Ngram Search Engine with Patterns Combining Token, POS, Chunk and NE Information

Satoshi Sekine
Computer Science Department
New York University
sekine@cs.nyu.edu

Kapil Dalwani
Computer Science Department
Johns Hopkins University
kapildalwani@gmail.com

Abstract

We developed a search tool for ngrams extracted from a very large corpus (the current system uses the entire Wikipedia, which has 1.7 billion tokens). The tool supports queries with an arbitrary number of wildcards and/or specification by a combination of token, POS, chunk (such as NP, VP, PP) and Named Entity (NE). It outputs the matched ngrams with their frequencies as well as all the contexts (i.e. sentences, KWIC lists and document ID information) where the matched ngrams occur in the corpus. It takes a fraction of a second for a search on a single CPU Linux-PC (1GB memory and 500GB disk) environment.

1. Introduction

We developed a search tool for ngrams extracted from a very large corpus (the current system uses the entire Wikipedia, which has 1.7 billion tokens). The tool supports queries with an arbitrary number of wildcards and/or specification by a combination of token, POS, chunk (such as NP, VP, PP) and Named Entity (NE). It outputs the matched ngrams with their frequencies as well as all the contexts (i.e. sentences, KWIC lists and document ID information) where the matched ngrams occur in the corpus. It takes a fraction of a second for a search on a single CPU Linux-PC (1GB memory and 500GB disk) environment.

This system is an extension of the previously published ngram search engine system (Sekine 08). The previous system can only handle tokens and unrestricted wildcards in the query, such as "* was established in *". However, being able to constrain the wildcards by POS, chunk or NE is quite useful to filter out noise. For example, the new system can search for "NE=COMPANY was established in POS=CD". This finer specification reduces the number of outputs to less than half and avoids the ngrams which have a comma or a common noun at the first position or location in-formation at the last position.

The new system can output information on the documents from which the matching ngrams are extracted. In the current system, which uses the Wikipedia, the document information is the title of the Wikipedia page. For example, we can often find the person name in the title for the matched ngram for "He was born in *". Also, it is useful to have a back pointer to the entire article containing the matched ngrams to see the wider contexts.

The structure of the index is completely changed from the trie structure of the earlier system to an inverted index structure combined with an additional checking mechanism. The index size has been reduced greatly, from 2.4TB to 500GB, with a minor sacrifice in search speed.

2. Background

Large-scale linguistic knowledge discovery is needed to support semantic analysis for NLP applications. This is the so-called "knowledge bottleneck" problem and many researchers have tried to solve the problem using statistical methods on a large corpus (Hearst 92) (Collins and Singer 99) (Brin 99) (Hasegawa et al. 04). 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). Now, one of the major problems is the search. We need a capability for searching for patterns in a large corpus which is both fast and as flexible as possible. One of the solutions is to segment the entire corpus into small pieces and assign a CPU to each segment to search for the pattern, and gather the results in a map/reduce paradigm. However, this approach requires a very large number of machines, which is not affordable for many academic researchers.

Another solution to the problem is to use available search programs, such as Lucene. However, we need flexibility, for example, we need the capability to exactly match a query ngram with wildcards, and the capability to provide additional information, such as POS, chunk and/or NE. We found that it is possible to modify Lucene in order to achieve our goal, but we concluded that it was necessary to develop our own search system in order to allow for future extensions.

Resnik's system (Resnik 03) has similar functionalities to the functionalities provided in this system, but we believe we have improved on it in terms of scalability and speed. The demo and the project web page are no longer available.

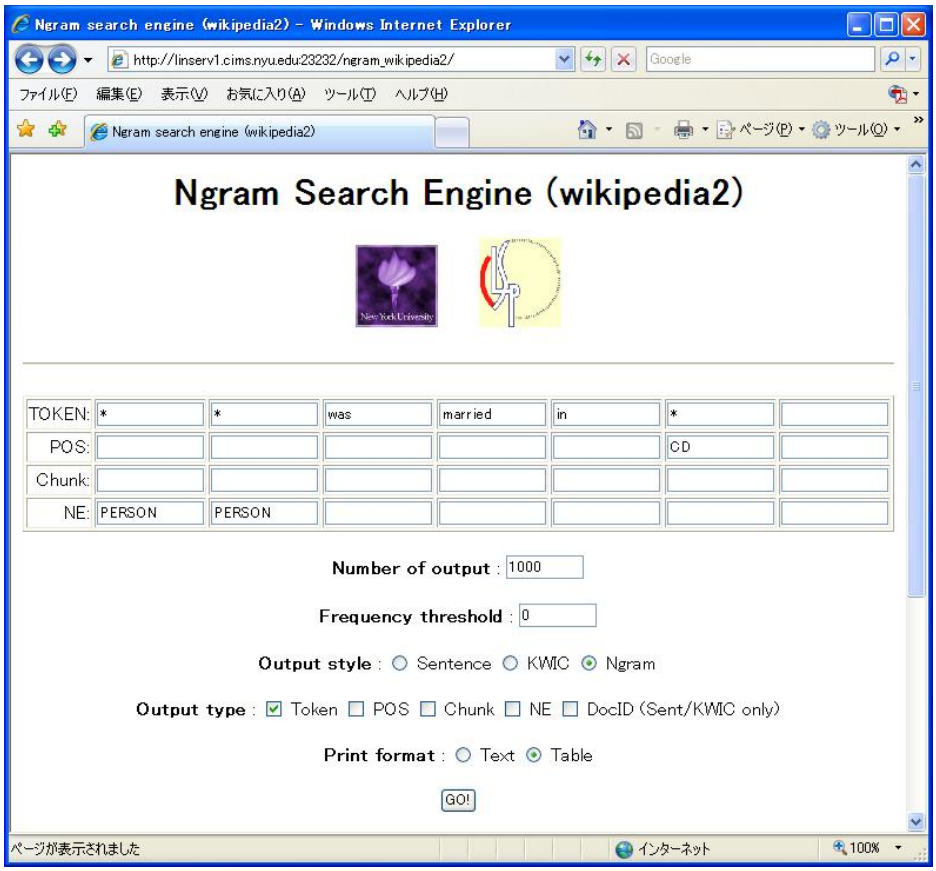

Figure 1. Query page

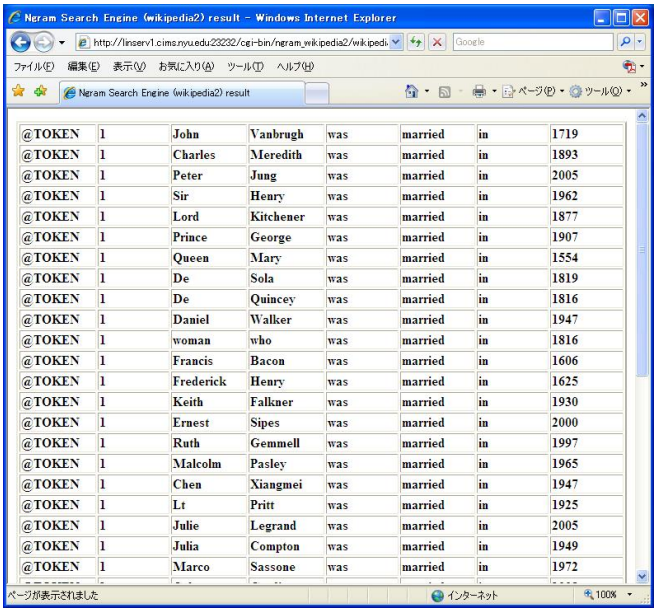

Figure 2. Output Page (ngram)

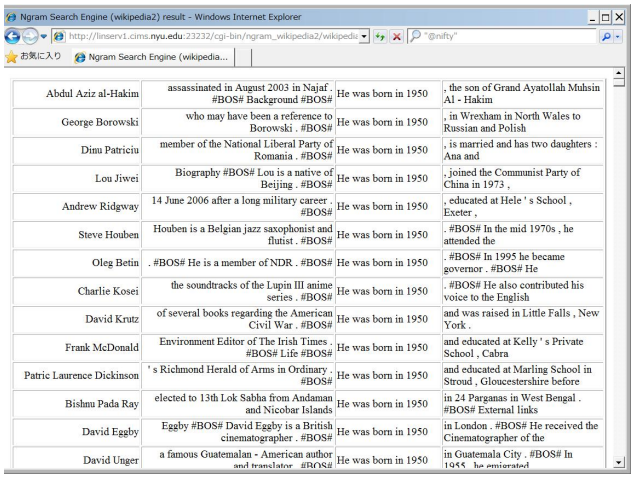

Figure 3. Output Page (KWIC with Doc ID)

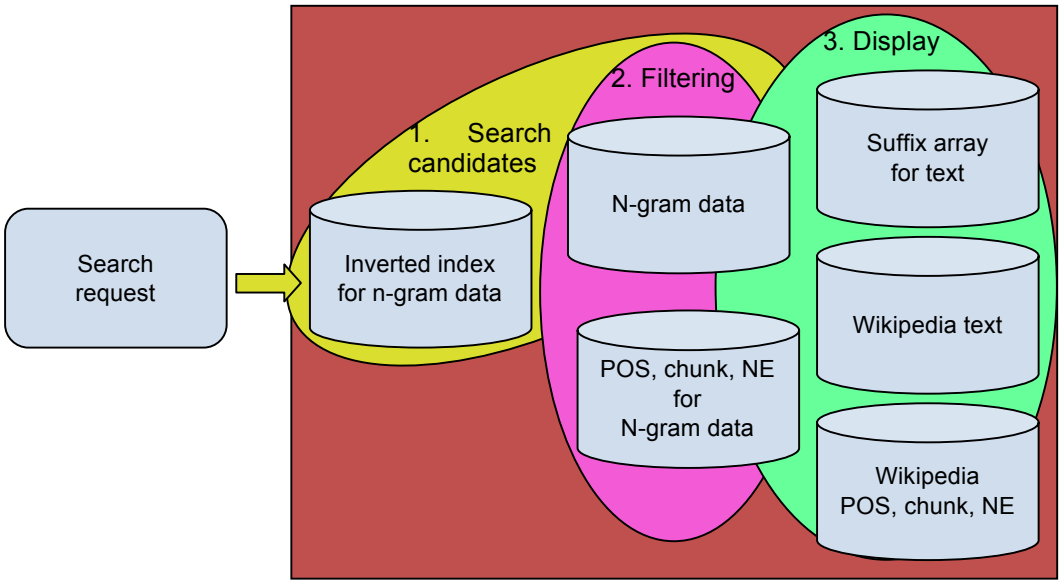

Figure 3. Data and Algorithm Overview

3. Snapshot

The Figures 1, 2 and 3 show a snapshot of the system. Figure 1 is the query page. Figure 2 is the output page of ngram output and Figure 3 is the output page of KWIC output with document ID (i.e. Wikipedia entry title).

In the query page (Figure 1), the user types the query ngram with tokens, POS's, chunk or NE information up to 7gram. The user can also specify the number of outputs, the frequency threshold (the minimum frequency to be displayed), the output style (sentence, KWIC or ngram), the output type (token, POS, chunk, NE and/or document information in case of sentence or KWIC output) and the print format (in text or table). The output will be displayed according to the specifications.

4. Data and Search Algorithm

4.1 Data

We used Wikipedia as the target corpus in the current system. Google Ngram can also be used, but because it doesn't contain the original sentences, we chose Wikipedia and create ngram data out of it by ourselves in order to show the original data from where the ngrams are extracted. It is static html documents of Wikipedia as of 18:12, June 8, 2008 version, provided at the following URL http://static.wikipedia.org/downloads/2008-06/en/. It has 1.7 billion tokens, 200 million sentences and 2.4 million articles. The sentences were tagged by the Stanford POS tagger and NE tagger (Stanford tagger), and assigned chunks by the OAK system (OAK system). The same data (actually, the current data available at the site has more annotations than the data explained here) is available at the following URL:
http://nlp.cs.nyu.edu/wikipedia-data.

The numbers of distinct ngrams are shown in Table 1. The numbers are comparable to the Google ngram data, as we have no frequency threshold. We made up to 7grams instead of 5grams in Google ngram. We did not collapse the digits unlike Google Ngram data.

Table 1. Number of distinct ngrams

N	Wikipedia ngrams	Google ngram
1	8M	13M
2	93M	315M
3	377M	977M
4	733M	1,314M
5	1,006M	1.176M
6	1,173M	-
7	1,266M	-
#	0	30

#: frequency threshold

4.2 Algorithm Overview

Figure 3 shows the overview of the steps in the ngram search engine and the data. Basically, there are three steps in the search, 1) searching candidates, where the system search candidate ngrams which matches to the query using tokens, 2) filtering, where the candidate ngrams are filtered using the constraint of additional information (POS, chunk and NE), and 3) displaying the results to the user. In the following subsections, we will describe each step and data in detail.

4.3 Search Candidate

Searching candidates is the first step in the search. It tries to make a list of matched ngrams using an inverted index for the tokens given in the query. Because of this, the query needs to have at least one token. Inverted index is a standard technique to search items. In our case, the inverted index is created for tokens at each position of all length of ngrams (note that we have unigram to 7gram in the data). The inverted index essentially contains a set of ngramIDs that have a certain token at a certain location of certain length of ngram. Because the number of ngrams for each index are varid (from 55 million which is the number of "," at a certain position, to the number of very infrequent (e.g. singleton) tokens, such as "Mizuk" or "consiety". In order to save time and disk space, we used three types of posting list (inverted index for tokens) implementations; bitmap, list of ngram IDs and encoding into pointer in the case of singletons.

The bit map technique is used only those whose frequency is more than 1% of all the ngrams. For example, the number of distinct 7grams is 1.27 billions, tokens whose frequencies are more than 12.7 million use this strategy. Because of the implementation time limitation, we did not use any compression technique. We used 1.27 billion bits in case of 7gram, and the bit corresponding to a certain ngramID is on (1), if the ngram has the token at the certain position. Looking up the information is very fast, despite the length of the inverted index for the token.

If the frequency of the token is one (singleton), we put the information in the area of pointer by setting up the top bit on (Note that the maximum number of ngram, 1.27 billions can be expressed by 31 bits).

Otherwise, we use a list. Because the list can be created in advance, we can use the static list, instead of dynamic link list which needs more space.

When there are more than two tokens in the query, we have to match the lists. Matching two lists of length n and m can be implemented in min(O(n+m), O(n log(m)), O(m log(n))). Unless n or m is very small, we need to use the matching algorithm by looking at the list sequentially in time O(n+m), which is not very fast even we can sort the index in advance. In order to speed up the index matching, we implemented "look ahead" algorithm (Moffat and Zobel 96). Once we find a match, we skip some of the elements in the list and jump to the element at a certain distance. If the new element found by jumping is still smaller than the element we are currently looking for, we can earn the time of looking the skipped elements. If the jumped element is bigger than the one we are looking for, we will go back to the original position and searching continues from the next element. It wastes only one look up. Based on the algorithm, it is empirically most efficient to look ahead the square root of the index size when you advance the pointer at the index matching.

4.4 Filtering

The second step, filtering, is needed to match the ngram to the requested POS, chunk and/or NE information. If we implement all those information for each ngram using one byte each, we need 21 extra bytes in case of 7gram. It results in 27GB. So, we encoded the information by combining the information from the individual tokens of each ngram. For example, for 7grams, the POS information for the 7 tokens takes 7 bytes, if we record the POS information for a single token in 1 byte. However, the actual number of combinations of 7 POS's is not that large, because there are many ngrams that have the same POS patterns. For example, the number of POS pattern for 7gram is 125 million and the number of POS patterns for 3gram is 61 thousand, which can be encoded in less than 4 bytes, yielding a large reduction in disk size (in total 200MB) .

We check if the candidate ngram can match the query by finding the POS, chunk or NE information satisfy the query, if such information is requested.

4.5 Display

The third step, display, is done once the ngrams to be displayed are found. The use can specify the following information in addition to the ngram query.

- Number of output (default 1000)
- Frequency threshold (default 3)
- Output style: sentence, KWIC or Ngram
- Output type: token, POS, chunk, NE, DocID
  User can specify multiple types
  DocID can work only for sentence and KWIC
- Print format: text, table (html-table)

Display will be done according to the choice of those options.

The ngrams is sorted in the order of frequency in advance so that it can easily display the ngrams in the order of frequency. The suffix array is used to display the sentence and KWIC list quickly. Each ngram has information of the starting position and ending position in the suffix array it matches, so it can find the matched sentences very quickly. The POS, chunk and NE information is stored in parallel to the text information so that it can be displayed quickly when it is requested. Document information, including the title of the document (in our case the title of Wikipedia page) is displayed using the offset information in the text.

4.6 Data Size

We use 6 kinds of data as shown in Figure 6. Inverted index is used in the candidate search and its size is 108GB. It contains bit map index and list index. The

ngram data, including the token information and the position in the suffix array is 260GB. It is POS, chunk and NE information for unigram to 7gram, which is used in the filtering, is 100GB. As is explained in section 4.4, the information is compressed. The size of suffix array and text data is 8GB each (as each token and pointer is encoded in 4 bytes and the total text is 1.7 billion) and the size of POS, chunk and NE information is 2GB each (as each information is encoded in 1 byte). The size of other information, such as document information, dictionary information, POS, chunk and NE label information etc, is 40GB. So, the total data size of the system is about 500GB. We understand the size can be reduced easily without a big loss of the speed and usability.

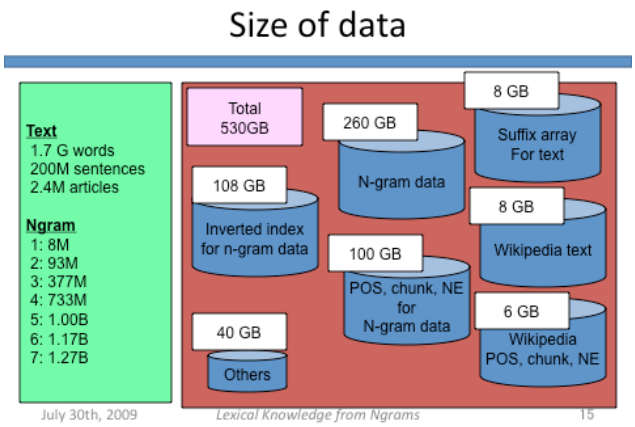

Figure 4. Data size

5. Evaluation and Demo System

In order to evaluate the accuracy and the speed of the system, we created 600 sample queries and test the system. The samples are extracted from existing ngrams, replacing zero to two tokens by wildcards, POS, chunk or NE randomly and seeing if the original ngram is included in the searched results. Note that it is possible to include extra ngrams because of generalization, but it should not be judged as error. The system runs without any incorrect output and the average runtime for each query was 0.34 second. The demo system is avail-able at the following URL. We have already received requests to index different corpora (including Open American National Corpus).

Demo URL: http://nlp.cs.nyu.edu/nsearch.

6. Future Work

The future work includes the following:

1. Including other information

We are planning to implement a search engine using structured data, such as dependency or parse. For example, "tgrep" in Penn Treebank provides this functionality, but we believe our implementation can improve the scalability and the search speed.

2. Longer ngrams

Our current implementation can be extended to longer ngram search, compared to the previous implementation using trie structure. The longer queries are desirable for semantic knowledge discovery.

3. Smaller index

We have observed that the index can be reduced without great loss of speed. For example, the bitmap implementation is can be compressed by a standard compression technique, but also ngram data and other data can be candidates of more compression.

4. Reduce the indexing requirements

The current index creation needs a machine with a large memory. We used a machine with 96GB memory. This is not desirable if people want to use this search engine on many different corpora. We would like to find method for indexing in smaller memory machine.

7. References

Michael Collins and Yoram Singer. "Unsupervised Models for Named Entity Classification". 1998. In Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP-99).

Marti A. Hearst. "Automatic Acquisition of Hyponyms from Large Text Corpora". 1992. In Proceedings of Conference on Computational Linguistics (COLING-92), Nantes, France.

Takaaki Hasegawa; Satoshi Sekine; Ralph Grishman "Discovering Relations among Named Entities from Large Corpora". 2004. In the proceedings of Association of Computational Linguistics (ACL-04). Barcelona, Spain

Alistair Moffat and Justin Zobel. "Inverted Files for Text Search Engines". 2006. ACM Computing Surveys, 38(2):1-56, July 2006.

Philip Resnik and Aaron Elkiss. "The Linguist's Search Engine: Getting Started Guide". 2003. Technical Report: LAMP-TR-108/CS-TR-4541/UMIACS-TR-2003-109, University of Maryland, College Park

Satoshi Sekine. "A Linguistic Knowledge Discovery Tool". 2008. In Proceeding of COLING08.

Stanford tagger: http://nlp.stanford.edu/software/tagger.shtml

OAK system: http://nlp.cs.nyu.edu/oak

Wikipedia tagged data: http://nlp.cs.nyu.edu/wikipedia-data