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

Ayça ACAR - 1249853
Erhan SUNAR - 1250745
June 14, 2004

Contents

1 The Spelling Correction Problem 2

2 Approach 2

3 Notation 3

4 Implementation 3
   4.1 Edit Distance Metrics 3
       4.1.1 Edit Distance with Linea Traces 3
       4.1.2 q-grams 3
   4.2 Recognizing and Generating Strings in the Language 4
   4.3 Determining the Root 4
       4.3.1 Root Determination with q-grams 4
   4.4 Generating Candidate Words from Given Root 4
       4.4.1 Getting Solutions from Corrupted Roots 5
       4.4.2 Getting Solutions from Corrupted Affixes 5

5 Results 6

6 Conclusion 6
Abstract

In this project we implemented a program that analyzes the given incorrect word and suggests alternative correct forms to this word. This project is especially designed for Turkish. Turkish is an agglutinative language thus it needs special approaches different than other languages such as English. In such languages a simple lexicon may be enough to correct word forms. In this project we have mostly used the papers of Kemal Oflazer ([1], [2], [3]).

1 The Spelling Correction Problem

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.

Here the word “resemble” is tricky. To be more scientific we must define the resemblance. This is also another hard issue to calculate the distance between two strings. The most common ones are q-gram and linear traces. We will return to these topics in the following sections.

2 Approach

Since Turkish is an agglutinative language we have to consider all the derivations from the root to the surface form of the word. Thus the notion of the dictionary here is the dictionary of known roots. Therefore our problem has two subproblems

1. Determine all the roots from the dictionary that can be the root of misspelled word

2. Generate all the possible words that resemble the given character string

3. Mark the most possible ones according to keyboard layout

In the first step we will try to determine the roots by using the string distance metrics. If we try to apply distance metrics to all the roots in the lexicon it will take too long time. Thus we applied some heuristics in this part.
In the second step we used pckimmo [8] to add affixes and generate surface forms.
In the last part we used keyboard layouts to figure out some possible common errors. Thus correct the word according to this layout
3 Notation

Our notation in this paper and in the program is as follows:
\[ X = x_1, x_2, \ldots, x_m \]
\[ Y = y_1, y_2, \ldots, y_n \]
\( X \) surface form of the incorrect word
\( Y \) candidate word
\( \Sigma \) alphabet of the language
\( |X| \) length of \( X \)
\( X[0] \) = empty string
\( X[i] = x_1, x_2, \ldots, x_i \)
\( X[i : j] = x_i, x_{i+1}, \ldots, x_j \)

4 Implementation

4.1 Edit Distance Metrics

4.1.1 Edit Distance with Linea Traces

The edit distance calculates the minimum number of insertion, deletion, replacement operations applied on the two strings to convert one to another.

**Definition 2 (Edit Distance)**

\[
\begin{align*}
ed(X[i], Y[j]) &= \begin{cases} 
ed(X[i], Y[j]) & \text{if } x_{i+1} = y_{j+1} \\
1 + \min \{ & \ed(X[i-1], Y[j-1]), \\
& \ed(X[i+1], Y[j]), \\
& \ed(X[i], Y[j+1]) \} & \text{if } x_i = y_{j+1} \text{ and } x_{i+1} = y_j \\
1 + \min \{ & \ed(X[i-1], Y[j-1]), \\
& \ed(X[i+1], Y[j]), \\
& \ed(X[i], Y[j+1]) \} & \text{otherwise} 
\end{cases}
\end{align*}
\]

\[ 
ed(X[0], Y[j]) = 1 \quad 1 \leq j \leq n \]
\[ 
ed(X[i], Y[0]) = 1 \quad 1 \leq j \leq n \]

4.1.2 q-grams

A q-gram is simply a substring of length q:

**Definition 3 (q-gram distance)** Let \( g \in \Sigma^q \) a q-gram. Let \( G(X[m])[g] \) denote the total number of occurrences of \( g \) in the string \( X \) of length \( m \). The
4.2 Recognizing and Generating Strings in the Language

To recognize and generate the strings in the language we used PCKIMMO [8]. Mainly ”generate” and ”recognize” functions of it. However we could not find a way to include PCKIMMO into our source like an API. Thus we called it with system calls. We give the input to PCKIMMO as a text file and got the output in the same way. This approach highly slowed down our program’s running time.

4.3 Determining the Root

The determination of the root is the hearth of our program because the later steps largely done with the help of PCKIMMO and straightforward. If any root has edit distance from all prefixes of the misspelled word less than the threshold $t$, then it is a candidate root.

Definition 4 (The set of all possible roots)

$$PR(X, t) = \{ r | ed(X[i], r) \leq t \text{ and } 1 \leq i \leq m \quad r \in R \}$$

4.3.1 Root Determination with q-grams

We used q-grams to decrease the number of candidate roots small enough to be determined by edit distance. We first construct our static q-gram database from Oflazer’s Turklex(which is written for PCKIMMO for Turkish morphology) program’s database. To make program faster in each program start we load this database into memory.

This database consists of the roots and bi-grams of these roots. We first take the first three bi-grams of the incorrect word. Then we union the group of words those have one of these bi-grams.

After that we use edit distance with linear traces to decrease the size of candidate roots.

4.4 Generating Candidate Words from Given Root

Here our main tool was PCKIMMO, first we try to find the corruptions in the root then in the suffixes. The determination of the corruption in the roots was very successful both in time and correction ratio (we called PCKIMMO only once). However the correction in the suffixes was unsuccessful. Because we have to call PCKIMMO several hundreds time. We could not able to get any results from this part.
4.4.1 Getting Solutions from Corrupted Roots

We define two new metrics to get solutions from corrupted roots:

Definition 5 (Prefix Dedit Distance) *The prefix edit distance between* $X$ *and* $r$ *is:*

$$\text{pred}(X, r) = \min \{ r | \text{ed}(X[i], r) | 1 \leq i \leq m \}$$

Definition 6 (Alignment Index) *The set of alignment indexes of* $r$ *in* $X$ *is*

$$\text{index}(X, r) = \min \{ i | \text{ed}(X[i], r) = \text{pred}(X, r) \}$$

When $\text{pred}(X, r) = t$, the remaining part of $X$ after alignment with root $r$ must completely occur in $Y$ after $r$ to satisfy $\text{ed}(X, Y) \leq t$.

That is $Y$ must be in the form $Y = \text{surface}(\text{concatenate}(r_{\text{ex}}, X[i+1] : m))$

After using these to generate candidate solutions we push them to PCKIMMO to check they are valid. Then we take valid ones as new candidate set. Then we use the keyboard layout to mark the most possible ones. As we see below the key “G” has neighbors ”E,R,T,D,G,C,V,B”. Thus it is more possible for anyone to press any of these keys instead of ”G” then other keys. So if we find two candidates for the word incorrect word “atora” as ”agora” and ”asora” the correct form is possibly ”agora” because ”g” is the neighbor of ”t”.

4.4.2 Getting Solutions from Corrupted Affixes

We tried to add affixes until they reached a certain threshold, to the found roots then push them into PCKIMMO. Then testing them if they are close
enough to misspelled word (with edit distance). If not pushing some new (or
already used) affixes. Then continuing like that until we find the candidates.
As we mentioned before this approach did not become very successful be-
cause PCKIMMO worked so slow and the only interaction way of PCKIMMO
was through the file system.
We used a cut-off distance not to over generate the words.

Definition 7 (Cut-off Distance)

\[
cuted(X[m], Y[n]) = \begin{cases} 
\min\{(H[i, n]|1 \leq i \leq n + t}\} & \text{if } n < t \\
\min\{(H[i, n]|n - t \leq i \leq n + t}\} & \text{if } t < n \leq m \\
\min\{(H[i, n]|n - t \leq i \leq )\} & \text{if } m < n \leq m + t \\
(n - m) & \text{if } m + t \leq n 
\end{cases}
\]

\[
H(i, j) = ed(X[i], Y[j]) \quad \text{where} \quad 1 \leq j \leq n, \quad 1 \leq i \leq m
\]

5 Results

The results are on the results.txt

6 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. In this project we have
implemented the spelling correction algorithm based on two-level morphol-
ogy.
References


