CS6370: Natural Language Processing

Spell Check Assignment Report

Indian Institute of Technology Madras

Chinni Chaitanya, EE13B072
ee13b072@smail.iitm.ac.in

Venkatesh Maligireddy, EE13B041
ee13b041@smail.iitm.ac.in

Swaroop Kotni, EE13B030
ee13b030@smail.iitm.ac.in

Pronnoy Noel, EE13B029
ee13b029@smail.iitm.ac.in

September 22, 2016

Contents

1 Introduction

This assignment attempts to implement *spell checker and correction* programs for text, which will correct the erroneous word[1] present in it. The assignment is categorized into the following three parts,

- **Word checker**, a program which checks and corrects standalone erroneous words[2].
- **Phrase checker**, a program which examines phrases and corrects the erroneous words. We assume *phrases* contain about 5 words.
- **Sentence checker**, a program which examines sentences and corrects the erroneous words. We assume *sentences* contain about 30 words.

In all the three cases, we assume that there is only **one** erroneous word.

1.1 Directory structure

The submission contains the following folder structure, ... Add folder structure ...

2 Data generation

2.1 Corpa used for data generation

2.2 Data generated using these corpa

3 Word checker

The *word checker* aims to implement a program to inspect the standalone erroneous word given as input, and come up with a set of possible candidates with their probability of correctness and ranking. The probability of each word is calculated both ***statistically (Bayesian probability)*** [1] as well as ***phonetically (editex)*** [2]. All the main ideas regarding ranking and probabilities are taken from [1] and [2].
This whole implementation can be broken into the following four sub-tasks,

- **Task-1** Generation of candidate set for the given erroneous word
- **Task-2** Calculate the Bayesian probability for each word in the candidate set
- **Task-3** Calculate the phonetic probability for each word in the candidate set
- **Task-4** Combine both Bayesian and phonetic probabilities and rank the words in the candidate set

[1] In phrases and sentences, the erroneous word need not be misspelled but might be contextually incorrect.
[2] The word given might be correct word also.

3.1 Generation of candidates

The aim of this task is to generate the set of candidate words or the candidate set for the erroneous word given. A crude way to do this is to generate all possible strings for the given erroneous word by adding, deleting, substituting and reversing the letters in the given word and check whether that string exists in the dictionary. This works, but is computationally intensive. Hence, we need to intelligently store the words in the dictionary such that, the search doesn't take much time. Instead of the additional step to generate the possible strings and check if they exist in the dictionary, we could intelligently fetch the words from the dictionary itself by some measure. One such measure is *Levenshtein distance*. The following sub-sections attempt to address these ideas.

3.1.1 Levenshtein distance

3.1.2 Burkhard-Keller Trees or BK-Trees

3.1.3 Trie structure

3.1.4 Performance of BK-Trees and Trie

3.2 Ranking the candidates statistically

3.2.1 Bayesian probability

3.3 Ranking the candidates phonetically

The aim of the phonetic ranking is measure how similar the two given words sound, when pronounced. There are many methods which attempt to calculate the phonetic similarity. For example, *Soundex, Phonix, Q-gram method, Agrep, Idapist* and *Editex*. Based on the results of performance of each method and the simplicity of implementation as per [2], we chose to implement the ***Editex*** method for the phonetic ranking.

3.3.1 Editex

Editex is the phonetic distance measure of two words, similar to the *Levenshtein distance*. **Lesser the phonetic distance, greater is the phonetic similarity between two words**. This method is an improved implementation of *Soundex* and *Phonix*. All the letters in the English literature are classified into groups[3]. Each group contains letters, which generally sound similar in words when pronounced. *Editex* takes into account the silent letters like `'h'` and `'w'` also unlike Soundex or Phonix. By comparing the letters in the candidate word and erroneous word, the phonetic distance is computed, which is similar to the Levenshtein distance. This computation is implemented using the concept of *dynamic programming*.

[3]Each letter can be in multiple groups.

3.4 Combined ranking of candidates using both statistical and phonetic ranking

The probabilities obtained from both the Bayesian and Editex are used to compute the final probability for the candidate. This operation is a function of the probabilities of both Bayesian and Editex.
We must give importance to both the probabilities since we should prefer the candidate which is statistically very frequently occurred (measured with Bayesian probability) and also sounds similar (phonetic probability) to the erroneous word.
If a candidate has very high phonetic probability but low Bayesian probability (or vice-vesra), and another candidate which has average phonetic as well as Bayesian probabilities, intuitively, we prefer the second candidate. Hence, to account for the best candidate, **we computed the final probability of each candidate as the product of it's Bayesian and phonetic probabilities**.
The final ranking of candidates is done in the descending order of their corresponding final probabilities.

3.5 Observations

4 Phrase and Sentence checker

The *context checker* attempts to detect and suggest a correction to an erroneous word present in a phrase or sentence. Here, we cannot assume that the erroneous word is misspelled. It might be contextually incorrect like for example, the word **peas* in the sentence, *"He was awarded the Nobel peas prize"*. The word *peas* is not misspelled. But it is contextually incorrect. So the aim of this *context checker* is to detect the erroneous word *peas*, and suggest a correction, *peace*.
Also, the sentence might contain correct words which are combined for example, consider the sentence, *"Standing in the halloffame"*. Here **halloffame* is the erroneous word but is combined by correct words. Our context checker must detect it and suggest a correction *hall of fame*.
The suitable candidate is predicted using the *Bayesian hybrid* method and using *collocations* as described in the paper [3].

4.1 Text tokenization

This attempts to tokenize the input sentence and detect the words like *halloffame* and split them into *hall*, *of* and *fame*. But if the split is not possible or if the word already exists in the dictionary (for example *cupboard*), tokenization will not alter the word. This is coded using the concept of *dynamic programming*.

References

[1] Mark D. Kemighan , Kenneth W. Church, William A. Gale. *A Spelling Correction Program Based on a Noisy Channel Model*

[2] Justin Zobel, Philip Dart. *Phonetic String Matching: Lessons from Information Retrieval*

[3] Andrew R. Golding. *A Bayesian hybrid method for context-sensitive spelling correction*