Lexical Normalization of Twitter Data

Yitao Deng

1. Introduction

The aim of this report is to evaluate several correction methods for normalizing Twitter data. The normalization means changing words from Twitter, which is out of English dictionary, to words found in the dictionary.

Several studies [Justin Zobel and Philip Dart (1996); Navarro Gonzalo, et al (2001)] show that both orthography and phonetics methods are suitable for approximate string matching (ASM). The normalization of Twitter Data is an application of ASM, and both methods were applied in this report. Orthography methods included Levenshtein Distance, Damerau-Levenshtein Distance and N-Gram Distance. Phonetics methods contained Soundex and Editex. Combinations of both methods were also implemented. Three main evaluation metrics were applied to assess the quality of normalization. Three custom strategies were added into the normalization to promote the quality of correction methods.

The Twitter data used in normalization was obtained from Bo Han and Timothy Baldwin (2011). It was separated into separated single-word in order to be easily processed. The twitter data, separated tokens and English dictionary used by this report could be accessed from here[1].

2. Evaluation Metrics

There are three main evaluation metrics in this report: Accuracy, Precision and Recall.

2.1.Accuracy

Every normalized token has a prediction list containing one or more words after the normalization. However, words in list maybe not the canonical form for this token. If the **only** word in one-word list, or the **first** word in multi-words list is the canonical form of this token, the normalization of this token is **accurate**.

The Accuracy is defined as follows.

$$\text{Accuracy}=\frac{\text{NA}}{\text{N}}$$

NA:the number of accurately normalized token

N:the number of normalized token

2.2 Precision

If the prediction list of one token contains the canonical form of this token, no matter which position the canonical form locates at in the list, the normalization of this token is **right**.

The Precision is defined as follows.

$$\text{Precision} = \frac{\text{NP}}{\sum_{n=i}^{N} L_i}$$

NP:the number of rightly normalized token

L_i : the number of word in the prediction list of normalized token i

N:the number of normalized token

2.3 Recall

The Recall is defined as follows.

$$\text{Recall} = \frac{\text{NP}}{\text{N}}$$

NP:the number of rightly normalized token

N:the number of normalized token

2.4 Others

The runtimes are measured under the similar operating environment and used to make sure methods are time-affordable. The total accurate token, total right token and total predicted word are also recorded for detailed comparison.

3. Orthography Methods

3.1.Levenshtein Distance(LD)

Levenshtein Distance is defined by Levenshtein, Vladimir I (1966). It measures the minimum

[1] https://app.lms.unimelb.edu.au/bbcswebdav/pid-6149963-dt-content-rid-24614018_2/xid-24614018_2

number of single-character edits, including insertions, deletions and substitutions, in order to change one word into the other. The scores of edits (match, insertion, deletions, substitution) is (0, 1, 1, 1) for standard Levenshtein Distance.

3.1.1.Basic

This method library for Python3 is from Antti Haapala (2014). For each normalized token, LD was applied to find the prediction list containing words with least distance in dictionary. Each word in list was compared to the canonical form of this token to calculate the evaluation metrics. (See Table 1)

TotalAccToken	209	Accuracy	0.111111
TotalRightToken	430	Recall	0.228601
TotalPreWord	18283	Precision	0.023519
TotalToken	1881	Runtime/s	208.089

Table 1: Results of Levenshtein Distance

3.1.2.Limited Prediction List

The average predicted words of each token in Table 1 is around 10, which makes Precision to a low rate. Prediction list was cut-offed at 5th to improve this matric. (see Table 2)

TotalAccToken	209	Accuracy	0.111111
TotalRightToken	322	Recall	0.171185
TotalPreWord	5911	Precision	0.054474
TotalToken	1881	Runtime/s	204.527

Table 2: Results of Levenshtein Distance with limited prediction list length

The Precision in Table 2 is doubled compared to Table 1, as a result of half prediction list length. However, the Recall is reduced by 23%, which means about 100 canonical forms are dropped by the truncation.

3.2.Damerau-Levenshtein Distance(DLD)

Damerau-Levenshtein Distance [Bard, Gregory V. (2007)] is a variant of LD. One additional operation – transposition of two adjacent characters – is added into the operation edits.

This method library for Python3 is from Geoffrey Fairchild (2016). In this case, (1,1,1,1) was used for the score tuple (insertion, deletion, substitution, transposition). Limited list length was also implemented. (see Table 3)

No limit		Cut-off at 5th	
Accuracy	0.111642	Accuracy	0.111642
Recall	0.229133	Recall	0.169590
Precision	0.023416	Precision	0.053921
Runtime/s	447.896	Runtime/s	461.539

Table 3: Results of Damerau-Levenshtein Distance (no limit and cut-off at 5th)

The results are quite similar to Table 1&2. Since transposition is the only additional operation added into DLD compared to LD, and it often occurs as typo, the similar results could show that there is few typo in this Twitter data.

3.3.N-Gram Distance (NGD)

N-Gram Distance [Navarro Gonzalo, et al (2001)] is based on n-gram, which is a contiguous sequence of n characters in a string. Each string has its own set of n-gram, and NGD between two strings could be calculated from the size of two set of n-gram and their intersection.

This method library for Python3 is from Graham Poulter (2017). Both 2-Gram Distance and 3-Gram Distance were calculated, and cut-off at 5th and 3rd were also applied. (see Table 4)

N		Cut-off at 5th	Cut-off at 3rd
2	Accuracy	0.176501	0.176501
	Recall	0.225412	0.212121
	Precision	0.058717	0.087673
	Runtime/s	291.514	304.483
3	Accuracy	0.171185	0.171185
	Recall	0.225943	0.211057
	Precision	0.058856	0.087233
	Runtime/s	170.125	176.819

Table 4: Results of N-Gram Distance (cut-off at 5th and 3rd)

Compared to LD and DLD, NGD has good improvement on the Accuracy and cut-off does not obviously affect the Recall. The reason for that is NGD has higher distinction for distance, which means few words has same distance to the same token. That contributes to a better ranked prediction list.

For LD and DLD, sometimes there exist more than 10 words with same distance to this token, so that the canonical form has small chance to be the first word in list.

4. Phonetics Methods

Soundex and Editex [Zobel Justin and Philip Dart (1996)] were implemented in this report. Both of them are based on translation table. The table separates the alphabet (a-z) into several subsets based on sound of character, and these subsets are translated to corresponding numbers (0-9), respectively. Then a string can be converted to a sequence of number according to the table.

The translation tables are different between these two method. Soundex is a no ranking method, and only gives a result whether strings are similar. It is unsuitable to do normalization individually and combined with LD in section 5. Editex can produce a ranked distance by applying recurrence function.

These method libraries for Python3 are from Chris Little (2015). Due to the runtime, only one Editex test without limited list length was undertaken. (see Table 5)

TotalAccToken	270	Accuracy	0.143540
TotalRightToken	363	Recall	0.192982
TotalPreWord	7415	Precision	0.048954
TotalToken	1881	Runtime/s	67439.1

Table 5: Results of Editex without limit

Compared to LD, the Accuracy and Precision increase largely. That means Editex performs better than LD and DLD, which represents a better ranked prediction list. The result consists with the fact that Twitter user prefer to type abbreviation of words based on sound rather than the whole words.

5. Combined Methods

LD and DLD present a good improvement on the Precision by direct cut-off of prediction list, but lose the Recall because of poorly ranked list, mentioned in Section 3.3. Since NGD and Editex provide better ranked list, applying them on the LD and DLD prediction list rather than direct cut-off, would avoid the Recall decrease and improve the Accuracy.

Several combined methods were applied[2]. Results are showed in Table 6.

	LD-2gram	LD-3gram	LD-Editex
Accuracy	0.182881	0.181818	0.135566
Recall	0.214779	0.214779	0.160552
Precision	0.069679	0.069691	0.079536
Runtime/s	268.816	260.907	258.168
	DLD-2gram	DLD-3gram	LD-Soundex
Accuracy	0.178628	0.180223	0.070175
Recall	0.220095	0.216374	0.105263
Precision	0.071342	0.070148	0.026052
Runtime/s	443.295	456.889	276.999

Table 6: Results of Combined Methods (cut-offed at 5^{th})

Compared to LD and DLD, all results of LD-NGD and DLD-NGD acquire large improvement on Accuracy, Recall and Precision, even slightly better than NGD. It shows that they optimize prediction list without Recall decrease.

LD-Editex and LD-Soundex do not give good results, indicating it is unsuitable to follow a phonetics method after orthography method, because being similar in spelling can largely differ in sound.

6. Custom Strategies

Three custom strategies based on Twitter user habits were introduced to promote the performance of normalization.

6.1.Add words into the dictionary

The Twitter data shows that words like 'lol', 'rt', 'gotta', 'hehe', 'idk' and 'imma' are highly used in Twitter, and their canonical form are themselves. However, they are not included in the original dictionary. Adding these words into the dictionary would lead to large increase of Accuracy and Recall.

6.2.Add token itself into prediction list

Tokens containing - and ' have a high chance to be the canonical form of itself. The same to tokens ending with 'ing', 'ed', 's' and 'es'. Tokens ending with 'in' are used to represent 'ing'. Thus,

[2] LD-2gram means LD followed by 2-gram; the rest can be interpreted in the same way.

tokens which match the regex of /*ing*$|*ed*$|*s*$|*es*$|-|\'/ were inserted into the first position of prediction list. The same to tokens ending with 'in' after changed to end with 'ing'.

6.3.Add first matched word into prediction list of specific tokens

Tokens like 'you', 'to', 'thanks' and 'are' are highly represented by 'u', '2', 'thx' and 'r', because of similar sound and convenience. They cannot be converted by neither orthography methods nor phonetics methods. Directly adding the word into the first position of prediction list would be a solution.

6.4.LD with custom strategies

LD with custom strategies was applied to verify the performance. (See Table 7)

No-limit		Cut-off at 5th	
Accuracy	0.364699	Accuracy	0.364699
Recall	0.518341	Recall	0.452950
Precision	0.059771	Precision	0.144089
Runtime/s	247.063	Runtime/s	232.959

Table 7: Results of Levenshtein Distance with custom strategies (no limit and cut-off at 5th)

All metrics are significantly promoted by custom strategies, doubling each metrics compared to previous best methods. These three custom strategies are extremely useful in normalization of Twitter data.

7. Conclusion

This report estimated orthography methods (Levenshtein Distance, Damerau-Levenshtein Distance and N-Gram Distance), phonetics methods (Soundex and Editex) and combination of both by three main metrics (Accuracy, Precision and Recall) in lexical normalization of Twitter data.

Editex performs better than LD and DLD. NGD presents a best quality among applications of individual method. LD followed by 2-gram improves LD largely and surpasses NGD.

Three custom strategies based on Twitter user habits significantly promote the performance of LD and double each metrics compared to LD-2gram.

References

Antti Haapala. 2014. *python-Levenshtein 0.12.0*. https://pypi.python.org/pypi/python-Levenshtein

Bard, Gregory V. 2007, Spelling-error tolerant, order-independent pass-phrases via the Damerau–Levenshtein string-edit distance metric, In *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers: 2007, Ballarat, Australia, January 30 - February 2, 2007*, Conferences in Research and Practice in Information Technology, 68, Darlinghurst, Australia: Australian Computer Society, Inc. pp. 117–124

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: Maknsens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Portland, USA. pp. 368–378.

Chris Little. 2015. *abydos 0.2.0.* https://pypi.python.org/pypi/abydos

Geoffrey Fairchild. 2016. *pyxDamerauLevenshtein 1.4.1*. https://pypi.python.org/pypi/pyxDamerauLevenshtein

Graham Poulter. 2017. *ngram 3.3.2*. https://pypi.python.org/pypi/ngram

Justin Zobel and Philip Dart. 1996. Phonetic string matching: Lessons from information retrieval. In *Proceedings of the ACM-SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland. 166–173.

Levenshtein, Vladimir I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*. 10 (8): 707–710.

Navarro Gonzalo, Baeza-Yates Ricardo, Sutinen Erkki, Tarhio, Jorma. 2001. Indexing Methods for Approximate String Matching. *IEEE Data Engineering Bulletin.* 24 (4): 19–27.