J. Basic. Appl. Sci. Res., 3(1s)603-607, 2013

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

A Method for Finding Similar Documents on the Basis of Repetition-Based Filtering

Masumeh Ghasemi Mahsayeh[1] and Ali Mahjur[2]

[1]Department of Computer Engineering, Islamic Azad University of Shabestar, Masumeh
[2] Malek Ashtar University of Technology

Received: June 10 2013
Accepted: July 8 2013

ABSTRACT

There are different discussions underway about data mining in massive datasets and one of them is finding similar items. Different items in this regard can be taken into consideration. One of the relevant items is looking to find similar documents. Finding similar documents can be turned into a set-based question and this can be done within the framework of shingling. Furthermore, using the members of these sets, one can avoid investigating all the pairs of documents to find similar items and simply investigate the documents which are similar to each other with a higher probability. Here we can note the length-based filtering and prefix indexing which operate this way. In these methods, the repetition of the items is not taken into consideration while they can influence the results of the similarity. In the present article, we have provided a method for finding similar documents by taking into consideration the repetition of each member in the document. In fact, the main objective of the present article is to create a solution for reducing the comparison of the pairs of documents for finding the similar documents as much as possible.
KEYWORDS: data mining, shingling, length-based filtering, prefix indexing, repetition-based filtering.

1. INTRODUCTION

During the past two decades, the technical capabilities of the man in producing and collecting data have remarkably increased. Therefore, with the development of databank systems and the given the great volume of data restored in these systems, we need methods by which we can examine and process the data restored in the systems, and then provide the users with the resulting information. These items laid the foundations of the science of data mining. One of the important issues in data mining is the finding of similar items [1]. The objective of this branch of the science of data mining is to create solutions for finding similar items which belong to a massive set of data. For instance, this massive set can be all the pages existing in the web or a massive set of textual documents. In the present article, our focus is on finding similar documents [2]. It should be noted that the aspect of similarity which we are looking for is similarity in the level of words and characters in a text, and we don't look for similarity in the meanings and concepts of the text.

In the present article, we first investigate the issue of similarity and then take note of similarity for the documents. Then the methods of shingling, length-based filtering and prefix indexing have been investigated for finding similar documents. After these items, a method for finding similar items based on repetition-based filter has been presented and the practical results are presented accordingly. Finally, the conclusion and the direction of the future studies have been determined.

2. Similarity

Having in mind the main issue, we concentrate on a certain concept of similarity. This can be a similarity between the sets which is dependent on the value of their intersection [3]. In order to analyze this type of similarity, we propose and review the Jaccard similarity.

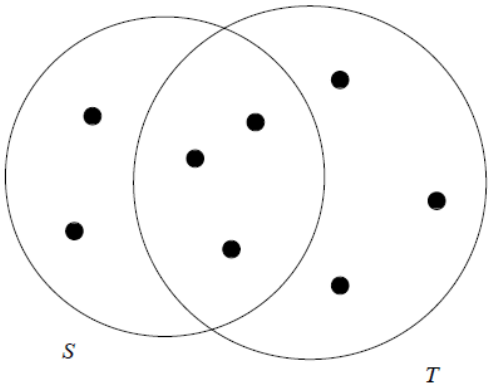
2.1. Jaccard similarity

The Jaccard similarity of two sets is equivalent to the proportion of the sets' intersection to their union. The Jaccard similarity of the two sets T, S is shown as SIM (S, T) and is calculated like the relationship 1 [4]:

$$SIM(S,T) = \frac{S \cap T}{S \cup T} \tag{1}$$

For instance, the Jaccard similarity for the sets S and T is calculated in the Fig. 1:

Corresponding Author: Masumeh Ghasemi mahsayeh, Department of Computer Engineering, Shabestar Branch, Islamic Azad University, Iran. Email address: Masumeh.Ghasemi@gmail.com



$$SIM(S,T) = \frac{S \cap T}{S \cup T} = 3/8$$

Fig. 1. the Jaccard similarity of the sets S and T

One group of issues which the Jaccard similarity covers well is the finding of similar textual documents in a big set [5]. Finding the similarity of the texts has different applications some of which include finding repetitive documents, finding documents close to each other and locating scientific and literary plagiarism.

3. Relevant solutions

3.1. Shingling method

One of the most effective methods for identifying the similar textual documents is shingling. In this method, the documents are displayed in the form of sets. It means that sets are created using the documents. These sets include short strings which are selected from the documents. These short strings are called shingles. It means that shingles are selected of each document, and will be placed in the set related to that document. Using the Jaccard similarity of these sets, the amount of their similarity can be calculated [6].

As we know, a document is a string consisted of characters. A k-shingle for a document equals every sub-string with the length of “k” that has appeared inside that document. In this method, we select a set of k-shingles which are repeated once or more and allocate it to that document [5]. For instance, suppose that the document D contains the string abcdabd and the value of “k” is 2. Then a set of 2-shingles for the document D will be in line with the set {ab, bc, cd, da, bd}. Please note that that sub-set “ab” has appeared inside the document D two times, but is taken into consideration as a shingle only once.

We can select every constant value for “k” that we like. If we take “k” very small, then many of the strings selected with the length of k will exist in the majority of the documents. Then we will have documents which have a set of shingles with high Jaccard similarity while they may have no similar statement or even phrase. The fact that how big we should take “k” depends on the length of the documents and the largeness of the set of characters. At any rate, the important point which we should bear in mind is that “k” should be selected so big that the possibility that each specific shingle appears in each document will be low [7].

One of the issues pertaining to the selection of shingles is the way of selecting the white spaces. By white spaces, we mean such characters as “blank,” “tab,” “new line”, etc. There are several methods of investigation regarding this issue. One is that every trail of one or more characters of white spaces be replaced with a white space or totally removed. As a result, the shingles which are consisted of two or more words can be easily identified [3].

3.2. Filtering method

In the shingling method, in order to find the similar documents, the Jaccard similarity of all the pairs of documents should be calculated and then the similar documents should be obtained. It’s obvious that such a task needs a great amount of time. Even if the number of documents is high, this method is practically impossible. So, we need to avoid investigating all the pairs of documents and review those which are similar with a higher probability. For this purpose, we need to do a pre-processing on the sets resulting from the shingling method so that we find the candidate pair for investigation. It means that we have to filter the documents in a way or another.

To implement filtering, each set can be displayed in the form of a string. To display a set, one can first sort the items of the reference set (the union of all sets) based on a certain order. Then the set can be displayed as a list of items. These items are sorted based on the order of the items of the reference set. The relevant list is a string of characters which are items of the reference set. For instance, assume that there’s a reference set of 26 letters where the order of ordinary alphabet is used. In this case, the {d, a, b} set will be shown as the abd string [5].

3.2.1. Length-based filtering

The easiest way to employ the string showed in the previous section is sorting the strings based on their length. Then, each “s” string will be compared to the “t” strings which appear after “s” in this list and sequence and are not so lengthy. Suppose that the upper endpoint in the Jaccard distance between the two strings equals J. The Jaccard distance of two sets

equals one minus the Jaccard similarity of those sets. The length of each x string will be shown as L_x. Please note that $L_s \leq L_t$. The intersection of the intervals shown by the "s" and "t" strings cannot be more than the number of L_s members, while their union has at least as many members as the number of L_t members. As a result, the Jaccard similarity of the "s" and "t" strings which is shown by SIM (s, t) is at most L_s / L_t. This means that in order to compare "s" and "t", $J \leq L_s / L_t$ or its equivalent $L_t \leq L_s / J$ [6].

Suppose that "s" is a string with the length of 9 and we are looking for strings with a Jaccard similarity of at least 0.9. Then "s" will be compared with the strings which are sorted in a sequence based on length and have a length of at most 9 / 0.9 = 10. This means that "s" will be compared to the strings with a length of 9 which are put in order after "s" and all the strings which have a length of 10. It's not needed that "s" be compared to other strings. Now, suppose that the length of "s" string is 8 instead of 9. Then, it will be compared with the strings which are located after it and have a length of 8/0.9 = 8.89. It denotes that a string with the length of 9 is too lengthy to have a Jaccard similarity of 0.9 with "s". So, "s" will only be compared with the strings with a length of 8 which are sorted in sequence after "s".

3.2.2. Indexing based on prefix

In addition to the length, there are other features for the strings which are used in order to limit the number of comparisons made to identify the pair of similar strings. Here, the simplest choice is that we create one index for each symbol. Please note that we named each of the items of the reference set a symbol for the strings. The symbols are actually the shingles of the reference set. In the prefix indexing method, for each string like "s", one prefix will be selected. This prefix includes "p" first symbols of the "s" string. Then the "s" string will be added to the index for each of the symbols existing in the "p" first symbol [7].

The index for the symbol of one bucket includes the strings which are compared to each other. For every other string like "t", given that SIM (s, t) $\geq$ J, there's at least one symbol in its own prefix that is included in the prefix "s", as well. Suppose that SIM (s, t) $\geq$ J but the string "t" does not have any of the "p" first symbols of the "s" string. Then the Jaccard similarity of "s" and "t" will equal $(L_s - p) / L_s$. To make sure that there's no need for a comparison between "s" and "t", it would be determined that $J > (L_s - p) / L_s$. In this case, the minimum value of p which is the length of prefix can be calculated using the relationship 2 [7]:

$$p = \lfloor (1 - J) L_s \rfloor + 1 \tag{2}$$

Now suppose that J = 0.9. If $L_s = 9$, then p = [0.1 * 9] + 1 = [0.9] +1 = 1. It means that "s" will be indexed only under the first symbol. Each "t" string which does not have the first "s" symbol but is indexed using that symbol has a Jaccard similarity of less than 0.9 with "s". Suppose that the "s" string looks like bcdefghij. Then "s" will be indexed only under the "b" symbol. Assume that "t" is not started with b. Here, we will have two assumptions:

1) If "t" starts with "a", and SIM (s, t) $\geq$ 0.9, then "t" can only have the form of abcdefghij. In this case, "t" will be indexed under both "a" and "b". Since $L_t = 10$, then "t" will be indexed based on the length of its prefix symbols which equals [0.1 * 10] + 1 = 2.
2) If "t" starts with "c" or a symbol after it, then the maximum value of SIM (s, t) will be realized when "t" has the form of cdefghij. In this case, SIM (s, t) = 8/9 < 0.9.

Therefore, with J = 0.9, strings with a length of 9 or smaller than 9 will be indexed by their first symbol, strings with a length between 10 and 19 will be indexed by their first two symbols and strings with a length of 20 and 29 will be indexed by their first two symbols, and so on.

Since we want to solve many-many problems and many-one problems, we can use the indexing plan using two methods. In the many-one problem, the objective is that a document be matched and accorded with all the existing documents and its similar documents be found. However, in the many-many problem, the objective is to investigate all the documents and find the documents which are similar to each other.

With regards to a many-one problem, index will be created for the whole database. However, in order to match the new "S" set with the sets existing in the database, that set will be turned into a "s" string. This string is called probe. Then the length of this prefix which equals $[(1 - J) L_s] + 1$ will be determined. Now, for each of the symbols existing in the prefix "s", the index bucket of that symbol will be referred. Then "s" will be compared with all the strings existing in that bucket [8].

If we want to solve the many-many problems, we start with a database empty from strings and indices. Every "S" set will be considered as a new set for solving the many-many problems. Then "S" will turn into "s" strings. This string will be considered as a probe string in the many-one problem. However, after an index bucket is investigated, the "s" string will be added to that bucket. Therefore, the "s" string will be compared to the next strings which can be accorded to them [8].

4. The proposed method for repetition-based filtering

As it was noted, the shingles which are selected from the documents are placed inside a set. We don't have repetition in the sets, that is, the repetitive shingles will not be placed in the sets. In this case, the repetition of the shingles does not have an impact on the amount of similarity between the documents. For instance, suppose that the value of "k" equals 2. In this case, if the string "ab" is repeated four times in a document, then this shingle will be taken into consideration in the calculation of the amount of repetition only once. In this case, it's possible that a state emerges for the documents in which the similarity calculated has a high degree of error. In order to solve this, we can presume a repetition factor for each shingle and allocate the repetition of the shingle in the document to it. It means that the shingles

resulting from the documents will have fields which determine their repetition. In this case, in order to calculate the similarity between the two documents, we can use the relationship 3:

$$similarity = \frac{the\ sum\ of\ the\ factors\ of\ the\ repeition\ of\ items\ existing\ in\ the\ intersection\ of\ the\ two\ sets}{the\ sum\ of\ the\ factors\ of\ thehhh\ reptition\ of\ all\ items\ existing\ in\ the\ two\ sets} \quad (3)$$

It's clear that this method provides a more precise similarity between the documents, because the repetition includes every shingle. However, in this method, the volume of processing has increased to a small degree.

For instance, suppose that we have the three documents D_1 = abcdhk, D_2 = ababmh and D_3 = ababol. In such a case, the shingles and their repetition factors will look like the Fig 2.

D_1	ab	bc	cd	dh	hk
repetition	1	1	1	1	1

D_2	ab	ba	bm	mn
repetition	2	1	1	1

D_2	ab	ba	bo	ol
repetition	2	1	1	1

Fig 2. Shingles and their repetition factor

The similarity of these documents in three states of the value of real similarity is presented in the Table 1 without considering the repetition and by taking into consideration the repetition:

Table 1. the comparison between the similarities of the documents in three different states

Similarity Document	Real similarity	Similarity without repetition	Similarity with repetition
D_1 & D_2	0.33	0.125	0.3
D_1 & D_3	0.33	0.125	0.3
D_2 & D_3	0.66	0.33	0.6

As it can be seen, similarity with repetition is closer to the documents' real value of similarity.

Considering each shingle or symbol provides a more precise similarity for each pair of the documents; however, to find similar documents, we should review all the pairs of documents with this method and so we should carry out filtering. As it was investigated before, we should index all the documents based on the prefix symbols in order to avoid reviewing all of them. Then we would compare the members of each index to find the documents which are similar to each other. Now, this indexing can be done based on the repetition of each symbol in the document aside from the prefix symbols, in such a way that if a symbol in two documents has repetitive factors with a high difference, then these two documents most probably don't have a high-degree similarity. Therefore, they can be disposed of and their value of similarity may not be calculated.

Here, we assign one index to each symbol and repetition factor. If we create an index for each repetition of the shingle, then the number of indices will be very high and the results would not be appropriate. Therefore, it's better that we take intervals for the repetition factor. For example, we can name the repetition in the range of 1 to 50 as A interval, 51 to 100 as B interval, etc. The selection of these intervals can be based on the value of the repetition factors of the symbols. Now, we will have one bucket for each symbol and repetition interval. So, in this method, for the sake of indexing, one should first calculate the prefix length like before. Then, the document will be sent to the relevant bucket on the basis of each of the prefix symbols and the repetitive interval which belongs to that shingle (given its repetition factor).

For example, suppose that the prefix of the "s" string contains "m" symbols with a repetition factor of 32. In this case, the string "s" will be indexed to the bucket (m, A). Moreover, if the prefix of the "t" string contains the symbol "m" and its repetition factor equals 86, then the "t" string will be indexed to the bucket (m, B). It means that identical buckets will not be sent, while both of them would be sent to the bucket "m" regardless of repetition.

4.1. Practical results

In order to review the methods presented, first the shingling algorithm was implemented irrespective of the repetitions and then by taking into consideration the repetition of the shingles. Then the prefix indexing method and the method presented in the article, that is, repetition-based filtering were implemented. The implementation of the algorithms was carried out using the VB.Net programming language. Then this algorithm was implemented on the NSF Research Awards Abstracts (http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html) dataset. In this collection of data, the abstracts of papers at the NSF institute were collected from 1990 to 2003. We implemented the algorithms on the 2003 documents. The number of the documents is 645. In the shingling algorithm, the parameter k=5 and in the indexing algorithm, J=0.9 have been assumed. The shingles are sorted alphabetically. The repetitive intervals which are assumed for the factor of shingles' repetition are shown in the Table 2.

Table 2. repetitive intervals

Range	Name of repetitive interval
1 – 10	A
11 – 20	B
21 – 30	C
31 - 40	D
41 – 50	E
51 – 60	F
61 – 70	G
71 – 80	H
81 – 90	I
91 – 100	J
other	K

The results of run this algorithms on the dataset are shown in the Table 3.

Table 3. results of the algorithm presented

Method	Time of Shingling algorithm	Time of Indexing algorithm	Total time	Number of created buckets
Without repetition	00 : 18	5 : 38	05 : 56	6810
With repetition (algorithm presented)	01 : 7	10 : 45	11 : 52	6960

As it can be seen, the number of buckets has increased in the method presented earlier on, which means that the documents are segregated in a better way. Therefore, in order to find similar documents, we need fewer comparisons.

5. Conclusion

In the present article, we first reviewed some of the methods of finding the similar documents from a massive set of data. The first method which was investigated was the shingling method and it was shown that how the issue of similarity can be turned into a set-based issue. Then, in order to prevent comparison, all the pairs of documents were reviewed using methods based on filtering. These methods include length-based filtering and prefix indexing. In these methods, only the pairs which are similar to each other with a higher probability will be compared. After reviewing these methods, a method for finding similar documents based on the repetition of each shingle or symbol in the document was presented. In the practical results, it was also shown that the method presented needs fewer comparisons for finding the similar documents than the previous methods.

In addition to the features of shingle in the prefix and their repetitions, there are other features for the strings based on which we can index the documents. Of these features one can mention the length of prefix or the situation of the shingles in the string. Presenting an algorithm for indexing the documents through annexing several other appropriate features can be a suitable research ground in this regard.

REFERENCES

1. Mitra, S., Pal, S. K. and Mitra, P, Data mining in soft computing framework: A survey, IEEE transactions on neural networks, 2002 13(1), 3-14.
2. Manber, U, Finding similar files in a large file system, In Proceedings of the USENIX winter 1994 technical conference (Vol. 1), 1994.
3. Brin, S, Near neighbor search in large metric spaces, Department of Computer Science Stanford University, 1995.
4. Bank, J. and Cole, B, Calculating the jaccard similarity coefficient with map reduce for entity pairs in Wikipedia, Wikipedia Similarity Team, 2008.
5. Rajaraman, A. and Ullman, J. D, Mining of massive datasets, Cambridge University Press, 2011.
6. Chaudhuri, S., Ganti, V. and Kaushik, R, A primitive operator for similarity joins in data cleaning, In Data Engineering, 2006, ICDE'06, Proceedings of the 22nd International Conference, p. 5-5.
7. Xiao, C., Wang, W., Lin, X., Yu, J. X. and Wang, G, Efficient similarity joins for near-duplicate detection, ACM Transactions on Database Systems (TODS), 2011, 36(3), 15.
8. Bayardo, R. J., Ma, Y. and Srikant, R, Scaling up all pairs similarity search, In Proceedings of the 16th international conference on World Wide Web, 2007, p. 131-140.