Spelling Errors Correction Based on Bayesian Model with Language Model and Noisy Channel

Linyang He(15307130240)

October 18, 2017

Abstract

Spelling error correction is a easy and basic application of language model, which can be used in many ways such as search engine and IME. In this project, our method is base on bayesian model. With respect to the bayesian model, the prior probability get be measured by noisy matrix, while the likelihood function should be related to the language model. The noisy matrix can be estimated by a large dataset of spelling error. The language model can be trainned from NLTK corpus. Further, there're a few methods of smoothing which can enhance the performance of the language model including the Kneser-Ney algorithm. In this report, we will see the accuracy as well as the time variance on different smoothing methods and different corpuses.

1 Introduction

The detection and correction of spelling errors is a important part of word-processors. Those algorithms used by this subject is also used in other fileds like **optical character recognition** and **handwriting recognition**. So it's a quite essential subject for the Language Processing beginners to learn about language model and noisy channel model. Spelling error is caused by wrong typing usually. And almost all the edit difference between the typos and original word is lower than or equal to 2. So we can use the noisy channel to generate a list of candidates according to this. Finally, we can use language model to determine the optimal candidate.

The rest of the report is structured as following: Section 2 introduces the background we may need. Section 3 describes the data and the experimental setup. Section 4 presents the experiment result of the accuracy and time varying from different corpuses and smoothing methods. Section 5 summarizes related work while Section 6 concludes our report.

2 Background

2.1 Noisy Channel Model

When people type words, there might be some disturbance leading them to type wrong, which we call it as 'noise'. So the noisy channel is like a blackbox, the input is the correct word, while the output is the noisy word. If we want to get the original word, we can use numerous spelling errors dataset to estimate the 'blackbox'. Once we get the noisy matrix representing the channel, we can use it to decode our noise word conversly. Given the noise word, we can guess the original word according to Bayes formula:

$$P(w_o|w_n) = \frac{P(w_n|w_o) \times P(w_o)}{P(w_n)}$$

Here w_o denotes the original word, while w_n denotes the noise word. To obtain the original word, we maximize the posterior probability.

$$\bar{w}_o = \text{argmax}\ \frac{P(w_n|w_o) \times P(w_o)}{P(w_n)}.$$

Considering the noise word is given,

$$\bar{w}_o = \text{argmax} P(w_n|w_o) \times P(w_o)$$

Here $P(w_o)$ is the language model of the original word in a certain corpus. And the likelihood function $P(w_n|w_o)$ means how probable is the noise word given the original word. This probability can be obtained from the "Noisy Matrix". These matrices denotes the frequence distribution of each letter's possible movements. We know that there are 4 basic ways to get a noisy word: (1) Delete; (2) Insert; (3) Substitute; and (4) Revesal. Therefore we can get 4 matrices of each kind of movement. These matrices can be counted from spelling errors dataset. Once we have the movements matrices, we can get our likelihood function as

$$P_{w_n|w_o} = \begin{cases} \dfrac{del[w_{i-1}, w_i]}{count[w_{i-1}, w_i]} & \text{delete,} \\[2ex] \dfrac{ins[w_{i-1}, w_i]}{count[w_{i-1}]} & \text{insert,} \\[2ex] \dfrac{sub[w_i, w_i]}{count[w_i]} & \text{substitute,} \\[2ex] \dfrac{rev[w_i, w_{i+1}]}{count[w_i, w_{i+1}]} & \text{reversal.} \end{cases}$$

2.2 Language Model

Language model is to compute the probability of a certain sentence or a series of words

$$P(w_1 w_2 ... w_n) = \prod_i P(w_i|w_1 w_2 ... w_{i-1})$$

To simplify the model, we can use n-gram, which according to Markov assumpotion, the probability of the current word is related only to the $n-1$ words before rather than all of them. That is

$$P(w_1 w_2 ... w_n) \approx \prod_i P(w_i|w_{i-k} ... w_{i-1})$$

Usually, we use unigram, bigram, trigram and 4-gram more often. And our report will use bigram as example for demonstrating in this report. So, for bigram,

$$P(w_i|w_1 w_2 ... w_{i-1}) \approx P(w_i|w_{i-1})$$

To estimate the bigram probability, we have

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}.$$

To get the counts, we can compute the frequency of each word and each bigram in our corpus.

2.2.1 Naive Model

What if the count of a word is zero in the corpus? This will lead to a divided by zero error. To avoid this kind of error, we should prevent the denominator becoming zero. Usually, there are some smoothing methods may help which we will talk about later. The thing is, a lot of these methods may lead to discounting the probability of possible n-grams greatly with impoving the probability of some imposibble words a lot, while some better algorithms leads to computation complicacy heavily.
Therefore, sometimes, if our machine is not quite good, the naive model can be considered. By the word 'naive', it means we don't smooth the model quite much to avoid the zero dividing error. Instead, if the counts of certain words are zero, we just turn them into a infinitesimal ϵ. It will not cause the zero dividing error still will not promote the probability of some impossible words a lot inexplicably. That is

$$P(w_i|w_{i-1}) = \begin{cases} \epsilon & c(w_{i-1}, w_i) = 0 \text{ or } c(w_{i-1}) = 0, \\ \dfrac{c(w_{i-1}, w_i)}{c(w_{i-1})} & \text{otherwise.} \end{cases}$$

Besides, since the language model probability is a extremely small number which may lead to underflow in our machine. To prevent such problems occurs, all the multiplication should be turn into addition of related logarithms. So if any 0 occurs, we turn it into ϵ.

2.2.2 Smoothing Methods

The naive model is a quite rough model actually. If a possible word doesn't occur in our corpus, the naive model works quite poorly. So more ideal models should consider smoothing methods. Here are some main methods. Also, they are the models we will experimente on at the same time.

1. **Laplace and add-k smoothing**. Laplace, also called as add-1 methods, is a special case of the add-k model obviously. In this case, the probability of the bigram

   $$P(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + k \times V}.$$

   When k is equal to 1, it's Laplace model. Here V denotes the type length of our corpuse, or in other words, the length of the vocabulary.

2. **Unigram prior smoothing.** This model introduces unigram's information to the bigram, which is like

   $$P(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + kP(w_i)}{c(w_{i-1}) + k \times V}.$$

   This methods can be regarded as a interpolation algorithm.

3. **Absolute discounting smoothing.** Considering the add-k smoothing methods, what we do is actually sparing some possibility of those $counts > 0$ bigrams to some bigrams not showing in our corpus. So, in turn, we can just discount some counts of the bigrams existing. Plus, to get more information, we should add the interpolation item. So the formula should be

   $$P_{abso}(w_i|w_{i-1}) = \frac{c(w_i, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$

   As for the discounting number, as a matter of experience, is approximately equal to 0.75 if the count is larger than 2, 0.5 if the count is 1 and 0 for the count 0. With respect to *lambda*, we will cover it in the last methods – the Kneser-Ney smoothing model.

4. **Kneser-Ney smoothing.** Still now, this kind of model is the most ideal algorithm. The novel point of this methods is introducing a probability of continuation. It measures how a word can continue in the sentence. We have

   $$P_c = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_k) : c(w_{j-1}, w_j) > 0\}|}$$

   K-N smoothing can be regarded as a more sophisticated absolute discounting methods:

   $$P(w_i|w_{i-1}) = \frac{max(c(w_i, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_c(w_i)$$

   With respect to the weight λ, we have

   $$\lambda(w_{i-1}) = \frac{d}{c_{w-1}}|\{w : c(w_{i-1}, w) > 0\}|.$$

3 Project

3.1 Data

3.1.1 Spelling Error Data

We use the Peter Norvig's list of errors as our speling error data. As a matter of convenience for counting, we handle the list to another list, which each line contains only one noise word and one original word.

3.1.2 Corpus

NLTK provides several corpuses based on news article. We will use *reuters* and category news in *brown* corpus. Specificly, we use the first 100000 words of reuters and whole words of brown.

3.1.3 Vocabulary

This vocabulary may help us to dectect if a word is a real word. Still there is few noise in the vocabulary, which is more close to the reality.

3.2 Setup

3.2.1 Typo Dectection

1. **Non-word Dectection**. This type of wrong word is quite easy to find. If a word is not in the vocabulary, then it's a typo.

2. **Real word Dectection**. This kind is more complicated by comparison. If a sentence contains no non-word error, we have to dectect each word in the sentence by n-grams. If one n-grams' language model probability is lower than any other, then we believe that one word of this n-gram contains a real word spell error. So we may assume that every word in this n-gram is a typo.

3.2.2 Candidates Generating

After we have found or assumed the typo(s), we can generate the right word candidates. By deleting, inserting, substituting and swaping some letters in the typo(s), we have a series of otherwords. If the edit difference between these words and the wrong word is lower than or equal to 2 as well as the words are in the vocabulary, we believe they are candidates of the typo. This is how we generate the candidates both for the non-word error and real word error.

3.2.3 Finding the Optimal Candidate

With the candidates, we should find the optimal one. We use the noisy channel and language model to get the optimal candidate. The noisy channel part is $P_{w_n|w_o}$ as we have discussed in the background section. What should be paid attention to is, we don't need to get the language probability of the whole sentence, instead, all we need to care about is those n-grams containing the typos. For example, if we use bigram, and the typo is w_i, the bigrams needing to computate are just (w_{i-1}, w_i) and (w_i, w_{i+1}). So what we do is to maximize the

$$P = P(w_{noise}|w_i) \times P(w_i|w_{i-1}) \times P(w_{i+1}|w_i)$$

for each word in the candidates list. If the probability is the maxima, then the candidate is optimal.
When we use the language model to compute the probability, we may use different smoothing methods to get a optimal model for our problem.

4 Results

Here is the experiment results differing from different corpus and different smoothing methods. Here are the results when we use reuters corpus.

Smoothing	**Naive**	**Laplace**	**Add-0.1**	**Add-0.5**
Accuracy	93.3%	90.6%	90.3%	88.7%
Time	66.1s	65.9s	67.3s	65.7s

Smoothing	**UniPri**	**AbsDis**	**K-N**
Accuracy	91.5%	93.3%	90.1%
Time	64.7s	136.5s	192s

And when brown news is used:

Smoothing	**Naive**	**Laplace**	**AbsDis**	**K-N**
Accuracy	82.2%	81.6%	81.5%	70%
Time	64.5s	64.8s	65.9s	203.7s

5 Summary

According to the table above, we can find that strangely, the naive model works the best. We guess this is because based on the test data and the corpus, other sophiscated models enhance the zero probability ngrams too greatly, which may harm the language model on the contrary rather than improve the performance. Besides, we find that the model based on the reuters corpus is far better that that on the brown news reuters. We guess there're certain reasons perhaps leading to such phenomenon.

1. The type length of brown news(14394) is far lower than reuters corpus(1300000), the language model trained by the former works poorlier than the latter.

2. The test data may be generated from the reuters corpus, so the language model based on reuters works better.

6 Conclusion

In this report, we demonstrate how we use noiss channel and language model to generate the candidates and use language model to find the optimal candidate. Besides, we use different smoothing methods and diffenrent corpuses to estimate the performance by the accuracy and time.