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

Jianfeng Gao  
Microsoft Research, Redmond  
jfgao@microsoft.com

Xiaolong Li  
Microsoft Corporation  
xiaolong.li@microsoft.com

Daniel Micol  
Microsoft Corporation  
danielmi@microsoft.com

Chris Quirk  
Microsoft Research, Redmond  
chrisq@microsoft.com

Xu Sun  
University of Tokyo  
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 \( Q = q_1, \ldots, q_l \), we want to find the best spelling correction \( C = c_1, \ldots, c_l \) among all candidates:

\[
C^* = \arg \max_C P(C|Q) \tag{1}
\]

Applying Bayes’ Rule, we have

\[
C^* = \arg \max_C P(Q|C)P(C) \tag{2}
\]

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 focuses on how to make the model “smarter” via how to better estimate the probability of unseen words (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 LMs. In addition, we observe that search queries are composed in a language style different from that of regular text. We thus train multiple LMs using different texts associated with Web corpora and search queries.

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 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 (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 spellings and word usag in queries (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, e.g., whether a candidate appears as a Wikipedia document title. We thus generalize the speller to a ranker-based system. Let \( f \) be a feature vector of a query and candidate correction pair \((Q, C)\). The ranker maps \( f \) to a real value \( y \) that indicates how likely \( C \) is a desired correction. For example, a linear ranker maps \( f \) to \( y \) with a weight vector \( w \) such as \( y = w \cdot f \), 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 efficiency, our speller 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 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 weighted edit distance:

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

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 \( w^L = (w_1, ..., w_L) \) according to

\[
P(w^L) = \prod_{i=1}^{L} P(w_i | w_{1:i-1}) = \prod_{i=1}^{L} P(w_i | w_{1:i-1})
\]

where the approximation is based on a Markov assumption that each word depends only upon the immediately preceding \( n-1 \) words. In a spelller, 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 spelller.

#### 4.1 Web Scale Language Models

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

\[
P(w_i | w_{i-2}w_{i-1}) = \begin{cases} C(w_i | w_{i-2}w_{i-1}) - D(C(w_i | w_{i-2}w_{i-1})) & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{i-2}w_{i-1})P(w_i | w_{i-1}) & \text{otherwise} \end{cases}
\]

where \( C(\cdot) \) is the count of the \( n \)-gram in the training corpus and \( \alpha \) is a normalization factor. \( D(C) \) is a discount function for smoothing. We use modified absolute discounting (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 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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Body</th>
<th>Anchor</th>
<th>Title</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total tokens</td>
<td>1.3T</td>
<td>11.0B</td>
<td>257.2B</td>
<td>28.1B</td>
</tr>
<tr>
<td>Unigrams</td>
<td>1.2B</td>
<td>60.3M</td>
<td>150M</td>
<td>251.5M</td>
</tr>
<tr>
<td>Bigrams</td>
<td>464.1M</td>
<td>1.1B</td>
<td>8.3B</td>
<td>13.1B</td>
</tr>
<tr>
<td>Trigrams</td>
<td>60.0B</td>
<td>1.4B</td>
<td>3.1B</td>
<td>3.1B</td>
</tr>
<tr>
<td>4-grams</td>
<td>148.5B</td>
<td>2.3B</td>
<td>5.1B</td>
<td>4.6B</td>
</tr>
<tr>
<td>Size on disk*</td>
<td>12.8TB</td>
<td>183GB</td>
<td>395GB</td>
<td>393GB</td>
</tr>
</tbody>
</table>

* 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).
The smoothing method can be implemented similarly by 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 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 backoff parameter.

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 c1, ..., cK, then each is replaced with a new non-empty word sequence q1, ..., qK. Finally these phrases are permuted and concatenated to form the misspelled Q. Here, c and q denote consecutive sequences of words.

To formalize this generative process, let S denote the segmentation of C into K phrases c1, ..., cK, and let T denote the K replacement phrases q1, ..., qK – we refer to these (c, q) 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. Assuming a uniform probability over segmentations, the phrase-based probability can be defined as:

C: “disney theme park”
S: [“disney”, “theme park”]
T: [“disnee”, “theme part”]
M: (1 → 2, 2 → 1)
Q: “theme part disnee”

Figure 2: Example demonstrating the generative procedure behind the phrase-based error model.
As is common practice in SMT, we use the maximum approximation to the sum:

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

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

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

### 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 $i$th 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 $Q$. The least cost alignment $A^*$ between $Q$ and $C$ is computed 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 ($c_i$, $q_t$). Once the word alignment is fixed, the final permutation is uniquely determined, so we can safely discard that factor. Thus we have:

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

For the sole remaining factor $P(T|C, S)$, we make the assumption that a segmented query $T = q_1 \ldots q_k$ is generated from left to right by transforming each phrase $c_1 \ldots c_k$ independently:

$$P(T|C,S) = \prod_{k=1}^{k} P(q_k|c_k), \tag{9}$$

where $P(q_k|c_k)$ 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).

### 5.2 Training the Error Model

Given a set of $(Q, C)$ pairs as training data, we follow a method commonly used in SMT (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", which we denote as $q$.


---

**Figure 3.** A sample of query reformulation sessions from 3 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".

---

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+theme+part&aq=f&aqi=&oq=hp&gs_sm=spell&spell=1
- http://www.google.com/search?hl=en&ei=888s5wnwq4cM4gS6jyCwBQ&sa=X&oi=spell&resnum=0&ct=result&cd=1&ved=0CA4QBSjAg&biw=1087&bih=784&q=harry+potter+theme+park&spell=1

Yahoo:


Bing:

and then clicks on the resulting spelling suggestion "harry potter theme park". We can "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 strong indication that the spelling suggestion is desired. In this study, from 1 year of sessions, we extracted ~120 million pairs. We found the data set very clean because 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. 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 \( Q_1 \neq Q_2 \). 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 consider a query misspelled, we assumed it 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 query logs from a commercial search engine. The spelling of each query is manually 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 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.

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).

6.1 System Results

Table 1 summarizes the main results of different spelling systems. Row 1 is the baseline speller 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 spellers 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 spellers outperform significantly (p < 0.01) the traditional system based on the noisy channel model (Row 2 vs. Row 1); (2) the spelling 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 spellers. 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 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. The results show that (1) higher order LMs produce lower perplexities, especially when moving beyond unigram models; (2) as expected, the query LMs are most predictive for the test queries, though they are from independent query log snapshots; (3) 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...
Table 2. Spelling correction results using different LMs trained on different data sources.

<table>
<thead>
<tr>
<th>#</th>
<th>System</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baseline w/o EM</td>
<td>88.6</td>
<td>72.0</td>
<td>47.0</td>
</tr>
<tr>
<td>2</td>
<td>1 + query 4-gram</td>
<td>90.1</td>
<td>75.6</td>
<td>56.3</td>
</tr>
<tr>
<td>3</td>
<td>1 + body 4-gram</td>
<td>89.9</td>
<td>75.7</td>
<td>54.4</td>
</tr>
<tr>
<td>4</td>
<td>1 + title 4-gram</td>
<td>89.8</td>
<td>75.4</td>
<td>54.7</td>
</tr>
<tr>
<td>5</td>
<td>1 + anchor 4-gram</td>
<td>89.9</td>
<td>75.1</td>
<td>55.6</td>
</tr>
<tr>
<td>6</td>
<td>1 + G1T 4-gram</td>
<td>89.4</td>
<td>75.1</td>
<td>51.5</td>
</tr>
</tbody>
</table>

Table 3. Spelling correction results using different error models. B&M is a substring error model. It estimates

\[
P(q|c) = \max_{R, T} \prod_{i=1}^{\min(|R|, |R_i|)} P(T_i|R_i),
\]

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 ranker-based baseline systems with and without the error model (EM) feature. The error model is based on weighted edit distance of Eq. (3), where the weights are learned on some manually annotated word-correction pairs (which is not used in this study). Rows 3 and 4 are the B&M models using different maximum substring lengths, specified by \( L \). \( L=1 \) reduces B&M to the weighted edit distance model in Row 2. Rows 5 and 6 are PBEMs with different maximum phrase lengths. \( L=1 \) reduces PBEM 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 model than B&M in that it can benefit more from increasingly larger training data. 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 – further improvements are likely given a larger data set.

7 Conclusions and Future Work

This paper explores the use of massive Web corpora and search logs for improving a ranker-based search query spell checker. We show significant improvements over a noisy channel spell checker using fine-grained features, Web scale LMs, and a phrase-based error model that captures inter-word dependencies. There are several techniques we are exploring to make further improvements. First, since a query spell checker is developed for improving the Web search results, it is natural to use features from search results 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 extraction. 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


