Computational Linguistics and Intellectual Technologies:
Proceedings of the International Conference “Dialogue 2018”

Moscow, May 30—June 2, 2018

A STUDY OF MACHINE LEARNING ALGORITHMS APPLIED TO GIS QUERIES SPELLING CORRECTION

Fomin V. V. (wadimiusz@gmail.com)
Novosibirsk State University, Novosibirsk, Russia

Bondarenko I. Yu. (i.yu.bondarenko@gmail.com)
2GIS, Novosibirsk, Russia

The problem of spelling correction is crucial for search engines as misspellings have a negative effect on their performance. It gets even harder when search queries are related to a specific area not quite covered by standard spell checkers, such as geographic information systems (GIS). Moreover, standard spell-checkers are interactive, i. e. they can notice a misspelled word and suggest candidate corrections, but picking one of them is up to the user. This is why we decided to develop a spelling correction unit for 2GIS, a cartographic search company. To do this, we have extracted and manually annotated a corpus of GIS lookup queries, trained a language model, performed various experiments to find the best feature extractor, then fitted a logistic regression using an approach suggested in SpellRuEval, and then used it iteratively to get a better result. We have then measured the resulting performance by means of cross-validation, compared at against a baseline and observed a substantial increase. We also present an interpretation of the result achieved by calculating and discussing the importance of specific features and analyzing the output of the model.

Keywords: spell checker, geographic information system, language model, text corpus, local search

ИССЛЕДОВАНИЕ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ ИСПРАВЛЕНИЯ ОПЕЧАТОК В ЗАПРОСАХ К ГИС

Фомин В. В. (wadimiusz@gmail.com)
Новосибирский государственный университет, Новосибирск, Россия

Бондаренко И. Ю. (i.yu.bondarenko@gmail.com)
2ГИС, Новосибирск, Россия

1. Introduction

The problem of spelling correction has a long history of research. Works in this area generally tend to belong to either of the following fields:

1. candidate search, i. e. suggesting corrections for a typo, such as the pioneer work of the field [Damerau 1964];
2. candidate selection, i. e. evaluating the candidate suggestion quality to select the best suggestion, such as [Kernighan, Church, Gale 1964].

Lately, there has been a research into the problem of context-sensitive spell-checking, i. e. using the word context for candidate selection, for instance [Golding, Roth 1999]. Some try to embed various approaches into a single candidate selection system, such as [Gao et al. 2010].

However, both tasks are rather challenging if the spell check algorithm is designed for specific use, e. g. for geographic information system (GIS) lookups such as electronic maps developed by Russian cartographic company 2GIS. Candidate search, on one hand, may pose a problem as standard spell checkers often appear unaware of GIS-related words, such as names of streets, companies etc, such as in the following example, where Hunspell treated a correct query as a typo and suggested this correction, undoubtedly because this tool is too general:

(1) *хоум кредит банк → ухом кредит банк*
(The part before "→" represents the original query,
the part after "→" represents the suggestion).

On the other hand, choosing a candidate may also be a problem because of the specific language used in the area. Search queries differ drastically from other forms of language, as syntactic and morphological information are much less helpful than usual.

Although there are papers describing spelling correction of search queries, such as [Martins, Silva 2004] for candidate search and [Wilbur, Kim, Xie 2004] for

candidate ranking, technical details of most search engine spell-checkers are a trade secret, which makes the problem under discussion even more challenging.

We decided to develop a unit aimed specifically at correcting search query typos. This means that we had to create a corpus for supervised learning, design a feature extractor (which included training a language model), design and fit a spell-checking model, and evaluate its performance.

2. Related work

One of the oldest works related to correcting spelling errors is [Blair 1960]. It introduces the idea of spelling corrections based upon a list of correctly spelled words and a string metric. Although the string metric suggested in this work gained no popularity, the approach itself is crutial to the task of spelling correction.

The two classical works for the task are [Damerau 1960] and [Levenshtein 1966], upon which the most popular string metric, the so-called Damerau–Levenshtein distance, is based. The distance between two words is the number of operations it takes to turn one word into the other one.

The problem of candidate search is considered in [Brill, Moore 2000], a paper which discusses the noisy-channel model, a way of determining which correction is better for a certain typo by assigning each typo a probability and using it to determine the score of each correction. This, however, means that the model is incapable of taking the context into account and deciding whether or not a correction suits its context.

An interesting approach to spelling correction of search queries is presented in [Cucerzan, Brill 2004]. The authors of this paper suggest a spell-checker designed for search queries should rely upon the statistcs of search queries and correct them iteratively, in several steps, making a typo less malign wth each iteration. This is opposed to the usual approach that relies upon a list of correct words and suggests correcting a typo in one move.

3. Text corpus

We have used two query corpora to fit our model. We refer to the first corpus as “supervised” and to the second as “unsupervised”. The first corpus consists of 14,400 manually annotated queries. Each entry in this corpus looks like this:

(2) *большой сухаревский переклок 23/25,большой сухаревский переклок 23 25,большой сухаревский переулок 23 25,1*

The entries contain four fields. The first field is the original query, the second field represents the result of a basic preprocessing, the third one is a gold-standard correction, and the fourth one is a binary classification label, where “1” stands for “This query contains a typo” and “0” stands for “This query does not need being corrected”.

This corpus is used for supervised learning and for evaluating its performance metric.

The second corpus contains around one million raw queries. This corpus was used for creating a language model.

4. Model

Our model can be accessed at [Fomin 2017].

Our suggested correction lookup unit was influenced by the work of Cucerzan and Brill [Cucerzan, Brill 2004]. The approach presented in this work (and implemented in our model) is as follows. When we use dictionaries that only consist of properly spelled words for candidate search task, we have to pick from two extremities: only considering corrections that are close to the original word or including corrections that are rather distant from the analyzed word.

In the first case we are bound to fail in any complicated case like

(3) *anol scwartegger*

instead of

(4) *arnold schwarzenegger*

In the second case we have to process a stupendous number of corrections, some of which may be erroneously taken for the right ones, which also leads to poor model performance. The solution is to take misspelled words into account when performing the candidate search, and to make the spelling correction model process the sentence several times iteratively, so that every time the misspelling becomes less malign:

(5) *anol scwartegger → arnold schwartnegger → arnold schwarznegger → arnold schwarzenegger*

Our candidate ranking system was inspired by the work of Sorokin and Shavrina. [Sorokin, Shavrina 2016] We have created a feature extractor which takes in two strings (one of them being the original query, the other the suggested correction) and returns the following features:

1. Length of the correction.
2. Simple Levenshtein distance between the original query and the correction.
3. The score of the correction evaluated by a SRILM [Stolcke 2002] ngram-model.
4. The number of out-of-vocabulary words in the corrections.
5. The number of vocabulary words that became out-of-vocabulary after the correction.
6. The number of out-of-vocabulary words that became vocabulary words after the correction.
7. The number of words whose correction is more frequent than the original correction.
8. Simple Levenshtein distance between the original tokens and their corrections, where the original tokens are only considered if they are out-of-vocabulary.
9. Simple Levenshtein distance between the original tokens and their corrections, where the original tokens are only considered if they are in the vocabulary.
10. The number of original tokens whose corrections are 1 Levenshtein operation far from the originals.
11. The number of space symbol deletions.

12. The number of space symbol insertions.
13. The number of out-of-vocabulary words that can be split into two vocabulary words.
14. Weighted Levenshtein distance where the weight of a substitution operation is the distance of the corresponding symbols on a phone keyboard layout.
15. Weighted Levenshtein distance where the weight of a substitution operation is the distance of the corresponding symbols on a phone keyboard layout and the weight of a deletion operation is 10.
16. Weighted Levenshtein distance where the weight of a substitution operation is the distance of the corresponding symbols on a phone keyboard layout and the weight of a insertion operation is 10.
17. Simple Levenshtein distance between the phonetic codes (described in [Sorokin, Shavrina 2016]) of the original query and the suggestion.
18. The number of corrections of the form "*а → о*, *о → а*, *е → и*, or *и → е*".
19. The number of corrections of the form "*ы → и*, *ё → о*, *ю → у* after *ж*, *ч*, *ш*, *щ*".
20. The number of corrections of the form "*цы → ци*" or "*ци → цы*".
21. The number of corrections of the form "*ыва → ова*".
22. The number of corrections of the form "*аро → оро*" or "*ало → оло*".
23. The number of corrections of the form "*э → е*".
24. The number of corrections of the form "*ща → ще*".
25. The number of corrections of the form "*пре → при*" or "*при → пре*".
26. The number of corrections of the form "*э → и*".
27. The number of corrections of the form "*ё → йо*" or "*е → йо*".
28. The number of corrections of the form "unvoiced consonant → its voiced equivalent", or "unvoiced consonant → its voiced equivalent".
29. The number of corrections of the form *"зн → здн"*, "*сн → стн*", "*сл → стл*", "*нст → нтст*", "*здн → зн*", "*стн → сн*", "*стл → сл*", or "*нтст → нст*".
30. The number of corrections of the form *"хк → гк"*.
31. The number of corrections of the form "*н → нн*", "*с → сс*", "*м → мм*", "*ф → фф*", or vice versa.
32. The number of corrections of the form "*ь → ъ*" or "*ъ → ь*".
33. The number of corrections of the form "insertion of *ь* as the fourth-to-last letter".
34. The number of corrections of the form "*тся → ться*" or "*ться → тся*".

We have used SRILM [Stolcke 2002] and our untagged corpus to fit a language model used in the feature extractor as the feature #3. After we tried running our spell checker with models at different smoothing types and context windows and found that the best performance is achieved when a bigram model with Kneser-Ney smoothing is used. The fact that no bigger order of ngram-model (trigrams etc.) was required is remarkable and somewhat characteristic of the language of search queries.

Features from 18 to 34 were included in order to capture some typical cases of typos, so that the suggestions that assume a more usual typo would be considered a better corrections by the model.

We have also tried using several morphological or simple syntactic models. Attempting to train our model to generalize morphological properties of the corpus, we trained

a SRILM model on a version of the "unsupervised" corpus with words replaced by morphological tags, making TreeTagger [Schmid 2013] do the tagging; we also tried to replace all the words but the most frequent ones. We tried to make our model understand simple syntax (such as the fact that a word in dative case is expected after the preposition к) by leaving prepositions intact and replacing other words with morphological tags. We gave up these ideas as they did not improve the model performance.

We have also found that it is better to replace numbers in addresses of building by an artificial token, like this:

(6) *семёновская 9 → семёновская <NUMBER>*

This is because these numbers are not informative in spell-checking and have a bad effect on the model performance.

After features are extracted, we create the training set using the approach described in [Sorokin, Shavrina 2016]. For each query, we have one "winner suggestion" and a lot of "loser suggestions". We then compute the difference between the "winner suggestion" and each of the "loser suggestions" and assign them the label "1". Then, we take the same vectors multiplied by −1 and assign them with the label "0". We then transmit the resulting training set to a logistic regression model. By doing so, we train this model to maximize feature weights if the feature indicates that the suggestion is good and minimize feature weights of features that indicate that the suggestion is bad. This approach is equivalent to the one used in [Sorokin, Shavrina 2016], and it appeared very helpful.

After the training is done, we take the test set, generate a list of suggestions, put it through the trained logistic regression model, and claim that the suggestion with the highest score is the winner. After this is done, the winner sentence itself is now treated as the misspelled sentence which is to be corrected. This is repeated iteratively until the model recognizes that no further improvement can be done:

(7) *краснопресн → краснопресне → краснопресненеская → краснопресненская*

5. Evaluation

Evaluation in machine learning tasks consists of two main steps, namely picking and computing an evaluation metrics and comparing the results of the system in question against a baseline algorithm. When evaluating a spell-checking algorithm, both steps pose a certain difficulty.

First, as discussed in [Sorokin et al. 2016], typical spelling correction performance metrics such as the percentage of correctly processed sentences (or, in our case, queries) are not representative enough because they do not take the number of misspellings and corrections into account, and also because they make no difference between various cases such as making an undesired correction or leaving out a misspelled word that was to be corrected, so a model which performs no corrections at all would still have a 90% sentence accuracy upon a corpus with 10% misspelled sentences.

Second, comparing the model against a baseline would require the output of an algorithm on our corpus which is difficult because autonomous spell-checkers

designed for search engine queries are not open-source systems, or available for researchers; typically, nor are their technical details.

We have treated these problems as follows.

For evaluating our model, we have implemented the same approach as the organizers of SpellRuEval described in [Sorokin et al. 2016]. Instead of using primitive performance metrics, we treated each word or word group that was and/or should have been corrected as a separate case, marking them as true positives, false negatives or false positives. Cases where the original query had contained a typo and was corrected by the system in the desired way were considered to be true positives. Instances where the typo was not affected by the spell checking system, or was corrected in a way different from the golden standard were treated as false negatives. Occasions when a properly spelled word was mistaken for a misspelling were treated as false positives.

We have then evaluated precision, recall and F1-measure using standard formulas for binary classification. We have performed a cross-validation over 5 folds to do so.

We have used Hunspell as a baseline, with the following results:

	F1-score	Precision	Recall
Our algorithm	73.5%	94.8%	58.6%
Hunspell	20.8%	19.6%	22.2%

Details of the evaluation are presented below.

Hunspell. First, we have put our corpus through Hunspell [Németh 2003], a widespread interactive spell-checker, by asking it word by word whether the given word is correct and replacing the supposed misspellings by the best of all suggestions Hunspell could make (if any). It is important to note that it was not our goal to test our dictionary against Hunspell's standard dictionary; we wanted to find out whether we succeeded in taking the language of the corpus into account and if our language modeling actually resulted in an improvement. Thus, we updated its standard list of correct tokens with a list of all streets, names of enterprises, cafés etc, so that we are sure that the difference in models' performance, if we observe any, is a caused by the difference in approaches to candidate selection and not in the dictionaries. A comparable result in Hunspell and our model would mean that language modeling was an unsuitable approach for correcting GIS search queries; observing our model's performance exceed qualities would mean that modeling the language of search queries for spelling correction is a good idea. Since Hunspell sorts the candidate corrections from most to least probable, we considered the first suggestion to be Hunspell's choice. The gap between the results of the two systems presented above is an evidence that language modeling is crucial for correcting the spelling of search queries and that our attempt to implement a system able to perform language modeling for spelling correction of search queries was successful.

SpellRuEval. We could not properly compare our results with the results of SpellRuEval winners [Sorokin, Shavrina 2016] since the source code is not available for us; however, it is interesting to note that they demonstrate an F1-score of 75%, which is a comparable result. It is hard to draw any conclusions from that because this score

was achieved on quite a different corpus. On one hand, the lack of grammatical information and the amount of unusual tokens make the spell-checking of search queries a harder task. On the other hand, the plenty of morphological and syntactic information in regular texts enlarges the number of candidates and makes the candidate selection a more subtle task. In general, the fact that the winners of SpellRuEval achieved the result of 75% suggests that it is an acceptable result, although this is much less of a solid evidence than the baseline algorithms discussed above.

6. Analysis

After the training, we have extracted the feature weights from our logistic regression to know how much each feature contributes to the final decision of the model. A huge absolute value of a feature weight means that it plays an important role in taking decisions (a large positive coefficient means that examples with larger values of this feature tend to belong to class “1”, whereas a large negative weight means that examples with larger value of this feature belong to class “0”). The result is presented on the following bar chart:

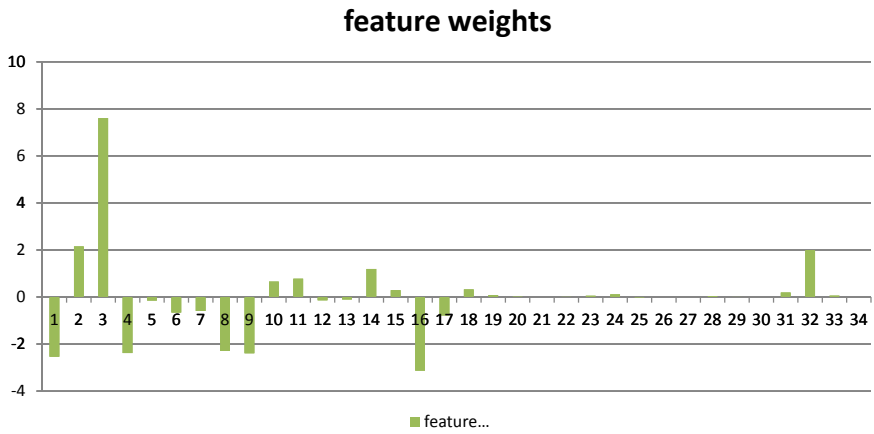

Fig. 1. The weights of various features in our feature extractor; each bar has a number and represents the feature of the same number; the height of a bar represents the weight of the corresponding feature

Here, each number represents a certain feature; for more clarity, we briefly repeat the descriptions of features whose detailed versions are available in part 3 “Model” (OOV stands for out-of-vocabulary):

1	Correction length	18	*а* → *о*, *о* → *а*, *е* → *и*, or *и* → *е*
2	Simple Levenshtein distance	19	*ы* → *и*, *ё* → *о*, *ю* → *у* after *ж*, *ч*, *ш*, *щ*
3	Ngram-model score	20	*цы* → *ци* or *ци* → *цы*
4	OOV in corrections	21	*ыва* → *ова*

5	Vocabulary → OOV	22	*аро* → *оро* or *ало* → *оло*
6	OOV → vocabulary	23	*э* → *е*
7	Corrections that are more frequent than the originals	24	*ща* → *ще*
8	Simple Levenshtein distance, only OOV originals	25	*пре* → *при* or *при* → *пре*
9	Simple Levenshtein distance, originals in vocabulary only	26	*э* → *и*
10	1-operation corrections	27	*ё* → *йо* or *е* → *йо*
11	Space deletions	28	unvoiced → voiced, or voiced → unvoiced
12	Space insertions	29	*зн* → *здн*, *сн* → *стн*, *сл* → *стл*, *нст* → *нтст*, *здн* → *зн*, *стн* → *сн*, *стл* → *сл*, or *нтст* → *нст*
13	OOV that can be split into two vocabulary words	30	*хк* → *гк*
14	Weighted keyboard layout Levenshtein distance	31	*н* → *нн*, *с* → *сс*, *м* → *мм*, *ф* → *фф*, or vice versa
15	Weighted Levenshtein distance with insertion weight 10	32	ь → ъ or ъ → ь
16	Weighted Levenshtein distance with deletion weight 10	33	insertion of ь as the fourth-to-last letter
17	Simple Levenshtein distance between phonetic codes	34	*тся* → *ться* or *ться* → *тся*

It is worth noting that the ngram-model score, represented by the feature #3 on Fig. 1 above, is by far the most important component of the feature vector. A high estimated probability of a query correction based upon the language of the corpus as a is an important evidence that the correction under discussion is to be accepted.

Other features with a high positive weight include feature #32, which will be discussed later, and, strangely, feature #2 which represents the simple Levenshtein distance between the original and the correction. This may be because there are several string metric features in the corpus, sometimes with close results, so that their contribution is shared and the weight of this specific feature is somewhat dependent on the initialization.

Other features of importance have a negative weight which means that they are useful for rejecting bad suggestions, rather than selecting a good one. This includes feature #1, which represents correction length and suggests that shorter corrections are better; feature #4 which means the number of out-of-vocabulary words in the correction; and features #8, #9, and #16, representing various string metrics. It is also interesting to note that feature #16 that signifies weighted Levenshtein distance with substitutions weighted proportionally to the distances on a keyboard layout and deletions weighted 10 times as high as insertions, turned out to be the most successful string metric. This might be because it represents the properties of search queries

where substitutions are caused by the so called fat finger syndrome (mistakingly hitting adjacent keys on a keyboard) and typos that delete a character are much more common than those that insert one.

It is also interesting that features from #18 to #34 that were designed to represent typical typo cases mostly did not work out. The only feature of this type that has a clearly high absolute value of the weight is #32, which treats cases like:

(8) *сьесть → съесть*

This is due to the fact that letter ъ, on most phone keyboard layouts, can only be accessed via a certain manipulation with the key ь. Other features have a relatively low importance, either because they are represented by other features already, such as feature #18 vs. feature #17 for instances like

(9) *улеца → улица,*

or because it represents a relatively rare type of misspellings not quite relevant for the corpus, such as feature #29 for cases like

(10) *агенство → агентство.*

Besides, we have analyzed the output of our model to find the kinds of incorrect behavior that the described system is prone to. Although no particular regularities are evident in the output, it could be inferred from the discussed data that the most complicated cases that lead our system to false negative result are those whose originals are rarely seen in the search queries, such as

(11) *осташелвскпя → осташелвскпя*

instead of

(12) *осташелвскпя → осташевская,*

and queries chopped off at the middle of the string:

(13) *залес → залесс*

instead of

(14) *залес → залесского.*

7. Acknowledgements

The authors are grateful to Alexander Krinitsyn for his valuable advice. We also thank 2GIS who provided us with a corpus of GIS search queries and Botan Investments for supporting students' interest to machine learning.

References

1. *Blair C. R.* (1960), A program for correcting spelling errors. Information and Control. Vol. 3, №1, pp. 60–67.
2. *Brill E., Moore R. C.* (2000). An improved error model for noisy channel spelling correction. In Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (pp. 286–293). Association for Computational Linguistics.
3. *Cucerzan S., Brill E.* (2004), Spelling correction as an iterative process that exploits the collective knowledge of web users, Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, Copenhagen.
4. *Cucerzan S., Brill E.* (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing.
5. *Damerau F. J.* (1964), A technique for computer detection and correction of spelling errors, Communications of the ACM, Vol. 7, № 3, pp. 171–176.
6. *Fomin V.* (2017), A term project on spell-checking. [ONLINE] Available at: https://github.com/wadimiusz/spellchecker [Accessed 20 February 2018].
7. *Gao J. et al.* (2010), A large scale ranker-based system for search query spelling correction, Proceedings of the 23rd International Conference on Computational Linguistics, Beijin, pp. 358–366.
8. *Golding A. R., Roth D.* (1999), A winnow-based approach to context-sensitive spelling correction, Machine learning, Vol. 34, № 1–3, pp. 107–130.
9. *Kernighan M. D., Church K. W., and Gale W. A.* (1990), A spelling correction program based on a noisy channel model, In Proceedings of the 13th conference on Computational linguistics, Avignon, pp. 205–210.
10. *Martins B., Silva M. J.* (2004), Spelling correction for search engine queries, Advances in Natural Language Processing, Springer, Berlin, Heidelberg, pp. 372–383.
11. *Németh L.* (2003) Hunspell. [ONLINE] Available at: http://hunspell.github.io/ [Accessed 20 February 2018].
12. *Schmid H.*(2013) Probabilistic part-of-speech tagging using decision trees, New methods in language processing, Manchester, p. 154.
13. *Sorokin A. A. et al.* (2016), Spellrueval: the first competition on automatic spelling correction for Russian, Proceedings of the Annual International Conference "Dialogue", Moscow.
14. *Sorokin A. A., Shavrina T. O.* (2016), Automatic spelling correction for Russian social media texts, Proceedings of the International Conference "Dialog", Moscow, pp. 688–701.
15. *Stolcke A.* (2002), SRILM-an extensible language modeling toolkit, Seventh international conference on spoken language processing, Denver.
16. *Wilbur W. J., Kim W., Xie N.* (2006), Spelling correction in the PubMed search engine, Information retrieval, Vol. 9, № 5, pp. 543–564.