Spell Correction Task

Jiaqi Wang
Harbin Institute of Technology
wjqkoko@foxmail.com

Abstract

Spell correction is applied in all aspects of our lives and has important significance to the real society. In this assignment, our approach is to use edit distance, n-gram language model and channel model to deal with the task, and finally got 96.20 % accuracy on ans.txt. Furthermore, we compared and analyzed the results in the dimensions of language model window sizes, edit distances, whether to use channel, and the influence of corpus.

1 Goal

Write a toy system for spelling correction.Both components of channel model and language model should be implemented as introduced in lecture 3.2. For evaluation, you will be provided a text file consisting of 1000 sentences extracted from news articles.

2 Environment

Conda 3.7, Python 3.7, Pycharm CE

3 Methodology

Given a sentence $X = \{x_1, x_2, ...x_n\}$ with n tokens as input, the spell correction task is to predict a correct sentence $W = \{w_1, w_2, ...w_n\}$, where w_i is the correct word. In this assignment, the input also has error numbers of every sentence on ans.txt. We use Bayesian formula $\widehat{w} = argmax_{w \in V} P(x|w)P(w)$, where $P(w)$ is the language model , $P(x|w)$ is the channel model estimating the probability that w_i is wrongly written as x_i and V is vocabulary. Our strategy is to correct the non-word errors firstly, and then correct real-word errors if the detected errors by non-word correction is less the corresponding error numbers from ans.txt.

3.1 data processing

dataloder.py contains functions to load data of the file into memory

It is necessary to pay attention to cleaning data, such as: case conversion, removal of symbols, numbers, etc. The functions are as follows:

- load-testdata(): load data from testdata.txt, return error numbers of every sentence and 1000 test sentences.
- load-edit(): load statistics for misspelled single characters. data source is https://norvig.com/ngrams/.
- load-vocab(): Load vocab from vocab.txt which requires to clean data.
  - **Removing symbols and numbers.** These vocabulary are of no help to the subsequent calculations, otherwise the accuracy will be reduced. Accordingly, the symbols and numbers appearing in the test sentences will be processed separately.
  - **Changing uppercase to lowercase.** Otherwise the word will not be found because of case mismatch, and the accuracy rate will also be reduced.
- load-ngram(n): load data in ans.txt to the dictionary of language model, n stands for n-gram.
- unigram-reuters(): load data to the dictionary of unigram model. data source is reuters in nltk. The dictionary is stored as file unigram-reuters.txt.
- bigram-reuters(): load data to the dictionary of bigram model. data source is reuters in nltk.The dictionary is stored as file bigram-reuters.txt.

- trigram-reuters(): load data to the dictionary of trigram model. data source is reuters in nltk.The dictionary is stored as file trigram-reuters.txt.

3.2 edit distance

Given two strings a and b on an alphabet Σ, the edit distance $d(a, b)$ is the minimum-weight series of edit operations that transforms a into b. The edit operations are as follows:

Insertion of a single symbol. If $a = uv$, then inserting the symbol x produces uxv. This can also be denoted $\epsilon \to x$, using ϵ to denote the empty string.

Deletion of a single symbol changes uxv to uv ($x \to \epsilon$).

Substitution of a single symbol x for a symbol $y \neq x$ changes uxv to uyv ($x \to y$).

Transposition of a single symbol x for a symbol $y \neq x$ changes $uxyv$ to $uyxv$ ($xy \to yx$).

edit.py: deal with problems related to edit distance.

- edit1(word): return the candidate words with an edit distance of 1
- edit2(word): return the candidate words with an edit distance of 2
- edittype(word,error): return the edit type between the two words, which is in $\{ins, del, sub, trans\}$ separately standing for insertion, deletion, substitution and transposition.

3.3 spell correction

There is a defined class $spellcorrection$ which specializes in spell correction in spell-correct.py.

The strategy of spell correction is divided into two steps: non-word correction and real-word correction. To correct the non-word errors, we should firstly detect the word not in vocab, and then explore all the candidate words of the wrong word, using language model and channel model to calculate the probability, finally select the candidate with the highest probability as the correct word. If the number of errors detected by non-word correction is less the the corresponding error numbers from ans.txt, correct the real-word errors whose steps are similar with non-word correction.

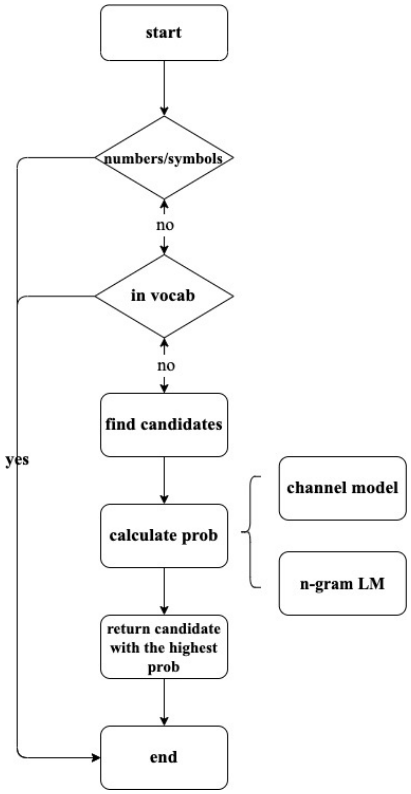

Figure 1: non-word correction flow chart

Language Model: $\{candidate, j, sentence\}$ where j represents the index of candidate in the sentence and a n-gram dictionary $ngram\_dict$ will be the input as we calculate the language probability from the dictionary

According to the choice of language model, it is divided into three types: $unigram$——calculate the probability of candidate, $bigram$——calculate the probability of candidate and its **one adjacent neighbor**, $trigram$——calculate the probability of candidate and its **two adjacent neighbors**.

If the window size in n-gram is greater than one, we need to consider the problem of sentence boundaries. If the phrase does not appear in the dictionary, return -1e3 by default.

Channel Model: Given the $\{candidate, word\}$ and a single-edit dictionary $edit\_dict$, we calculate the probability from the dictionary.

Use $edittype$ functions in **3.2** to return the type of edit, and find the corresponding probability in $edit\_dict$. If candidate and word are the same word, return 0.95 by default.

Word Correction: The structure of non-word correction is shown in Fig 1. We remove symbols,

	with channel	without channel	with channel	without channel
unigram/1	73.00	72.00	69.70	69.90
bigram/1	78.10	76.80	77.40	75.70
trigram/1	77.10	74.20	76.70	73.40
unigram/2	84.10	86.90	84.90	82.30
bigram/2	**96.20**	94.50	**95.00**	93.20
trigram/2	94.80	91.30	93.50	89.90

Table 1: the left two columns are results trained on ans.txt and the right two columns are results trained on geuters corpus from nltk

λ		smooth	
0.5	91.30	-1e2	79.50
1	94.10	**-1e3**	**94.10**
2	**95.00**	-1e4	91.20
5	94.20	-1e5	89.60

Table 2: λ is the balance parameter between language model and channel model, smooth is the probability when can not find the corresponding words in the dictionary

numbers and detect the wrong words not in the vocabulary. Then discover candidate words and calculate the corresponding probability to choose the correct word.

Discover candidate words: Find all words that may be transformed by one edit distance (insert, delete, substitute, transpose), and then choose the words exists in the vocabulary. If there is no suitable candidate word within the range of one edit distance, find all words that may be transformed by two edit distance and add them to the set of candidates.

Probability calculation: For language model, calculate the probability p_{LM} about the candidate word . For channel model, calculates the probability $p_{Channel}$ between the error word and the candidate. p_{LM} and $p_{Channel}$ are added to obtain the final probability p, then we will choose the candidate with the highest probability p as the final correct word.

Real word error : After non-word correction, if the error number is less than the corresponding error number from ans.txt, correct the real-word error in the sentence. Except adding the original word to candidates, the others are the same as non-word correction.

Finally, keep the capitalization of the correct word consistent with the original word and replace the word in the original sentence.

4 results

Table 1 and 2 show the experiment results with six different models on ans.txt .These model are different from four perspectives: n-gram, edit distances, whether to use channel model, the kind of corpus.

language model: unigram vs bigram vs trigram. Under the same setting of others, bigram performs better than unigram and trigram. The reason is that unigram may ignores contextual information and training data may be insufficient for trigram. Besides, we try serveral smooth parameters in Table 2 and find -1e3 is the best.

edit distance: 1 vs 2. Under the same setting of others, 2 performs better than 1. We suppose that the edit distance of 2 can better capture the correct candidate words.

channel model: with vs without. Under the same setting of others, with channel model performs better than with channel model.

corpus: internal corpus vs external corpus. Under the same setting of others,internal corpus performs better than external corpus. Table 1 shows the experiment results, model trained on ans.txt performs better than it trained on reuters from nltk.

Balance between language model and channel model. For better optimization, we use $\widehat{w} = argmax_{w \in V} P(x|w)P(w)^{\lambda}$ instead of bayesian formula. We experiment the parameter λ in table and found 2 is the best.

5 conclusion

In this assignment, we used edit distances, n-gram language model and channel model to deal with the spell correction task and got 96.2 $\%$ on ans.txt. For further experiment, we will consider using dynamic programming in real-word correction and balance the non-word and real-word correction strategy more cautiously.