Spell Check Competition

CS6370: Natural Language Processing

CS14B044 M.Uday Theja
CS14B051 S.Sai Teja Reddy

March 20, 2018

1 Introduction

In this assignment ,we implement three modules - word spell check, phrase spell check and sentence spell check. The only difference being that in the later two we are provided with some context for spelling correction.
The objective of this assignment is to provide suggestions for spelling correction as closely as a human reader would do.

2 Methodology

2.1 Resources

2.1.1 Corpus

- Brown Corpus
- Reuters Corpus

2.1.2 Dictionary

- http://norvig.com/ngrams/enable1.txt
- http://www.bragitoff.com/wp-content/uploads/2016/03/Dictionary-in-csv.zip

2.1.3 Bi-Gram Frequencies

- Frequent n-grams data based on Corpus of Contemporary American English - https://www.ngrams.inf

2.1.4 Homonyms

- http://myenglishgrammar.com/list-25-homonyms.html
- http://usefulenglish.ru/writing/homonyms-short-list
- http://www-01.sil.org/linguistics/wordlists/english/homophones/homophones103.txt
- https://www.thoughtco.com/homonyms-homophones-and-homographs-a-b-1692660

2.2 Preprocessing

2.2.1 Dictionary

We took the union of sets of words present in the two resources mentioned above. We included a word in our dictionary only if it was present in either WordNet or PyEnchant's English dictionary.
The words were stored in a hash-map as unique keys to reduce probe time.

2.2.2 Corpus

For calculation of priors, we needed the frequency counts of words in the corpus. To reduce the probe time for a given word, the frequency counts of all the words were pre-calculated and stored in a hash-map such that the probe time is $\mathcal{O}(1)$.

2.2.3 Bi-Grams Index and Postings Lists

For finding the candidate words through bi-grams overlap method, we generated a bi-gram index map which contains all bi-grams that occur in any term in the dictionary as keys and each postings list points from a bi-gram to all dictionary terms containing that bi-gram. For a given word, the candidate words are generated in $\mathcal{O}(n)$ time where n is the length of largest postings list.

2.2.4 Word BiGrams

For a given word bigram frequency in the form w1 w2 f , we store this in the form of two hashmaps, one gives w1[w2] = f and the "reverse" hashmap gives [w2][w1] = f. The need for two such hashmaps will be clear later.

2.2.5 Homonyms

We stored a list of homonyms in a hash-map where, for a given word as the key, the value of that key is the list of homonyms for that key, such that the probe time is $\mathcal{O}(1)$.

2.2.6 Soundex

For every word in our dictionary we generated the word's soundex code. Then we stored the words in a hashmap with keys as soundex codes.

2.3 Algorithm

2.3.1 Word Spell Check

For generating the set of candidate words for a word we use three submodules.

Firstly, for a given typo word, we generate all words within edit distance 1 and edit distance 2 and add those present in our dictionary into our candidate word set.

In the second sub-module we use the bi-gram index map to traverse the entire vocabulary containing atleast one bi-gram present in the typo, sort all the matching words in descending order based on jaccard score and append the top 25 words into our candidate word set.

Finally we generate the typo word's soundex code and add all the words in the dictionary with the same soundex code to our candidate word set.
For ranking words in our candidate word set we use the following custom defined score,

$$score = \frac{\textbf{soundex-score} * \textbf{jaccard-score} * log(\textbf{corpus-frequency})}{e^{(\textbf{edit-distance}+1)}}$$

- **soundex-score:** The soundex score quantifies how similar the soundex codes of two words are, giving a higher score if the words are phonetically similar.
- **jaccard-score:** The jaccard score quantifies how similar two words are by quantifying the intersection over union of the set of k-grams of those words
- **corpus-frequency:** This number counts the number of occurrences of the given word in the corpus. A smoothing of 1.5 is added to always keep the log of this value positive.
- **edit-distance:** The edit distance between two words gives the cost to transform one word into another where the operations include deletion, insertion, substitution and reversal and the cost of each operation is here assumed to be 1.

2.3.2 Phrase Spell Check and Sentence Spell Check

We have two scenarios,

- **Typo is a misspelled word not present in the dictionary.**
  We tokenize the input phrase/sentence into a list of words. For the wrong word we

take immediate two neighbours of the word say, w1 and w2.Then from the word bigrams hashmap we get the set of words which frequently appear after w1 and before w2.We take the union of these sets and remove words with edit distance more than 3.If this set doesnot have candidates we go to the next case.Otherwise we score the words using the following score,

$$score = \frac{log(\textbf{bigram-frequency})}{e^{(\textbf{edit-distance}+1)}}$$

  - **bigram-frequency:** This the number of times the candidate word appears after the previous word or before the next word in the corpus

- **Typo is a valid word present in the dictionary.**
  We tokenize the input phrase/sentence into list of words and for every word we generate candidate words using the above method discussed in Word Spell Check and also add homonyms for each word into candidate set and generate all possible combinations of phrases/sentences using all the words in the candidate set.
  We then rank these combinations of phrases/sentences using the following score

$$score = \frac{\textbf{prob-score} * \textbf{soundex-score}}{e^{(\textbf{edit-distance}+1)}}$$

$$\textbf{prob-score} = \prod_{i=1}^{n-1} P(\frac{w_{i+1}}{w_i}) = \prod_{i=1}^{n-1} \frac{\textbf{bigram-frequency}(w_i w_{i+1}) + 0.5}{\textbf{corpus-frequency}(w_i) + |V|}$$

  - **soundex-score:** The soundex score quantifies how similar the soundex codes of two words are, giving a higher score if the words are phonetically similar.
  - **edit-distance:** The edit distance between two words gives the cost to transform one word into another where the operations include deletion, insertion, substitution and reversal and the cost of each operation is here assumed to be 1.

3 Results

All the results presented are on the intermediate evaluation validation set.
We use Mean Reciprocal Rank to score our suggestions.
On the given validation set of standalone words we achieve a MRR of 0.678
For phrases we get an MRR of 0.594.
In the case of sentence spell check we get a MRR of 0.4.
These values show improvements from the intermediate evaluation values of 0.630,0.570,0.280 respectively.
The reasons for these improvements will be discussed in the next section.
We further split the phrase and sentence inputs into two sub inputs- ones where there is a

spelling mistake and ones without one.
For phrase spelling correction,
MRR for no spelling mistake correction was 0.324.
When there was a spelling mistake the MRR increased drastically to 0.768.
Similarly for sentence spell check,
MRR for no spelling mistake correction was 0.28.
When there was a spelling mistake the MRR increased to 0.444.

4 Discussion

One major assumption we make is that a human writer usually makes a spelling mistake between similarly sounding words.If any homonyms are present in our candidate set we have to give that word a higher score.This is something we failed to implement in our intermediate submission.
To score the "phoenetic similarity" between two words we use the edit distance between their respective soundex codes. Integrating the soundex score into our ranking metric helped in improving our MRR scores.
There is one form of spelling correction which our code doesnot handle, i.e. usage errors. For example we disregard the cases where 'between' and 'among' are used in place of each other. This is because the bigram jaccard similarity is 0 and the edit distance is 7.
Also, there is a tradeoff between data availability and more context. Currently to understand context we just look at the two immediate neighbours. Using trigrams woulve have increase the context information we have but the bottle neck is that the trigram frequency of many common phrases is not available.

5 Future Work

- Instead of plain word bigram frequencies, parts of speech tagged bigram frequencies along with POST probabilities can be used. This will reduce the size of the candidate set significantly.
- Using weighted edit distance
- Using metaphone algorithm to identify "phoenetic similarity" instead of soundex