Improving Time Efficiency of TF-IDF Algorithm for Dynamic Data Streams

Sidra Saher, Qurat-ul-Ain Akram*, Kashif Javed and Sarmad Hussain
Al-Khawarizmi Institute of Computer Science
(University of Engineering and Technology, Lahore)
{sidra.sehar, ainie.akram, kashif.javed, sarmad.hussain}@kics.edu.pk*

Abstract

Different applications including search engines, plagiarism detection systems and recommender systems need to crunch the data frequently after a specific period of time. The data indexing and retrieval for such applications is becoming popular research area due to availability of huge electronic content through Internet, which is growing rapidly on daily basis. TF-IDF based term weighting scheme is commonly used to extract features of a document which are used for relevant document searching. In this paper, TF-IDF algorithm is analyzed and an efficient implementation of TF-IDF algorithm is proposed to handle such dynamic text data. Two major improvements of the traditional TF-IDF algorithm are proposed; (1) Algorithm-1: Expansion of IDF using logarithm properties and (2) Algorithm-2: Store lookup of term frequency, document frequency and IDF for reuse in next batch and efficiently update IDF of terms. The systems are evaluated on dataset of 100,000 Urdu documents. Traditional TF-IDF algorithm takes 2676520 ms (10256 ms for IDF calculation) to process complete dataset. Algorithm-1 takes 2676272 ms (10008 ms for IDF calculation), showing time efficiency in IDF calculation. Algorithm-2 is specifically designed to handle the dynamic growth of the data, which calculates the new IDF of a term using the previously computed IDF. This algorithm takes 707823 ms to process 100,000 documents. Another contribution of this study involves reduced memory consumption for TF-IDF vectors using the sparse representation of vectors. This reduces 8,945,445 MB to 353 MB to store TF-IDF of 11,724,976 terms computed from 100,000 Urdu documents.

1. Introduction

Due to rapid advancement in the development of Information and Communication Technologies (ICTs), electronic content is easily accessible in the form of blogs, online articles, books, magazines, newspapers, research papers and theses etc. To provide real time accessibility of the relevant content, an efficient information retrieval system is important. The research in the fields of data mining, search engines and similarity computation among documents for plagiarism detection is becoming popular to handle huge dynamic text data which is increasing on a daily basis. Processing of such large dataset is a challenging task. To develop such applications, text data is processed in such a way so that meaningful information can be extracted efficiently.

Various term weighting schemes such as Mutual Information, Okapi, LTU [1-3]and TF-IDF are used to process massive datasets. Among all these schemes, TF-IDF based term weighting scheme is commonly used technique [4] to extract features of the document which are used for searching the relevant document, text mining and text classification etc.

To process the text for similarity computation, data is processed and converted into the structured form which can easily be used for similarity computation. For this, three structures are used for document representation, which include inference network [5], probabilistic [6] and Vector Space Model (VSM). Among all these models, VSM is most widely used model [7]. The VSM contains a vector representation of weighted terms of a document.

There are two ways to find the similarity between the documents; (1) Local similarity and (2) Global similarity. Local similarity refers to the technique which tries to find the sequence of words which are same in two documents. Usually cosine similarity based on vector space model of terms (*n-grams, $n>1$*) is used to find the similarity between two documents. However, Global similarity based techniques rely on the unigram do not cover the context. The similarity results of such technique are meaningless in cases when two documents have exactly similar terms but their sequence is never the same. Therefore, researchers prefer to use local similarity for applications which requires sequence of the words to be matched [8].

For huge datasets, term weighting schemes are applied for query retrieval or similarity calculation purposes. A lot of research in this area has emphasized on importance of using phrases as terms. Due to improved accuracy of results, bigram and trigram based phrases are preferred over unigrams [9]. In case of Global similarity [8], similarity score between two documents is highly unreliable. Therefore, we cannot predict the documents to be exactly similar irrespective of high similarity score. Due to this reason, trigrams are used as terms for enhanced accuracy of the system.

To develop real time system for search engine, plagiarism detection system etc. huge amount of dataset is required. In addition, it is also essential to find the document similarity with minimum time. Hence, term weighting scheme needs to be implemented efficiently. The re-computation of term weights in case of dynamically growing data on daily basis is also challenging, even if there is small change in the actual content. The major contributions of this study are given below

- Algorithm-1 incorporates an efficient implementation of TF-IDF calculation for huge data.
- Algorithm-2 is an efficient implementation of TF-IDF calculation for dynamically growing data on daily basis.
- The memory consumption of TF-IDF vectors is also reduced by using sparse representation of vectors instead of dense vector representation.

The rest of the paper is organized as follows: Section 2 describes related work. Section 3 provides explanation of proposed algorithm by using logarithm properties and lookup storage to improve computational time of TF-IDF. Section 4 involves a brief discussion of dataset used for experimentation. The experimental evaluation and their results are discussed in Section 5 and finally Section 6 concludes research study.

2. Related work

The idea of inverse document frequency was first proposed by Jones [10]. Jones emphasized the idea of specificity by introducing the concept of collection frequency i.e. the words which are less frequent in a collection should have more weight [10]. Salton and Yu [11] used same technique in information retrieval and observed retrieval effectiveness by using precision and recall as evaluation metrics [11]. In addition, Salton and Yu [11] also figured out that the TF-IDF algorithm performed well in document retrieval. TF-IDF is a term weighting scheme which assigns weights to terms depending on their significance in the corpus. Its major principle is based on the fact that a more frequent term in a document with less frequency in overall documents (i.e. document frequency) will have high TF-IDF weight and vice-versa.

The calculation of TF-IDF is carried out using (1) which is based on the term frequency (TF) and inverse document frequency (IDF), and are calculated by using following equations

$$TF-IDF(t,d) = TF(t,d) * IDF(t) \quad (1)$$

Term frequency is calculated as follows

$$TF(t,d) = count\ of\ t\ in\ d \quad (2)$$

Where *t, d* stands for term (i.e. trigrams in this study) and document respectively. The IDF is calculated using (3)

$$\mathrm{IDF(t)} = \log\left(\frac{\mathrm{TotalDocSize}}{\mathrm{DF(t)}}\right) \quad (3)$$

Where, TotalDocSize is the total number of documents in the corpus, which is processed to compute the TF-IDF of all terms in corpus. In addition the DF is the Document Frequency which is computed using (4).

$$\mathrm{DF(t)} = \sum_{\mathrm{i=1}}^{\mathrm{TotalDocSize}} \mathrm{td} \quad \begin{cases} \mathrm{td} = 1 & \text{if t in } \mathrm{d_i} \\ \mathrm{td} = 0 & \text{otherwise} \end{cases} \quad (4)$$

Where *td* denotes the presence of a term in a document.

The calculation of the IDF is further improved by using the concept of smoothing introduced in [12] as can be seen in (5).

$$\mathrm{IDF(t)} = 1 + \log\left(\frac{\mathrm{TotalDocSize+1}}{\mathrm{DF(t)+1}}\right) \quad (5)$$

TF-IDF is extensively explored to utilize TF-IDF for text classification, document retrieval and plagiarism detection systems [13-15]. A lot of research is done to improve TF-IDF algorithm in terms of accuracy [14, 16, 17]. In addition, a few efficiency improvements of the TF-IDF are also suggested. Bin and Yuan [18] presented a technique for the efficient computation of TF-IDF from large data using Hadoop as main framework. Hadoop supports data distribution on multiple machines. In addition Map/Reduce scheme was also used for fast calculation of TF-IDF. The main shortcoming of the work [18] is that the data is assumed to be static which means data will not be updated once it would be indexed. Gu et al. [19] used parallel cloud computing framework which is based on GPU and MapReduce to improve the efficiency of TF-IDF algorithm.

3. Methodology

In context of document relevance, n-grams based similarity calculation is used to provide very accurate results with emphasis on use of trigrams as a term [9]. TF-IDF algorithm uses VSM to represent TF-IDF computed over each document in complete corpus. In this study, an efficient implementation of TF-IDF is proposed. As a first step, Traditional Algorithm of TF-IDF calculation is implemented with efficient lookup of term frequency and document frequency. Algorithm-1 is proposed which involves improvement in time by use of logarithmic expansion for IDF formula. Most of the real world applications like search engines, plagiarism detection systems and other information retrieval applications have dynamically growing data. Addition of a few more documents compels recalculation of weights of all terms in all the documents. Therefore, Algorithm-2 is designed in such a way that when a huge corpus is indexed using TF-IDF term weighting scheme. The term frequency, document frequency and inverse document frequency lookups are stored. These lookups are then used to update TF-IDF on increment of new documents. In this way, term frequency of only newly updated documents is calculated. The IDF of new terms is calculated using document frequency lookup. The terms which do not occur in newly updated documents undergo a slight update in their already computed IDF values. It reduces the overhead of re-computation and improves efficiency. The overhead of memory is also reduced by conversion of dense representation of TF-IDF vector to sparse representation.

3.1. Traditional TF-IDF Algorithm

A very simple implementation for TF-IDF calculation is carried out in [20]. The implementation of this algorithm is computationally analyzed and an efficient implementation is proposed to reduce the redundant calculations. In order to reduce computational time, TF and DF for all documents is calculated within the same iteration over all documents. This reduces a lot of computational time. Separate lookup is maintained for each document to store the TFs. For document frequency a global DF lookup is created to maintain the document frequency of each term.

Once, the complete global DF lookup is maintained over all the documents, another lookup is also maintained for IDF (using (5)) by iterating over global DF lookup. For TF-IDF calculation, it will iterate over each term of every document just once and use the values of TF and IDF stored in respective lookups.

3.2. Algorithm-1

The original equation for IDF i.e. (5) is analyzed further. By applying the logarithm this computation is further reduced as can be seen in (6).

$$\mathrm{IDF(t)} = 1 + \log(\mathrm{TotalDocSize} + 1) - \log(\mathrm{DF(t)} + 1) \tag{6}$$

As can be seen in (6), $(1 + \log(\mathrm{TotalDocSize} + 1))$ will be calculated only once for all the terms of all the documents. In addition $\log(\mathrm{DF(t)} + 1)$ requires to be computed for each term and another subtraction operation is required. In addition, running time of division operation for n-digit number is $O(n^2)$ and running time of subtraction operation is $O(n)$, thus saving more computational effort [21].

In traditional TF-IDF algorithm, the expression $log((TotalDocSize + 1)/(DF(t) + 1))$ is calculated P times i.e. time complexity is *O(P)*, where P denotes total number of terms in global term lookup but due to this algorithmic modification redundant calculation of $\log(\mathrm{TotalDocSize} + 1))$ will be reduced to single time calculation i.e. time complexity is reduced to *O(1)*.

3.3. Algorithm-2

Various real time information retrieval and online text similarity based applications such as plagiarism detection system and search engines are based on indexing of huge data. Majority of these systems are developed to handle dynamic data, resulting in incorporation of the results computed on the newly indexed and already indexed data. This is usually carried out by indexing the complete data (new and old documents). Once, a significant amount of data is processed (e.g. 5.5 million documents) then number of new documents is minimum may be 1-3% of the already indexed content. The document metadata information (document name, URL, last modified date, etc.) is also maintained while developing such huge systems. Therefore, before starting indexing of complete dataset (existing and new), document filtering can be applied on recently crawled data to filter all those documents which are not indexed previously based on metadata information. These documents are referred to as NewBatch and remaining documents which are already indexed documents are referred to as PreviousBatch. The NewBatch and PreviousBatch terminology is used throughout this paper.

In order to re-index complete data including data of PreviousBatch and NewBatch, a lot of computational effort and time will be utilized. Although, the number of documents in NewBatch is minimal. Based on analysis, a term can be categorized into one of the following categories

- New Terms: Terms which are only present in NewBatch.
- Common Terms: Terms which are present in both PreviousBatch and NewBatch.
- Old terms: Terms which are only present in PreviousBatch.

For new terms, term frequency, document frequency and inverse document frequency are computed using (2), (4) & (6) respectively, from documents in NewBatch. The DFs of terms which are common between PreviousBatch and NewBatch are computed from NewBatch and will be added in DFs of respective terms already stored in the respective DF lookup of PreviousBatch. Then, the IDF and TF-IDF are computed using the same traditional way.

Inverse document frequency of old terms which are only present in PreviousBatch is not required to be recomputed for obvious reasons. As document frequency lookup, inverse document frequency lookup and term frequency lookup of each term of the PreviousBatch are also maintained and stored separately to minimize the re-computation time. Thus IDFs of old terms can be calculated from already computed IDFs in PreviousBatch by addition of an expression dependent only on document size of PreviousBatch and NewBatch.

In order to calculate the IDF for old terms during re-indexing of the dynamically growing data, the respective IDFs of the PreviousBatch are processed in such a way that DocSize of the NewBatch denoted with NewDocSize, is incorporated. More precisely the IDFs are calculated using (6). Equation 6 for PreviousBatch can be written as follows

$$\mathrm{IDF_{Previous}} = 1 + \log\big((\mathrm{PreviousDocSize} + 1)\big) - \log(\mathrm{DF(t)} + 1) \qquad (7)$$

We can modify (7) for incorporating TotalDocSize when NewBatch is indexed with PreviousBatch.

$$\mathrm{IDF_{New}} = 1 + \log\big((\mathrm{PreviousDocSize} + 1) + \mathrm{x}\big) - \log(\mathrm{DF(t)} + 1) \qquad (8)$$

Where *PreviousDocSize* denotes the number of documents in PreviousBatch and *x* denotes the number of documents in NewBatch.

The term $\log\big((\mathrm{PreviousDocSize} + 1) + \mathrm{x}\big)$ can be further solved by simplifying the expression in logarithm. Let *k* denotes the term $(\mathrm{PreviousDocSize} + 1)$, then the term $log(k + x)$ can be written as [22]

$$log(\mathrm{k} + \mathrm{x}) = log\left(\mathrm{k}\left(1 + \frac{\mathrm{x}}{\mathrm{k}}\right)\right)$$

Substituting value of *k* in (8), following expression is obtained

$$\mathrm{IDF_{New}} = 1 + log\left((\mathrm{PreviousDocSize} + 1) \times \left(1 + \frac{\mathrm{x}}{(\mathrm{PreviousDocSize+1})}\right)\right) - \log(\mathrm{DF(t)} + 1) \qquad (9)$$

Using Multiplicative property of Logarithm in (9)

$$\mathrm{IDF_{New}} = 1 + \log\left(1 + \frac{\mathrm{x}}{(\mathrm{PreviousDocSize} + 1)}\right) + \log(\mathrm{PreviousDocSize} + 1) - \log(\mathrm{DF(t)} + 1)$$

$$= 1 + \log(\mathrm{PreviousDocSize} + 1) - \log(\mathrm{DF(t)} + 1) + \log\left(1 + \frac{\mathrm{x}}{(\mathrm{PreviousDocSize} + 1)}\right)$$

Clearly, the bold part is the same as (7). It can be replaced with Previous IDF value.

$$\mathrm{IDF_{New}} = \mathrm{IDF_{Previous}} + \log\left(1 + \frac{\mathrm{x}}{(\mathrm{PreviousDocSize} + 1)}\right) \qquad (10)$$

Equation 10 shows that we can update IDF of old terms by addition of an expression dependent on NewDocSize and PreviousDocSize. This expression needs to be calculated only once before update of IDF of all the old terms.

The main advantage of using this approach is that instead of calculating IDF of old terms incorporating the total document size, we just need to calculate the expression $\log\left(1 + \frac{\mathrm{x}}{(\mathrm{PreviousDocSize+1})}\right)$ only once which is based on *PreviousDocSize* and *x* i.e. NewDocSize. This approach works well for vast dynamic data streams when data is being updated frequently. In such cases IDFs can be efficiently updated without any redundant re-computations.

3.4. Avoiding Extra Memory Consumption

Next step involves intelligent representation of TF-IDF vector so that it occupies less space in memory. The TF-IDF vector size for each document is the total terms computed from the complete dataset. Since each document does not contain all the terms therefore majority of the terms contain zero in a document. To solve this issue, dense representation of TF-IDF vector of a document is converted to sparse by excluding terms having TF-IDF value of zero. An example of dense to sparse vector representation is given in Fig.1.

$$dense: [5.0\ \ 0.0\ \ 0.0\ \ 4.0\ \ 0.0\ 2.0]$$

$$Sparse = \begin{cases} indices: 0,3,5 \\ values: 5.0\ 4.0\ 2.0 \end{cases}$$

Fig.1 Dense and sparse representation

4. Dataset

Traditional Algorithm, Algorithm-1 and Algorithm-2 are tested on dataset of Urdu web pages and results are evaluated.

5.7 Million Urdu web pages are crawled [23]. Their dataset is not publically available as it is crawled from various authenticated Urdu websites. A subset of this dataset is selected for the performance evaluation of the proposed algorithms. Therefore, 0.1 Million Urdu documents are used for testing. The detailed statistics of selected data is given in Table 1.

Table 1. Data statistics

Total Documents	100000
Average Words per Document	438
Average Lines per Document	10
Average words per line	34

5. Experiments and Results

Urdu has space insertion and deletion issues. Hence, unlike English, the words cannot be extracted by processing the space. To handle this issue, a pre-processing is applied on the complete dataset to resolve such issues. Urdu word segmentation is applied to the dataset used for evaluation purpose, which converts the sequence of Urdu ligature to the best sequence of Urdu words of a sentence.

In addition, pre-processing is applied which involves normalization, diacritics and punctuation marks removal. Then content of a document is processed and trigrams as terms are extracted and stored so that TF-IDF weighting can be applied.

As a first experiment, 100,000 documents are processed to compare the performance of Traditional Algorithm and Algorithm-1. The results are given in Fig.2. As can be seen in Fig.2, the Algorithm-1 outperforms Traditional Algorithm with efficient computation results using the properties of logarithm. Traditional Algorithm takes 2676520 ms to process complete dataset for the computation of IDF, whereas Algorithm-1 takes 2676272 ms to process same dataset for IDF calculations.

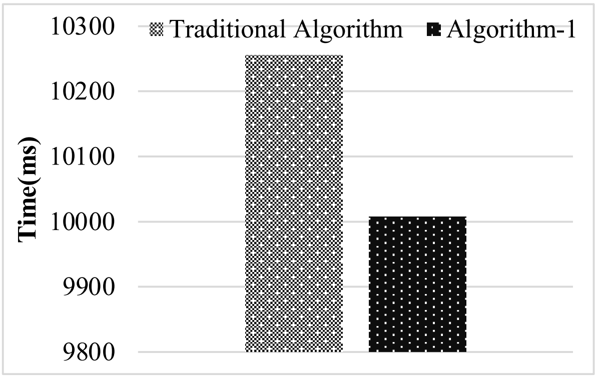


Fig. 2. Time comparison for IDF calculation

As Algorithm-1 reduces redundant computations of expression $log((TotalDocSize + 1)/(DF(t) + 1)$ by employing logarithm properties, Moreover, it converts division operation for IDF computation of each term to subtraction operation. So, the difference between the time taken for IDF calculation between Traditional Algorithm and Algorithm-1 is somehow evident.

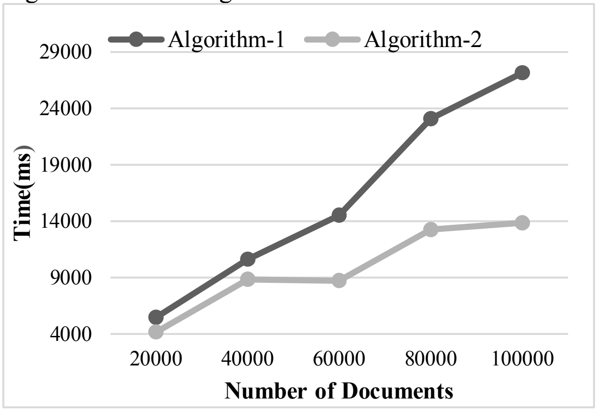


Fig.3. Time comparison for IDF calculation

The second experiment is carried out to find the difference between execution time of Algorithm-1 and Algorithm-2. While comparing IDF calculation time of Algorithm-1 with Algorithm-2, we will also include the execution time for document frequency calculation along with IDF calculation time because we store the lookups of document frequency along with IDF after the execution of each batch. Due to this reason, redundant document frequency and IDF calculation for common and old terms in PreviousBatch and NewBatch are minimized. This technique also reduces a lot of computational effort and time.

By visualizing the trend in the graph as can be seen in Fig. 3, it can be observed that increment of 20,000 documents within each new batch results in increased execution time for both algorithms. However, this increment in execution time is very mild in case of Algorithm-2 and very rapid in case of Algorithm -1.

Another point worth noticing is the trend of trigrams within each batch. The total number of unique trigrams in 100,000 documents is given by about 11 Million. If each batch introduces 20,000 new documents.

Table 2. Number of trigrams in each batch

Document Batches	*Total Trigrams*	*New Trigrams in current batch*	*Trigrams common with previous batch*	*Trigrams only in previous batch (%of total trigrams)*
Batch1(20,000)	3,239,453	None	None	None
Batch2(40,000)	6,238,318	2,998,865	732,171	2,507,282 (41%)
Batch3(60,000)	7,558,982	1,320,664	319,048	5,919,270 (78%)
Batch4(80,000)	9,743,136	2,184,154	833,066	6,725,916 (69%)
Batch5(100,000)	11,724,976	1,981,840	715,117	9,028,019 (77%)

There must be some terms which are not present in PreviousBatch and hence their IDF is calculated using (6). In our experiment 26% new terms are introduced in each batch on average. Some terms are present in previous and new batch as well hence their document frequency is updated and then their IDF is calculated. In our experiment, on average 7% common terms are generated with each NewBatch. However, the old terms are actually not being used in NewBatch and hence their document frequency does not change compelling us to update their IDF using the expression $\log\left(1+\frac{x}{(\mathrm{PreviousDocSize}+1)}\right)$. For each new batch, the IDF of 66% of total trigrams need to be updated using (10) on average. Their detailed statistics are shown in Table 2.

When evaluating Algorithm-2 in terms of accuracy, we find out that Algorithm-2 exhibits 100% accuracy. Although, it shows drastic reduction in computational time but its accuracy is evaluated to be same as that of Algorithm-1.

Third experiment as shown in Fig.4 is the pictorial view of time efficiency of two proposed algorithms executed over 5 batches for end to end TF-IDF calculation.

While dividing batches, it is ensured that Algorithm-1 is executed by increment of 20,000 documents in each batch because Algorithm-1 does not store any lookup for each batch. However, execution of Algorithm-2 is carried out by dividing 100,000 documents into 5 batches. Each NewBatch contains 20,000 new documents only and does not contain any document from previous batch.

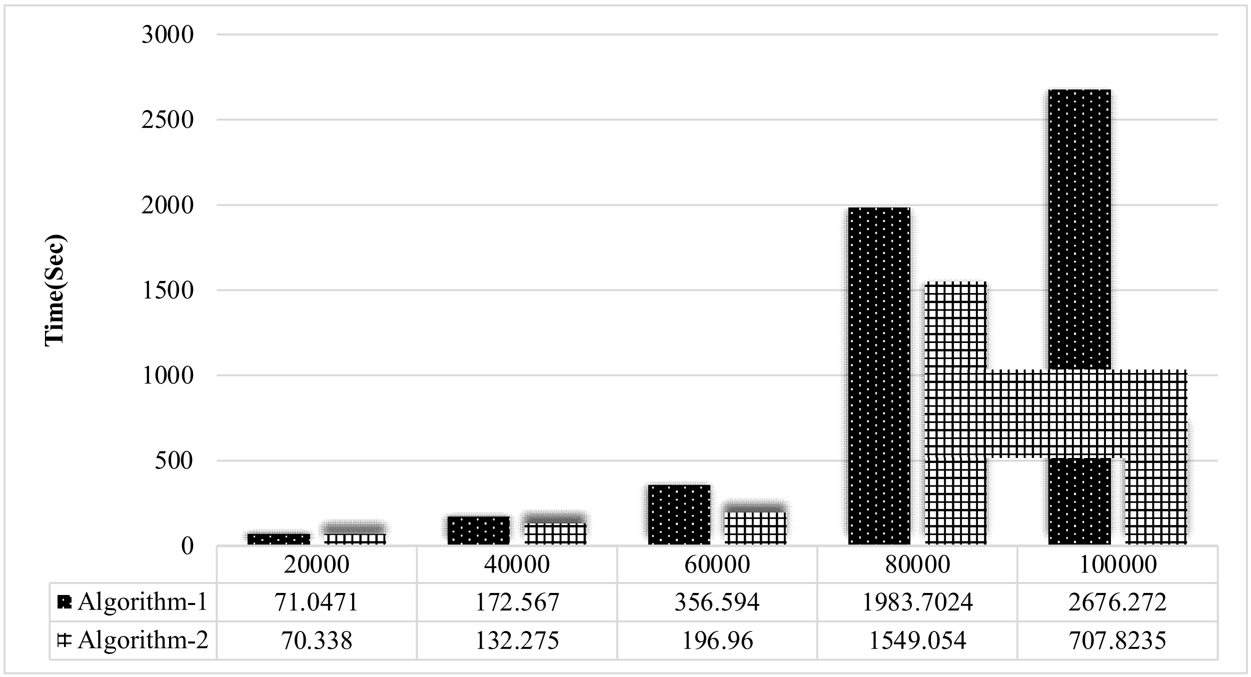
By viewing the bar values for Algorithm-1 in Fig.4, it is evident that the time for TF-IDF calculation increases with the increase in number of documents in each batch. Algorithm-1 involves variation in (5) and the effect of this variation is evident while IDF calculation. Algorithm-2 involves storage of term frequency, document frequency and IDF lookups after execution of each batch. These lookups are then used in NewBatch for IDF calculation, update and TF-IDF calculation. Similarly, we can observe in Fig.4 that Algorithm-1 takes 44 minutes for executing 100,000 documents whereas Algorithm-2 takes 11.7 minutes for execution of 100,000 documents.

Forth experiment involves observing the extent of memory reduction by incorporating sparse representation of TF-IDF vectors created after TF-IDF calculation. As total number of unique trigrams in 0.1 million documents is 11,724,976. So, creating TF-IDF matrix will occupy 1×10^5 rows and about 11.7×10^6columns. This will make the total entries of matrix as

$$\textit{No. of rows} \times \textit{No. of columns} = 11.7 \times 10^{11}\textit{entries}$$

As each TF-IDF entry is stored as a double in memory, so total memory consumed for TF-IDF matrix of 0.1 million documents is given by 8735.787 GB. This is practically almost impossible to store in main memory

By observing the vector of TF-IDF for each document, it was found that they contain a lot of null values and they are redundantly occupying memory. By converting the dense representation to sparse representation for each document. It was concluded that the total number of non-zero terms in TF-IDF vectors of 100,000 documents are 30,841,481. So, for sparse representation of 100,000 documents we need 30,841,481 double entries and same amount of integer entries as shown in Fig. 1. So, total memory occupied by TF-IDF vectors of 0.1 million documents will be 353 MB which is far less than space occupied by dense representation. Thus, we save 8735.558 GB. This is almost 99.996% reduction in memory being used in case of dense representation.



	20000	40000	60000	80000	100000
■ Algorithm-1	71.0471	172.567	356.594	1983.7024	2676.272
⊞ Algorithm-2	70.338	132.275	196.96	1549.054	707.8235

Fig. 4 TF-IDF calculation using Algorithm-1 and Algorithm-2 over 5 batches

6. Conclusions and Future Work

In this research study, the efficiency of TF-IDF algorithm is improved. The existing approaches for efficiency improvements of TF-IDF algorithm for huge amount of data involve hardware level enhancements for parallel computing. Most of the work is based on static data. In this paper, two algorithms are presented. Algorithm-1 is slight modification of efficient implementation of Traditional Algorithm. For 100,000 documents Traditional Algorithm takes 2676520 ms, whereas Algorithm-1 shows an improvement of 248 ms compared to Traditional Algorithm. On the other hand, Algorithm-2 contains stored lookups of term frequency, document frequency and IDF after execution of each batch and these lookups are used for TF-IDF calculation when each NewBatch is uploaded. It performs very well when data for TF-IDF calculation is being updated dynamically. In our experiment, for Algorithm-2, 100,000 documents are processed divided into 5 batches. Each batch exhibits an increment of 20,000 documents. Final batch has 100% accuracy and shows drastic time efficiency compared to Algorithm-1 for processing 100,000 documents. Algorithm-1 takes 2676272 ms whereas algorithm-2 takes 707823 ms for execution of 100,000 documents.

Another major contribution involves employing sparse representation of TF-IDF vectors. It saves a lot of memory and reduces 8,945,445 MB to 353 MB to store TF-IDF of 11,724,976 terms computed from 100,000 Urdu documents.

Future enhancements in this work include modifying TF-IDF algorithm in such a way that we can execute it on a number of machines concurrently and thus it will divides the execution time of TF-IDF calculation equivalent to the number of machines used for this process.

7. References

[1] M.-G. Jang, S. H. Myaeng, and S. Y. Park, "Using mutual information to resolve query translation ambiguities and query term weighting," presented at the Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics, College Park, Maryland, 1999.

[2] S. E. Robertson and S. Walker, "Okapi/Keenbow at TREC-8," in TREC, 1999.

[3] C. Buckley, A. Singhal, and M. Mitra, "New Retrieval Approaches Using SMART: TREC 4," in TREC, 1995.

[4] W. Na, W. Pengyuan, and Z. Baowei, "An improved TF-IDF weights function based on information theory," in 2010 International Conference on Computer and Communication Technologies in Agriculture Engineering, 2010, vol. 3, pp. 439-441.

[5] H. Turtle and W. B. Croft, "Evaluation of an inference network-based retrieval model," ACM Trans. Inf. Syst., vol. 9, no. 3, pp. 187-222, 1991.

[6] K. Sparck Jones, S. Walker, and S. E. Robertson, "A probabilistic model of information retrieval: development and comparative experiments: Part 2," Information Processing & Management, vol. 36, no. 6, pp. 809-840, 2000/11/01/ 2000.

[7] L. Guoping, K. Y. Lee, and H. F. Jordan, "TDM and TWDM de Bruijn networks and ShuffleNets for optical communications," IEEE Transactions on Computers, vol. 46, no. 6, pp. 695-701, 1997.

[8] B. Stein and S. M. zu Eissen, "Near Similarity Search and Plagiarism Analysis," in From Data and Information Analysis to Knowledge Engineering, Berlin, Heidelberg, 2006, pp. 430-437: Springer Berlin Heidelberg.

[9] H. Chim and X. Deng, "Efficient Phrase-Based Document Similarity for Clustering," IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 9, pp. 1217-1229, 2008.

[10] K. Sparck Jones, "A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL," Journal of Documentation, vol. 28, no. 1, pp. 11-21, 1972.

[11] G. Salton and C. T. Yu, "On the construction of effective vocabularies for information retrieval," SIGIR Forum, vol. 9, no. 3, pp. 48-60, 1973.

[12] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," Computer Speech & Language, vol. 13, no. 4, pp. 359-394, 1999/10/01/ 1999.

[13] S. Basnayake, H. Wijekoon, and T. Wijayasiriwardhane, "Plagiarism detection in Sinhala language: A software approach," Gloria Scientiam - Golden Jubilee Commemorative Volume, Faculty of Science, University of Kelaniya (2017), 10/01 2017.

[14] H. Wu and N. Yuan, "An Improved TF-IDF algorithm based on word frequency distribution information and category distribution information," presented at the Proceedings of the 3rd International Conference on Intelligent Information Processing, Guilin, China, 2018.

[15] K. Sugathadasa et al., "Legal Document Retrieval Using Document Vector Embeddings and Deep Learning: Proceedings of the 2018 Computing Conference, Volume 2," 2019, pp. 160-175.

[16] L. Cheng, Y. Yang, K. Zhao, and Z. Gao, "Research and Improvement of TF-IDF Algorithm Based on Information Theory," in The 8th International Conference on Computer Engineering and Networks (CENet2018), Cham, 2020, pp. 608-616: Springer International Publishing.

[17] I. Yahav, O. Shehory, and D. Schwartz, "Comments Mining With TF-IDF: The Inherent Bias and Its Removal," IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 3, pp. 437-450, 2019.

[18] L. Bin and G. Yuan, Improvement of TF-IDF Algorithm Based on Hadoop Framework. 2012.

[19] Y. Gu, Y. Wang, J. Huan, Y. Sun, and W. Jia, An Improved TFIDF Algorithm Based on Dual Parallel Adaptive Computing Model. 2018, pp. 657-663.

[20] AdnanOquaish. (2019, 17- 09- 2019). AdnanOquaish/Cosine-similarity-Tf-Idf-. Available: https://github.com/AdnanOquaish/Cosine-similarity-Tf-Idf-

[21] En.m.wikipedia.org. (2019, 18-09-2019). Computational complexity of mathematical operations. Available: https://en.m.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

[22] E. Series and S. Art. (2019, 28-11-2019). Mathematics Stack Exchange. Available: https://math.stackexchange.com/questions/2033324/express-lnx-with-a-3-as-taylor-series

[23] H. M. Shafiq, B. Tahir, and M. A. Mehmood, "Towards building a Urdu Language Corpus using Common Crawl," presented at the LKE 2019 : 7th International Symposium on Language & Knowledge Engineering, Dublin, Ireland, Oct 29, 2019 - Oct 31, 2019, 2019.