A Large Scale Ranker-Based System for Search Query Spelling Correction

Jianfeng Gao
Microsoft Research
Redmond, WA, USA
jfgao@microsoft.com

Xiaolong Li
Microsoft Corporation
Redmond, WA, USA
xiaolong.li@microsoft.com

Daniel Micol
Microsoft Corporation
Munich, Germany
danielmi@microsoft.com

Chris Quirk
Microsoft Research
Redmond, WA, USA
chrisq@microsoft.com

Xu Sun
Dept. of Mathematical Informatics
University of Tokyo, Tokyo, Japan
xusun@mist.i.u-tokyo.ac.jp

Abstract

This paper makes three significant extensions to a noisy channel speller designed for standard written text to target the challenging domain of search queries. First, the noisy channel model is subsumed by a more general ranker, which allows a variety of features to be easily incorporated. Second, a distributed infrastructure is proposed for training and applying Web scale *n*-gram language models. Third, a new phrase-based error model is presented. This model places a probability distribution over transformations between multi-word phrases, and is estimated using large amounts of query-correction pairs derived from search logs. Experiments show that each of these extensions leads to significant improvements over the state-of-the-art baseline methods.

1 Introduction

Search queries present a particular challenge for traditional spelling correction methods. New search queries emerge constantly. As a result, many queries contain valid search terms, such as proper nouns and names, which are not well established in the language. Therefore, recent research has focused on the use of Web corpora and search logs, rather than human-compiled lexicons, to infer knowledge about spellings and word usages in search queries (e.g., Whitelaw et al., 2009; Cucerzan and Brill, 2004).

The spelling correction problem is typically formulated under the framework of the noisy channel model. Given an input query $Q = q_1 ... q_I$, we want to find the best spelling correction $C = c_1 ... c_J$ among all candidate corrections:

$$C^* = \underset{C}{\operatorname{argmax}} P(C|Q) \quad (1)$$

Applying Bayes' Rule, we have

$$C^* = \underset{C}{\operatorname{argmax}} P(Q|C)P(C) \quad (2)$$

where the error model $P(Q|C)$ models the transformation probability from C to Q, and the language model (LM) $P(C)$ models the likelihood that C is a correctly spelled query.

This paper extends a noisy channel speller designed for regular text to search queries in three ways: using a ranker (Section 3), using Web scale LMs (Section 4), and using phrase-based error models (Section 5).

First of all, we propose a ranker-based speller that covers the noisy channel model as a special case. Given an input query, the system first generates a short list of candidate corrections using the noisy channel model. Then a feature vector is computed for each query and candidate correction pair. Finally, a ranker maps the feature vector to a real-valued score, indicating the likelihood that this candidate is a desirable correction. We will demonstrate that ranking provides a flexible modeling framework for incorporating a wide variety of features that would be difficult to model under the noisy channel framework.

Second, we explore the use of Web scale LMs for query spelling correction. While traditional LM research mainly focuses on how to make the model “smarter” (e.g., how to better estimate the probability of an unseen word (Chen and Goodman, 1999); and how to model the grammatical structure of language (e.g., Charniak, 2001)), recent studies show that significant improvements can be achieved using “stupid” n-gram models trained on very large corpora (e.g., Brants et al., 2007). We adopt the latter strategy in this study. We present a distributed infrastructure to efficiently train and apply Web scale *n*-gram LMs. In addition, we observe that search queries are composed in a language style different from that of regular text. We thus train different LMs using different text streams associated with Web corpora and search logs.

Third, we propose a phrase-based error model that captures the probability of transforming one multi-term phrase into another multi-term phrase. Compared to traditional error models that account for transformation probabilities between single characters or substrings (e.g., Kernighan et al., 1990; Brill and Moore, 2000), the phrase-based error model is more effective in that it captures inter-term dependencies crucial for correcting real-word errors, prevalent in search queries. We also present a novel method of extracting large amounts of query-correction pairs from search logs. These pairs, implicitly judged by millions of users, are used for training the error models.

Experiments show that each of the extensions leads to significant improvements over its baseline methods that were state-of-the-art until this work, and that the combined method yields a system which outperforms the noisy channel speller by a large margin: a 6.3% increase in accuracy on a human-labeled query set.

2 Related Work

Prior research on spelling correction for regular text can be roughly grouped into two categories: correcting non-word errors and real-word errors. The former focuses on the development of error models based on different edit distance functions (e.g., Kucich, 1992; Kernighan et al., 1990; Brill and Moore, 2000; Toutanova and Moore, 2002). Brill and Moore's substring-based error model, considered to be state-of-the-art among these models, acts as the baseline against which we compare our models. On the other hand, real-word spelling correction tries to detect incorrect usages of a valid word based on its context, such as "peace" and "piece" in the context "a _ of cake". N-gram LMs and naïve Bayes classifiers are commonly used models (e.g., Golding and Roth, 1996; Mangu and Brill, 1997; Church et al., 2007).

While almost all of the spellers mentioned above are based on a pre-defined dictionary (generally either a lexicon against which the edit distance is computed, or a set of real-word confusion pairs), recent research on query spelling correction focuses on exploiting noisy Web corpora and query logs to infer knowledge about misspellings and word usage in search queries (e.g., Cucerzan and Brill 2004; Ahmad and Kondrak, 2005; Li et al., 2006; Whitelaw et al., 2009). Like those spellers designed for regular text, most of these query spelling systems are also based on the noisy channel framework.

3 A Ranker-Based Speller

The noisy channel model of Equation (2) does not have the flexibility to incorporate a wide variety of features useful for spelling correction, such as whether the candidate correction appears as a title in a Wikipedia document. We thus generalize the speller to a ranker-based system. Let **f** be a feature vector extracted from a query and candidate correction pair (Q, C). The ranker maps **f** to a real value y that indicates the likelihood that C is a desired correction. For example, a linear ranker maps **f** to y with a learned weight vector **w** such as $y = \mathbf{w} \cdot \mathbf{f}$, where **w** is optimized for accuracy on human-labeled (Q, C) pairs. Since the logarithms of the LM and error model probabilities can be included as features, the ranker covers the noisy channel model as a special case.

For the sake of efficiency and flexibility, our speller system operates in two distinct stages: candidate generation and re-ranking.

In candidate generation, an input query is first tokenized into a sequence of terms. For each term q, we consult a lexicon to identify a list of spelling suggestions c whose edit distance from q is lower than some threshold. Our lexicon contains around 430,000 high frequency query unigram and bigrams collected from 1 year of query logs. These generated suggestions are stored in a lattice.

We then use a decoder to identify the 20-best candidates from the lattice according to Equation (2), where the LM is a backoff bigram model trained on 1 year of query logs, and the error model is approximated by the weighted Levenshtein edit distance function as

$$-\log P(Q|C) \propto \text{EditDist}(Q, C) \tag{3}$$

The decoder uses a standard two-pass algorithm. The first pass uses the Viterbi algorithm to find the best C according to the model of Equations (2) and (3). The second pass uses the A-star algorithm to find the 20-best corrections, using the Viterbi scores computed at each state in the first pass as heuristics.

The core component in the second stage is a ranker, which re-ranks the 20-best candidate corrections using a set of features extracted from (Q, C). If the top C after re-ranking is different from Q, C is proposed as the correction. We use 96 features in this study. In addition to the two features derived from the noisy channel model, the rest of the features can be grouped into the following 5 categories.

1. **Surface-form similarity features**, which check whether C and Q differ in certain patterns,

e.g., whether C is transformed from Q by adding an apostrophe, or by adding a stop word at the beginning or end of Q.

2. **Phonetic-form similarity features**, which check whether the edit distance between the metaphones (Philips, 1990) of a query term and its correction candidate is below some thresholds.

3. **Entity features**, which check whether the original query is likely to be a proper noun based on an in-house named entity recognizer.

4. **Dictionary features**, which check whether a query term or a candidate correction are in one or more human-compiled dictionaries, such as the extracted Wiki, MSDN, and ODP dictionaries.

5. **Frequency features**, which check whether the frequency of a query term or a candidate correction is above certain thresholds in different datasets, such as query logs and Web documents.

4 Web Scale Language Models

An n-gram LM assigns a probability to a word string $w_1^L = (w_1, \ldots, w_L)$ according to

$$P(w_1^L) = \prod_{i=1}^{L} P\left(w_i \middle| w_1^{i-1}\right) \approx \prod_{i=1}^{L} P\left(w_i \middle| w_{i-n+1}^{i-1}\right) \tag{4}$$

where the approximation is based on a Markov assumption that each word depends only upon the immediately preceding n-1 words. In a speller, the log of n-gram LM probabilities of an original query and its candidate corrections are used as features in the ranker.

While recent research reports the benefits of large LMs trained on Web corpora on a variety of applications (e.g., Zhang et al., 2006; Brants et al., 2007), it is also clear that search queries are composed in a language style different from that of the body or title of a Web document. Thus, in this study we developed a set of large LMs from different text streams of Web documents and query logs. Below, we first describe the n-gram LM collection used in this study, and then present a distributed n-gram LM platform based on which these LMs are built and served for the speller.

4.1 Web N-gram LM Collection

Table 1 summarizes the data sets and Web scale n-gram LMs used in this study. The collection is built from high quality English Web documents containing trillions of tokens, served by a popular commercial search engine. The collection consists of several data sets built from different Web sources, including the different text fields from the Web documents (i.e., body, title, and anchor texts) and search query logs. The raw texts extracted from these different sources were preprocessed in the following manner: texts are tokenized based on white-space and upper case letters are converted to lower case. Numbers are retained, and no stemming/inflection is performed. The n-gram LMs are word-based backoff models, where the n-gram probabilities are estimated using Maximum Likelihood Estimation with smoothing. Specifically, for a trigram model, the smoothed probability is computed as

Dataset	**Body**	**Anchor**	**Title**	**Query**
Total tokens	1.3T	11.0B	257.2B	28.1B
Unigrams	1.2B	60.3M	150M	251.5M
Bigrams	11.7B	464.1M	1.1B	1.3B
Trigrams	60.0B	1.4B	3.1B	3.1B
4-grams	148.5B	2.3B	5.1B	4.6B
Size on disk[#]	12.8TB	183GB	395GB	393GB

[#] N-gram entries as well as other model parameters are stored.

Table 1: Statistics of the Web n-gram LMs collection (count cutoff = 0 for all models). These models will be accessible at Microsoft (2010).

$$P(w_i | w_{i-2} w_{i-1}) = \tag{5}$$

$$\begin{cases} \dfrac{C(w_{i-2}w_{i-1}w_i) - D\left(C(w_{i-2}w_{i-1}w_i)\right)}{C(w_{i-2}w_{i-1})} & \text{if } C(w_{i-2}w_{i-1}w_i) \\ \alpha(w_{i-2}w_{i-1}) P(w_i | w_{i-1}) & \text{otherwise} \end{cases}$$

where $C(\cdot)$ is the raw count of the n-gram in the training corpus and α is a normalization factor. $D(C)$ is a discount function for smoothing. We use modified absolute discounting as the discount function (Gao et al., 2001), whose parameters can be efficiently estimated and performance converges to that of more elaborate state-of-the-art techniques like Kneser-Ney smoothing in large scale data (Nguyen et al., 2007).

4.2 Distributed N-gram LM Platform

The platform is developed on a distributed computing system designed for storing and analyzing massive data sets, running on large clusters consisting of hundreds of commodity servers connected via high-bandwidth network.

We use the SCOPE (Structured Computations Optimized for Parallel Execution) programming model (Chaiken et al., 2008) to train the Web scale n-gram LMs shown in Table 1. The SCOPE scripting language resembles SQL which many programmers are familiar with. It also supports C# expressions so that users can easily plug-in customized C# classes. SCOPE supports writing a program using a series of simple data transformations so that users can simply write a script to process data in a *serial* manner without wondering how to achieve parallelism while the SCOPE

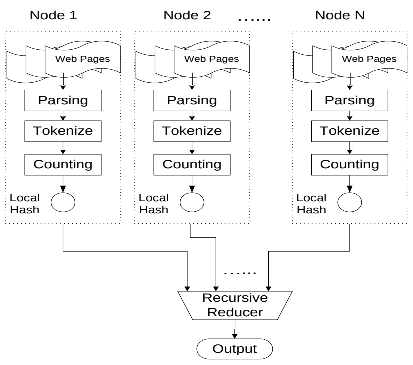

Figure 1. Distributed 5-gram counting.

compiler and optimizer are responsible for translating the script into an efficient, parallel execution plan. We illustrate the usage of SCOPE for building LMs using the following example of counting 5-grams from the body text of English Web pages. The flowchart is shown in Figure 1.

The program is written in SCOPE as a step-by-step of computation, where a command takes the output of the previous command as its input.

```
ParsedDoc=SELECT docId, TokenizedDoc
   FROM @"/shares/…/EN_Body.txt"
   USING DefaultTextExtractor;

NGram=PROCESS ParsedDoc
   PRODUCE NGram, NGcount
   USING NGramCountProcessor(-stream
   TokenizedDoc -order 5 -bufferSize
   20000000);

NGramCount=REDUCE NGram
   ON NGram
   PRODUCE NGram, NGcount
   USING NGramCountReducer;

OUTPUT TO @"Body-5-gram-count.txt";
```

The first SCOPE command is a SELECT statement that extracts parsed Wed body text. The second command uses a build-in Processor (NGramCountProcessor) to map the parsed documents into separate *n*-grams together with their counts. It generates a local hash at each node (i.e., a core in a multi-core server) to store the (*n*-gram, count) pairs. The third command (REDUCE) aggregates counts from different nodes according to the key (*n*-gram string). The final command (OUTPUT) writes out the resulting to a data file.

The smoothing method can be implemented similarly by implementing the customized smoothing Processor/Reducer. They can be imported from the existing C# codes (e.g., developed for building LMs in a single machine) with minor changes.

It is straightforward to apply the built LMs for the ranker in the speller. The *n*-gram LM platform provides a DLL for *n*-gram batch lookup. In the server, an *n*-gram LM is stored in the form of multiple lists of key-value pairs, where the key is the hash of an *n*-gram string and the value is either the *n*-gram probability or the backoff parameter.

In our test data, the average query length is 2.7 words. We consider 20 candidate corrections in ranking, so the average number of *n*-grams requested per query is 54. Assume that computing a 5-gram probability requests 9 key lookups (5 *n*-gram probabilities and 4 backoff parameters), it requests around 500 key lookups per query. Our results show that the platform can serve about 800 key lookups per millisecond. So the process of calculating a 5-gram LM feature for a query takes less than 1 millisecond.

C:	"disney theme park"	*correct query*
S:	["disney", "theme park"]	*segmentation*
T:	["disnee", "theme part"]	*translation*
M:	(1 → 2, 2→ 1)	*permutation*
Q:	"theme part disnee"	*misspelled query*

Figure 2: Example demonstrating the generative procedure behind the phrase-based error model.

5 Phrase-Based Error Models

The goal of an error model is to transform a correctly spelled query C into a misspelled query Q. Rather than replacing single words in isolation, the phrase-based error model replaces sequences of words with sequences of words, thus incorporating contextual information. The training procedure closely follows Sun et al. (2010). For instance, we might learn that "*theme part*" can be replaced by "*theme park*" with relatively high probability, even though "*part*" is not a misspelled word. We use this generative story: first the correctly spelled query C is broken into K non-empty word sequences $\mathbf{c}_1, \ldots, \mathbf{c}_k$, then each is replaced with a new non-empty word sequence $\mathbf{q}_1, \ldots, \mathbf{q}_k$, finally these phrases are permuted and concatenated to form the misspelled Q. Here, $\mathbf{c}$ and $\mathbf{q}$ denote consecutive sequences of words.

To formalize this generative process, let S denote the segmentation of C into K phrases $\mathbf{c}_{1\ldots}\mathbf{c}_K$, and let T denote the K replacement phrases $\mathbf{q}_1\ldots\mathbf{q}_K$ – we refer to these $(\mathbf{c}_i, \mathbf{q}_i)$ pairs as *bi-phrases*. Finally, let M denote a permutation of K elements representing the final reordering step. Figure 2 demonstrates the generative procedure.

Next let us place a probability distribution over rewrite pairs. Let $B(C, Q)$ denote the set of S, T, M triples that transform C into Q. If we assume a uniform probability over segmentations, then the phrase-based probability can be defined as:

$$P(Q|C) \propto \sum_{\substack{(S,T,M)\in \\ B(C,Q)}} P(T|C,S) \cdot P(M|C,S,T) \quad (6)$$

As is common practice in SMT, we use the maximum approximation to the sum:

$$P(Q|C) \approx \max_{\substack{(S,T,M)\in \\ B(C,Q)}} P(T|C,S) \cdot P(M|C,S,T) \quad (7)$$

5.1 Forced Alignments

Although we have defined a generative model for transforming queries, our goal is not to propose new queries, but rather to provide scores over existing Q and C pairs that will act as features for the ranker. Furthermore, the word-level alignments between Q and C can most often be identified with little ambiguity. Thus we restrict our attention to those phrase transformations consistent with a good word-level alignment.

Let J be the length of Q, L be the length of C, and $A = a_1, \ldots, a_J$ be a hidden variable representing the word alignment between them. Each a_i takes on a value ranging from 1 to L indicating its corresponding word position in C, or 0 if the ith word in Q is unaligned. The cost of assigning k to a_i is equal to the Levenshtein edit distance (Levenshtein, 1966) between the i^{th} word in Q and the k^{th} word in C, and the cost of assigning 0 to a_i is equal to the length of the i^{th} word in Q. We can determine the least cost alignment A^* between Q and C efficiently using the A-star algorithm.

When scoring a given candidate pair, we further restrict our attention to those S, T, M triples that are consistent with the word alignment, which we denote as $B(C, Q, A^*)$. Here, consistency requires that if two words are aligned in A^*, then they must appear in the same bi-phrase $(\mathbf{c}_i, \mathbf{q}_i)$. Once the word alignment is fixed, the final permutation is uniquely determined, so we can safely discard that factor. Thus we have:

$$P(Q|C) \approx \max_{\substack{(S,T,M)\in \\ \mathcal{B}(C,Q,A^*)}} P(T|C,S) \quad (8)$$

For the sole remaining factor $P(T|C, S)$, we make the assumption that a segmented query $T = \mathbf{q}_1 \ldots \mathbf{q}_K$ is generated from left to right by transforming each phrase $\mathbf{c}_1 \ldots \mathbf{c}_K$ independently:

$$P(T|C,S) = \prod_{k=1}^{K} P(\mathbf{q}_k|\mathbf{c}_k), \quad (9)$$

where $P(\mathbf{q}_k|\mathbf{c}_k)$ is a phrase transformation probability, the estimation of which will be described in Section 5.2.

To find the maximum probability assignment efficiently, we use a dynamic programming approach, similar to the monotone decoding algorithm described in Och (2002).

Google:

http://www.google.com/search?
hl=en&source=hp&
q=***harrypotter+sheme+part***&aq=f&oq=&aqi=

http://www.google.com/search?
hl=en&ei=rnNAS8-oKsWe_AaB2eH1CA&
sa=X&oi=spell&resnum=0&ct=
result&cd=1&ved=0CA4QBSgA&
q=***harry+potter+theme+park***&spell=1

Yahoo:

http://search.yahoo.com/search;
_ylt=A0geu6ywckBL_XIBSDtXNyoA?
p=***harrypotter+sheme+part***&
fr2=sb-top&fr=yfp-t-701&sao=1

http://search.yahoo.com/search?
ei=UTF-8&fr=yfp-t-701&
p=***harry+potter+theme+park***
&SpellState=n-2672070758_q-tsI55N6srhZa.
qORA0MuawAAAA%40%40&fr2=sp-top

Bing:

http://www.bing.com/search?
q=***harrypotter+sheme+part***&form=QBRE&qs=n

http://www.bing.com/search?
q=***harry+potter+theme+park***&FORM=SSRE

Figure 3. A sample of query reformulation sessions from three popular search engines. These sessions show that a user first issues the query "harrypotter sheme part", and then clicks on the resulting spell suggestion "harry potter theme park".

5.2 Training the Error Model

Given a set of (Q, C) pairs as training data, we follow a method commonly used in SMT (Koehn et al., 2003; Och and Ney, 2004) to extract bi-phrases and estimate their replacement probabilities. A detailed description is discussed in Sun et al. (2010).

We now describe how (Q, C) pairs are generated automatically from massive *query reformulation sessions* of a commercial Web browser.

A query reformulation session contains a list of URLs that record user behaviors that relate to the query reformulation functions, provided by a Web search engine. For example, most commercial search engines offer the "did you mean" function, suggesting a possible alternate interpretation or spelling of a user-issued query. Figure 3 shows a sample of the query reformulation sessions that record the "did you mean" sessions from three of the most popular search engines. These sessions encode the same user behavior: A user first queries for "harrypotter sheme part", and then clicks on the resulting spelling suggestion "harry potter theme park". In our experiments, we "reverse-engineer" the parameters from the URLs of these sessions, and deduce how each search engine encodes both a query and the fact that a user arrived at a URL by clicking on the spelling suggestion of the query – an important indication that the spelling suggestion is desired. In this study, from 1 year of query reformulation sessions, we extracted about 120 million (Q, C) pairs. We found the data set very clean because all these spelling corrections are actually clicked, and thus judged implicitly, by many users.

In addition to the "did you mean" functionality, recently some search engines have introduced two new spelling suggestion functions. One is the "auto-correction" function, where the search engine is confident enough to automatically apply the spelling correction to the query and execute it to produce search results for the user. The other is the "split pane" result page, where one half portion of the search results are produced using the original query, while the other half, usually visually separate portion of results are produced using the auto-corrected query.

In neither of these functions does the user ever receive an opportunity to approve or disapprove of the correction. Since our extraction approach focuses on user-approved spelling suggestions, we ignore the query reformulation sessions recording either of the two functions. Although by doing so we could miss some basic, obvious spelling corrections, our experiments show that the negative impact on error model training is negligible. One possible reason is that our baseline system, which does not use any error model learned from the session data, is already able to correct these basic, obvious spelling mistakes. Thus, including these data for training is unlikely to bring any further improvement.

We found that the error models trained using the data directly extracted from the query reformulation sessions suffer from the problem of underestimating the self-transformation probability of a query $P(Q_2{=}Q_1|Q_1)$, because we only included in the training data the pairs where the query is different from the correction. To deal with this problem, we augmented the training data by including correctly spelled queries, i.e., the pairs (Q_1, Q_2) where $Q_1 = Q_2$. First, we extracted a set of queries from the sessions where no spell suggestion is presented or clicked on. Second, we removed from the set those queries that were recognized as being auto-corrected by a search engine. We do so by running a sanity check of the queries against our baseline noisy channel speller, which will be described in Section 6. If the system thought an input query was misspelled, we assumed it was an obvious misspelling, and removed it. The remaining queries were assumed to be correctly spelled and were added to the training data.

6 Experiments

We perform the evaluation using a manually annotated data set containing 24,172 queries sampled from one year's worth of query logs from a commercial search engine. The spelling of each query is manually judged and corrected by four independent annotators. The average length of queries in the data sets is 2.7 words. We divided the data set into non-overlapped training and test data sets. The training data contain 8,515 (Q, C) pairs, among which 1,743 queries are misspelled (i.e. $Q \neq C$). The test data contain 15,657 (Q, C) pairs, among which 2,960 queries are misspelled.

The speller systems we developed in this study are evaluated using the following three metrics.

- **Accuracy:** The number of correct outputs generated by the system divided by the total number of queries in the test set.
- **Precision:** The number of correct spelling corrections for misspelled queries generated by the system divided by the total number of corrections generated by the system.
- **Recall:** The number of correct spelling corrections for misspelled queries generated by the system divided by the total number of misspelled queries in the test set.

We also perform a significance test, a t-test with a significance level of 0.05. A significant difference is read as significant at the 95% level.

In our experiments, all the speller systems are ranker-based. Unless otherwise stated, the ranker is a two-layer neural net with 5 hidden nodes. The free parameters of the neural net are trained to optimize accuracy on the training data using the back propagation algorithm (Burges et al., 2005), running for 500 iterations with a very small learning rate (0.1) to avoid overfitting. We did not adjust the neural net structure (e.g., the number of hidden nodes) or any training parameters for different speller systems. Neither did we try to seek the best tradeoff between precision and recall.

6.1 System Results

Table 1 summarizes the main results of different spelling systems. Row 1 is the baseline speller

#	System	Accuracy	Precision	Recall
1	Noisy channel	85.3	72.1	35.9
2	Linear ranker	88.0	74.0	42.8
3	Nonlinear ranker	89.0	74.1	49.6
4	3 + PBEM	90.7	78.7	58.2
5	3 + WLMs	90.4	75.1	58.7
6	3 + PBEM + WLMs	91.6	79.1	63.9

Table 1. Summary of spelling correction results.

where the noisy channel model of Equations (2) and (3) is used. The error model is based on the weighted edit distance function and the LM is a backoff bigram model trained on 1 year of query logs, with count cutoff 30. Row 2 is the speller using a linear ranker to incorporate all ranking features described in Section 3. The weights of the linear ranker are optimized using the Averaged Perceptron algorithm (Freund and Schapire, 1999). Row 3 is the speller where a nonlinear ranker (i.e., 2-layer neural net) is trained atop the features. Rows 4, 5 and 6 are systems that incorporate the additional features derived from the phrase-based error model (PBEM) described in Section 5 and the four Web scale LMs (WLMs) listed in Table 1.

The results show that (1) the ranker is a very flexible modeling framework where a variety of fine-grained features can be easily incorporated, and a ranker-based speller outperforms significantly ($p < 0.01$) the traditional system based on the noisy channel model (Row 2 vs. Row 1); (2) the speller accuracy can be further improved by using more sophisticated rankers and learning algorithms (Row 3 vs. Row 2); (3) both WLMs and PBEM bring significant improvements (Rows 4 and 5 vs. Row 3); and (4) interestingly, the gains from WLMs and PBEM are additive and the combined leads to a significantly better speller (Row 6 vs. Rows 4 and 5) than that of using either of them individually.

In what follows, we investigate in detail how the WLMs and PBEM trained on massive Web content and search logs improve the accuracy of the speller system. We will compare our models with the state-of-the-art models proposed previously. From now on, the system listed in Row 3 of Table 1 will be used as baseline.

6.2 Language Models

The quality of n-gram LMs depends on the order of the model, the size of the training data, and more importantly how well the training data match the test data. Figure 4 illustrates the perplexity results of the four LMs trained on different data sources tested on a random sample of 733,147 queries from the search engine's May

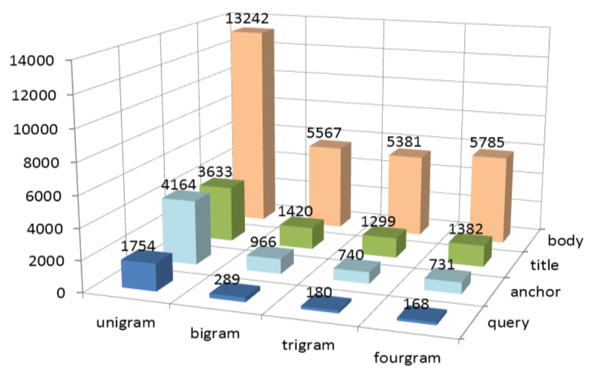

Figure 4. Perplexity results on test queries, using n-gram LMs with different orders, derived from different data sources.

2009 query log. The results suggest several conclusions. First, higher order LMs in general produce lower perplexities, especially when moving beyond unigram models. Second, as expected, the query LMs are most predictive for the test queries, though they are from independent query log snapshots. Third, it is interesting to notice that although the body LMs are trained on much larger amounts of data than the title and anchor LMs, the former lead to much higher perplexity values, indicating that both title and anchor texts are quantitatively much more similar to queries than body texts.

Table 2 summarizes the spelling results using different LMs. For comparison, we also built a 4-gram LM using the Google 1T web 5-gram corpus (Brants and Franz, 2006). This model is referred to as the G1T model, and is trained using the "stupid backoff" smoothing method (Brants et al., 2007). Due to the high count cutoff applied by the Google corpus (i.e., n-grams must appear at least 40 times to be included in the corpus), we found the G1T model results to a higher OOV rate (i.e., 6.5%) on our test data than that of the 4 Web scale LMs (i.e., less than 1%).

The results in Table 2 are more or less consistent with the perplexity results: the query LM is the best performer; there is no significant difference among the body, title and anchor LMs though the body LM is trained on a much larger amount of data; and all the 4 Web scale LMs outperform the G1T model substantially due to the significantly lower OOV rates.

6.3 Error Models

This section compares the phrase-based error model (PBEM) described in Section 5, with one of the state-of-the-art error models, proposed by Brill and Moore (2000), henceforth referred to as

#	System	Accuracy	Precision	Recall
1	Baseline	89.0	74.1	49.6
2	1+ query 4-gram	90.1	75.6	56.3
3	1 + body 4-gram	89.9	75.7	54.4
4	1 + title 4-gram	89.8	75.4	54.7
5	1 + anchor 4-gram	89.9	75.1	55.6
6	1 + G1T 4-gram	89.4	75.1	51.5

Table 2. Spelling correction results using different LMs trained on different data sources.

#	System	Accuracy	Precision	Recall
1	Baseline w/o EM	88.6	72.0	47.0
2	Baseline	89.0	74.1	49.6
3	1 + B&M, *L*=1	89.0	73.3	50.1
4	1 + B&M, *L*=3	89.2	73.7	51.0
5	1 + PBEM, *L*=1	90.1	76.7	55.6
6	1 + PBEM, *L*=3	90.7	78.5	58.1

Table 3. Spelling correction results using different error models.

the B&M model. B&M is a substring error model. It estimates $P(q|c)$ as

$$P(q|c) \approx \max_{\substack{R,T \\ s.t.|T|=|R|}} \prod_{i=1}^{|R|} P(T_i|R_i), \qquad (10)$$

where *R* is a partitioning of correction term *c* into adjacent substrings, and *T* is a partitioning of query term *q*, such that |*T*|=|*R*|. The partitions are thus in one-to-one alignment. To train the B&M model, we extracted 1 billion term-correction pairs (q, c) from the set of 120 million query-correction pairs (Q, C), derived from the search logs as described in Section 5.2.

Table 3 summarizes the comparison results. Rows 1 and 2 are our baseline ranker-based speller systems with and without the error model (EM) feature. The error model is of Equation (3), and is based on weighted edit distance, where the weights are learned on a small set of manually annotated term-correction pairs (which is not used in this study). Rows 3 and 4 are two versions of the B&M model using different maximum substring lengths, specified by *L*. Setting *L*=1 reduces B&M to the weighted edit distance model in Row 2. Rows 5 and 6 are PBEMs with different maximum phrase lengths. PBEM in Row 5 where *L*=1 is equivalent to a word-based error model. The results show the benefits of capturing context information in error models. In particular, the significant improvements resulting from PBEM demonstrate that the dependencies between words are far more effective than that between characters (within a word) for spelling correction. This is largely due to the fact that there are many real-word spelling errors in search queries. We also notice that PBEM is a more powerful and sophisticated model than B&M (e.g. PBEM contains several magnitudes more model parameters than B&M), and can benefit more from increasingly larger training data extracted from search logs. As shown in Tables 4 and 5, whilst the performance of B&M saturates quickly with the increase of training data, the performance of PBEM does not appear to have peaked in our study – further improvements are likely given a larger data set.

#	# of word pairs	Accuracy	Precision	Recall
1	Baseline w/o EM	88.55	71.95	46.97
2	1M	89.15	73.71	50.74
3	10M	89.22	74.11	50.92
4	100M	89.20	73.60	51.06
5	1B	89.21	73.72	50.99

Table 4. The performance of B&M error model (*L*=3) as a function of the size of training data (# of word pairs).

#	# of (*Q*, *C*) pairs	Accuracy	Precision	Recall
1	Baseline w/o EM	88.55	71.95	46.97
2	5M	89.59	77.01	52.34
3	15M	90.23	77.87	56.67
4	45M	90.45	78.56	57.02
5	120M	90.70	78.49	58.12

Table 5. The performance of PBEM (*L*=3) as a function of the size of training data (# of (*Q*, *C*) pairs).

7 Conclusions and Future Work

This paper explores the use of massive Web corpora and search logs for improving a ranker-based search query speller. We show significant improvements over a noisy channel speller using fine-grained features, Web scale LMs, and a phrase-based error model that captures intern-word dependencies. There are several techniques we are exploring to make further improvements. First, since a query speller is developed for improving the Web search results, it is natural to use features from search results (e.g., the frequency of a query term in the snippets of the retrieved documents) in ranking, as studied in Chen et al. (2007). The challenge is efficiency. Second, in addition to query reformulation sessions, we are exploring other search logs from which we might extract more (Q, C) pairs for error model training. One promising data source is clickthrough data (e.g., Agichtein et al, 2006; Gao et al., 2009). For instance, we might try to learn a transformation from the title or anchor text of a document to the query that led to a click on that document. Finally, the phrase-based error model is inspired by phrase-based SMT systems. We are introducing more SMT techniques such as alignment and translation rule exaction. In a broad sense, spelling correction can be viewed as a monolingual MT problem where we translate bad English queries into good ones.

Acknowledgments

The authors would like to thank Andreas Bode, Mei Li, Chenyu Yan and Kuansan Wang for the very helpful discussions and collaboration. The work was done when Xu Sun was visiting Microsoft Research Redmond.

References

Agichtein, E., Brill, E. and Dumais, S. 2006. Improving web search ranking by incorporating user behavior information. In *SIGIR*, pp. 19-26.

Ahmad, F., and Kondrak, G. 2005. Learning a spelling error model from search query logs. In *HLT-EMNLP*, pp. 955-962.

Brants, T., and Franz, A. 2006. Web 1T 5-gram corpus version 1.1. Technical report, Google Research.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. 2007. Large language models in machine translation. In *EMNLP-CoNLL*, pp. 858 - 867.

Brill, E., and Moore, R. C. 2000. An improved error model for noisy channel spelling correction. In *ACL*, pp. 286-293.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, and Hullender, G. 2005. Learning to rank using gradient descent. In *ICML*, pp. 89-96.

Chaiken, R., Jenkins, B., Larson, P., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. 2008. SCOPE: easy and efficient parallel processing f massive data sets. In *Proceedings of the VLDB Endowment*, pp. 1265-1276.

Charniak, E. 2001. Immediate-head parsing for language models. In *ACL/EACL*, pp. 124-131.

Chen, S. F., and Goodman, J. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(10):359-394.

Chen, Q., Li, M., and Zhou, M. 2007. Improving query spelling correction using web search results. In *EMNLP-CoNLL*, pp. 181-189.

Church, K., Hard, T., and Gao, J. 2007. Compressing trigram language models with Golomb coding. In *EMNLP-CoNLL*, pp. 199-207.

Cucerzan, S., and Brill, E. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP*, pp. 293-300.

Freund, Y. and Schapire, R. E. 1999. Large margin classification using the perceptron algorithm. In *Machine Learning*, 37(3): 277-296.

Gao, J., Goodman, J., and Miao, J. 2001. The use of clustering techniques for language modeling -application to Asian languages. *Computational Linguistics and Chinese Language Processing*, 6(1):27–60, 2001.

Gao, J., Yuan, W., Li, X., Deng, K., and Nie, J-Y. 2009. Smoothing clickthrough data for web search ranking. In *SIGIR*, pp. 355-362.

Golding, A. R., and Roth, D. 1996. Applying winnow to context-sensitive spelling correction. In *ICML*, pp. 182-190.

Joachims, T. 2002. Optimizing search engines using clickthrough data. In *SIGKDD*, pp. 133-142.

Kernighan, M. D., Church, K. W., and Gale, W. A. 1990. A spelling correction program based on a noisy channel model. In *COLING*, pp. 205-210.

Koehn, P., Och, F., and Marcu, D. 2003. Statistical phrase-based translation. In *HLT/NAACL*, pp. 127-133.

Kucich, K. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377-439.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707-710.

Li, M., Zhu, M., Zhang, Y., and Zhou, M. 2006. Exploring distributional similarity based models for query spelling correction. In *ACL*, pp. 1025-1032.

Mangu, L., and Brill, E. 1997. Automatic rule acquisition for spelling correction. In *ICML*, pp. 187-194.

Microsoft Microsoft web n-gram services. 2010. http://research.microsoft.com/web-ngram

Nguyen, P., Gao, J., and Mahajan, M. 2007. MSRLM: a scalable language modeling toolkit. Technical report TR-2007-144, Microsoft Research.

Och, F. 2002. *Statistical machine translation: from single-word models to alignment templates.* PhD thesis, RWTH Aachen.

Och, F., and Ney, H. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4): 417-449.

Philips, L. 1990. Hanging on the metaphone. *Computer Language Magazine*, 7(12):38-44.

Sun, X., Gao, J., Micol, D., and Quirk, C. 2010. Learning phrase-based spelling error models from clickthrough data. In *ACL*.

Toutanova, K., and Moore, R. 2002. Pronunciation modeling for improved spelling correction. In *ACL*, pp. 144-151.

Whitelaw, C., Hutchinson, B., Chung, G. Y., and Ellis, G. 2009. Using the web for language independent spellchecking and autocorrection. In *EMNLP*, pp. 890-899.

Zhang, Y., Hildebrand, Al. S., and Vogel, S. 2006. Distributed language modeling for n-best list re-ranking. In *EMNLP*, pp. 216-233.