# Spelling Correction Combining the Language Model and Noisy Channel Model

王曌 (Student ID:18210980068)

*Abstract*—Fixing spelling errors is an integral part of writing in the modern world, whether this writing is part of texting on a phone, sending email, writing longer documents, or finding information on the web. This assignment is to write a toy system for spelling correction. I do this program to correct non-word spelling error in sentences using N-gram Language Models, Noisy Channel Model, Error Confusion Matrix and Damerau-Levenshtein Edit Distance.

*Index Terms*—Spell Correction, Language Model, Noisy Channels

## I. THEORY FOUNDATIONS

The call correction tries to choose the most likely spelling correction for w. There is no way to know for sure, which suggests we use probabilities. We are trying to find the correction c, out of all possible candidate corrections, that maximizes the probability that c is the intended correction, given the original wrong word w:

$$argmax_{c \in candidates} P(c|w)$$

By Bayes' Theorem this is equivalent to:

$$argmax_{c \in candidates} \frac{P(c)P(w|c)}{p(w)}$$

Since P(w) is the same for every possible candidate c, we can factor it out, giving:

$$argmax_{c \in candidates} P(c)P(w|c)$$

Show the above algorithm by the Figure.

```
Function Noisy Channel Spelling(word x, dict D, lm, editprob) returns correction
    if x∉D
        candidates, edits ← All strings at edit distance 1 from x that are ∈ D, and their edit
        for each c, e in candidates, edits
            channel ← editprob(e)
            prior ← lm(x)
            score[c] = log channel + log prior
        return $argmax_c score[c]$
```

Fig1:Noisy Channel Model for spelling correction for non-word errors

A. N-gram Language model

Models that assign probabilities to sequences of words are called language models.

(1) N-gram: N-gram refers to contiguous sequence with the length of N from a text or other corpus. We can establish probabilistic language model on N-gram sets based on the form of a (n - 1)-order Markov model, which make it possible to evaluate the chance of words in any situation, even predict the forward words or backward words. However, when there is a typo in the text, the N-gram model will notice the error, because the chance of error spell word in such situation appearance is relatively low to ordinary words.

(2) Smoothing: In real corpus, the training data is usually insufficient, which results the estimated distribution of many events x comes zero: p(x) = 0. However, the semantic space is much wider than our training data space, which means so-called "unseen words" in our model might indeed exist in the real problem. As a result, we should smooth our model to make it more robust in any situation. The intuitive method is to add frequency for each word in dictionary, which proposes add-k Laplace Smoothing:

$$P^{*}_{Add-k}(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n) + k}{Count(w_{n-1}) + kV}$$

B. Noisy Channel model

(1) Edit Type: There are four edit type: deletion, insertion, substitution, reversal

(2) Confusion Matrix: In general, a confusion matrix lists the number of times one thing was confused with another. Thus for example a substitution matrix will be a square matrix of size 26×26 (or more generally |A|×|A|, for an alphabet A) that represents the number of times one letter was incorrectly used instead of another. Following Kernighan et al. (1990) I'll use four confusion matrices.

del[x,y]: count(xy typed as x)
ins[x,y]: count(x typed as xy)
sub[x,y]: count(x typed as y)
rev[x,y]: count(xy typed as yx)

Once we have the confusion matrices, we can estimate P(w|c) as follows(where $c_i$ is the ith character of the correct word c) and $w_i$ is the ith character of the typo w:

$$\mathrm{P(w|c)} = \begin{cases} \dfrac{del[w_{i-1}, c_i]}{count[w_{i-1}c_i]}, if\ deletion \\ \dfrac{ins[w_{i-1}, c_i]}{count[c_{i-1}]}, if\ insertion \\ \dfrac{sub[w_i, c_i]}{count[c_i]}, if\ substitution \\ \dfrac{rev[c_i, c_{i+1}]}{count[c_i c_{i+1}]}, if\ \text{reversal} \end{cases}$$

In order to get a better performance, we often use smooth method to regulate the 4 matrixes.

## II. Experiment

The environment is Ubuntu 16.04.4 LTS.

There are 4 parts in my program.

（1）Preprocessing

・ Take reuters corpus as an example, change this corpus into list and generate N-gram dictionary by counting the number of unigram, bigram and trigram.

・ Clean testdata by subtracting the punctuation. Change testdata and vocab into list.

(2) Language model

・ The probability that c appears as a word of English text.

```
unigram
1.(c(mistake)+1)/(N+V) is -11.929815
2.(c(tokyo)+1)/(N+V) is -13.634563
bigram
1.(near future/near) is -6.575316
2.(increase investment)/ increase) is -9.537147
trigram
1.(undeveloped gold project/undeveloped gold ) is -9.942612
2.(the general downturn/ the general) is -9.942612
```

(3) Channel model

・ Candidate list: the correction c out of all possible candidate corrections

```
candidate of acress is
edit distance = 1
[caress, access, actress,across,acres,acres]
edit distance = 2
[caress,access,actress,across,acres,acres,press,stress,screws,Actress,Across,Acres,Acre
s,caress,caress,ogress,duress,dress,Press,Dress,press,dress,Press,Dress,caress,chess,cr
ass,cross,crews,crest,crests,cares,caress,cares,assess,afresh,arrest,arrests,airless,addres
s,address,agrees,areas,access,access,access,actress,actress,aches,aches,access,aces,ac
es,across,across,acres,across,acres,acre,acres,acres]
```

・ Edit type: Get single edit errors' type

```
edit type of 'could' to 'coul' is
('Deletion', 'd', '', 'c', 'cd')
```

・ Channel model: The probability that w would be typed in a text when the correction is c.

```
channel model of 'could' to 'coul' is
-1.6094379124341003
```

(4) Spell Correction: Combining the language model and channel model, choose the candidate with the highest combined probability and correct the non-word error. Show three lines of my final result as following.

```
1 protectionist
2 Tokyo
3 retaliation
```

## III. Result

Comparing the final results among different hyperparameters.

(1) **Different corpus:** Fixed N-gram as bigram, k = 0.01

| Corpus | Reuters | Brown |
|---|---|---|
| Accuracy(%) | **85.9** | 80.1 |

(2) **Different N-gram:** Fixed corpus as reuters, k = 0.01

| N-gram | unigram | bigram | trigram | quadrigram |
|---|---|---|---|---|
| Accuracy(%) | 85.6 | **85.9** | 85.7 | 85.4 |

(3) **Add-k Smoothing:** Fixed corpus as reuters, bigram

| k | 0.01 | 0.1 | 1 |
|---|---|---|---|
| Accuracy(%) | **85.9** | 85.3 | 84.2 |

## IV. Conclusion

According to the result, it can been seen that I get the highest accuracy 85.9% when the parameters are bigram, reuters, add-0.01 smoothing. It is beyond cognition that bigram has a higher accuracy than trigram, which may inspire us that N-gram is not the bigger, the better. With different smoothing method and n-gram, I can choose the appropriate one considering the balance of time costs and accuracy.

## V. Reference

[1] How to Write a Spelling Corrector
[2] Spell-errors.txt
[3]JamesH.Martin. An introduction to natural language processing, computational linguistics, and s[M]. 2005.
[4]Eric Brill and Robert C. Moore. An improved error model for noisy channel spelling correction. In Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, ACL ’00, pages 286–293, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.