Spell Check

CS14B037 Anvesh | CS14B042 Satish G

CS6370: Natural Language and Processing

---

1. Introduction

Spell check is a program that flags word in the input that may not be spelled correctly. Our program has three sub-programs in it:

1. Word spell check
2. Phrase spell check
3. Sentence spell check

1.1. Word spell check

Given a set of standalone erroneous words(not in dictionary), our objective is to suggest the corrections for each word.

1.2. Phrase/Sentence spell check

Given a set of Phrases(Sentences), our objective is to correct the erroneous word in the phrase/ sentence keeping the context into account. In general, a phrase has only 3-4 words and a sentence will have > 8 words. Constraint is that there'll be at most one error in each sentence.

2. Dataset

Corpus: *Google Web Trillion Word Corpus*
Dictionary: *Bragitoff dictionary* $\cup$ *Dictionary from Beautiful Data*
Bigram, Trigram word frequencies: *COCA*
Confusion matrices(for noisy channel model), and many other required frequency data files are taken from *Beautiful Data*.

3. Methodology

3.1. Word spell check

Intermediate:

- **Candidate words generation**: A union of words(in dict) at an edit distance 1 and 2 from the given error word, and words(in dict) having at least two bigrams in common with the error word.
- **Pruning**: Words with an edit distance greater than 3 are pruned .
- **Ranking**: Ranking of the words is given based on the below score formula, (plus one smoothing is used)

$$score = \frac{prior(word)+1}{editdistance(word,error\_word)}$$

Post-Intermediate:

- **Candidate words generation**: A union of words(in dict.) at an edit distance 1 and 2 from the given error word, and words(in dict.) having at least two bigrams in common with the error word.
- **Pruning**: The above candidate words are sorted based on the below score formula, and top 50 words are sent for ranking.

$$score = \frac{prior(word)+1}{3^{editdistance(word,error\_word)}}$$

- **Ranking**:
  - The words are given a score based on the Noisy Channel Model i.e,$(t = error\_word, c = word)$

$$score = Pr(t|c) * prior(c)$$

$$Pr(t|c) \approx \begin{cases} \frac{del[c_{p-1},\ c_p]}{chars[c_{p-1},\ c_p]}, & \text{if deletion} \\ \frac{add[c_{p-1},\ t_p]}{chars[c_{p-1}]}, & \text{if insertion} \\ \frac{sub[t_p,\ c_p]}{chars[c_p]}, & \text{if substitution} \end{cases}$$

- The phonetic score is computed using the double-metaphone library. The scoring is based on the rule, if primary matches primary then its a **strong** match, if primary matches secondary then its **weak**, when secondary matches secondary then its **minimal** otherwise no match. So we gave a score of 10^4, 10^2, 10 and 1 respectively, and multiplied it to the previous(above) score.
- The words are ranked based on the final score.

3.2. Phrase/Sentence spell check

- **Candidate words generation:** In case we find a word which is not in the dictionary, then suitable candidate words are generated for that word from the above word spell check program. If all the words are from dictionary, then candidate words are generated for each word in the phrase(sentence). The word is also added to its candidate list.
- **Ranking:**
  - New phrase(sentence) is generated by replacing each word of phrase (sentence) with each of its candidate word, without disturbing other words of phrase(sentence).
  - log-likelihood value of a generated phrase is calculated using **trigram** language model.

$$P(W) = \Pi_{i=3}^{len(W)} p(w_i|w_{i-2}, w_{i-1})$$

$$p(w_i|w_{i-2}w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, wi)}{count(w_{i-2}, w_{i-1})}$$

  - Phrases(sentences) are ranked based on their individual log-likelihood value.

4. Results

The following results are generated for the given intermediate evaluation data over the **intermediate** submitted code.

	Word	Phrase	Sentence
Time(sec)	49.64	44.78	90.47
MRR	0.363	0.337	0.225

4.1. Error Analysis

- In our intermediate code, we calculated the score assuming that it is inversely proportional to **(edit distance)**[1] and has less effect on ranking. This reflected in poor results of our program. Later we changed it to $3^{editdistance}$

- Also, we implemented Noisy Channel Model and considered the phonetics factor for ranking of a word.

- Examples from given evaluation data:

  – **coud**

Intermediate	you	your	our	out	could
Noisy Channel	could	cloud	cod	–	–
Noisy Channel + Phonetics	cod	could	cloud	–	–

Improved when Noisy channel is implemented. But adding phonetic algorithm had negative effect on it(as 'coud' and 'cod' have high phonetic score.).

  – **jeprodise**

Intermediate	promise	reproduce	improvise	jeopardise
Noisy Channel	promise	reproduce	jeopardise	–
Noisy Channel + Phonetics	jeopardise	promise	cloud	–

Less improved when Noisy channel is implemented. But adding phonetic algorithm had more positive effect on it(as 'jeprodise' and 'jeopardise' have high phonetic score.).

– **beleive**

Intermediate	live	receive	release	believe
Noisy Channel	belive	believe	receive	–
Noisy Channel + Phonetics	belive	believe	receive	–

Improved when Noisy channel is implemented. But adding phonetic algorithm had no effect on it(as 'beleive', 'belive', and 'believe' have same phonetic score.

The following results are generated for the given intermediate evaluation data over the **final** submitted code.

	Word	Phrase	Sentence
Time(sec)	52.88	44.54	93.18
MRR	0.68	0.511	0.313

5. Future work

- POS tagging can be added for training the model.
- Using CFG to parse phrases/sentences into POS tagging and improve performance.

6. References

- A Spelling Correction Program Based on a Noisy Channel Model (Research paper)
- http://spark-public.s3.amazonaws.com/nlp/slides/languagemodeling.pdf
- https://norvig.com/spell-correct.html
- http://norvig.com/mayzner.html
- https://www.ngrams.info/
- http://norvig.com/ngrams/
- https://pypi.python.org/pypi/Metaphone/0.4