COMP90049 Project 1: Lexical Normalisation of Twitter Data

Student Name: Andres Medina
Student ID: 828936

1. Introduction

With over 500 million daily tweets and over 328 million users (Twitter, 2017), "Twitter" has become a highly attractive source of real-time data for analysis. However, the vast amount of information sometimes contrasts with the quality of the same. Messages with typing errors, twitter abbreviations, special characters and slang words are very common within this social network.

The goal of this project is to try to overcome this quality limitation by generating a lexical normalisation of a list of tweets by using some approximate string matching techniques. In addition, an assessment of their performance based on accuracy, precision and recall will be included.

2. Dataset

The dataset for this project is composed by a sample of Twitter messages gathered as part of a research of Han and Baldwin (2011). The dataset consists of a random sampling of 549 English tweets, each one composed of several tokens which will be used as an input to transform the tweet into its canonical form.
In addition to the dataset, two more files were used as input to the project:

- A file with the expected lexical normalisation output of every token within the dataset. E.g. For the word "pix", the file contains the output "pictures".
- A dictionary file with all the word that should be considered as "valid words" from this project perspective.

3. Evaluation Metrics

For evaluating the performance and correctness of the lexical normalisation task, the following indicators were used:

- Precision: Indicates the proportion of correct tokens translated from the total amount of translations attempted.

- Recall: Indicates the average proportion in which the correct translation is contained in the candidate subset proposed by the translation method.

The execution time is also included as part of the results but it is merely informative. It will not be used as an evaluation driver for the corresponding methodologies.

4. Methodology

The methodology used for transforming every token into its canonical form, considers classifying those tokens into three different categories:

- **IV (In Vocabulary):** The token is part of the given dictionary therefore it is considered a valid term.

- **OOV (Out of Vocabulary):** The token is not part of the given dictionary. In these cases, an approximate string matching method was used to find the more similar term inside the dictionary.

- **NO (Non-Candidates):** The token will not be translated as it has a special meaning. E.g. Twitter hashtags (#)

For classifying all the tokens into these three categories, a cascade process was applied:

1. **Step 1**: Remove punctuation

A regex operation was applied to remove

every term that is entirely composed of one or more non-alphanumeric tokens. E.g. "…" or "!".

2. **Step 2**: Special terms identification

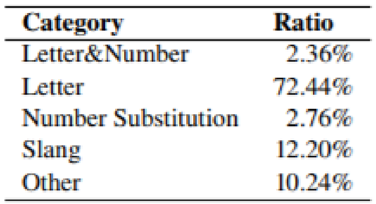
A regex operation was applied to identify tokens that are composed of non-alphanumeric tokens in addition to alphanumeric ones. E.g. "#heatchuuuuu", "@justtheebaddest". Any term identified in this step is classified as **non-candidate (NO)**.

3. **Step 3**: Terms in dictionary

Every token is searched in the given dictionary file. If the term exists in the dictionary, it is considered a valid term and therefore is classified as **in-vocabulary (IV).**

4. **Step 4**: Terms out of dictionary

In this last step, the remaining tokens **(Out of vocabulary - OOV)** are analyse in order to find the more similar term inside the dictionary. According to an analysis made by Han and Baldwin (2011) to 254 ill-formed words from 449 randomly selected twitters, they found that the OOV terms in those cases could be classified in the following categories:



Category	Ratio
Letter&Number	2.36%
Letter	72.44%
Number Substitution	2.76%
Slang	12.20%
Other	10.24%

Table 1: Ill-formed word distribution

Every category refers to the following:

-"**Letter&Number"** Instances which have both extra/missing letters (e.g. b4 "before")

- **"Letter"** Instances where letters are missing (e.g. shuld "should").

-**"Number Substitution"** Instances where numbers have been substituted for letters (e.g. 4 "for").

- **"Slang"** Instances where internet slang is used (e.g. "lol" refers to "laughing out loud")

The variety of categories shown in table 1, show evidence that different methods of approximate string matching should be used.

4.1. **Global Edit Distance (GED)**

In general, the GED is defined as the minimum cost of any sequence of edit operations that edits P into T or vice versa (Hasan et al, 2015). For this project, the Levenshtein distance variation (insertion, delete and replace are valid operations) was used[1].

This method was used for finding terms with the lowest global edit distance and it was found particularly useful for translating words in the "Letter" category.

E.g.

Input	Output	distance
tomoroe	tomorrow	2
shuld	should	1

Table 2: Examples of "Letter" categories translation using LD

In the other hand, the method didn't perform well in cases where twitter abbreviation or slang words were present.

E.g.

Input	Output	distance
ppl	people	3
rt	retweet	5

Table 3: Examples of abbreviation and slang words translation using LD

Finally, in cases where words have transpose letters, this method didn't perform well either.

E.g.

Input	Output	distance
shtim	smith	4

Table 4: Example of transpose letters translation using LD

[1] Imported by using "editdistance 0.3.1" python library.

4.2. **Phonetic Algorithms**

These kinds of methods consider the phonetic similarity rather than the string structure similarity. This is relevant to the task since tweets contains several abbreviations and unusual terms that in many cases sound alike but doesn't have a similar string structure.

Because of his nature, Soundex algorithm[2] was found to be very sensitive in relation to the first letter of the word. E.g. "night" and "knight" have different Soundex code. Same happens with "nd" and "and".

In addition, as it compresses every word into a 4 characters code, the number of elements into the candidate's set is very large which seriously affect the precision of the method.

Finally, according to Zobel and Dart work, Soundex also makes the error of transforming dissimilar-sounding strings such as **catherine** and **cotroneo** to the same code, and of transforming similar sounding strings to different codes (Zobel and Dart, 1996)

4.3. **Hybrid Solution**

A third approach was also considered for the translation task. A hybrid solution that uses GED as a first filter for establishing a subset of candidates and later, Soundex for narrowing even more the set of candidates. Finally, for narrowing the final set into a unique candidate for each token, a probability of occurrence ranking was used[3]. The most likely word of the subset was chosen.

In addition to the mentioned methods, a baseline was included in order to have a benchmarking starting point. This baseline was defined as a method that only includes the IV words as output. We expect that the results of the baseline show 100% of precision and a recall lower that all of the evaluated methods.

[2] Imported by using "jellyfish 0.5.6" python library

[3] The ranking was made by using a dictionary created by Jonathan Harris (http://www.wordcount.org) which was based in the British National Corpus.

5. **Results**

The results for the four steps transformation task are presented in table 5 and summarized in figure 1:

	Precision	Recall	Exec. Time [min]
Baseline	92.95%	68.93%	9.80
Levenshtein	3.91%	72.80%	50.14
Soundex	3.56%	72.70%	26.68
Hybrid Solution	56.26%	71.49%	51.58

Table 5: Summary of algorithms results for IV + OOV

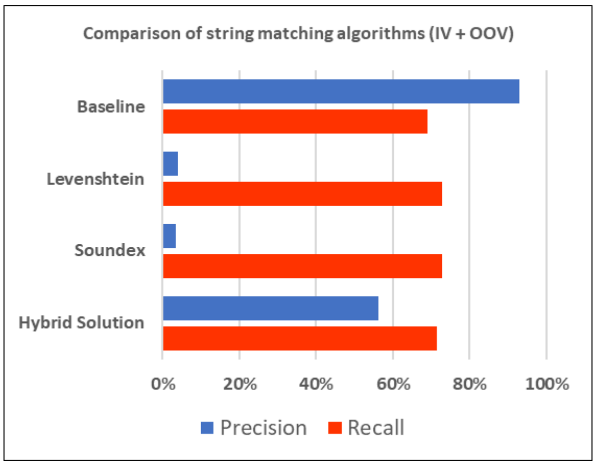


Figure 1: Summary of algorithms results for IV + OOV

5.1. **Considerations**

Contrary to what was expected, the baseline shows a precision of 92.95% instead of 100%. This means that some words that are in the dictionary differs with the expected output in the test file. Probably the test file was created by using other dictionary which certainly, could affect the results of the project.
Eg.

Baseline

Input	Classification	Output
'PIX'	IV	'PIX'

Test File

Input	Classification	Output
'PIX'	OOV	'PICTURES'

Table 6: Example of baseline output v/s test file output

In addition, by inspecting the test file we found cases where more than one output is expected by a given input.

Eg.

Test File

Input	Classification	Output
'ND'	OOV	'AND'
'ND'	IV	'ND'

Table 7: Example of multiple output for same input in test file

Based on those considerations and we will focus only in the OOV words results. These results can be found in the following figure:

	Precision	Recall	Exec. Time [min]
Levenshtein	4.73%	14.97%	50.14
Soundex	0.18%	14.60%	26.68
Hybrid Solution	9.41%	9.92%	51.58

Table 7: Summary of algorithms results for OOV

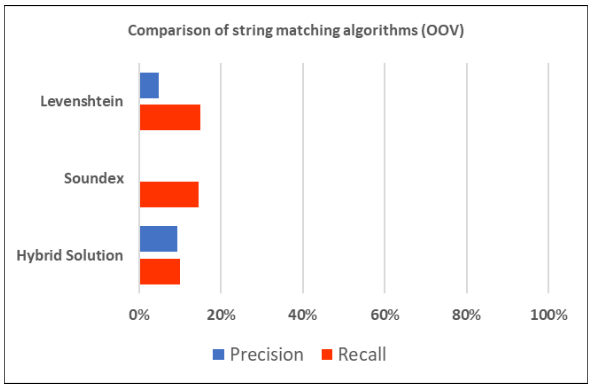


Figure 2: Summary of algorithms results for OOV

5.2. **Analysis**

Overall, it's clear that the best evaluated solution is the hybrid approach with an 9.41% of precision and an 9.92% of recall. Even when the Levenshtein and Soundex algorithms achieve a better recall, their precisions are considerably lower than the hybrid approach. In relation to the speed, the Soundex algorithm is considerably faster than its competitors.

The low recall of all the algorithms could be explain since the test file was created by using another dictionary file. It is highly probable that the provided dictionary doesn't have all necessary words for making a proper translation.

6. **Improvements**

Potential improvements are:

1. Consider other global edit distance methods. E.g. Damerau-Levenshtein Distance or Normalised Levenshtein Distance

2. Consider other phonetic algorithms more specific for the English language. E.g. Metaphone or Double Metaphone

3. Include a sentence structure analysis. E.g. Identify if the intended word is a noun, verb or adjective. Use this information as another driver to choose the right output.

4. Include a sentiment analysis. E.g. If some sentences have a clear sentiment polarity, the expected output word will probably follow the same polarity

5. Include an n-gram based method to improve the precision and recall of the prediction.

6. Include a dictionary with the slang words and their translation. This would help to tackle the "slang" words category inside the OOV words.

7. **Conclusions**

After evaluating the Levenshtein distance and Soundex methods, it is clear that both algorithms proved to be imprecise and not appropriate for the lexical normalisation task by their own. However, a simple hybrid approach shows a substantial improvement in the precision, results that suggested that it is possible to achieve acceptable levels of performance for the lexical normalisation task. Improvements to these approaches were included in the section 6 of the report.

References

[1] 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.

[2] Jonathan Harris, 2017 Dictionary of Probability of occurrence of English words <http://www.wordcount.org>

[3] Syeda Shabnam Hasan, Fareal Ahmed and Rosina Surovi Khan, 2015. Approximate String Matching Algorithms: A Brief Survey and Comparison. International Journal of Computer Applications (0975 – 8887) Volume 120 – No.8

[4] Hiroyuki Tanaka, 2016. Edit Distance Python library github. <https://pypi.python.org/pypi/editdistance>

[5] James Turk and Michael Stephens, 2017. Jellyfish Python Library github. <https://github.com/jamesturk/jellyfish>

[6] About Twitter website, viewed 06th September 2017, <https://about.twitter.com/en/company>

[7] Zobel, Justin and Dart, Philip. 1996. Phonetic string matching: Lessons from information retrieval.in Proceedings of the Eighteenth ACM SIGIR International Conference on Research and Development in Information Retrieval, Zurich, Switzerland, August 1996, pp. 166-173.