CENG 463: Introduction to Natural Language Processing
Project Survey
Project Topic: Spelling Correction

Ayça ACAR - 1249853
Erhan SUNAR - 1250745
June 1, 2004

OUR TOPIC

Spelling correction is a very important concept for any system that processes text. Creation of the text is prone to errors, people make typing errors or they sometimes ignore the correct spelling of the word. Therefore they need spelling corrector systems. Another issue where spelling correction is highly necessary is optical character recognition systems. These systems often make errors during recognition due to bad quality of input, changes in fonts or insufficient training. Today, many spelling correction functions are available in many languages like English, French, etc. However there are not many for Turkish.

In this paper we will present the main approaches to the problem of spelling correction. At first we will define the spelling correction problem. Then we will state the general approaches to the problem. After that we will discuss the spelling correction in agglutinative languages such as Turkish and Finnish.

Definition 1 (Spelling Correction) Du and Chang [4] defines the spelling correction problem as follows: From a set of known words dictionary, find those words that most resemble a given misspelled character string.
GENERAL APPROACHES

The simplest method for implementing a spelling corrector is to use an interactive spelling checker. Each time the spelling checker finds a misspelled word, the user indicates its correctly spelled form and the spelling checker remembers this selection. However, this approach may have undesirable results in many cases.

Another simple approach is to have the common misspelled words and their correct spellings in the dictionary. For instance, "greatfully" may be stored as the misspelled form of "gratefully". Whenever the user misspells a word, it will be replaced by its correct form if the misspelling exists in the dictionary. This method is not very useful in practical life since it is not possible to keep all the misspellings in the dictionary.

Since these basic methods are not of much use, other techniques that take the nature of misspellings into consideration have been proposed. It was found out that 80 percent of all spelling errors are due to [5]:

1. transposition of two letters
2. one extra letter
3. one letter missing
4. one letter wrong

We can simply calculate the possible number of a word. Assuming an alphabet of 29 characters and a word with n letters, the four typing errors could cause the following numbers of errors:

\[
\begin{align*}
\text{One wrong letter} & \quad 28n \\
\text{One extra letter} & \quad 29(n + 1) \\
\text{One missing letter} & \quad n \\
\text{Transposition} & \quad n - 1
\end{align*}
\]

DEC-10 spelling corrector system is one of the correctors that take these four rules as a basis. According to its algorithm, first the dictionary is checked to see if the token is in the dictionary. If it is not, a list of words that might be the correct spelling of the given token is prepared. To construct this list, every word in the dictionary that would result in the given misspelled token when one of these four rules is applied is marked as a candidate. For example, to check if the problem is one extra letter, each character is deleted one at a time and the resulting token is searched in the dictionary. This
method is quite successful in simple errors, but it cannot correct more complex spelling errors that result from combinations of the four rules. Another problem of this method is computational complexity.

A method proposed to decrease the computational complexity to this algorithm is to use hashing[6]. The idea behind this approach is to store every word $x$ in a hash table $|x| + 1$ times where $|x|$ denotes the length of $x$. Each time one of the letters of $x$ is omitted. This form of storage will make it easy to handle the errors that are based on the four rules. If $w$ is the typed misspelled word and it is the word $x$ with a missing letter, it will match one of the $|x| + 1$ forms of $x$ and the error will be corrected. If $w$ has an extra letter, deletion operations will be applied to $w$ and it will match $x$ itself. The other two rules will be similarly handled.

Another method suggested is the use of longest common substrings. The misspelled token is compared to every word in the dictionary and the word that has the longest common substring with it is chosen as the most probable one. This is a very expensive algorithm.

Other methods include considering the keyboard layout and phonetic properties of the words. While using keyboard layouts the programmer assigns some probabilistic values to keyboard buttons and tries to guess the error type that the user made. For example if the user writes "kSABA", this method can easily understand that the user accidentally pressed "CAPS key" when trying to press "a key" and correct it as "kasaba". The latter one uses the phonetic nature of the language to test if the word is correct and correct it if it is uncorrect. For example the letter ˘ g cannot be at the beginning of any word in Turkish.

Also there are methods[7] which uses contextual information of text. These methods uses some heuristics to get the meaning of the sentence and then corrects the word accordingly. These methods are very successful at some specific topics compared to methods. For example the writing of words "peace" and "piece" or "quiet" and "quite" are very similar to each other. They can be written instead of the other. The regular spelling correctors cannot able to correct these type of errors however context dependent correctors can easily correct them. For example "a peace of bread" or "a piece of bread".
SPELLING CORRECTION IN AGGLUTINATIVE LANGUAGES

Considering the definition, spelling correction may be seen as an easy task for languages like English or French. However, this problem differs for languages like Turkish and Finnish since lexical forms are generated in a different way in these languages. According to morphological classifications, natural languages like Turkish, Finnish and Hungarian are in a class called "agglutinative languages". In this group of languages, words are combinations of several morphemes. The words consist of a root and several affixes combined to it in order to modify or extend its meaning. This word structure makes spelling correction different than other languages because in agglutinative languages the list of actual words are much larger than the entries in the dictionaries, due to productive word formation by derivational and inflectional affixations and the method of choosing the word that most resembles a given word from a dictionary does not result in very accurate results.

Thus these type of languages must be handled different than others types of languages. There are two type of approaches to this problem in agglutinative languages:

The first approach is the Error-tolerant Finite State Recognition [1]. Error-tolerant recognition with a finite state recognizer (FSR) can be defined informally, as the recognition of all strings in the regular set (accepted by the FSR), and additional strings which can be obtained from any string in the set by a small number of unit editing operations of insertion, deletion, replacement, and transposition of adjacent symbols.

The error-tolerant recognition needs an error metric to measure the how much a string differ from another. The edit distance calculates the minimum number of operations to convert one string to another:

**Definition 2 (Edit Distance)**

\[
\begin{align*}
ed(X[i], Y[j]) &= \text{if } x_{i+1} = y_{j+1} \\
ed(X[i+1], Y[j+1]) &= \begin{cases} 
ed(X[i], Y[j]) & \text{if } x_i = y_{j+1} \text{and } x_{i+1} = y_j \\
1 + \min \{ \ned(X[i-1], Y[j-1]), \\
\ned(X[i+1], Y[j]), \\
\ned(X[i], Y[j+1]) \} & \text{otherwise} \\
\end{cases}
\end{align*}
\]
As we see above this method constructs an automata and searches with an error threshold.

The second solution alternative, is to use a method that is based on two-level morphology and a search method [2]. In this method, in order to overcome the difficulty with agglutinative languages, the problem should be handled as two subproblems:

1. All the roots from the dictionary that can be a candidate root for the misspelled word should be determined.

2. All possible words that resemble the given word should be generated in a systematical way.

At this point, one of the key points is how to measure the resemblance ratio. In other words, how shall we choose the word that resembles the given misspelled word the most? Some metrics have been proposed for this problem and the most widely accepted ones are edit distances (definition 2) and q-grams.

A q-gram is a substring of length q. The q-gram distance is based on the counting occurrence of same q-grams in two words. Most used q-grams are
bi-grams and tri-grams.

If we try to compute the edit distances with the word to correct with all the words in the dictionary it can consume lots of time. However a static list of all bi-grams can be precomputed and stored in a database. Each word containing that bi-gram has one at that column and zero otherwise. By intersecting the columns in the dictionary (the columns that have bi-grams in the word to be corrected) we can narrow down possible roots. Then the rest can be again narrowed by the edit distance algorithm.

After determination of root the second phase (generation of candidate words) begins. The words are generated and compared by the edit distance metric and sorted by relevance.

**CONCLUSION**

As a result spelling correction is a necessary part in any word processing program, in an optical character recognition program or any other program that uses text. There are several approaches to this problem. All of them has some cons and pros. The usage of them is application dependent and it is harder to implement in agglutinative languages.
References


