Spell Checker-Report
CS6370

Aritra Ghosh(CS11B062),Sai Kiriti(CS11B007)

September 28, 2014

1 Summary

The Spell Checker is one of the novel problems which has been the subject of research from the advent of NLP. Here we design one spell checker both using previously proposed and extending it adding some of our own ideas. It is important in this note a misspellt word is assumed to be one which is not always the case.For eg There is possibility of misspelling advise and advice. However we do not consider such cases for the case of the sentence, although we could use the method to such cases. It will only increase the complexity of the method but not the essential principle.In the first part ie the word case we use the standard Bayesian approach based on "A Spelling Correction Program Based on a Noisy Channel Model ". In the case of the phrase Spell Checker we use a Bayesian method based on both context words and Part of Speech Tagging.We use some of the ideas presented in the paper "Combining Trigram based and Feature based Methods for Context- Sensitive Spelling correction" We also analyze the performance of the designed spell checker on several test cases. In the end we discuss some ways in which the spell checker can be extended to work better and possible modifications . The Spell Checker was implemented in Python using several libraries which are mentioned in line.

2 Source Code

We summarize some of the important files and how the code is organized

- *Metaphone* contains the Metaphone algorithm
- $ngrams\_info$ contains the part which deals with generating statistics from n-gram info
- sentence.py contains the functions for POS tagging.
- phrase.py contains functions for context -word method
- ranking.py contains the functions which are used for word spell check
- candidate_gen.py contains the various methods used for generating candidates
- main.py is the file used for the Spell Check as a whole.
- test_main.py is used for validating the Word Spell Check on the Wikipedia dataset for misspelt words

3 Corpora

- Unix Dictionary which consists 72751 words which is used to identify correct words. This also inludes some Proper nouns.
- We use some files from the Natural Language Corpus Data: Beautiful Data for the Word Spell Check
- We use the Ngrams-info for the Phrase and Sentence Spell Checker.
- We use the Brown Corpus for the POST tagging method although the results are not used

4 Word Spell Check

4.1 Preprocessing

We used the default Unix Dictionary as our dictionary. We removed all special characters from the Dictionary, converted all words to lower case and manually removed some commonly misspelt words. We also used to Pickle to store some of the object to make the techniques computationally more efficient.

4.2 Candidate Generation

We generate the plausible candidate replacements using 4 different methods

4.2.1 Edit distance

We use a BK-tree for storing the words in the dictionary. This is adapted to discrete metric spaces. The distance used can be any metric distance. In our case we use the Damerau–Levenshtein distance which considers transpositions in additions to substitutions, deletions, insertions used in the Levenshtein distance. In the seminal paper, Damerau not only distinguished these four edit operations but also stated that they correspond to more than 80% of all human misspellings.Also the searching in BK-tree is very efficient as compared to other data structures. Here we note that search means returning a list of approximate matches to the word.We only return words within a DL distance of 2.

4.2.2 Inverted Trigram index

We store for for every word in the dictionary we use a inverted trigram list i.e for every trigram we store the list of words it occurs in. Now given a misspelt word we find the trigrams in it and then look up the index to see the words which have more than a certain threshold of trigrams common with the misspelt word. We also use also Jaccard Similarity to shortlist the candidates. Empirical evaluation shows that trigram works best when compared to other n-gram values.

4.2.3 Phonetic Algorithm

We use a python implementation of the Metaphone algorithm(Double Metaphone). It is more efficient than the Soundex algorithm by using information about variations and inconsistencies in English spelling and pronunciation to produce a more accurate encoding, which does a better job of matching words and names which sound similar. Thus we store for every word in the dictionary the metaphone encoding(Sort of Inverted index). Given the misspelt word, we find the metaphone encoding and what others words in the dictionary share the same metaphone encoding. We note that these set of words have been added to take into account for homonyms like “throughout” and ”thruout” where the edit distance does not suffice.

4.2.4 Commonly misspelt words

We use Wikipedia list and Brown Corpus for a list of commonly mispelled words. In case we get such a misspelt word , we add the correct word to our possible candidates. This is more of an empirical approach but this is essential as this gives us a better idea from actual data and such candidates do not appear from the above methods Eg: f9 for fine. In a certain sense they capture the temporal sense since usage of words changes in time and the spellings used in Internet are very different from the usage.

4.3 Ranking candidates

This is a crucial part of the Spell Check process.This process constitutes ranking all the possible candidates given by the above methods.So given a wrong word t we want to find a c that maximises $P(c/t)$. We know from Bayesian theory that

$$P(c/t) = P(t/c)P(c)/P(t)$$

. Since $P(t)$ is same for every possible c , we essentially want to maximize $P(t/c)P(c)$ where $P(c)$ refers to the prior model of word probabilities and $Pr(t/c)$, is a model of the noisy channel that accounts for spelling transformations on letter sequences. The rest of this section details how these values are estimated. To estimate the priors we use the corpus given http://norvig.com/spell-correct.html. Inorder to account for words that do not occur in the corpus we perform smoothing(We use Add one smoothing here). This ensure we do not bias our results by the prior and thus non- occurring words can also occur in the answer set. The likelihood estimates are using the confusion matrix method as mentioned in the paper. We have the letter level statistics of the deletions,institutions, substitutions and transpositions.We use Maximum Likelihood estimates(MLE's) to get the estimates of the letter level estimates and then assume two different errors are indepenent and thus take product of the probabilities.Even in the case of the likelihood we need some smoothing for those mistakes that do not occur.An interesting point to note is the fact that in calculation of the likelihood the word t can be generated from c in several ways even with the same edit distance. We add up the probabilities of all such ways. In the end we use another method to modify the final results to account for phonetics. We check if the words in the result have metaphone encoding same as the given misspelt word and

also if the beginning letter is same as the that of the misspelt word as it is unlikely to misspell the first letter. Thus we change the score based on this and take a combination based on this. We see this affects the results quite amazingly.

5 Phrase Spell Check

In the case of Phrase spell check there can be several words in the phrase that can be misspelt. As is clear we cannot directly invoke the word spell checker for the misspelt words in the case of phrase as it depends on the context Eg: piece of cake vs peace of mind.We use ideas from [2]. So once the misspelt words are identified we use the spell checker to generate the possible candidates and take only the top-k ones. We see in general k=20 works well.Now we have the possible canddidates called Confusion set $C = \{w_1, w_2, \ldots w_n\}$. Depending on the context the inference as to which would be the best substitution.The idea is that each word w_i in the confusion set will have a characteristic distribution of words that occur in its context; thus to classify an ambiguous target word, we look at the set of words around it and see which w_i' distribution they most closely follow.The method is formulated using the Bayesian Framework.The task is to pick the word w_i that is most probable, given the context words c_j observed within a k-word window of the target word.

$$P(w_i|c_{-k} \ldots c_{-1} c_1 \ldots c_k) = \frac{P(c_{-k} \ldots c_{-1} c_1 \ldots c_k|w_i) P(w_i)}{P(c_{-k} \ldots c_{-1} c_1 \ldots c_k)}$$

Due to sparsity of data we use the naive Bayes assumption.

$$P(c_{-k} \ldots c_{-1} c_1 \ldots c_k|w_j) = \prod_{j \in -k,\ldots,-1,1,\ldots k} P(c_j|w_i)$$

We estimate these using the MLE estimate. We also remove of stop words from the context as they do not help in discrimation. In the case of phrases it is especially important to note that we do not have sufficient context. Therefore we take all the words in the context.The priors can be easily estimated as explained in the Word Spell Checker.

6 Sentence Spell Check

A part of the Sentence Spell Check is similar to the Phrase Spell Check. We use the context words approach with the modification here that we only take a k-word window and not the whole sentence into the context unlike the Phrase. We also use another method using ideas from the paper [3].In this part we use Part of Speech trigrams to encode the context.This method works well when the words to be discriminated have different Parts of Speech(POS).In case the words to be discriminated have the same POS we use the word in the confusion set is the most common representative of its part-of-speech class.

The probability of the sentence with that tagging; that is:

$$P(W) = \sum_{T} P(W, T)$$

where T is a tag sequence for sentence W. $P(W, T)$ is estimated by

$$P(W, T) = P(W|T)P(T) = \prod_{i} P(w_i|t_i) \prod_{i} P(t_i|t_{i-2}t_{i-1})$$

This model ignores the co-occurence between the words - the first product assumes that each word's occurence is independent of the words around it, and the second assumes POS trigram probabilities.The different values required are estimated using the corpus using the fairly straightforward MLE estimates including smoothing wherever needed.For the POS tagging we use the Stanford POS Tagger of the NLTK toolkit in python.We then choose the sentence with with highest $P(W)$. We use the Brown Corpus to estimate the values. One of the problems we face here is that the tagger only gives us a single Tag sequence for a given sentence. Thus this boils down to $max_{c_i} P(c_i|t_i)P(t_i|t_{i-2}t_{i-1})$.If the parts of speech are the same for the two different candidates this would futher reduce to $max_{c_i} P(c_i|t_i)$. Thus we use a weighted score comprising of this POS method which in a certain sense captures the order and the context window which captures the syntax.

When there are more than one word which is misspelt word we solve for each one ignoring all other misspellt words from the context. Although it is possible to exhaustively try all possible options all the words this would increase the complexity a lot.

7 Some Results

In this section we analyze some cases in which the spell checker works and and some in which it fails to work.

7.1 Word Spell Check

We use the Commonly misspelt words from the Wikipedia list and calcuate the accuracy after some preprocessing .We obtain an accuracy of 91.6% ie a total of 329 in 3601 words. For this we only take the first candidate obtained. We see that most of the errors are in the homophones.

7.2 Phrase Spell Check

As we have no dataset to test, we use the test set to validate our method We obtain an MRR of 1 for all phrases excepting for the case "presient of united states" where we we get a MRR of 1/2. We ignore the cases at the last which involve the splitting the sentence.

7.3 Sentence Spell Check

As with the case of the Phrase , we do not have a data set, We use the given Test set for check. We obtain a MRR of 1 for all cases except "ramed forces ","cort " example and "hopitals"(we get hospitals as the first suggest here) where we get a MRR of 1/2. We think this is primarily beccause of the sparsity of data.

8 Possible Extensions

In this assignment we have used the same ranking procedure for words produced by the different candidate generation methods.For example it would be better to rank words produced by the phonetic algorithms separately as many of them may have a greater edit distance but still may be the correct word.So we would like to work on ranking the candidate words based on the source of the error and combine the results intelligently. In our case we only consider the noisy channel model based on typographic errors for the purpose of ranking. In the spell checker we would like to explore how to use

collocations for the purpose of Spell Check.Although we use it in some sense as a part of the POST method we would like to study methods which use it directly.We would also like to use Word net to improve the Spell Checker. This would help to correct words which are not misspelt too.

References

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

[2] Andrew Golding *A Bayesian hybrid method for context-sensitive spelling correction*

[3] Andrew Golding Yves Schabes *Combining Trigram-based and Feature-based Methods for Context-Sensitive Spelling Correction*