Lexical Normalisation of Twitter Data

Siddharth Malhotra
The University of Melbourne
smalhotra1@student.unimelb.edu.au

ABSTRACT

In this report, we discuss two major spelling correction methodologies on the problem of tweet normalisation specifically on Twitter data. Firstly, discuss the combination of Levenshtein Distance, Peter Norwig Algorithm and n-gram context matching, namely LRPN approach. The second method involves first preprocessing the text, tokenisation is performed through NLTK, a lexicon and confusion set is built, scoring and ranking of possible interpretation of Out-of-vocabulary words (OOVs),decoding the confusion lattice thus formed using SRI-LM toolkit, namely NLTK-SRI technique. Further, we perform a critical analysis of these approaches by calculating the *precision* and *recall* values.

KEYWORDS

Lexical Normalisation; Phonetic Matching; Levenshtein distance; Peter Norvig's Algorithm; N-Gram; NLTK; SRILM

1 INTRODUCTION

Microblogging constitutes the modern method of using social networks. As there is a high input cost associated with using the mobile phone keypads, users tend to shorten their words. Twitter, for instance allows 140 or fewer characters, and generally contains hash tags and @ symbols. The proceeding sections describe in detail the various techniques that are applied to identity: Known or in-vocabulary words; punctuation and special symbols (both general and Twitter specific); and candidates for normalisation. We then apply various normalisation techniques to correct out of vocabulary *OOV* tokens.

2 TEXT NORMALISATION TECHNIQUES

2.1 Text Pruning

Before we begin analysing our tokens, we would want to categorise and extract the tokens which we would want to consider for Normalisation. Our focus is to segregate out tokens into various categories, such as In Vocabulary, Non Candidates and Out of Vocabulary text. We use $labelled - tweets.txt$ for extracting the labelled tokens using *listing.py*. This gives us tokens in a line-by-line format.

Ahmed [1] tokenises and classifies strings into three main categories In-Vocabulary, Out of Vocabulary (*OOV*) and Non candidate tokens (*NO*). For the purpose of out study, we run a Levenshtein difference analysis using *distance.py* and place tokens with difference 0 into in-vocabulary terms. The Levenshtein distance is calculated as per the number of substitutions, deletions or additions to be made to the input string to convert the string L to D.

Corpus	Features
labelled-tweets.txt	53KB
labelled-tokens.txt	127KB
tokens.txt	49 KB (8841 words)

Table 1: Corpura used for Normalisation

In order to identify non-candidates (NO), we run a regex string check, on all the terms to classify strings beginning with punctuation symbols, character @ or # or http websites using *Outliers.py*. Thus, candidates without the presence of in vocabulary terms and special characters are marked as candidates for lexical normalisation and are termed Out-of-Vocabulary (OOV).

Usually these are terms that have been misspelt of have been abbreviated in order to fit Twitters 140 character limit. Our *results.txt* document now contains all the IN and NO words up till this point. For the remaining candidates we perform the normalisation techniques. **Table 1** represents the corpura of the words we have used for normalisation.

2.2 *LRPN* Technique

Normalisation Process:

Calculating Edit Distance: First, the terms with edit distance of 2 are identified and results are stored in an array. These contain words based textual similarity. For this, we use the python-Levenshtein 0.12.0 package. The phrase edit distance is used to refer specifically to Levenshtein distance.The dictionary terms (dict.txt) is searched for matches based on Levenshtein Distance. We perform this step through *distance.py* and segregate our input token strings *dictionarywords* and *nondictionary* words.

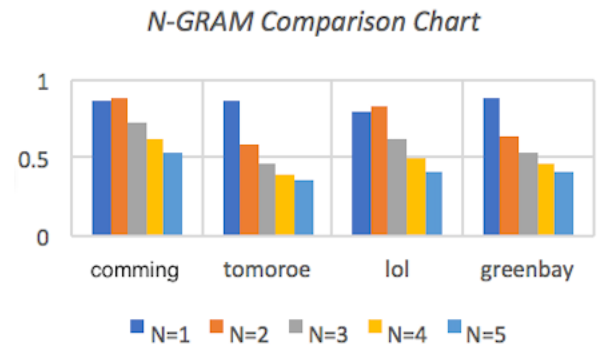
Note: Although it is possible to retrace the dictionary words corresponding to the lowest value of the edit distance, we take a conscious decision, based on evaluation of several edit distance measures (global and local) on not doing so as the accuracy of these words extracted will not be optimal in most cases. Subsequently, we combine several techniques to approach the same.

Peter Norvig's Algorithm: Norvig [5] introduced an algorithm for the approximate matches. This returns one match deemed closest to the query by the algorithm. Peter Norwig's algorithm considers corrections that require two simple edits. The following steps are performed: Selection Mechanism, Candidate Model, Language Model and Error Model. We use *PeterNorwig.py* for evaluating the Peter Norwig correction. We combine Peter Norvig with N-gram context matching for increasing the accuracy of normalisation.

N-gram matching: $n - gram$ is a contiguous sequence of n items is analysed from a given sequence of text or speech. An n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". Larger sizes are sometimes referred to by the value of n in modern language, e.g., "four-gram", "five-gram", and so on.

For each of the OOV's we run different $n - gram$ values from $N = 1$ to $N = 5$ values using *generate_resultsimilarity.py*. We do this as n-gram may not give us the appropriate string match for lower values of ngram. Through this we analyse the best ngram decimal values we have obtained for each of the OOV tokens and use *gen_nresult.py*, to pull out the highest ngram value corresponding to each word. For instance, say the OOV word 'commig', the value we get for $N = 1$ is 0.875000000 and the value for $N = 2$ is 0.888888889.

We further maintain a list of all the best N-values for each word and maintain a count using *gennhighest.py*. From this we make a deduction to find what is the most occurring value of N, giving us the highest results and we further use our chosen N-value to backtrack the possible dictionary word. We use *gen_correspondword.py* for getting the possible dictionary word.



2.3 *NLTK-SRI* Technique

Gouws et al. [2] first conduct a context free analysis (at the linguistic level) of all the non-English words. Thus terms such as '2day', 'today' are consider distinct words. Next, they perform normalisation on the noisy text to recover the surface form of the message and record the contextual dimensions, such as, client and location. A text cleaner is built to handle morphophonemic variations. The cleanser marks @username with *USR* and hashtags that are preceded before the end of messages are removed. Hashtags within the middle of the word are removed.

Next, NLTK tokeniser is used to split all tokens that include punctuation symbols. $(\backslash w^*)\ ([\ P])\ (\backslash w^*) \longrightarrow [\backslash 1, \backslash 2, \backslash 3]$ can be used for the same. A confusion network *CN* is built by comparing token to the words in lexicon *L*. All the tokens be it *IN* or *OOV* or punctuation is added tot the CN with probability 1.0. Under the data directory *common_abbrs.csv* maintain a for optimising referencing. A transliteration look up table is built to convert text such as 't0day' $\longrightarrow$ ['t0day','today'],refer **Table 2**. Ranking is performed through a heuristic function (sim()) for each candidate word.

Confusion Set that is produced this way is joined with the previous set. Finally, lattice is decodes with the use of probabilistic finite state grammar format. At this point, SRI-LM's $lattice - tool$ is used to find the best path through the lattice and training of the set is done as follows. For selecting clean tweets, the following measure was used. Lastly, training of 30M words is done on this set.

$$\frac{Number\ of\ OOV}{Number\ of\ IV + 1} < P; where P = 0.5. \tag{1}$$

Character	Transliteration candidates
1	'1','1','one'
2	'2','to','too','two'
3	'3','e','three'
4	'4','a','for','four'
5	'5','s','five'
6	'6','b','six'
7	'7','t','seven'
8	'8','ate','eight'
9	'9','g','nine'
0	'0','o','zero'
'@'	'@','at'
"&"	'&','and'

Table 2: Transiliteration lookup table

3 COMPARISON AND REVIEW

Precision is computed as in equation (2).

$$Precision = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{2}$$

Recall is computed as in equation (3).

$$Recall = \frac{Number\ of\ correct\ words}{Total\ number\ of\ words} \tag{3}$$

LRPN Technique. Major disadvantage of LRPN strategy is execution cost is quite high. The combination of these strategies provides a much better result than when these are run in a standalone manner.

Using the edit distance, we are able to find out that there are 6652 dictionary words of the 8841 tokens. We are also able to eliminate the non candidate words which are 533. Using Peter Norwig's algorithm we were able to increase the accuracy. Through n-gram technique we found that n=1 and n=2 were of our purpose, thus we made 3312 predictions. In total, 351 words were correctly predicted these techniques. Note: We include the dictionary and non candidate words in the total prediction count. Thus, the total number of predictions adds up to 10497 and correct predictions and the number of correct predictions are 7536.

$$Precision = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{7536}{10497} = 71.8 \tag{4}$$

$$Recall = \frac{Number\ of\ correct\ words}{Total\ number\ of\ words} = \frac{7536}{8841} = 85.2 \tag{5}$$

NLTK-SRI Technique. Substantially faster than a LRPN Technique for resolving non candidate words. Also gives a substantial increase in the accuracy. Certain simple rules can we used to clean up the OOV words, but noisy text cleanser is useful to correct long tail of OOV's. Note: For each word, there are approximately 10 predictions thus making the file and computation size. We have scaled down this analysis to two-thirds of the token file.

$$Precision = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{5432}{67390} = 8.06 \tag{6}$$

Although the number of correct predictions are high, the number of predictions is large is.number.

$$Recall = \frac{Number\ of\ correct\ words}{Total\ number\ of\ words} = \frac{5432}{6739} = 80.6 \tag{7}$$

The accuracy of NLTK-SRI technique is always above 90%. An Example (from result.txt in TextCleanser-master), if the input is 'wooow of course , i shoulda known that was comingg', the output is 'wow of course, i should known that was coming'.

REFERENCES

[1] B. Ahmed. Lexical normalisation of twitter data. *Science and Information Conference (SAI)*, 2015. URL https://arxiv.org/pdf/1409.4614v4.pdf.

[2] S. Gouws, D. Metzler, C. Cai, and E. Hovy. Contextual bearing on linguistic variation in social media. *Proceedings of the Workshop on Languages in Social Media*, pages 20–29, 2011. URL http://dl.acm.org/citation.cfm?id=2021109.2021113.

[3] B. Han and T. Baldwin. Lexical normalisation of short text messages: Makn sens a# twitter. *49th Annual Meeting of the Association for Computational Linguistics*, pages 368–378, 2011. URL http://www.aclweb.org/old_anthology/P/P11/P11-1038.pdf.

[4] K. Max and J. Kalita. Syntactic normalization of twitter messages. *International conference on natural language processing*, 2010. URL http://cs.uccs.edu/~jkalita/work/reu/REUFinalPapers2010/Kaufmann.pdf.

[5] P. Norvig. How to write a spelling corrector. URL http://norvig.com/spellcorrect.html.