Data Analysis Project: Leveraging Massive Textual Corpora Using n-Gram Statistics

Andrew Carlson, Tom M. Mitchell, Ian Fette

May 2008
CMU-ML-08-107

Data Analysis Project: Leveraging Massive Textual Corpora Using n-Gram Statistics

Andrew Carlson[*] **Tom M. Mitchell**[*] **Ian Fette**[†]

May 2008
CMU-ML-08-107

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Machine Learning Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[†]Institute for Software Research, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Work reported herein has been supported in part under the NSF Cyber Trust initiative (Grant #0524189), under ARO research grant DAAD19-02-1-0389 ("Perpetually Available and Secure Information Systems") to Carnegie Mellon University's CyLab, by DARPA under the CALO/PAL program, and by a Yahoo! Graduate Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Keywords: machine learning, spelling correction, ontology population, large data sets

Abstract

We study methods of efficiently leveraging massive textual corpora through n-gram statistics. Specifically, we explore algorithms that use a database of frequency counts for sequences of tokens in a teraword Web corpus to correct spelling mistakes and to extract a list of instances of some category given only the name of the target category. For spelling correction, we use a novel correction algorithm and demonstrate high accuracy in correcting both real-word errors and non-word errors. For category extraction, we show promising preliminary results for a variety of categories. We conclude that n-gram statistics provide an efficient way to use information contained in a massive corpus of text using relatively simple algorithms. The report ends with a reflection on problems encountered, possible solutions, and future work.

1 Introduction

Massive textual corpora such as the World Wide Web hold great amounts of information. However, extracting machine-interpretable representations of that information is a challenge. Without a vast number of processors working in parallel, it is not currently feasible to run expensive algorithms over every web page. For example, running light-weight natural language processing (NLP) methods, such as part-of-speech (POS) tagging or noun phrase chunking, was estimated to take hundreds of days on a modern machine for a 1TB corpus, and running a syntactic parser on such a corpus would take nearly 400 years [17]. However, for some problems it has been shown that simple algorithms, given enough training data, can provide a good trade-off between computation time and accuracy [2, 17]. This report explores methods of efficiently leveraging such massive textual corpora through n-gram statistics, specifically, the n-gram data set released by Google in 2006 [3]. The data collection gives the number of times an ordered sequence of tokens occurs in a collection of public web pages containing one trillion words. Frequency counts for sequences of lengths one, two, three, four, and five are reported.

We present algorithms that use this database of frequency counts for sequences of tokens to correct spelling mistakes and to extract a list of instances of a category given only the name of the target category. For spelling correction, we use a novel correction algorithm and demonstrate high accuracy in correcting both real-word errors and non-word errors. For category instance extraction, we show promising preliminary results.

Details and analysis of the data set are presented in Section 2. Our methods and experiments for the spelling correction problem are discussed in Section 3. Preliminary work on category instance extraction is presented in Section 4. Finally, conclusions and discussion of future work are presented in Section 5.

2 Data Set

Our methods explore using a massive corpus of text through statistics derived from that corpus. For this work, we chose the Google n-gram data set [3]. The data set was released in 2006, and is available through the Linguistic Data Consortium. The data collection gives the number of times an ordered sequence of tokens occurs in a collection of public web pages containing over one trillion words. Sequences of lengths one, two, three, four, and five are reported. The data was tokenized in a manner similar to that used in the Penn Treebank corpus [14]. Punctuation is generally split into a single token, with the exception of apostrophes, which start a new token, but letters after them are kept intact (e.g. *they're* becomes *they 're*). Some deviations were made from the Penn Treebank conventions, such as keeping URLs or e-mail addresses as single tokens. The text was split into sentences, and the tokens '<S>' and '</S>' were added at the start and end of sentences, respectively.

The n-grams reported were thresholded to limit the size of the data set. Specifically, tokens that occurred fewer than 200 times were replaced by the token '<UNK>', and n-grams that occurred fewer than 40 times are not reported. Punctuation and capitalization are preserved in the data set. This data set is 87GB on disk, without any database indexes. To enable fast queries for our experiments, we loaded the data set into a MySQL database and indexed each table, which brought the total size of the database up to roughly 250GB.

A sample of the raw text in the distribution of the data set is:

```
environment is protected to     62
environment is protected while  75
environment is provided ,       516
environment is provided .       1048
environment is provided ;       43
environment is provided </S>    157
```

2.1 Data Representation

2.1.1 Database Tables

The n-gram data that we have is well suited for storage in a database. Each set of n-grams corresponds to a separate table, where the first n fields store the n tokens of the n-gram, and the last field stores the frequency of that n-gram in the teraword Web corpus. As such, we have five separate tables to store the n-grams, one for each size of n.

To efficiently store the n-gram data, we represent each token in the corpus with an integer, because the 4-byte storage size is less than the average size of storing a variable-length string (the average length of which is 8.2 characters). As such, our tables have n 32-bit integer columns (where n is the size of the n-gram), and a 64-bit integer column to store the number of times this n-gram occurred in the teraword corpus. This has a number of advantages, namely that the size of a record is a small, fixed width that makes it ideal for indexing in something like a B+ Tree. The size of a 5-gram record is only $5 * 4 + 8 = 28$ bytes.

2.1.2 Database Indexes

There are many n-grams that start with the same first few tokens. As such, indexes over multiple columns are especially important. Five rows of the 5-gram table are shown in Table 1. Observe that the first four tokens are the same (because they have the same integer token ID values). It might seem that the first token of the n-gram would be good enough to support efficient index usage, but as an example of why this is not so, consider the following. For even an uncommon token like ‘burlesque’, there are 498 3-grams that are started with this token. For common tokens like ‘Microsoft’, the problem gets worse— it appears at the beginning of over 87,000 3-grams. As a more extreme case, the token ‘the’ begins 16,658,607 distinct 3-grams, and 34,686,019 distinct 5-grams.

Without indexes, this data takes up approximately 87GB. With a single B+tree index on each table (where the index covers all the columns of the table), the on-disk size grows to over 216GB.

The spelling correction algorithm uses exact match queries for every n-gram size, so the exact match indexes on each table will allow these queries to be executed efficiently. The category extraction algorithm uses queries where completions of some prefix are required, so indexing the

token1	token2	token3	token4	token5	frequency
11168801	12903446	4147	37240	9558928	69
11168801	12903446	4147	37240	9588371	97
11168801	12903446	4147	37240	9704735	464
11168801	12903446	4147	37240	9878974	52
11168801	12903446	4147	37240	10003468	179

Table 1: Content of table 5grams

tables with the first token, then the second, then the third, etc. allows these queries to be executed efficiently.

2.2 Analysis

There are a total of 13,588,391 distinct tokens in the data set, which are combined into 314,843,401 unique 2-grams, 977,069,902 3-grams, 1,313,818,354 4-grams, and 1,176,470,663 5-grams.

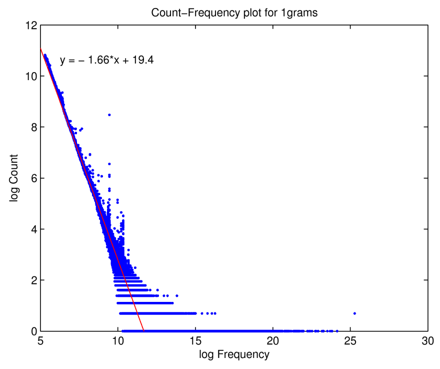
Not surprisingly, the n-grams appear to follow power-law distributions. That is, a small percentage of the n-grams account for a large percentage of the text on the web. Figure 1 shows power law plots for the 1- through 5-grams. The effects of the thresholding are clearly visible—not reporting n-grams that occurred fewer than 40 times appears to cut off a very large number of n-grams.

3 Spelling Correction

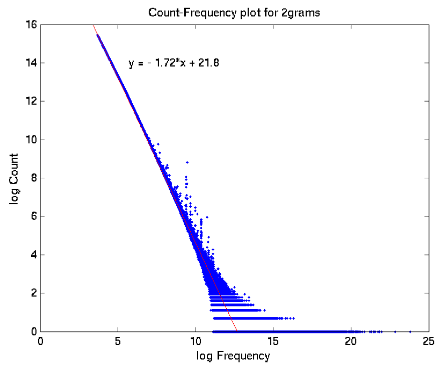
3.1 Introduction

In this section, we apply memory-based learning techniques to the problem of correcting spelling mistakes in text, using a very large database of token n-gram occurrences in web text as training data. Our approach uses the context in which a word appears to select the most likely candidate from words which might have been intended in its place, as determined by the sequence of words most frequently observed in the n-gram training data. Our novel correction algorithm extends previous work by considering each possible position where the word in question can occur in an n-gram and selecting the most discriminative position.

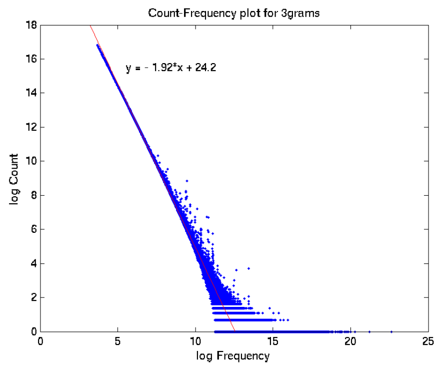
We evaluate the performance of our algorithm on both *real-word errors* and *non-word errors*. Real-word errors are lexically valid words that are not noticed by conventional dictionary-based spelling correction packages when used incorrectly (e.g. using *piece* instead of *peace*). We exceed the accuracy of previous memory-based approaches to this task for both newswire and general English text. Non-word errors are errors where a word not found in a dictionary was used instead of an intended dictionary word. We show that our algorithm can accurately recommend corrections for non-word errors by re-ranking a list of spelling suggestions from a widely used spell-checking package. The correct word is usually selected by our algorithm.



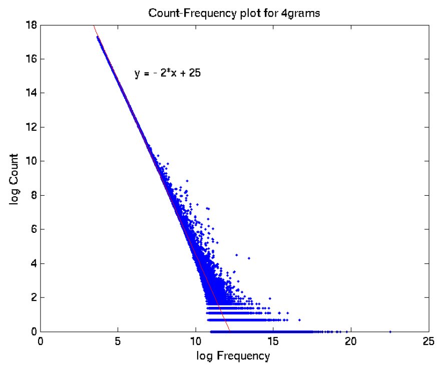
(a) 1-grams



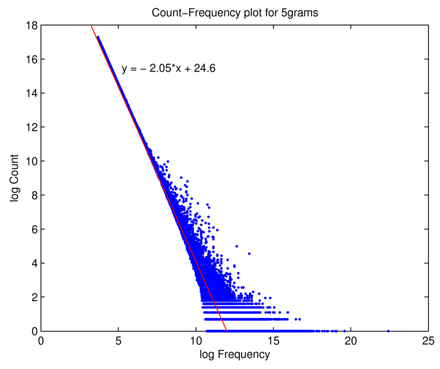
(b) 2-grams



(c) 3-grams



(d) 4-grams



(e) 5-grams

Figure 1: Count-Frequency plot of n-grams, $n \in [1, 5]$

The rest of the section is organized as follows. Section 3.2 discusses related work. Section 3.3 describes our methodology, while Section 3.4 presents results of applying that methodology on various data sets. We discuss probabilistic models used and a possible future direction of using statistical language models in Section 3.5. We present our conclusions in Section 3.6.

3.2 Related Work

Approaches to finding real-word errors have frequently taken as input a list of *confusion sets*, where a confusion set is a set of dictionary words that are often incorrectly used in place of one another. Typical types of confusion sets include homophones (e.g. *piece* and *peace*), words with low edit distance (e.g. *form* and *from*), and words like *among* and *between* where the correct word is determined by the rules of grammar. When correcting spelling errors in an input text, whenever a member of a confusion set occurs in the text, the problem is to decide if a different member of that confusion set was intended in its place.

Banko and Brill [2] use a memory-based learning approach to select the correct member of a confusion set by collecting n-gram statistics from training data and choosing the member most frequently observed in the context of the word before and word after the target word. They show a roughly log-linear improvement in accuracy relative to the size of the training corpus. Their largest corpus has 10^9 words, three orders of magnitude smaller than the Google n-gram corpus. We report higher accuracy than their results, in line with their observed log-linear trend in improvement.

Liu and Curran [13] build a 10 billion word corpus from a web crawl and use the memory-based learner approach of Banko and Brill. They also give results using the Gigaword corpus, a 2 billion word corpus of relatively “clean” text from newswire and similar sources. They achieve the highest reported accuracies for true cross-domain experiments where the training data is not drawn from the same distribution as the test data. Our methods use much more training data than these previous memory-based methods, and a more sophisticated algorithm which chooses the most discriminative context to select the correct word, to exceed the previously stated best performance.

Golding and Roth explore a discriminative classification-based approach to this problem [10]. Given text that demonstrates correct usage of the words, classifiers are trained to discriminate the intended member of a confusion set from the context words near a member of that confusion set, their part-of-speech tags, and conjunctions of those features. Their work applies mistake-based classification algorithms to this problem, which can be relatively expensive to train and require large amounts of memory for the large feature spaces used. Later work explores methods of pruning features to avoid some of these problems [5]. Our work uses simpler methods with much more training data, and training our models consists of simply counting n-gram statistics in a corpus. Additionally, their approaches tend to perform poorly on true cross-domain experiments, where the training data differs significantly in domain from the test data. We train on a general Web corpus and evaluate on two domains, giving a more realistic evaluation of our system.

Church and Gale present an approach to correcting non-word errors using a noisy channel model to predict a probable intended word given a misspelled word [6]. Their use of contextual information from n-gram statistics showed a small improvement in accuracy. They found that simple MLE-based approaches to using the n-gram statistics actually degraded performance. We find that a simple MLE-based algorithm for estimating probabilities of intended words from their context works very well. Their work used much less training data (about 100 million words versus our 1 trillion), which explains why our simpler approach works well.

An alternative method of using web-scale data for natural language tasks is to use search engine counts, which removes the need to store a large database of aggregated data. Lapata and Keller used this approach with results from the search engine Altavista on the task of spelling correction [12]. However, Liu and Curran showed that this approach gives much lower accuracy than using accurate counts from a collection of web pages. This is because hit counts returned by search engines are typically estimates, and not well-suited for use in NLP systems.

3.3 Methods

Our method for context-sensitive spelling correction chooses the most likely word from a list of candidates, given the context of adjacent words from the sentence in which the word in question appears. This requires us to determine which sequence of words (the context, with each of the candidates substituted in turn) is most probable. To estimate these probabilities we use the Google n-gram corpus previously described. Our procedure is described generally in Section 3.3.1, and specifically for correcting real-word errors and non-word errors in Sections 3.3.2 and 3.3.3.

3.3.1 General Methodology

Our general algorithm takes as input a set of candidate words $C = \{c_1, c_2, ..., c_k\}$, a database of n-gram statistics for sizes 1 to m, and context words less than m words to the left and right of the target word being corrected, where w_{-i} is the ith word before, and w_j is the jth word after. For example, C could be $\{peace, piece\}$, m could be 2, and the context words could be $w_{-1} =$ *a* and $w_1 =$ *of*. Our algorithm recommends a word $c \in C$ using the procedure:

Consider each position in which each candidate word c could appear within an n-gram of size m. Select the position with the highest ratio between the highest count and the second highest count, and return the word c that corresponds to the highest count. If all queries had counts of zero, decrement the maximum n-gram size and repeat the procedure. In our example, we would consider the 2-grams *a peace, a piece, peace of,* and *piece of.* If no matches were found any of those 2-grams, we would then consider the 1-grams *peace* and *piece* and select the more frequent word.

Only n-grams that occurred at least 40 times were reported in the training data, so if we do not find a match for a query, we optimistically assume a count of 39.

In some cases, we do not have enough context words. We use only context from within the same sentence, so when the candidates appear near the beginning or the end of a sentence, we might not have a full set of context words. In this case, we restrict the positions in which a word can appear to those with sufficient context.

It should also be noted that the actual query issued to the database might not always directly line up with the context words. This is caused by tokenization issues, namely the fact that a word like "they're" becomes two separate tokens, "they" and "'re", to match the tokenization used in the Google n-gram database. In this case, one of the tokens from the context will be ignored, as the word with the apostrophe takes up two words in the n-gram query.

3.3.2 Real-Word Error Methodology

One application of this methodology is to correct real-word errors where a real word was used while a different word was intended. Sets of such words are called *confusion sets*, and include sets of words such as $\{they're,\ their,\ there\}$, and $\{hear,\ here\}$. The application of our method to this problem is straightforward. Given some text to correct, whenever we find a word that is a member of a (pre-defined) confusion set, we use the method described in Section 3.3.1 to choose the correct word from the confusion set given its context.

In this setting, the left and right context are the sets of words appearing before and after the word that belongs to a confusion set, and the set of candidate words are those words that belong to the same confusion set. E.g. if we were presented with "Johnny is going to their house for dinner tonight", the left context would be "Johnny is going to", the right context would be "house for dinner tonight", and the set of candidate words would be $\{their,\ they're,\ there\}$.

We tested this method on the Brown corpus, which is a collection of American English drawn from a variety of sources, and the Wall Street Journal corpus, a collection of articles from the Wall Street Journal. Both corpora are subsets of the text from the Penn Treebank corpus. Since these are widely used corpora, we were concerned that their text might occur enough on the web to bias the results, as the n-gram data came from web pages. However, we tested many queries and found only one complete copy of the Brown corpus and a few excerpts from each corpus online. We believe that the threshold of 40 occurrences used in the n-gram data is enough to prevent these pages from biasing the results. For each corpus, we looked for any occurrence of a word in a confusion set in the text. We assumed that the text was grammatical, meaning that the word used in the sentence was the correct member of the confusion set, and tested whether our method predicted the same member of the confusion set as being correct. This same methodology was used on the same corpus in the results reported by Liu and Curran [13], which we compare to in Section 3.4.1.

3.3.3 Non-Word Error Methodology

A second application of this method is towards re-ranking suggestions made by a spell checking program when a word not found in the dictionary is accidentally used instead of a dictionary word. Spell checkers are often used to correct such errors, but it's not always clear what word is the correct replacement for a mis-typed word. For instance, should someone type "funn", should this be interpreted as "fun", "funny", "funk", or something else? To handle this, spell checkers often present users with a list of choices of correctly spelled words from which to choose. This list is typically generated based on edit distance from the non-word typed by the user without take context into account. Our goal is to choose the correct word from this list presented by a spell checker using the context in which the error occurs.

To test this approach, we took a novel (Fame and Fortune, by Horatio Alger) that had not previously been indexed by Google, and as such our results would not be biased by the text showing up in search engines. An e-book of this novel was released by Project Gutenberg on May 28, 2007, well after the release date of the n-gram data set. Searching Google for a number of phrases from the book returned zero results. Our goal then was to introduce errors into the text, and see if we could use our method, in conjunction with a spell checker, to correct these errors.

We introduced three types of errors into the text - we added characters, deleted characters, and substituted characters. Each of these errors was introduced into a word with probability 0.05. In the case of added characters, we chose a number of characters by sampling from a Normal(0,.3) and taking the ceiling of its absolute value as the number of characters to insert. For each of these characters, we chose a position for insertion (defined as the position before which to insert a letter, including the "end" position). We then inserted a character that was "close" to the following character. We did this by creating a distance mapping from each character to each other character, where the distance roughly approximates the L-1 distance on a standard U.S. keyboard (roughly L-1 in the sense that we used an exception for keys that are only one diagonal space apart, which we said have a distance of 1). We then choose a "close" key by choosing a distance by sampling a Normal(0,4) distribution, and again taking the ceiling of its absolute value, and then selecting one of the keys with this distance randomly.

We also deleted characters from words. The number of deletions was calculated in the same manner as the number of additions, and the specific characters to be deleted were selected uniformly at random. Finally, we introduced substitution errors into the text, where again the number of substitutions was calculated in the same way as the number of additions was calculated, the specific positions where errors were to be introduced were selected uniformly at random, and the character to substitute was chosen according to the same "closeness" metric as described in addition of characters.

For each word into which we introduced an error, we used GNU Aspell [1], an open source spell checking library, to generate candidates for spelling corrections. If the correct word did not appear within the first ten candidates, we ignored the instance and moved on to the next. (This often occurred when too severe an error, or combination of errors, was introduced.) We then used our method to determine which of the candidates was the correct suggestion. The left and right context were the (un-perturbed) words surrounding the word in which the error was introduced, and the candidate set was the set of words suggested by Aspell. We calculated both the rank of the correct word in the list returned by Aspell, as well as the rank produced by our method.

This method of introducing errors created 3,373 misspelled words where the correct word was present among the top 10 suggestions. 943 of these words were the result of a character deletion only, 1,410 were the result of a character insertion only, 938 were the result of a character substitution only, and the rest were the result of multiple types of errors.

3.4 Results

3.4.1 Real-Word Error Correction

Our experiments for real-word error correction, where we choose the member of a confusion set that best fits a given context, aim to answer the following questions: Does using a very large corpus improve accuracy, and does considering each possible position where the target word can occur to select the most discriminative position help?

In the experiments that follow, "N-grams (limit 3)" gives results for training on the Google n-gram data with the same algorithm as Banko and Brill [2] (considering the 3-gram centered on the target word). "N-grams (fixed)" gives results for using 5-grams centered on the target word. "N-

Training Data	Test Corpus	Average Accuracy	Weighted Average Accuracy
Gigaword	Brown	90.7	94.6
Web corpus	Brown	91.8	95.4
N-grams (limit 3)	Brown	92.6	95.6
N-grams (fixed)	Brown	94.2	96.3
N-grams (sliding)	Brown	**95.2**	**96.8**
Gigaword	WSJ	93.7	96.1
Web corpus	WSJ	93.3	95.1
N-grams (limit 3)	WSJ	93.4	95.4
N-grams (fixed)	WSJ	94.3	95.9
N-grams (sliding)	WSJ	**95.8**	**96.7**

Table 2: Comparison to previous results for correcting real-word spelling errors on Brown and WSJ corpora. Highest accuracies are in boldface.

grams (sliding)" gives results for the full algorithm, where we consider all possible positions in the 5-gram window for the target word. Comparing to results from Liu and Curran [13], "Gigaword" gives results for the Banko and Brill algorithm on a 2 billion word corpus of newswire text, and "Web corpus" gives results from 10 billion words worth of web page text.

"Average accuracy" reports the accuracy averaged over the 21 confusion sets used in the experiments. "Weighted average accuracy" is the per-token accuracy, where confusion set accuracies are weighted proportional to their frequency in the test corpus.

Table 2 compares the accuracy of our method to previous results. Our full method achieves higher accuracy on both the Brown and Wall Street Journal corpora than previous work using memory-based learners. The other rows in the table give results reported by Liu and Curran on the same test data with the same confusion sets. Our accuracy using 3-grams exceeds their results for the Brown corpus, and does not quite match on the Gigaword/WSJ combination. However, the WSJ and Gigaword corpora are both from the newswire domain, which probably helps performance significantly. These results show that using the larger corpus does indeed improve accuracy.

The previous work, when tested on the WSJ data after having been trained on the "web corpus" falls behind the fixed 5-gram method. When we add in the sliding window approach, where we no longer require that the n-grams be centered on the word in question, but rather use the positioning that is most discriminative, our accuracy increases to the point where we have the highest average accuracy on the Brown corpus by a 3.4% margin, and our average accuracy on the WSJ corpus exceeds previous work by 2.1%. We conclude that using larger n-grams improves performance significantly. Considering all possible positions for the target word appears to be an additional source of improvement.

Table 3 gives percentage accuracy for spelling correction on the Brown corpus on a per-confusion set level. The Confusion Set Average is the average accuracy across the confusion sets, and the Weighted Average is the average accuracy on a per-token level. We see that some confu-

Confusion Set	Examples	N-gram types used 3-gram (fixed)	5-gram (fixed)	5-gram (sliding)
accept, except	253	96.4	97.2	97.2
affect, effect	249	97.6	97.6	98.8
among, between	1099	82.9	86.1	87.4
amount, number	643	78.2	87.6	89.9
begin, being	805	97.0	97.8	97.8
cite, sight, site	160	73.8	82.5	88.1
country, county	500	92.8	93.0	94.0
its, it's	2158	95.4	95.8	96.3
lead, led	264	87.1	90.2	88.6
fewer, less	478	94.6	94.6	94.4
maybe, may be	592	97.5	97.8	98.0
I, me	6347	98.9	99.0	99.0
passed, past	442	93.4	93.9	95.2
peace, piece	331	91.2	92.7	95.5
principal, principle	201	89.1	93.0	95.0
quiet, quite	358	96.4	96.1	98.3
raise, rise	155	94.2	94.2	95.5
than, then	3175	95.5	95.7	97.3
their, there, they're	5576	97.5	98.0	97.2
weather, whether	361	97.0	97.2	98.1
your, you're	1072	97.2	97.4	98.6
Confusion Set Average		92.6	94.2	95.2
Weighted Average		95.6	96.3	96.8

Table 3: Test accuracies by confusion set for correcting real-word errors on the Brown corpus for various n-gram types.

sion sets have lower accuracy than others— it is hard to distinguish between *among* and *between*, and relatively easier to distinguish between *affect* and *effect*. This seems sensible because *affect* and *effect* are likely to have different types of words nearby in the text, since *affect* is a verb and *effect* is usually a noun. *Among* and *between* share the same part of speech, and the local context in the text might not be informative about which is correct.

3.4.2 Non-Word Error Correction

Figure 2 shows the accuracy of our method for re-ranking suggestions from a spell checker for words where errors have been introduced by inserting, substituting, or deleting characters. Insertion errors are the easiest to recover from. Aspell ranks the correct word first 43.1% of the time, while our method using 1-grams and 5-grams ranks the correct word first 79.1% and 92.4% of the time, respectively. For deletion errors, Aspell ranks the correct word first 33.1% of the time, compared to 60.1% and 84.9% for our method with 1-grams and 5-grams. Accuracy on substitution errors is similar, with Aspell ranking the correct word first 30.9% of the time, and our method 67.4% and 86.4% of the time for 1- and 5-grams, respectively.

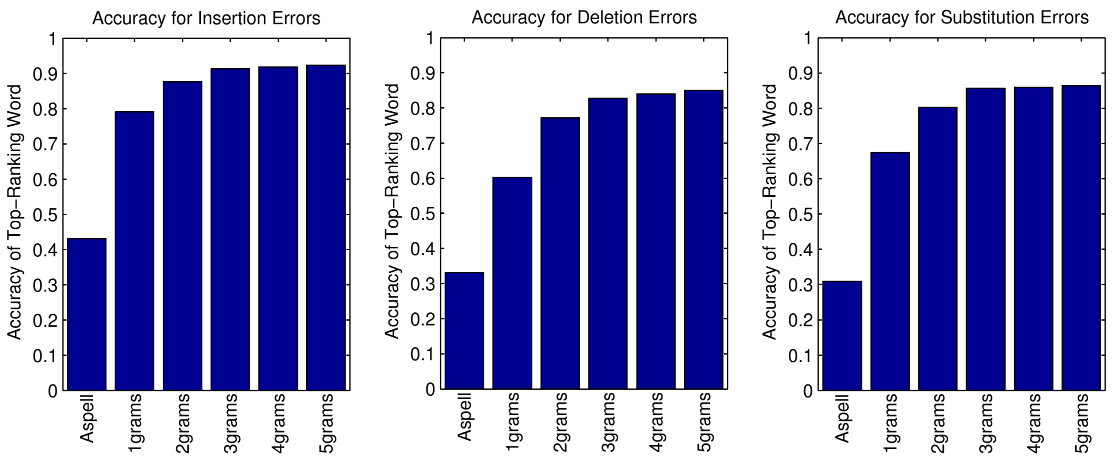


Figure 2: Results of re-ranking spell checker suggestions for non-word errors per error type.

The accuracy using word frequency alone (1-grams) is better than what Aspell produces, and the addition of context yields an improvement in accuracy of roughly 10-25%, depending on error type. However, as Figure 2 also shows, the improvement between 3-grams and 5-grams is much smaller. It may be worth using only 3-grams to save space, as they offer the best trade-off between space and accuracy.

Because spelling correction systems return a ranked list of suggestions, we also evaluate how often the correct word is ranked among the top k words. Figure 3 shows the "recall at k" for Aspell, 1-grams, and 5-grams, for k from 1 to 10 (recall that we discarded instances when the correct word did not appear in the top 10 results, hence all methods converge to 100% at k=10). Here, we see that the method using 5-grams usually has the target word in the top 2 or 3 ranked slots. The re-ranking using 1-grams is a significant improvement over Aspell, but often has candidates ranked fourth or fifth. The ranking given by Aspell is much worse than our methods, with a significant number of examples ranked in the lower half of the recommendations.

3.5 Probabilistic Models

This section gives a probabilistic justification for the sliding window method, as well as a discussion of how we might use statistical language models to combine evidence from multiple n-grams.

3.5.1 Probabilistic Justification for the Sliding Window Method

Modeling the Probability of an n-gram The sliding window method uses a separate multinomial model of the probability of observing a specific value of an n-gram for each size n. Parameters for those models are estimated from the n-gram data using maximum likelihood estimates:

$$\hat{P}(w_1, w_2, \ldots, w_n) = \frac{1}{Z_n} count(w_1, w_2, \ldots, w_n)$$

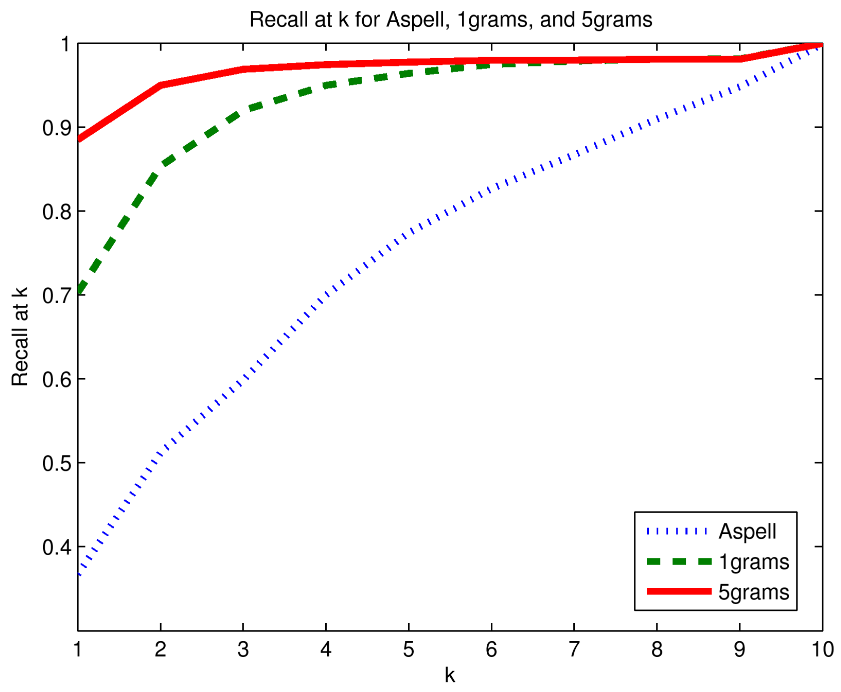


Figure 3: Recall at k for re-ranking the top 10 recommendations made by GNU Aspell for non-word errors using 1-grams and 5-grams.

where $Z_n = \sum_{w_1,w_2,\ldots,w_n} count(w_1, w_2, \ldots, w_n)$ is a normalizing constant and the *count* function gives the number of times a sequence of tokens was observed in the n-gram training text.

The Sliding Window Method The sliding window method chooses the value of a target token, denoted by t, from a candidate set of tokens, denoted by $S = \{s_1, s_2, \ldots, s_n\}$ (for example, $\{$ *peace*, *piece* $\}$). It is given K different contexts, $c_1, \ldots, c_K$ (for example, 'like a __', 'a __ of', and '__ of cake', where the blank indicates the place where the candidate token should be inserted). The method chooses the candidate token s_i where i is chosen as

$$\arg\max_i \max_k P(t = s_i | c_k, t \in S)$$

That is, we choose the candidate token and context that maximize the conditional probability of that token given the context and given that the target token is a member of the candidate set S. We use the estimate $\hat{P}$ for the distribution P:

$$\begin{aligned} \arg\max_i \max_k \hat{P}(t = s_i | c_k, t \in S) &= \arg\max_i \max_k \frac{\hat{P}(t = s_i, c_k, t \in S)}{\hat{P}(c_k, t \in S)} \\ &= \arg\max_i \max_k \frac{count(s_i, c_k)}{\sum_j count(s_j, c_k)} \end{aligned}$$

where $count(s, c)$ is the n-gram frequency count for the n-gram formed when s is inserted into the blank in the context c.

Because maximizing a probability is equivalent to maximizing the odds of that probability, we can equivalently maximize the odds of the conditional probability:

$$\begin{aligned} \arg\max_i \max_k \hat{P}(t = s_i | c_k, t \in S) &= \arg\max_i \max_k \frac{\hat{P}(t = s_i | c_k, t \in S)}{1 - \hat{P}(t = s_i | c_k, t \in S)} \\ &= \arg\max_i \max_k \frac{\hat{P}(t = s_i | c_k, t \in S)}{\sum_{j \neq i} \hat{P}(t = s_j | c_k, t \in S)} \\ &= \arg\max_i \max_k \frac{count(s_i, c_k)}{\sum_{j \neq i} count(s_j, c_k)} \end{aligned}$$

For cases where the candidate set S has two members, this is exactly what the sliding window method does. For cases where the set has more than two members, the sliding window method finds the highest ratio between the highest count and the second highest count for members of the candidate set. This is similar to, but not equal to, the method described here. However, it is rare to have a case where more than two candidates have high counts in the same context, so the two methods have similar behavior in practice.

Note that the sliding window method treats each position of the window as if it were an independent context, does not estimate the density of sequences with length greater than five (since this is the maximum n-gram size in our data set), and cannot combine evidence from n-grams of different sizes. The next subsection discusses an approach based on language modeling, which provides an estimate of the joint density of longer sequences of tokens, and can address these shortcomings of the sliding window method.

3.5.2 An Alternative to the Sliding Window: Language Models

We could alternatively model the joint probability of the sequence of observed tokens with each candidate substituted for the token in question, and then choose the candidate token that maximizes this joint probability. Models of the joint probability of a sequence of tokens are commonly called *language models*. Such an approach would combine evidence from several different windows, rather than choosing one window for its decision as the sliding window method does.

If the target token is the jth token, then we choose token s_i from the candidate set S by choosing i according to

$$\arg\max_i P(w_1, w_2, \ldots, w_{j-1}, s_i, w_{j+1}, \ldots, w_N)$$

If we assume that the tokens are generated by a fourth-order Markov process, so that the distribution of a token only depends on the four tokens before it, then we can write this as:

$$\begin{aligned}= \arg\max_i \quad & P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)P(w_4|w_3, w_2, w_1)P(w_5|w_4, w_3, w_2, w_1)\ldots \\ & \times P(s_i|w_{j-1}, w_{j-2}, w_{j-3}, w_{j-4})P(w_{j+1}|s_i, w_{j-1}, w_{j-2}, w_{j-3})P(w_{j+2}|w_{j+1}, s_i, w_{j-1}, w_{j-2}) \\ & \times P(w_{j+3}|w_{j+2}, w_{j+1}, s_i, w_{j-1})P(w_{j+4}|w_{j+3}, w_{j+2}, w_{j+1}, s_i)\ldots \\ & \times P(w_N|w_{N-1}, w_{N-2}, w_{N-3}, w_{N-4})\end{aligned}$$

In the maximization, we can ignore all terms that do not depend on i:

$$\begin{aligned}= \arg\max_i \quad & P(s_i|w_{j-1}, w_{j-2}, w_{j-3}, w_{j-4})P(w_{j+1}|s_i, w_{j-1}, w_{j-2}, w_{j-3})P(w_{j+2}|w_{j+1}, s_i, w_{j-1}, w_{j-2}) \\ & \times P(w_{j+3}|w_{j+2}, w_{j+1}, s_i, w_{j-1})P(w_{j+4}|w_{j+3}, w_{j+2}, w_{j+1}, s_i)\end{aligned}$$

Estimating Parameters of the Model Using a language model that assumes that tokens are generated by a fourth-order Markov process requires that we estimate the probability of observing a token given the four tokens preceding it. We can obtain the maximum likelihood estimates those parameters from training data as

$$\hat{P}(w_5|w_1, w_2, w_3, w_4) = \frac{count(w_1, w_2, w_3, w_4, w_5)}{count(w_1, w_2, w_3, w_4)}$$

where the $count$ function gives the number of times a sequence was observed in a corpus of training text. We can use the n-gram data for these counts, but because of the 40 occurrence threshold in the n-gram data set, these are not exactly equal to the MLE estimates for the conditional probabilities.

These counts are often zero, which results in zero or undefined probability estimates. Various techniques to smooth such estimates using smaller n-grams have been considered in the language modeling literature. Exploring these techniques is left as an item for future work. The most immediately promising method is the *Stupid Backoff* method [4], which estimates unnormalized conditional probabilities using a very simple backoff method. If the conditional probability of some token given the previous m tokens in estimated to be 0, then a backoff estimate of some constant α times the estimate using only the previous $m - 1$ tokens is used instead, recursively. This method yielded good performance on statistical machine translation tasks given a large corpus of training data.

3.6 Conclusions

We have shown that using a massive database for spelling correction with memory-based learning techniques yields high accuracy at correcting both real-word errors and non-word errors. Additionally, we extended previous approaches by considering each possible position where the spelling error could occur inside the n-grams and demonstrated that it improved accuracy.

4 Extracting Instances of a Category

4.1 Introduction

In this section, we consider the problem of creating a list of instances of some category. For example, for the category *fabrics*, we would want to create a list including instances such as ‘silk’, ‘cotton’, and ‘suede’, among others. We consider an approach that extracts instances of a category from a large corpus of text using n-gram statistics given only the name of the category and a set of generic patterns. Solutions to this problem are useful because instances of a category can be used to improve performance on tasks such as named entity recognition, where, for example, lists of instances of a category (referred to as *gazetteers*) are commonly used to identify geographical locations, names of months, people, and so on. Of most interest to the authors, the most confident extracted instances for a category can be used as input to a semi-supervised learning system which uses a small set of high-confidence examples.

This problem is well-suited to solutions using a large database of n-gram statistics because methods exist that exploit lexico-syntactic patterns of local context (e.g. “cities such as Chicago” indicates that the instance ‘Chicago’ is a member of the category *city*). Methods exploiting these patterns have computed confidence scores for extractions using frequency-based methods, so that if Chicago occurs in contexts that indicate cities often, we are more confident that it is indeed a city. Using n-gram statistics allows one to see that “cities such as Chicago” occurs, for example, 3802 times in the Google teraword Web corpus, without having to crawl thousands of Web pages or send a query to an external search engine (which typically offer a limited quota of queries per day, and have a limit of one query per second).

4.2 Related Work

Hearst pioneered the use of lexico-syntactical patterns to extract hyponym relations from text [11]. This work used patterns like "*category* such as NP" to infer that NP is a hyponym of *category*.

The work of Hearst inspired the development of the KnowItAll system [9], which applied similar techniques to the Web. Occurrences of the lexico-syntactic patterns were retrieved using a search engine, and a redundancy-based ranking algorithm, along with unsupervised assessment techniques based on co-occurrence with the category name, were used to rank extractions,

The OntoSyphon system [15] used similar pattern-based extraction methods to populate an ontology. This work considered several different redundancy-based ranking methods, and their conclusions inspired our methods.

Pantel et al. extract *is-a* relations from a large (15GB) corpus [17]. They find that pattern-based methods, similar to those that we use in this work, are more scalable and yield better results than slower methods that use more syntactically sophisticated analysis.

4.3 Methods

Given a category c and a set of patterns P, we search for occurrences of the patterns to find instances i. Our set of patterns P consists of the patterns "c such as i", "such c as i", "c {,} including i", and "c {,} especially i", where {,} indicates an optional comma, c is the plural form of the category (a single noun), and i is the instance to be extracted. The other patterns suggested in the Hearst work, "i and other c" and "i or other c", are not used by our current method, because the database indexes present on our tables do not support suffix-oriented queries.

To assess the confidence of each extracted instance, we use a score that takes into account the number of times the instance was extracted by a pattern, and the number of times the instances occurs in any context. Specifically, we score each extracted instance i from those patterns with:

$$Score_{str-norm-thresh}(i,c) = \frac{\sum_{p \in P} hits(i,c,p)}{\max(hits(i), Hit_{25})}$$

where $hits(i,c,p)$ is the frequency count for the n-gram with instance i and category c filling the slots in pattern p, $hits(i)$ is the frequency count for instance i occurring on its own, and Hit_{25} is the 25th percentile hit count for all instances i extracted for category c. This prevents extremely rare instances from having very high scores, which frequently results in typographical errors being highly ranked. This method has been shown to be an effective ranking method [15].

4.3.1 Filtering the Extracted Instances

Using the methods for extracting and ranking instances mentioned above results in some undesirable instances being extracted. Much of this is the result of prefixes of multi-token instances being extracted. For example, for the category *metals*, the instance 'stainless' might be extracted because the 4-gram "metals such as stainless" occurs many times on the Web. However, if we also extract the instances 'stainless steel', we can deduce that 'stainless' is a prefix and filter it from the extracted instances. Our method filters these instances that are prefixes of other instances.

In other cases, we might extract an instance 'tarnished' for the category *metals*, because "metals including tarnished" might occur many times. We can filter out such extraction by using a part-of-speech tagger to tag the extracted instances. Based on this reasoning, in our method, we filter out any instances that have tokens tagged as non-nouns. We also ignore any instances that use the special tokens that indicate an unknown word or a sentence boundary.

Since the data set is case-sensitive, we often extract the same instance with different capitalizations. In this case, our method merges these extractions into one instance by summing all of the counts and selecting the most frequently extracted version. For example, if both 'apples' and 'Apples' are extracted, we use the more frequently extracted version (in this case, 'apples'), and sum the number of times the two versions were extracted.

4.4 Results

To evaluate our method, we extracted instances for 27 different categories. Many of the categories were taken from Hearst's original paper, which gave examples of hyponym relations found in *Grolier's*, an electronic encyclopedia. Others were chosen with the aim of making a broad set of categories with both very specific and very general ones. Table 4 shows the top 10 ranked extractions for each of the 27 categories. The results are mixed, but generally most categories have accurate top 10 extractions.

In many cases, the results are a mix of named entities and subclasses that are hyponyms of the category. For example, *agencies* has both 'Centrelink' and 'police departments', and both 'Frosted Flakes' and 'oatmeal' are extracted as *cereals*. The desirability of these results depends on the application.

In the case of the category *people*, the extractions 'pornography producer' and 'puddle months' both come from Web spam (nonsense text that tries to boost search engine rankings). These phrases do not occur very often on the Web, so they have falsely inflated rankings. Such phrases might be combated by detecting Web spam before extracting n-grams, or by using a scoring function that takes into account how many different patterns extract an instance, and favors instances that are extracted by multiple patterns.

Our filtering methods require instances to be tagged with POS tags that start with the prefix 'NN'. This filters out many valid years and dates which have numeric tokens, because the POS tags for these types of tokens do not start with 'NN'. This helps to explain the poor results for the *years* and *dates* categories.

Table 5 gives the number of instances extracted for each category. There is a very high variance in this number across the categories.

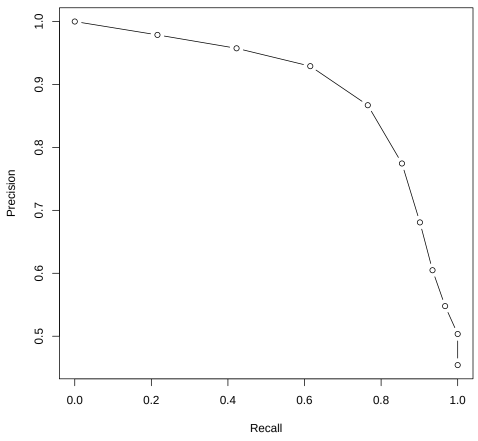
Figure 4 shows precision-recall curves for the categories *countries*, *cereals*, and *fabrics*. These curves show the precision at various levels of recall. Figure 4(a) shows that the highest ranked countries are quite accurate. It is possible to extract roughly 80% of the countries with nearly 90% precision. Figure 4(b) shows that the top ranked cereals are correct, with an incorrect extraction only occurring far down in the ranked list of extractions. Results for *fabrics* are shown in Figure 4(c). To extract all 58 fabrics, we would have a precision of about 70%. The top ranked fabrics are more accurate, but precision drops fairly quickly.

Category	Top 10 Extractions
agencies	theGeneral.com, Centrelink, UNICEF, police departments, credit bureaus, Ogilvy, AusAID, HIPPA, WFP, Equifax
amphibians	salamanders, frogs, toads
antibiotics	erythromycin, penicillins, tetracycline, ampicillin, vancomycin, streptomycin, telithromycin, clarithromycin, ciprofloxacin, clindamycin
bivalves	mussels, clams, oysters
cereals	oatmeal, sorghum, Frosted Flakes, wheat, Cheerios, oats, maize, rye, millet, bran
cities	Chicago, Beijing, San Francisco, Los Angeles, London, Netrate Discount Hotels, Atlanta, Boston, Paris, Seward
companies	Visto, computer jobs, Alamo Cheap Car, airtours, wager works, Sainsburys, automobile manufacturers, Alamo Cheap Hire, Alamo Cheap Rental, Alamo Car Hire
continents	Antarctica, Brisbane, BMW, geography, North America, Citibank, Africa, Brazil, South America, Atlantis
countries	Russia, Brazil, Japan, Iran, France, Germany, Thailand, Indonesia, Egypt, Argentina
dates	birthdays, weekends, wedding, London, baby, quarter start, reception times, dates, December, Brighton
dyes	auramine, rhodamine, thiazole, methylene blue, trypan, SYBR Green, propidium iodide, acridine, ethidium bromide, fluorescein
fabrics	silk, cotton, wool, linen, mohair, nylon, velvet, stripes, satin, Coolmax
flatworms	*no extractions*
fruits	apples, oranges, bananas, grapes, strawberries, peaches, mangoes, mango, peach, pineapple
fungi	email directories, Chlamydia species, Beauveria bassiana, Pythium, Aspergillus, yeasts, Rhizoctonia, Stachybotrys, Alternaria, Candida albicans
governments	Britain, school districts, counties, Saudi Arabia, Tribal governments, cities, municipalities, Egypt, Australia, Jordan
hydrocarbons	ethane, benzene, toluene, hexane, methylene chloride, pentane, aromatics, benzo, dichloromethane, cyclopentane
ideologies	socialism, communism, Marxism, fascism, liberalism, nationalism, Nazism, transhumanism, Bolshevism, conservatism
industries	telecommunications, agriculture, mining, chemicals, electronics, textiles, pharmaceuticals, biotechnology, tourism, finance
infections	pneumonia, tuberculosis, colds, bronchitis, meningitis, MRSA, chlamydia, influenza, thrush, athlete
institutions	community colleges, orphanages, pension funds, research centres, Historically Black, money remitters, credit unions, science museums, nursing homes, parliaments
legumes	alfalfa, lentils, soybeans, clover, peanuts, peas, chickpeas, soybean, kidney beans, vetch
liqueurs	sloe gin, Amaretto, Grand Marnier, Benedictine, lemon, Harvey, orange
locations	airports, Orlando, hotels, Brisbane, schools, restaurants, India, addresses, London, New York
minerals	pyrite, magnetite, Vitamins A, monazite, magnesium, titanium dioxide, calcite, Orotates, biotite, kaolinite
months	October incidents, leap years, renewals, monsoon, collaborators, HBO, Showtime, carrots, VAT, proposals
organisms	fungi, humans, algae, Coliform bacteria, E. coli, mammals, Salmonella, yeast, yeasts, viruses
organizations	Cisco Partners, Mustang Survival, National Fire, Amnesty International, transit systems, Big Brothers, Ducks Unlimited, auxiliaries, labor unions, MoveOn.org
people	MA Urbina, pornography producer, Pinckwerts, puddle months, Farhad Wadia, Craig Carman, Thomas Highs, Stuart Eisenstadt, Bill Merriam, judge magistrates
phenomena	earthquakes, Metroid, lightning, El Niño, hurricanes, clairvoyance, El Nino, telepathy, floods, fire
planets	Hoth, Mars, Jupiter, Geonosis, Yavin IV, Venus, Mercury, Saturn, Pluto, sci
protozoa	giardia lamblia, Cryptosporidium
rivers	tributaries, Helen, water
rocks	limestone, granite, sandstone, basalt, granites, shale, sandstones, gneiss, gabbro, shales

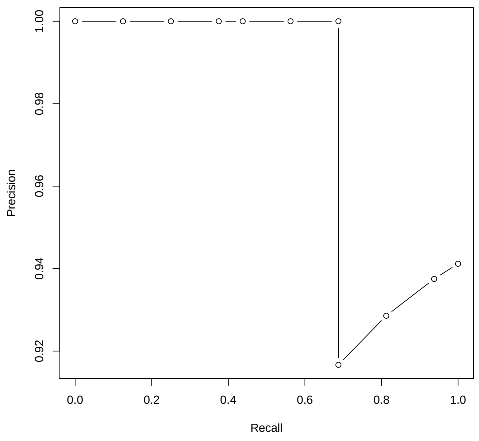
Table 4: Top 10 extractions for a variety of categories, sorted in descending order of confidence score.

Category	Extractions	Category	Extractions
agencies	474	liqueurs	7
amphibians	3	locations	377
antibiotics	46	minerals	143
bivalves	3	months	35
cereals	17	organisms	168
cities	525	organizations	842
companies	1945	people	869
continents	20	phenomena	222
countries	469	planets	16
dates	26	protozoa	2
dyes	12	rivers	3
fabrics	84	rocks	23
flatworms	0	seabirds	7
fruits	79	species	706
fungi	27	students	250
governments	74	substances	339
hydrocarbons	39	teams	143
ideologies	14	times	61
industries	449	towns	153
infections	178	waterfowl	7
institutions	393	years	272
legumes	20		

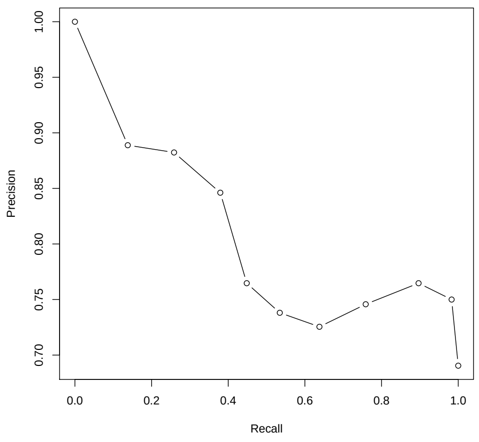
Table 5: Number of extractions for each category.



(a) countries (213 correct extractions)



(b) cereals (16 correct extractions)



(c) fabrics (58 correct extractions)

Figure 4: Precision-Recall curves for three categories.

4.5 Future Work

In future work, we might want to further filter the extracted instances for each category, by using additional patterns to check them. The KnowItAll system uses a similar approach for its assessment [9].

We could use a pattern learning approach to bootstrap off of the initially extracted instances, and discover additional patterns that indicate members of the category.

A fundamental limit of extracting category instances from n-gram data is the limit of the size of the token sequences. It can be difficult to tell if an instance is actually a prefix of some longer prefix. A possible method to check instances for completeness is searching for the instances in the context of the extraction patterns from the n-gram data, retrieving the Web pages that match, and checking to see if the noun phrases are complete, given the full context from the Web page.

5 Conclusions and Future Work

We have shown that n-gram statistics from a very large corpus can provide high accuracy with relatively simple algorithms for spelling correction. Additionally, pattern-based methods can extract high-quality instances of categories in far less time than would be required if each document were to be processed individually.

Here we summarize some conclusions and questions for future work.

5.1 Conclusions

- *The n-gram data is a good source of information.* We have shown the usefulness of the data for two different applications, and it has already been shown to be useful for machine translation [4].

- *Simple approaches to spelling correction work well, given lots of training data.* A simple memory-based learning method was shown to yield state-of-the-art results on spelling disambiguation tasks given a massive training corpus.

- *Working with large amounts of data takes time.* Building the database from the raw n-gram data took roughly a week on a machine with adequate disk space and 4 GB of RAM. Once the database was created and the tables were indexed, building new applications that use the database was easy, but there was a large startup investment.

- *Web data is noisy.* A prime example is the phrase ‘species such as anchovyto’ which is a typographical error, and appears on nearly 500 pages on the same web site. This caused it to have an misleadingly high frequency count in our data set. Special techniques are needed to deal with these inconsistencies, such as duplicate detection. Also, numerous other category instance extractions came from Web spam sites, which had misleading nonsense text that was generated to seem like English text.

5.2 Questions for Future Work

- *What other applications are there for n-gram-based methods, and do we also see good performance on those applications?* Many other applications are well-suited to algorithms that use n-gram data, such as machine translation (which drove the creation of the data set), topic detection and tracking, Web spam detection, and named entity recognition, just to name a few.

- *Are there other types of summary statistics that would be useful?* Cooccurrence statistics have been shown to be very useful in assessing the semantic similarity of two words or phrases. These can in fact be calculated from n-gram models, but could be very useful to have precalculated for fast semantic similarity calculations. For example, to assess the semantic similarity of the pair of words 'apple' and 'eat', we could compute the ratio between the estimated probability of the two words occurring together within a 5-token window and the probability that we would estimate for this event if we assumed that occurrences of 'apple' and 'eat' were independent events. To compute this, we must sum the frequency counts for all n-grams that contain both 'apple' and 'eat', which could require the database to access many millions of records.

- *How can we avoid the issues that token-based n-gram models have with noun phrase boundaries?* In our experiments with category extraction, we found that we often extracted incomplete instances (e.g. 'newspapers such as the New York Times' would be truncated since it is 7 tokens long). One solution to this problem is to preprocess the corpus before counting the n-grams, and combining multi-token named entities into single tokens [8]. The detection of multi-token named entities can actually make good use of the n-gram statistics by looking at the tendency of multi-word expressions to occur together [7].

- *What impact did the thresholding have?* N-grams that occurred fewer than 40 times were not reported in the data. This could have positive effects, as it filters out spurious token sequences, but it also could have negative effects, leaving out useful information. Finding the right trade-off between database size and such thresholds seems like an important consideration. We could study the effect of higher thresholds than 40 by setting our own higher threshold, ignoring records below this threshold, and observing the effect on spelling correction accuracy.

- *Could we supplement the n-gram data with other annotations?* Supplementing the n-gram data with part-of-speech tags, named entity designations, and other annotations seems useful. For example, part-of-speech tags have been shown to be useful in the spelling correction task [10].

- *Can we create standard tools to lower the barrier to entry in large-scale research?* The Google n-gram data set was a good start to answering this question. The data set needed a large cluster of machines to compute, but can be stored on a single PC with a few hundred GB of disk. However, the general barrier to entry for large-scale computing is still high. Researchers such as Pantel have started to discuss standard processing frameworks for scaling up to huge corpora [16].

References

[1] Kevin Atkinson. *GNU Aspell*, 1998. Software available at `http://aspell.net/`.

[2] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Meeting of the Association for Computational Linguistics*, pages 26–33, 2001.

[3] Thorsten Brants and Alex Franz. Web 1t 5-gram version 1, 2006.

[4] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, 2007.

[5] Andrew J. Carlson, Jeffrey Rosen, and Dan Roth. Scaling up context-sensitive text correction. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*, pages 45–50. AAAI Press, 2001.

[6] Kenneth W. Church and William A. Gale. Probability scoring for spelling correction. *Statistics and Computing*, 1991.

[7] Doug Downey, Matthew Broadhead, and Oren Etzioni. Locating Complex Named Entities in Web Text. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, Hyderabad, India, January June–December 2007.

[8] Doug Downey, Stefan Schoenmackers, and Oren Etzioni. Sparse information extraction: Unsupervised language models to the rescue. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 696–703, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[9] O. Etzioni, M. J. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[10] Andrew R. Golding and Dan Roth. Applying winnow to context-sensitive spelling correction. In *International Conference on Machine Learning*, pages 182–190, 1996.

[11] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

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

[13] Vinci Liu and James R. Curran. Web text corpus for natural language processing. In *EACL*. The Association for Computer Linguistics, 2006.

[14] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[15] Luke McDowell and Michael J. Cafarella. Ontology-driven information extraction with ontosyphon. In *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 428–444. Springer, 2006.

[16] Patrick Pantel. Data catalysis: Facilitating large-scale natural language data processing. In *Proceedings of the International Symposium on Universal Communication (ISUC-07)*, pages 201–204, 2007.

[17] Patrick Pantel, Deepak Ravichandran, and Eduard Hovy. Towards terascale knowledge acquisition. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 771, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon