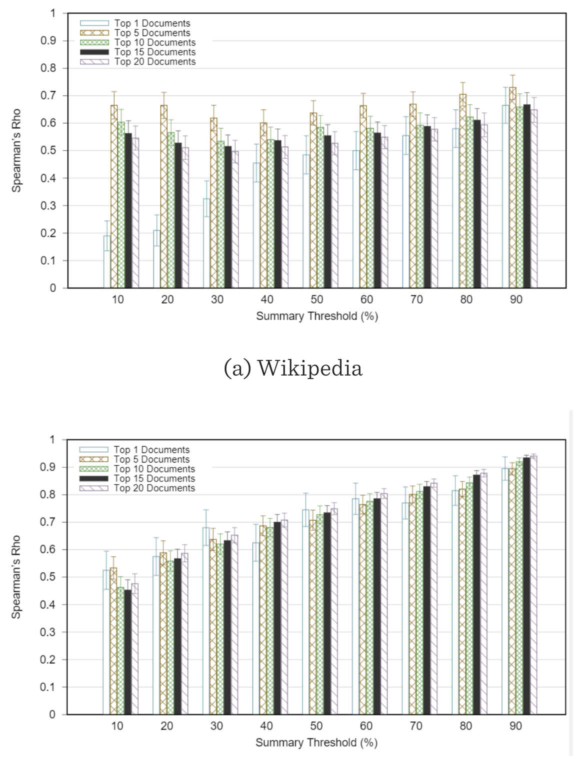
Experiment 2 shows that summarized datasets yield higher ranking scores compared to the actual dataset. Moreover, nonoverlapping results always have lower ranking scores which show less relevance to the search queries.

6.3. Experiment 3: Spearman's Rho Correlation

Figure 6(a) shows average Spearman's rho correlation of ranking scores using summarized and actual datasets for 200 search queries on Wikipedia dataset. The Spearman's rho measure varies between 0.5 and 0.7 for most of the summary thresholds. However, we observed minimum Spearman's rho for Top 1 results obtained using actual dataset and 10% summary threshold. We observed the best results for Top 5

Figure 6

Experiment 3 results show average Spearman's rho correlation and standard deviation for ranking scores of 200 search results



(a) Wikipedia

(b) Project Gutenberg

documents using Spearman's rho measure. There is no specific relationship with increasing summary threshold with Spearman's rho measure, however, the maximum results are observed using 90% summary threshold.

Figure 6(b) shows average Spearman's rho correlation of ranking scores using summarized and actual datasets for 200 search queries on Project Gutenberg. The Spearman's rho measure varies between 0.45 and 0.92. However, we observed minimum Spearman's rho for Top 15 results obtained using actual dataset and 10% summary threshold. We observed the best results for Top 20 documents using Spearman's rho measure on 90% summary threshold. An increasing Spearman's rho trend is observed on increasing summary threshold.

This experiment shows that good correlation, higher than 0.5, exists between search results obtained using actual dataset and summarized results using different summary thresholds.

6.4. Precision and Recall Using Actual and Summarized Dataset

We consider the search results obtained from the actual dataset as ground truth to compute precision and recall of search results obtained using summarization datasets of different summary thresholds. We compute precision and recall using the following formulas:

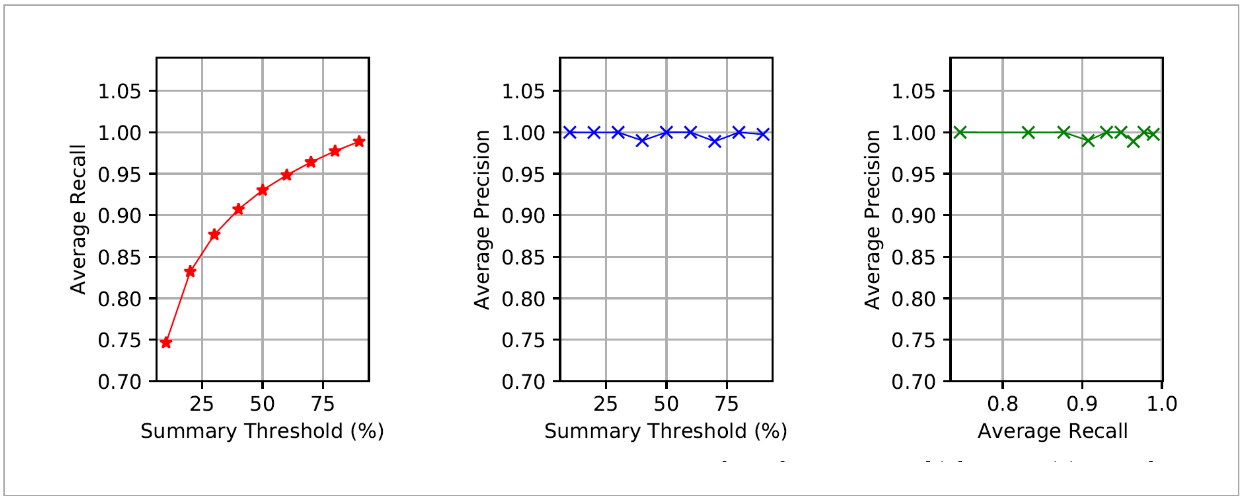
$$\text{Recall} = \frac{R_S}{R} \tag{6}$$

$$\text{Precision} = \frac{R_S}{R_a}, \tag{7}$$

where R is the number of documents returned by a query on the actual dataset, R_a is the total number of documents returned by a query from the summarized dataset, and R_s is a number of similar documents between results returned by a query from the actual and the summarized datasets. Figure 7 shows recall, precision, and the precision-recall graph of all 200 search queries executed on actual Project Gutenberg and summarized datasets. The recall is increasing as number % of summary threshold increases. However, average precision remains very high and close to 1. The high precision results are justified as we are using extractive-based summarization method which selects the sentences rather building new sentences. Therefore, summarized datasets are a subset of the actual dataset and yielding results, which is also a subset of the ground truth. Due to this behavior, we also observe higher precision and recall behavior similar to Figure 7 for the Wikipedia dataset.

Figure 7

Average precision and recall for the different summary threshold for Project Gutenberg dataset. We consider results obtained using actual dataset as ground truth. using actual dataset as ground truth



6.5. Experimental Summary

We summarize our experimental evaluation in Table 2. It shows index size & time decreased, ranking improved, overlapping, and Spearman's rho correlation using different summary thresholds for both datasets in comparison with the baseline datasets (without summarization). The proposed text summarization method yields higher ranks for the search documents returned using both datasets compared to the baseline using a smaller value of the summary threshold. The index size and time decrease significantly using a smaller value of summary thresholds. However, overlapped results with baseline decrease on the smaller values of the summary threshold. We observed a good correlation of search results using all summarized datasets with the baseline.

We recommend using 10% summary threshold to use centroidbased text summarization method to significantly reduce the index size and time. It also improves the search results relevance and provides good ranking correlation with the actual dataset.

In the proposed system, the process of creating summaries for each document introduces CPU time overhead. Due to the linear time execution time complexity of the extractive text summarization algorithm introduce only minimal overhead [36].

Table 2

Experimental summary showing index size and time decreased, average ranking improved, average overlapping results, and average Spearman's rho for different summary threshold for both datasets

Dataset	Summary Threshold %)	Index Size Decreased (%)	Index Time Decreased (%)	Avg Rank Improved (%)	Avg Overlapping (%)	Avg Spearman's Rho
Wikipedia	**10**	**75**	**47**	**35.77**	**18.55**	**0.52**
	20	69	44	24.79	23.38	0.51
	30	47	42	14.86	32.72	0.51
	40	38	40	8.82	42.46	0.53
	50	29	36	5.61	47.89	0.56
	60	23	34	3.41	50.00	0.57
	70	17	32	0.99	52.42	0.60
	80	11	31	-0.98	55.43	0.62
	90	05	28	-0.38	62.73	0.68
Project Guetenberg	**10**	**82**	**80**	**47.92**	**54.92**	**0.49**
	20	69	66	37.33	62.05	0.58
	30	60	56	29.47	67.51	0.64
	40	50	46	22.95	69.45	0.68
	50	38	39	17.82	73.80	0.73
	60	32	24	12.41	78.02	0.78
	70	22	17	8.75	79.69	0.81
	80	13	10	5.67	82.23	0.85
	90	05	07	1.94	88.58	0.92

An online system, which creates a summary for new text documents instantly, will not introduce any notable performance issue. However, for a batch processing system, in which a large dataset required to generate the summaries, we recommend using Hadoop and Spark [21] to parallel process the large datasets for reducing the overhead of creating the summaries.

7. Conclusion and Future Work

Providing an efficient text search services for a large dataset is an interesting research topic. In this paper, we used an automatic text summarization method based on an extractive approach to reducing the index size of large datasets. Our experimental evaluation shows a maximum of 82% reduction in the index size and 80% in index generation time when using text summarization method with 10% summary threshold. The relevance of search results obtained from summarized datasets is higher than baseline datasets. Moreover, the correlation between search results is good. However, the best overlapping results are 54% using the Project Gutenberg dataset. Automatic text summarization is an effective method to help to reduce the index size significantly with the better relevance of the search results.

We are currently identifying the best threshold in extractive-based text summarization method to improve the overlapping results with the actual dataset. Moreover, we are planning to build a cloud-based application using Apache Lucene to provide full-text search index minimization services.

Acknowledgment

The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges (APC) of this publication.

References

1. Abe, H., Shima, K., Sekiya, Y., Miyamoto, D., Ishihara, T., Okada, K. Hayabusa: Simple and Fast Full-Text Search Engine for Massive System Log Data. Proceedings of the 12th International Conference on Future Internet Technologies. ACM, 2017, 2.
2. Alguliyev, R. M., Aliguliyev, R. M., Isazade, N. R., Abdi, A., Idris, N. Cosum: Text Summarization Based on Clustering and Optimization. Expert Systems, 2019, 36(1), e12340.
3. Amalina, F., Hashem, I. A. T., Azizul, Z. H., Fong, A. T., Firdaus, A., Imran, M., Anuar, N. B. Blending Big Data Analytics: Review on Challenges and a Recent Study. IEEE Access, 2019.
4. Bar-Ilan, J., Mat-Hassan, M., Levene, M. Methods for Comparing Rankings of Search Engine Results. Computer Networks, 2006, 50(10), 1448-1463.
5. Buttcher, S., Clarke, C. L., Cormack, G. V., 2016. Information Retrieval: Implementing and Evaluating Search Engines. MIT Press.
6. Christopher, D. Manning, P. R., Schutze, H. Introduction to Information Retrieval. Cambridge University Press, Ch. 4,5, 2008.
7. Claude, F., Farina, A., Martınez-Prieto, M. A., Navarro, G. Universal Indexes for Highly Repetitive Document Collections. Information Systems, 2016, 61, 1-23.
8. Cohen, J. D. Highlights: Language-and Domain-Independent Automatic Indexing Terms for Abstracting. Journal of the American Society for Information Science, 1995, 46(3), 162.
9. Cuggia, M., Bayat, S., Garcelon, N., Sanders, L., Rouget, F., Coursin, A., Pladys, P. A Full-Text Information Retrieval System for an Epidemiological Registry. Studies in health technology and informatics, 2010, 160 (Pt 1), 491-495.
10. Cutting, D. Apache Lucene - Apache Lucene core. URL https://lucene.apache.org/core
11. Dahale, M. Text Summarization for Compressed Inverted Indexes and Snippets. SJSU ScholarWorks, 2014.
12. Gambhir, M., Gupta, V. Recent Automatic Text Summarization Techniques: A Survey. Artificial Intelligence Review, 2017, 47 (1), 1-66.
13. Gani, A., Siddiqa, A., Shamshirband, S., Hanum, F. A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation. Knowledge and Information Systems, 2016, 46(2), 241-284.
14. Gao, S., Chen, X., Li, P., Ren, Z., Bing, L., Zhao, D., Yan, R. Abstractive Text Summarization by Incorporating Reader Comments. Proceedings of the AAAI Conference on Artificial Intelligence, 2019, 33, 6399-6406.

15. Garcelon, N., Neuraz, A., Benoit, V., Salomon, R., Burgun, A. Improving a Full-Text Search Engine: The Importance of Negation Detection and Family History Context to Identify Cases in a Biomedical Data Warehouse. Journal of the American Medical Informatics Association, 2016, 24(3), 607-613.

16. Ghose, A., Ipeirotis, P. G., Li, B. Examining the Impact of Ranking on Consumer Behavior and Search Engine Revenue. Management Science, 2014, 60(7), 1632-1654. https://doi.org/10.1287/mnsc.2013.1828

17. Goncalo Oliveira, H., Filipe, R., Rodrigues, R., Alves, A. Using Lucene for Developing a Question-Answering Agent in Portuguese. 8th Symposium on Languages, Applications and Technologies (SLATE 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

18. Grainger, T., Potter, T. Solr in Action, 1st Edition. Manning Publications Co., Greenwich, CT, USA, 2014.

19. Hanauer, D. A., Wu, D. T., Yang, L., Mei, Q., Murkowski-Steffy, K. B., Vydiswaran, V. V., Zheng, K. Development and Empirical Usercentered Evaluation of Semantically-Based Query Recommendation for an Electronic Health Record Search Engine. Journal of Biomedical Informatics, 2017, 67, 1-10. https://doi.org/10.1016/j.jbi.2017.01.013

20. He, Y., Xiang, S., Kang, C., Wang, J., Pan, C. Cross-Modal Retrieval via Deep and Bidirectional Representation Learning. IEEE Transactions on Multimedia, 2016, 18(7), 1363-1377. https://doi.org/10.1109/TMM.2016.2558463

21. Iqbal, B., Iqbal, W., Khan, N., Mahmood, A., Erradi, A. Canny Edge Detection and Hough Transform for High Resolution Video Streams Using Hadoop and Spark. Cluster Computing, 2020, 23 (1), 397-408. https://doi.org/10.1007/s10586-019-02929-x

22. Jansen, B. J., Spink, A. An Analysis of Web Searching by European Alltheweb.com Users. Information Processing and Management, 2005, 41 (2), 361-381. https://doi.org/10.1016/S0306-4573(03)00067-0

23. Jansen, B. J., Spink, A., Pedersen, J. A Temporal Comparison of Altavista Web Searching: Research Articles. Journal of the Association for Information Science and Technology, 2005, 56(6), 559-570. https://doi.org/10.1002/asi.20145

24. Jing, Y., Zhang, C., Wang, X. An Empirical Study on Performance Comparison of Lucene and Relational Database. International Conference on Communication Software and Networks, ICCSN'09, 2009, 336-340. https://doi.org/10.1109/ICCSN.2009.96

25. Knight, K., Marcu, D. Summarization Beyond Sentence Extraction: A Probabilistic Approach to Sentence Compression. Artificial Intelligence, 2002, 139(1), 91-107. https://doi.org/10.1016/S0004-3702(02)00222-9

26. Kononenko, O., Baysal, O., Holmes, R., Godfrey, M. W. Mining Modern Repositories with Elasticsearch. Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, ACM, New York, NY, USA, 2014, 328-331. https://doi.org/10.1145/2597073.2597091

27. Labrinidis, A., Jagadish, H. V. Challenges and Opportunities with Big Data. Proceedings of the VLDB Endowment, 2012, 5(12), 2032-2033. https://doi.org/10.14778/2367502.2367572

28. Ledeneva, Y., Gelbukh, A., Garcıa-Hernandez, R. Terms Derived from Frequent Sequences for Extractive Text Summarization. Computational Linguistics and Intelligent Text Processing, 2008, 593-604. https://doi.org/10.1007/978-3-540-78135-6_51

29. McCandless, M., Hatcher, E., Gospodnetic, O. Lucene in Action, Second Edition: Covers Apache Lucene 3.0. Manning Publications Co., Greenwich, CT, USA, 2010.

30. Mihalcea, R., Tarau, P. Textrank: Bringing Order Into Texts. In: Lin, D., Wu, D. (Eds.), Proceedings of EMNLP 2004. Association for Computational Linguistics, Barcelona, Spain, 2004, 404-411.

31. Oweis, N. E., Owais, S. S., George, W., Suliman, M. G., Snasel, V. A Survey on Big Data, Mining: (Tools, Techniques, Applications and Notable Uses). In: Abraham, A., Jiang, X. H., Snasel, V., Pan, J.-S. (Eds.), Intelligent Data Analysis and Applications. Springer International Publishing, Cham, 2015, 109-119. https://doi.org/10.1007/978-3-319-21206-7_10

32. Prasad, A., Patel, D. Lucene Search Engine: An Overview. DRTCHP International, 2005, 10.

33. Radev, D. R., Jing, H., Stys, M., Tam, D. Centroid-Based Summarization of Multiple Documents. Information Processing and Management, 2004, 40(6), 919-938. https://doi.org/10.1016/j.ipm.2003.10.006

34. Ramos, J. Using Tf-Idf to Determine Word Relevance in Document Queries. Proceedings of the First Instructional Conference on Machine Learning, 2003, 242, 133-142.

35. Salton, G., Singhal, A., Mitra, M., Buckley, C. Automatic Text Structuring and Summarization. Information Processing and Management, Methods and Tools for the Automatic Construction of Hypertext, 1997, 33(2), 193-207. https://doi.org/10.1016/S0306-4573(96)00062-3

36. Shao, L., Zhang, H., Jia, M., Wang, J. Efficient and Effective Single Document Summarizations and a Word-Embedding Measurement of Quality, 2017. arXiv preprint arXiv:1710.00284. https://doi.org/10.5220/0006581301140122

37. Silverstein, C., Marais, H., Henzinger, M., Moricz, M. Analysis of a Very Large Web Search Engine Query Log. SIGIR Forum, 1999, 33(1), 6-12. https://doi.org/10.1145/331403.331405

38. Spink, A., Ozmutlu, S., Ozmutlu, H. C., Jansen, B. J. US Versus European Web Searching Trends, 2002, 36(2), 32-38. https://doi.org/10.1145/792550.792555

39. Steinbach, M., Karypis, G., Kumar, V. A Comparison of Document Clustering Techniques. In KDD Workshop on Text Mining, 2000.

40. Totaro, G., Bernaschi, M., Carbone, G., Cianfriglia, M., Di Marco, A. Isodac: A High Performance Solution for Indexing and Searching Heterogeneous Data. Journal of Systems and Software, 2016, 118, 115-133. https://doi.org/10.1016/j.jss.2015.11.043

41. Tsai, C.-W., Lai, C.-F., Chao, H.-C., Vasilakos, A. V. Big Data Analytics: A Survey. Journal of Big Data, 2015, 2(1), 21. https://doi.org/10.1186/s40537-015-0030-3

42. Wang, J., Lin, C., He, R., Chae, M., Papakonstantinou, Y., Swanson, S. Milc: Inverted List Compression in Memory. Proceedings of VLDB Endow, 2017, 10(8), 853-864. https://doi.org/10.14778/3090163.3090164

43. Wang, J., Zhang, T., Song, J., Sebe, N., Shen, H. T. A Survey on Learning to Hash. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018, 40(4), 769-790. https://doi.org/10.1109/TPAMI.2017.2699960

44. Wang, M., Holub, V., Murphy, J., OSullivan, P. High Volumes of Event Stream Indexing and Efficient Multi-Keyword Searching for Cloud Monitoring. Future Generation Computer Systems, 2013, 29(8), 1943-1962. https://doi.org/10.1016/j.future.2013.04.028

45. Williams, K., Giles, C. L. Improving Similar Document Retrieval Using a Recursive Pseudo Relevance Feedback Strategy. 2016 IEEE/ACM Joint Conference on Digital Libraries (JCDL), 2016, 275-276. https://doi.org/10.1145/2910896.2925468

46. Wu, T., Shyng, H., Chou, J., Dong, B., Wu, K. Indexing Blocks to Reduce Space and Time Requirements for Searching Large Data Files. 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, 398-402. https://doi.org/10.1109/CCGrid.2016.18

47. Yan, H., Ding, S., Suel, T. Inverted Index Compression and Query Processing with Optimized Document Ordering. Proceedings of the 18th International Conference on World Wide Web. WWW'09, ACM, New York, NY, USA, 2009, 401-410. https://doi.org/10.1145/1526709.1526764

48. Yousefi-Azar, M., Hamey, L. Text Summarization Using Unsupervised Deep Learning. Expert Systems with Applications, 2017, 68, 93-105. https://doi.org/10.1016/j.eswa.2016.10.017

49. Zar, J. H. Spearman Rank Correlation. Encyclopedia of Biostatistics, 1998.

50. Zhang, J., Long, X., Suel, T. Performance of Compressed Inverted List Caching in Search Engines. Proceedings of the 17th International Conference on World Wide Web. WWW '08. ACM, New York, NY, USA, 2008, 387-396. https://doi.org/10.1145/1367497.1367550

51. Zhu, X., Huang, Z., Cheng, H., Cui, J., Shen, H. T. Sparse Hashing for Fast Multimedia Search. ACM Transactions on Information Systems (TOIS), 2013, 31(2), 9. https://doi.org/10.1145/2457465.2457469