NLP Homework1: Spelling Correction

Deng Wen 17307130171

Fudan University

October 17, 2019

I. Introduction

In this homework, I am required to write a toy system for spelling correction. By using the knowledge learned in class. I trained the unigram and bigram model from the reuters corpus in NLTK. And used three different smoothing methods to improve the model. Then I build the channel model by using the 1-edit error data collected online. Finally with some processing on the testdata, I combined the two models and got a result with accuracy over 90%.

II. Preparatory Work

- Unigram and Bigram

Load the reuters corpus from NLTK, and count the unigram and bigram, then save them to text files for later uses.[1]

- Channel model

1. From the Norvig , I import the 1-edit error count to bulid confusion matrix.

2. And for every wrong word, I generate candidate words within two edits, and use the form like "error1+error2" to compute the channel probability.

- Smoothing

I used the three smoothing methods when compute the Language Model probability. It was written as the function form in the program.

1. Add-1 smoothing. For every possible word or bigram, we add 1 on their frequencies.

2. Good turing. Use the bigrams that only appear once to estimate the word never appeared before.

3. Interpolation. Combine the bigrams and unigrams to compute the probability.

4. Back off. If no bigram, use unigram instead.

- Data Processing

1. Add every sentence "<s>" in the front. And if it ends with ".", replace it with "<s>", if not add "</n>" denote unfinished sentence.

2. For word like the form "'s", "n't" and so on. We split it into two word. (eg:he's→he, 's; wouldn't→would, n't) And we define an stop words set **special_w** to store {"'s","n't" $\cdots$}. And when correcting later, we ignore words in it.

3. When output the answer, for normal word we print " " + word, for word equal to "<s>" or "</n>", we print nothing, for word equal to "</s>", we print ".", for other words in **special_w** we print word directly.

- Correction Details

When doing correction, I first find all words that not in vocab (ie: Non-word error), and correct them. After that, if number of corrections in one sentence matches the testdata's errors, skip to next sentence. If not, it's real word error, then we take every word as possible wrong word and try to correct them.(Note: see "Readme.txt")

[1] In every sentences, "<s>" means beginning, "</s>" means end while "</n>" means unfinished sentence.

III. Results

After I implemented all the code all above, then I run it on our testdata with four different smoothing methods.I got following results:

Smoothing methods	Run time	Accuracy
Add-1	148s	85.10%
Good Turing	155s	17.00%
Interpolation	158s	94.20%
Back off	159s	59.00%

IV. Result Analysis

1. From the results in the table above, we can see that the Interpolation smoothing method outperforms others in this problem. For Add-1 smoothing method, the accuracy reaches 85.10%, but the method leads to a big reduction in probability for the bigram or unigram with higher frequency.

2. For Good turing method, I am surprised that it has such low accuracy (maybe I was wrong in coding), afer checking the probability, I found in the Language Model, bigrams that only appear once have a rather large amount, then it may asign a great probability for the unusual bigrams.

3. For Interpolation method, After repeat tests I found when $\lambda_{bigram} = 1 - \lambda_{unigram}$, $\lambda_{unigram} = 10^{-8}$, the accuracy is near to its maximum. This means the power of unigrams is very little in this question. And this is also because unigram usually has much higher frequency than its bigram (eg: unigram["the"] » unigram[("the", "problem")]).

4. For Back off smoothing method, the accuracy only achieves 59.00%. I thought this is easy to be understood: For in 2. and 3. we have know that the power of unigram is very little which means when backing off, giveing unigram too much weights, may leads to low accuracy.

V. Summary

In this project, I received a lot by writing the Spelling Correction system. I learned how the Language Model and Channel Model works, and gained a deep understanding in features of different smoothing methods. Of course, I have also improved my coding skill in python. However the program still has a relatively long time (about 2.5 minutes for the testdata), I will work on improving it later.