A Chinese Word Correction Approach

LIU Hanwen

20463791, hliubp@connect.ust.hk

Guided by Prof. CHEN Lei

Hong Kong University of Science and Technology, Hong Kong

Abstract

There is a long history of English spelling correction, however, the researches of Chinese words correction are not that common. In this report, we propose a Chinese words correction system with detection and correction functions based on n-gram language model and Chinese text segmentation. The detection core focused on the continuous singletons while the correction core focuses on the the shape and pronunciation similarity of characters. According to the experimental results, our design performs acceptable detection and correction rates.

Keywords: NLP, CKJ processing, spelling correction, n-gram language model

1. Introduction

Spelling error is a nerve-wracking problem in articles and documents. To guarantee the quality of the articles and documents, words correction is extremely necessary. Manual word correction is a conman choice in last fifty years, however, as the rapid development of natural language processing (NLP), automatic methods are more and more popular in spelling correction field.

Based on western world's leading position of information technologies in early years, English spelling checkers and correctors have a very long history. While preparing high quality documents, Kernighan, Church, and Gale (1990)'s idea of 'noisy channel model' and 'correct' system have become a group of standard tools. However, when facing the same correction problem in Chinese, people are not able to present many methods. The study of Chinese words correction not only starts much later, but is also much more difficult than English spelling correction. Since Chinese is more likely to be ambiguous, the methods used in English spelling correction are not suitable for Chinese words correction. For example, one of the biggest differences between English spelling correction and the Chinese words correction is the segmentation problem. English words in a sentence are segmented while Chinese words are not. However, same words with different segmentation may have different meanings. Thus, Chinese words correction research progress is slow. In conclusion, a good Chinese spelling checker would be very popular among users and hold large potential profits.

In this research, we propose a Chinese words correction method which achieves two main goals,

1. Error words detection,
2. Error words correction.

Error words detection is to find out the error words in the sentences while correction is to find out the corresponding correct words to replace the error words. Our approach mixes the n-gram language model technology and the similarity measurement. An n-gram language model is a type of probabilistic language model for predicting the next word in such a words sequence in the form of a $(n`1)$–order Markov model. A similarity measurement tool can help our correction application find out similar candidates.

2. Related Work

Spelling correction was first proposed by Peterson for English in 1980[1]. And it can be mainly divided into single word and context-sensitive spelling correction technology. For the single word spelling error, it commonly uses dictionary-based method. It matches the original word with all the words in dictionaries to determine whether the word has spelling errors. For the context-sensitive spelling errors, there are two major kinds of processing methods: Rule-based methods and Statistics-based methods. Rule-based methods use some rules generated from relevant grammars, the collocation of words, syntactic knowledge, etc, for spelling correction.

Mangu and Bill (1997) proposed a transition-based learning method for spelling correction. Based on three key rules they found from training data, they builds a reliable and simple system for English spelling correction. It finds and ranks the candidates by the statistical model. Atwell and Elliott (1987) proposed a spelling correction approach using n-gram and part-of-speech language models. Cucerzan and Brill (2004)[2] presented a query spelling checker built by a query log, a trust dictionary and a noisy channel mode. As an improvement, a spelling error model learned from search query logs was proposed in 2005 by Ahmad and KondrakAhmad and Kondrak [3]. As the rapid improvement of the computing power, large scale language models are presented more and more frequently. In 2010, Gao et al. proposed a ranker based system for search query spelling correction. It makes full use of web scale language models and create many significant features for correction like surface-form similarity, phonetic-form similarity, entity, dictionary, and frequency features. Researchers from commercial enterprises are also paying much attention on this field for commercial profits. Both Microsoft and Google has developed services for spelling checking. Google's services offer a Java API which has well extensibility.

As for Chinese spelling correction, the first commercial product in the market on words correction is 啄木鸟(WoodPecker, Shih et al., 1992). However, it is only a error words detection application which does not has the ability to correct the error words. The detection procedure is composed of two major steps,

1. Pseudo-segmentation: Locate the characters (singletons) which can be associated with neighboring characters to form a polysyllabic (i.e., multi-character) words, based on an 80,000-word Chinese dictionary;
2. Scoring and deciding: For each singleton, compute a score using the scoring formula $S = F(fwd, f_1, f_2)$, where S is the singleton score, fwd is the word frequency of the singleton as a monosyllabic word, f_1, f_2 are the connection strength (mutual information between two characters) of the singleton combined with the right-hand and left-hand neighboring characters, respectively.

However, the precision of it is low which is only about 2.5%. What's worse, it is only a detection tool without correction. To solve these two problem, Chao-Huang Chang[4] presented a corrector with a character dictionary of similar shape, pronunciation, meaning and input-method-code to handle the spelling correction task. It is composed of two mechanisms,

1. Composite confusing character substitution;
2. Advanced word class 2-gram language model.

The characters in the input sentence are first substituted by their corresponding composite confusing character sets one by one. A composite confusing set is the collection of similar characters to a Chinese character from multiple views of shape, pronunciation, meaning, and input keystroke sequence. The substitution step produces several sentence hypotheses for the input sentence. Then, an advanced word class bigram language model, such as inter-word character bigram (IWCB) or SA-class bigram can be used for scoring each sentence hypothesis. Finally, the best scored sentence hypothesis is compared with the input sentence to determine the typos and their corrections. Experiments show that the proposed approach is very effective for handling with the two mentioned problems. Although Chang's approach has improved the correction technology, it can hardly deal with the insertion or deletion errors. Zhang and his teammate's work changed the situation in 2000. Their work distinguishes the matching schema between English and Chinese which significantly improves the result compared with Chang's approach. Segmentation can also be a tool for correction, for example, Huang et al. used CKIP to build the corrector. Besides, manually error templates are also useful, Hung et al. used them to build a strong corrector.

Chinese Pinyin is the most frequently-used input method, through generating the Pinyin error input model, Zheng ei al.[5] introduced a method based on a generative model and the typed wrong types to correct spelling errors. The characters with similar shapes are also leading to spelling error. Liu et al. (2011) has designed a similarity measurement of characters which considers extended Cangjie codes.

3. Chinese Spelling Error

There are four different types of spelling error in Chinese which are shown in table 1.

The first type of spelling error, similar pronunciation words, is the most frequent one in reality. It contains the homophone or characters with similar pronunciation. For example, the word '因[才(cai3)]施教' should be '因[材(cai3)]施教' where '才' and 材' have the same pronunciation and their Pinyins are the same (cai3).

The second type is characters with similar shape. Since Chinese keystrokes are complex and well-designed, some characters may have very closed shape with other character which leads to spelling error. For example, the word '[侯]车室' should be '[候]车室' where '侯' and '候' have similar shape.

The third type is words with similar meaning. Although this kind of errors are not very common, meaning errors are very hard to be detected and corrected. This kind of errors are words containing different characters will similar meaning. For example, the word '既往不[究]' should be '既往不[咎]' where '究' and '咎' have the similar meanning (punishment).

Table 1: Spelling error types

Error Type	Error Word Example	Correct Word Example
Similar pronunciation	因[才(cai3)]施教	因[材(cai3)]施教
Similar Shape	[侯]车室	[候]车室
Similar Meaning	既往不[究]	既往不[咎]

4. System Design

As a Chinese words correction system, it will be faced with two main tasks:

1. How to detect error words from the sentences?
2. How to correct error words?

In this section, I will talk about the framework of the design in detail and explain how can our work handle these two difficult tasks. I will divide the system design into Detection and Correction two parts.

4.1. Preparation: Words Segmentation Tool

Every work among CKJ(Chinese, Korean, Japanese) languages processing should have the segmentation as the first step because CKJ's essay is not segmented which is totally different from western languages'. To do segmentation, we choose an open-source toolkit Jieba with Python API.

Jieba Chinese text segmentation[6] is one of the best segmentation modules in Chinese segmentation field. According to the documentation, it has several features:

- Support three types of segmentation mode: Accurate Mode attempts to cut the sentence into the most accurate segmentations, which is suitable for text analysis. Full Mode gets all the possible words from the sentence. Fast but not accurate. Search Engine Mode, based on the Accurate Mode, attempts to cut long words into several short words, which can raise the recall rate. Suitable for search engines.
- Supports Traditional Chinese
- Supports customized dictionaries
- MIT License

Jieba is based on a prefix dictionary structure to achieve efficient word graph scanning. In additions, it builds a directed acyclic graph (DAG) for all possible word combinations and use dynamic programming to find the most probable combination based on the word frequency. What's more, for the words not mentioned in the dictionary, a HMM-based model is used with the Viterbi algorithm. However, note that we consider to turn off the HMM-based model detection because through the HMM-based model the error words may be segmented. It will affect our error words detection and the detail will be talked in the subsections below.

4.2. Preparation: KenLM

N-gram language model is one of the most popular language model. It is more reliable and concise than its competitors. We can query the model by the form of $P(w_n|w_1^{n-1})$ where w_1^n is a n-gram. Kenneth Heafield and his teammates improved the model estimation[7] and query[8] part of the model which named KenLM. The improvement is based on states pre-storing. In short, Kenneth gave out a state function

$$s(w_1^{n-1}) = (w_m^{n-1}, \{b(w_i^{n-1})\}_{i=m}^{n-1}$$

where where m is the minimal context from the paper's section 4.1 and b is the backoff penalty. The traditional method compute the observed entry with longest matching history is

$$P(w_n|w_1^{n-1}) = P(w_n|w_f^{n-1}) \prod_{i=1}^{f-1} b(w_i^{n-1}).$$

Thus, the probability $P(w_n|w_f^{n-1})$ is stored with w_n^f and the backoffs are immediately accessible in the provided state $s(w_1^{n-1})$. KenLM's features are listed below:

- Faster and lower memory than SRILM and IRSTLM.
- On-disk estimation with user-specified RAM.
- Two data structures for time-space tradeoff.
- Binary format with mmap. Or load ARPA files directly.
- If you have the appropriate libraries installed, it can also read text and ARPA files compressed with gzip, bzip2, or xz.
- Threadsafe.
- More opportunities for hypothesis recombination. If the model backs off, State stores only the matched words. The FullScore function also returns the length of n-gram matched by the model.
- Querying has few dependencies: a C++ compiler and POSIX system calls. Filtering and estimation are multi-threaded, so they depend on Boost.
- Supports models of any order greater than one (recompilation required for orders more than 7).
- Thorough error handling. For example, ARPA parse errors include a message, the problematic string, the byte offset, and the file name. Compare with IRSTLM.
- Loading progress bar.
- Tests. These depend on Boost.

- Querying supports n-grams containing ¡unk¿ tokens; these appear in models built with restricted vocabulary.
- Permissive license means you can distribute it unlike SRILM. There isn't a form to fill out before you can download.

Since this report is not major in n-gram language model improvement, we only make a brief introduction and if you are interested in it, you can Kenneth's website[9] for more information.

4.3. Preparation: Similarity Measurement

To correct the error words, similarity measurement can help us in candidates finding. In English, similarity measurement is extremely simple. Euclidean distance, edit distance and other similarity measurements can be implemented easily. However, the situation in Chinese is totally different. Chinese characters can hardly be implemented simple distance measurements. Therefore, we define a similarity measurement in Chinese characters based on their shapes and their pronunciations. Since the idea is come from SoundShape Code(SSC)[10], we name it SSC version 2 (SSC2). Our SSC2 has two parts, one parts is the similarity measurement of pronunciation while the other one is the shape. Actually, SSC2 is not a 'code', but a measurement algorithm. It abandons the 'code' form, calculates each part's weighted similarity instead.

Since all the characters' pronunciation(s) can be represented by Pinyin(s), the pronunciations similarity can be calculated by the nearest edit distance between two characters' Pinyins. Note that, one character may have more than one Pinyin, so we only consider two characters nearest Pinyins. It is very common for people in China to have some mistaken spelling patterns. For example, 'ing' often be spelled as 'in'. Thus, we consider that the Pinyins like 'ying' and 'yin' are the same. Actually in reality, huge part of Chinese pronounce and threat them the same. The patterns are shown below.

- 'ch' ≈'c',
- 'zh' ≈'z',
- 'sh' ≈'s',
- 'ing' ≈'in',
- 'eng' ≈'en',
- 'ang' ≈'an'.

Compared to the pronunciations similarity measurement, shapes similarity measurement is much more difficult. We consider three kind of similarities of shapes:

(a) Sijiao Code,
(b) Stroke Counts,
(c) Character Image Distance.

Sijiao code is a coding method which describe characters' shapes in the four corners: top left corner, top right corner, lower left corner, lower right corner. Different shape patterns of characters can be described with different Sijiao codes. It is created by Wang Yunwu in 1925. The characters with same Sijiao code may probably have similar shapes. For example, '候' and '侯' are similar and their Sijiao code(2728) are the same.

However, two characters with same Sijiao code may be very different with each other in some cases. For example, '日' and '量' have the same Sijiao code(6010), but they are totally different in shape. Thus, we also consider the number of strokes of characters and the character images.

We define the similarity as:

$$PronunciationSimilarity(PS) = \frac{Edit(PY_a, PY_b)}{AvgLength(PY_a, PY_b)}$$

where PY_x is the Pinyin of character 'x', $Edit(x, y)$ is the edit distance between item x and item y and $AvgLength(x, y)$ is the average length of item x and y.

$$SijiaoSimilarity(SJS) = \frac{Edit(Sijiao_a, Sijiao_b)}{4}$$

where $Sijiao_x$ is the Sijiao code of character 'x'.

$$nStrokesSimilarity(STS) = \frac{|n_a - n_b|}{AvgLength(n_a, n_b)}$$

where n_x is the strokes count of character 'x'.

$$CharacterImageSimilarity(CIS) = \frac{Size(Img_a \cap Img_b)}{Size(Img_a \cup Img_b)}$$

where $Size(x)$ is the elements count of set x and the Img_x is the binary image of character 'x'.

$$Similarity(a, b) = w_{PS} * PS(a, b) + w_{SJS} * SJS(a, b) + w_{STS} * STS(a, b) + w_{CIS} * CIS(a, b)$$

where w_x is the weight of item x and it requires $\sum w = 1$.

4.4. Detection

Most of the detection works usually use language models to score the text and make detection by finding the outliners which is also called model-based. Since segmentation is the first step of the CKJ processing, we can do the detection job with the help of Jieba segmentation module.

According to Jieba's documentation and source code, its main algorithm is based on a prefix dictionary structure to achieve efficient word graph scanning with a directed acyclic graph (DAG) for all possible word combinations. Thus, it can be threated as a model-based detection tool. Since most of the words in Chinese have the length over two, the continuous singleton words lying in the segmentation result are probably error words. Note that I have mentioned before, we turn off the HMM-based model function in Jieba which prevent it from detecting new word which may probably be the error word.

Thus, we propose a method of detection with Jieba. Define that the input Chinese text is $text_{in} = \{ch_1, ch_2, ..., ch_n\}$ where n is the length of the text. *Step1,* cut the input text into words by Jieba and then the $text_{input}$ will be divided into

$$Words = \{Word_1^{ch_1,ch_2,...,ch_{l_1}}, Word_2^{ch_{l_1+1},...,ch_{l_1+l_2}}, ..., Word_m^{...,ch_{l_{m-1}+l_m}}\}$$

where l_i is the length of the i^{th} word and the number of words is m.

The core logical framework of detection is shown in Figure1. For each $Word_i$ in $Words$, if it is a singleton word and not in frequent singleton words list, then our detection core will push it into a stack called $stack_{error}$ for temporary storing; if it is not, then our detection core will check $stack_{error}$, if $stack_{error}$ is not empty, it will send $stack_{error}$ to the corrector framework which will be talked about in next subsection. Finally, the output words will be original correct words and the corrected error words.

It may be a little bit hard to understand, but don't worry, here is an example:

Step 1. Raw input: [他必定胸多吉少，但是我仍然相信他能够逢胸化吉摆脱困境。]
Step 2. Segmented: [他/必定/胸/多/吉/少/，/但是/我/仍然/相信/他/能够/逢/胸/化/吉/摆脱/困境/。]
Step 3. Start: $stack_{error}$ = [], $output$ = []
Step 4. '他': Singleton - Frequent - $stack_{error}$ is Empty - $output$ = [他]
Step 5. '必定': Not Singleton - $stack_{error}$ is Empty - $output$ = [他必定]
Step 6. '胸': Singleton - Not Frequent - $stack_{error}$ = [胸]
Step 7. '多': Singleton - Not Frequent - $stack_{error}$ = [胸多]
Step 8. '吉': Singleton - Not Frequent - $stack_{error}$ = [胸多吉]
Step 9. '少': Singleton - Not Frequent - $stack_{error}$ = [胸多吉少]
Step 10. '，': Singleton - Frequent - $stack_{error}$ = [胸多吉少] - correction - $output$ = [他必定凶多吉少，] - $stack_{error}$ =[]
Step 11. '但是': Not Singleton - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是]
Step 12. '我': Singleton - Frequent - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我]
Step 13. '仍然': Not Singleton - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我仍然]
Step 14. '相信': Not Singleton - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我仍然相信]
Step 15. '他': Singleton - Frequent - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我仍然相信他]
Step 16. '能够': Not Singleton - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我仍然相信他能够]
Step 17. '逢': Singleton - Not Frequent - $stack_{error}$ = [逢]
Step 18. '胸': Singleton - Not Frequent - $stack_{error}$ = [逢胸]
Step 19. '化': Singleton - Not Frequent - $stack_{error}$ = [逢胸化]
Step 20. '吉': Singleton - Not Frequent - $stack_{error}$ = [逢胸化吉]
Step 21. '摆脱': Not Singleton - Frequent - $stack_{error}$ = [逢胸化吉] - correction - $output$ = [他必定凶多吉少，但是我仍然相信他能够逢凶化吉摆脱] - $stack_{error}$ =[]

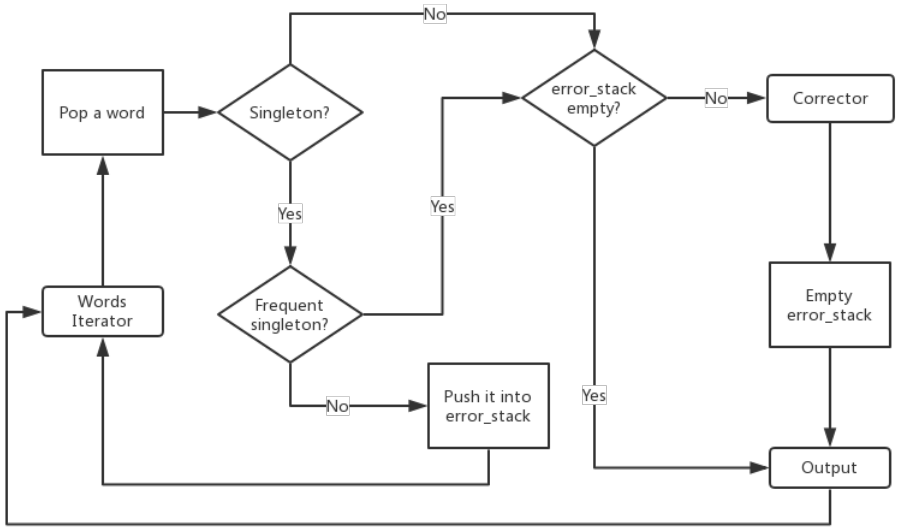

Figure 1: Detection core.

Step 22. '困境': Not Singleton - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我仍然相信他能够逢凶化吉摆脱困境]

Step 23. '。':Singleton - Frequent - $stack_{error}$ is Empty - $output$ = [他必定凶多吉少，但是我仍然相信他能够逢凶化吉摆脱困境。]

The frequent singleton words list is customized by users which contains words like personal pronouns, articles and so on. Note that the punctuation marks are also treated like a frequent singleton word. For example, the frequent words list can be [你/我/他/的/是/就/在/，/。/；...]. Since these singleton words will not be considered as an error word, our detection core are not able to detect the error containing these words. Though less words in this list may raise the marking rate, the false positive rate will be much higher. Thus, the frequent words should be most of the words which user may hardly spell them wrong.

4.5. Correction

I have mentioned the correction framework in the Detection section and in this section I will talk about it in detail. Before the main procedure of correction, we should first build a n-gram language model by KenLM. The corpus we used for training is a JSON file contains over 200MB Weibo contents(provided by Prof. CHEN Lei). The model is stored as a binary file on disk and we will load and query the model with its Python API. Note that before training, we extracted the contents from the JSON file and segment it with Jieba. Besides, We also created a vocabulary file.

The input of the correction framework is a group of singleton words which may be the error word. We define the singleton words sequence as Seq_{error}. Note that Seq_{error} may have more than one corresponding correct word. For example, if Seq_{error} = [ABCD], then the corresponding correct words combinations can be:

(a) 'ABCD',
(b) 'A"BCD',
(c) 'AB"CD',
(d) 'AB"C"D',
(e) 'A"BC"D'
(f) 'A"B"CD',
(g) 'ABC"D',
(h) 'A"B"C"D'.

Thus, our corrector will try all combinations when finding candidate words. For every combination, our corrector select all the possible words in the vocabulary file to measure their similarity with original error word. After measuring the similarity, it will pick out the words with highest similarity as candidates. Note that only the words with the same length as the error word will be selected to do the similarity measurement. Besides, the number of combinations is very large when facing with many singleton words which takes very long time to process. To deal with this situation, we have to do optimization. It is easy to find out that there are mountains of repetitive works when finding candidates. In the example above, case 3 and case 4 have same error word 'AB', so the similarity measurement calculation will be do twice. Thus, we set up a cache variable, the cache variable will temporary save the similarity values between each error word and the words in vocabulary file. Next time the corrector asks for similarity values will be much faster. According to our test, the time after optimization is only half of before in same situation.

All the candidates are similar with the original error words, but how to determine which one to replace the original error words? The answer is to combine all the candidates with prefix words respectively and find out which candidate has the highest probability standing there based on the n-gram model. First, Our corrector will ask the detection core for prefix n words, where n can be customized. The default n is 2, so the example below is based on $n = 2$. The selected candidate of this part will be:

$$argmax\{Score(p \cup c)|c \in C\}$$

where $Score(text)$ is the probability of $text$ queried from the KenLM language model we have built before, p is the prefix words set with length of n and C is the candidates set while c is each candidate in it.

Take the example shown in detection part, the prefix two words are p =['他', '必定'] and the candidates are C = ['凶多吉少', '熊多极少']. The selection will be '凶多吉少', because $Score$('他必定凶多吉少') > $Score$('他必定熊多极少'). *Note that it is only an example, in reality the candidates will be much more.*

5. Experimental Results

According to Chang's paper[4], the performance indices of a Chinese words correction system are defined as follows:

- **A**: the number of characters in the input text;

- **B**: the number of characters marked as errors by the system, (including false-alarms);
- **C**: the number of spelling errors corrected by the system;
- **D**: the number of (genuine) spelling errors detected by the system;
- **E**: the number of spelling errors in the input text;
- **B-rate**: B/A, marking rate;
- **P-rate**: D/B, precision rate;
- **D-rate**: D/E, detection rate;
- **C-rate**: C/E, correction rate.

Our KenLM language model is built by the 222MB Weibo contents which provided by Prof. Chen. The Sijiao codes of common characters are provided by the repository SimilarCharactor[11] on Github. The testing data contains four kind of documents:

- A speech draft: "We shall fight on the beaches" Chinese version - Winston Churchill;
- A regulation document: Section 1 of the party constitution of CCP;
- An encyclopedia document: Wikipedia page '香港' content;
- A short novel: Part of '围城' - Qian Zhongshu.

All the data are in the folder 'test_data'. The experiment method is:

Step 1. Load the test data.
Step 2. Correct the original text with our system.
Step 3. Randomly select 100 characters in the test data and change them to other characters.
Step 4. Correct the original text with our system.
Step 5. Record the results.

Step 2 is to find out the number of fault alarms. The number of fault alarms should be as low as possible. Note that selecting 100 characters to replace is based on the length of the test documents. Every group of test data will be tested 5 times.

Table 2 shows the results of our experiments.

The average correction rate is 45.25%. From the result we can infer that the low detection rate and the high fault alarms rate make the average correction rate lower than we excepted. However, the correction rate of right detection cases is about 84.60% which is acceptable. Note that actually the experiments results are not that compellent because the scale of our test data is too tiny compared to other researches. However, the performance of the corrector will only better than these experimental results and here are the reasons. First, the number of characters is about 10% of the test document which is too large because it is almost impossible that 10% of document

Table 2: Experimental Results

Test Document	A/B/C/D/E	B-Rate	P-Rate	D-Rate	C-Rate
Speech	1513/76/41/45/100	5.02%	59.21%	45.00%	41.00%
Party constitution	2249/102/57/59/100	4.53%	57.84%	59.00%	57.00%
Wikipedia	1184/88/38/44/100	13.45%	50.00%	44.00%	38.00%
Part of '围城'	10000/222/45/70/100	2.22%	31.53%	70.00%	45.00%
Average	6946/122/45.25/54.5/100	7.02%	44.68%	54.50%	45.25%

are spelling errors in real life. Second, because of the tiny length of the test data, 10% characters replacing may only generate 100 errors which is a small-scale. Third, the characters replacing may create less than 100 errors. For example, if two characters replacing occurs in same one original word, errors will be only one. Fourth, the characters replacing on frequent words like '的' is not useful for testing our system because our system will avoid these errors. This kind of errors are also not frequent in reality.

6. Future Work

Although the performance of our correction system is acceptable, there are some problems which should be solved in the future.

First of all, the correction speed of out design is extremely low. We have designed some elaborate algorithm to speed up the system, for example, we calculated common characters' similarity values with each other in advance. However, the combinations number is large and the comparison times are many, so the total speed of correction is very low. Because of the low speed, we can only test our corrector with small test data which may not be compellent.

The next problem is the detection accuracy. We have noticed that if there are several singleton words appear which are not error words, the detector will treat them as error words since this is how our detection algorithm works.

The last problem is the candidates selecting algorithm in the correction part. We select the candidates by combining the candidates with the prefix words and query the language model for score. However, the score of candidates with longer length may probably get higher score. For example, when correcting the sentences '平民刘备鉴持卟鞋', the prefix words are '平民' and '刘备' and the candidates are '坚持/补鞋' and '坚持不懈'. Although the first candidate is more similar, but the score of the first candidate may probably be lower than the second one, because the 4-gram score usually lower than 3-gram score.

7. Conclusion

In this report, we present a Chinese words correction system with detection and correction functions based on n-gram language model and Chinese text segmentation. According the experimental results, our design performs acceptable detection and correction rates. Although the design is still an unshaped product with some problems, it can detect and correct simple error words with accuracy and speed.

The project is still in progress. If you have any question, please feel free to contact us. The code is also hosting on Github(https://github.com/liu-hanwen/ChineseCorrector), you can make your contributions on it.

8. Acknowledgement

This project is an independent project of HKUST MSBD. Thanks Prof.Chen and RAs for their help and I am grateful cooperating with my teammate CHEN, Jinren.

9. Reference

[1] J. L. Peterson, Computer programs for detecting and correcting spelling errors, Communications of the ACM 23 (1980) 676–687.

[2] S. Cucerzan, E. Brill, Spelling correction as an iterative process that exploits the collective knowledge of web users, in: Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing.

[3] F. Ahmad, G. Kondrak, Learning a spelling error model from search query logs, in: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics, pp. 955–962.

[4] C.-H. Chang, A new approach for automatic chinese spelling correction, in: Proceedings of Natural Language Processing Pacific Rim Symposium, volume 95, Citeseer, pp. 278–283.

[5] Y. Zheng, C. Li, M. Sun, Chime: An efficient error-tolerant chinese pinyin input method, in: IJCAI, volume 11, pp. 2551–2556.

[6] fxsjy, Jieba - github.com, `https://github.com/fxsjy/jieba`.

[7] K. Heafield, I. Pouzyrevsky, J. H. Clark, P. Koehn, Scalable modified Kneser-Ney language model estimation, in: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, Sofia, Bulgaria, pp. 690–696.

[8] K. Heafield, KenLM: faster and smaller language model queries, in: Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation, Edinburgh, Scotland, United Kingdom, pp. 187–197.

[9] K. Heafield, Kenlm homepage, `https://kheafield.com/code/kenlm/`.

[10] D. China, A chinese text similarity measurement algorithm based on ssc, `https://blog.csdn.net/chndata/article/details/41114771`.

[11] contrl4, Similarcharactor, `https://github.com/contr4l/SimilarCharactor`.