Bulletin of the *Transilvania* University of Braşov • Vol 4(53), No. 2 - 2011
Series III: Mathematics, Informatics, Physics, 141-146

A PROBABILISTIC MODEL FOR SPELLING CORRECTION

Lucian SASU[1]

Abstract

Spelling correctors are widely encountered in computer software applications and they provide the most plausible replacements for words that are presumably incorrect. The paper proposes a spelling correction method starting from the Damerau-Levenshtein edit distance and using Bayesian decision theory. The resulted algorithm is tested on a bag of words from the New York Times news articles.

2000 *Mathematics Subject Classification:* 68T37, 68T50, 68T10.
Key words: Spell correction, Levenshtein distance, Damerau-Levenshtein distance, Bayesian decision theory, restricted edit distance.

1 Introduction

Spell checking and spell suggestion are nowadays widely used in word processors, spell checkers and web search engines. Most popular spell checkers starts from a thesaurus containing correctly spelled words and confront them to the words a user writes. Other systems rely on a set of documents for which the correct forms of the words and their frequencies are provided – the so-called bag-of-words model. This approach is used in our paper. Other linguistic sources that can be used are lists of frequently misspelled words, search query logs, domain specific data sources, *e.g.* IT or medical terms and technical abbreviations.

The proposed model for providing correct and plausible forms of given word relies on two components. The first one is an edit distance function and gives the closeness between the word to be corrected and other words from the thesaurus. An edit distance gives the number of transformations one has to make on the given word in order to reach a correct form. These transformations include deleting, inserting, editing individual letters and exchanging pairs of characters. The paper includes presentations of the Levenshtein and Damerau-Levenshtein distances, commonly used to measure the spelling similarity of two words.

The second component is based on Bayes' theorem, which allows us to include the prior probability of a candidate word. Intuitively, it make sense to favor the most frequent words in use as being proposed as correct forms. The frequency gives a criterion for tie breaking, that is to make a decision for some candidate words that have the same distance from

[1]Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: lmsasu@unitbv.ro

the one to be corrected: we consider the most popular words are the the most plausible suggestions, all other factors being equal; this makes our system adaptive, as some of the values used to make predictions are built based on the provided data. Beside this intuitive support, the Bayesian decision theory is known to lead to minimizing the average probability of the error [1].

The resulted model produces spelling suggestions based both on the similarity between the misspelled word and on the frequencies of the words in the thesaurus. As all we need here is the thesaurus and the frequencies of the words, the bag-of-word structure is ideal in our case; the grammar structure or word order is irrelevant for now.

The structure of the paper is as follows: section 2 describes and compares two popular metrics used to compute word similarities. In section 3 we give the probabilistic model built on the language model and that is able to incorporate any string distance function. Finally, section 4 describes the data source used for building and testing the spelling corrector and contains the experimental results.

2 Edit distances between words

2.1 The Levenshtein distance between words

The Levenshtein distance is a widely used metric for measuring how different two given words are. The differences are considered here as the minimum number of editing operations. In this case an editing consists of insertion, deletion, or substitution of a single character. It was introduced in 1966 by Vladimir Levenshtein [2]. Beside spell checking systems, his algorithm is used as an auxiliary tool for optical character recognition [3] and natural language translation [4]. The algorithm implementing computation of this distance is realized through dynamic programming and it was discussed in [5]. For two words $X = x_1 \dots x_m$ and $Y = y_1 \dots y_n$, we denote by $D[i, j]$ the minimal number of edit operations one has to make in order to transform the substring $x_1 \dots x_i$ into $y_1 \dots y_j$, for $1 \leq i \leq m$, $1 \leq j \leq n$. $D[m, n]$ is the sought Levenshtein distance between X and Y. The algorithm relies on the following theorem:

Theorem 1. *The transformation of a string $X = x_1 \dots x_m$ into $Y = y_1 \dots y_n$ can be made with $D[m, n]$ edit operations, where $D[i, 0] = i$, $D[0, j] = j$ for $0 \leq i \leq m$ and $0 \leq j \leq n$, and for $1 \leq i \leq m$, $1 \leq j \leq n$ $D[i, j]$ is defined as:*

$$D[i,j] = \begin{cases} D[i-1, j-1], & \text{if } x_i = y_j \\ \min\{D[i-1, j], D[i, j-1], D[i-1, j-1]\} + 1, & \text{if } x_i \neq y_j \end{cases} \quad (1)$$

Proof. The proof is made by contradiction and can be found, for example, in [6]. □

The pseudocode for the algorithm computing the matrix D is given below. Its complexity is $\Theta(m \cdot n)$, both for execution time and for required memory. Its memory requirement can be further improved by considering only the current and previous lines for each cycle from lines 8–12. As a side note, the algorithms parallelize poorly due to high data dependency. In the above algorithm the cost for every operation is considered to be 1, but it

works for every positive cost assigned to each operation as done in Needleman–Wunsch algorithm [7]. Although in Needleman–Wunsch algorithm the problem is to maximize the similarity between two sequences, it has been shown that it is equivalent to minimizing the Levenshtein distance [8].

```
EditDistanceLevenshtein(X, Y)
 1  m ← length[X]
 2  n ← length[Y]
 3  allocate space for the matrix D[0...m, 0...n]
 4  for i ← 1 to m
 5      D[i, 0] ← i
 6  for j ← 1 to n
 7      D[0, j] ← j
 8  for i ← 1 to m
 9      for j ← 1 to n
10            if x_i = y_j cost ← 0
11      else  cost ← 1
12            D[i, j] ← min {D[i − 1, j] + 1, D[i, j − 1] + 1, D[i − 1, j − 1] + cost}
```

2.2 The Damerau-Levenshtein distance

The three editing operations considered by the Levenshtein distance can be seen as solutions for noises appearing due to misspelling. However, when typing, a human is exposed to another more likely mistake: exchanging two adjacent letters. In [9], the author argues that more than 80% of misspellings are due to letter transpositions. Hence, the Damerau-Levenshtein distance refers to the minimal number of insertions, deletions, editing and transpositions [9].

There are two versions of this algorithm: no substring is edited more than once — also called "restricted edit distance" or "optimal string alignment" — and another one without this restriction, allowing for adjacent transpositions. To understand the difference between the two, we consider the following example: X=CA and Y=ABC. The optimal string alignment distance is 3: CA→A→AB→ABC, while by using the later approach we get edit distance 2: CA→AC→ABC. Exchanging two letters by mistake occurs often, but producing a typo by adding letters between the switched characters can be seen as very unlikely. Hence, we consider here only the restricted edit distance.

The `RestrictedEditDistance` algorithm for optimal string alignment is essentially the same as the one for computing the Levenshtein distance with the following two statements added after line 12, inside the "**for** j" cycle:

```
13 if i > 1 and j > 1 and x_i = y_{j−1} and x_{i−1} = y_j
14       D[i, j] ← min(D[i, j], D[i − 2, j − 2] + cost)
```

3 A probabilistic model for spell suggestion

We consider here Bayes' theorem [10]: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. We use it to predict the probability of a correct spelling for a given word as:

$$P(\text{correct spell}|\text{word}) = \frac{P(\text{word}|\text{correct spelling})P(\text{correct spelling})}{P(\text{word})} \tag{2}$$

Note that eq. (2) contains the apriori probability $P(\text{correct spelling})$. We find this a natural requirement, as the prior probability corresponds to the language model itself: the more frequent a word is used, the more likely is to be the intended write of the misspelled word. In our case, we approximated the apriori probability of a correct word with the relative frequency of that word.

The term $P(\text{word}|\text{correct spelling})$ can be seen as a measure of the distance between the two words used as arguments. To express this distance one can start from any of the metrics described above. As the higher the distance between the words X and Y, the lower the likelihood of X being the corrected version of Y, we conveniently consider:

$$P(\text{word}|\text{correct spelling}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{edf(\text{word}, \text{correct spelling})}{2\sigma^2}\right) \tag{3}$$

where $edf(\cdot,\cdot)$ is an edit distance function. Any normalized distance can be used instead of the Gaussian from (3). The Gaussian without the scaling factor $(\sigma\sqrt{2\pi})^{-1}$ frequently appears in other classification approaches, *e.g.* for building vector spaces when using the kernel trick [11]. The appropriate value for σ is set empirically. Obviously, $P(\text{word}|\text{correct spelling})$ reaches the maximal value if and only if word = correct spelling.

The denominator of eq. (2) is calculated by using the law of total probability:

$$P(\text{word}) = \sum_{cs} P(\text{word}|cs)P(cs) \tag{4}$$

where *cs* stands for correctly spelled word. In order to reduce the number of computations, as *cs* we considered only these words whose lengths are sufficiently closed to the word to be corrected.

The algorithm for suggesting spelling corrections for a given word is as follows:

`SpellCorrect(`X`,` *bow*`,` $maxDist$`,` *edf*`,` $maxSuggestions$`)`

1 $S \leftarrow$ the set of correctly spelled words *cs* from *bow* with
  $|length(cs) - length(X)| \leq maxDist$
2 **for** $cs \in S$
3 compute $P(cs|X)$ according to equations (2)–(4) and using the *edf*
4 **return** the first $maxSuggestions$ terms from S having the highest $P(cs|X)$ values

The parameters of `SpellCorrect` are as follows: X is the word to be corrected, *bow* is the bag-of-words data used to perform spelling corrections, $maxDist$ is used to limit the set of word to be considered as potential correct forms for X, *edf* is a function computing the

edit distance between two words, and *maxSuggestions* limits the number of suggestion the algorithm returns. Note that one could use the Maximum a posteriori principle [1] and return only one result from this procedure, but in practice multiple results are more useful.

When implementing the `SpellCorrect` algorithm, one can improve the resulted code by skipping the calculation of $P(\text{word})$, unless the conditional probability is required. Furthermore, when searching for cs words fulfilling $edf(cs, X) \leq maxDist$, one can stop the computation of the matrix D when $D[i, j]$ exceeds $maxDist$ for both edit distance functions considered in sections 2.1 and 2.2.

4 Experimental results and future works

4.1 Experimental results

The above described algorithm was implemented in C# under NET Framework 4.0. The documents we used were text collections in the bag-of-words form as provided by [12]. From the five text collections we used the one corresponding to New York Times news articles. It contains 300000 documents, 102660 distinct words and a total of approximately one million words.

The best value for σ in eq. (3) was empirically found to be 0.1. As *edf* parameter in `SpellCorrect` algorithm we considered the restricted edit distance, based on the discussion on Levenshtein and general Damerau-Levenshtein distances from section 2.2. Table 1 gives the first $maxSuggestions = 3$ spelling suggestions for few words we tested, and we set $maxDist = 5$.

Misspelled word	Spell suggest. (cs)	Conditional prob. (eq. 2)	Likelihood (eq. 3)	Apriori prob. $P(cs)$	Restricted edit dist.
speling	spelling	0.82460	$7.69 \cdot 10^{-22}$	$2.04 \cdot 10^{-5}$	1
	spewing	0.17539	$7.69 \cdot 10^{-22}$	$4.33 \cdot 10^{-6}$	1
	spending	$1.03 \cdot 10^{-64}$	$5.52 \cdot 10^{-87}$	0.00035	2
hotal	total	0.50641	$7.69 \cdot 10^{-22}$	0.000277	1
	hotel	0.49358	$7.69 \cdot 10^{-22}$	0.000270	1
	local	$8.08 \cdot 10^{-66}$	$5.52 \cdot 10^{-87}$	0.000617	2
peice	price	0.46735	$7.69 \cdot 10^{-22}$	0.00047	1
	peace	0.32102	$7.69 \cdot 10^{-22}$	0.00032	1
	piece	0.20833	$7.69 \cdot 10^{-22}$	0.00021	1

Table 1: Spell suggestions for some test words provided by the the proposed algorithm.

Obviously, changing the documents one uses to model the prior probabilities $P(cs)$ might change the order of spelling suggestions: thus, “hotel” might get a better rank than “total” when the misspelled “hotal” is given as input. Based on the provided data, this is the best one can do. Improving the quality of the spelling suggestions might arise from a list of frequently misspelled words and/or from considering the context.

The theoretical motivation of the method and the test results show that the proposed algorithm has a suitable behavior for a spell checker system and produces expected results.

References

[1] Duda, R.O., Hart, P.E., D. G. Stork, D.G., *Pattern Classification* (2nd Edition), Wiley-Interscience, 2001.

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

[3] Withum, T.O., Kopchick, K.P., Oxman O.I., *Modified Levenshtein distance algorithm for coding*, United States Patent No. 7664343 B2, 2010.

[4] Xiao Sun, Ren, F.,Degen Huang, *Extended super function based Chinese Japanese machine translation*, International Conference on Natural Language Processing and Knowledge Engineering, 1-8, 2009.

[5] Wagner, R.A., Fischer M. J., *The String-to-String Correction Problem*, Journal of the ACM **21** (1974), 168-173.

[6] Gusfield, D., *Algoritms on String, Trees, and Sequences*, Cambridge University Press, 1997.

[7] Needleman, S.B., Wunsch, C.D., *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, Journal of Molecular Biology **48 (3)** (1970), 443-453.

[8] Sellers, P.H., *On the theory and computation of evolutionary distances*, SIAM Journal on Applied Mathematics **26 (4)** (1974), 787-793.

[9] Damerau, F. J., *A technique for computer detection and correction of spelling errors*, Communications of the ACM, **7** (1964), 171–176.

[10] Feller, W., *An Introduction to Probability Theory and Its Applications*, Willey, (1968).

[11] Jäkel, F. and Schölkopf, B., Wichmann, F. A., *A tutorial on kernel methods for categorization*, Journal of Mathematical Psychology **51** (2007), 343-358.

[12] Frank, A., Asuncion, A., *UCI Machine Learning Repository* [http://archive.ics.uci.edu/ml], Irvine, CA: University of California, School of Information and Computer Science, 2010.