SuperOCR for ALTA 2017 Shared Task

Yufei Wang
Computer Science and Engineering
UNSW, Sydney, Australia
yufei.wang@student.unsw.edu.au

Abstract

This paper describes the SuperOCR system submitted for the ALTA 2017 shared task, which aims at correcting noisy OCR output for the Trove database. We used heuristic rules and patterns in submitted system and we apply language model to further improve our system. Experiment shows that language model plays an vital role in performance. Surprisingly, a tri-gram language model outperforms LSTM language model in this task.

1 OCR Post-Correction

ALTA 2017 shared task [1] aims at Optical Character Recognition (**OCR**) post-correction for Trove database (Holley, 2010) [2]. OCR extracts text from image, allowing further language analysis. However, OCR is inherently error-prone, in particular for old scanned documents. High quality OCR analysis result benefits downstream NLP task, including named entity recognition (NER) (Mac Kim and Cassidy, 2015) and information extraction (Taghva et al., 2006). In this shared task, given a set of OCR raw-correction pairs, participators are required to build a system to automatically and accurately correct the OCR-ed documents.

Our submitted system achieved averaged F1 score 16.82%, which is 2^{nd} best system. Later, we further improved our submitted system to 20.72% by using context and weighted OCR error information. However, The wining system had achieved averaged F1 score 32.99%, indicating that our system still has large margin to be improved.

Our submitted system mainly targeted in (a) correcting word-level errors, e.g. words with characters being dis-recognized during OCR process, and (b) delete frequent noisy text pattern. For (a), we first filtered out normal words using vocabulary list; then we applied correction to those error-like tokens. For (b), we extracted the frequent corrected patterns from the aligned documents. The experiment result shows that language model and high-quality vocabulary list are vital to boost performance. Surprisingly, a simple tri-gram language model outperforms a state-of-the-art LSTM language model in selecting candidate words for correction.

This paper is organized as follows: Section 2 profiles data set and OCR errors. Section 3 introduces submitted system and improved system. Section 4 shows the experiment result. We summarize our finding in Section 5.

2 OCR Documents and Errors

In this section, we first summarize some basic statistics of the OCR documents from this shared task in Table 1.

#Docs	6000
#avg. Words	571.9
#avg. Errors	39.6
#Error Ratio	6.93%

Table 1: Statistics of Share Task Data

We are also interested in the types of correction made by annotators in this data set. To extract the correction, we first align document pairs and unaligned words in raw documents are the corrections. We characterize them largely based on the vocabulary list used in our final system. We refer words in the list to "in-vocabulary" (**IV**), otherwise "Out-of-Vocabulary" (**OOV**). We categorize these corrections into the following types:

- Single Word Split: Split an IV into two

[1] http://www.alta.asn.au/events/sharedtask2017/description.html
[2] http://trove.nla.gov.au/

OOVs. **21.89%**

- Single Word Correction: Change an OOV to an IV. **18.16%**
- Words Deletion: Delete words from OCRed Text. **12.79%**
- Multiple Words Merge: Merge multiple words into a single IV. **8.05%**
- Punctuation Transform: Change a punctuation to another punctuation. **4.52%**
- Known Word Correction: Change an IV to another IV. **4.18%**
- Unknown Word Modify: Change an OOV to another OOV. **3.95%**
- Character Case: Change the character case of a word. **1.44%**
- Other: Other Corrections **25%**

It should be noted that, the correction Single Word Split often splits valid words into two OOVs randomly, for example, randomly modifying "yesterday" to "yes" and "terday"; modifying "Australian" to "Austra" and "lian" etc. We suspect that this is caused by a text processing errors. The correction distribution also shows the difficulties of this shared task as only 6.93% of words are modified while majority of words remain unchanged. Even worse, around 25% of errors are multiple words correction, which cannot be solved by checking single words.

Lastly, we analysis the length of the continuous corrected word sequence in the data set. As shown in Table 2, around 75% errors are length 1. Therefore, most of OCR errors are not continuous and separated by context words.

Len.	1	2	3	4	5+
%	74.9%	13.2%	5.7%	3.9%	2.3%

Table 2: Error Length Distribution

3 System Description

There are two important factors to consider when designing system:

- **Candidate Filtering** In our data, only 6.9% of words are corrected by annotators, which means that we are facing an imbalanced situation. If we apply correction to every word in document, it will generate a lot of false positive examples. Our intuition is that, a valid word is unlikely to be an error in OCRed text.
- **Independent Correction** As shown previously, most of errors are isolated by their unchanged context. Correction using sequence modeling may not be helpful as the dependency between errors are weak. Therefore, we correct words individually in our system.

Therefore, we design our system as shown in Alg 1. The system includes following post-correction components:

- ProcRawText: Correct frequent errors and split text into tokens.
- Word Filtering: Filtering out most of correct words in OCRed text.
- Correction: Correct an OOV word into a non-OOV word.
- ProcessKnownWord: Correct a non-OOV word to another non-OOV word.

Both of our systems follow the above framework and they only differ in strategies used in each component.

Algorithm 1: System Framework

```
Data: OCRed Text
$WordList$ = ProcRawText($OCR\_Text$);
Create $CorrectedList$ ;
foreach $w \in WordList$ do
    if $w$ should be Corrected then
        $w$ = Correction($w$);
    else
        $w$ = ProcessKnownWord($w$);
    Add $w$ To $CorrectedList$;
return TextJoin($CorrectedList$) ;
```

3.1 Submitted System

In our submitted system, we mainly used heuristic rules and patterns obtained from training data to correct text.

In *ProcRawText* part, we used three strategies:

1. Deleting a set of errors patterns with format "-* " and "- * " where "*" stands for one of "$<$ $*$ $>$ i j : ? ! 1 ; l". These patterns are the most frequent deleted error sequences in training pairs. Interestingly, these patterns often result in valid words being splitted and their character shapes are similar to each other. So these errors may be caused by similar noise in image input.

2. Splitting text based on white space and removing length one tokens except for "a" and "A". These length one tokens tend to be noise in the data set. Although "I" is indeed a valid word, our experiment result shows that there are much more noisy "I" than the valid one. So, we remove it as well.

3. the leading and following non-alphabet characters are removed from each word except for the following punctuation "." and ",". We skip numbers and punctuation in this step.

In $Word\ Filtering$ part, we constructed a vocabulary list by merging the most frequent 15000 words from corrected OCR documents in training data and most frequent words 10000 from 1 Billion Word Language Model Benchmark [3]. Given the vocabulary list, OOVs or words with most three non-alphabet characters (words with four or more non-alphabet characters are too noisy to be corrected) are selected as correction target. All other words remain unchanged.

In $Correction$ part, we extracted frequent single correction pairs in training data (e.g. "tne" $\Rightarrow$ "the"). If a candidate matched one of the pairs, we would correct it. Otherwise, we exhaustively searched for words in vocabulary list that are k edit distance from the correction target. We refer these two methods as **word-level correction** and **exhaustive correction** respectively. Finally, we will correct the character case based on the original word shape.

We skipped $ProcessKnownWord$ stage in our submitted system.

3.2 Language model enhanced System

This system was submitted after the shared task. Following (Tursun and Cakici, 2017), we used tri-gram KenLM (Heafield et al., 2013) [4] language model. Experiment result shows that tri-gram language model is sufficient for this task. To train the language model, we used the corrected documents in training data and lower-cased all words before the training. Intuitively, language model would capture the context information and therefore, it is helpful when we rank the correction candidate.

In $ProcRawText$ part, we still used strategy 1 and 2 in submitted system. However, we changed the tokenization method to the one used in the provided evaluation script. This method splits punctuation from words and provides better boundaries between words and punctuation, but, this could potentially lead to inappropriate punctuation split in noisy text. For example, given noisy word "-Mr." whose ground truth of is "Mr.", our method splits it into "-Mr" and ".", making it impossible to merge the punctuation back. To tackle this issue, we collected the frequent cases from training data to correct the errors before tokenization.

In $Word\ Filtering$ part, we constructed the vocabulary list by combining non-singleton words that are no shorter than 5 in training corrected text and words that are no longer than 4 in 1 Billion Language model benchmark.

In $Correction$ part, we additionally applied character error information to suggest correction candidates. We first extracted all Single Word Correction pairs (18.16% of all errors) and then aligned each of them in character level. We refer this as **character-level correction**. Besides, we corrected an OOV by merging or splitting if we can obtain IVs.

To combine both language model and word/char transformation information, we applied "Noisy Channel Model" (Mays et al., 1991) to select optimal candidates. Formally, we tried to find the optimal word c for correction target w such that it maximize $P(c|w)$, as shown in 1:

$$\arg\min_{c} Pr(w|c) * Pr(c) \quad (1)$$

where $Pr(c)$ is the language model score indicating how likely it should be there given the context; while $Pr(w|c)$ is the error model indicating how like w is an error of c. To unify our character-level, word-level and exhaustive correction, we grouped error pairs with same correction target together and normalize their count as weight. We always assigned exhaustive correction a constant weight. Note that, we only applied character transformation once to each word, the combination of trans-

[3]http://www.statmt.org/lm-benchmark/

[4]https://github.com/kpu/kenlm

formation have not been considered here.

Given a word, its context window and candidates list, (a) we calculated the language model score for all candidates and original word, with context, which is a window with size of 5, which includes itself and its previous and following two context words. Candidates words that receive higher score than original word are chosen for comparison using above model. If no candidate words get higher score than the original word, we remained original word unchanged.

In $ProcessKnownWord$, our improved model applied known word transformation correction by using the frequent patterns in training data. If a known word was found in the patterns, we used the above Noisy Channel Model to decide if we should make correction.

4 Experiment Result and Discussion

In this section, we show two performance measure, the randomly sampled development set (**Dev.**) and final test set (**Test.**) performance for our systems. During system development, we split provided documents into 5500 documents for training and 500 documents for validation. The performance is shown in Table 3.

System	Dev.	Test.
Sub Sys.	17.72%	16.82%
LM-Imprved Sys.	20.68%	20.72%

Table 3: Performance for both System

Table 3 shows that the language model boosts system performance by around 4%. Additionally, our new system no longer suffered from overfitting as the submitted system did. This indicates that the context information provided by language model surpassed hand-crafted rules in earlier system.

4.1 Ablation Study

To show the effectiveness of each components, an ablation study is conducted for our final system in this section. Note that all reported performance is based on development data set.

Table 4 shows that language model is the most vital component in the system. This shows the importance of context modeling components for spelling correction task. In addition, the contribution of the vocabulary list cannot be neglected. We manually investigated the vocabulary of corrected OCR text and found that low frequent words tend to be noisy. Many of these low frequent words should have been corrected during the annotation process. This indicates that the quality of training data needs to be improved.

System	ave. F1	$\triangle$
Full Sys.	20.68%	-
- LM	9.51%	**-11.17%**
- Vocabulary	15.08%	-5.6%
- Prepossess	18.85%	-1.83%
- Multi Word & Trans.	20.23%	-0.45%
- Wegt. Error	20.53%	-0.15%

Table 4: Ablation Performance. **LM**: Set language model score to be 1.0; **Vocabulary**: Only using words in corrected training data; **Prepossess**: Disable patterns and word cleaning in beginning; **Multi Word & Trans**: No word merge split and known word trans correction; **Wegt. Error**: Removing the Error Model, all weights for correct candidates are 1.0.

4.2 Optimal Language Model

In our final system, we used a tri-gram traditional language model. Will higher order language model or advanced neural model continuously improve the performance? We conduct an experiment regarding the order of language model in this section.

In this experiment, we applied a language model based on LSTM (Hochreiter and Schmidhuber, 1997). Comparing with transitional language model, LSTM language model can be viewed as ∞-order because LSTM can capture long-range dependent information using the cell and hidden information (Hochreiter and Schmidhuber, 1997). In our experiment, we used the tensorflow implementation [5] of (Kim et al., 2016). We did not change the default parameter setting in the source code as they are optimized based on English Penn Treebank (PTB)(Marcus et al., 1993). We used the corrected documents with the same text prepossess technologies as we train KenLM. We also experimented with uni-, bi-, 4-, 5-, 6-gram KenLM for comparsion. Note that, we used LSTM language model in the same way as we use KenLM.

The performance for different language models

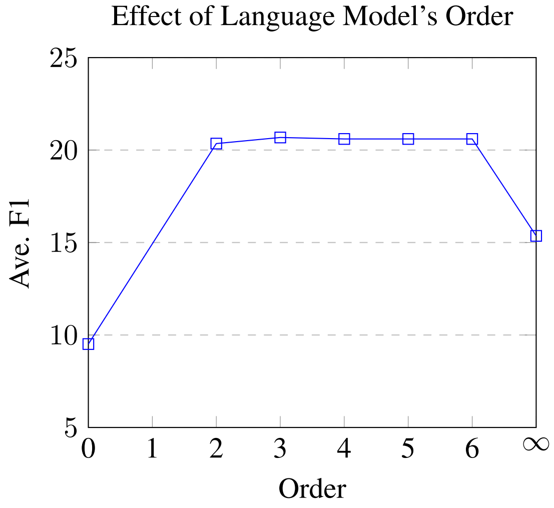
[5]https://github.com/dhyeon/character-aware-neural-language-models

are shown in Fig. 4.2. As the result show, bi- and tri-gram language models have already provided satisfying performance. Higher order language model even slightly decrease performance by 0.1%. Surprisingly, the LSTM-based language model dramatically decrease the performance by over 5%. During the LSTM model training, we monitored average perplexity over development set. The final model performance is around 80 perplexity which is a reasonable performance compared with (Kim et al., 2016), showing no overfitting in model training. We argue that two possible reasons could explain this:

1. The training data for neural networks is too noisy. It has been shown that neural networks cannot work well when training on noisy data. (Natarajan et al., 2013)

2. In the task of OCR post-correction, correcting errors only require nearby words, rather than long-dependency information would provide noisy information.

We can conclude that bi- and tri-gram language model are the optimal choice for OCR post-correction task.



5 Conclusion

We applied heuristic rule and patterns to the task of OCR post-correction. We further apply language model to boost the performance. Our system finally achieve average F1 score 20.68%, a 2_{nd} score in all submitted systems. Experiment result suggest that 3-order language model is more capable in modeling context information than state-of-the-art LSTM-based language model when ranking correction candidates.

Acknowledgments

We would like to thank Prof. Wei Wang for discussion, Dr. Stephen Wan and Dr. Diego Moll-Aliod for their time in proofreading this paper. We would also like to thank the organizers of the shared task for their support.

References

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Rose Holley. 2010. Trove: Innovation in access to information in australia. *Ariadne* (64).

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Sunghwan Mac Kim and Steve Cassidy. 2015. Finding names in trove: Named entity recognition for australian historical newspapers. In *Proceedings of the Australasian Language Technology Association Workshop 2015*. pages 57–65.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Eric Mays, Fred J Damerau, and Robert L Mercer. 1991. Context based spelling correction. *Information Processing & Management* 27(5):517–522.

Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. 2013. Learning with noisy labels. In *Advances in neural information processing systems*. pages 1196–1204.

Kazem Taghva, Russell Beckley, and Jeffrey Coombs. 2006. The effects of ocr error on the extraction of private information. In *Document Analysis Systems*. Springer, volume 3872, pages 348–357.

Osman Tursun and Ruket Cakici. 2017. Noisy uyghur text normalization. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*. pages 85–93.