Bekli: A Simple Approach to Twitter Text Normalization

Russell Beckley
Oregon Health and Sciences University
Portland, Oregon
beckleyr@ohsu.edu

Abstract

Every day, Twitter users generate vast quantities of potentially useful information in the form of written language. Due to Twitter's frequently informal tone, text normalization can be a crucial element for exploiting that information. This paper outlines our approach to text normalization used in the WNUT shared task. We show that a very simple solution, powered by a modestly sized, partially-curated wordlist—combined with a modest re-ranking scheme—can deliver respectable results.

1 Introduction

Twitter is an immense, living collection of written language from all over the world. Every day, Twitter publishes a staggering 500 million tweets[1]. The content of Twitter is virtually unlimited, and has proven useful for much research, including epidemiology: Chew and Eysenbach (2010); and sentiment analysis: Barbosa and Feng (2010), Bakliwal et al. (2013), Rosenthal et al. (2015), Li et al. (2014).

It would take many readers to keep up with Twitter's output, but, fortunately, we have natural language processing (NLP) methods that can automatically filter, condense, or extract information from text. However, NLP approaches are typically trained on formal edited text, and struggle with the informal, unedited text of Twitter. But there is a well-known way to mitigate this problem: *text normalization*, i.e. replacing non-standard tokens with their standard equivalents, yielding text that will be more agreeable to NLP.

[1]https://blog.twitter.com/2013/new-tweets-per-second-record-and-how

One flavor of non-standard writing—what I have previously focused on—is what I call "vernacular orthography" (VO). VO is spelling that indicates itentional non-standard pronunciation, such as when the string "dat" stands in for "that". While numerous papers offer solutions for text normalization (e.g. Han and Baldwin (2011), Yang and Eisenstein (2013), Zhang et al. (2013), Sproat et al. (2001), Li and Liu (2014)), and a few build models based on phonemic similarity (e.g. Kobus et al. (2008), Choudhury et al. (2007)), none to our knowledge have addressed VO in particular. This paper, too, addresses the general normalization problem, but uses lessons learned attempting to normalize VO.

2 System Architecture

The architecture of this system is very simple, consisting of three main parts: (1) a substitution list, (2) a couple of rule based components, and (3) a sentence level re-ranker. This provides for a fast per-token performance.

2.1 Substitution List

Most of the work is done by a semi-supervised substitution list consisting of ordered pairs. The first member of each pair is a string representing a non-standard word. The second member of each pair is a list of strings representing candidate replacements for the word. For example, one pair is ("n", ("and, in")). There are just over 45,000 pairs, where only the first 2000 are hand-curated.

To create the list, we use a collection of tweets (see "Resources Employed" section) and a derived dictionary, described presently. The dictionary has the 18,000 standard words most frequent in the tweet collection. For the construction of the dictionary, a

Proceedings of the ACL 2015 Workshop on Noisy User-generated Text, pages 82–86,
Beijing, China, July 31, 2015.

word is considered to be standard if it has at least four characters, and is found in the CMU pronouncing dictionary, or, if it has fewer than three characters, it is found in the Norvig dictionary[2] and is sufficiently frequent (where sufficient frequency depends on the word length).

Now we want to find the most frequent OOVs that need to be normalized. We tokenize the twitter set and filter out all tokens that appear in our dictionary. We also filter out all tokens that do not match the format of normalizable tokens as specified by the shared task e.g. tokens that have non-alphanumeric characters other than an apostrophe('). Lastly, we filter out those tokens that could be normalized by our rule-based components (described in next subsection).

We count the occurrences of each OOV-candidate token type, sort by the count, and return the resulting list. This puts the most useful candidates first and provides for efficient use of annotation time. Suitable replacements require human judgement and occasional reference to outside sources. The outside sources were (1) Urban Dictionary, which is very useful for slang and acronyms (2) Twitter, which tells you how a word is most often used on Twitter, and (3) the training set provided for the shared task, which tells you how to normalize in ambiguous cases (e.g. "laughing out loud" v. "laugh out loud").[2]

In addition to these hand-curated entries, we added the Lexical normalization dictionaries, UniMelb and UTDallas, provided for the shared task. From these lists we took all entries not already in the hand-curated list.[2]

With this initial list in place, we ran it on the training set and analyzed the errors, looking specifically at false positives and false negatives. We sorted the tokens that caused these errors according to a formula that estimated what change would occur to the f-score if the token was to be removed or added to the list. If the token represented a false negative, we would estimate the change to f-score we would get by adding it to the list, assuming that it's substitution would always be the word most often associated with it in the training set. If it was false positive, we would estimate the change to f-score we would get by deleting it from the list.

This analysis revealed some weak spots in the list. First, there were a number of false positives caused by differing beliefs regarding what counts as non-standard. For example, there are several contractions (e.g. "gonna", "gotta", "wanna", and "ain't") that are not usually considered standard (rarely seen in The Wall Street Journal), and have straight forward normalizations, that are nonetheless considered to be in-vocabulary in the task. These words were removed from the substitution list, and added to a new list—a "do-not-normalize" list.

Furthermore, there were of course a number of false negatives. Many of these come from tokens that are in the dictionary, but that are often used in a non-standard way in informal speech. For example, "wit" is in standard dictionaries, referring to an intellectual feature; however it often appears in Twitter as a non-standard variation of "with", as in "you wit me hea?" Likewise on Twitter, "cause" almost always means "because". Such tokens were added to the substitution list.

It might be supposed that using the training set in this way could lead to severe over-fitting. To avoid this, we didn't make any adjustments for tokens appearing less than three times as a false positive, true positive, or false negative. The results show that any over-fitting was not severe, since the test f-score was just one point less than the training f-score.

2.2 Rule-based components

We also experimented with several rule-based components, two of which— because they applied in the greatest number of cases in the training set—were used in the final system. These components were the "ing" rule and the "coool" rule.

The "ing" rule looks for cases in which the verbal suffix "-ing" is altered to an "-in", "-en", or "-n", such as when "busting" becomes "bustin". If the test token is in the dictionary, the component generates no candidates. If the token is not in the dictionary, the component checks if the word ends with "-in", "-en", or "-n" proceeded by certain consonants, and if so, checks for the likelihood of additional syllables. If those conditions hold, it replaces the identified ending with "ing", and if the result is in the dictionary, it becomes a candidate.

The "coool" rule attempts to normalize text that,

[2]See "Resources Employed" section.

for emphasis, repeats characters, as in "Thaatt iss reallyyyyy neeeeat!" To generate candidates, the "coool" rule finds every run of more than two repeated characters and reduces the length of the run to two. For every one of these runs, we assume that the original had either one or two of that character in that place. We consider every string that can be created by reducing a subset of the two-character runs to one character each, and return only those strings that occur in the dictionary. For example, if the original token is "thaatt", we consider "thaatt", "thaat", "thatt", and "that", but return only "that", being in the dictionary.

When the system is run with only the "ing" and "coool" rules, plus the sentence level re-ranker, we get a precision of .81 and a recall of .09. However, when combined with the rest of the system, it's contribution is insignificant. It seems that the most frequent instances of these rules are already in the substitution list, so the rules do not generate enough true positives to offset their generated false positives.

Along with "ing" and "coool", we tried a number of similar rule-based components. For example, we looked for cases where "th" is replaced by either "d", "f", or "t". Another example is the "double consonant" rule, based on the idea that when a word ends with two consonants, and both are voiced or both are unvoiced, the second consonant is dropped. For example "wrist" becomes "wris". These are widespread phenomena, but not widespread enough on Twitter for the true positives to outweigh the false positives. A more sensitive rule or a better sentence-level re-ranker would be needed to make these components beneficial.

2.3 Sentence Level Reranker

The third major component of the system is the sentence level-re-ranker. This is, in short, a bigram Viterbi algorithm.

Bigrams were collected from our set of ten million tweets. Bigrams with any out-of-vocabulary tokens were ignored. From this set, for each bigram (t_1, t_2), we computed $prob(t_2|t_1)$ with Laplacian smoothing. These became the transition probabilities in our Viterbi problem.

At test time, we generated candidates for each token. If the substitution list had an entry for an original token, all suggested substitutions in that entry became candidates. For each of these candidates, c, we initialize a weight, w_c, where $w_c = 2 \times (rank(c) + 1)$, and $rank(c)$ refers to c's position in the list for the current token's entry in the substitution list. If the "ing" rule or the "coool" rule generated answers, those would also be candidates. For the "cool" rule, the weight was the number of deletions required to get the candidate from the original token. For the "ing" rule, the weight was, somewhat arbitrarily, 1. Finally, the original token is a candidate with a weight of 0. These weights were the emission weights for Viterbi, and were treated as $-\log(prob(c))$. For each original token in the test set, we generated on average .04 other candidates.

The system then constructed a lattice from the tweet and all of its normalization candidates. With Viterbi dynamic programming it found the maximum probability path through the lattice. Words in the maximal path were taken to be the correct word for the corresponding token.

At the time of the shared task, I compared this approach to a simpler approach, in which, for any original token, the system ignored the context and selected the normalization with the greatest emission score (emission score as defined above). At first, the Viterbi method added 10 percentage points $f1$ over this method. However, after the the shared task was finished, I discovered that the greatest-emission method had an error. Having fixed that error, and rerunning the system on the training data, I discovered that this greatest-emission rule, on the training data, gives better results than the Viterbi system used for the shared task: for $f1$ scores, the Viterbi approach gets a .768 while the greatest-emission score is .816. Note that, for the Viterbi approach, the test score, .757, is not much less than the training score. In summary, (1) the Viterbi approach, as implemented, is probably not the best, and (2) the overall normalization approach I describe in this paper is probably better than the shared task results suggest.

3 Resources Employed

The computations for training and testing were done on a MacBook. Required computational resources were minimal. Data resources are as follows:

A. **CMU pronouncing dictionary:** "an open-source machine-readable pronunciation dictio-

Team Name	precision	recall	f1
IHS_RD	0.8469	0.8083	0.8272
USZEGED	0.8606	0.7564	0.8052
bekli	0.7732	0.7416	0.7571
gigo	0.7593	0.6963	0.7264
lysgroup	0.4592	0.6296	0.531

Table 1: Team Results for the unconstrained task.

nary for North American English that contains over 134,000 words and their pronunciations"[3]

B. **count_1w.txt from Peter Norvig**: "The 1/3 million most frequent words, all lowercase, with counts." [4]

C. **10 million tweets** collected by Steven Bedrick, of Oregon Health and Sciences University.

D. **WNUT Lexical normalisation dictionaries**[5]

   a. UniMelb

   b. UTDallas

E. **WNUT training set**.[6]

F. **Urban Dictionary:** a highly inclusive user-generated online dictionary.[7]

4 Results

Table 1 shows the results for the unconstrained task. The project described in the present work is named "bekli". The training set consisted of 2024 tweets with a total of 3928 tokens that needed to be normalized. The test set consisted of 1967 tweets with a total of 2738 tokens that needed to be normalized Baldwin et al. (2015).

5 Conclusion

The results show that a simple strategy with minimal computational resources can go along way. For example, the space required for the list and rule-based components is negligible. The only element that requires some heavy lifting is the sentence-level reranker with its long list of bigrams.

However, as I described above, selecting the token with the greatest-emission probability actually works better than my bigram approach, and requires far less computation. This leaves the question: what results could be achieved using a better re-ranker, one that successfully exploits context? Such a reranker would, among other benefits, make it feasible to use rules or substitutions that are not, without using context, capable of high precision.

Another question remaining is how much better can we do by expanding the curated segment of the list—if we, for example, double the size? This would would still allow a program to have a very small computational imprint, while doing nearly the work of a more sophisticated system.

[3]http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[4]http://norvig.com/ngrams/count_1w.txt

[5]http://noisy-text.github.io/norm-shared-task.html

[6]http://noisy-text.github.io/norm-shared-task.html

[7]http://www.urbandictionary.com/

References

Akshat Bakliwal, Jennifer Foster, Jennifer van der Puil, Ron O'Brien, Lamia Tounsi, and Mark Hughes. 2013. Sentiment analysis of political tweets: Towards an accurate classifier. In *Proceedings of the Workshop on Language Analysis in Social Media*, pages 49–58, Atlanta, Georgia, June. Association for Computational Linguistics.

Timothy Baldwin, Marie Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text (WNUT 2015)*, Beijing, China.

Luciano Barbosa and Junlan Feng. 2010. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 36–44.

Cynthia Chew and Gunther Eysenbach. 2010. Pandemics in the age of twitter: content analysis of tweets during the 2009 h1n1 outbreak. *PloS one*, 5(11):e14118.

Monojit Choudhury, Rahul Saraf, Vijit Jain, Animesh Mukherjee, Sudeshna Sarkar, and Anupam Basu.

2007. Investigation and modeling of the structure of texting language. *Int. J. Doc. Anal. Recognit.*, 10(3):157–174, December.

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 368–378, Stroudsburg, PA, USA. Association for Computational Linguistics.

Catherine Kobus, François Yvon, and Géraldine Damnati. 2008. Normalizing sms: Are two metaphors better than one? In *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 441–448, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chen Li and Yang Liu. 2014. Improving text normalization via unsupervised model and discriminative reranking. In *Proceedings of the ACL 2014 Student Research Workshop*, pages 86–93, Baltimore, Maryland, USA, June. Association for Computational Linguistics.

Jiwei Li, Alan Ritter, and Eduard H. Hovy. 2014. Weakly supervised user profile extraction from twitter. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 165–174.

Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M Mohammad, Alan Ritter, and Veselin Stoyanov. 2015. Semeval-2015 task 10: Sentiment analysis in twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval.*

Richard Sproat, Alan W Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of non-standard words. *Computer Speech & Language*, 15(3):287–333.

Yi Yang and Jacob Eisenstein. 2013. A log-linear model for unsupervised text normalization. In *EMNLP*, pages 61–72.

Congle Zhang, Tyler Baldwin, Howard Ho, Benny Kimelfeld, and Yunyao Li. 2013. Adaptive parser-centric text normalization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (1)*, pages 1159–1168.