Spelling Alteration for Web Search Workshop

City Center – Bellevue, WA
July 19, 2011

Microsoft Research

bing

Microsoft

Table of Content

Acknowledgements

The organizing committee would like to thank all the speakers and participants as well as everyone who submitted a paper to the Spelling Alteration for Web Search Workshop. Special thanks to the Speller Challenge winners for helping build an exciting and interactive workshop.

Last but not least thank you to the Bing team to bring in their experiences working on real world large scale web scenarios.

Organizing committee

Description

The Spelling Alteration for Web Search workshop, co-hosted by Microsoft Research and Microsoft Bing, addresses the challenges of web scale Natural Language Processing with a focus on search query spelling correction.

The goal of the workshop is the following:

- provide a forum for participants in the Speller Challenge (details at http://www.spellerchallenge.com, with a submission deadline for the award competition on June 2, 2011) to exchange ideas and share experience
- official award ceremony for the prize winners of the Speller Challenge
- engage the community on future research directions on spelling alteration for web search

Organizers

Evelyne Viegas, Jianfeng Gao, Kuansan Wang
Microsoft Research, One Microsoft Way Redmond, WA 98052

Jan Pedersen
Microsoft, One Microsoft Way Redmond, WA 98052

Schedule

Time	Event/Topic	Presenter	Chair
08:30	Breakfast		
09:00	Opening	Evelyne Viegas - Microsoft Research	Jianfeng Gao
09:10	Award presentation	Harry Shum - Corporate Vice President Microsoft	
09:30	Report	Kuansan Wang - Microsoft Research	
10:00	Break – snacks		
10:30	A Data-Driven Approach for Correcting Search Queries	Gord Lueck - Independent Software Developer, Canada	Jianfeng Gao
10:50	CloudSpeller: Spelling Correction for Search Queries by Using a Unified Hidden Markov Model with Web-scale Resources	Yanen Li, Huizhong Duan, ChengXiang Zhai - UIUC, Illinois, USA	
11:10	qSpell: Spelling Correction of Web Search Queries using Ranking Models and Iterative Correction	Yasser Ganjisaffar, Andrea Zilio, Sara Javanmardi, Inci Cetindil, Manik Sikka, Sandeep Katumalla, Narges Khatib, Chen Li, Cristina Lopes - University Of California, Irvine, USA	
11:30	TiradeAI: An Ensemble of Spellcheckers	Dan Stefanescu, Radu Ion, Tiberiu Boros - Research Institute for Artificial Intelligence, Romania	
11:50	Spelling Generation based on Edit Distance	Yoh Okuno - Yahoo Corp. Japan	
12:00	Lunch		

13:00	A REST-based Online English Spelling Checker "Pythia"	Peter Nalyvayko - Analytical Graphics Inc., USA	Kuansan Wang
13:15	Why vs. HowTo: Maintaining the right balance	Dan Stefanescu, Radu Ion - Research Institute for Artificial Intelligence, Romania	
13:30	Panel Discussion	Ankur Gupta, Li-wei He - Microsoft Gord Lueck, Yanen Li, Yasser Ganjisaffar, Dan Stefanescu - Speller Challenge Winners	
14:30	Wrap up		

A Data-Driven Approach for Correcting Search Queries

A Submission to the Speller Challenge

Gord Lueck*
gord.lueck@utoronto.ca

ABSTRACT

Search phrase correction is the challenging problem of proposing alternative versions of search queries typed into a web search engine. Any number of different approaches can be taken to solve this problem, each having different strengths. Recently, the availability of large datasets that include web corpora have increased the interest in purely data-driven spellers. In this paper, a hybrid approach that uses traditional dictionary-based spelling tools in combination with data-driven probabilistic techniques is described. The performance of this approach was sufficient to win Microsoft's Speller Challenge in June, 2011.

1. INTRODUCTION

Spelling correctors for search engines have the difficult task of accepting error-prone user input and deciphering if an error was made, and if so suggesting plausable alternatives to the phrase. It is a natural extension to the existing spell checkers that are common in document editors today, with the added expectation that suggestions leverage additional context provided by other terms in the query. In addition, it is common for search phrases to legitimately include proper nouns, slang, multiple languages, punctuation, and in some cases complete sentences. The goal of a good search phrase corrector would be to take all factors into consideration, evaluate the probability of the submitted query, and to suggest alternative queries when it is probable that the corrected version has a higher likelihood than the input phrase.

A good spelling corrector should only act when it is clear that the user made an error. The speller described herein also errs on the side of not acting when it is unclear if a suggested correction is highly probable. Common speller implementations tend not to suggest phrases that have similar probability if the input query is sane.

*Corresponding Author

Another useful metric to measure input queries woul dbe to determine the number of relevan tsearch results the query has. However, without the luxury of a large search engine from which to measure number of results for candidate phrases, this implementation relies upon probability data solely from Microsoft via their web-ngram service[6].

This paper gives an overview of the methods used to create such a search phrase correction service. An overview of the problem as formulated by the challenge administrators is given, including an outline of some of the assumptions and models that were used in the creation of this algorithm.

A key component to the entry is an error model that takes into consideration probabilities as obtained through a historical bing search query dataset while estimating frequencies of errors within search queries. The widely available hunspell dictionary[1] speller is used for checking for the existence of a dictionary word matching each word in the input query, and for suggesting potential corrections to individual words. Any algorithmic parameters present in the final algorithm are also described, and methods used to fix those parameters are described. Some specific optimizations for the contest are described, and recommendations are made for improvement of the expected recall metric of the contest evaluator.

We describe our approach to the problem, some assumptions that we made, and describe an error model that we formulated to approximate frequencies and locations of errors within search queries. Our algorithm is described, as well as methods for calculating parameters that were used in the final submission to the challenge[4].

2. PROBLEM FORMULATION

The algorithm is evaluated according to the published EF1 metric. Suppose the spelling algorithm returns a set of variations C(q), each having posterior probabilities P(c|q). S(q) is the set of plausable spelling variations as determined by a human, and Q is the set of queries in the evaluation set. The expected precision, EP, is

$$EP = \frac{1}{|Q|} \sum_{q \in Q} \sum_{c \in C(q)} I_P(c, q) P(c|q)$$

and, the expected recall is

$$ER = \frac{1}{|Q|} \sum_{q \in Q} \sum_{a \in S(q)} \frac{I_R(C(q), a)}{|S(q)|}$$

$$\frac{1}{EF1} = 0.5(\frac{1}{EP} + \frac{1}{ER})$$

where the utility functions are:

$$I_R(C(q), a) = 1 if a \in C(q) \text{ for } a \in S(q), 0 \text{ otherwise}$$

$$I_P(c, q) = 1 if c \in S(q), 0 \text{ otherwise}$$

For a given search query q, the algorithm will return a set of possible corrections, each with an assigned probability, such that the probabilities P(c|q) add to unity for $c \in Q$. A successful spelling algorithm will return a set of possible corrections and probabilities that maximizes EF1.

2.1 Assumptions

A number of assumptions guided our submission:

- Binary operators (AND, OR, NOT) are not supported in the search engine. That is, every word or phrase in the query is by default an 'AND' operation.
- Punctuation and its affect on the submitted query would be negligible.
- Most queries would have spaces correctly located within the printable characters.
- The evaluation dataset would be similar to the evaluation dataset in terms of error type, error frequency, phrase length, and frequency of english words vs. proper nouns.
- Dictionary based text corrections would be able to provide the basis for the most obvious spelling errors in the query string.

These assumptions may not be valid for all spelling domains, but seemed to be true based on inspection of the TREC[5] dataset that was to be used for evaluation. These assumptions were not validated, either, but were generated through some iterative testing on the evaluation dataset. It was determined that a sufficient speller could be formulated using basic dictionary based software tools, the joint probability n-gram service provided by Microsoft, and some balancing of probabilities.

3. ALGORITHM

There are two main sections of the algorithm - a generation step wherein a set of alternative spellings are postulated, and second, a step where these alternatives are trimmed and sorted via a probability calculation.

3.1 Generation of Alternatives

The speller recommends corrections for each individual term resulting from the input query having been split on whitespace. Each term is compared against a well known english dictionary. If the term does not exist in the dictionary, the original term plus the N_t most likely spelling suggestions were obtained from this service, and added to the set of possible terms. These individual term corrections were generated from the well known hunspell library[1], using the United States English dictionary that comes with the ubuntu operating system[2]. This implementation only considered an English language dictionary.

3.1.1 Trimming the Search Space

The generation of alternative terms was repeated for each term t in the search phrase. In doing this, the algorithm generates a potentially large number of candidate phrases, as the corrected phrase set consists of the set of all possible combinations of candidate terms. That is a maximum of $(n_t+1)^t$ possible corrections. For a typical query of 5 terms, each having 4 suggestions, that means checking 3125 different combinations of words for probabilities. If more than 100 potential corrections were found, the search space was reduced by halving the number of individual term queries until the number fell below 100. In practice, this trimming step was only rarely performed, but was left in to meet basic latency requirements for a web service. In practice there was no penalty in the contest for having a long running service, and the algorithms' accuracy was likely not affected by this trimming step.

3.2 Probabilistic Error Model

The algorithm is required to return the term P(c|q), the probability that the requester meant the phrase c when q was input. By Bayes' theorem,

$$P(c|q) = \frac{P(q|c) * P(c)}{P(q)} \quad (1)$$

We can reduce this problem for the generation of a single query correction set - for a single query, P(q) is constant and can be ignored.

$$P(c|q) \backsim P(q|c) * P(c) \quad (2)$$

Microsoft's n-gram service provides the ability to query various datasets the relative probability of an arbitrary n-gram c. Our algorithm made little attempt to modify or tune these data for $P(c)$ by altering the Microsoft bing source or date. The crux of the problem, then is to generate a model for $P(q|c)$, the probability that the search query q was input given a possible correction c.

It seems reasonable that $P(q|c)$ decreases exponentially with increasing edit distance between q and c, but that the length of the query is also important. It was postulated that there is an average error rate for typing search queries. The more a user types, the more errors they are likely to make. It was postulated that the probability of an error occurring in a string is a function of its' length and the levenshtein distance from the query to the corrected query.

We modelled our error term $P(q|c)$ based on this. The logarithmic probability as received from the web-ngram service $log(P(c))$ was modified as follows to calculate $P(c|q)$:

$$log(P(c|q)) = log(P(c)) - \frac{r_e lev(q, c)}{|q|}, \quad (3)$$

where lev is the well known levenshtein distance between two strings, and $|q|$ is the length of the query string q. r_e is an unknown constant that represents a kind of error rate for search queries. Combining 3 and 2, we have a simple means

of calculating the probability of a correction given the input query, and for any individual correction out of the generated candidates,

$$P(c_i|q) = \frac{P(c_i|q)}{\prod_{c_j \in C} P(c_j|q)} \quad (4)$$

is the returned probability, such that all probabilities add to one.

3.3 Phrase Splitting

Sometimes no plausable alternate spelling candidates are generated as described above. This can occur if all terms existed in the dictionary or the original query was simply the most optimal according to the above metric. For these queries a second method of correction was attempted, using Microsoft's word breaker service. The search phrase was submitted, without whitespace, to the service advertised as part of the n-gram service[3]. Any alternate versions of the phrase returned were also evaluated according to that described in *Probabilistic Error Model*.

3.4 Determination of Parameters

Two parameters were tuned to the publicly available evaluation dataset before the test was run onthe speller. N_t, the number of term suggestions per input term, and e_R, the error penalization term, were both tuned using a greedy search over the input space. It was determined that the optimal settings were $N_t = 2$ and $e_R = 36$.

Interestingly enough, only two suggestions per term were requested from the hunspell library for words that did not exist in the dictionary. There was little change in the final score with $N_t > 2$.

The other interesting fact was that probabilities were penalized by a factor of 10 according to the error rate of the input query. With $e_R = 36$, the algorithm reduces the probability by a factor of 10 for each edit occurring in a string of length 36. Longer strings are penalized less for having an error, shorter strings are penalized more.

4. SCORE OPTIMIZATION

4.1 Thresholding

A few optimization steps were completed before returning a final result. If the original query was still the most likely phrase of all of the candidates examined, our algorithm returns the search query unchanged, with a probability of 1.0. From a users' standpoint, it would not be desireable in this case to suggest a correction to the input phrase.

The evaluation metric for the contest is the harmonic mean of expected precision (EP) and expected recall (ER). There is a slight weakness in the recall metric in that it is possible that a speller returns many incorrect modifications to a search phrase with very low probability in an effort to gain a high recall score. A perfect recall score can be obtained if any speller returns all of those corrections in the evaluation dataset, despite the probability calculated by the speller.

If, upon completion of the algorithm, there are any number of candidate strings with a score less than 10x less likely than the most likely result, these phrases are returned, but their probabilities are minimized to a very small number ($1x10^{-15}$). The higher probability remaining phrases are increased accordingly, such that the total of all corrections remains 1.0.

4.1.1 Common Errors

A number of very common transformations in the English language were applied to the input query, and returned with this same very low, probability. These include adding and removing the most commonly misplaced characters from the input query. In practice, this not a good strategy but one that simply optimizes for the flawed Expected F1 (EF1) metric for the purposes of the contest only.

To determine the common transformations, the evaluation dataset was examined in terms of the types of errors that occurred, and the characters that were most likely in error. To generate this set of characters, the character differences were calculated in the evaluation dataset for cases where the input phrase had at least one correction as determined by expert. A simple script generated and counted these characters when they were involved in an edit operation mapping the input query to a corrected version. These characters included, in order, ' ', 's', 'e', 't', 'a', 'r', 'i', 'u', 'l', 's', 'n' and so on. These characters were used to calculate by brute force a very large set of possible corrections, and assigned these the same miniscule probability used above, adjusting the algorithmically calculated corrections accordingly to maintain a sum of one across all corrections.

5. CONCLUSIONS

The use of already established algorithms for English language spell checking and word suggestion as well as very large datasets that supply probability information has proved to be a successful approach to writing a speller. This submission made no attempt to tag parts of speech or named entities, but to use only a basic dictionary corrector and the microsoft n-gram web service to approximate probabilities of input strings.

The lack of good quality test data in this space motivated the author to tune towards error statistics in an evaluation dataset. In practice, the extensibility of this approach to a wider audience would need to be tested in a production environment with a wider variety of data.

There is certainly a risk of overfitting any algorithm to a narrow set of training data, only very basic properties of the evaluation dataset were used to tune this algorithm. The error rate, the approach of penalizing corrections with high error rates, and the general method of generating suggestion candidates seems valid.

It is recommended that any future contest based on the expected recall metric be altered to consider the posterior probabilities of the returned corrections.

The algorithm as presented does have its shortfalls. This particular implementation performs very poorly when presented with misspelled proper nouns, slangs, and punctuation variations. No effort was made to accommodate different lexicons. More work is needed in these areas before a

worthy spelling corrector based on this technique could be useful in a production setting.

6. REFERENCES

[1] Hunspell: open source spell checking, stemming, morphological analysis and generation, 2011. http://hunspell.sourceforge.net/.

[2] Ubuntu - Details of package hunspell-en-us in maverick, 2011. http://packages.ubuntu.com/maverick/hunspell-en-us.

[3] Word breaking is a cinch with data - Microsoft Web N-Gram, 2011. http://blogs.msdn.com/b/webngram/archive/2010/11/22/wordbreakingisacinchwithdata.aspx.

[4] Microsoft Research - Speller Challenge, 2011. http://web-ngram.research.microsoft.com/spellerchallenge/Default.aspx.

[5] Text REtrieval Conference (TREC), 2011. http://trec.nist.gov/.

[6] Microsoft Web N-gram Services - Microsoft Research, 2011. http://web-ngram.research.microsoft.com/.

CloudSpeller: Spelling Correction for Search Queries by Using a Unified Hidden Markov Model with Web-scale Resources

Yanen Li, Huizhong Duan, ChengXiang Zhai

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801
{yanenli2, duan9, czhai}@illinois.edu

ABSTRACT

Query spelling correction is a crucial component of moden search engines that can help users to express an information need more accurately and thus improve search quality. In participation of the Microsoft Speller Challenge, we proposed and implemented an efficient end-to-end speller correction system, namely CloudSpeller. The CloudSpeller system uses a Hidden Markov model to effectively model major types of spelling errors in a unified framework, in which we integrate a large-scale lexicon constructed using Wikipedia, an error model trained from high confidence correction pairs, and the Microsoft Web N-gram service. Our system achieves excellent performance on two search query spelling correction datasets, reaching 0.970 and 0.940 F1 scores on the TREC dataset and the MSN dataset respectively.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query Alteration*

General Terms

Algorithms, Performance, Experimentation

Keywords

CloudSpeller, Spelling Correction, Hidden Markov Model

1. INTRODUCTION

The Text Information Management group at the University of Illinois at Urbana-Champaign has participated in the Microsoft Speller Challenge. The task of the challenge is to develop a speller that generates most plausible spelling alternatives for a search query. This paper is a report of our proposed methods, experiments and findings about the problem based on the results and analysis on two spelling datasets.

Spelling correction has a long history [10]. Traditional spellers focused on dealing with non-word errors caused by misspelling a known word as an invalid word form. They typically rely on manually created lexicons and simple distance measures like Levenshtein distance. Later, statistical models were introduced for spelling correction, in which the error model and n-gram language model are identified as two critical components [3]. Whitelaw *et al.* alleviated the cost of building error model by leveraging the Web to automatically discover the misspelled/corrected word pairs [15].

With the advent of the Web, the research on spelling correction has received much more attention, particularly on the correction of search engine queries. Compared with traditional spelling tasks, it is more difficult as more types of misspelling exist on the Web. Research in this direction includes utilizing large web copora and query log [4, 5, 2], employing large-scale n-gram models [Brants et al. 2007], training phrase-based error model from clickthrough data [13] and developing additional features [7].

To address the challenges of spelling correction for search queries, we propose several novel ideas which are implemented in an efficient end-to-end speller system (called CloudSpeller) of high precision and recall. First, we construct a large and reliable lexicon from Wikipedia. The large size (over 1.2 million words) of the lexicon overcomes the limitation of using a small traditional lexicon, such as identifying human names and neologisms. A clean dictionary is also critical for handling non-word spelling errors, which is the most frequent error type in search queries. Second, we propose a Hidden Markov Model to model all major types of spelling errors into a unified framework. A Top-K paths searching algorithm is designed to efficiently maintain a small number of highly confident correction candidates. Third, the candidate paths are finally re-ranked by a ranker which takes into account two major types of features, one from the error model, the other from the n-gram language model. We train the error model with a set of query correction pairs from the web. Web scale language model is obtained by leveraging the Microsoft Web N-gram service. We demonstrate that these two types of features are sufficient for building a highly accurate speller for web queries. Such a system also has the advantage in efficiency compared to systems employing tens of features [7].

2. PROBLEM SETUP AND CHALLENGES

Given a search query, the task of search query spelling

correction is to find the most effective spelling variant of the query that would retrieve better results than the original query. Formally, let Σ be the alphabet of a language and $L \subset \Sigma^*$ be a large lexicon of the language. We define a general search query correction problem as:

Given query $q \in \Sigma^*$, find top-K $q' \in L$ such that $P(q'|q) \in maxK_{t \in L^*} P(t|q)$,

where $P(q'|q)$ is the conditional probability of q' given q that follows the general noisy channel framework.

The problem of search query spelling correction is significantly harder than the traditional spelling correction. Previous researches show that approximately 10-15% of search queries contain spelling errors [5]. We have identified four major types of errors in search queries. (1) non-word substitution, e.g. insertion, deletion, misspelling of characters. This type of error is most frequent in web queries, and it is not uncommon that up to 3 or 4 letters are misspelled. This type of error can be corrected accurately using the noisy channel model with a trusted lexicon. (2) confusable valid word substitution, e.g. "persian golf" → "persian gulf". This type of error can be effectively addressed by the context sensitive spellers[9]. (3) cancatenation of multiple words, e.g. "unitedstatesofamerica" → "united states of america". This problem can be tackled by n-gram model based word breaker [14] (4) splitting a word into parts, e.g. "power point slides" → "powerpoint slides". For each type of errors in search query spelling correction, there are effective solutions. However, predicting the error type given a query is difficult, and it is quite common that more than one type of errors cooccur in a query. Therefore, a successful speller for web queries requires a solution addressing all types of spelling errors with high accuracy.

3. THE CLOUDSPELLER ARCHITECTURE

The CloudSpeller system accepts a search query as input. Then a unified HMM model generates a small list of most likely candidate corrections (paths). After that, a ranker will score and re-rank those candidate paths, and finally generate the top-K corrections as the output. In this section we describe the unified HMM model and the ranker. We will also describe the critical components our HMM model and ranker rely on, namely the large-scale lexicon, the error model and n-gram model.

3.1 An HMM Model for Query Correction

We adopt a generative model for spelling correction. More specifically, we employ a Hidden Markov Model (HMM) for this purpose. The generative process follows a word-by-word process. At the beginning, the user has a word in its correct form in mind, it is then transformed through a noisy channel and becomes potentially misspelled. In this process, it is not only possible to misspell the given word into another word, but also sometimes possible to split the word into several words, or even combine the two types of misspellings. When the user has a second word in mind, he or she may have similar misspellings as the previous word, but may also incorrectly attach the word (or part of it) to the previous word. Note that this HMM is more general than the existing HMMs used for spelling correction [5] because it can model many different kinds of spelling errors.

Formally, let $\theta = \{A, B, \pi\}$ be the model parameters of the HMM, including the transition probability, emission probability and initial state probability. Given a list of query words (obtained by splitting empty spaces), the states in a state sequence are one-to-one corresponding to the query words except for the merging state. Each state is represented by a phrase in Σ^*. Theoretically the phrase in a state can be chosen arbitrarily, however for the sake of efficiency we reduce the state space by only choosing a phrase in the lexicon L^* such that $dist(s,t) \leq \delta$, where $dist(s,t)$ is the edit distance between the state phrase s and word t in the query. Each state also has the type information; indicating whether the state is a substitution, merging, splitting or NULL state. In order to accommodate a merging state we introduce the NULL state. The NULL state doesn't emit any phrase, and the state transition probability is always equal to 1. For the model parameter A, B, we employ the bigram model probability as the state transition probabilities A and use the error model probabilities as the emission probabilities B. After this treatment, the probability of generating the original query from a state sequence (path) is the product of all phrase-to-phrase error model probabilities and the total language modeling probability of the path.

Figure 1 illustrates our HMM model and a generative example. In this example, there are three potential errors with different error types, e.g. "goverment" → "government" (substitution), "home page" → "homepage" (splitting), "illinoisstate" → "illinois state" (concatenation). The state path showed in Figure 1 is one of the state sequences that can generate the query. Take state s_2 for example, s_2 is represented by phrase *homepage*. Since s_2 is a merging state, it emits a phrase *home page* with probability $P(\textit{home page}|\textit{homepage})$ according to the error model. Subsequently s_2 transits to state s_3 with probability $P(s_3|s_2)$ according to the bigram language model.

With this model, we are able to come up with arbitrary corrections instead of limiting ourselves to an incomprehensive set of queries from query log. By simultaneously modeling the misspellings on word boundaries, we are able to correct the query in a more integrated manner. Moreover, to handle the large number of potential corrections, we have designed and implemented a dynamic programming algorithm to compute the Top-K corrections efficiently. If there are n states in a state path, and the maximum number of candidate words for each query term is M, the computational complexity of our algorithm is $O(M^2 \cdot n \cdot K)$. Experiments results show that the recall of Top-40 corrections obtained by this algorithm is about 98.4% in the TREC dataset and 96.9% in the MSN dataset.

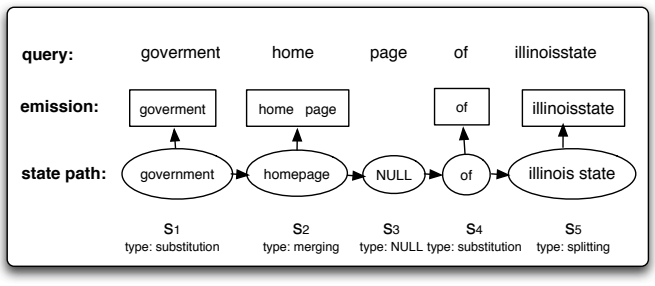


Figure 1: HMM model for query spelling correction

3.2 Candidate Paths Re-ranking

After generating a small list of candidate paths, we propose a ranker to re-rank these paths based on the weighted sum of error model probability and n-gram model probability. We find that probabilities from these two components are most important factors for web query correction. However, a successful speller requires a careful combination of these probabilities. Specifically, for a candidate path q' with n state nodes. We score q' based on the following 7-parameter interpolated model:

$$S_p = \left[\sum_{n=1}^{6} w_n \cdot logP(q_n|q'_n)\right] + w_7 \cdot log(P'(q')) \quad (1)$$

where $P(q_n|q'_n)$ is the error model probability of transforming the phrase q_n from q'_n, $P'(q')$ is the n-gram probability of the phrase in path q'. And $w_n \in \{w_1, ..., w_6\}$ is determined by the type of the state node s_n according to the following rule:

if $q'_n \in L$:
  if q'_n is transformed to q_n by concatenation: $w_n = w_1$
  else if q'_n is transformed to q_n by splitting: $w_n = w_2$
  else: $w_n = w_3$
else if $q'_n \notin L$:
  if q'_n is transformed to q_n by concatenation: $w_n = w_4$
  else if q'_n is transformed to q_n by splitting: $w_n = w_5$
  else: $w_n = w_6$

The model parameters $\{w_1, ..., w_7\}$ are trained by the Powell search algorithm [12] on the development dataset.

3.3 A Large-scale Trusted Lexicon

We find that with a clean vocabulary, it will significantly improve the performance of spelling correction. However, to obtain such a clean vocabulary is usually difficult in practice. To do this, we make use of the Wikipedia data. Particularly, we select the top 2 million words from Wikipedia by their word frequencies, and automatically curate the obtained words by removing those frequent but illegitimate words from the vocabulary. This curate process involves checking if the word appears in the title of a Wikipedia article, comparing the bigram probability of other words etc. Finally we obtained 1.2 million highly reliable words in the vocabulary.

3.4 Error Model Training

The error model intends to model the probability that one word is misspelled into another (either valid or invalid). Previous studies have shown that a weighted edit distance model trained with a sufficient large set of correction pairs could achieve a comparable performance with a sophisticated n-gram model [6]. Meanwhile, a higher order model has more tendency to overfit if the training data is not large enough. Given these considerations, we adopt the weighted edit distance model as the error model in our system. More specifically, we follow the study of Duan and Hsu [6] to model the joint probability of character transformations as the weighted edit distance. In this model, the basic edit operation is defined as a pair of characters from source and destination of the correction, respectively. Null character is included in the vocabulary to model the insertion and deletion operation. The misspelled word and its correction are viewed as generated from a sequence of edit operations. The parameters in this model are trained with an EM algorithm which iteratively maximizes the likelihood of the training set of correction pairs. To obtain a proper set of correction pairs for training, we leverage the existing spelling services of major search engines (Google and Bing). We submit the queries to the spelling services and record the corrections once consensus is reached.

3.5 Use of Web N-gram Model

Another important factor in selecting and ranking the correction candidates is the prior probability of a correction phrase. It represents our prior belief about how likely a query will be chosen by the user without seeing any input from the user. In this work we make use of the Web n-gram service provided by Microsoft [1]. Web n-gram model intends to model the n-gram probability of English phrases with the parameters estimated from the entire Web data. It also differentiates the sources of the data to build different language models from the title, anchor text and body of Web pages, as well as the queries from query log. In our study, we find the **title** model is the most effective for query spelling correction. We hypothesize that this may be because the training data for query model is much noisier. Particularly, misspelled queries may be included in the training data, which makes it less effective for the task of spelling correction. Despite trained with the Web data, Web n-gram model may also suffer from data sparseness in higher order models. To avoid this issue, we make use of the **bigram** model in building our spelling system.

4. EXPERIMENTS AND DISCUSSION

In order to evaluate the performance of CloudSpeller, we have tested it on two query spelling correction datasets. One is the TREC dataset based on the publicly available TREC queries (2008 Million Query Track). This dataset contains 5892 queries and corrections annotated by the Speller Challenge organizers. There could be more than one plausible corrections for a query. In this dataset only 5.3% of queries are judged as misspelled. We also annotated another dataset that contains 4926 from the MSN queries, for each query there is at most only one correction. About 13% of queries are judged as misspelled in this dataset, which is close to the error rate of real web queries. We divide the TREC and MSN datasets into training and test set evenly. CloudSpeller is trained on the training sets and finally evaluated on the TREC test set containing 2947 queries and MSN test set containing 2421 queries.

4.1 Results

We follow the same evaluation metrics as the Speller Challenge, and report results on TREC and MSN datasets in Table 1. Top 40 corrections are used in the default setting of CloudSpeller. The results indicate that CloudSpeller is of very high precision and recall in TREC dataset. In the MSN dataset which is considered harder since it has more misspelled queries, CloudSpeller also achieves high precision of 0.912 and recall of 0.969. This suggests CloudSpeller is very effective for handling spelling errors in search queries overall. We also break down the results by error types so that we can see more clearly the distribution of types of spelling errors and how well our system addressing each type of errors. We present the results of this analysis on Table 2. The breakdown results show that most queries are in the group of "no error", which are much easier to correct than the other

three types. As a result, the overall excellent performance was mostly because the system performed extremely well on the "no error" group. Indeed, the system has substantially lower precision on the queries with the other three types of errors. The splitting errors seem to be the hardest to correct, followed by the concatenation errors, and the substitution errors seem to be relatively easier.

Table 1: Results on TREC and MSN dataset

dataset	#queries	precision	recall	f1
TREC	2947	0.955	0.984	0.970
MSN	2421	0.912	0.969	0.940

Table 2: Results by spelling error type

dataset	error type	% queries	precision	recall	f1
TREC	no error	94.7	0.983	0.986	0.984
	substitution	3.9	0.391	0.970	0.557
	concatenation	0.8	0.352	0.929	0.510
	splitting	0.6	0.301	0.945	0.457
MSN	no error	87.0	0.971	0.973	0.972
	substitution	10.1	0.475	0.904	0.623
	concatenation	1.6	0.328	0.886	0.479
	splitting	1.3	0.304	0.866	0.450

4.2 Number of Spelling Corrections

The Speller Challeng encorages participants to generate all plausible query corrections. However it's unknown that how many spelling corrections are enough. In this section we investigate the effect of number of spelling corrections to the results. We have carried out experiments on five correction size (1,5,10,20,40) on both datasets. Figure 2 summarizes the results. It's clear that a bigger number of corrections leads to higher recall, and the most drastical increase of recall lies from 1 correction to 5 corrections. But the correction size has no effect on precision on both datasets, which suggests that the correction size doesn't affect the top-ranked spelling correction.

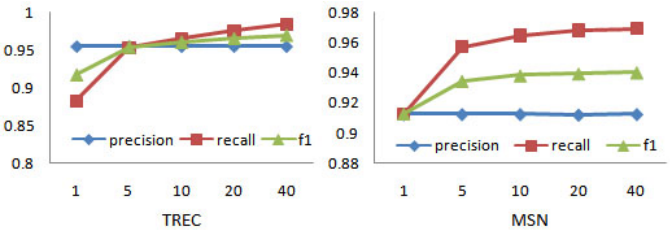


Figure 2: Results by number of corrections

4.3 Effect of the Lexicon Size

The size of the trusted lexicon is an important factor influencing the speller correction result. In order to investigate the effect of lexicon size we conducted a set of experiments on both datasets according to different lexicon sizes (ranging from 100,000 to 900,000). Results in Figure 3 shows the effect of lexicon size is significant on precision: the precision increases as the lexicon size increases. However the recall is not sensitive to the lexicon size.

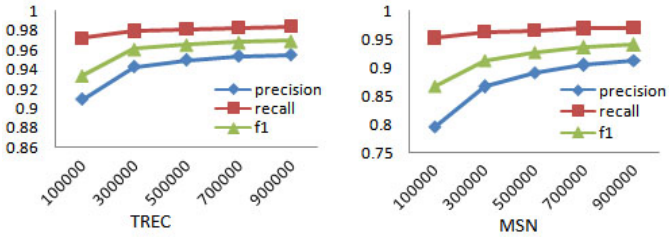


Figure 3: Correction results by lexicion size

4.4 Clean VS Noisy Lexicon

In Table 3 we show the effects of using a clean lexicon for improving the precision and recall of the spelling system. We can see that for both two test dataset, there is noticeable improvement in precision. By removing the noise in the automatically constructed lexicon, our system is able to find matches for candidate queries more precisely. It is interesting that we also observe small improvement in recall for the second test dataset. This is reasonable as we have to limit the number of output candidates in our system due to performance consideration. By reducing the number of matches against the noisy terms, we are able to include more promising results in our ranker, and hence able to improve the overall recall.

Table 3: Clean VS noisy lexicon

dataset	lexicon type	precision	recall	f1
TREC	clean lexicon	0.955	0.984	0.970
	noisy lexicon	0.950	0.984	0.966
MSN	clean lexicon	0.912	0.969	0.940
	noisy lexicon	0.896	0.967	0.930

5. CONCLUSIONS

The key novelty of our system lies in the unified Hidden Markov model that successfully models all major types of spelling errors in search queries, which is under addressed by previous works. The large and clean lexicon, advanced error model and n-gram model are also critical to our system. In the future, we want to directly train the HMM model with examples, removing the need to re-rank the candidate paths. We are also investigating a better way to cache n-gram probabilities, which is crucial to the speed of our system.

6. ACKNOWLEGEMENTS

This work is supported in part by MIAS, the Multimodal Information Access and Synthesis center at UIUC, part of CCICADA, a DHS Center of Excellence, by the National Science Foundation under grants IIS-0713581 and CNS-0834709, and by a gift grant from Microsoft.

7. REFERENCES

[1] http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx.

[2] F. Ahmad and G. Kondrak. Learning a spelling error model from search query logs. In *HLT/EMNLP*. The Association for Computational Linguistics, 2005.

[3] E. Brill and R. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, 2000.

[4] Q. Chen, M. Li, and M. Zhou. Improving query spelling correction using web search results. In *EMNLP-CoNLL*, pages 181–189. ACL, 2007.

[5] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[6] H. Duan and B.-J. P. Hsu. Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 117–126, New York, NY, USA, 2011. ACM.

[7] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A large scale ranker-based system for search query spelling correction. In C.-R. Huang and D. Jurafsky, editors, *COLING*, pages 358–366. Tsinghua University Press, 2010.

[8] A. R. Golding. A bayesian hybrid method for context-sensitive spelling correction. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 39–53, Boston, MA, 1997. ACL.

[9] A. R. Golding and D. Roth. Applying winnow to context-sensitive spelling correction. *CoRR*, cmp-lg/9607024, 1996. informal publication.

[10] K. Kukich. Techniques for automatically correcting words in text. *ACM computing surveys*, 24(4), 1992.

[11] L. Mangu and E. Brill. Automatic rule acquisition for spelling correction. In *Proceedings of the 14th International Conference on Machine Learning (ICML-97)*, pages 187–194, Nashville, TN, 1997. Morgan Kaufmann.

[12] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964.

[13] X. Sun, J. Gao, D. Micol, and C. Quirk. Learning phrase-based spelling error models from clickthrough data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 266–274, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[14] K. Wang, C. Thrasher, and B.-J. P. Hsu. Web scale nlp: a case study on url word breaking. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 357–366, New York, NY, USA, 2011. ACM.

[15] C. Whitelaw, B. Hutchinson, G. Chung, and G. Ellis. Using the web for language independent spellchecking and autocorrection. In *EMNLP*, pages 890–899. ACL, 2009.

qSpell: Spelling Correction of Web Search Queries using Ranking Models and Iterative Correction

Yasser Ganjisaffar, Andrea Zilio, Sara Javanmardi, Inci Cetindil,
Manik Sikka, Sandeep Katumalla, Narges Khatib, Chen Li, Cristina Lopes
School of Information & Computer Sciences
University of California, Irvine
Irvine, CA, USA
yganjisa@ics.uci.edu

ABSTRACT

In this work we address the challenging problem of Web search queries in order to build a speller that proposes the most plausible spelling alternatives for each query. First we generate a large set of candidates using diverse approaches including enumerating all possible candidates in edit distance of one, fuzzy search on known data sets, and word breaking. Then, we extract about 150 features for each query-candidate pair and train a ranking model to order the candidates such that the best candidates are ranked on the top of the list. We show that re-ranking top results, iterative correction, and post-processing of the results can significantly increase the precision of the spell checker. The final spell checker, named *qSpell*[1], achieves a precision of 0.9482 on a test data set randomly collected from real search queries. qSpell won the 3rd prize in the recent Microsoft's Speller Challenge.

1. INTRODUCTION

The spelling correction problem is typically formulated using the noisy channel model [9]. Given an input query q, we want to find the best correction c^* among all the candidate corrections:

$$c^* = \underset{c}{\text{argmax}} P(q|c)P(c),$$

where $P(q|c)$ is the error model probability which shows the transformation probability from c to q, and $P(c)$ is the language model probability which shows the likelihood that c is a correctly spelled query.

The noisy channel model only considers error model and language model probabilities. However, a Web-scale spelling correction system needs to consider signals from diverse sources to come up with the best suggestions. We treat the spelling correction problem as a ranking problem where we first generate a set of candidates from the query and then rank them by learning a ranking model on the features extracted from them. The ranking process is expected to bring the best candidates to the top of the ranked list.

We extract several features from query-candidate pairs and use a machine learning based ranking algorithm for training a ranking model on these features. The top-k results of this ranking model are then re-ranked using another ranker. This step further increases the quality of the ranked list.

Given that it is hard to generate the best candidates for queries that require several corrections, we use an iterative approach which applies one fix at each iteration. In the final step, we use a rule-based system to process the final ranked list and decide how many candidates from the top of the list should be suggested as possible good candidates for this query. In the next sections, we describe more details of our spell correction system.

[1]http://flamingo.ics.uci.edu/spellchecker/

Spelling Alteration for Web Search Workshop, July 2011,
Bellevue, WA, USA

2. ERROR MODEL

For computing error model probabilities we used the approach proposed by Brill in [9]. To train this model, we needed a training set of $\langle s_i, w_i \rangle$ string pairs, where s_i represents a spelling error and w_i is the corresponding corrected word. We extracted these pairs from the query reformulation sessions that we extracted from the AOL query log [1]. We processed this query log and extracted pairs of queries (q_1, q_2) which belonged to the same user and were issued within a time window of 5 minutes. The queries are further limited to cases where the edit distance between q_1 and q_2 is less than 4 and the user has not clicked on any result for query q_1 and has clicked on at least one result for query q_2. We processed about $20M$ queries in this query log and extracted about $634K$ training pairs. Following the approach proposed in [9] for each training pair (q_1, q_2), we counted the frequencies of edit operations $\alpha \rightarrow \beta$. These frequencies are then used for computing $P(\alpha \rightarrow \beta)$, which shows the probability that when users intended to type the string α they typed β instead.

To extract edit operations from the training set, we first aligned the characters of q_1 and q_2 based on their Damerau-Levenshtein distance [13]. Then for each mismatch in the alignment, we found all possible edit operations within a sliding window of a specific size. As an example, we extract the following edit operations from the training pair $\langle$`satellite`, `satillite`$\rangle$:

- Window size 1: `e` $\rightarrow$ `i`;

Table 1: Language Model Datasets

Dataset	No. of tokens	No. of ngrams
Google	1T	615M (up to trigrams)
Yahoo	3B	166M (up to trigrams)
Wikipedia Body	1.4B	134M (up to trigrams)

- Window size 2: `te` → `ti`, `el` → `il`;
- Window size 3: `tel` → `til`, `ate` → `ati`, `ell` → `ill`.

We counted the frequency of each of these edit operations in the training set and used these frequencies for computing probability of each edit operation and picking the most probable edits when computing the error model probabilities for different candidates.

3. LANGUAGE MODELS

We implemented three different smoothed language models in our system: Stupid Backoff [8], Absolute Discounting [10], and Kneser Ney [10] smoothing. Moreover each of these language models is computed on three different datasets: Google ngrams [7], Yahoo ngrams [3], and Wikipedia body ngrams (Table 1). Each of these data sets has different properties and we were expecting this diversity to improve the accuracy of our speller. The Google ngrams dataset is collected from public Web pages and therefore also includes many misspellings as well. On the other hand, the Yahoo ngrams dataset is collected from news Websites and therefore is cleaner. However, in terms of number of tokens it is much smaller than Google ngrams dataset. Both of these datasets are based on crawls of the Web in 2006 and therefore do not cover new topics. For this reason, we also created an ngram dataset from Wikipedia articles by processing the dump of Wikipedia articles content released in Jan 2011 [2]. Table 1 shows the properties of these datasets. We used a MapReduce cluster to normalize the ngrams in these datasets. For example "`The`", "`the`" and "`THE`" are all normalized to "`the`" and their frequencies are aggregated.

Each of these language models has parameters that need to be tuned. We used parallel grid search on a MapReduce cluster to tune the parameters of these Language models for unigram, bigram and trigram levels. To train these language models, we used a training set of (q, c^*) pairs, where q is a query and c^* is the optimal candidate for this query. The goal of the training process was to find optimal values for language model parameters such that the conditional probability $P(c^*|q)$ is maximized:

$$P(c^*|q) = \frac{P(q|c^*)P(c^*)}{\sum_i P(q|c_i)P(c_i)}$$

As for the training set, we extracted $634K$ pairs of (q, c^*) from the AOL query log as explained in Section 2.

In addition to the above-mentioned word based language models, we also used a character based language model. We used a Hadoop job to extract the frequencies of 1–5 character grams from the Google data set and used it with Stupid Backoff smoothing as an additional language model. Note that this language model is not sparse and therefore we rarely need to backoff to lower-order ngrams. Therefore the choice of the smoothing method does not have any significant effect on the features that are extracted from this language model.

4. NGRAMS SERVICE

Given that we needed to query the ngram datasets very frequently it was important to have a service which can return the frequency of each given ngram in a short amount of time. For this purpose, we used the "*compress, hash and displace*" algorithm as described in [6] to create a minimal perfect hash (MPH) function for each dataset. This hash function is constructed on a set of N grams and assigns a unique number between 1 and N to each of them. This unique number can then be used to read the frequency of the ngram from an array.

While the hash function is guaranteed to return a unique number for each of the ngrams in the corpus, it still returns some unique number between 1 and N for any other string that has not been in the corpus. To reduce the occurrence of these false positives, we followed the approach suggested in [12]. In this approach, in addition to storing the frequencies of ngrams, we stored a fingerprint for each ngram as well. These fingerprints help in detecting a significant portion of the false positives.

Figure 1 demonstrates this process. The hash function returns a unique number for each ngram. This unique number is then used as an index to read the fingerprint and frequency of the ngram from an array. We first check the fingerprint of ngram and verify that it matches the expected fingerprint. If fingerprints do not match, we return zero frequency. Otherwise the frequency value which is stored in the array is returned.

We store a 20-bit fingerprint for each ngram. The largest frequency in Google data sets is 23,142,960,758 which can be stored in 35 bits. However, there are only 528,917 unique frequency numbers in this data set. Therefore, to save memory space, we sorted the unique frequency numbers and stored them in an array with 528,917 elements. Then for each ngram instead of storing the raw frequency of the ngram, we stored the index into this unique frequencies array (Figure 1).

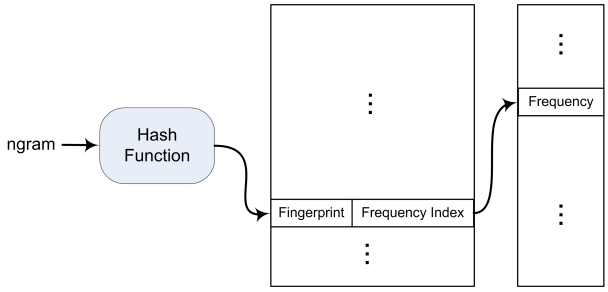


Figure 1: A memory-efficient data structure for fast look up of ngram frequencies

Using this approach we would be saving 15 bits per ngram. We were able to fit the Normalized Google ngrams dataset (unigrams to trigrams) in about 3.2GB of memory and the look up time was about 3 micro seconds. We measured the false positives rate by submitting the four-grams and five-grams in this data set and the rate was 9.6×10^{-7} which was small enough for our purpose.

5. CANDIDATE GENERATION

We implemented three different candidate generators in

Table 2: Word segmentation with different data sets and language model smoothing techniques

Dataset	Language Model Smoothing	Recall
Google	Stupid Backoff	97.06%
Google	Absolute Discounting	96.03%
Google	Kneser Ney	90.86%
Yahoo	Stupid Backoff	95.11%
Yahoo	Absolute Discounting	94.39%
Yahoo	Kneser Ney	93.44%
Wikipedia	Stupid Backoff	94.59%
Wikipedia	Absolute Discounting	93.40%
Wikipedia	Kneser Ney	91.85%

our system that generate about 3,000 candidates for each query. First, a character-based candidate generator generates all possible candidates within an edit distance of 1. It considers replacing each character with all possible characters in the alphabet, transposing each pair of adjacent characters, deleting each character and etc.

A quick analysis of the AOL query logs showed that 16% of the query corrections that we had extracted from this query log differ from the original query only in adding/removing spaces. The followings are some examples:

- `ebayauction` → `ebay auction`
- `broccoliandcheesebake` → `broccoli and cheese bake`
- `i cons` → `icons`

In order to handle this class of queries, we implemented the word segmentation algorithm described in [15] with some minor changes. For each possible segmentation, we query a language model to compute the probability of different segmentations of the query. The most probable segmentations are then added to the candidate list.

In order to find out which data set and language model smoothing is more appropriate for the word segmentation task, we filtered the (q, c^*) pairs that were extracted from AOL query logs and extracted 40,000 pairs where the difference of query q and the expected candidate c^* is only in space. Then we used this data set to see which choice of the ngrams data sets and language model smoothing techniques maximized the probability of generation of c^* after applying the word segmentation algorithm on the query. Table 2 shows the results. As this table shows, the Stupid Backoff language model on Google ngrams data set outperformed the others and therefore we used this combination in our word segmentation module.

None of the above two candidate generators can generate candidates where corrections are at edit distances of more than 1 and not only in adding or removing spaces. For example, for the query "`draigs list irvine`" we want to have "`craigslist irvine`" as a candidate. We used the Flamingo package[2] to perform fuzzy search and quickly find known unigrams with a small edit distance to unigrams and bigrams of the query. We extracted 790K most popular unigrams from the Google ngrams dataset and 947K most popular unigrams from the Wikipedia dataset. This step resulted in a set of 1.3M unigrams which were indexed by the Flamingo package.

[2]http://flamingo.ics.uci.edu/

6. LEARNING TO RANK CANDIDATES

In this section, we describe the details of the machine learning approach that we took for training a ranking model that orders candidates. We first needed to have a training data set. During the speller challenge competition, participants were provided with a data set of publicly available TREC queries from 2008 Million Query Track [4]. The data set also includes human judgments for correct spellings of each query as suggested by several human assessors. However, queries in this data set are sampled from queries which have at least one click in the .gov domain [5]. Therefore this data set was highly biased towards a special class of queries and was not a good representative for the general search queries.

6.1 Training Data Set

To have an unbiased data set for training and testing the ranking models, we randomly sampled 11,134 queries from the publicly available AOL and 2009 Million Query Track query sets and asked 8 human assessors to provide spelling corrections for these queries. Each query was judged by at least one human assessor. Queries for which the human assessors had provided at least one suggestion which was different from the original query were reviewed by at least another human assessor. In order to assist the human assessors in providing the most plausible suggestions for each query, we had designed an interface that was showing Google and Bing search results for each query.

A total of 12,042 spelling suggestions were proposed for the queries. From these suggestions 2,001 of them are different from the original query; while in 905 of the cases the difference is only in adding or removing spaces. Out of the remaining 1,096 cases, 73% of the suggestions are at edit distance of 1 from the original query and 23% of the suggestions are at edit distance of 2 from the original query.

We used 6,000 of the queries in our data set for 5-fold cross validation and determining which features are helpful to be included in the feature set. The remaining 5,134 queries are kept untouched for evaluation purposes. We refer to this dataset as JDB2011 in the rest of this paper and it is available publicly[3]. A spell checker which always returns the original query without any change will get a baseline precision of **0.8971** on the test queries.

In order to train a ranking model on this data set, we need to assign a label to each query-candidate pair to show the preference for different candidates. Then the ranking model would be trained on these samples to learn to rank better candidates on top of the others. As mentioned in section 5, we generated about 3,000 candidates for each query. We want the candidates that match suggestions provided by human assessors to be ranked on top of the list. Then we prefer candidates which are within an edit distance of 1 from any of these suggestions and etc. Therefore, we labeled the candidates according to this preference logic and used these labels for training the ranking model.

6.2 Ranking Features

We extracted 89 *raw features* for each query-candidate pair. These features include: error model features, candidate language model features, query language model features, surface features capturing differences of the query and

[3]http://flamingo.ics.uci.edu/spellchecker/

Table 3: Entity Datasets

Source	No. of Entities
Wikipedia titles	3.1 M
dmoz domains	2.1 M
dmoz titles	6.5 M
IMDB	3.3 M
Porno domains	448 K

the candidate, frequency of the query and the candidate and their substrings in different lists such as Wikipedia titles, dmoz and imdb. Table 3 shows the number of entities in different lists that we used for this purpose.

In addition to these raw features, we also extracted 49 *meta features* for each query-candidate pair. Meta features are computed by applying some basic operations on the original ranking features. For example, we expected a good candidate to be highly probable based on the error model probability, $P(q|c)$. Also we expected the ratio of the $P(c)/P(q)$ to be high. Therefore we used the following meta-feature which combines these three ranking features:

$$\log P(q|c) + \log P(c) - \log P(q).$$

6.3 Ranking Model

We used an Ensemble of gradient boosted trees [16] for learning a ranking model from the extracted features. Given that only about 15% of the queries in our data set needed spell correction, the training data set was imbalance and therefore the ranker was preferring the candidate which is equal to query (as this option is correct in 85% of the cases). For handling this problem we randomly selected several balanced datasets from the original training set and trained different ensembles on each of them. The final ranker is a bagged ensemble that uses the average of these ensembles to determine the ordering of candidates. As mentioned in [11], the bagged ensemble also results in a more accurate model that also has lower variance. The precision at position 1 for this ranker on JDB2011 test set is **0.9055**.

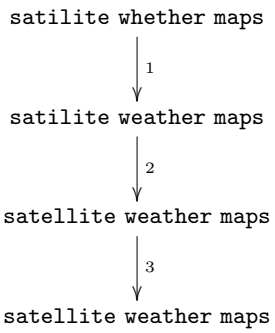
7. RE-RANKING TOP CANDIDATES

Re-ranking of top search results has shown to improve the quality of rankings in information retrieval problems [14]. Given that in spelling correction the main focus is on the top of the ranked list of candidates, we added a re-ranker module on top of the ranker module. This significantly improved the results of the original ranker. There are two main reasons for the success of the re-ranker: 1) It focuses only on top-k results and therefore it is solving a much easier problem compared to the original ranker which needs to consider about $3,000$ candidates for each query. 2) In the re-ranking phase we added some more features which are extracted from top k results. These features include the score computed in the ranking phase, average, max, min, and standard deviation of the scores of the top-k candidates, etc. In contrast to the original ranking features, which are only extracted from query-candidate pairs, these new features also consider the top candidates of the query and therefore include valuable information that may help in the ranking process.

The precision at position 1 on JDB2011 test set after re-ranking increases to **0.9412** which is significantly better than the precision of the ranking module.

8. ITERATIVE CORRECTION

The spelling correction approach that was described in the previous sections does not handle more complex cases where multiple terms in the query need to be corrected, or a single term needs multiple corrections and the correct term is not among the originally generated candidates. For handling these cases, we use iterative correction to apply one correction at each iteration. For example, for the query "`satilite whether maps`" we have the following iterations:



As this example shows, we stop when there is no change the query in the last iteration. The precision at position 1 on JDB2011 test set after applying iterative correction increases to **0.9453**.

9. FINAL POST-PROCESSING

In the final phase and after iterative correction is stopped, we need to pick one or more candidates from the top of the ranked list of candidates. For this purpose, we used post-processing rules that process the final ranked list of candidates and decide which ones should be picked as suggested corrections. For example, most of the times we want to suggest the top ranked candidates as well as other candidates that are on top of the list and their score is within a threshold from the score of the top-ranked candidate and differ from this candidate only in adding or removing space or in singular/plural cases.

In addition, we used some other rules that target different classes of queries. For example, given that a large portion of search queries are navigational queries, we use different lists compiled from dmoz and porno blacklists (Table 3) to detect domain names and treat them specially. In case of domain names, we show the original query and the most probable segmentation of the query. For example for query, "`annualcreditreport`" we suggest both "`annualcreditreport`" (which is a domain name) and "`annual credit report`" which is its most probable segmentation.

After applying the post-processing rules, the precision at position 1 on JDB2011 test set increases to **0.9482**.

Acknowledgments

Authors would like to thank Amazon.com for a research grant that allowed us to use their MapReduce cluster and Alexander Behm for his advise on using the Flamingo package. This work has been also partially supported by NIH grant 1R21LM010143-01A1 and NSF grants OCI-074806 and IIS-1030002.

10. REFERENCES

[1] AOL query log. `http://www.gregsadetsky.com/aol-data/`.

[2] English wikipedia dumps. `http://dumps.wikimedia.org/enwiki/latest/`.

[3] Yahoo! Webscope dataset, N-Grams, version 2.0. `http://research.yahoo.com/Academic_Relations`.

[4] Microsoft speller challenge trec data, 2011. `http://tinyurl.com/6xry3or`.

[5] J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, and E. Kanoulas. Million query track 2008 overview. In E. M. Voorhees and L. P. Buckland, editors, *The Sixteenth Text REtrieval Conference Proceedings (TREC 2008)*. National Institute of Standards and Technology, December 2009.

[6] D. Belazzougui, F. Botelho, and M. Dietzfelbinger. Hash, displace, and compress. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 682–693. Springer Berlin / Heidelberg, 2009.

[7] T. Brants and A. Franz. Web 1T 5-gram Version 1, 2006.

[8] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In *In EMNLP*, pages 858–867, 2007.

[9] E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 286–293, 2000.

[10] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 310–318, 1996.

[11] Y. Ganjisaffar, R. Caruana, and C. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *SIGIR 2011: Proceedings of the 34th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2011.

[12] D. Guthrie and M. Hepple. Storing the web in memory: space efficient language models with constant time retrieval. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 262–272, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[13] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[14] I. Matveeva, A. Laucius, C. Burges, L. Wong, and T. Burkard. High accuracy retrieval with multiple nested ranker. In *SIGIR*, pages 437–444, 2006.

[15] P. Norvig. *Beautiful Data*, chapter Natural language corpus data, pages 219–242. Calif.: OŠReilly, 2009.

[16] Q. Wu, C. Burges, K. Svore, and J. Gao. Ranking, boosting and model adaptation. Technical report, Microsoft Technical Report MSR-TR-2008-109, 2008.

TiradeAI: An Ensemble of Spellcheckers

Dan Ştefănescu
Institute for Artificial Intelligence, Romanian Academy
Calea 13 Septembrie nr. 13
Bucharest, 050711, Romania
0040213188103
danstef@racai.ro

Radu Ion
Institute for Artificial Intelligence, Romanian Academy
Calea 13 Septembrie nr. 13
Bucharest, 050711, Romania
0040213188103
radu@racai.ro

Tiberiu Boroş
Institute for Artificial Intelligence, Romanian Academy
Calea 13 Septembrie nr. 13
Bucharest, 050711, Romania
0040213188103
tibi@racai.ro

ABSTRACT
This paper describes three different offline query spell checkers that have been entered in the Microsoft Speller Challenge competition and a method of combining their output in order to obtain better results. The combination is justified by the fact that each of the spellers has been developed with a different design and that, on the Speller Challenge TREC Data, their errors are complementary, i.e. they do not make the same mistakes.

Categories and Subject Descriptors
G.3 [**Probability and Statistics**]: – *statistical computing*, H.1.2 [**User/Machine Systems**]: – *human information processing*, H.3.3 [**Information Search and Retrieval**]: – *query formulation, relevance feedback*, I.2.6 [**Learning**]: – *parameter learning, induction*, I.2.7 [**Natural Language Processing**]: – *language models*.

General Terms
Algorithms, Experimentation, Human Factors, Languages.

Keywords
Spellchecking, Maximum Entropy classifiers, Viterbi search, language models, WordNet, ensemble classifiers.

1. INTRODUCTION
Approximately 30.2% of the world population uses the Internet in searching for diverse information [9]. Whether is the results of their favorite sports team or the weather forecast for the next day, these millions of people take their queries to a search engine in hope to find useful information. Queries are often typed in haste, sometimes on devices having small keyboards and usually by non-native English speakers searching for information in English. This is why misspelling, due to typographic or cognitive errors [13], is an ordinary fact among search engines queries. Consequently, the users' preference towards a search engine or another is highly influenced by their capability to deal with such errors and the performance of a web searcher will ultimately depend on how it manages to detect what the users are really looking for.

Studies show that 10 to 12 percent of all query terms entered into Web search engines are misspelled [10, 4] due to a variety of reasons [6], such as the accidentally hitting of an adjacent key on the keyboard (*movius* – *mobius*), typing quickly (*teh* – *the*), inconsistent spelling rules (*goverment* – *government*), ambiguous word breaking (*health care* – *healthcare*) or just new words (*exenatide*, *baggravation*, *funkinetics*).

Search engines have two ways of helping their users express and retrieve the desired information: the *offline spelling correction* or the *online spelling correction* [6]. The latter has some advantages over the former, first of all because it can eliminate the mistakes in spelling and the ambiguity of the query in real time, while the user is still typing – something very similar to auto-completion. It is nowadays employed by well-known search engines like *Bing*, *Google* or *Altavista* and requires a large amount of *as much as possible* training data (both correct and misspelled), which can be automatically acquired by gathering users' input and analyzing it.

This paper shortly presents three different offline query spell checkers enrolled in the Microsoft Research Speller Challenge competition. The individual experiments revealed that they behave differently, often suggesting variants that are differently ranked and usually making different mistakes. This brought us in the position of considering combining their results, as it is well-known that, when certain conditions are met, more decision makers (classifiers) working together are likely to find a better solution than when working alone [5, 14]. Such conditions require similar performance for the decision makers and, in addition, that they should not make similar errors.

The paper describes each of the spellers and the method of combining their results.

2. THE FIRST ALGORITHM
The dataset used as the primary resource for this algorithm is based on the 2006 *Google Web 1T 5-gram Corpus*[1] (GC) (an alternative taken into consideration was the Books N-gram Corpus [11] but it did not have a good impact on the accuracy of the algorithm). To build the speller's lexicon, only occurrences more frequent than 300,000 were taken into consideration (in order to avoid misspellings) and also all words obtained from the Princeton WordNet 3.0 (PWN) [7] ignoring their frequencies. Using several English inflection rules such as plural forming on regular nouns, present participle of verbs and superlative and comparative of adjectives, the lexicon was enriched with more than 30,000 words.

Another way to enrich the lexicon was by adding co-occurrences existing in PWN, which are considered *strong n-grams*: *william jefferson clinton*, *september 11*, etc. They contain up to nine terms and have a high weight in computing the final score. Strong n-grams are also used for finding missing words between terms if it

[1] http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

happens that a query contains a strong n-gram minus one or more terms. Other additions are proper names of people, countries and cities.

Table 1 - Strong n-grams from PWN

2-grams	58125
3-grams	9400
4-grams	1726
5-grams	302
6-grams	76
7-grams	22
8-grams	21

The proposed algorithm is a basic two-step approach.

The first step is statistical ranking based on the GC lexicon, bi and trigrams and the PWN. It uses a modified Levenshtein distance (LD) [12] algorithm and the weighed frequency of terms in the GC (see the equation below). We also split the existing candidates into smaller words[2] and create concatenation variations.

The second step takes the first ranked alternative and compares its score to the original query score. If the difference does not exceed a certain threshold (see below), it returns the original query as the first spelling option. The scores of the spelling variants are then normalized with the first result being assigned a higher probability for better precision. Thus, we obtain an overall precision of about 95% on the Speller Challenge TREC dataset (4 percent points over the baseline).

The problem with the classical LD is that it does not make distinctions between candidates that have nothing to do with the original query and those who are corrections of common mistakes. Producing good suggestions requires that all the words in the query be processed and variations created to accommodate both mistypes and inappropriate use of terms. Classifying alternatives that are not misspellings is very difficult. The difference between out-of-vocabulary terms and typos is insignificant in many cases.

The modified LD that was used to create term variations takes into consideration common mistakes such as misspelling of two consecutive letters, transposition etc. (e.g. *"Carribbean"*→ *"Caribbean"*, *"porgramming"*→*"programming"*). Word length is a factor that indicates a higher probability of mistake on longer terms. We do not apply this method on words with less than 4 letters. Instead we use a list of common mistakes of these words. The resulting query alternatives are ranked using unigrams, bigrams and trigrams (see the next equation). The score is multiplied by a factor dependent on the above-mentioned LD between the original query and the alternative query:

$$S(q1) = \left(\begin{array}{c} \alpha * \sum f(w_i) + \beta * \sum f(w_i, w_{i+1}) + \\ \gamma * \sum f(w_1, w_{i+1}, w_{i+2}) \end{array} \right) * F_{(q,q1)}$$

where:

w_i is the i^{th} term of the query $q1$;

$\alpha = 0.0001, \beta = 0.005, \gamma = 0.9949$ are n-gram weights tuned to Speller Challenge TREC dataset;

$f_{(t1,\dots t3)}$ are frequency counts;

$F_{(q,q1)} = 10^d$ is a factor dependent on Levenshtein distance d between the original query q and the spelling variant $q1$.

When calculating the score for an alternative query, we discard extremely high frequency words. They are not used in unigrams and bigrams and cannot start or end a trigram. For instance, when referring to "*best of luck"*, the score will not be influenced by the unigram *"of"* or the bigrams "*best of"* and "*of luck*". On the other hand, *"of"* can be in a middle of a trigram, so the score will be given by the following elements: unigram *"best"*, unigram *"luck"* and trigram "*best of luck"*.

The GC is outdated (2006) and this makes it unreliable when computing the score for choosing the best spelling suggestion. Due to this inconvenience and based on the statistical fact that about 85% of the queries in the Speller Challenge TREC dataset were correct (not misspelled at all), we decide not to replace the original query if its probability is 50% less than that of the best alternative. Also, in situations where the highest ranked alternative is the same as the original query but containing different inflected variants of the original query terms (plural, superlative, comparative), the original query will be preferred.

3. THE SECOND ALGORITHM

The second algorithm is very simple and has three main steps: (i) compacting the given query, (ii) select correct suggestions for each word in the obtained query, according to a database of unigram frequencies, and (iii) choose the combination of suggestions which has the maximum probability according to the available language model. The main resources employed by the speller are:

- a unigram table containing Google Web 1T 5-gram Corpus (GC) unigrams with frequencies above 10,000, Princeton WordNet 3.0 (PWN) word-forms, various named entities: names of persons, cities, minerals, cyclones and dinosaurs extracted from public sites like *Wikipedia*[3] or *NOAA*[4] and various bigrams obtained by splitting all unigrams in the GC, into two words that make sense. This table is kept in memory for fast operations;
- GC bigrams and trigrams kept in a fast *Berkeley* DB interrogated through a REST Web Service.

In the first step, the speller compacts the queries, which means that terms in the initial queries are glued if they can form words that make sense (according to the unigram table) with a reasonable high frequency (above 1,000,000). The idea is to have the same consistent approach in correcting a query, weather it is compact or it has spaces inserted. Here, "*health care*" becomes "*healthcare*" and "*web service*" becomes "*webservice*".

While iterating from the leftmost to the rightmost term in the query, two things happen: first, the consecutive mono-character terms are glued ("*n o a a*" becomes "*noaa*" and "*u s*" becomes "*us*"); second, two terms t_1 and t_2 are glued into t_1t_2 if the following constraints are simultaneously satisfied: (i) t_1 is not a term composed of other mono-character terms, (ii) t_2 is not a mono-character term, (iii) t_1 and t_2 are not stop words unless they

[2] It is highly important not to split terms needlessly. For instance, instead of breaking *"improving"* into *"imp"* (mythological being), *"rov"* (remotely operated vehicle) and *"ing"* (bank) or "*imp"* and "*roving"* (a long and narrow bundle of fibers), we leave it in place based on the frequency (17,855,962 for "*improving*") and the length of the parts.

[3] http://en.wikipedia.org/wiki/Main_Page

[4] http://www.nhc.noaa.gov/aboutnames.shtml

form certain words like *byproduct* or *online* and (iv) t_1t_2 can be found in the unigram table with a frequency larger than 1,000,000.

In the second step, selecting the best correct suggestions for each term in the query is done by employing a complex lexical similarity measure (between words) which is mainly based on the Levenshtein Distance (LD), length difference in characters, unigram frequencies and the longest common substring:

$$score(t,s) = \frac{\alpha}{(LD(t,s)+1)^k * (LD(t_m,s_m)+1)^\beta}$$

where t is a query term, s is a possible suggestion for t and k is a constant (0.66), while α and β are variables depending on the above parameters. t_m and s_m are modified versions of the t and s obtained by getting rid of duplicate letters and by replacing some character sequences in the following way: *space* is replaced by *empty string*, *ph* by *f*, *y* by *i*, *hn* by *n*, *ha* by *a*. Notice that adding 1 to the LD score values ensures a non-zero denominator. For example, if t = *corect* and s = *correct*, then $\alpha = 1$, $\beta = 0.2$, and *score*(t,s) = 0.87. If t = *corect* and s = *corect*, $\alpha = 0.53$, $\beta = 0.33$ and *score*(t,s) = 0.56.

Using this measure, the algorithm computes scores against all the unigrams in our table, using multi-threads for an improved speed. Most of the variants are discarded from the start based on length differences, and so, this process is fast, a complete search for a query term being completed in approximately half of a second. In the end, the best top ten suggestions are kept for every term.

In step three, the algorithm validates and scores the different combinations of suggestions according to the GC language model. This is done by using a modified version of the *Viterbi* algorithm [15] that allows for multiple results output. For smoothing, the speller uses an interpolation function that would allow for all the parts of an n-gram to contribute to the final score:

S(*abc*) = if(constrains are satisfied) λ*0.5 + 0.25*S(*ab*) + 0.25*S(*bc*)

S(*ab*) = if(constrains are satisfied) λ*0.5 + 0.25*S(*a*) + 0.25*S(*b*)[5]

These equations show that this model gives high importance to the terms (which can now be considered words) in the middle of the queries. In order to take into account an n-gram, the following constrains must be satisfied:

- at least one of the words must be the best suggested unigram variant;
- the words in the n-gram must not be part of a stop-words list;
- the log-likelihood like score (LL) must be above a threshold;

This approach successfully brings on the first place "*capitol hill massacre*" for the input query "*capital hill massacre*", "*commonwealth of virginia*" for "*commonwealth of virgina*" or "*jedi knight*" for "*jedy night*".

In the end, different alternatives of the best returned spelling variant are added.. Spaces are inserted between words containing only consonants (as they are probably abbreviations), compressed words like "*us*" turn into "*u s*", indefinite articles are added in front of long words, as well as the genitive particle "*s*" and plurals like "*veterans*" turn into "*veteran s*" to also allow for the genitive.

Finally the speller outputs the most probable 50 suggestions plus the original query if is not already there. Using a so called *decay function* similar to the one used by the first algorithm, it assigns more than 99% of the probability mass to the best suggestion, the remaining mass being split between the others. The reason for using the decay function relies on the fact that the speller had a very high precision on the TREC training data (around 95%). The large number of suggestions allows it to aim for a high recall too.

4. THE THIRD ALGORITHM

The third algorithm divides the query alteration process into *error detection* and *error correction* based on the fact the second process necessarily depends on the first process but the first process may well stand on its own.

The error detection mechanism is based on a classifier that given an input term chooses between two classes: "*leave it*" (the term is considered to be correctly spelled and consequently should be left unchanged) and "*change it*" (the term is not recognized and spelling variants should be generated). We observed that, in accordance with the Speller Challenge TREC Data – a gold standard collection of correct and misspelled queries along with the correct variants, detecting when a query term is correctly or incorrectly spelled depended on a variety of features such as:

- a rare, frequent or very frequent threshold; word frequencies were extracted from unigrams of Google Web 1T 5-gram Corpus (2006 release, version 1). We observed that while the majority of rare words needed to be changed and the majority of very frequent words needed to be left alone, a part of frequent words needed to be changed (e.g. "capital" → "*capitol*" from a query like "*capital hill history*");
- English morphology clues: inflection (plural for nouns, past tense for verbs, etc), words with a repeating consonant (e.g. "*carribean*" → "*caribbean*"), proper nouns (e.g. "*acey byrd*" → "*acey byrd*" not "*ace bird*"), word is a functional word (preposition, conjunction, pronoun, determiner, etc.), word is not functional (noun, adjective, adverb, verb).

In order to decide whether or not to propose spelling variants to a given query term, we employed a Maximum Entropy (ME) [1] classifier that will make this decision based on our features-encoded knowledge of that term. Specifically, we have used the following feature functions: *isRare*, *isFrequent*, *isVeryFrequent*, *isInflected*, *isProperNoun*, *isFunctionWord*, *isNoun*, *isVerb*, *isAdjective*, *isAdverb*, *hasDoubleConsonant*.

We have trained the ME classifier on a collection of spellchecking data gathered from multiple sources[6] among which the *aspell* test data and the *Wikipedia* collection of editing spelling errors. The best accuracy of this classifier is 90.74% for the "change it" class, 93.04% for the "leave it" class and 91.89% overall.

The error correction algorithm seeks to optimally and simultaneously do two different things: select the most probable spelling variant for each query term while, at the same time, selecting the most probable spelling alternative to the given query. In order to cope with split and joined terms, we first recursively generate all possible split and join variants for the given input query such that each query term is joined with the neighboring terms and/or split in at most three parts.

For each query term that the classifier deems "change it", we generate the most 30 frequent spelling variants that are at most two string edit operations away from the misspelled term. The frequencies are extracted from the GC and the threshold was experimentally set to 5×10^4 (if the frequency of the alternative is below this threshold, the alternative is discarded). If it is not

[5] We write S and not P since these are not probabilities.

[6] http://www.dcs.bbk.ac.uk/~roger/corpora.html

changeable, we only generate the most frequent 15 string edit distance on variants more frequent than 5×10^4 that form a frequent (threshold of 10^2) bigram or trigram with surrounding words (the frequencies of bigrams and trigrams are also determined from the GC). We have added this generation step in order to compensate for the approx. 7% of classifier failures in the case of "leave it" class and also because, for the current version of the speller, we were not able to program the classifier to take the context of the input word into account.

For each query term we also generate variants that are semantically and/or morphologically related to the term if and only if the term is deemed non-changeable by the Maximum Entropy classifier. Thus, we want to generate plurals for nouns or present participles for verbs since these forms are often used when formulating queries. Also, we would like to generate different correct alternative spellings such as "*tumor*" and "*tumour*". To do this, we used Princeton WordNet 3.0 (PWN) and extracted an associative list of all literals that are two string edit operations away and belong to semantically related synsets (*synonyms*, *hypernyms*, *meronyms*, etc.).

The algorithm for searching the best spelling alternative for the given input query uses the Viterbi algorithm [15] with a beam of 5 (it remembers the most probable 5 solutions at each point in the search path). We output the first 20 spelling alternatives plus the original query (21 spelling alternatives).

The transition probabilities are given by the inverse of the perplexity of the alternative query up to that point. The language model that was used by the Viterbi procedure is also based on the GC. It's a trigram language model that uses the Jelinek-Mercer interpolation [3] with fixed, experimentally determined weights: 0.8 for trigram probability, 0.19 for bigram probability and 0.01 for the unigram probability.

In order to be able to rank relevant queries higher (queries that are more likely to produce results) we have trained the MITLM [8] language modeling toolkit (with ModKN smoothing) over a corpus consisting of:

- the "AOL query log" corpus [2] aiming at obtaining low perplexities (and top ranking) for queries that have been asked by other users. This is our way of enforcing both correct English syntax (or, at least, OK-for-searching English syntax) and similarity to users' interests;
- a collection of PWN expressions with at least 2 words per expression each of them repeated 100 times in order to give it statistical significance;
- the correct part of the Speller Challenge TREC Data;
- an existing selection of 4,000 English Wikipedia documents aiming at enriching the language model with named entities and terminological data.

5. THE COMBINER

The combining method we employed is a simple voting system. Since the spellers assign probabilities to the spelling variants they return, their values are comparable. The experiments performed on the training data revealed that the three algorithms frequently suggest differently ranked variants and usually make different mistakes. We considered that the conditions for combining the algorithm were met.

We decided that each alternative spelling should receive a score computed as the sum of the probabilities given by each of the spellers. For this to happen, we switched off (for all spellers) the decay function that assigns over 99% of the probability mass to the first top suggestion. The next step was to normalize the scores and reorder the entire list according to the new values. If the obtained top suggestion was different from the original one and the difference between its score and that assigned to the original was lower than 0.1, we gave credit to the original and swap the suggestions. The table below shows the performances of the spellers and the combiner on both Bing Test Data and TREC training data.

Table 2: Official results for the three spellers and the combiner

Speller	F1 on Bing Test Data	F1 on TREC Training Data
1st algorithm *NemoRitor-WrapperSpeller* author: Tiberiu Boroş	76.07%	~96%
2nd algorithm *TNGS-tgr* author: Dan Ştefănescu	71.19%	~95%
3rd algorithm *DR. Speller-langmodrank* author: Radu Ion	65.66%	~94%
Combiner **4th place at Microsoft Speller Challenge**	**79.50%**	**~97%**

6. CONCLUSIONS

After the Microsoft Speller Challenge ended, we had a chance to study the speller's httpd logs and collect the actual Bing Test Data which comprises of 1,325 queries. Comparing these queries with those from Speller Challenge TREC Data, we noticed how much more "noisy" the Bing Test Data is by comparison: there are a lot of queries composed of "cryptic" terms such as chemicals, proper names, names of web sites, units of measure, different acronyms, hard-to-split terms, etc. most of which our spellers tried to correct as they did not recognize them. This is one of the main reasons explaining the large difference between the performances obtained on the Training Data and those obtained on the Test Data (about 18%). Other reasons might be:

- test data has a completely different ratio of short vs. long queries than training data;
- test data has a completely different ratio of correct vs. incorrect queries than training data;

Looking at the results, it seems obvious that a system returning the given queries unchanged (as the baseline) would have surely been one of the top winners.

One should also notice that the Microsoft recall measure favors the systems returning a large number of spelling alternatives in the hope that all the correct ones would be covered. This could be speculated by the some systems that could return thousands of spelling alternatives, luckily obtaining a recall close to 100%. We consider such a high number of alternatives as being utterly irrelevant to practical purposes of spell checking.

7. ACKNOWLEDGMENTS

This work has been supported by the Romanian Ministry of Education and Research through the STAR project (no. 742/19.01.2009).

8. REFERENCES

[1] Berger, A., Della Pietra, S., Della Pietra, V. 1996. A maximum entropy approach to natural language processing. *Journal of Computational Linguistics* 22(1), 39—71.

[2] Brenes, D.J., and Gayo-Avello, D. 2009. Stratified analysis of AOL query log. Journal of Information Sciences 179, 1844—1858.

[3] Chen, S.F., and Goodman, J. 1999. An empirical study of smoothing techniques for language modeling. *Journal of Computer Speech and Language* 13, 359—394.

[4] Cucerzan, S., and Brill, E. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP*.

[5] Dietterich, T.G. 1998. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Journal of Neural Computation* 10(7), 1895—1924.

[6] Duan, H., and Hsu, B.-J. 2011. Online Spelling Correction for Query Completion. In Proceedings of the International World Wide Web Conference, WWW 2011, March 28–April 1, Hyderabad, India.

[7] Fellbaum, C. 1998. WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA.

[8] Hsu, B.-J., and Glass, J. 2008. Iterative Language Model Estimation: Efficient Data Structure & Algorithms. In *Proceedings of 9th Annual Conference of the International Speech Communication Association (Interspeech)*, Brisbane, Australia, September 22-26.

[9] http://www.internetworldstats.com/stats.htm, "Internet usage statistics, the internet big picture, world internet users and population stats", 23.06.2011

[10] http://xldb.di.fc.ul.pt/xldb/publications/spellcheck.pdf, Spelling Correction for Search Engine Queries, Bruno Martins and Mário J. Silva

[11] Jean-Baptiste, M., Shen, Y.K., Aiden, A.P., Veres, A., Gray, M.K. 2010. The Google Books Team. Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, Steven Pinker, Martin A. Nowak,Erez Lieberman Aide, Quantitative Analysis of Culture Using Millions of Digitized Books.

[12] Levenshtein, V.I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10, 707.

[13] Naseem, T., and Hussain, S. 2007. Spelling Error Trends in Urdu. Evaluation 41, Issue: 2, 117—128.

[14] Tufiş, D., Ion, R., Ceauşu, A., and Ştefănescu, D. 2006. Improved Lexical Alignment by Combining Multiple Reified Alignments. In *Proceedings of the 11th EACL 2006*, pp. 153–160, Trento, Italy.

[15] Viterbi, A.J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Journal of IEEE Transactions on Information Theory* 13, 260–269.

Spelling Generation based on Edit Distance

Yoh Okuno
Yahoo Japan Corporation
9-7-1 Akasaka, Minato-ku, Tokyo, Japan
nokuno@nokuno.jp

ABSTRACT

We propose a spelling generation algorithm based on edit distances. Our idea is to get high recall without reducing its expected precision. Whereas general precision has trade-off relation with general recall, there is theoretically no trade-off between the expected precision and recall, the evaluation measures of Microsoft Speller Challenge. We used a bit tricky way: the output candidates with very small or zero probability hurts little expected precision. The candidates are generated based on edit distance from the input query, so that they can increase the recall. In this case, edit distance is the number of edit operations to a character: insert, delete, replace, and swap. Output size is limited to some bytes, because the bottleneck of latency in this strategy is the network.

1. INTRODUCTION

Microsoft Speller Challenge is the first contest of spelling correction for search engine query. We had to read thoroughly the rule at first. We noticed that there are some fuzziness around rules that allows utilizing external API or even dataset. Expected F1 (EF1) is adopted as evaluation measure[1]. It has two components: one is Expedted Precision (EP) and anotheris Expected Recall (ER). EP is defined as below:

$$EP = \frac{1}{|Q|} \sum_{q \in Q} \sum_{c \in C(q)} I_P(c, q) P(c|q) \tag{1}$$

Where q is an input query in test dataset Q, c is a query in correct set $C(q)$ of query q, $I_P(c, q)$ is the indicator function whether $c = q$ or not. ER is defined as below:

[1]http://spellerchallenge.com/Rules.aspx

$$ER = \frac{1}{|Q|} \sum_{q \in Q} \sum_{a \in S(q)} I_R(C(q), a) / |S(q)| \tag{2}$$

Where a is an answer in speller output set $S(a)$, $I_R(C(q), a)$ is the indicator function whether a is in $C(q)$. Finally, $EF1$ is defined as a combination of EP and ER.

$$EF1 = \frac{2}{\frac{1}{EP} + \frac{1}{ER}} \tag{3}$$

Note that ER contains no information about probability, so it should be simply called as recall. Now we can set probability to enhance EP without bothering ER.

2. TRIALS

At first, we submitted our speller to the contest's official evaluation system. But soon we noticed that the system doesn't show the actual component of EF1: EP and ER. We developed our original evaluation script, allowing us to know which of EP or ER is bottleneck. Then we tried some spellers as below and Table 1 shows the results of trials.

- Overfit speller: This speller achieves top in current leader board (May 26, 2011) with 1.0 Expected F1 and 275 Latency. Of course, this works well on only TREC dataset, and doesn't work on final Bing dataset, because output candidates are exactly same to the annotation of TREC dataset.
- Noop speller: This speller actually does nothing and return exactly the same one as the input query. This speller achieved 0.91 EF1, 0.94 EP and 0.87 ER. This result implies that recall might be bottleneck.
- Bing speller: We tried the bing speller API[2]. Although this was expected to achieve high score, its score was actually 0.91 EF1: lower than Noop speller. This result reminded us that the annotation policy of TREC dataset is quite differ from commercial search engine query speller. TREC anotation policy prefers variousness of spellings, rather than one exact spelling.

[2]http://www.bing.com/developers/

Table 1: Result of Trials

Speller	EP	ER	EF1
Overfit speller	1.0	1.0	1.0
Noop speller	0.9472	0.8773	0.9109
Bing speller	0.9463	0.8765	0.9101
Bing+Noop speller	0.9467	0.8788	0.9115
Final	0.9472	0.9018	0.9239

- Bing+Noop speller: This speller is a combination of Noop speller and Bing speller. The output is an input query itself and a query that was corrected by Bing speller. Each candidate have the same probability 0.5.
- Edit1 speller: Our first edit distance based speller with one edit operation.
- Edit2 speller: Our second edit distance based speller allowing two edit operation.
- Edit2+anagram speller: Two edit operation or one swap (anagram) operation is allowed in this speller.
- Time limit speller: This speller limits the CPU time in spelling generation.
- Line limit speller: This speller limits the number of output lines.
- Byte limit speller: This speller limits the bytes of output size.

3. FINAL SUBMISSION

We adopted edit-distance-based approach to generate speller candidates. Edit distance means the number of edit operations as below:

- insert: insert a character at any place in the input query. Example: microsoft -> micronsoft
- delete: delete a character in the input query. Example: microsoft -> micosoft
- replace: replace a character in the input query with different character. Example: microsoft -> microzoft
- swap: swap two character in the input query between different position. Example: microsoft -> microzoft

We expanded edit operation with additional swap operation.

Our final submission adopts following strategy:

1. Output input query itself with probability 1.
2. Output candidates which have edit distance 1 with input query.
3. Output candidates like an anagram of input query, swapping two characters.
4. Output candidates which have edit distance 2 with input query.
5. If total output size as bytes was over some constant, exit program.

All but first candidate can have very low probability or even zero probability. They might raise recall, without reducing expected precision. The EP for TREC dataset is 0.9472, equal to Noop speller when those probabilities are zero. Our speller might generate exactly same candidate, because two edit operations might restore the original query or swapping same characters.

We developed this alogrithm as PHP script at first, but the latency was so large. Then we switched to C++ CGI scripts to improve its latency. Note that we don't need any data management. After some parameter tuning, we decided to adopt 50MB as the value of size limit. Our speller achieved 0.9239 EF1 and 43912 latency on TREC dataset.

4. CONCLUSION

We developed very simple spelling generation algorithm based on edit distance. This approach achieves high recall if latency is not limited. We apologize that our speller might have been caused large responses, but we didn't intend to overflow the speller evaluation system of Microsoft. Although this approach is a bit tricky, it does not cheat the rule or evaluation measures.

We considered standard noisy channel model and utilization of external API, but these might drop expected precision. Our simple approach depends on no dataset or external APIs, leading to its concise implementation. Although we have a trie implementation [3] which supports fuzzy search, it needs sufficient query logs to train language model.

Thank you for reading.

[3]https://github.com/nokuno/stakk

A REST-based Online English Spelling Checker "Pythia"

Peter Nalyvayko
Analytical Graphics, Inc.
Exton, PA 19341, USA
(347) 387-3074
petervn1@yahoo.com

ABSTRACT
The paper describes the implementation of an online English spell checker that was developed for the Microsoft Spelling Challenge Contest. The spelling correction program proposes a list of plausible corrections ordered by probability. The probabilities are computed using the noisy channel model [6] trained using a list of common misspellings. The misspelling detection phase involves generating a key for misspelled word and locating words in the dictionary whose keys collate closely with the key of the misspelled word [4]. The correction phase involves choosing plausible candidates from the list of candidates selected during the detection phase. The algorithm also tries to correct spelling errors caused by word boundary infractions. The modified Levenshtein distance is used to compute how similar the proposed correction and the misspelled words are. In addition to the above, the algorithm uses Princeton WordNet synsets database to look up words missing from the similarity dictionaries and for alternative spellings' lookup. Finally, the algorithm relies on Microsoft N-Gram service to evaluate the posterior and joint probabilities used to rank and choose best spelling correction candidates.

Categories and Subject Descriptors
D.3.3 English Spelling Correction

General Terms
Algorithms, Experimentation

Keywords
Automatic Spelling Correction, Similarity Key, N-grams, WordNet, Lucene, Apache, LingPipe, Tomcat, NLP.

1. INTRODUCTION[1]
The spell checker is a REST-based Web Service (further referred to as the program) implemented as a Java Servlet which runs in the Tomcat environment behind Apache Web Server. The APIs used include LingPipe [12], Apache Lucene [13] and MIT WordNet API [11]. The following is a sample of the output produced by the program:

Query	Output
Strontium 90	Strontium 90 1.000000
californiastate	california state 1.00000
satelite photos earth	satellite photos earth 0.9601
coch retirement home	coach retirement home 0.71114322 couch retirement home 0.16817989 coch retirement home 0.12067688

The output produced by the program may contain multiple spelling suggestions each one corresponding to a plausible spelling correction candidate chosen and ranked high enough to be included into the result set.

The corrections are listed in a non-increasing order of their probability. The posterior probability of the correction is computed and returned along with the spelling correction candidate.

The overarching criterion at the core of the design of the spelling correction program was to avoid as much as possible using any language specific information to remain as flexible as possible.

In this paper I describe an online spell checking service which attempts to address spell checking challenges by combining several different strategies to solve various spelling problems that arise due to a variety of possible types of misspellings.

2. Three Problems of the Spelling Correction
Kukich in her survey of the existing spelling correction techniques [1] identified three distinctive problems: 1) **non-word error detection**; 2) **isolated non-word error correction**; 3) **real-word context dependent word correction**. While the first and the second problems have been extensively studied and a number of various techniques have been developed (dictionary lookup, n-grams), the third problem poses a significant challenge [1].

3. Why spelling correction is difficult
The three problems of the auto spelling text correction become even more exaggerated and complex when trying to auto-correct short input queries that carry almost no or very little contextual information, frequently do not follow any proper grammar and potentially contain a large number of acronyms or proper nouns. The input queries can also belong to a wide range of different topic categories, thus presenting a significant challenge to the dictionary-based lookup techniques that expect the dictionaries to be topic specific.

4. Training Data and Similarity Key Dictionaries

The program was trained using publicly available corpora from the Gutenberg online project which was used to generate spelling correction dictionaries. The program uses dictionaries which entries are ordered alphabetically based on a similarity key generated for each dictionary entry to choose a sub-set of plausible spelling correction candidates. I'll discuss similarity key concept later in greater details. The original concept of using similarity key was first described by Joseph J. Pollock and Antonio Zamora in [4]. Along with each entry in the dictionary is also stored a frequency of the word associated with the entry. The frequencies are used to evaluate the prior probabilities of the spelling correction.

The process of training the spell checker consists of generating two dictionaries which entries are ordered based on the similarity and skeleton keys [4]. The spell checker uses both dictionaries during non-word spelling correction phase to look up and select a range of plausible candidate corrections.

To further strengthen the detection and correction performance, the program also uses the Princeton WordNet Synset databases to look up rare words and find alternative spellings. WordNet database is publicly available for download at http://wordnet.princeton.edu/wordnet/.

The noisy channel model was trained using publicly available list of common misspellings. The misspellings were automatically parsed to generate four confusion matrices used to evaluate the likelihood of a misspelling [6].

5. Correcting Spelling Errors

5.1 Query Language Detection

In order to avoid spell checking the queries written in language other than the language that was used to train the spell checker (English in this case), the program uses a simple vector-space based classifier which, given an input query, returns a language class. Queries classified as non-English are not spell checked. The classifier maintains two sets of test vectors with the same dimensionality: $\mathbf{w_e}$ for English and $\mathbf{w_s}$ for Spanish. The classifier maps the input query $Q = \{w_1, w_2, \ldots, w_k\}$ into a binary vector using the following rule:

$$\boldsymbol{x} = \{x_1, x_2 \ \ldots, x_n\}, \text{ where } x_i = \begin{cases} 1, & w_{e_i} \in Q \\ 0, & otherwise \end{cases}$$

Where n is the number of elements in the test vector $\mathbf{w_e}$, and k is the number of tokens in the input query. The program computes the angular distance between the input vector and the test vectors:

$$\theta_e = \cos^{-1} \frac{x \cdot w_e}{\|x\| \|w_e\|}$$

$x \cdot w_e$ denotes a dot product of the input and the test vectors. $\|x\| \|w_e\|$ is a product of the lengths of the input vector and the test vector. The equation is used to evaluate angular distances between the input vector and the vectors representing Spanish and English. The language of the input query is chosen to be English if the angular distance between the input vector and the test vector representing English is smaller than the angular distance between the input vector and the vector representing Spanish.

5.2 Isolated Non-word Detection and Correction

5.2.1 Isolated non-word error detection

The first phase of the spelling correction program is to identify potentially misspelled words and suggest plausible spelling correction candidates. The program relies on the similarity key dictionaries sorted by the omission and skeleton keys [4] to look up a term and return a list of possible corrections.

Few words need to be said about the omission and skeleton keys. The skeleton key is constructed by leaving the first letter of the word in its place followed by unique consonants in order of occurrence and then followed by the unique vowels in the order of occurrence.

The omission key is constructed as follows: the unique consonants come in the reverse order of their omission (hence the name) frequency order, followed by the unique vowels in the original order.

The consonant omission frequency order is a result of the work done by Joseph J. Pollock and Antonio Zamora [4] and is as follows:

RSTNLCHDPGMFBYWVZXQKJ

which basically means that "R" is omitted more often than "S", and so forth.

So, for example, the misspelling "carribean" and its correct form "caribbean" will have the following skeleton and omission keys:

Word	Skeleton Key	Omission Key
carribean	CRBNAIE	BCNRAIE
caribbean	CRBNAIE	BCNRAIE

As it is apparent from the above table, both misspelled and correct words have the same skeleton and omission keys. This is not always a case, as in the following example:

Word	Skeleton Key	Omission Key
adust	ADSTU	DTSAU
adjust	ADJSTU	JDTSAU
adult	ADLTU	DLTAU

Both "adjust" and "adult" are plausible corrections, however a set of plausible candidates returned by the dictionary lookup using the omission key will unlikely contain the "adjust". Likewise, the results returned by the dictionary lookup using the skeleton key are also unlikely to contain the desired correct word, unless the search window has been made very large.

Based on the experimental findings, to overcome some of the shortcomings associated with the similarity key based dictionary lookups, the spelling error detection used by the program includes a trie-based lookup that utilizes a trie structure and the minimum edit distance algorithm to search for all possible corrections in a trie tree [10], whenever the similarity key dictionary lookup was unable to detect any plausible candidate corrections.

5.2.2 Isolated non-word error correction

Each plausible correction candidate discovered during the detection phase is compared against the misspelling using the Levenshtein distance with the following edit costs for each of four possible corrections:

Deletion	1.2
Insertion	1.2
Substitution	2.0
Transposition	1.0

In addition, the insertions that result in the same character being inserted more than once are discounted and are 1/5 of the regular insertion cost. In other words, the distance between the misspelling "cost" and "cooost" is only 0.48.

The candidate corrections with the distance from the misspelled word exceeding some pre-defined threshold are discarded. The rest of the plausible candidate corrections are added to the list of plausible candidate corrections.

5.3 Scoring

During the isolated non-word error correction phase the program evaluates posterior probability of each plausible candidate correction using the Bayes' equation:

$$P(w|c) = \frac{P(c|w)P(w)}{P(c)}$$

Where $P(w)$ is a prior probability of the candidate; $P(c|w)$ is a likelihood that the word *w* was mistyped, and $P(c)$ is a normalizing constant.

The prior probability $P(w)$ of the potential spelling correction is the word frequency counter divided by the total number of the tokens in the training corpora.

The likelihood $P(c|w)$ of a misspelling given a correct word is estimated based on the confusion matrices compiled during the training phase [6]:

$$P(c|w) = \begin{cases} \frac{del[w_{i-1}, w_i]}{chars[w_{i-1}, w_i]} \; if \; deletion \\ \frac{add[w_{i-1}, c_i]}{chars[w_{i-1}]} \; if \; addition \\ \frac{subs[c_i, w_i]}{chars[w_i]} \; if \; substitution \end{cases}$$

Where $del[w_{i-1}, w_i]$ is the number of times the characters $w_{i-1}w_i$ were typed as w_{i-1}; $add[w_{i-1}, c_i]$ is the number of times the character w_{i-1} was typed as $w_{i-1}c_i$; $subs[c_i, w_i]$ is the number of times w_i as typed as c_i; and $revs[c_i, w_i]$ is the number of times that w_iw_{i+1} was typed as $w_{i+1}w_i$. The $chars[w_{i-1,}w_i]$ and chars$[w_i]$ is the number of times $w_{i-1,}w_i$ or w_{i-1} appeared in the training corpora.

The product of the posterior probabilities of each misspelling in the query string is the total posterior probability of the corrected sentence given the input query.

5.4 Word boundary infraction misspellings

When parsing the query, the assumption is that the word boundaries are defined by the white space such as blanks, tabs, etc. However, based on the test data set there is a significant number of spelling errors caused by either split or run-on words. Kukich [1] found that a total of 15% of all non-word errors were due to either split or run-on words. The program attempts to correct the word boundary infractions and proposes a set of spelling candidates which are then ranked based on the joint probability of the correction times distance between the misspelling and the candidate correction. Microsoft N-gram service is used to evaluate the correction candidates against the online n-gram corpus.

5.4.1 Spelling errors resulting in split words

To simplify the correction of the spelling errors caused by insertion of a blank space, the algorithm assumes that the split is limited to just two adjacent words. The split word error correction consists of two phases. During the first phase the input sentence is analyzed for any potential split word errors. The algorithm iteratively removes blank spaces between adjacent words and adds the newly formed words to a list of potential correction candidates.

Once all candidates are discovered, the algorithm moves to a next phase during which the conditional probability of each candidate is evaluated. The candidates are then sorted in a non-descending order based on their probabilities. The candidate with the highest posterior probability is chosen.

5.4.2 Spelling errors caused by run-on words

The program explicitly handles the spelling errors caused by the run-on words. First, a sub-set of all possible run-on word corrections is selected. To avoid combinatorial explosion, the sub-set is limited to the combinations consisting of no more than a pre-determined maximum number of blank space deletions. The total number of possible corrections is:

$$f(n,k) \quad = \sum_{i=1}^{k} \frac{(n-1)!}{(n-i)!\,(i-1)!}$$

Where *n* is a length of the incorrectly spelled word, *k* is a maximum number of the blank space deletions, and *f(n,k)* is a function that returns a number of possible candidate corrections. For example, given a misspelled word "*californiastate*", and *k=4* (the maximum number of deleted blank spaces), the sub-set of possible permutations will have the total number of elements equal to:

$$f(n=15, k=4) \quad = \sum_{i=1}^{4} \frac{(15-1)!}{(15-i)!\,(i-1)!}$$

Evaluating the above equation gives us *f(n,k) = 470*, in other words there are 470 distinct spelling candidates that will be generated by the run-on word correction algorithm. The table below lists some of the candidates generated by the run-on correction algorithm:

"c aliforniastate"
"ca liforniastate"
"ca li forniastate"
"ca li fornia state"
"c ali fornias tate"

To enumerate all possible run-on permutations, the program uses a memorization optimization technique from dynamic programming to improve the running time by decreasing the computational complexity in time by "memorizing" the results of evaluation of sub-sequences.

5.5 Alternative Spellings

There is a category of spelling corrections that does not really require a correction. Rather, during the last phase the spelling correction program tries to suggest alternative spellings by looking up words in the WordNet database. WordNet offers rich, albeit limited to English, lexical network of meaningfully related words and concepts. The WordNet is a useful tool for natural language processing and computational linguistics [11].

For words that are present in the WordNet databases, the program extracts the synsets and analyzes the lemmas that have the same meaning. The program computes the minimum edit distance between the alternative spelling and the original input and discards the alternative spellings with the minimum edit distance cost exceeding some pre-defined threshold. The alternative spelling is then added to the list of plausible corrections. The probability of the alternative spelling is evaluated and ranked according to its conditional probability. For example, the program will detect that the word "center" has the alternative spelling "centre" and propose both as plausible spelling candidates.

5.6 Final Ranking

Once the error detection and spelling correction phases are done, the program generates all possible spelling correction combinations of the entire input sequence using the lists of plausible candidates generated by the previous phases on the algorithm.

Each of the generated sequences is re-evaluated using the Microsoft N-Gram service by computing its joint probability. Each sentence is then re-ranked using its computed joint probability and the scores computed during error correction phases. The combined ranking is used to sort the plausible corrections in a non-decreasing order. The corrections scored below some pre-defined threshold, are discarded. The remaining corrections are normalized so that the sum of the probabilities adds up to 1.

5.7 Evaluation

The accuracy of the spelling corrections was evaluated using the F-measure, the harmonic mean of precision and recall. The evaluation results below are based on the sub-set of the TREC dataset manually annotated by the human experts:

Test Set	F-Measure	Precision	Recall
207 queries	0.9319834906	0.9439781203	0.9202898550
409 queries	0.9162777685	0.9317057259	0.9013524264
1507 queries	0.9185730419	0.9352770727	0.9024552090

5.8 Known Issues

5.8.1 Input Query Language Classification

The technique used to determine the language can be improved to map the input query into a weighted vector where the element of the vector represents some metric associated with a word, for example its overlay frequency.

5.8.2 Training corpora

It is necessary to clean the training corpora to get rid of intentionally misspelled words some books tend to have (for example, "War and Peace" by Alexei Tolstoy contain a relatively large amount of foreign and intentionally distorted text).

5.8.3 Word boundary infractions

When correcting the run-on words, the program assumes the correct and misspelled sequences differ only in the number of blank spaces. In other words, no corrections, other than restoring deleted blank spaces, are required. This constraint was imposed as a trade-off between the precision and running time.

5.8.4 Real word error detection and correction

Real-word correction phase is not currently implemented, although there is an ongoing effort to add the real-word correction to the algorithm. There are several possible directions among which the text collocation [9] and neural nets [7].

5.8.5 Detection of Proper Nouns

The detection and correction of words that represent proper nouns still remains an elusive task due to ever growing body of proper nouns that not only include names of places and people, but also the names of the web sites, companies, etc. No dictionary lookup technique can possibly account for every variety out there. Therefore, the current version of the program uses a rule-based, pattern matching approach where it tries to detect whether the input query represents a name of the web site using known patterns.

6. ACKNOWLEDGMENTS

Many thanks to Microsoft Research Team for making the N-Gram service available to facilitate the Spelling Challenge Contest.

7. REFERENCES

[1] Kukich, K. 1992. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys.* Vol 24, No. 4 (Dec. 1994). Belkore, 445 South Street, Morristown, NJ 07962-1910.

[2] Jurafsky, D., Martin J., 2009. "Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics", 2nd edition, Prentice Hall.

[3] Damerau, F. 1963. A Technique for Computer Detection and Correction of Spelling Errors. In *Communications of the ACM, Vol. 7, No. 3 (Mar. 1964)* IBM Corporation, Yorktown Heights, NY

[4] Pollock J., Zamora A. 1984. Automatic Spelling Correction in Scientific and Scholarly Text. *Communications of the ACM*, Vol. 27, No. 4 (Apr. 1984).

[5] Mitton, R, 1996. *English Spelling and the Computer*. Harlow, Essex: Longman Group. Longman, London & New York.

[6] Kernigan, M., Church, K., Gale, W., 1990. A Spelling Correction Program Based on a Noisy Channel Model. AT&T Laboratories, 600 Mountain Ave., Murray Hill, NJ.

[7] Hodge V.J., Austin J., 2002. A comparison of a novel neural spell checker and standard spell checking algorithms. Pattern Recognition, 35 (11). pp. 2571-2580.

[8] Cavnar W.B., Trenkle J.M., 1994. N-Gram-based Text Categorization.

[9] Smadja F., 1993. Retrieving Collocations from Text: Xtract. Computational Linguistics, Volume 19, No.1.

[10] Fredkin, E. 1960. Trie memory. Commun ACM 3, 9, (Sept.), 490-500.

[11] WordNet, A Lexical Database for English. Princeton University. http://wordnet.princeton.edu/

[12] LingPipe Tool kit, http://alias-i.com/lingpipe/

[13] Apache Lucene Full-Text Search Engine Library, http://lucene.apache.org/