NLP: Spell-Check Assignment Report

Team 5: CS13B053, CS13B060, CS13B061, CS13B062

21st September 2016

Introduction

Spell Checker is a software program or program feature designed to locate misspelled words and notify the user of the misspellings. Depending on the spell checker, the feature may either auto correct the word or allow the user to select from potential corrections on the misspelled word. For example, Microsoft Word shows a wavy red underlines for spelling mistakes and runs a spell check program on the entire document and provides suggestions for each misspelled word. Google spell checker is another powerful example. Hence, spell checkers are an extremely useful programs that help humans to resolve the typos, misspellings and mistakes in the words and documents. With the aim to build one such spell checker ourselves, we start this assignment.

The spell checker we built is capable of detecting errors in words, phrases and sentences and provides appropriate suggestions. The subsequent sections describe the details of our spell checker.

Word Spell Checker

The task of building a word spell checker was subdivided into two sub-tasks. The first sub-task was to generate the probable candidates that can be potential correct word for the given erroneous word. The second sub-task was used to rank the candidates generated by first task such that the ranking reflects the probability of replacement for the generated candidates.

Generating Candidates

We used two ways for generating the candidates, modeling the two ways in which the error can be generated. Given an erroneous word, the first method considers all the words in the standard dictionary that are at a fixed Levenshtein Distance (commonly known as edit distance) from the erroneous word. The second method takes care of the type of error that can be generated because of missing spaces.

Edit Distance Based Candidates

Here for a given word we find all the words in the standard dictionary that are a fixed edit distance away from the given query (erroneous) word. Our methods takes care of all the candidate words that are ≤ 2 edit distance away from the query word but may also include some candidate words that can be 3 edit distance away. Following are the the steps taken to achieve the task:

1. Generate modified dictionary derived from the original dictionary. For each word in the standard dictionary, terms with an edit distance ≤ 2 (with only deletes operation allowed) are included in the modified dictionary. Also for each word w_0 in the modified dictionary, we store a list which contains all the words in the standard dictionary that can generate w_0. (Note: This does not have to be done for each query but only once during a pre-calculation step.)

2. For a given query q, generate query set Q. A query set contains the query word along with all the terms which are an edit distance ≤ 2 (deletes only allowed) from the query word.

3. Search each term from the set Q in the modified dictionary. For all the terms w matched in the modified dictionary, add the corresponding words from the standard dictionary which generated w (the words from which w was originated) into the candidate set. This could be easily done by taking union of all the list (generated in the pre-computation step) corresponding to each word in the query set Q.

Split Error Based Candidates

Error such as 'halloffame' instead of 'hall of fame' needs to be handled separately as the above algorithm for ranking purposes cannot handle these errors. This is because we did not had the data from which we can estimate the insertion probabilities of spaces (used during the ranking stage). To handle these errors where spaces are deleted, we used the following algorithm to generate the candidates.

For a word $w[1 \dots n]$, we considered all split words $w[1 \dots i], w[i \dots j]$, $w[j \dots n]$, where $0 \leq i < j \leq n$ (considering upto two space insertions). Now for a given split we check if all words of the split $w[1 \dots i], w[i \dots j], w[j \dots n]$ are present in the dictionary. If it satisfies the condition we include the split in the candidate list.

Ranking Candidates

Both ranking lists along with the scores are generated for the two type of candidates using two different methods. Later they are merged together using a merger function which uses the scores and weights them (depending of some conditions) to generate a final ranking.

Ranking of Edit Distance Based Candidates

- We use the modified noisy channel model for ranking the probable candidates. For query q_0 for each candidate w_i was assigned a score as shown below:

$$Score(w_i|q_0) = \log(P^*(w_i)^\lambda * P^*(q_0|w_i))$$
$$P^*(w_i) = \frac{N_i}{N}$$

  Here $P^*(w_i)$ is the estimate of the prior probability of the word $P(w_i)$ in the English literature and $P^*(q_0|w_i)$ is the conditional probability of transforming q_0 from w_i given q_0.

- The prior is estimated by taking the ratio of the frequency N_i of the word w_i and total number of words N in the corpus. The conditional probability $P^*(q_0|w_i)$ of transforming the candidate word to the query word (given the query) is computed by adding all the probabilities in which candidate word can be transformed to the query word by only performing insertion, deletion, substitution and reversal operations. Also the number of such operations should be less than or equal to sum of query and word length.

- The algorithm for such probability computation was developed by modifying the Wagner–Fischer algorithm that is used to find the minimum edit distance between two words.

- λ is used to determine the weightage given to the priors. More the λ, more is the weightage given to the prior. Appropriate value of λ was selected based on the performance of the model.

Ranking of Split Error Based Candidates

- We used simple Bayesian approach for assigning score to such candidates. For each query q_0 and candidate $w[1 \dots i], w[i \dots j], w[j \dots n]$, the score is assigned as follows:

$$Score((w[1 \dots i], w[i \dots j], w[j \dots n])) = \log\Big(\prod P^*(w[i \dots j])\Big)$$
$$P^*(w[i \dots j])) = \frac{N_{i \dots j}}{N}$$

  We compute the probability of occurrence of the candidate, which is proportional to product of word probabilities of the split words. The $P^*(w[i \dots j])$ is the estimate of probability of word $w[i \dots j]$ in English literature.

Merging Rankings

Now a merger function was used to merge the two ranking lists. For fixing the ranking of split error based candidates, the modified scores of edit distance

based candidates and original score of split error based candidates are linearly transformed and then the two ranking were compared to give the final rankings for split error based candidates. However based on some conditions the linear transformation can be different for different query words. Now once the rankings of split error based candidates is determined, the rankings of edit distance based candidates are decided by sorting them based on their original scores.

Phrase & Sentence Spell Checker

For the phrase and sentence spell checker, we used a modified version of the context-sensitive spell check algorithm by AR Golding described in *'A Bayesian hybrid method for context-sensitive spelling correction'*. Also since most of the parts of this method are similar to the word based spell check, the description of method is kept concise.

The simple description of the algorithm is as follows:
Iterate over the words of the phrase/sentence. The current word represents the target word. For the target word, we generate the candidate set of words, the words which can potentially replace the target word in the phrase/sentence. We then assign score for each such candidate. At the end we give the suggestions based on these scores. The subsequent sub-sections describe the algorithm in more detail.

Generating Candidates

Collect the sets of commonly confused words (confusion sets), containing sets of words from dictionary which are often confused with each other. Now for a query word, the candidate set contains all other words in its confusion set (if the target word is present in any of the confusion sets) and all words in the dictionary which are ≤ 2 edit distance away from the target word (computed in same was as candidate generation in word spell checker).

Ranking Candidates

Now, for all the candidate words, we give them a score based on the context, the target word, and the candidate's prior probability. The main idea behind ranking the candidates for a given word is described below:

$$Score(w|c_{-k}\dots c_k, w) = \log\Big(\big(\prod_i \Pr(c_i|w)\big)^{\alpha_0} * \Pr(w'|w)^{\alpha_1} * \Pr(w)^{\alpha_2}\Big)$$

The following things should be noted:

- w is the replacement suggested.
- w' is the incorrect (corrupted) word.
- $c_{-k}\dots c_k$ is the context. (Here k is taken to be 3).

- $(\Pi \Pr(c_i|w))$ describes the posterior probability of observing the context, given the replacement.
- $\Pr(w'|w)$ describes the "corruption probability".
- $\Pr(w)$ is the prior of the replacement.

These probabilities are estimated from the corpus. Due to sparsity in observations, smoothing technique was used to in estimation of the probabilities. Also for the besr results, α_0, α_1 and α_2 are set so as to give the best performance. The candidates are then sorted based on their scores. The previous two steps are done for all the target words and corruption probabilities are recorded. Finally top three suggestions are given in order of these probabilities of all candidate words of all target words.

Smoothing Technique Used

We used the additive smoothing (also called Laplace smoothing) technique for unseen bigrams and all other probability estimations.

Observations

- We got decent performance for word based spell check when we performed our own test on a test set containing more than 4000 most commonly misspelled words.
- For word spell check, most of the time the intended word was among the top three candidates.
- For phrase/sentence spell check, the results were decently good.

Dictionary, N-grams & other Resources Used

The data for the assignment was used the following sources:

- Dictionary: '100,000 most popular words'
- Character bigrams and counts: Generated using Norvig's big.txt file.
- Word N-grams: 3-grams from Corpus of Contemporary American English
- Commonly confused words: alphadictionary.com
- Confusion Matrices (edit counts): Appendix of Kernighan et al. A Spelling Correction Program Based on a Noisy Channel Model.

References

- FAROO: 1000x Faster Spelling Correction Algorithm
- Golding, Andrew R. "A Bayesian hybrid method for context-sensitive spelling correction."
- Kernighan et al. "A spelling correction program based on a noisy channel model."