A Linguistic Knowledge Discovery Tool: Very Large Ngram Database Search with Arbitrary Wildcards

Satoshi Sekine

New York University

715 Broadway, 7th floor, New York, NY 10003

sekine@cs.nyu.edu

Abstract

In this paper, we will describe a search tool for a huge set of ngrams. The tool supports queries with an arbitrary number of wildcards. It takes a fraction of a second for a search, and can provide the fillers of the wildcards. The system runs on a single Linux PC with reasonable size memory (less than 4GB) and disk space (less than 400GB). This system can be a very useful tool for linguistic knowledge discovery and other NLP tasks.

1 Introduction

Currently, NLP research is shifting towards semantic analysis. In order to understand what a sentence means, we require substantial background knowledge which must be gathered in advance. Building such knowledge is not an easy task. This is the so-called "knowledge bottleneck" problem, which was one of the major reasons for the failure of much AI research in the 1980's. However, now, the circumstances have quite changed. We have an almost unlimited amount of text and machine power has drastically improved. Using these fortunate assets, research on knowledge discovery in NLP is booming. The work by (Hearst 92) (Collins and Singer 99) (Brin 99) (Hasegawa et al. 04) are only a few examples of this research direction. Notice that most of these methods use local context. For example, a lexico-syntactic pattern, like "NP such as NP" can extract hyponym relationships (Hearst 92), and contexts between two named entities can indicate a relationship between those names (Hasegawa et al. 04).

It is quite natural to believe that the larger the corpus, the larger and the more reliable the discovered knowledge can be. However, it leads to problems in terms of speed and machine power. In order to solve these problems, some people use commercial search engines (Chklovski and Pantel 04). However, using such search engines has serious problems: 1) Only a limited number of results available, 2) Only a limited number of accesses permitted, 3) Mysterious ranking function, 4) Unstable results over a long time period, 5) Slow, as it is over the internet. Another idea to overcome this difficulty is to create one's own (maybe smaller) search engine (Cafarella and Etzioni 05) (Shinzato et al. 08). Although creating one's own search engine has advantages (one of which is the freedom to design the form of the query; such as POS and dependency), it is a huge, expensive task; not everybody can afford to make a search engine. More seriously, not everybody can use it as he/she may want.

In this paper, we will propose an alternative solution which should enable researchers with modest resources to conduct research using huge corpora for knowledge discovery. It is an ngram search tool with the following requirements.

Requirements

1. It searches for ngrams which include an arbitrary number of wildcards, such as "* such as * and", "Mr. * said", "from * to * by *" or "* attack by * * on *".
2. It returns the fillers of the wildcards as well as ngram frequencies
3. It produces the results in a fraction of a second (for most reasonable queries)
4. It runs on a single PC
5. It needs only a reasonable amount of memory (4GB) for processing
6. It needs only a reasonable amount of disk space (400GB) for indexing 10^8-10^9 ngrams

2 Algorithm Overview

There are two reasonable choices for the search algorithm. One is "inverted indexing" (used by *"lucene"* and others) and the other is "trie". We used trie. Using inverted indexing, it is easy to create an index at the cost of runtime speed. We prefer an algorithm which requires more complicated indexing in order to achieve the speed in searching. Trie is an indexing tree structure in which each node represents a symbol (in our case, words) and each link represents the sequence of the symbols (in our case n-gram). Searching can be done by traversing the tree, which is usually done in time constant in the size of the corpus. However, it is important to mention that the trie structure is order sensitive. For example, if the query includes wildcards, such as "*Mr. * * said yesterday*", searching the trie is not an optimal solution.

One naive solution is to create a search system (or a trie) for each possible combination of wildcards. For example, for the query pattern *"Mr. * * said yesterday"*, we should prepare a search system for modified ngrams which have the first, fourth and fifth words of the original ngrams as the first three words. For ngrams of length N, the number of possible combinations of literals and wildcards is 2^N. In theory, if we make that many search systems, we can solve the problem. However, the number of search systems is too large considering the number of ngrams we aim to handle (Table 1). Although we applied two implementation techniques to reduce the size of index (which will be described in the next section), it is still likely that we could not satisfy requirement #6.

We solved the problem by using a single search system for different kinds of search patterns, reducing the number of needed search systems significantly. It can be observed that a pattern with wildcards at suffix positions can be searched using the same trie used for patterns without those wildcards. Also, we don't always need to start the trie by indexing the first word. If we build an alternative trie which starts by indexing the second word, we can cover more patterns with fewer tries. For example, using the trie constructed to search for 5-grams "DEABC" (We will call this a "trie pattern": each letter represents a literal, with A representing the first token in the original ngram), four "search patterns" (i.e. ngram pattern used in the queries), "AB*DE", "A**DE", "***DE" and "***D*", can be searched efficiently.

We found that the minimum number of trie patterns needed to cover all possible search patterns of length N is $_{N/2}C_{N.}$ We have constructed minimal sets of patterns for all N up to 9. Once the system receives a query with one or more wildcards, it finds the trie pattern which covers the search pattern of the query.

3 Implementation

We also implemented two ideas to reduce the size of the index. One is related to a common technique to reduce the size of trie nodes by deleting the index of unique suffixes. In addition, we don't store the remaining data within the trie. We just store the ngram ID at the node in order to further reduce the size of the index. Because there are many search systems, storing the ngram data in a single master database saves a lot of space. When the user wants to see the words of an ngram which was identified by the search, the system retrieves the ngram from the master database using the ngram ID. The benefit of this technique is quite large (more than 50% reduction of index size in 9gram), as many ngrams have long unique suffixes.

The other idea is based on the fact that once the ngram data is provided, no update will be requested (i.e. insertion or replacement procedure in the trie is not necessary). We can eliminate the pointers to the parents and the siblings for each node, which contributes to about 30% additional reduction in the index size.

Because the tries are very large, we divide them into segments (128 segments for 9-grams and 118 for 5-grams), so that an individual trie index segment is small enough to fit in a memory of modest size (requirement #5). Each segment contains a lexicographically contiguous sequence of ngrams. Furthermore, we use *"mmap"* to get the index into memory from the disk, so that only the portions of the trie segment that are actually used will be loaded.

System Flow

We will briefly describe the system flow

1) First, all the words in the input query are looked up in the dictionary. If there are out of vocabulary words, then there is no ngram which matches the query. If all words are known, find the appropriate trie pattern for the input query based on the locations of wildcards.

2) Then the appropriate segment(s) of the trie data is found for the search query. It was done

by searching the table of segment index sorted in lexicographic order.
3) Now, the search in the trie is performed. Note that there are 16,128 tries for 9 -gram (128 segments for 126 trie patterns). If the search ends with an internal node, there is more than one matched ngram. If the search ends with a terminal node, just one ngram matches. If the trie ends before the end of query, you have to retrieve the ngram from the master database and match the retrieved ngram against the query.
4) Once one or several ngram IDs are identified as matched ngram(s), the system will output the information. There are three output modes. a) Only instance and type frequency are printed. b) Only ngram IDs are printed. These two types can be achieved quickly without consulting the ngram master database. c) Ngram instances are printed using the ngram master database.

4 Experiments and Evaluation

4.1 Ngram data

We implemented this using two ngram data sets.
Google 1T Web 5gram
This is a part of the ngram data provided by Google through LDC (Web 1T). The data was generated from approximately 1 trillion word tokens of publicly accessible Web pages.

9-grams from 82 years of newspaper
For knowledge discovery purposes, 5-grams are generally unsatisfactory. A 5-gram can only cover 2 words of context on each side of a single word term. So we extracted 9-grams from a number of newspaper corpora available to us. Including NANTC: LATWP(94-97), NYT, REUFF, REUTE, WSJ (94-96), BBN GigaWord corpus (news archive only): BBC(99-06), People Daily, Taipei Times, The Hindu (00-06), Arab News, Gulf News, India Times (01-06), AQUAINT corpus: APW, NYT (98-00), Xinhua (96-00), CSR corpus: WSJ (87-94).

These corpora are cleaned up by several methods, because some of them have article duplications/minor variants and some of them contain many non-sentences. We use a simple method to reduce such noise, which is to "uniq" all the sentences for each year of each newspaper and count each distinct sentence only once. This is not a perfect solution, but it reduces a lot of noise to an amount almost unnoticeable for the ngram search result. The statistics of the data are shown in Table 1.

Corpus	Google 1T	82 yrs. News
Original Text	1 T words	1.8 G words
Ngram	5-gram	9-gram
Threshold	40	2
#of ngrams	1,176,470,663	119,456,373
# of patterns	10	126
# of nodes	1.4G x 10	160M x 126
Index size	277GB	322GB

Table 1 Statistics of the data

4.2 Example

Obviously, the tool is useful for knowledge discovery tasks for hyponym relations, binary relations, name extraction, relations between names and so on. It is also useful to extract more specific relations, such as "what kind of attack occurred by whom on what/when" by searching for "* *attack on* * * *by* * **". Figure 1 shows a snapshot of the tool's output on this query.

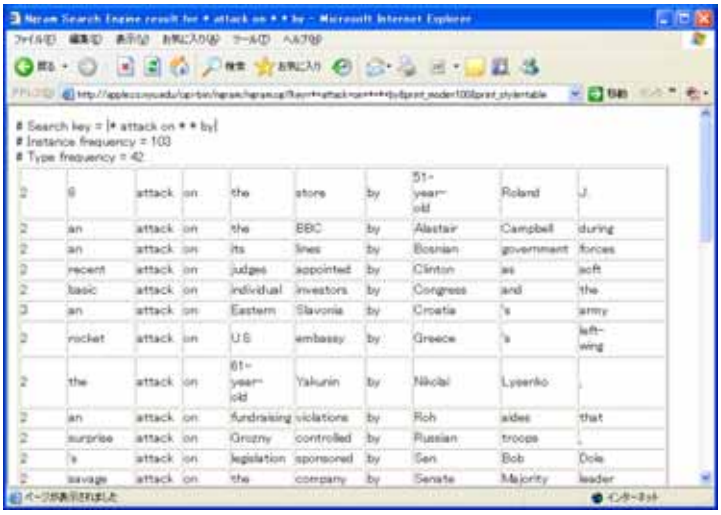

Figure 1 Snapshot of the output for "* *attack on* * * *by* * * *"

Importantly, because of the speed, we can modify the query in an interactive manner. The speed is helpful for batch processing, too, when we need to search millions of patterns.

4.3 Evaluation

The speed evaluation was conducted using 9grams. We randomly selected 1017 9grams from the original data to form a test set, and also created another test set in which 1 to 3 words are randomly replaced by wildcards in those 9grams. We measure the average time to find the ngrams. In the case of a query with wildcards, we checked if the original ngram is found among all returned ngrams to verify the accuracy of the tool. The system runs in three output modes (we use a file output rather than *stdout*). Producing Ngram output takes a long time if the number of ngrams is very large, so we limited the number of ngrams to be printed to 100. In the experiment, the number of returned ngrams ranged from 1 to 100 with an average of 1.26. The result is shown in Table 2 and the times are given in milliseconds.

Print mode	Freq.	IDs	Ngram
Original	19	19	20
With wildcards	19	19	29

Table 2 Speed of search (Newspaper 9gram)

5 Related Work and Discussion

One of the most related works is the CMU-Cambridge Statistical LM toolkit (CMU-Cambridge). It's a tool to search ngrams in Speech fields. However, it does not handle wildcards, which is important for knowledge discovery purposes. There are several ngram tools in the field of genome sequence analysis (MUMmer) (REPuter). However, the size of the alphabets are quite different (handful of genome bases vs. about 1.08 million words), and their main purpose is to discover similarity or repetition. These systems are not directly useful for linguistic knowledge discovery purposes. As a document retrieval tool, there is a public domain search engine, such as "*lucene*" (Lucene). However, its primary purpose is document retrieval and the inverted index algorithm can't handle well very frequent terms. Searching is relatively slow.

It should be noted that creating the index is a huge task. It took about 2 months using five 4GB-memory machines. However, it took only about one week with 64GB-memory machine.

As for the future direction, we are improving the tool to show the original sentences from which the ngram was extracted. We have some evidence that a longer context is needed for the knowledge discovery purpose.

This tool is merely a tool, but we believe it's a very powerful tool for linguistic knowledge discovery among other related objectives. The next step we would like to take is to apply this tool to find linguistic knowledge for NLP applications.

Acknowledgements

This research was supported in part by the National Science Foundation under Grant IIS-00325657. This paper does not necessarily reflect the position of the U.S. Government. We would like to thank our colleagues at New York University, who provided useful suggestions and discussions, including Prof. Grishman.

References

CMU-Cambridge Statistical LM toolkit homepage: http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html

Lucene homepage: http://lucene.apache.org/

MUMmer hompage: http://mummer.sourceforge.net/

REPuter homepage: http://bibiserv.techfak.uni-bielefeld.de/reputer/

Web 1T 5-gram. by Google. LDC Catalog No.: LDC2006T13

M. J. Cafarella and O. Etzioni. "A Search Engine for Natural Language Applications". 2005. In Proc. of World Wide Web Conference.

S. Brin. "Extracting Patterns and Relations from the World Wide Web". 1998. In Proc. of Workshop on Web and DataBase-98.

T. Chklovski and P. Pantel. "VerbOcean: Mining the Web for Fine-Grained Semantic Verb Relations". 2004. In Proc. of EMNLP-04. pp. 33-40.

M. Collins and Y. Singer. "Unsupervised Models for Named Entity Classification". 1998. In Proc. of EMNLP-99.

M. Hearst. "Automatic Acquisition of Hyponyms from Large Text Corpora". 1992. In Proc. of COLING-92.

T. Hasegawa, S. Sekine and R. Grishman "Discovering Relations among Named Entities from Large Corpora". 2004. In Proc. of ACL-04.

K. Shinzato, T. Shibata, D. Kawahara, C. Hashimoto and S. Kurohashi. "TSUBAKI: An Open Search Engine Infrastructure for Developing New Information Access Methodology", 2008. In Proc. of the 3rd IJCNLP-08.