Building a Data-Driven Search Engine Spelling Corrector

Bas Niesink
University of Twente
P.O. Box 217, 7500 AE
Enschede, The Netherlands
s.w.r.niesink-1@student.utwente.nl

Stefanie Stevens
University of Twente
P.O. Box 217, 7500 AE
Enschede, The Netherlands
s.a.stevens@student.utwente.nl

ABSTRACT

Search engines perform better with faultless search queries. In this paper, data-driven domain-specific spelling correctors are designed and evaluated. The attained accuracy of the best model is 0.675.

Keywords

spelling correction; FAROO; Norvig; bigrams; language model

1. INTRODUCTION

People frequently make spelling errors and typing mistakes. For persons, these errors have little effect on reading comprehension. For computers, however, these errors are problematic, especially for search engines. Erroneous search queries result in bad results, since the misspelled words differ from the correct ones on websites. In addition, users do not always know how to spell names, for example. To still provide the results a user expects, the search engine must correct errors made by the user.

1.1 Problem statement

The University of Twente (*UT*) has its own search engine[1] to help find information on the UT network, among other sources. The search engine can be used, for example, to find information about courses, buildings, and student associations. Currently, the UT search engine does not contain a spelling corrector, while it could substantially improve the results' quality. Since most searches are specific to the UT, a spelling corrector could be devised which is trained to operate within this domain. Instead of dictionaries it could utilize websites related to the UT, or search queries entered in the system, to learn the terms that users of the engine may wish to find. This approach is called *data-driven*.

This research attempts to design and evaluate a data-driven spelling corrector for UT Search. New methods and existing ones will be combined to create this system. An overview of these methods is given in sections 2, 3, and 5.

Most techniques use *bigrams* to construct fast language models and to preserve the relation between adjacent words[4][16]. However, inverted bigrams may solve different errors than regular bigrams (section 2.2). Therefore, the performance of both bigram types will be measured in this research.

According to BRON, people do not notice it when search engines correct errors within 200 milliseconds. Thus, the correction speed of the devised systems will be compared to that.

[1]http://search.utwente.nl/

1.2 Research questions

The aim of this investigation can be summarized with the following research questions:

- RQ1 What method yields the highest spelling correction accuracy?
- RQ2 Does this method handle corrections within 200 milliseconds?
- RQ3 Does the addition of inverted bigrams lead to a higher accuracy?

2. BACKGROUND

2.1 Tokenization

Tokenization is the conversion of sentences into separate words or tokens, often by using whitespace or special characters as delimiters. Tokenization does have downsides, for instance, when `American-football players` is split into `American`, `football`, and `players`, the relation to the sport *American-football* is lost.

2.2 N-grams and bigrams

To preserve the relation between adjacent words *n-grams* can be used. These consist of tuples of n adjacent tokens. The string `American-football players` is split into `(American, football), (football, players)` when creating 2-grams after tokenization. Thus, holding information about the sport *American football* and the fact that the players are *football players*. N-grams of length *2* are called *bigrams*.

A variation of the bigram is the *inverted* bigram, which is a bigram with swapped tokens. Inverted bigrams may be useful to, for instance, allow searches on *Twente University* when only *University Twente* was present in the training set.

2.3 Language model

Language models are utilized to calculate the probability of a token, or sequence of tokens occurring in a language. This can be a real language, such as English, or a domain-specific one such as the terms related to the UT.

A simple language model is described by Chen and Goodman and uses the frequency of tokens and bigrams to estimate the likelihood of a sentence [5]. Additive smoothing is applied for unknown terms.

If bigrams were not used in the model, instead of the most probable sentence, the sequence of most likely tokens would be selected. This could mean, for instance, that instead of

Twente University the system could suggest *Twenty University* as the correction, since *Twenty* occurs more frequently than *Twente* in most texts.

2.4 Levenshtein distance

The Levenshtein distance[14] measures the minimum number of single character edits needed to transform a string into another one. Edits include the insertion, deletion, and replacement of characters. The distance can be used to find correction candidates and help in determining the most likely one.

3. RELATED WORK

3.1 Norvig

Peter Norvig proposed a method that corrects up to two errors in a token [18]. If the entered token is known it is returned. Otherwise, it returns the candidate with one or two mistakes and the highest likelihood. If non exists, the original query is returned.

3.2 FAROO

The approach by FAROO works by storing tokens with all possible character deletions up to distance 2, for each token in a training set [9]. The same deletions are generated when a token needs to be corrected. The lists of tokens with deleted characters are then compared to find possible corrections.

4. RESEARCH APPROACH

4.1 Supplied data

UT Search is currently operational and logs all entered search queries. A log containing 14,810 queries was provided by the University. In addition, a web crawl of approximately 40 GB was provided, containing websites related to the university. A subset of the crawl will be used to train the spelling correction algorithms, since the entire crawl is too large for the system to handle.

4.2 Correction methods

With help of the query log and literature knowledge, techniques will be selected, created, and combined to form different versions of spelling correctors. These variants need to be trained and tested to determine the best variant and answer the research questions.

The base of the correction algorithms will be a *language model* (section 2.3) which uses bigrams and applies *additive smoothing*. The model gives the likelihood of a sentence to occur in a language. This likelihood can be combined with other features to yield a score used for candidate selection.

4.3 Testing performance

There are three options to test the performance of the correction algorithms. Firstly, an annotated list of queries, such as the *query log*, could be used for automated testing. Here an expert defines the correctly spelled queries.

Secondly, the quality of the search results attained by the corrected queries could be used as a metric[7][12]. In this case, test users could be asked to rank the results of the original and corrected queries to see which one performs best.

Finally, test users could enter both erroneous and errorless queries and judge the corrections. By indicating whether the entered query contained errors, several metrics can be derived, such as the F_1 *score* and the accuracy. This method requires test users who posses knowledge of the University of Twente.

The third option will be used in this research with accuracy as the performance metric. The latter was chosen since it is equally important not to add errors to errorless queries as it is to correct bad queries and the accuracy does not discriminate on the type of mistakes made by the spelling corrector.

Aside from feedback on the corrections, the time needed to calculate the corrections will be stored, since this information is required to answer the second research question.

5. TECHNIQUES

5.1 Token candidate generation

There are two ways in which candidates for single tokens will be generated in this research. The first is the approach of *Peter Norvig* (section 3.1). Contrary to the original implementation, our implementation also takes words into account which did not occur in the training set, and returns the union of all candidates instead of the most probable correction. The main reason for this is that most queries consist of multiple tokens and the goal is to find the most probable query, instead of the sequence of the most probable single tokens. Therefore, all token candidates need to be supplied by the algorithm instead of the most likely candidate.

The second approach uses *FAROO's* method (section 3.2). Again, the implementation was modified to yield all possible candidates instead of the top candidate. Furthermore, since the web sites in the data crawl are very large, the set of precalculated data will become extremely large. As a result, the correction times needed may grow exponentially and the systems may not be able to store all data in the main memory. To mitigate this problem, a feature was added to the systems using FAROO to only store the top n most frequent tokens and remove the rest, to save space and reduce the correction effort.

To limit the time complexity of the algorithms and to remove unlikely correction candidates, the maximum edit distance of the tokens was limited. A maximum edit distance of *2* was chosen for each token in a query, for both candidate generation methods. This means that the query itself may contain more than two errors, namely two per token at most.

To determine the best correction candidate of a query, the Cartesian product of the token candidates is calculated. For each query candidate in this set, the likelihood is calculated. This likelihood is then used by the different spelling correction algorithms to determine the final score of the query candidate. The corrected query with the highest score is returned to the user.

5.2 Sliding

The disadvantage of calculating the Cartesian product of all token candidates is the exponential growth of the search space. To solve this, two greedy approaches are implemented in the correction algorithms which *slide* over pairs of tokens, from left to right, to reduce the search space while processing the query.

The first approach is the *soft* slide. At each step in the

sliding process, the top halve most likely query candidates thus far is kept. The *hard* slide, however, only stores the single best candidate at each step.

Both approaches reduce the memory footprint of the application and lower the calculation times. However, this may result in a lower quality of the correction results.

5.3 Maximum query edit distance

The techniques described earlier all allow two errors to be made in each token of the query. Consequently, corrections may be proposed for an unrealistic number of errors. Furthermore, these techniques do not solve errors caused by (the lack of) spaces. To solve this problem, an additional method is added which looks at the number of mistakes made in the entire query, instead of single tokens, and also adds and removes spaces.

5.4 Bigram inverses

Bigram inverses (section 5.4) may be used to solve problems in which two words where entered in reverse order. The inverses are optionally taken into account by the algorithms when counting bigrams in the lanuage model, by adding the counts of the inverses to the regular counts.

5.5 First letter bonus

It is expected that the first letter of search tokens is less frequently misspelled than the other characters. To test this hypothesis, and possibly improve performance, an optional bonus score is added to token correction candidates which have a matching first letter.

5.6 Edit distance bonus

Intuitively, a correction candidate which is more similar to the misspelled tokens is more likely to be the right one than a candidate that contains more differences. Therefore, bonus scores are added for this feature as well.

5.7 Threshold

To prevent unlikely candidates from being returned as the correction, an optional likelihood threshold was implemented[21][13]. When the threshold is not reached the original query is returned.

5.8 Cleaning

To prevent the model from becoming too large when training on large data sets, the data set is cleaned periodically. During the cleaning process only the tokens and n-grams which occur more than a certain number of times are kept[19]. This may affect the correction quality[13].

6. IMPLEMENTATION

The models and variations were implemented using Python. In addition to stand-alone training and testing, the application is able to run as a REST server. When deployed as a server, the application logs all queries ran, including their execution times, and the feedback received from test users. The implemented models are listed in table 2 of the appendix.

In addition, a web application was developed using HTML and JavaScript. This application is utilized by the test users to correct queries and provide feedback to the corrections. The web application sends the queries and feedback to the Python server for processing.

A selection of the websites from the web crawl had to be made, since the provided crawl was too large for the system to handle, even with a cleaning process implemented. The university website itself and websites which are likely to contain common terms of the university were selected. The list of selected websites and their corresponding sizes can be found in table 1 of the appendix.

7. RESULTS AND DISCUSSION

7.1 Problems

Several problems occurred during preliminary testing and the final tests. Firstly, although only 3.8 GB of websites were selected for training, the training process took too long. Therefore, only 10 percent of the data, chosen randomly at run-time, was used.

Secondly, although cleaning lowered the memory footprint of the server substantially, it still consumed a lot of memory. Since the application ran on a shared server, after a period of inactivity the application's data in the main memory was moved to the *swap*. Consequently, the first query corrections made after an inactive period were slow, since the data had to be moved to the main memory again.

Thirdly, queries with three or more words took so much time to process that the testing system became unusable. Therefore, during the testing process, the query length was limited to two words. As a result, the *sliding* algorithms and correction of long queries could not be tested, even though long queries occur frequently in the UT Search query log.

Finally, the technique that limits the number of edits in the entire query (section 5.3) did not work well in practice. While the implementation was functional and able to correct more complex mistakes, it took as long as 5 minutes to correct queries of low complexity. Therefore, it was not included in the final tests.

7.2 Testing

The test users - UT students - submitted 239 queries in the final test, all of which were used to measure the correction times. For 200 of these queries valid feedback was received. For one query invalid feedback was received and for 38 others no feedback at all.

The users were allowed to submit errorless queries to the system and could, for each query, indicate if they had. Only 13% of the queries entered did not contain errors.

7.3 Accuracy

The highest accuracy attained by a model was 0.675. This model corrected 68% of the erroneous queries but introduced errors to 33% of the errorless ones. This last problem may be solved by not correcting queries that already pass a likelihood threshold in the language model, although such queries may still be erroneous. Another option is to propose corrections similar to Google's *did you mean* functionality, instead of applying them anyway.

The model used FAROO for candidates generation and did not take inverted bigrams into account. It did not use sliding but this may be due to the imposed limitation of two words per query. Furthermore, it did not use likelihood thresholds, but it did give bonuses for equal first letters and a low edit distance. This answers the first and third research questions.

The exact results can be found in table 3 of the appendix.

7.4 Speed

On average, the algorithms required 347 milliseconds to correct a query. However, preliminary testing revealed that the time needed strongly depends on the training set's size. By decreasing this size, reducing the time below 200 milliseconds should be no problem, given a maximum query length of two words.

The best model required 1288 milliseconds on average but this is likely caused by the high start-up time required after a period of inactivity, since the best model was the first in line to be tested for each query. Since the model is very similar to other models, including ones using more complex techniques, the system is probably much faster than the measurements indicate.

To answer the second research question, the current implementations did not operate within 200 milliseconds. However, with a reduction of the model size, a limited query length, and more main memory the time constraint can be reached.

8. CONCLUSIONS AND FURTHER WORK

In this research eighteen data-driven spelling correctors were developed, trained and tested. The best performing one attained an accuracy of 67.5 percent. This model did not include inverted bigrams in its model, which may indicate that these do not lead to a higher accuracy.

The systems were not able to handle large amounts of data. In the end around 380 MB of websites was used for training. This was still too much data to attain correction times below 200 milliseconds.

Further research should focus on handling large amounts of data, for instance, by using *MapReduce* to create a language model. In addition, other ways need to be found to increase the accuracy of the models and increase their speeds.

9. REFERENCES

[1] D. Boswell. CSE 256 (Spring 2004)"Language Models for Spelling Correction". *B Siklósi et al/Computer Speech and Language xxx (. . .*, 2004.

[2] A. Boyd and D. Meurers. Data-Driven Correction of Function Words in Non-Native English. pages 1–3, Oct. 2011.

[3] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, Apr. 1973.

[4] A. Carlson and I. Fette. Memory-based context-sensitive spelling correction at web scale. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 166–171. IEEE, Sept. 2007.

[5] S. F. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. 1998.

[6] M. Choudhury, M. Thomas, A. Mukherjee, A. Basu, and N. Ganguly. How Difficult is it to Develop a Perfect Spell-checker? A Cross-linguistic Analysis through Complex Network Approach. Mar. 2007.

[7] V. Dang and B. W. Croft. Query reformulation using anchor text. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 41–50, New York, New York, USA, Feb. 2010. University of Massachusetts Amherst, ACM.

[8] A. Elghafari, D. Meurers, and H. Wunsch. Exploring the data-driven prediction of prepositions in English. In *COLING '10: Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1122–1130. University of Tubingen, Association for Computational Linguistics, Aug. 2010.

[9] FAROO. 1000x faster spelling correction algorithm, 2012.

[10] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A Large Scale Ranker-Based System for Search Query Spelling Correction. pages 1–9, Aug. 2010.

[11] P. Hsu. Online Spelling Correction for Query Completion. pages 1–10, Feb. 2011.

[12] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. *WWW*, pages 387–396, 2006.

[13] A. Koul. Spelling Alteration for Web Search Workshop. pages 1–32, July 2011.

[14] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–, Feb. 1966.

[15] Y. Li, H. Duan, and C. X. Zhai. Cloudspeller: Spelling correction for search queries by using a unified hidden markov model with web-scale resources. *Spelling Alteration for Web Search Workshop*, 2011.

[16] A. f. C. Linguistics. Learning a Spelling Error Model from Search Query Logs. pages 1–8, Nov. 2005.

[17] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, July 2008.

[18] P. Norvig. How to write a spelling corrector, 2007.

[19] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? 2000.

[20] T. Sullivan and A. Kulkarni. CS 276 Programming Assignment 1: K-Gram Spelling Correction and Lucene. pages 1–10, Dec. 2008.

[21] C. Whitelaw, B. Hutchinson, G. Y. Chung, and G. Ellis. Using the web for language independent spellchecking and autocorrection. In *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, pages 890–899. Google Inc., Association for Computational Linguistics, Aug. 2009.

APPENDIX

Table 1: Website selection

Website	Size (in MB)
blackboard.utwente.nl	4.4
my.utwente.nl	1.3
utwente.nl	2.5
www.ewi.utwente.nl	2.1
wwwhome.cs.utwente.nl	0.1
wwwhome.ewi.utwente.nl	8.6
www.union.utwente.nl	9.0
www.utwente.nl	3805.0
	3833.0

Table 2: Model variants

		Slide					
Variant	Candidate	Soft	Hard	Inverse bigrams	Threshold	Edit bonus	First letter bonus
0	FAROO	0	0	0	-	1.2	.95
1	FAROO	1	0	0	-	1.2	.95
2	FAROO	0	1	0	-	1.2	.95
3	FAROO	0	0	0	-	-	-
4	FAROO	0	0	1	-	1.2	.95
5	FAROO	1	0	1	-	1.2	.95
6	FAROO	0	1	1	-	1.2	.95
7	FAROO	0	0	1	-	-	-
8	Norvig	0	0	0	-	1.2	.95
9	Norvig	1	0	0	-	1.2	.95
10	Norvig	0	1	0	-	1.2	.95
11	Norvig	0	0	1	-	1.2	.95
12	Norvig	1	0	1	-	1.2	.95
13	Norvig	0	1	1	-	1.2	.95
14	Norvig	0	0	1	-	-	-
15	Norvig	0	0	1	-110	-	-
16	Norvig	0	0	1	-90	-	-
17	Norvig	1	0	1	-	1.5	.50

Table 3: Model performance

	Accuracy			Errorless query		Erroneous query		
Variant	Errorless	Erroneous	Total	Wrong	Valid	Wrong	Valid	Avg. time (in MS)
0	.667	.676	.675	9	18	56	117	1288
1	.259	.197	.205	20	7	139	34	417
2	.259	.179	.190	20	7	142	31	83
3	.630	.647	.645	10	17	61	112	325
4	.667	.665	.665	9	18	58	115	652
5	.259	.208	.215	20	7	137	36	356
6	.259	.202	.210	20	7	138	35	110
7	.667	.670	.670	9	18	57	116	441
8	.778	.624	.645	6	21	65	108	322
9	.630	.324	.365	10	17	117	56	265
10	.630	.295	.340	10	17	122	51	226
11	.778	.584	.610	6	21	72	101	261
12	.630	.312	.355	10	17	119	54	248
13	.630	.306	.350	10	17	120	53	235
14	.778	.601	.625	6	21	69	104	254
15	.926	.387	.460	2	25	106	67	252
16	.926	.335	.415	2	25	115	58	251
17	.667	.306	.355	9	18	120	53	262