Spelling correction using N-grams

David Sundby
Lund Institute of Technology, Sweden

david.sundby@gmail.com

Abstract

This article presents a spelling correction model prioritizing word suggestions according to the natural context of the sentence they are targeted. The architecture of the developed solution are described, the word suggestion algorithms used are depicted. The data used to allow this model to work are presented along with a comparison with a more common type of word correction used in everyday applications.

1 Introduction

Nowadays spelling correction is widely used in a growing set of applications. As processing capacity has increased both on Personal Computers and network computers automatic spelling correction has become a matter of course. Spelling correction has become more and more automated over the last decade, and the average user of computers is aware of that such spelling correctors are build into applications i.e. office suites. What the average user is not aware of is how the correction system operates, and what model it follows.

Most common spelling checkers use a model that provides probabilistic matrices of what keyboard character A the typist meant to press when wrongly spelling a word by pressing character B. What this model for spelling correction does not take into hand is the words in the context of the sentence it is located in. This means that using such a spelling correction model may force the system to provide high prioritized word suggestions that would actually contribute in the building of bad sentences.

The goal of this project is to build a spelling checker that suggests words and prioritizes them according to the context of the sentence the badly spelled word is located in. By providing this kind of prioritizing the words with the highest priority in the suggestion list will also be the words that are most natural in the context of a correctly built sentence.

The rest of this article is structured as followed: We will start by describing the architecture of the spelling checker before giving a brief description of the word suggestion algorithms used. Then we will dig deeper into what kind of data that we use for prioritizing the suggested words and how we use it. In the next section we will present an experiment to compare the implemented spelling correction application against a commonly used one using the prioritizing model described above. In the preceding section we will present the results before finally concluding.

2 Architecture

Before describing how prioritizing is done we will briefly describe the architecture of the implemented spelling correction application. The implementation can be divided into three main steps, executed in the same order; initialization, indexing and spelling correction.

Initialization: In the initialization phase all the data is loaded into memory. This data includes the dictionary and the analysis data used for prioritizing. In worst case scenarios the number of lookups done by the spelling checker is up to 100-500 dictionary and analysis data lookups for

each word misspelled. The data is mainly loaded into memory to be able to decrease the execution time of lookups made in the dictionary and in the analysis data.

Indexing: When all the data has been loaded into memory indexes are made for both the dictionary data and the analysis data. The indexes consists of two character indexes (26x26 indexes) all having an integer value. Using the dictionary as an example this integer value represent the position in the dictionary where the first occasion of a word having e.g. the character 'a' as the first character (single character index) or the first occasion of a word having e.g. the two characters 'ab' as the first and second character (double character index). Providing such indexes allows lookups in a small partition of the data instead of traversing the whole data set x-number of times in worst case scenarios.

Spelling correction: When all data has been loaded into memory and the indexes are made, the actual spelling correction application is launched. The interface consists of a text area similar to ordinary text editor's i.e. MS Word. Words within the text area are looked up against the dictionary. Words that are not seen in the dictionary are marked for word suggestion. Word suggestions are returned using four word suggestion algorithms, namely *insertion*, *deletion*, *substitution* and *reversal*. These algorithms will be explained in the next section. Each method implemented as the four algorithms returns a set of word suggestions, which after all algorithms has finished are combined. Together the combined results produce a list of unique words representing the list of word suggestions for a badly spelled word. When all word suggestions have been collected the words are prioritized using the analysis data. The analysis data consists of frequencies of bigrams and trigrams extracted from a small corpus[1] consisting of a collection of excerpts from financial papers. If the suggested word, set into the context of the sentence using bigrams and trigrams, does not give frequencies higher than 2, a unigram frequency is used instead. In such cases the application collects the unigram frequency of the word suggestions and prioritize those with the highest frequency first. The unigram frequency is simply the number of occasions of a given word located in our test corpus. We will explain all the data extractions made on the test corpus in section 4.

[1] http://www.daviddlewis.com/resources/testcollections/reuters21578/

3 Word suggestion algorithms

In this section we will explain the algorithms used to collect word suggestions for wrongly spelled or wrongly typed words. In the implementation of the spelling correction application we used four word suggestion algorithms; insertion, deletion, substitution and reversal. If you already are familiar with these algorithms you can skip to section 4.

3.1 Insertion

Insertion is an algorithm that corrects wrongly spelled words missing a character. Using insertion we insert each character in the alphabet, which in our case are A-Z, in each character position of the wrongly spelled or wrongly typed word, see figure 1.

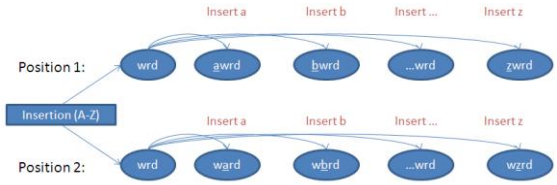

Figure 1: Insertion

From the figure the wrongly spelled word is "*wrd*". Using insertion we start by inserting the character 'a' in position 1 of the word, then we insert the character 'b' in the same position and all the way to 'z'.
For the second row we do the same for character position 2 of the word, we insert the character 'a' in position 2, then the

character ‘b’ and all the way to character ‘z’. For each character insertion the prepared word will be looked up and inserted into the word suggestion list if and only if the word is seen in the dictionary.

3.2 Deletion

Deletion is an algorithm which deletes characters from a misspelled word. Using deletion we delete one character at a time and look up the prepared word in each deletion step to see if the word is seen in the dictionary. In figure 2 we have the word “*worrd*”, which in deletion step 3 and 4 prepares the correct word “*word*”.

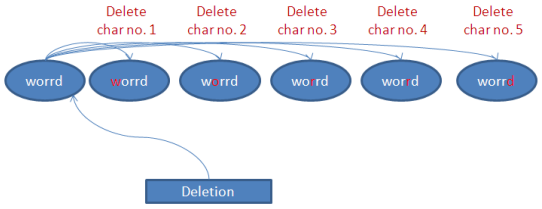

Figure 2: Deletion

As for insertion each word look-up seen in the dictionary using deletion will be inserted into the word suggestion list.

3.3 Substitution

Usually correct typos made by the typist. If the typist meant to e.g. press the character ‘g’ a common mistyping is to instead press one of the characters physically close to ‘g’ on the keyboard, e.g. by pressing ‘f’, ‘v’, ’b’ ,’h’ and so on. Using substitution we substitute each character making up the word, one at a time by a character from the alphabet, in our case starting with ‘a’ and ending with ‘z’, see figure 3.

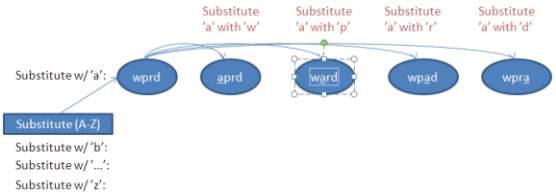

Figure 3: substitution

When substitution has finished all word suggestions are inserted into word suggestion list.

3.4 Reversal

Reversal is used to correct misspellings where the typist has typed a character at the wrong location of a word, e.g. by spelling the word “*typist*”, wrongly “*tpyist*”. Using reversal we change the location of all characters in the wrongly typed word one at a time.

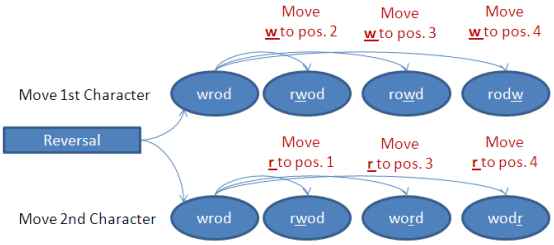

Figure 4: reversal

From figure 4, we start by moving the first character (‘w’) to character position 2, then to character position 3 and all the way to the last character position in the word. Then we do the same for the second character until all characters has been interchanged. Finally, all suggestions are inserted into word suggestion list.

4 Data analysis

For our data analysis we used a small financial corpus consisting of ca. 4 million words. From this corpus we managed to extract approximately 20 000 unique words found in the dictionary. For each of these unique words we did three data extractions from the corpus, unigram frequency, bigrams and trigrams. What we here call trigrams is in reality a set of two bigrams, we will explain this later on.

Unigram: The first data extraction we made was collecting the unigram frequency for each unique word from the test corpus. The unigram frequency is simply the number of occasions of a specific word from the corpus used. We use unigrams for prioritizing when the frequency of the bigrams or the trigrams in the context of the sentence, gives frequencies lower than two.

Bigrams: The second data extraction we composed was the collection of two sets of bigrams related to all unique words extracted from the corpus. For each

unique word (n) we located each occasion of it in the corpus and recorded the word in front of it (n-1) and the word behind it (n+1). E.g. for the sentence "*the man walked down the street*", where *walked* is the unique word, the bigram from the first extraction would be "*man walked*" and the bigram from the second extraction would be "*walked down*". Each unique bigram n-1 and n+1 are counted and the frequency is stored as analysis data for the spelling checker. This data is used for prioritizing word suggestions in the natural context of the sentence were the word correction is located.

Trigrams: A more precise name for it in our context would be frequencies of two bigrams added together. In the bigrams extractions described above the frequencies of two different types of bigrams are collected, n-1 and n+1. The frequencies from these two bigram data sets are added together in our spelling correction application. E.g. for the sentence "*the man walked down the street*", where "*walked*" is the unique word a "real" trigram frequency would be the number of occasions of the string "*man walked down*" from the test corpus. Instead we use the frequency of the bigram "*man walked*" (n-1) plus the frequency of the bigram "*walked down*" (n+1). Bigram frequencies i.e. these put together are what we call a trigram to differentiate it from the bigrams. This means that the bigram (n+1) only strengthens or in some cases make the difference of whether or not to use unigram frequency for prioritizing.

5 Experiment

As with most projects it is important to make a point and compare it with solutions already made to similar problems. To collect this data we have decided to compare our solution with a spelling correction application known to most of us, which is that one built into Microsoft Word 2007. In this section I will present the data and explain the experiment itself before presenting the results and a discussion of the experiment in section 6.

Since the test corpus that we used in the project is a collection of excerpts from financial papers, we found it most natural to include financial papers in our experiment as well. We used excerpts from two articles from an online financial paper, forbes (2009). To these collected texts we manually added 100 spelling errors. The reason why we chose to manually add spelling errors was mainly for two reasons;

First of all, we found it difficult to find blogs and articles which included spelling mistakes. An explanation for this can be that most modern tools used to write articles or blogs have a built in spelling checker. Another explanation can be that the search engine[2] that we used to browse for relevant texts filters out web pages that include bad language, i.e. spelling errors and typos.

Second if we found any articles or blogs which included a few spelling errors many of these were not relevant for our spelling checker, because of its restrictions. E.g. for the wrongly spelled word "*typissst*" our spelling checker would not find the correct word "*typist*". This would require the word suggestion algorithm *deletion* to delete two characters ("ss"). Since our implementation only handles transformation on one level, which for the deletion algorithms case would be one character deletion, such spelling errors would not get any word suggestions. Anyhow for each of these 100 spelling mistakes we collected the disambiguation accuracy (DA) from 1 to 5 (DA1-DA5) for both the MS words spelling checker and for our own. DA1 is true only if the correct word is located as priority one, DA2 is true if the correct word is located as priority one or priority two and so on. E.g. if we typed the wrongly spelled word "*typsit*", while we meant to write "*typist*" and the spelling correction return a list of suggestions with "typist" in priority three, the DA1 and DA2 would be

[2] http://www.google.com

false while DA3, DA4 … , DAn would be true. The goal is of course to get the correct word as high as possible up in the priority list. The results and comparison are presented in the next section.

6 Discussion and Results

From *table 1* we can see the results from the experiment described in section 5. The Disambiguation Accuracy 1-5 is listed for MS Word and for our implementation using bigrams (n-1) and using trigrams (two bigrams added together (n-1 and n+1)), see section 4. Between each DA columns, a frequency column is in place. This column tells us how many percent of the prioritizing were done using unigram frequency. We recall that when the bigram or trigram frequency is lower than 2, a unigram frequency is used for prioritizing instead.

From the table we can see that our implementation gives better results than MS Word's. Using our model 95 percent of the word suggestions has the correct word as the highest priority (DA1) against only 67 percent for MS Word. As we move down to lower priorities (DA2-DA5) we can see that the DA for the two spelling checkers balance out. From DA5 we can see that MS Word have a score of 94 percent, which means that the correct word were not present in the suggestion list for 6 percent of the misspelled words. This does not mean that MS Word was not able to find the correct word using its correction algorithms, but that it prioritized it lower than position 5 (MS Word only includes the five highest priorities).

	DA1	No. of freq	DA2	No. of freq	DA3	No. of freq	DA4	No. of freq	DA5	No. of freq
MS Word	67 %	-	82 %	-	88 %	-	93 %	-	94 %	-
SpellingChecker (bi)	95 %	57 %	99 %	56 %	99 %	56 %	99 %	56 %	100 %	56 %
SpellingChecker (tri)	95 %	51 %	99 %	51 %	99 %	51 %	99 %	51 %	100 %	51 %

Table 1: Disambiguation accuracy and frequency from experiment

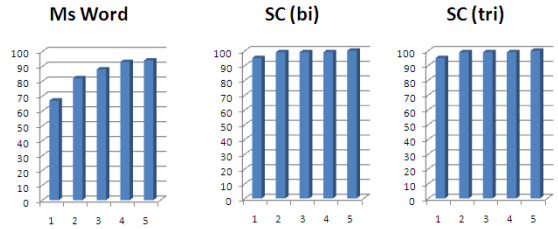

Table 2: Column chart of Disambiguation Accuracy

Even though the results collected are quite good, it is important to remember that the two implementations use completely different prioritizing models. Spelling checkers i.e. Microsoft Word's use a prioritizing model which is similar to the one described in Kernighan et al. (1990). This model builds on a model referred to as *a noisy channel model*. This model basically depends on that the user of the system knows which word that he or she are writing but that some "noise" is added during the word creation in the form of typos or spelling errors. Typos here refers to that the typist of the system presses the wrong button on the keyboard during the building of a word. The system also has a probabilistic matrix of all probabilities of what keyboard character A the typist meant to press when wrongly spelling a word by pressing character B. The most obvious prioritizing strategy here is that the characters closest to a character A has a higher priority than one in a different part of the keyboard. This model works well for the most "common" spelling errors, which include hitting the wrong character by pressing one of its neighbors instead. From the experiment we included both common and not so common spelling errors. The most common spelling errors, i.e. pressing a neighbor character when misspelling a word, gives good result for both spelling checkers. When it comes to not so common spelling errors, MS Word fails. Here are some examples from the experiment where MS Word failed to compete with our model:

> **Example 1:** Wrongly spelled excerpt "*contracts oir the*" where *oir* is the wrongly spelled word for "*or*". MS Word set the correct word "*or*" as priority three in the

word suggestion list, our model set "*or*" to priority one.

Example 2: Wrongly spelled excerpt "*control orve the*" where *orve* is the wrongly spelled word for *over*. MS Word did not include "*over*" to the word suggestion list (D1-D5), our model set "*over*" to highest priority.

From these two examples we can see where the model used by MS Word fails. Basically this is because such spelling errors are not of those this model builds on. It builds on favoring the most common spelling errors and does not take into hand the natural context of the sentence where the suggested words are meant to be inserted.

7 Conclusion

Looking at the results from section 6 we can conclude that the results from our spelling checker has good results compared to those of MS Word's. The difference between the two models is that for our solution we use corpus data analysis to create priority lists with words that are natural in the context of the sentence with highest priority. MS word uses a "noisy channel" like model. This model basically use probabilistic matrixes consisting of probabilities of which button on the keyboard the typist meant to press when constructing a misspelled word by pressing a certain "wrong" character.

By using corpus analysis data, as used in our solution, we have a higher possibility of having words high in the priority list that are natural in the context of the sentence. On the other side this depends on what kind of spelling mistakes the user has typed. Using the "noisy channel" like model, used by MS Word, most common spelling errors are recorded and usually have the correct words high in the priority list. This leads us to the question whether or not it is really necessary to have a different model, one that is natural to the context of the sentence the word suggestions are supposed to be inserted into.

Using the corpus analysis model also give high priority to the correct word for spelling errors that are "not so common". Put in another way spelling errors including keyboard characters that are far away from the character the typist should have pressed. Since the corpus data model brings equally good results for the most common spelling errors and expands with also giving good results for not so common spelling errors, this model is preferable as long as we can keep its execution time within acceptable limits. This model also gives us assurance that the words that are natural in the context of the sentence are given high priority.

8 Acknowledgements

The author would like to thank Pierre Nugues for supervision, motivation and constructive contributions to the project.

9 References

forbes, 2009, http://www.forbes.com/feeds/ap/2009/12/05/general-us-obama-economy_7185836.html

forbes, 2009, http://www.forbes.com/feeds/reuters/2009/12/05/2009-12-05T201531Z_01_N06186635_RTRIDST_0_ECUADOR-OIL.html

Kernighan D. Mark, Church W. Kenneth, Gale A. William, 1990, *A spelling correction Program Based on a Noisy Channel Model* , International Conference On Computational Linguistics - Proceedings of the 13th conference on Computational linguistics - Volume 2, pp 205-210