2ID90 Spell Checker Assignment
template report

Group 19: Daan de Graaf, Yoeri Poels

April 4, 2017

1 Introduction

In this project we explore the creation of a spell checker, used for correcting english sentences with at most 2 errors per sentence (and no consecutive incorrect words). We do this using an approach of natural language processing we were taught in this course: reasoning with probabilities about the words and sentences that have to be corrected. In this report we will describe how we approached the problem, how we tried to solve it, and why we solved it the way we did.

2 Overall approach

To correct a sentence, our approach was to go through all the words in the sentence and find the best possible word in this place. We do this by generating a candidate set of words: we create this by taking all words with a Levenshtein Distance of 1 (giving insertions, deletions and substitutions a weight of 1). If the word is also in the vocabulary (and thus a valid word), we also add this to the candidate set. We then check for the probability of the typo of every word in the candidate set (and a constant for no typos), how common the word is, and how well it fits by looking at its 'neighbour' words. This gives us the following algorithm:

```
String Spellchecker(phrase) {
    String correctPhrase //the correct phrase
    for (every word in phrase) {
        candidateWords = words with Levenshtein Distance of 1 to word in vocabulary;
        if (word is in vocabulary) {
            add word to candidateWords;
        }
        for (every candidateWord in candidateWords) {
            candidateWord.probability = probability of typo * probability word occuring
                                        * probability of word occuring before/after its neighbours;
        }
        bestWord = word from candidateWords with the highest probability
        add bestWord to correctPhrase;
    }
    return correctPhrase;
}
```

Algorithm 1: Overall Approach

3 Phrase generation and evaluation

During phrase generation we find the best suggestion for a given phrase. A suggestion is generated word by word from left to right, using the previous word in the phrase if available to create bi-grams. For each word in the phrase we calculate the probability that it is the correct suggestion. This probability is calculated in three stages:

1. Candidate words generation (using the confusion matrix to find likely typos)
2. Calculating the probability of the occurrence of each of the candidate words
3. Calculating the probability of the occurrence of the bi-gram ending with the candidate word

We then combine these probabilities to obtain the most likely word. Stringing these together then yields the final suggestion for the correct phrase.

3.1 Candidate words generation

Candidate words are all words w_c^i that have a Damerau-Levenshtein distance of exactly 1 from the word in the phrase (w_p), plus that word itself (because it might be correct) with a predefined probability for a typo (0.9 in our implementation). For each of these words we calculate $P(t|c)$, where $t \subseteq w_p \wedge c \subseteq w_c$, such that t is a typo and should be replaced by c. For our purposes, we use:

$$P(t|c) = \frac{confusionCount(t,c)}{biCharCount(t)}$$

Here $confusionCount(t,c)$ is the number of times that t is a typo and c was intended, according to the confusion matrix. $biCharCount(t)$ is the number of times that the biChar t occurs in the corpus. The intuition behind this formula is that it calculates how often t should be changed out of all the times it occurs.

3.2 Unigram occurrence probability

We now calculate the probability of occurrence of each candidate word using the number of times it occurs in the corpus:

$$P(w_c) = \frac{NGramCount(w_c)}{corpusSize}$$

We divide the number of times the candidate word occurs in corpus divided by the total number of words in the corpus. This measures how common a word is.

3.3 Bigram probability

At this stage we calculate how likely the combination of the candidate word and the previous word is. To calculate the probability of the bigram we use Kneser-Ney smoothing:

$$P_{KN}(w_i|w_{i-1}) = \frac{max(c(w_{i-1},w_i) - \delta, 0)}{\sum_w' c(w_{i-1},w')} + \lambda_{w_{i-1}} \cdot P_{KN}(w_i)$$

Where for w_{i-1}:

$$\lambda_{w_{i-1}} = \frac{\delta}{\sum_w' c(w_{i-1},w')} |\{w' : 0 < c(w_{i-1},w'\}|$$

And for a unigram w_i:

$$P_{KN}(w_i) = \frac{|\{w' : 0 < c(w',w_i)\}|}{|\{(w',w'') : 0 < c(w',w'')\}|}$$

In order to express this in java, we derive:

$$c(x,y) = getNGramCount(x,y)$$
$$\sum_w^{\prime} c(w_{i-1},w') = biGram1Total.get(w_{i-1})$$
$$|\{w' : 0 < c(w_{i-1},w'\}| = biGram1.get(w_{i-1})$$
$$|\{w' : 0 < c(w',w_i)\}| = biGram2.get(w_i)$$
$$|\{(w',w'') : 0 < c(w',w'')\}| = biGramCount$$

Where $biGram1Total$ is almost identical to $biGram1$, only the first increments using the count attached to each biGram. Or in java code:

$$biGram1.put(w1, biGram1.getOrDefault(w1,0) + 1)$$
$$biGram1Total.put(w1, biGram1.getOrDefault(w1,0) + count);$$

Applying the appropriate substitutions yields a correct java implementation.

3.4 Combining

The probability of a typo $P(t|c)$ defines the noisy channel probability. Multiplying $P(w_c^i)$ and $P_{KN}(w_i|w_{i-1})$ yields the language model probability. To obtain the final probability we combine these two:

$$P = P(t|c) * P(w_c^i)^\lambda * P_{KN}(w_i|w_{i-1})$$

Where λ is a calibration parameter (set to 0.3 in our implementation).

Give a full discussion of how you have implemented phrase generation and have the best candidate sentence is selected. In particular, describe what rule the confusion matrices and bi-grams play a role in attaching a value/probability to a candidate sentence. Also describe what type of smoothing you use and how you have implemented it.

4 Results and evaluation

Allebei

Provide an overall assessment of your programs. What type of errors is it good at to catch and repair, which type or errors are missed or wrongly repaired. Explain what could be done, in principle, to improve your program. Discuss how you have calibrated the relevant parameters.

5 Advanced enhancements

Yoeri

Several extensions and enhancements of the basic set-up of the spell checker are possible. Describe explicitly what you have incorporated from other sources than the course material to improve the performance of your program and in what way the performance did improve indeed.

6 Conclusions and contributions

A short logical summing up of the main reported results and a statement on the contributions of each of the authors.

	implementation	**documentation**	**total #hours**
Daan	60%	40%	22
Yoeri	40%	60%	21

- *At least the given columns in the table need to be filled in, add columns if needed.*
- *Add comments to clarify your table entries when necessary.*

References

[1] Wolfgang Ertel. *Introduction to Artificial Intelligence.* Springer Publishing Company, Incorporated, 1st edition, 2011.