Spellchecker

Martijn Gösgens 0914954
Dominique Sommers 0895679

March 29, 2016

1 Algorithms

1.1 Candidate words

Algorithm 1 getCandidateWords(String word, double numWords)

mapOfWords to be filled with all candidate words
for all Letters in the alphabet **do**
  for all Positions in word **do**
    Insert letter in position
    Put resulting string in *mapOfWords*
for all Letters in word **do**
  Delete letter from word
  Put resulting string in *mapOfWords*
for all Letters "a" in word **do**
  for all Letters "b" in word **do**
    Swap letters "a" and "b" in word
    Put resulting string in *mapOfWords*
for all Letters "a" in the alphabet **do**
  for all Letters "b" in word **do**
    Replace letter "b" with letter "a" in word
    Put resulting string in *mapOfWords*
resultMapOfWords to be filled with possible good candidate words
for all Words in *mapOfWords* **do**
  $NoisyChannelProbability \leftarrow confusionCount/Normalization$
  if $NoisyChannelProbability > 0$ **then**
    Put the candidate word in *resultMapOfWords*
    $Probability \leftarrow log_{10}(NoisyChannelProbability) + log_{10}(WordProbability)$
if word itself is contained in the vocabulary **then**
  Put word in *resultMapOfWords*.
  if $\#words > 1$ **then**
    $Probability \leftarrow log_{10}((\#words - 1)/\#words)$
  else
    $Probability \leftarrow log_{10}(0.5)$
Return *resultMapOfWords*

This function gets all the candidate words of the input word. These candidate words have a maximum Damerau-Levenshtein distance of 1 which means that the word is only altered by at most 1 insertion, deletion, transposition or substitution. We then determine the probability for each word w in the map, that the writer wanted to type w but mistakingly typed the word we have received. We estimate this using Bayes rule: we want to know $P(w|O)$ (O is the observation), this equals

$$\frac{P(O|w) \times P(w)}{P(O)}$$

We neglect the $P(O)$ term since it is a common demoninator of all candidates. We now estimate $P(w|O)$ using the confusion matrix, normalised by the number of typos that could possible be made in that part of the word and we estimate $P(w)$ using smoothening. We then save this probability in $\log_{10}$ since the numbers tend to get too small.
If the word itself is in the vocabulary, this word is also added to the result. To determine its probability we recall that a sentence can either have zero, one or two errors and we estimate the chance that this word is corrected by (numWords $-\frac{0+1+2}{3}$/numWords if numWords is greater than one. Otherwise, we only have a single word without a surrounding sentence and we estimate the chance that this word is correct by $\frac{1}{2}$.

1.2 Find correct word

Algorithm 2 findCorrect(String previousSentence, String nextSentence, double currentProbability, int correctionsMade)

```
if nextSentence is empty then
    Add the probability that the last word was the end of the sentence
    return the resulting sentence with its probability
else if correctionsMade == 2 then
    Add the probability that the rest of the sentence is correct
    return the resulting sentence with its probability
else
    initialize bestSentence
    for word : getCandidateWords(nextWord) do
        Add the probability to currentProbability
        Add the word to the sentence
        Remove the word from nextSentence
        if word is a correction then
            correctionsMade++
        candidate = findCorrect(previousSentence, nextSentence, currentProbability, correctionsMade)
        if candidate has a higher probability then bestSentence then
            bestSentence = candidate
    return bestSentence
```

This algorithm fits the candidate words of each (possibly) misspelled word in the sentence such that it is semantically most likely (according to the bigrams). It recursively adds words and returns the sentence with the highest probability. In each iteration, we seperate three cases:

1. The algorithm has reached the end of the sentence: it now calculates the chance that the last word was the end of the sentence by seeing how often the last word was followed by "EoS" (this chance is altered by smoothening, as you will see in the next section). We then return this sentence with its probability.

2. The algorithm has used both its corrections: If the previous corrections are right, then the next words must all be correct. We calculate the probability that this is the case using bigrams. Then we return this sentence with its probability.

3. None of the above: We now add each candidate of the next word to the sentence, process probabilities and call findCorrect() again. We then return the sentence with the candidate which ended up in the most probable sentence (according to our calculations).

The probability is determined in the following way: we want to know $P(w_1, ..., w_n)$, we estimate this by $P(w_n|w_{n-1}) \times ... \times P(w_1|w_0)$. Calling the above algorithm should thus result in the most probable sentence according to our calculations and data.

1.3 Good-Turing Smoothing

Algorithm 3 getSmoothedCount(String $NGram$)

Construct a sorted map $freqOfFreq$ with the frequency of all occurring words as keys and the frequency of these frequencies as values
$maxFreq \leftarrow$ the most frequent word
$sumCounts \leftarrow$ the sum of all frequencies
$c \leftarrow$ the frequency of the word $NGram$
if $c == 0$ **then**
    Return $freqOfFreq.get(1)/sumCounts$
else if $c == maxFreq$ **then**
    Return $c/sumCounts$
else
    $N_c \leftarrow freqOfFreq.get(c)$
    $nextC \leftarrow$ the lowest frequency higher than c
    $N_{c+x} \leftarrow freqOfFreq.get(nextC)$
    Return $(c + x) * N_{c+x}/N_c$

For the smoothing of the counts of the words, we chose to use Good-Turing Smoothing with a slight adjustment. The smoothing of the counts is to determine the chance of a word occurring again. The smoothed count will be slightly lower than the concrete count since there are words that have never occurred before. This difference is determined by the frequency of the frequency of the words. With the Good-Turing algorithm the smoothed count will be determined be the frequency of the frequency of the word and the freq. of the frequency + 1. But since there are gaps between frequencies, especially when the frequency gets higher, it serves no value to take the freq. of the frequency + 1 since this is zero. In our implementation we therefore take the next occurring frequency to get a value. When the gap between frequencies is high, this gets more unreliable. There is a solution to avoid this by drawing a line of the frequencies which is the best-fit power law, but we did not have the time and knowledge to implement this yet.

2 Results

The results of our spell checker are quite good. Most of the sentences we have tested are corrected in the right way, including those on Peach. When the sentence contains more than two errors, the corrector will never correct the sentence in the right way since it will never alter more than two words. Moreover, the corrector cannot handle apostrofs.

3 References

1. Ertel, W. Introduction to Artificial Intelligence.
2. Jurafsky, D.; Martin, J.H. Speech and Language Processing.

4 Contribution

Martijn Gösgens: I implemented getCandidateWords that was needed to find the possible corrections for each word and I implemented the findCorrect algorithm that was needed to semantically fit these words in a sentence. I decided to make the findCorrect algorithm such that it would never correct more than two errors since else it would very likely to correct words which do not need correction. I also helped debugging the smoothening.
Dominique Sommers: I implemented the Good-Turing Smoothing algorithm and decided to take the next frequency instead of add-one smoothening or the more complicated Good-Turing Smoothing with the best-fit power law. Besides that, I wrote most part of the report. I also made a function to get the best candidate word for every word in the sentence, but this was later adjusted and corrected by Martijn adding the bigrams of the sentences.

5 Manual

The spellchecker takes sentences as input. These can consist of multiple words and multiple sentences. An important thing to note is that the spellchecker can only correct words with a Damerau-Levenshtein distance of 1 which means that each input word may only be altered by at most 1 insertion, deletion, transposition or substitution. The words in the input sentences may not have interpunction or any other kind of character than the english alphabet, the apostrophe (') is also forbidden. All letters of all words should be in lower case. After giving the input, the spellchecker will check for spelling mistakes and automatically replace them to the correct spelling of the word.