COMP90049 Report: Lexical Normalisation of Twitter data

Vedant Chauhan, 892758

1. Introduction

A “tweet” or Twitter message consists of 140 or fewer characters containing @ and # symbols. This 140 character limit encourages users to use shorthand notations and use spellings with grammatical errors and minimum elisions e.g. ‘c’ for see. In order to make sense of Twitter messages it is necessary to convert them into a canonical form making it consistent with the dictionary. This process is called lexical normalisation of individual words or tokens [1]. A Twitter message can be lexically analysed by breaking it into tokens and identifying each token on a case by case basis before normalisation techniques can be applied to make sense of the elisions and acronyms used in the message. The proceeding sections describe various techniques that are applied to identify “in vocabulary” words, “non-candidates”, and “out of vocabulary” words in a Twitter message [6].

1.1. In Vocabulary Tokens

The first step is to identify tokens that are an exact match in the dictionary i.e. “in vocabulary” (IV) words. In this project we have a lexicon of 234,371 words (dict.txt) to identify IV words. Tokens that fall outside this vocabulary are candidates for normalisation and if not fit for normalisation then classified as non-candidates.

1.2. Non-Candidate Tokens

In a normal twitter message, we have @, hash tags, punctuation symbols (ignored in this project), numbers, or any special characters which cannot be normalized, as “#” can be used to mark keywords in a tweet, and “@” can be used for tagging people are termed as “non-candidate” (NO) symbols are not processed for normalisation and can be identify by regular expressions.

1.3. Out of Vocabulary Tokens

Words which fall outside of vocabulary defined by dict.txt file, and does not contain any special characters, Twitter specific symbols, punctuations are marked as “out of vocabulary” (OOV) words and are a candidate for normalisation. These words are misspelt or shortened due to the character limit of Twitter. Transforming these words into their canonical form (closest match of the token from *dict.txt* file using normalisation technique) is done in this project.

2. Analysis

In this Project, Global Edit distance [3] and n-gram distance [5] are used as evaluation metrics. The following is the overview of normalisation process:

- Twitter messages file (tweets.txt) is read, and based on this file each token is identified as IV, OOV, and NO words.
- Punctuations are ignored in the tweet due to large dataset (As code is written in Python, large dataset reduces the efficiency and taking punctuations will increase the results file).
- Special characters, numbers, and Twitter specific symbols are marked as NO tokens.
- Remaining tokens are classified as IV if there is an exact matches in the *dict.txt* file with global edit distance (0L), and OOV tokens are processed for normalisation.
- OOV tokens are closest matched using global edit distance and n-gram distance. The closest match is shown in the result file.
- Efficiency of the program is calculated, time taken by the program to execute.
- The result of all tokens are stored in the *tokensResult.txt* in the format as follows: Token TAB Code TAB Canonical_Form where Code: IV, NO, AND OOV.

The following section explain in detail the normalisation techniques used to find the closest match for OOV tokens.

2.1. Global Edit Distance

The token is compared against the 234,371 words contained in the *dict.txt* file. The first set of matches is gathered by calculating the *Levenshtein distance* [4] between the tokens obtained from *tweets.txt* file and the words in the dictionary [2].

Levenshtein distance (LD) is an approximate string matching algorithm that measures the difference between two sequences. It is defined as mínimum number of single character edits (insert, delete, replace) required to change one word into the other. The parameters set for the Global edit distance as Match = 0, Insert/Replace/Delete = 1 leads to Levenshtein distance. The advantage of this distance is that we get the equivalent distance between two strings using this method for example if two strings are equal we get the distance as 0. For example,

String1 = "test" and String2 = "test", then LD (String1, String2) = 0 (no transformation needed), and if

String1 = "test" and String2 = "text", then LD (String1, String2) = 1 (one replacement 's' to 'x').

This technique matches each token with words in a dictionary file (dict.txt). This generates the first set of approximate matches based on the similarity of tokens. The results are crucial to determine IV and OOV tokens. As, discussed earlier LD = 0 gives IV. Other values of LD leads to OOV and are send to n-gram distance.

This distance matches the words exactly which have just one or two characters insertion, deletion or replacement. The words where some letters are getting repeated e.g. 'l' in "allow", where we can avoid adding one score to the insertion of the letter which is same as the previous letter will reduce the LD of such words.

2.2. N-Gram Distance

The token from LD is send to n-gram distance (n = 2) and compared against words in *dict.txt* file. Those matches are deemed fit as the closest match of a token.

N-Gram Distance is sequence of n items from a given sequence of speech or text. The items can be letters, words, syllables etc. All combinations of adjacent words in the source text leads to n-gram. The higher the n, more context is added. For example,

2-gram of "test" are 'te', 'es', 'st', we can also count boundary words i.e. '#t', 'te', 'es', 'st', 't#', where # represents boundary.

Simplicity and scalability are two advantages of n-gram. Optimum length depends on application – if n-gram is too short, important differences can't be captured, but if too long, general pattern of token can't be captured.

2.3. Improvements

The project doesn't give the best results for a tweet. There are multiple improvements that can be performed. First, with ever increasing acronyms in social media, combination of normalisation techniques is important rather than one. The accuracy can be improved by using Soundex (words with same soundex result will be returned) in combination with the implementation. The problem with soundex algorithm using alone is that it ignores the first character of the word i.e. keep it same during derivation of the resultant string, which can be the problem as there are many words which have the first letter as silent. Using this with LD, n-gram can give us the exhaustive possibilities and improve the prediction of the system.

3. Conclusion

It's a great challenge to normalise tokens with high accuracy with number of variations possible for a token. Complexity of acronyms, syllables used in social media is increasing day by day. It is important to consider various normalisation techniques available and choose the best suitable technique. The implementation of edit distance and n-gram does not effectively solve the problem of normalisation of Twitter data, but combinations of technique like, edit distance, Soundex, and n-gram will be a good start for a better result and accuracy.

References

[1] Ahmed B. Lexical Normalisation of Twitter Data. InScience and Information Conference (SAI), 2015 2015 Jul 28 (pp. 326-328). IEEE.

[2] Nicholson J., Approximate String Matching techniques (presentation). Available at: https://app.lms.unimelb.edu.au/webapps/blackboard/content/contentWrapper.jsp?content_id=&displayName=Lecture-Capture&course_id=_336172_1&navItem=content&href=/webapps/osc-BasicLTI-bb_bb60/frame.jsp%3Fcourse_id%3D_336172_1%26id%3Dlectur%26

[3] Edit Distance. https://en.wikipedia.org/wiki/Edit_distance

[4] Levenshtein Distance. https://en.wikipedia.org/wiki/Levenshtein_distance

[5] N-Gram Distance. https://en.wikipedia.org/wiki/N-gram

[6] 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, Portland, USA. pp. 368–378.