New Tools for Web-Scale N-grams

Dekang Lin,[*] Kenneth Church,[†] Heng Ji,[♮] Satoshi Sekine,[▽] David Yarowsky,[†] Shane Bergsma,[‡] Kailash Patil,[†] Emily Pitler,[◇] Rachel Lathbury,[♯] Vikram Rao,[♭] Kapil Dalwani,[†] Sushant Narsale[†]

[*]Google, Inc. (lindek@google.com),
[†]Johns Hopkins University (kenneth.church@jhu.edu, yarowsky@cs.jhu.edu, kailash@jhu.edu, kapild@cs.jhu.edu, sushant@jhu.edu),
[♮]City University of New York (hengji@cs.qc.cuny.edu),
[▽]New York University (sekine@cs.nyu.edu),
[‡]University of Alberta (sbergsma@ualberta.ca),
[◇]University of Pennsylvania (epitler@seas.upenn.edu),
[♯]University of Virginia (rlathbury@virginia.edu),
[♭]Cornell University (vr59@cornell.edu)

Abstract

We introduce a new set of tools for working with web-scale N-gram data. These tools lower the barrier for working with web-scale text, and create a new platform for acquiring large-scale linguistic knowledge. They will allow novel sources of information to be applied to long-standing natural language challenges.

1. Introduction

The overall performance of machine-learned NLP systems is often ultimately determined by the size of the training data rather than the learning algorithms themselves (Banko and Brill, 2001). The web undoubtedly offers the largest textual data set. Text from the web has been found useful in a diverse range of NLP applications (Kilgarriff and Grefenstette, 2003).

While the web provides a fantastic linguistic resource, collecting and processing data at web-scale is beyond the reach of most academic laboratories. Previous approaches have relied on search engines to collect online information (Grefenstette, 1999; Turney, 2001; Keller and Lapata, 2003; Chklovski and Pantel, 2004; Lapata and Keller, 2005; Nakov and Hearst, 2005). There are a number of drawbacks in using search engines (Kilgarriff, 2007). While broad-coverage search engines can work well when counts for a small number of queries are needed, they are hopelessly inefficient when millions of queries are required. Search engines are therefore inadequate for building large-scale linguistic resources, such as lists of named-entity types or clusters of distributionally-similar words. Furthermore, search engines offer a very impoverished query language. Queries that match capitalization, punctuation, and annotations such as part-of-speech are not supported. There have been efforts to develop search engines that support different kinds of linguistic queries, but so far these have used much smaller document collections than those indexed by commercial search engines (Cafarella and Etzioni, 2005; Banko et al., 2007). Another option is to apply NLP-strength post-processing to the pages returned by a search engine for a particular query (Nakov and Hearst, 2005). For efficiency reasons this is again only possible at a small-scale.

An alternative to processing web-scale text directly is to use the information provided in an *N-gram corpus*. An N-gram corpus is an efficient compression of large amounts of text. An N-gram corpus states how often each sequence of words (up to length N) occurs. To keep the size manageable, N-grams that occur with a frequency below a particular threshold can be filtered.

The data of Brants and Franz (2006), commonly referred to as the Google N-gram Corpus, provides a widely-used corpus of N-gram counts, taken from a trillion words of online text. A number of recent NLP systems have used counts from this corpus (Vadas and Curran, 2007; Yuret, 2007; Kummerfeld and Curran, 2008; Bergsma et al., 2009). Although this N-gram data is much smaller than the source text from which it was taken, it is still a very large resource, occupying approximately 24 GB compressed, and containing billions of N-grams in hundreds of files. Special strategies are needed to effectively query large numbers of counts. Some of these strategies include pre-sorting queries to reduce passes through the data, hashing (Hawker et al., 2007), storing the data in a database (Carlson et al., 2008), and using a trie structure (Sekine, 2008). While these strategies allow for much faster retrieval of information than using a search engine, the kinds of information that can be queried remains fairly impoverished, as the Google N-gram Corpus contains only words and counts.

We propose tools for working with enhanced web-scale N-gram corpora that include richer levels of source annotation, such as part-of-speech tags. We describe a new set of search tools that make use of these tags, and collectively lower the barrier for lexical learning and ambiguity resolution at web-scale. These tools were developed during a six-week research workshop at the Center for Language and Speech Processing at Johns Hopkins University, and will be released publicly under an open-source license.

2. Overview

Section 3. describes the N-gram data used in our implementations of the different search tools.

The tools have different strengths and are therefore appropriate for different uses. Indeed, we relied on each of these tools for different NLP applications during our research at the workshop. Feedback from users helped define the ultimate functionality of the tools.
The first set of tools allows for very expressive search queries using the idea of *rotated N-grams* (Section 4.). A simple query language allows the user to construct detailed search patterns and gather information from the matching N-grams. For applications that require arbitrarily expressive searches over words and tags, including the use of regular expressions, these tools should be used. They are the main focus of this paper.
Another set of tools are based on suffix arrays (Section 5.). These tools provide very efficient retrieval of the top-K results for a query. They also provide the quickest access to frequency information for exact-match queries (i.e., ones that do not use tags or regular expressions).
Finally, we describe tools that additionally index syntactic chunk and named-entity information (Section 6.). These tools also link the retrieved N-grams back to their location in the original text. They are therefore most useful for applications requiring richer levels of annotation and cross-reference.

3. Tagged N-gram data

3.1. Web-scale N-grams

The data of Brants and Franz (2006) provides a widely-used corpus of N-gram counts, taken from a trillion words of online text. We built the tools in Section 4. and Section 5. to make use of a new N-gram corpus, created from the same source text as this earlier N-gram data, but with several enhancements. First, duplicate sentences are discarded. This is important because in a web corpus, the use of legal disclaimers and other boilerplate text causes some sentences to occur millions of times, skewing the N-gram statistics.[1]
Second, to filter garbage text, we only keep sentences 20 to 300 bytes long and with less than 20% of characters being digits or punctuation. Third, we convert all digits to '0' and replace all URLs with '⟨URL⟩' and e-mail addresses with '⟨EMAIL⟩.'
The remaining sentences were part-of-speech tagged using the tag-set of the Penn Treebank (Marcus et al., 1993) via the TnT Tagger (Brants, 2000). TnT was used because of its accuracy (96.7% on the Treebank) and efficiency.[2]

[1]Likely duplicate sentences in the original Google N-gram corpus have recently been observed, and their effects on extracting lexical information discussed, at `nlpers.blogspot.com/2010/02/google-5gram-corpus-has-unreasonable.html`

[2]TnT is an HMM-style tagger, and does not use bi-lexical features. It can therefore perform poorly on tagging decisions involving bi-lexical dependencies, such as distinguishing between a past-tense verb (VBD) and a past participle (VBN), where the specific relation between the verb and noun is important (e.g. *troops stationed* (VBN) vs. *troops vacationed* (VBD) or *lessons learned* (VBN) vs. *students learned* (VBD)). Overall, tagging errors are fairly consistent in the corpus, and are reflected in the aggregate statistics (more on this in Section 5.). One of the applications we investigated was to design a post-processor to fix VBN-VBD errors using N-gram statistics.

After tagging, the N-gram counts were then collected, keeping only unigrams that occur more than 40 times and 2-to-5-grams that occur more than 10 times. There are 4.1 billion N-grams in the resulting database. Each N-gram entry in the database also indicates how often the N-gram occurs with each part-of-speech tag sequence. E.g.:

```
flies 1643568 NNS|611646 VBZ|1031922
```

– indicating that flies occurred 1.6 million times, and was tagged roughly six hundred thousand times as a plural noun and one million times as a verb.
To allow efficient access to the N-gram entries, and to collect and aggregate information over the entries, we developed data structures and tools, which we discuss in Section 4. and Section 5..
We plan to make the new N-gram data available to the NLP research community.

3.2. Wikipedia N-grams

The tools in Section 6. use a tagged and cleaned Wikipedia corpus,[3] automatically annotated with part-of-speech, chunk and named-entity information. Part-of-speech and chunk tags were annotated by the OAK system,[4] and named-entities were annotated by the Stanford NE tagger.[5] We extracted N-grams from the annotated corpus, including N-grams up to seven tokens in length, without any frequency thresholds. The corpus contains 1.7 billion tokens and there are 4.55 billion unique N-grams in the resulting N-gram database. The data is available under GNU Free Documentation License.

4. Tools for Rotated N-grams

This section describes our first set of tools for matching and counting N-grams. We first describe the notion of a *rotated N-gram*. We discuss how our data is structured using this concept. We then describe the programs, patterns, and commands that work with this data. The tools can be used in batch mode or when a block of N-grams is returned using a search key, and we discuss the advantages of each approach. Finally, we describe some applications of our tools, including an approach to creating semantic clusters from the distribution of phrases in the rotated N-gram data

4.1. Data organization

We would like to take an arbitrary input word or phrase and find all the N-grams containing that word. Once we have these N-grams, we can match them against arbitrarily complex patterns, such as regular expressions. E.g., suppose we are searching for expression involving the word *cheetah*. If we store the N-grams in alphabetical order, all the N-grams that begin with *cheetah* will be sequential, but those that contain *cheetah* in later positions will be distributed throughout the data. An expensive linear pass through all the data would therefore be required to retrieve all the matching N-grams.
One solution is to store multiple copies of each N-gram, *rotated* so that different words occur at the first position in

[3]http://nlp.cs.nyu.edu/wikipedia-data

[4]http://nlp.cs.nyu.edu/oak

[5]http://nlp.stanford.edu/software/CRF-NER.shtml

different copies. A phrase like "faster than a cheetah" will be rotated three times (with the "><" marker indicating the pivot of the rotation):

```
faster than a cheetah
than a cheetah >< faster
a cheetah >< than faster
cheetah >< a than faster
```

On the right-hand-side of the rotation marker, the words are stored in reverse order (this facilitates the clustering detailed in Section 4.6.). Each rotated version is stored with the corresponding count and part-of-speech tag data for the original N-gram. Once all the N-grams are rotated, we sort them alphabetically. In the new version, all the N-grams containing *cheetah* will therefore be consecutive. We call the input query (here, the word *cheetah*) the **search prefix** as it is the prefix of a block of rotated N-grams, and is used to retrieve this block. A search prefix can be any number of tokens up to the length of N-grams in the data.
We divide the rotated N-grams into 992 files of roughly 500MB each (unzipped). No prefix spans multiple files. We build an index over these files. When a search prefix is given as a query, we search the index file for its location in the data, and then seek to the appropriate location in the appropriate file, returning the matching rotated N-grams.
The matching N-grams can be further processed using the commands and patterns described below. A program called `search_prefix` takes as arguments a search prefix, an index (either one to load from disk or the address of a running index server) and an optional command (or set of commands) to run on the matching N-grams (Section 4.3.).
Rather than specifying a search prefix, the N-grams can also be processed in batch mode. The 992 rotated files can be divided among nodes in a computing cluster and processed in parallel. We used Hadoop on IBM/Google's academic cloud computing cluster. Section 4.6. describes one application of batch processing: building semantic clusters of distributionally-similar phrases.

4.2. Patterns

The patterns are mixed sequences of words and part-of-speech tags that can be specified to match exactly or with regular expressions. The patterns are specified using a fully-parenthesized, Lisp-style syntax. They consist of atomic patterns and composite patterns.
Atomic patterns match words or tags against strings or regular-expressions provided by the user. A simple pattern is (`word` = *WORD*), which is true if the token in a given position matches the provided *WORD*. Other constructions include (`tag` ~ *REGEXP*) which matches a tag against a regular expression and (`tag` in *LIST*) which is true if the tag is in the given list. Sequences of words or tags can also be matched. For example, `word-seq` matches a regular expression over the N-gram tokens:

```
(word-seq (got an? .* for Christmas))
```

This matches N-grams like "got an X-box for Christmas," "got a pony for Christmas," etc.
Composite patterns combine other patterns in union, intersection, or sequence. For example, (+ *PATTERN*), matches one or more consecutive subsequences that match the given pattern, (? *PATTERN*) can optionally match one pattern, while (`or` *PATTERN*$_1$ *PATTERN*$_2$ ... *PATTERN*$_n$) matches a sequence if any of the patterns match. The pattern (`seq` *PATTERN*$_1$ *PATTERN*$_2$ ... *PATTERN*$_n$) matches a consecutive sequence of the given patterns, as in:

```
(seq (word = a)
     (word = river)
     (tag ~ [^N].*))
```

This pattern matches all the instances where *a river* occurs and is not followed by another noun, matching instances like *a river runs*, while excluding matches for phrases like *a river boat* or *a river basin*.

4.3. Commands

A command, also known as an *extractor*, processes the results returned by matching the patterns against the N-grams. Commands can print, count, and format matching N-grams or parts of matching N-grams. For example, the command:

```
(print-ngram PATTERN [:max-match M])
```

will print all the N-grams that match the pattern (up to the first M, if the max-match option is included). The following is an example of using this command with the `search_prefix` program:

```
search_prefix SERVER 'learned'
'(print-ngram
   (seq (word = learned)
        (or (tag = DT) (tag = PRP$))
        (word = lessons)
     ))'
```

where SERVER is the hostname and port of a server hosting the index. This pattern tells us how often the past-tense verb *learned* takes the word *lessons* as a direct object. This particular command was used in a task where we looked at using N-gram counts to disambiguate VBN/VBD tags (see Footnote 2). A high count for this pattern indicates that *learned* is likely a VBN in *lessons learned*. The equivalent pattern counts are much lower for *students learned*, indicating that *learned* is a VBD in this context.
It is also possible to count how often particular strings match patterns, and how often they co-occur with other matched patterns. The *count* command can be used in batch mode as:

```
(count (seq (+ (tag ~ [NJ].*) :name NP)
            (? (word = ,))
            (word in (who which) :name RPr))
       :format ''$[NP] $[RPr]'')
```

This command counts how often sequences of nouns and adjectives (labeled as *NP*) are followed by one of two relative pronouns (RPr), and outputs the matches as follows:

```
...
recent conversation  which 10
recent debate        which 10
recent divorcee      who 60
recent meeting       which 232 who 13
recent opinion poll  which 24
...
```

Ji and Lin (2009) used similar patterns to learn which entities in text are animate (those tending to be followed by a *who*) and inanimate (those tending to followed by *which* or *where*). This information was shown to improve unsupervised person mention detection. They also constructed similar patterns to determine the grammatical gender and number of entities.

4.4. Modes of Operation

There are trade-offs between matching patterns using a search prefix or applying them to all the N-grams in batch mode. If the goal is to build a lexical resource, i.e., a collection of data that can applied in a variety of NLP applications, then batch mode is preferred, as it will extract all the information in the N-gram corpus. This was the approach taken to extracting gender, number, and animacy information.

Using the `search_prefix` programs are most useful for preliminary investigations, brain-storming, and proof-of-concept experiments. For example, sometimes counts are only needed for a small number of phrases, perhaps only those in annotated training and testing data. In this case, instantiating queries with the particular words in these examples might be faster than collecting the information for all phrases in the N-gram corpus. This is the approach we took for our experiments in VBN/VBD disambiguation.

If the `search_prefix` program will be called repeatedly, there are several ways to improve its efficiency. First of all, we provide functionality to allow a set of commands to be applied together. The commands then operate in parallel on the same block of returned N-grams. Secondly, it may be useful to develop an automatic way to choose a good search prefix, if more than one prefix is possible for the same pattern. If a common word is used, very large blocks of N-grams will be processed. One strategy we employed was as follows: we found the longest phrase that occurs in the instantiated pattern. In the case of ties, we chose the phrase whose first token had the lower unigram count. The unigram count is only a proxy for *number of N-grams that a phrase occurs in*. We really want to minimize the latter. However, the simpler approach was sufficient for our applications.

4.5. Pattern-Matching Applications

We already mentioned some applications of our tools. We used them to acquire lexical knowledge for improving part-of-speech tagging (VBN/VBD disambiguation) and also to learn noun gender, number, and animacy for person mention detection. In addition, significant work was done using these tools to extract lexical knowledge for classifying count vs. mass nouns, and to determine adjective order in generation.

Our tools and data will be especially helpful in a broad class of applications where mining information from search engines can conflate phrases (e.g. *Martin Luther*) with longer phrases that have the original as a prefix (e.g. *Martin Luther King Jr. Boulevard*).

For example, consider the problem of predicting noun countability. Countable nouns include *river* and *avocado*. For these, it is correct to refer to *a river*, *one river*, *three rivers*, *many rivers* etc. Uncountable, mass nouns include *water* and *luggage*. One cannot say *a water*, whereas *some water* and *much water* are acceptable.

There are reliable corpus-based indicators of countability. We can simply count how often a noun phrase occurs with countable or mass-specific pre-modifiers (Baldwin and Bond, 2003; Lapata and Keller, 2005; Peng and Araki, 2005). For example, we could determine countability by contrasting the corpus frequencies of *much water* versus *many water*. If we naïvely apply this pattern, however, we will match cases where water is actually only a prefix of a larger phrase, such as *many water bottles/towers/molecules/* etc. Clearly, "we need some mechanism for detecting [noun phrase] boundaries" (Baldwin and Bond, 2003). While corpus-based approaches have used part-of-speech tagging to detect phrase boundaries (Baldwin and Bond, 2003), research using search engines has, out of necessity, neglected the issue, and achieved lower performance (Peng and Araki, 2005; Lapata and Keller, 2005).

Similar issues will occur when using web-scale pattern-matching for any lexical property, whenever a noun occurs at the beginning or ending of a pattern.[6]

Our tools offer a solution: they allow the use of web-scale statistics, without compromising on the quality of the search patterns. Here, we can include requirements on neighbor tags as part of the search pattern. We saw an example of this pattern in Section 4.2. for getting counts for the word *river*. Here is another example that uses the count command in batch mode to extract the relevant statistics for matching phrases:

```
(count (seq (or (word ˜ [Mm]uch)
                (word ˜ [Mm]any)
                :name PreM)
            (+ (tag ˜ [NJ].*) :name NP)
            (tag ˜ [ˆN].*))
       :format ''$[NP] $[PreM]'')
```

The final pattern in the sequence matches any tag that does not begin with *N*, and therefore identifies noun boundaries. The command prints noun phrases and their count with pre-modifiers *much* and *many*. These commands can also be instantiated with specific nouns (for example, only those nouns in annotated training and testing data). We would then use the the aforementioned `search_prefix` program to extract the relevant information.

4.6. Semantic Clustering of N-grams

We now describe one large-scale application that uses the rotated N-gram data: a web-scale distributional clustering of phrases. We produce a clustering with ten million phrases via K-means clustering. We make this data publicly available.

[6]The problem may be less severe when the noun occurs at the beginning of the pattern. In such cases, the noun (e.g. *water*) is likely the head of any longer phrase that is matched (e.g. *bath water*), since it occurs as the suffix of this longer phrase. Conflating the noun with the longer phrase may therefore not be harmful if phrases and their heads tend to agree in the property of interest (e.g. countability).

Cluster 825		Cluster 883		Cluster 286	
Nissan Maxima	*right:*car	secrecy agreement	*left:*signed	Pramod Kumar	*left:*shri
Nissan Altima	*right:*parts	00-year lease agreement	*left:*sign	Krishna Kumar	*right:*singh
Buick Century	*right:*cars	deed of agreement	*right:*signed	Anil Kumar	*right:*kumar
Nissan Pathfinder	*left:*0000	memorandum agreement	*left:*signing	Dinesh Kumar	*left:*dr.
Infiniti G00	*right:*recalls	technology transfer agreement	*left:*under	Rajesh Kumar	*left:*mr
Nissan Sentra	*right:*engine	twinning agreement	*right:*between	Ashok Kumar	*right:*sharma
Pontiac Sunfire	*left:*new	operational agreement	*left:*into	KUMAR	*left:*dr
Mazda Miata	*left:*ford	memorandum of cooperation	*right:*with	Virendra	*right:*gupta
Isuzu Rodeo	*left:*used	0-year agreement	*left:*entered	Arun Kumar	*left:*mr.
Hyundai Elantra	*right:*sale	co-operative agreement	*left:*a	Kamlesh	

Table 1: Example phrasal clusters acquired using distributional clustering over N-gram data. For each cluster, the first column indicates the most canonical phrases in the cluster (by similarity to cluster centroid) while the second column indicates the most highly weighted elements of the centroid feature vector (*right:* indicates a word occurring on the right, *left:* indicates a word on the left). Recall that all digits were pre-converted to '0' in the N-gram source corpus.

Clusters allow us to generalize. Recently, a number of researchers in NLP have successfully used clusters to improve the performance of systems trained using supervised machine learning (Miller et al., 2004; Koo et al., 2008; Lin and Wu, 2009). Essentially, even if a word or phrase has not been observed in the training data, we may process it correctly in unseen data provided we have information about its cluster membership.

For example, suppose we have training data that indicates *Honda Accord* is a car. When applying our system, we may see phrases like *peace accord* and *Nissan Maxima*. Based purely on the identical grammatical heads, we might suspect *peace accord* is a car, while we would have no information for *Nissan Maxima*. However, if we knew the cluster memberships of the entities involved (Table 1) where *Honda Accord* is in Cluster 825 and *peace accord* is in Cluster 883, we could learn that all entities in Cluster 825 tend to be cars while none of the phrases in Cluster 883 are so. This knowledge would let us make the correct determination at test time.

Most previous large-scale efforts use the algorithm developed by Brown et al. (1992), e.g. Miller et al. (2004) and Koo et al. (2008). This algorithm operates over words. A word in isolation, however, can be ambiguous (like *accord*), whereas phrases are less so. Lin and Wu (2009) produce a clustering with 20 million phrases, but neither the web documents they used for clustering, nor their resulting clusters, are publicly available.

We follow Lin and Wu (2009) in using K-means to cluster phrases, but make our clusters publicly available. To make web-scale clustering practical, we cluster using our rotated N-gram data. We first define what we mean by *phrase*, and then apply this definition operationally to build our clusters.

A phrase is intuitively a sequence of words that functions as a single unit in a sentence. For example, while *degree in computer science* is a phrase (it can play many roles in text), we regard *degree in computer* on its own as not a phrase. Frequency is therefore an inadequate filter; *degree in computer* occurs 52,181 times in our corpus, and is of course more frequent than *degree in computer science*, which occurs 39,640 times. Lin and Wu (2009) define phrases to be sequences of words that are queried on a search engine. While this is a potentially useful heuristic, query data is not made publicly available. We instead use the following idea: N-grams that have low entropy of context are not phrases. For example, if *degree in computer* is always followed by one of a few specific tokens, it is not itself a phrase. We approximate choosing N-grams with high entropy of context by ensuring a certain number of unique left and right contexts co-occur with the N-gram, that a certain percentage of the left and right contexts are stop words, and that the phrase itself obeys some constraints such as not beginning with a stop word, not ending in a conjunction, preposition, or determiner, and not containing certain punctuation.

The main idea of clustering phrases using N-gram data is to use higher-order N-grams to extract the context (i.e. distributional) features for lower-order phrases. Phrases that are similar should have similar distributional features. This is a simple application of Harris (1985)'s distributional hypothesis: words (and phrases) that occur in similar contexts tend to have similar meanings.

We simultaneously identify phrases and extract their distributional contexts in a single pass through the N-gram data. For example, consider passing through the block of N-grams beginning with the phrase *degree in engineering*:

```
degree in engineering               31978
...
degree in engineering >< PhD          110
degree in engineering >< accredited    21
degree in engineering >< four-year     72
...
degree in engineering mechanics        63
degree in engineering management       63
```

As we encounter these N-grams, we have the total count of the phrase *degree in engineering* (31978) and counts for various left-contexts (*left:*PhD 110, *left:*accredited 21, *left:*four-year 72) and right-contexts (*right:*mechanics 63, *right:*management 63). We include as context features both adjacent tokens and those separated from the phrase by a stop word.

We use the Map-Reduce distributed programming paradigm (Dean and Ghemawat, 2008), and run experiments using Hadoop on IBM/Google's academic cloud computing cluster. Our mapper creates the count vectors for each example, while the reducer sums the frequency of each context across all examples (the global context counts). We then do another pass over the context vectors, converting the context counts to the mutual information between the phrases and the contexts (dividing the phrase-context co-occurrence count by the count of the phrase and the global count of the context, determined in the reduction part of the previous pass). This produces the feature vectors needed for clustering.

Finally, we run K-means clustering over the resulting feature vectors. We use as our distance metric the cosine similarity between feature vectors. K-means is used because it is computationally efficient and easily parallelized. We run 50 iterations using 1000 cluster centroids and random initialization.

We tuned our thresholds by running several development clusterings and assessing the results. The final run of our algorithm produced a clustering of 10 million phrases in 1000 clusters. The algorithm is found to intuitively cluster entities like cities, cars, movies, etc. Table 1 provides some example clusters. Cluster 286 is a collection of names of Indian origin. These were clustered because they occurred in the context of other Indian names. There are potentially many applications of such name collections.

We used the clusters in an experiment determining the scope of conjunctions. They were found to enable good disambiguation performance on this task, exceeding methods based on co-occurrence frequencies.

5. Suffix-array tools

A separate set of tools are based on suffix arrays. The tools take as input flat files such as the tagged N-gram data described in Section 3.. These flat files associate each N-gram key with parts-of-speech and frequency counts. We split the input files into chunks of roughly 2GBs each. The chunk size is selected so that we can afford random access within a chunk. We build indexes (suffix arrays) for each chunk.

Suffix arrays support a number of different types of queries. Given a list of patterns, it is easy to find N-grams that match the pattern in various ways including exact match, starts with and contains. With a variation of suffix arrays described in (Church et al., 2007), the indexes can be modified to return the top-K results by counts. This option is particularly useful for queries that would otherwise flood the user with too many matches.

A part-of-speech tagger was built on top of these tools. The tagger highlighted opportunities for improving the part-of-speech tags in the input files. The tagger incorrectly tagged 'work' as a verb in 'He drove to work' because of tagging errors in the input files. If the tagging errors in the input files were random, then 'more data would be better data,' but unfortunately, TnT often makes the same mistakes again and again.

The tools mentioned above were also applied to other types of flat files including the output of our K-means clustering, as well as the output of BBN's self-organizing units on Switchboard speech (Garcia and Gish, 2006).

6. Part-of-speech, chunk, & named-entity N-gram matching in Wikipedia

We also developed a search tool for corpora with richer levels of source annotation, beyond part-of-speech (POS) tagging. We used this tool to index our annotated Wikipedia data (Section 3.2.). The tool supports queries with an arbitrary number of wild-cards and also allows restricting the wildcards to particular POS, chunk (such as NP, VP, PP) or Named Entity (NE) types (person, location, and organization). It outputs matching N-grams with frequencies as well as with all the contexts of the N-gram in the original corpus (i.e. the source sentences, keyword-in-context lists and document ID information). The tool takes a fraction of a second for a search on a single CPU Linux-PC environment (using 1GB memory and 500GB disk).

This system is an extension of the N-gram search engine system described in (Sekine, 2008). The previous system can only handle tokens in the query, such as "* was established in *." However, finer specification of the wildcards by POS, chunk or NE is quite useful for filtering out noise. For example, the new system can search "NE=COMPANY was established in POS=CD." This finer specification halves the number of matching N-grams for this query, and avoids returning N-grams which have a comma or a common noun at the first position or a location in the last position. The structure of the index is completely changed from the trie structure of (Sekine, 2008). It now uses an inverted index structure with additional checking mechanisms. The index size has reduced greatly from 2.4TB to 500GB, with a minor sacrifice in search speed.

A separate paper provides full details of the architecture and algorithms of this particular search tool (Sekine and Dalwani, 2010).

Acknowledgments

We gratefully acknowledge Frederick Jelinek and the members of the Center for Language and Speech Processing at Johns Hopkins University for hosting the workshop at which this research was conducted. We thank the IBM/Google Academic Cloud Computing Initiative for providing access to their computing cluster. We also thank the National Science Foundation, Google Research, and the Defense Advanced Research Projects Agency for sponsoring the workshop, and Thorsten Brants, Fernando Pereira and Alfred Spector at Google for their help with providing the new N-gram data.

7. References

Timothy Baldwin and Francis Bond. 2003. Learning the countability of English nouns from corpus data. In *ACL*.

Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *ACL*.

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *IJCAI*.

Shane Bergsma, Dekang Lin, and Randy Goebel. 2009. Web-scale N-gram models for lexical disambiguation. In *IJCAI*.

Thorsten Brants and Alex Franz. 2006. The Google Web 1T 5-gram Corpus Version 1.1. LDC2006T13.

Thorsten Brants. 2000. TnT – a statistical part-of-speech tagger. In *ANLP*.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479.

Michael J. Cafarella and Oren Etzioni. 2005. A search engine for natural language applications. In *WWW*.

Andrew Carlson, Tom M. Mitchell, and Ian Fette. 2008. Data analysis project: Leveraging massive textual corpora using n-gram statistics. Technial Report CMU-ML-08-107.

Timothy Chklovski and Patrick Pantel. 2004. VerbOcean: Mining the web for fine-grained semantic verb relations. In *EMNLP*.

Kenneth Church, Bo Thiesson, and Robert Ragno. 2007. K-best suffix arrays. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*.

Jeffrey Dean and Sanjay Ghemawat. 2008. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Alvin Garcia and Herbert Gish. 2006. Keyword spotting of arbitrary words using minimal speech resources. In *IEEE Int. Conf. Acoust., Speech, Signal Processing*.

Gregory Grefenstette. 1999. The World Wide Web as a resource for example-based machine translation tasks. In *ASLIB Conference on Translating and the Computer*.

Zellig Harris. 1985. Distributional structure. In *The Philosophy of Linguistics*, pages 26–47. Oxford University Press.

Tobias Hawker, Mary Gardiner, and Andrew Bennetts. 2007. Practical queries of a massive n-gram database. In *Australasian Language Technology Association Workshop*.

Heng Ji and Dekang Lin. 2009. Gender and animacy knowledge discovery from web-scale N-grams for unsupervised person mention detection. In *PACLIC*.

Frank Keller and Mirella Lapata. 2003. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484.

Adam Kilgarriff and Gregory Grefenstette. 2003. Introduction to the special issue on the Web as corpus. *Computational Linguistics*, 29(3):333–347.

Adam Kilgarriff. 2007. Googleology is bad science. *Computational Linguistics*, 33(1).

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *ACL-08: HLT*.

Jonathan K Kummerfeld and James R Curran. 2008. Classification of verb particle constructions with the Google Web1T Corpus. In *Australasian Language Technology Association Workshop*.

Mirella Lapata and Frank Keller. 2005. Web-based models for natural language processing. *ACM Transactions on Speech and Language Processing*, 2(1):1–31.

Dekang Lin and Xiaoyun Wu. 2009. Phrase clustering for discriminative learning. In *ACL-IJCNLP*.

Mitchell Marcus, Beatrice Santorini, and Mary Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In *HLT-NAACL*.

Preslav Nakov and Marti Hearst. 2005. Search engine statistics beyond the n-gram: Application to noun compound bracketing. In *CoNLL*.

Jing Peng and Kenji Araki. 2005. Detecting the countability of english compound nouns using web-based models. In *IJCNLP: Companion Volume*.

Satoshi Sekine and Kapil Dalwani. 2010. Ngram search engine with patterns combining token, POS, chunk and NE information. In *LREC*.

Satoshi Sekine. 2008. A linguistic knowledge discovery tool: Very large ngram database search with arbitrary wildcards. In *COLING: Companion volume: Demonstrations*.

Peter D. Turney. 2001. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In *European Conference on Machine Learning*.

David Vadas and James R. Curran. 2007. Large-scale supervised models for noun phrase bracketing. In *PACLING*.

Deniz Yuret. 2007. KU: Word sense disambiguation by substitution. In *SemEval-2007: 4th International Workshop on Semantic Evaluations*.