International Journal of Database Theory and Application
Vol.7, No.6 (2014), pp.45-54
http://dx.doi.org/10.14257/ijdta.2014.7.6.05

Compound Method Based on Frequent Terms for Near Duplicate Documents Detection

Gaudence Uwamahoro[1], Zhang Zuping[1]*, Ambele Robert Mtafya[1] and Jun Long[1]

[1]*School of Information Science and Engineering, Central South University Changsha, 410083, China*

gauwa2002@yahoo.fr, zpzhang@csu.edu.cn, kakaambe@gmail.com, jlong@csu.edu.cn*

Abstract

Examining data to find similar data is a major problem in data mining and information retrieval. There are abundant documents that contain information. Most of those documents are duplicates or near duplicates and they increase storage space and cost time for searching for information needed. Reduction of dimensionality and well organization of data are the ways that can be used to solve the problem of efficiency. In this paper we proposed a method based mined frequent terms from each document to reduce the data size and efficient method for clustering documents that have close similarity between them. Using our method only 36.4% of original size has been used. The similarity between documents is based on frequent terms shared. Our method performs well on running time of O(n) whereas the current methods for clustering require O(n^3).

Keywords: *frequent term, Apriori, text clustering, near duplicate document, similarity measures*

1. Introduction

As the Internet technology increases, the data in form of electronic documents increase day by day storage and search for information needed seems to be complicated because of time and memory storage which affect effectiveness and efficiency in representing result to the user. It is hard to organize, analyze and present these documents manually. Research in information retrieval and data mining have opened their mind and oriented their research on haw to provide fast the result wanted by the user. To respond to the user's query, the search system produces a list of documents which are close to the user request as result. Identification of those documents costs time to serve result and that annoys the user for the wasting time in waiting. The main cause of low efficiency and effectiveness in search process is dimensionality and decrease it can allow a great improvement. Documents containing the information some are duplicates of others and near duplicates and there is advantage to identify them. Similar documents can be detected using checksumming technique but detection of near duplicates documents to know how documents are related each other is still an issue. Two documents are duplicates if they have identical document content and they are near duplicates if they have dissimilarities in their content as in [1].

Organize documents in groups also called clustering where documents that share some content in common are in the same group help to improve efficiency during searching for related documents. Some techniques from data mining have been proposed in information retrieval to improve efficiency during retrieving process. One of the techniques is the use frequent itemsets with origin from association rule mining of transaction dataset. A frequent itemsets is a set of frequent items, which appears in transactions more than a given threshold value called minimum support. Recent many studies on frequent itemsets are going on for text clustering and text classification. In this paper we propose a fast method based on terms that are frequents in document to detect near duplicate documents and organize them to facilitate the searching in. Documents which are near similar are grouped together.

2. Related Works

Recently duplicate and near duplicate documents detection is interesting topic in information retrieval. Several approaches to detect near duplicate documents have been proposed like fingerprinting, shingling, checksumming and bag of word, WPBADS algorithm and similarity measures also are used as in [2-4]. Being efficient only for small size document is the major drawback for those approaches. The shingling technique was proposed by Broder et al 1997 and that method has been used in [4, 5]. More shingles share two documents more documents are similar. The problem with shingles is that the size of shingles is greater than the size of document itself. To reduce the size of shingles fingerprinting method has been used in [6] where each shingles is fingerprinted using Rabin algorithm. There are several fingerprinting algorithms and are used in the reduced dimensions method. Another method proposed for duplicate documents detection in [5] is Locality Sensitive Hashing Algorithm (LSH). In that method, hash functions are classified to *k* Triad bands of *k* hash functions and all hash functions applied to the input document and the result is stored in each respective band. The hash function is specified for each band and for each pair of documents which are identical to each other. Xiao, W. and J. in [7] proposed a new filtering technique by exploiting the ordering information in prefix filtering. A method based on conceptual tree has been proposed in [8] where each document is presented as a tree. After fingerprinting technique was proposed as a way of dimensionality reduction, the use of mining frequent termsets from text currently is interesting topic in research community as it reduces drastically the dimensionality. Frequent itemsets have been used by Yanjun and Sajid in [9, 10]. For Sajid M. and others in addition to the mining frequent itemsets, they mined also some ignored itemsets with low support to produce important negative association rules. S.Murali have used no-overlapping partitions of text documents based on frequent termsets and generate clusters within partitions for documents collection as in [11].

3. Proposed Method

The proposed method for near duplicate documents detection is aimed for organizes well documents in groups of all documents that share the same features (frequent termsets selected from each document). That will help to know the location of documents related to each other. In our method we use Apriori algorithm to mine frequent termsets from each document in collection and that method will decrease the dimensionality of documents. Not all frequents termsets are candidate to be selected, our method consider only single frequent terms i.e. frequent 1-terms because we realized that all documents that contain patterns of frequent termset i.e. termset with different length contain also all patterns, example frequent termset "Chinese town movie"; that frequent is seen in documents that have together those three terms where each term is frequent term according to the Apriori algorithm principle. When is considered terms shared by pair of documents, those terms are considered separately. The frequent terms will be organized in different groups based on the portions of features documents have in common. There are 3 main steps in the proposed methods:

(1) Text preprocessing by removing all punctuations, stopwords, stemming using Porter algorithm and lowering all characters.

(2) Frequent termsets identification.

(3) Grouping documents based on features portion shared.

The additional preprocessing is creating chunks also called sentences with equal length to allow mining frequent termsets. With fixed minimum support i.e. number of sentences at least a term must appear in, the term is a frequent term if it fits the minimum support.

3.1. Algorithm Description

Let be D_n be a collection of documents and $D_n = \{d_1, d_2, d_3, \dots, d_n\}$ where n is the number of documents in collection. Each document d_i in D is a set of chunks, therefore $d_i = \{d_{i1}, d_{i2}, d_{i3}, \dots, d_{ik}\}$ where k is the number of sentences in d_i. Identification of frequent termsets from d_i is the following:

Mine-frequents Algorithm

```
Input: Phrases $T$ [ ]: chunks of document, minimum-support
Output: set of frequent termsets $F$
BEGIN
    // calculate terms that come frequently in chunks of each document
    $F_1$ ={1-termsets} in $T$   //where $T$ is the set of chunks generated from document
    For $k$=2, $F_{k-1} \neq \emptyset$; $k$++
        $C_k$ = apriori-compute ($F_{k-1}$); //where $C_k$ is set of frequent termsets candidates generated from $F_{k-1}$
        For all chunks $w$ in $T$
            $C_w$ = subset ($C_k$, $w$) where $C_k$ is a candidate termsets of size $k$
            For all frequent termsets candidates $d \in C_w$
                d.count ++;
            Next
        Next
        $F_k = \{d \in C_k \mid d.count \geq$ mini-support}; // $F_k$ is set of frequents that support minimum support
    Next
    Return = $\cup_k F_k$    //all frequent termsets generated
END
```

Only frequent 1-terms from each document are candidates to represent each document in collection. The similarity between documents is measured based on the number of frequent terms pairs of documents share. The more frequent 1-terms they share the more are similar. As our goal is to know all frequents terms documents share to conclude either are similar or near similar, we need a list of all frequent terms selected for each document in collection and list of documents the frequent termset appears in as in Table 1.

Table 1. Frequent Terms with their Documents

Frequent term	**Documents**
fq_1	d_1 ,d_2, d_7, d_9,
fq_4	d_2, d_5, d_{11}, d_3
fq_2	d_8, d_1, d_{12}, d_{10}
…	…, …, …, …

The proposed method allows minimizing computation time and storage space in memory is to improve efficiency and effectiveness. That is the reason we propose the use of tuples and triples relationships. The first relationship tuples help to know each frequent term in collection the document it belongs to as in Table 2.

Table 2. Frequent-documents Tuples

Frequent term	Document
fq_1	d_1
fq_3	d_5

fq_5	d_1
fq_7	d_{12}

Identification of each frequent term from the collection of all frequents terms from all documents is done by frequent-distribution algorithm which ends up with list of tuples.

Frequent-distribution Algorithm

```
Input: Dictionary of frequent terms and list of documents per each term
Output: A list of tuples (a, b) such that a is the document, b is the
frequent term in a
BEGIN
        ListTuple = {}
        For each D in Dict
                F = D.ListOfTerms()
                For each L in F
                        Tuple = (F.key, L)
                        ListTuple.add(tuple)
                Next
        Next
        Return (ListTuple)
END
```

The comparison of each document with any document in collection, the list of tuples created above is used to know other documents having the same frequent terms as the document to compare with. The expected result is to get a list of *(a*, *x*, *c*) triples where *a*, *c* are documents that share *x* frequent terms. We use Compute-Triples Algorithm (*CTA*) to identify all triples in collection.

Compute-Triples Algorithm(CTA)

```
Input: The list of tuples ListTuple [ ], ListOfDocuments [ ]  //documents in collection
Output: A list of triples // Example (a, b, c) such that a is the document having b as
frequent term and b also is in the document c
BEGIN
        For each D in ListOfDocument
                setOfFrequent = Reader (D.frequentTerm())
                setOfTuples = setOfFrequent ∩ ListTuples
                For each t in setOfTuples
                        triple = (D, t.doc, t.freqTerm)
                        ListTriples.Add(triple)
                Next
        Next
        Return(ListTriples)
END
```

The similarity is measured by number of frequent terms two documents have in common. Now we create a list of pairs of documents and the number of frequent terms they share using Group-Common Algorithm (*GCA*).

Group-Common Algorithm(GCA)

```
Input: ListTriples[ ]:The list of triples
Output: Triples counts :A list of pairs of  documents and the number of frequent terms
they share
```

```
BEGIN
        ListTriples.Sortby(docA, docB)
        For each record rc in ListTriples
                If current record = previous record
                        counter = counter + 1
                Else
                        Add counter to the list
                        Counter = 0
                End If
        Next
END
```

We create groups also called compounds of documents that share the same number of frequent terms.

```
Clustering-Compound Algorithm(CCA)
Input: Listcountuples [ ]:List of tuples and their count , Threshold
Output: Listcluster [ ]:Tuples in the same cluster
BEGIN
        counter = max(NumberTerm  in Listcountuples )
        While counter >threshold
                Tempset = All rows with counter as NumberTerm
                Listcluster.Append(Tempset)
                 counter = counter-1
        Next
        Return (Listcluster)
END
```

The principle of our method is that the document sharing the same number of frequent terms are in the same cluster. We also propose a method for grouping near duplicate documents according to the wanted number of groups.

```
Create_Clusters Algorithm
Input: ListTriples : The list of triples created by Compute-Triples Algorithm
 `      nc: Number of clusters to form
Output: ClusterIndex :Index of clusters, created according to the number of frequent
shared terms
BEGIN
        NbreCluster = 0
        TempCluster ={} // Empty set
        ClusterIndex = Empty Index
        Top_Cluster = Max(ListTriples.freqTerm)
        //The first step is to create all clusters in the list.  In this step, all document
        sharing the number of frequent terms will be in the same cluster
        For each record rc in ListTriples
                ClusterIndex.Add(rc.frequentTerm : rc.doc1, rc.doc2)
                If not exist (rc in ClusterIndex) Then
                        NbreCluster = NbreCluster + 1
                End If
        Next
        //The next step is to adjust the number of clusters according to the number given
```

```
by the user
    While NbreCluster > nc
        Counter = TopCluster – 1
        While not exists (ClusterIndex.counter)
            counter = counter – 1
        Next
        If counter = TopCluster – 1 Then
            For each rc in ListTriples
            ClusterIndex.Add(TopCluster : ClusterIndex.Value = Counter)
                ClusterIndex.Remove (counter)
            Next
        Else
            TopCluster = counter
        End if
    Next
    Return (ClusterIndex)
END
```

3.2. Time Complexity Comparison

Analyzing the time and space complexity is very important to know efficiency of execution of a program. The time complexity is measured by the number of elementary operations curried out during execution of program, whereas the space complexity is computer memory used by algorithm. The good algorithm is the one that minimize space and time. After features extraction from documents the proposed algorithm minimize space and time for increasing efficiency. To get similarity between documents most many algorithms are exponentials where for each feature the combination of all pair documents having in common that feature is made. For *n* features, the number of operations requires is $O(n^2)$ and cost high memory as the number of documents increases. To minimize time and space our proposed method uses small number of operations by using triples relationship and counting, grouping and aggregation operations. Using list of frequents from each document and intersection operation to know all tuples sharing the frequent terms with document in consideration, this operation is $O(n)$ as the algorithm must scan the list of frequent terms for n documents at each iteration.

Another operation required is to get list of all triples i.e. pairs of documents and features shared. This operation is linear as the algorithm scans the list of tuples for each document and count number of features in common and the number of operations depends of number of documents in collection. Sorting algorithm which requires n constant time is needed to put all pair's documents that share common features in the same group. Our algorithm is in the place it doesn't require any extra memory. The proposed algorithm to organize documents in groups follows hierarchical clustering method. As hierarchical method, our algorithm has initials which are number of groups of pair's documents sharing same features (frequent terms). Most of hierarchical clustering algorithms low efficiency caused by computing distance between each pairs of clusters to get the best ones to merge which costs $O(n^2)$ for initial steps. The distance between all pairs of documents and subsequent steps take time proportional to $(n-1)^2$, $(n-2)^2$,…and the sum of squares up to *n* is $O(n^3)$ which is cubic algorithm and it is difficult for this algorithm to run. Our algorithm the initial steps is constant we have already specified number of groups with distances between pairs of documents and it requires only $O(n)$ times for this operation. As the list of triples is sorted based on number of features shared, it allows us to take easily decision on merging two groups. We compare the gap between the number of features in common between pairs of documents in each group. To get number of cluster our algorithm will do two main operations which are grouping together all group according the number of features in common, and merging groups. For the first operation the time required is $O(n)$ where n is maximum frequent terms whereas the second operation needs $O(\log NbreCluster)$. The total time required is $O(n) + O(\log NbreCluster)$ and it is $O(n)$.

4. Experiment Results

The Figure 1 shows 13 groups of documents with their shares grouped by number of features in common.

Document1	Document2	No. of Freq.terms shared
cv383_13116.csv	cv986_13527.csv	79
cv115_25396.csv	cv274_25253.csv	72
cv256_14740.csv	cv908_16009.csv	44
cv597_26360.csv	cv765_19037.csv	43
cv146_18458.csv	cv946_18658.csv	40
cv211_9953.csv	cv345_9954.csv	37
cv328_10373.csv	cv433_10144.csv	36
cv174_9659.csv	cv759_13522.csv	35
cv074_6875.csv	cv612_5461.csv	35
cv309_22571.csv	cv411_15007.csv	33
cv421_9709.csv	cv992_11962.csv	32
cv439_15970.csv	cv644_17154.csv	30
cv560_17175.csv	cv922_10073.csv	30
cv501_11657.csv	cv819_9364.csv	29
cv501_11657.csv	cv506_15956.csv	29
cv327_20292.csv	cv508_16006.csv	28
cv320_9530.csv	cv519_14661.csv	28
cv317_24049.csv	cv552_10016.csv	28
cv512_15965.csv	cv717_15953.csv	28

Figure 1. Pairs Documents Groups

In our experiments we are based on the collection of thousand documents with different sizes. The similarity between pairs of documents is measured according to commons features (frequent terms) between them. Using our method all documents are in 40 named groups of documents with different number of documents in each group. There are at maximum 79 common frequent terms and the minimum is 1-frequent term. We represent those groups into two groups; the first group contains the following groups: 79, 72, 44, 43, 40, 37, 36, 35, 33, 32, and the second group contain groups named from 30 to 1 i.e. group 30, 29, 28, 27,…, 1 More features share more similar documents are. Those groups are: 79, 72, 44, 43, 40, 37, 36, 35, 33, 32, 30, 29 and 28.

The documents in above groups can also be in groups according to the number of groups wanted also called clusters. The running time using our method is shown in Table 3. The abbreviations used in table 3 such as Docs, D.RFqt size, RTFreqt(secs), D.AllTerms (size), R.TD.AllTerms(secs), RT.All Pairs, represent respectively documents(number of documents), size of documents represented by frequent terms, running time in seconds for comparing documents represented by frequent terms based on the number of frequent terms documents have in common and put documents sharing the same number of frequent terms in the same group, size of documents with all their terms, running time in seconds using All-Pairs method

Table 3. Document Size and Running Time

Docs	D.RFqt size	RTFreqt(secs)	D.AllTerms (size)	Reduced size(KB)	RT.All Pairs
150	171KB	7	469 KB	352(64.4%)	24
300	332 KB	22	932 KB	600(64.4%)	48
450	509 KB	33	1360KB	851(62.6%)	73
500	563 KB	38	1510KB	947(62.8%)	81
1000	1160 KB	67	3180 KB	2020(63.6%)	167

With collection of 1000 thousand files we calculate time used for comparison of all documents. More the size of collection increases more the time comparison increase. The Table 3 shows the time for comparing all documents by comparing terms shared and put them in different groups according to the shared terms. The Figure 2 shows time comparison between our proposed method in this paper and All Pairs method.

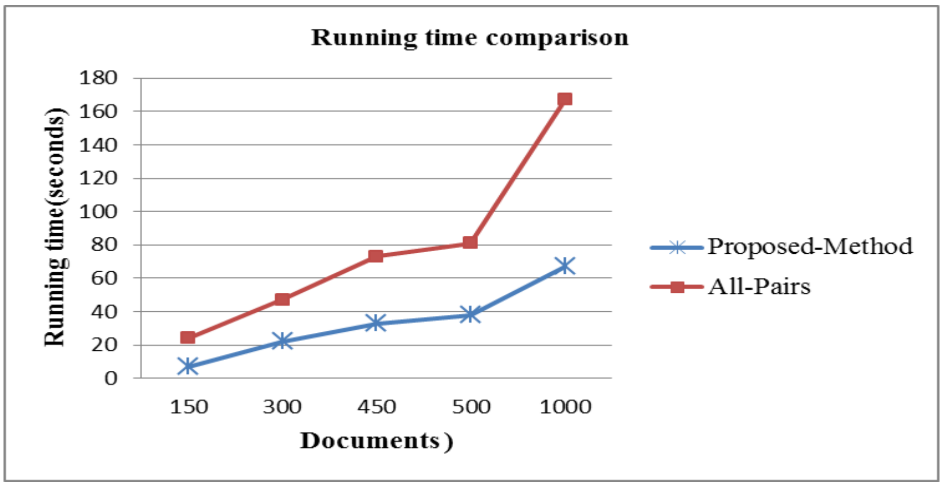

Figure 2. Time Comparison between Using our Proposed Method and All-Pairs Method

Using our method the size of collections of document has reduced at 63.56% and only 36.4% of total size is used and that leads to the increase of efficiency.

5. Conclusion

In this paper we have proposed a method based on frequent terms for near duplicate document detection. Each document is represented by frequent terms generated from that document for document dimensionality reduction which leads to efficiency during documents comparison. The similarity between documents is based on common frequent terms. We proposed a method to facilitate to locate related documents by organizing those documents in different groups where documents sharing same number of frequent terms are in the same group. We realized that hierarchical clustering method proposed is more efficient that the existing hierarchical clustering methods which require $O(n^3)$ running time whereas ours takes only $O(n)$.

Acknowledgements

Project supported the National Natural Science Foundation of China (Grant No. 61379109, M1321007) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20120162110077).We would like to thank the anonymous referees for their helpful comments and suggestions.

References

[1] J. D. SY Mudhasir, S. Sendhilkumar and G. S. Mahalakshmi, International Journal on Internet and Distributed Computing Systems, vol. 1**,** no. 1, **(2011)**.
[2] Y. -S. Lin, T. -Y. Liao and S. -J. Lee, Expert Syst. Appl., vol. 40**,** no. 5, **(2013)**.
[3] G.U.a.Z. Zuping, IJCSI International Journal of Computer Science Issues, vol. 10**,** no. 2, **(2013)**.
[4] P. R. Christopher D. Manning, Hinrich Schutze, Introduction to Information Retrieval, Cambridge University Press, Cambridge, UK **(2008)**.
[5] N.s. H. Naderi, M. N. farokhi, B. H. chegeni, International Journal of Computational Engineering Research, vol. 4**,** no. 3, **(2014)**.
[6] D. I. Ignatov and K. T. Janosi-Rancz, The International Scientific Journal of Sapientia, vol. 1**,** no. 2, **(2009)**.
[7] C. Xiao, W. Wang, X. Lin, J. X. Yu and G. Wang, ACM Trans. Database Syst., vol. 36**,** no. 3, **(2011)**.
[8] P. Lakkaraju, S. Gauch, and M. Speretta, Document Similarity Based on Concept Tree Distance, in Proceedings of the nineteenth ACM conference on Hypertext and hypermedia, **(2008)**, Pittsburgh, PA, USA.
[9] Y. Li, S.M. Chung and J. D. Holt, Data Knowl. Eng., vol. 64**,** no. 1, **(2008)**.
[10] M. S. Sajid Mahmood and A. Guergachi, The Scientific World Journal, **(2014)**.
[11] S. M. Krishna and S. D. Bhavani, European Journal of Scientific Research, vol. 42**,** no. 3, **(2010)**.