Normalization of Tweets

Soumya Ranjan Patra

Technical University of Eindhoven

s.r.patra@student.tue.nl

Abstract

Informal texts such as *tweets* present a plethora of information which can be used for text mining and data analytics purposes. But, it is very difficult to extract useful information out of them because of their noisy nature. Standard NLP[1] applications perform miserably on tweets. Therefore, normalizing tweets before performing NLP tasks on them is always a good idea. This domain has not attracted much attention from researchers. In this research, normalization of tweets is performed by first detecting OOV[2] words in a tweet and then generating a candidate set of all possible well formed transformations. Out of the candidate set, the best possible word is selected based on a rule based mechanism and the OOV word is replaced with the selected word.

1. Introduction

Twitter is an on-line microblogging service which allows users to post tweets with at most 140 characters. Because of this constraint, there is insufficient context information in tweets. People use various kinds of informal notations which makes information extraction extremely difficult. Table 1 presents some of the common problems in tweets. While writing tweets, people generally ignore the casing. In the first example, it can be seen that the name should have been *George Clooney*, but instead the capitalization is missing. Unlike the first problem, in some cases people use unnecessary capitalizations that hide the entities inside. For instance, in example 2, it is very difficult to identify *JENNIFER* as an entity. Abbreviated texts such as *How I Met Your Mother* to *HIMYM* in example 3 is another problem that needs to be dealt with. People also tend to shorten the texts as can be seen in example 4 where *iPhone* is shortened to *ifun*. Spelling mistakes, which, in most cases are intentional, is another problem that needs to be taken into account. Example 5 shows a sample spelling mistake which misspells *Peterson* to *ptersan*. Grammatical mistakes are also extremely common in tweets as people do not care to apply standard linguistic rules while writing.

[1]Natural Language Processing e.g. Named Entity Recognition, Sentiment Analysis.
[2]Out of Vocabulary

1	missing Capitalization	*george #Clooney* did a great job in *gravity.*
2	unreliable Capitalization	#JLO WAITIN TO SEE JENNIFER'S NEW CAR
3	Abbreviation	*HIMYM* starts in 10 mins
4	Shortened texts	@rk89 pls dnt 4get to brng ma ifun
5	Spelling mistakes	sry bt you r wrng this time mr *ptersan*

Table 1: sample tweets showing the problems in performing Information Extraction tasks

All these issues make it challenging for standard NLP tasks to be applied on tweets. For example, Stanford Named Entity tagger[3] perform very poorly on tweets as compared to formal texts such as news articles. These problems have motivated the need of research in the field of tweet normalization. Normalized tweets can be extremely useful in NLP applications which is not possible with thier noisy counterparts. Next section describes some of the work done in the area.

2. Related Work

There has been a little research done in the field of Tweet Normalization. Some of them are presented in this section. In their research, Aw et al.[1] built a phrase-based statistical model to normalize SMS texts. The model was trained with SMS texts and their corresponding manually corrected standard English forms. They obtained good results. For example, their model was able to differentiate when to replace *2* with *to* and when with *too*. But, they have not implemented any component for spelling correction. the model only performs well with transforming abbreviations.

Kaufmann & Kalita[2] start by hand annotating 1150 randomly sampled tweets. They preprocess these tweets by first removing unnecessary repetitions of words. To correct the spelling errors, they only use a table containing common SMS errors and their equivalents. While, this approach would work well with SMSs, spelling errors in tweets are far more worse compared to SMSs. Therefore, this approach does not seem sophisticated enough to handle such errors. After replacing the common errors, they remove the usernames and hash tags those do not contain any meaningful information. Finally, they use statistical machine translation with Moses[4] to normalize the tweets.

Han et al.[3] implement various normalization techniques to transform OOV words to their standard form. Given an OOV word, they generate a confusion set of possible transformations by removing repetitions of letters (*cooool* to *cool*), character edit distances (*earthquaki* to *earthquake*) and phonemic edit distances (*earthquick* to *earthquake*). Then they

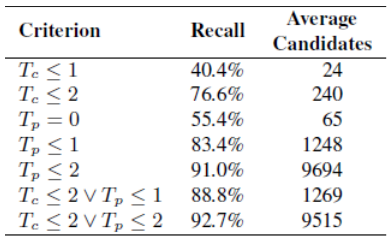
[3]`http://nlp.stanford.edu:8080/ner/process`
[4]`http://www.statmt.org/moses/?n=FactoredTraining.HomePage`

use language model and dependency-based frequency features to select the most likely candidate out of the confusion set. This is a good approach to normalize informal texts such as tweets. Because, it tries to solve different aspects of the problem such as identifying and replacing slangs, correcting spelling mistakes by considering both spelling and pronunciations. Therefore, this research takes their approach as a reference and tries to replicate and improve some of their results.

3. Research Problem

To make Twitter data suitable for text processing, normalization is an essential step. Han & Baldwin, in their paper, propose Lexical Normalization approach on tweets. As part of a research problem, it would be interesting to see if some of their results can be improved.

1. They start by collecting twitter stream data and filtering out English tweets with the help of a Language Model.
2. After collecting this data, they perform a spell check with the help of GNU aspell dictionary to find out the OOV words. They do this check for all the tokens. This could be optimized by first filtering out all the tokens those are already known entities such as names, organizations. In this way, not only can we avoid the chance of these words being considered as OOVs, but also reduce the number of checks.
3. To detect slangs, they use a list from internet containing a dictionary of only 5021 slangs. Clearly, this number is not sufficient to cover the slangs used in Twitter. For this research, a bigger dictionary is used that contains slangs from various domains.
4. Then they remove repetitions of redundant letters(*cooooool* to *cool*).
5. After performing these steps, they generate a possible set of IV[5] words from these OOV words. They do it by grouping the words within a certain character edit distance (Threshold T_c) and a phonetic edit distance (Threshold T_p). They claim that $T_c \leq 2 \vee T_p \leq 1$ is the best choice for candidate set generation. But, as shown in Fig.1, they still get 1269 candidates, which is quite large. This research experiments with different types of edit distances and with different T_c, T_p values to try reducing the average number of candidates while maintaining a high recall.



Criterion	Recall	Average Candidates
$T_c \leq 1$	40.4%	24
$T_c \leq 2$	76.6%	240
$T_p = 0$	55.4%	65
$T_p \leq 1$	83.4%	1248
$T_p \leq 2$	91.0%	9694
$T_c \leq 2 \vee T_p \leq 1$	88.8%	1269
$T_c \leq 2 \vee T_p \leq 2$	92.7%	9515

Figure 1: Different threshold values

[5]In Vocabulary

6. Based on a trigram language model (SRILM[6]), they rank all these candidates. Gathering clean (i.e without any OOV words) twitter data to train such a model poses a different kind of challenge. This research tries to address the problem by using two different N-gram models.
7. They compare their candidate selection effectiveness with a number of other models. While it would be an interesting to verify these results, it would not have been feasible considering the time constraints of the seminar.

4. Normalization

This section describes the step by step approach taken in this research for the processing of tweets.

4.1. Data Collection

The first task was to collect tweets for the normalization process. Twitter provides an API[7] to make this task easier. With the help of a python script given in Appendix A and the streaming API, tweets were collected on the track *America* and were stored in a *json* file.

4.2. Tokenization

After collecting the tweets, the next step was to tokenize them. This is a necessary step because it breaks the tweet into small meaningful pieces which will make the analysis a lot easier. It was performed with the help of *GATE TwitIE pipeline*[8]. GATE provides an open source tool with different plugins to perform text processing. *TwitIE* is one of the plugins specially designed to process tweets and is trained on Twitter-specific data. This is the reason why it was used for tokenization. It contains various modules each capable of performing different tasks. Figure 2 shows the pipeline with different modules.

The *Language Identification* module detected the language of the tweets. Only English tweets were considered for the normalization process. In the next step, TwitIE tokenizer module, which is a twitter-specific adaptation of the general-purpose tokenizers, generated a list of tokens. These tokens were further processed by modules such as *Gazetteer Lookup* and *Named Entity Recogniser* to detect if they are already known entities such as locations or persons. If that is the case, the tokens were marked as known entities. This approach separated already known entities, thereby significantly reducing the number of tokens that had to be tested for OOV words. Figure 3 presents a sample tweet after it is processed in the pipeline. The circled tokens are the known entities which needed no further checking for OOV words. Tokens containing hashtags and usernames were also not checked for OOV

[6]`http://www.statmt.org/moses/?n=FactoredTraining.HomePage`
[7]`https://dev.twitter.com`
[8]`http://gate.ac.uk/wiki/twitie.html`

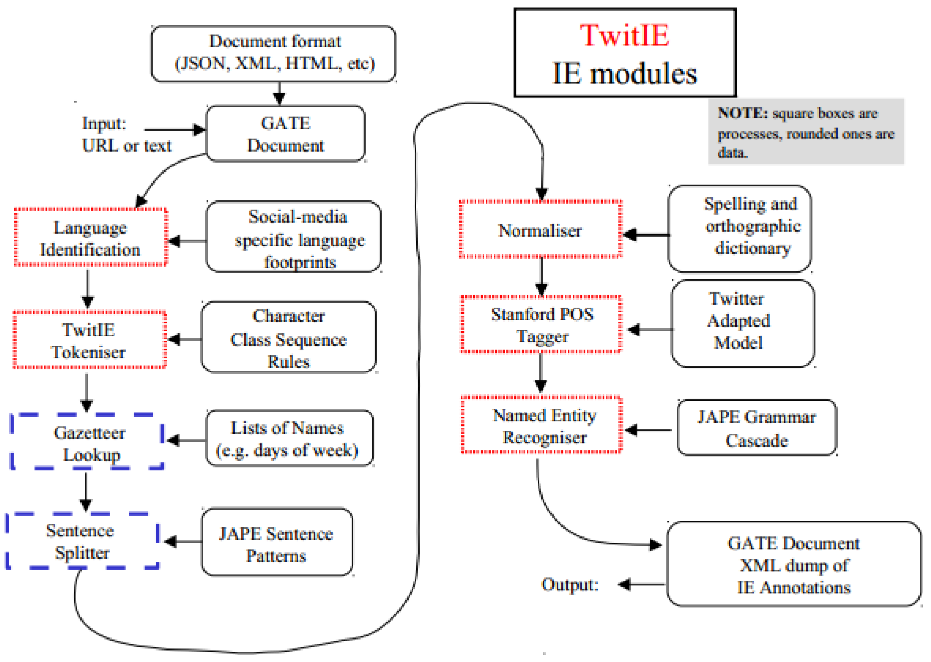
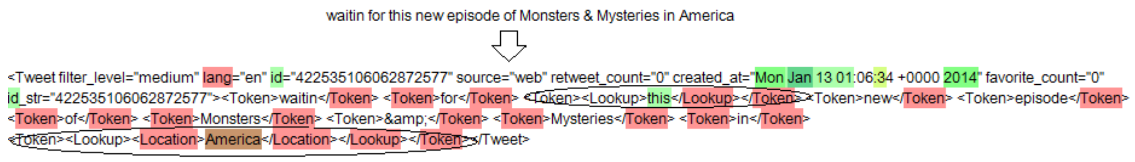

TwitIE
IE modules
NOTE: square boxes are processes, rounded ones are data.
Document format (JSON, XML, HTML, etc)
Input: URL or text
GATE Document
Language Identification
Social-media specific language footprints
TwitIE Tokeniser
Character Class Sequence Rules
Gazetteer Lookup
Lists of Names (e.g. days of week)
Sentence Splitter
JAPE Sentence Patterns
Normaliser
Spelling and orthographic dictionary
Stanford POS Tagger
Twitter Adapted Model
Named Entity Recogniser
JAPE Grammar Cascade
GATE Document XML dump of IE Annotations
Output:

Figure 2: Gate TwitIE pipeline for processing tweets[4]

as they may contain meaningful information and there is a high probability that the model will consider them as spelling mistakes.



waitin for this new episode of Monsters & Mysteries in America

```
<Tweet filter_level="medium" lang="en" id="422535106062872577" source="web" retweet_count="0" created_at="Mon Jan 13 01:06:34 +0000 2014" favorite_count="0"
id_str="422535106062872577"><Token>waitin</Token> <Token>for</Token> <Token><Lookup>this</Lookup></Token> <Token>new</Token> <Token>episode</Token>
<Token>of</Token> <Token>Monsters</Token> <Token>&amp;</Token> <Token>Mysteries</Token> <Token>in</Token>
<Token><Lookup><Location>America</Location></Lookup></Token></Tweet>
```

Figure 3: Before and after processing in TweetIE pipeline

4.3. Check for OOV Words

All the tokens those were not known entities, were tested to check whether they are Out of Vocabulary words. This was done with the help of Enchant spell checking library in python. Not only does the checker check for OOV words, but also suggests possible corrections if the checking fails. These suggestions form the first set of suggested candidates.

4.4. Repetition Removal

As suggested by Kaufmann and Kalita, before checking for slangs, all the words are checked for unnecessary repetitions. More than three continuous occurrences of letters were reduced to three. This step was essential because people often tend to use repetitions in their words to overemphasize the words.

4.5. Slang Detection

Each OOV word was first checked for slangs. A slang search engine from internet[9] was used for this purpose. With the help of regular expressions, web responses were parsed to check if the word was a slang. If the check succeeded, then the word was replaced and no further processing was performed on the word. Otherwise, the word was processed further for spelling corrections.

4.6. Confusion Set Generation

For all the OOV words those were not slangs, a confusion set of possible suggestions was generated by taking different character and phonemic edit distances. A number of experiments were done with different types of edit distances and the results are presented in the next section.

4.7. Ranking the Candidates

An N-gram model trained on clean twitter data would have been perfect for the purpose of ranking the candidates and selecting the best one. But, since twitter data is never "clean" (i.e. there are always OOV words) and it is very difficult to find a large dataset with all IV words, Microsoft's web N-Gram model[10] was used for this step. This model gives users opportunity to access petabytes of Bing's indexed data. Hence, even though this data, might to some extent, not fit Twitter perfectly well, it was chosen because of the vastness of its content. Each suggested candidate in the confusion set was then assigned a probability measure based on the 5-grams of the model. Output was a list of candidates sorted according to the assigned probabilities. This approach worked quite well with the collected tweets. To check, if adding another N-gram model can improve the ranking performance, NLTK[11] bigram model trained with Brown corpus was used. But, improvements were not significant enough to be reported.

4.8. Selecting the Best

After ranking all the candidates, the best one was selected based on a set of rules.

1. If a candidate was suggested by all the three suggesters (*i.e. dictionary, character edit distance and phonemic edit distance*), then it was considered the best. If there were more of such candidates, the one with the best ranking among them was selected as the best.
2. If a candidate was suggested by 2 out of the 3 suggesters, that word's ranking was improved by some points. Finally the word with the best ranking was termed as the best one.
3. If there were no such candidates which were common in any two suggesters, the best one was selected solely based on the ranking.

[9]`http://www.allslang.com`

[10]`http://web-ngram.research.microsoft.com`

[11]`http://nltk.org/`

5. Experiments

Through the Twitter stream, tweets were collected at random for the experiments and were stored in a json file (*America2.json* in the dropbox folder[12]). These tweets were then processed through the TwitIE pipeline and the output can be found in the *test.xml* file. Out of all, 1674 English tweets were considered for experiments. All the OOV words in these tweets were manually checked for their correct forms and a list was generated. This list acted as a correctness reference for measuring the performances of the experiments. A python script *Normalization.py* was written to perform the normalization process. Levenshtein distance[13] was chosen as the character edit distance measure. For Phonemic edit distance, Metaphone[14] and Soundex[15] measures were used. Different experiments were performed by taking different combinations of them and with different edit distances. The output files for all the experiments can be found in the folder *Results*. Each output folder contains 3 files. Original tweets along with their normalized forms are stored in the file *compare.txt*. For each OOV word, The file *oov.txt* contains an entry of the word, all the suggestions for that word, total number of suggestions and the best choice. The last line of this file presents the performance of the corresponding experiment. It also shows the OOV words for which the experiment was unsuccessful in finding the correct forms. All the slangs and abbreviations can be found in *slang.txt*.

6. Results & Evaluation

In their research, Han et al. experimented with 449 tweets where they found 2141 OOV words i.e. 4.77 OOVs per tweet on average. In this research, only 918 OOV words were found in 1674 tweets which gives 0.55 OOVs on average per tweet. This proves that recognizing and separating pre-known entities beforehand can reduce the number of OOV words to be tested. Even though different datasets were used for both the researches, these results could easily be replicated in their dataset.

Criterion	**Recall**	**Average Candidates**
$T_c \leq 1$	49.8%	5
$T_c \leq 2$	51.31%	22
$T_p = 0$	49.67%	13
$T_c \leq 2 \vee T_p = 0$	54.48%	29

Table 2: Results for experiments with different T_c and T_p values (levenshtein & Metaphone)

[12]`https://www.dropbox.com/sh/76m55plnzu6iasd/RsHVxIpfjr`
[13]`http://en.wikipedia.org/wiki/Levenshtein_distance`
[14]`http://en.wikipedia.org/wiki/Metaphone`
[15]`http://en.wikipedia.org/wiki/Soundex`

Table 2 shows the results for the experiments conducted where levenshtein distance was used as character edit distance T_c and Metaphone was selected as phonemic edit distance T_p. The average number of candidates for OOV words were found to be quite less compared to the paper. But, since they do not say which distances they have used, it would not be fair to compare these results. Implementing only character edit distances, $T_c \leq 2$ gives better recall that $T_c \leq 1$. But the average number of candidates are increased from 5 to 22. Implementing only phonemic edit distances, $T_p = 0$, the recall was similar to the case with $T_c \leq 1$. Combining both the strategies yielded better results then both of the individual strategies. Recall increased upto 54.48% with 29 average candidates per OOV word. Experiments with $T_p > 0$ needed too much computation time. For example, $T_p \leq 1$, which was only tested on a few of these words, resulted in 456 average number of candidates per word. Results for such experiments are not included in the table because they were only tested on a fraction of the dataset.

Criterion	**Recall**	**Average Candidates**
$T_p = 0$	50.65%	33
$T_c \leq 2 \vee T_p = 0$	59.18%	45

Table 3: Results for experiments with different T_c and T_p values (levenshtein & Soundex)

Similar experiments were performed by taking Levenshtein and Soundex as the edit distances. Results are shown in table 3. If we compare the results with table 2, we can see that the performance is improved but the number of candidates have also increased at the same time. All these experiments also prove that combining character edit distance with phonemic edit distance is a good strategy because they provide a greater probability in finding the correct form of an OOV word.

It is to be noted that, for all the results presented above, the recall is quite less compared to the results of the paper. To find out the reason behind this, an analysis was performed on their OOV words and the OOV words of this research. It was found that their list of OOV words only contained around 5% of Named Entities (e.g. *greenbay*). Whereas, list used in these experiments had around 15% of them (e.g. *JLaw, Zazeera, zayn*). These entities were not captured in the TwitIE pipeline because most of them are too noisy. For example, *JLaw* stands for *Jennifer Lawrence* and a more sophisticated NER system is required to capture such entities. Another reason is that they also perform normalization on ill-formed words[16] which was not done in this research. The analysis can be verified by looking at the list of words those could not be successfully suggested in the experiments. For each experiment, such a list is presented at the bottom of the file *oov.txt*. It can be seen from these lists that most of the words, those could not be suggested successfully are either irregular named

[16]Words containing irregular symbols such as '-', e.g. Corporate-conservative

entities (*JLaw, Zazeera, zayn*), ill-formed words (*Corporate-conservative, m-a-d-e-i-n, U-KISS, J-Law, so-california*) or different language words (*tetas, senos, pechos*).

Criterion	**Correct Candidate Selection**
$T_c \leq 1$	88.27%
$T_c \leq 2$	87.02%
$T_p = 0$	88.15%
$T_c \leq 2 \vee T_p = 0$	88.57%

Table 4: Successful correct candidate selection (levenshtein & Metaphone)

Table 4 shows the results for successfully selecting the correct candidate once the candidate was in the suggestion set. It can be easily inferred that the best candidate selection strategy works quite well in all the cases. Table 5 shows the same results with Soundex phonemic edit distance. In all the experiments, the performance of slang detection was quite good. More than 90% of the time, slangs were correctly detected and replaced with their well formed counterparts.

Criterion	**Correct Candidate Selection**
$T_p = 0$	81.07%
$T_c \leq 2 \vee T_p = 0$	88.17%

Table 5: Successful correct candidate selection (levenshtein & Soundex)

7. Conclusion & Future Work

The goal of this research was to verify and improve some of the results obtained by Han et al. in their paper. With the help of TwitIE pipeline, it was shown that number of OOV words could be significantly reduced by separating the pre-known entities beforehand. Through the conducted experiments, claim of the paper about combined edit distances was verified. The research also hinted that slang detection performance can be improved by using a larger corpora. A candidate selection strategy was suggested which was able to suggest the correct candidate with high probability.

The results for *Recall* in the experiments suggest that tweets can be noisier than one can imagine. Hence, to get better normalization, one needs to come up with sophisticated strategies. The performance of the experiments conducted in this research can be improved by implementing a more sophisticated NER component that works well with irregular named entities. The system can also be improved by integrating additional components such as *ill-formed word detection* and *different language word elimination.* Experiments with more types of edit distances can help in finding the best combination to yield better normalization. There is also some scope of improvement in *Tokenization* and *Boundry Detection*.

Appendix A. Twitter Stream Python Code

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import time

ckey='xxxxxxxxxxxxxxxxxxxxxxx'
csecret='xxxxxxxxxxxxxxxxxxxxxxx'
atoken='xxxxxxxxxxxxxxxxxxxxxxx'
asecret='xxxxxxxxxxxxxxxxxxxxxxx'

class listener(StreamListener):

    def on_data(self, data):
        try:
            saveFile = open('America2.json','a')
            saveFile.write(data)
            saveFile.write('\n')
            saveFile.close()
            return True
        except BaseException, e:
            print 'failed onData,',str(e)
            time.sleep(5)
    def on_error(self, status):
        print status

auth = OAuthHandler(ckey, csecret)
auth.set_access_token(atoken, asecret)
twitterStream = Stream(auth, listener())
twitterStream.filter(track=["America"])
```

References

[1] A. Aw, M. Zhang, J. Xiao, J. Su, A phrase-based statistical model for sms text normalization, in: Proceedings of the NAACL Workshop on Creating Speech and Text Language Data With Amazon's Mechanical Turk., Association for Computational Linguistics, pp. 33–40.

[2] M. Kaufmann, J. Kalita, Syntactic normalization of twitter messages, in: Proceeding COLING-ACL '06 Proceedings of the COLING/ACL on Main conference poster sessions, Association for Computational Linguistics, 2010.

[3] B. Han, T. Baldwin, Lexical normalisation of short text messages, in: The 49th Annual Meeting of the Association for Computational Linguistics, Portland, Oregon, USA, 2011.

[4] K. Bontcheva, L. Derczynski, A. Funk, M. A. Greenwood, D. Maynard, N. Aswani, Twitie: An open-source information extraction pipeline for microblog text, in: Proceedings of the International Conference on Recent Advances in Natural Language Processing, Association for Computational Linguistics, 2013.