Spell Checker Assignment Report

CS11B004 Aishwarya P
CS11B012 Dhivya E

September 28, 2014

1 Detection

We use the list of 172,819 words available in Peter Norvig's site as our dictionary and detect a word as a misspelling based on its presence or absence in the dictionary. Some really uncommon words like *belive* and *hart* are present in our dictionary and so we are not able to detect them by using just this technique. A simple way to overcome this would be to flag a dictionary word as an error if it has a low prior. We have, however, restricted ourselves to just the first method in this assignment.

2 Candidate Generation

2.1 Assumptions

Our method of candidate generation is based on two assumptions:

- the correct word is around the same length as the misspelling
- the set of bigrams obtained from the correct word has a high Jaccard similarity with the set of bigrams obtained from the misspelling

These assumptions do not hold true in case of framing errors. But it works well in practice for typographical and OCR errors.

2.2 Offline Computation

To speed up the process of candidate generation, we generate the set of bigrams obtained from each of our dictionary word and store the list of words which contain a particular bigram in a hashmap (reverse indexing).

2.3 Online computation

When a misspelled word is given, we do the following in order:

1. obtain the set of bigrams in it

2. find the set of dictionary words which share at least one bigram with the misspelled word

3. prune this set by imposing the length restriction of plus or minus three from the length of the misspelled word

4. for this pruned set, compute the Jaccard similarity based on the set of bigrams

5. Rank these words by their Jaccard similarity and select the top 400 word as candidates

2.4 Parameter Tuning

There are two parameters to be tuned for this process to give best results.

- maximum permissible length difference of candidate from the misspelled word = 3
- Number of candidates to pick after ranking according to Jaccard similarity = 400

These values were picked by testing the performance of this procedure on a list of 4000 pairs of misspelled words and their corrections we downloaded off the web.

3 Context-Insensitive Ranking of Candidates

3.1 Not another string edit distance based ranking technique!

We have implemented an unconventional method (different from the usual string edit distance from the Noisy Channel paper) to rank the candidates picked above. This is a slight modification of the Jaccard similarity of bigrams where we give non-zero score to bigrams which match partially due to insertion.

3.2 Why the default Jaccard similarity of bigrams is insufficient

Before we answer this question, we formally define what perfect and partial matches are. Two bigrams a_1a_2 and b_1b_2 are said to match perfectly if $a_1 = b_1$ and $a_2 = b_2$ and partially if one of the following is true:

- $a_1 = b_1$ and $a_2 \neq b_2$ (insertion or deletion)
- $a_2 = b_2$ and $a_1 \neq b_1$ (insertion or deletion)
- $a_1 = b_2$ and $a_2 = b_1$ (transposition)

In case of perfect matches, the default Jaccard similarity gives the pair of bigrams a score of 1 in numerator and 1 in denominator. However in case of partial matches, the default Jaccard similarity gives no score in the numerator. This is not desirable as we need to give lesser penalty to insertion, deletion and transposition errors and compared to complete non-matches. Hence, we propose the following modifications.

3.3 Modifications proposed

Extra weight is to be added to the numerator (over and above the cardinality of the intersection of bigrams) according to the following rules.

- the first bigram of the candidate and the misspelled word (the one containing ^) match. Example: tank, tsnk
- the first two letters of the candidate are interchanged in the misspelling. Example: ^tank#, ^atnk#
- the bigrams are partial matches

4 Diversity in Candidates

4.1 Hypothesis

One of the important parameters in a recommender system is the diversity of recommendations. The user gets annoyed if he is presented with several (only slightly different) variants of an article. Extending this analogy to the context of spell checking, presenting as our top five suggestions several variations of the same root word/lemma is not a good idea. So, to improve recommendation of words, we prune this candidate set of 400 words by picking only the word which has the highest score for a given lemma.

4.2 Example

For example, consider the misspelling riase : raises, raised and raise are all candidates corrections. But according to our similarity measure, only raise will get picked.

5 Filtering using POS tagging

5.1 Corpora used

- Brown
- Gutenberg

5.2 Data extracted

We POS tag every sentence in the corpora, count the frequency of POS trigrams and maintain a list of POS trigrams whose count crosses a particular threshold.

5.3 How we use it

We do this filtering only for top 20 words after accounting for the diversity in the candidate set as POS tagging is a time-consuming step. Every candidate replacement produces a maximum of three POS trigrams. We count how many of these are not there in our list of frequent POS trigrams. If this number is greater than a one, we reject the candidate, else we let is pass through.

5.4 Where is it useful

Consider the misspelling "eath" in the sentence *from the earth to the moon*. The candidate corrections "each" and "earth" reach this step. However, each gets rejected because two of the POS trigrams - *the each to* and *each to the* are infrequent.
We take the top 5 words which pass this step and consider them for context-based checking.

6 Calculating Context Measure

After reranking the candidates of

6.1 Corpora used

- Brown Corpus - 50,000 sentences
- Gutenberg Corpus - 100,000 sentences

6.2 Data extracted

We incorporate context by using what we call as "unordered cooocurrence". Two words are said to have cooccurred if they appear together in some sentence in the corpus. We calculate the counts of such cooccurrences of each pair of lemmas in the corpus.

6.2.1 Importance of lemmatizing before counting

It is important to note that the words in a sentence of the corpora have to be lemmatized before we compute the cooccurrence frequencies. The reasons are listed below.

- This decreases the sparseness of the matrix of cooccurrence counts.

- Lesser values have to be stored as our space is now reduced to N(lemmas) × N(lemmas) from N(words)×N(words) as N(words) is several times smaller than N(lemmas)
- It is a more logically sound method as cooccurrences are the properties of word meanings rather than the forms in which they occur. For example, all the three phrases "walk down the aisle", "walking down the aisle", "walked down the aisle" indicate the same fact that walk and aisle cooccur. So even if "walks down the aisle" had not been there in the corpus, we should expect it. Lemmatizing captures this information.

6.2.2 Good Turing smoothing of counts

The revised counts for the combinations that are never occurring is estimated by Good Turing Smoothing. We use the revised counts upto some value of frequency (we have chosen 5) and after this value, we assume that the counts are fairly reliable (without doing any power law extra/interpolation). The exact update equations are:

$$c^* = c \qquad \text{for } c > k$$
$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1-\frac{(k+1)N_{k+1}}{N_1}} \qquad \text{for } 1 \leq c \leq k$$

6.2.3 Estimating Good Turing unigram counts from Good Turing cooccurrence counts

In order to calculate valid conditional probabilities from the above Good Turing counts, it is essential to recalculate the counts of unigrams by summing the counts of their cooccurrence counts with all other words. Getting raw unigram counts from the corpus and smoothing them using Good Turing algorithm might result in non-interpretable probabilities, some of them greater than 1. This will also give non-zero counts to every word in the dictionary which is an additional perk.

6.2.4 Conditional probability of cooccurrence

Now that we have non-zero counts for bigrams and cooccurrences, we can calculate the conditional probability of cooccurrence.

6.3 Getting context scores for a candidate

6.3.1 Initial check for stop-word candidates and absence of non-stop words

As we have cooccurrence data only on non-stop-words, we cannnot compare two candidates if they are not both non-stop-words. For this reason, we perform an

initial check to see if any of the top 3 candidates obtained by the previous ranking are stop words. If this is the case, we pick the one with the highest score and either report if (if the sentence has no more errors) or combine it with candidates of other misspellings in the sentence (in case of more than one error). We do a similar check to see if at all non-stop-words are present in the phrase/sentence. Otherwise, we resort to the alternative suggested above.

6.3.2 More than one misspelling

If there are several misspellings in a sentence, we take the top 3 independent suggestions for each misspelling and combine them in all possible ways. Then, for each combination of candidate words in a neighborhood, we calculate its context score as the sum of the context scores of the individual candidates in that neighborhood.

6.3.3 Calculating (a monotone of) the posterior of the neighborhood

The context score for a word in a neighborhood is the logarithm of the posterior probability of the (lemmatized) non-stop-dictionary-words in that neighborhood. Here, we make the simplifying assumption that the context words are all conditionally independent given the candidate replacement. Let t be a candidate and $l_1, l_2, \ldots l_n$ be the lemmatized non-stop-dictionary-words in the given phrase or sentence

$$p(t|l_1 l_2 \ldots l_k) = \frac{p(l_1 l_2 \ldots l_k|t) \times p(t)}{p(l_1 l_2 \ldots l_k)} \quad (1)$$

$$p(t|l_1 l_2 \ldots l_k) \propto p(l_1 l_2 \ldots l_k|t) \times p(t) \quad (2)$$

$$\propto p(l_1|t) \times p(l_2|t) \ldots p(l_n|t) \times p(t) \quad (3)$$

$$\propto log(p(l_1|t)) + log(p(l_2|t)) \ldots log(p(l_n|t)) + log(p(t)) \quad (4)$$

6.3.4 Prior value from context-insensitive ranking

The substitution of values for conditional probabilities is simple enough - just plug in the values calculated in the previous step. However, for prior, we are not using the smoothed unigram count of the word (as one would expect).
This is because, based on the context-insensitive ranking of words, we have learnt something more useful about the prior of the candidates rather than a simple unigram frequency in the corpus. In fact, without any information about the context, this is what we would believe about the likelihood of a candidate being the right substitution for the misspelling. So we incorporate this prior belief as the prior of the candidate.

We pick the top five combinations and report them in the decreasing order of their context scores.