Collecting Valuable Information from Fast Text Streams

Baoyuan Qi[1,2], Gang Ma[1,2], Zhongzhi Shi[1], and Wei Wang[3]

[1] Key Lab of Intelligent Information Processing, Institute of Computing Technology, CAS, Beijing 100190, China
[2] University of Chinese Academy of Sciences, Beijing 100190, China
[3] Beijing Lexo Technologies Co., Ltd. Beijing 100080, China
{qiby,mag,shizz}@ics.ict.ac.cn, waywang@hotmail.com

Abstract. It has become a challenging work to collect valuable information from fast text streams. In this work, we propose a method which gains useful information effectively and efficiently. Firstly, we maintain an analyzer based on the Trie structure and the dynamic N-Gram tokenizer; secondly, unlike the traditional search engine principle, we consider the documents as a query by building the indexes for the whole query base. The experimental results show that it has the strong adaption ability, low latency and high quality support for the complex query combination compared with the conventional methods.

Keywords: Fast Text Stream, Information Collection, Trie, N-Gram.

1 Introduction

Text streams is a kind of data stream, which is composed of texts, such as news feed, blog, weibo, etc.. Knowledge discovery from data stream has attracted more research recent years [1-4]. Data stream mining in the time and space constraints challenges traditional methods and algorithms[5, 6]. There is always concept drift in the data streams, which make the prediction accuracy decrease along with the time [7, 8].

It is always easy for a user to retrieve relative records from a static document set. The inverted document index methods are always fast for querying. However, under the streaming, many problems always arise in the evolving environments. The first one comes from the analyzer that has to be used during indexing and search process. There are always new words emerging and combining in the coming documents and querying, the system has to reindex all the documents periodically to help improve search accuracy. Another problem among the conventional practice is the response time for the query with the increasing size of documents. The time for a query will expand excessively when the index base receives endless incoming documents.

User may change his query at any time and he only concerns the results since the newly query is settled. The case always holds in the monitor systems like network traffic and public safety. Under these systems, the speed of incoming data is very fast and the queries waiting for processed are plenty. To tackle this problem, we propose a

Z. Shi et al. (Eds.): IIP 2014, IFIP AICT 432, pp. 96–105, 2014.

method to effectively collect valuable information from text streams which cost less time and space and dynamically update the query base in time. In this paper we make contributions on various aspects:

- We design a new flexible analyzer to automatically expand and quickly adapted to user defined dictionary. We use Trie tree data structure to insert new items and at the same time the max length are recorded for further N-Gram tokenizer. The dynamically maintained length is used to implement a maximum segmentation analyzer and the tree is used as a fast dictionary look-up table. The advantage is new entry can be handled as soon as new data arrives, so no data will be lost. Still this can reduce the index size as we only use the user defined term to index the incoming documents.
- A new angle to collect valuable information from the endless text streams. Since there is always the time consuming and space demanding to index all the documents when user indexes the query base set. While constructing the query index, the analyzer changes its parameters accordingly. Newly coming document are tokenized by the analyzer and then searched in the index repository. The advantage is that each query is often short and simple; it is often easy to rebuild query index.

2 Related Work

Information retrieval (IR) obtains information relevant to a need from many documents. Many models have been proposed like Vector Space Model(VSM)[9], Okapi BM2.5, etc.. The web search engines like Google, Baidu are the most significant applications. The indexed documents are always static and need to be reindexed once new terms are used for indexing. To return the result in time, there is always query length limit in the input box as long as the operation between the query terms, like OR, AND, etc.. Batch-incremental index mechanism like Zoie[1] is always used in dynamic environment. The newly coming documents come first into the small index, it will merge when the small index becomes big enough. This strategy is helpful when the process time of the merge operation doesn't take too much time and it will hang the system when the index size goes larger.

The terminology prospective search[10] is widely used in text stream. There are a lot of applications based on this method, like Google News Alerts, Google Search. Rather than targeting full text search of infrequent queries over huge persistent data archives (historic search), this class targets full text search of huge numbers of querying over comparatively small transient real-time data. Our method extends this idea and we propose dynamical N-Gram tokenizer to reduce the index size and reduce the response time.

Trie tree, as a data structure makes its good performance in unknown words and variant length conditions[11, 12]. In this paper we use Trie tree to maintain the added

[1] http://javasoze.github.io/zoie/

terms extracted from the query phrases and make a fast lookup for prefix check. N-Gram, as another useful technology, is always used in computational language and translation[13, 14]. We integrate these two technologies to construct a dynamic analyzer that can be further used for indexing and searching.

3 The Proposed Method

In this paper we propose a method to effectively collect valuable information from text streams and it is composed of two key elements supporting the idea: the first one is maintaining an analyzer that adjust dynamically to the query and the maximum length used for N-Gram tokenizer; secondly, based on the tailored analyzer, the method collecting valuable data proposed by us is different from the conventional ones.

3.1 Dynamic Analyzer

The analyzer is essential for tokenizing the document in inverted document search system. The conventional analyzer is always static and doesn't keep pace with the input documents. Under the text stream environment, the data flow is always endless and the processing server should never stop. This causes serious problems including new words that will hurt the performance of the system. The common solution is to periodically rebuild all the documents or to use batch-incremental indexing method. These solutions will incur the high latency of the accurate results, the memory and space cost may be heavily grown along with the time goes by. The size of the data index is always expanding excessively and the valuable data always hides from normal search strategy.

To cope with the problems, we build and maintain a dynamic analyzer. We integrate the commonly used Trie data structure and N-Gram terminology to gain an effective and efficient analyzer.

Trie data structure coms from the word 'Re**trie**val' and it is a string tree. Its philosophy is to reduce time complexity by improving space complexity. The operations like INSERT and QUERY can be processed instantly and so it is commonly used as a lookup tables especially in PREFIX/SUFFIX computation settings. As an example to build a Trie, suppose the input terms are *T=[b, abc, abd, bcd, abcd, efg, hii]*, the constructed Trie is illustrated in Figure 1. The red circle indicates a complete term through the root, the white circle indicates an intermediate node.

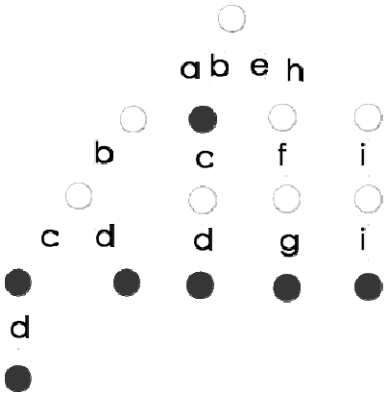


Fig. 1. An Illustrative Trie Example

From Figure 1 we can see it is easy to retrieve all term that begin with a specified prefix, for example *b* and *bcd* are all begin with letter *b.* Trie can be used in suggestion complete to improve the search experience. In this paper we use them as a lookup table to store and search the query base.

N-Gram partitions the input into N continuous segments used for predicting in the form of (N-1)-Markov. N-Gram is very simple to understand and easy to scale up. We can simply increase the parameter *N*, then a model can be used to store more context with a space–time tradeoff, enabling experiments to scale up very efficiently. As an example to generate an N-Gram sequence, suppose the sample *S=ABCA*, then the 1-gram sequences (also known as *unigram*) are [*A, B, C, A*], 2-gram sequences (also known as *bigram*) are [*AB, BC, CA*]. In this paper the base parameter *N* is set during the Trie construction process, so, it does not need predefined by an expert.

We implement the dynamic trie update algorithm in details in Algorithm 1. We sort the query to the leaf of the trie and update the maximum length of the query along with the construction process.

Algorithm 1. Dynamically update the Trie

Input: Given an empty or already existed Trie *trie* and query set *qset*.
Output: An updated *trie* and maximum length *N* in the query base.

1. Extract all meaning and single words from the *qset*. We do this because there may be operators and quoted keywords. For simplicity, all the quoted letters are extracted to the set *trimedQSet*.
2. **if** *trie* is empty **then**
3. Initialize *trie*;
4. *N*=0;
5. **for** each *trimedQ* in *trimedQSet* **do**
6. Use *trie* to *trimedQ* to the *trie* leaf;
7. **if** the length of *trimedQ* is larger than *N* **then**
8. *N:=length(trimedQ)*;
9. Output *trie* and *N*.

We segment the upcoming document according to the trie obtained in Algorithm 1, then extract the hit term and transfer them to the indexing or searching process. Algorithm 2 details how to segment the document accordingly. We segment the document content and return all the tokens existing in the trie.

Algorithm 2. Segment a document using analyzer based on N-Gram

Input: Given a document *doc*. *trie*, lookup tables for token search. *N*, the current maximum length among all the queries.
Output: The tokens existing in the trie.

```
1. Extract the content content from doc. For indexing there may be other meta
   information, like timestamp, author etc.. In this step we just use content part
   like title and text.
2. tokens=[];
3. for i:=0 to length(content) do
4.       for j:=i+1 to i+N do
5.            token:=content.substring(i,j);
6.            if trie.contains(token) then
7.                 tokens.add(token);
8. Output tokens.
```

In this part, we give two algorithms to show how to update trie and segment the document using N-Gram. The dynamic analyzer can insert the query fast and segment the input document, which makes the index size smaller than conventional method and collects valuable information according to the condition user specified.

3.2 Information Collecting Method

In the dawn of big data, the speed of the content generated grows fast. In some specific domains, people always want to focus the valuable information and filter out the information not relating to them. The traditional IR base methods require operating on the full data don't adapt to the data flow format. Another problem is the complex query conditions that will cost too much time to compute and load all the relative record from the index.

We propose an information-collecting method from another angle based on the previous dynamic analyzer. In the old-fashioned roadmap, documents are collected and indexed by the analyzer, then, the query are sent to the index repository, the possible BOOLEAN operation goes and the different RANK algorithms are used to sort the relative documents. Our work, however, index all the user query beforehand, we consider the incoming document as a query, if the score is larger than zero, we will collect it and store it, otherwise drop it. The score is a similarity between the query and the document, we could use VSM model to help calculating it easily. We illustrate the logic in Algorithm 3.

Algorithm 3. The method to collect valuable information from text streams

Input: Given endless text stream *ts*. *analyzers*, generated by each query.
Output: Add the valuable document to its own list.

```
1. while true do
2.     Pick up a document document from the text stream;
3.     for each query in the querySet do
4.         Segment document using the corresponding analyzer, we get SEG;
5.         Similarity_Score:=sim(query,SEG);
6.         if Similarity_Score > 0 then
7.             Add document to the document list of this query.
```

We assume there are plenty of query sets and they are complex, which is very common in the platform of monitoring system, like traffic control of network, sensitive information detecting for a corporation. We obtain the valuable information through checking the each document to the query base from which the score is calculated as a metric for similarity distance.

4 Experimental Results

We conduct experiments on real data and make comparative results according to various settings.

4.1 Data Sets

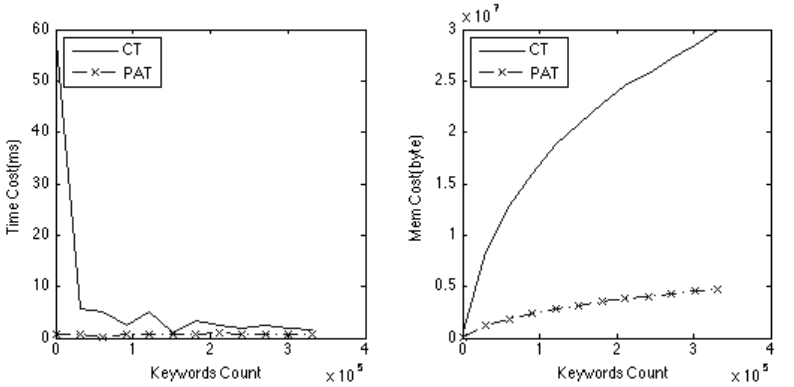
We use Chinese patent data to evaluate the method proposed in this paper. The date range of the data is between 1985 and 2011 years. We extract all the keywords from the raw data and remove duplicated ones, and we get 350,000 distinct keywords. To simulate the text stream, we sort the document by the date and then iterate them in order.

4.2 Measurements of Trie Implementations

Since Trie is essential for our method, we evaluate the state-of-the-art implementations. The patricia-trie(for short PAT)[2] and concurrent-trees (for short CT)[3] are used, and we compare the time cost and memory cost in each insertion with the number of keywords.

[2] https://code.google.com/p/patricia-trie/
[3] https://code.google.com/p/concurrent-trees/

All the keywords are inserted under the same condition; the results are illustrated in Figure 2. The time and memory cost change with the growth of the dictionary number. It is clear from the figure that PAT is always better than CT on both time and memory factors.



(a)Time Cost with the increase of the keywords count

(b)Memory Cost with the increase of the keywords count

Fig. 2. Comparison of Time Cost and Memory Cost between CT and PAT with the Number of Keywords

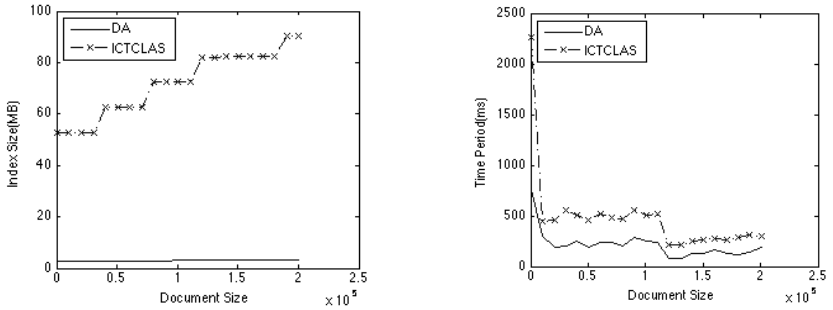
4.3 Performances of Analyzers

We evaluate our proposed dynamic analyzer(for short DA) with ICTCLAS[15] analyzer. ICTCLAS is based on HHMM and is considered as a baseline for many analyzers. In this paper we focus on not its performance but the index growth ratio and time cost about indexing speed with documents increase.

We use Lucene[4] as an index engine. The document set is constrained to the subset of the total data. We used all the CLAIM and KEYWORD field in the patent during 2012s. We got totally 219,144 patents and 105744 keywords for PAT and the final maximum length is 7. Additionally, the maximum number can be adjusted according to the current dictionary and is able to cover the entire possible window for indexing.

Figure 3 illustrates the results, from the figures we can see our proposed analyzer DA could reduce the index size drastically and still it could reduce the index latency in each update. We should notice that the keyword is added dynamically when the patent document arrives; it proves the applicability of the dynamic mechanism proposed in this paper.

[4] `http://lucene.apache.org`



(a) Index Size changes with the increase of document size

(b) Time Period changes with the increase of document size in each update

Fig. 3. Comparison of Index Size and Time Period between DA and ICTCLAS when Index Changes with the Document Size

4.4 Performances of Collecting Methods

We compare our proposed method (for short *Stream Search*) and the traditional batch-incremental method (for short *BI Search*). We test totally 40,000 patent documents. For text stream, the documents are sorted chronologically and are sent in order. For batch-incremental method, the size of documents is 1000 for each batches and the small index is merged to the big index. The Figure 4 gives an illustration of the two methods in time cost, what we can see is our method reduces the time cost to half of the batch-incremental method.

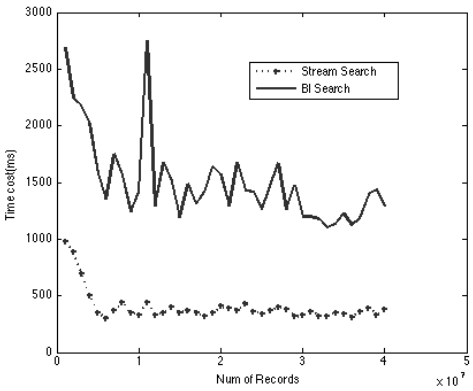
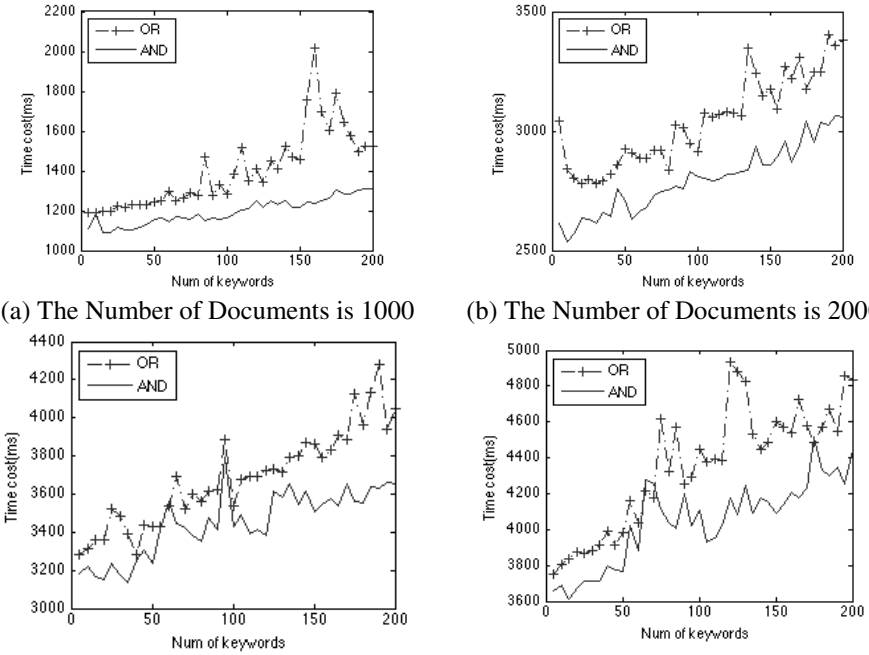


Fig. 4. Time cost between Stream Search and BI Search with the document Size

To evaluate the soundness when the query is very complex and the combination of different joint operations, we increase the number of keywords and use OR and AND operation to evaluate the time cost for the batches with different document size, the

results are listed in Figure 5. From the figures, we can see that the time increases linearly with the number of keywords, obviously, this phenomenon is reasonable in a practical production environment. Additionally, the AND operator is always cost much time than the OR operator.



(a) The Number of Documents is 1000 (b) The Number of Documents is 2000

(c) The Number of Documents is 3000 (d) The Number of Documents is 4000

Fig. 5. Comparison of Time Cost for OR and AND Operators with New Documents Added to the Text Stream

5 Conclusions

In this paper, we propose and evaluate a new information-collecting method from fast text streams. Our experimental results indicate that dynamic analyzer based on N-Gram performs well in the speed for tokenizing, smaller index size and its automatic adaption character. The method considering the query set as index base and the incoming document as query can significantly reduce the time cost in fast text stream environment.

Acknowledgments. The work was financed by the National Program on Key Basic Research Project (973) (No. 2013CB329502), National Natural Science Foundation of China (No. 61035003, 61202212, 61072085), National High-tech R&D Program of China (863 Program) (No.2012AA011003), National Science and Technology Support Program (2012BA107B02).

References

1. Gama, J., et al.: Knowledge discovery from data streams, Citeseer (2010)
2. Aggarwal, C.C.: Data streams: models and algorithms. Springer (2006)
3. Graham, C., et al.: Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. Found. Trends Databases 4(1-3), 1–294 (2012)
4. Muthukrishnan, S.: Data streams: Algorithms and applications. Now Publishers Inc. (2005)
5. Li, M., et al.: Time and space efficient spectral clustering via column sampling. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2011)
6. Zhang, Y., et al.: Space-efficient relative error order sketch over data streams. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006. IEEE (2006)
7. Xioufis, E.S., et al.: Dealing with concept drift and class imbalance in multi-label stream classification. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, vol. 2. AAAI Press (2011)
8. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with Drift Detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004)
9. Salton, G., Wong, A., Yang, C.-S.: A vector space model for automatic indexing. Communications of the ACM 18(11), 613–620 (1975)
10. Irmak, U., et al.: Efficient query subscription processing for prospective search engines. In: Proceedings of the 15th International Conference on World Wide Web. ACM (2006)
11. Kanlayanawat, W., Prasitjutrakul, S.: Automatic indexing for Thai text with unknown words using trie structure. In: Proceedings of the Natural Language Processing Pacific Rim Symposium (NLPRS 1997) (1997)
12. Kijkanjanarat, T., Chao, H.: Fast IP lookups using a two-trie data structure. In: Global Telecommunications Conference, GLOBECOM 1999. IEEE (1999)
13. Doddington, G.: Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In: Proceedings of the Second International Conference on Human Language Technology Research. Morgan Kaufmann Publishers Inc. (2002)
14. Brown, P.F., et al.: Class-based n-gram models of natural language. Computational Linguistics 18(4), 467–479 (1992)
15. Zhang, H.-P., et al.: HHMM-based Chinese lexical analyzer ICTCLAS. In: Proceedings of the Second SIGHAN Workshop on Chinese Language Processing, vol. 17, p. 2003. Association for Computational Linguistics (2003)