COMP90049 Project 1 Report: Lexical Normalization of Twitter Data

Shuai Wang

1. Introduction

The aim of this report is to assess the performance of token matching using Levenshtein Edit Distance and N-Gram Distance approximate matching methods on the problem of twitter normalization, and express the knowledge obtained from the project. Specifically, Edit Distance and N-Gram are applied to find the best matched word in a dictionary for each token in tweets dataset, and evaluate the efficiency of the two methods with illustrative examples and evaluation metrics which contain three measurable indicators: accuracy, precision and recall.

2. Dataset

The dataset [1] used in the project includes three parts: Labelled-tweets.txt, Labelled-tokens.txt and Dict.txt.

2.1. Labelled-tweets.txt

A collection of tweets, one per line. It has 549 tweets in total. Each tweet consists of a set of tokens delimited by space. For the convenience of processing, all upper-case (English) characters have been converted to corresponding lower-case form.

2.2. Labelled-tokens.txt

A collection of tokens, one per line. It has 8,841 tokens in total. The tokens are derived from the tweets in "labelled-tweets.txt" above. In practice, the token not containing at least one (English) alphabetical letter has no meaningful sematic like ",", ".", "?", ":", "…", etc. Thus, those tokens have been eliminated. The order of the tokens corresponds to the tweets.

The format of labelled token is below:

Token TAB Code TAB Canonical_Form

"Token" is derived from the labelled tweets file.
"Canonical_Form" is the normalized form of the token.
"Code" can be one of three below:

IV – "in vocabulary", which means the token can be found in dictionary. Its canonical form is itself.

OOV – "out of vocabulary", which means the token cannot be found in dictionary. Need normalization to provide a canonical form for the token.

NO – "Not a normalization candidate", which means there is no need to do normalization.

For example,

"grow IV grow" – "grow" is in dictionary, and the canonical form is itself "grow".

"goood OOV good" – "goood" is not in dictionary, and its canonical form is "good".

"@eskimo_joe_brrr NO @eskimo_joe_brrr" – ""@eskimo_joe_brrr" is not in dinctionary. No need to do normalization.

2.3. Dict.txt

A dictionary of a list of English words. It has 234,371 words in total.

3. Evaluation Metrics

Throughout this report, the following indicators will be used to evaluate the system:

Accuracy: For systems that predict only one token, which proportion are correct. The computing formula:

$$\text{Accuracy} = \frac{\textit{The number of correct predictions}}{\textit{The total number of predictions}}$$

Precision: For systems that predict one or more token(s), which average fraction are correct predictions among attempted predictions. The computing formula:

$$\textit{Precision} = \frac{\textit{The number of correct predictions}}{\textit{The total number of attempted predictions}}$$

Recall: For systems that predict one or more token(s), the proportion of tokens with a correct prediction. The computing formula:

$$\textit{Recall} = \frac{\textit{The number of tokens with a correct prediction}}{\textit{The total number of tokens}}$$

The labelled token file has provided canonical form tokens as ground truth which can be used to determine if a prediction(match) is correct or wrong. For this specific problem of token matching, the higher Accuracy value, Precision value and Recall value present better performance of the system.

4. Levenshtein Edit Distance

4.1. Basic

The system will find the Levenshtein Edit Distance between each token in labelled tokens file and each word in dictionary. To compute "accuracy", the system returns the word with lowest edit distance as the best match. Soundex

will be used where ties occur. One matched token per token. This is single-prediction system. To compute "precision" and "Recall", the system returns all words with the same lowest edit distance, so tie breaking will not be performed. This is multi-predictions system.

4.2. Algorithms

From token f to word t, given array A of |f|+1columns and |t|+1 rows. The algorithm of Levenshtein Edit Distance[4] follows:

```
function getDistance(f, t)
lf = strlen(f), lt = strlen(t);
A[0][0] = 0;
for(j=1; j<=lt; j++)
 A[j][0] = j * i;
for(k=1; k<=lf; k++)
 A[0][k] = k * d;
for(j=1; j<=lt; j++)
 for(k=1; k<=lf; k++)
 A[j][k] = min3(
 A[j][k-1] + d,//Deletion
 A[j-1][k] + i,//Insertion
 A[j-1][k-1] + equal(f[k-1], t[j-1]));//Replace or match
return A[lt][lf];
```

In particular, match(m) = 0; delete(d) = +1; insert(i) = +1; replace(r) = +1.

equal() return m if characters match, r otherwise.

The value of distance between f and t is A[lt][lf].

The system computes all distances from token f to each word t in dictionary. Comparing all distances, it finds the word(s) with the lowest distance, which is the best match(es). If multiple words get the lowest distance, use Soundex to break the tie.

The algorithm of Soundex[2] follows:

Soundex table has been used.

Letter	value
a, e, i, o, u, h, w, y	0
b, p, f, v	1
c, g, j, k, q, s, x, z	2
d, t	3
l	4
m, n	5
r	6

Table 1: Soundex table

The algorithm of Soundex follows:

```
function soundex(String f)
 translate(f);//Except for initial character, translate string characters according to table
 removerDuplicates(f);//Remove duplicates (e.g. 4444 -> 4)
 removeZeros(f);//Removes 0s
 truncate(f);//Truncate to four symbols
 return f;
```

For example, the Soundex representation of "king" is "k52".

For those words with the same distance, compute their Soundex representations, and then find the word with same Soundex representation, return it as best match; If multiple words have same Soundex representation, randomly choose one and return it. If there is no such a word, randomly choose one and return it.

4.3. Implementation

The Levenshtein Edit Distance and Soundex have been implemented in Java. Please check "README.txt" and source code to gain more details.

4.4. Result

The results are presented below:

Indicators	Values
Accuracy	72.1%
Precision	26.3%
Recall	74.2%

Table 2: Result of Levenshtein Edit Distance

4.5. Analysis

The Levenshtein Edit Distance has achieved 72.1% Accuracy, 26.3% Precision and 74.2% Recall.

For the Accuracy of 72.1%, Levenshtein Edit Distance is effective. In labelled tokens file, the tokens with a "code" of "NO" are excluded as they are not English words. Thus, the methods only used the tokens with a "code" of "IV" or "OOV". If a token is in a dictionary, the match word predicted by the method has a high possibility of correct prediction. For the tokens with a "code" of "OOV", they are not in dictionary, so the method is used to find the most approximate words not exact words in dictionary. This will generate wrong predictions which are different from canonical form of the tokens. It will reduce Accuracy. In general, Levenshtein Edit Distance performs well.

There is a huge difference between the Precision of 26.3% and Recall of 74.2%. The Recall is well, it refers to that Levenshtein Edit Distance matching algorithm can find majority of correct words for tokens. However, the method returns multiple possible words with same distance for each token, which leads to a huge denominator that is the total number of attempted predictions. So, the Precision is very low.

In general, Levenshtein Edit Distance matching technique performs good.

There is an illustrative example:

For single-prediction system to calculate Accuracy:

Token	Canonical_Form	Predicted Word	Right /Wrong?
new	new	new	√
pix	pictures	pix	×
comming	coming	coaming	×
tomoroe	tomorrow	tomorn	×
dang	dang	dang	√
effin	fucking	effie	×
greenbay	greenbay	greenback	×
bullshit	bullshit	bullit	×
calls	calls	calas	×
nd	and	ned	×

Table 3: Example of Levenshtein Edit Distance

Accuracy = 2/10 = 0.2

For multi-predictions system to calculate Precision and Recall:

Token	Canonical_Form	Predicted Word	Right/ Wrong?
new	new	new	√
pix	pictures	pix	×
comming	coming	coaming combing coming tomming	× × √ ×
tomoroe	tomorrow	tomorn tomorrow toorie tooroo totoro	× √ × × ×
dang	dang	dang	√
effin	fucking	effie elfin	× ×
greenbay	greenbay	greenback greenbark greenery greeney greenly greeny	× × × × × ×
bullshit	bullshit	bullit bullskin bullwhip	× × ×
calls	calls	calas call calla calli callo callus carls	× × × × × × ×
nd	and	ad and d ed	× √ × ×
		end id ind n na ne ned ni nid no nod nu od td ud	× × × × × × × × × × × × × × ×

Table 4: Example of Levenshtein Edit Distance

Precision = 5/49 = 0.102

Recall = 5/10 = 0.5

5. N-Gram Distance

5.1. Basic

In N-Gram Distance[3][5] technique, an n-gram is a contiguous sequence of n items from a token. The distance between two tokens refers to the difference between the numbers of n-grams of two tokens and two times of common n-grams. The computing formula follows

$$|G_n(s)| + |G_n(t)| - 2 * |G_n(s) \cap G_n(t)|$$

This is the N-Gram Distance between token s and token t.

5.2. Example

Assume n=2. There are tokens: “calls”, “call” and “called”.

2-grams of calls: #c, ca, al, ll, ls, s#

2-grams of call: #c, ca, al, ll, l#

2-grams of called: #c, ca, al, ll, le, ed, d#

2-Gram Distance between “calls” and “call”:

$$|G_n(calls)| + |G_n(call)| - 2 * |G_n(calls) \cap G_n(call)| = 6 + 5 - 2 * 4 = 3$$

2-Gram Distance between “calls” and “called”:

$$|G_n(calls)| + |G_n(call)| - 2 * |G_n(calls) \cap G_n(call)| = 6 + 7 - 2 * 4 = 5$$

Thus, “call” is a better for “calls”.

5.3. Algorithm

Compute all N-Gram Distances between token and each word in dictionary, and then choose the word with lowest distance as best match. Where ties occur, Soundex will be used to break tie.

5.4. Implementation

The N-Gram Distance and Soundex have been implemented in Java. Please check “README.txt” and source code to gain more details.

5.5. Result

The results are presented below:

Indicators	Values
Accuracy	67.2%
Precision	72.8%
Recall	67.9%

Table 5: Result of N-Gram Distance

5.6. Analysis

The N-Gram Distance matching algorithm has achieved 67.2% Accuracy, 72.8% Precision and 67.9% Recall.

For the Accuracy of 67.2%, the algorithm performs not that effectively. As the length of words in dictionary is not very long, the number of n-grams a word produces is quite small. Thus, it returns so many words with same N-Gram Distance for tokens, which makes tie-breaking more challenging. Therefore, the Accuracy has been highly impacted.

For the Precision of 72.8% and Recall of 67.9%, the algorithm can find most of correct matched words for tokens. Surprisingly, the algorithm has a higher Precision. This is because if a token is identical with a word, it will not perform N-Gram, just return the word, which reduces the number of tokens really need to do n-Gram. Therefore, the number of attempted prediction has been reduced. Hence, it has a higher precision.

There is an illustrative example:

For single-prediction system:

Token	Canonical_Form	Predicted Word	Right /Wrong?
new	new	new	√
pix	pictures	pix	×
comming	coming	coming	√
tomoroe	tomorrow	toetoe	×
dang	dang	dang	√
effin	fucking	effie	×
greenbay	greenbay	greeny	×
bullshit	bullshit	bullit	×
calls	calls	callus	×
nd	and	ned	×

Table 6: Example of N-Gram Distance

Accuracy = 3/10 = 30%

For multi-predictions system:

Token	Canonical _Form	Predicted Word	Right /Wrong?
new	new	new	√
pix	pictures	pix	×
comming	coming	coming	√
tomoroe	tomorrow	toetoe torororo	× ×
dang	dang	dang	√
effin	fucking	effie elfin fin	× × ×
greenbay	greenbay	greeny	×
bullshit	bullshit	bullit	×
calls	calls	call callus	× ×
nd	and	and d end ind n ned nid nod	√ × × × × × × ×

Table 7: Example of N-Gram Distance

Precision = 4/21 = 0.19

Recall = 4/10 = 0.4

6. Conclusions

In this project – Lexical Normalization of Twitter Data, the Levenshtein Edit Distance and N-Gram Distance have been introduced to approximately match tokens in tweets with words in dictionary while Soundex is used to break tie where one of multiple words with same distance needs to be choose. The algorithms have been implemented in Java. Three evaluation indicators like Accuracy, Precision and Recall have been calculated to evaluate the performance of the two approximate matching algorithms. Some illustrative examples are also provided. Comparing the results of two algorithms, N-Gram Distance Distance performs better than Levenshtein Edit.

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] M. Odell and R. Russell, "The soundex coding system," US Patents, vol. 1261167, 1918.

[3] Kondrak, G. 2005. N-gram similarity and distance. In Proceedings of the 12h International Con- ference on String Processing and Information Retrieval (Buenos Aires, Argentina). 115–126.

[4] Freeman, A. T., Condon, S. L., & Ackerman, C. M. (2006, June). Cross linguistic name matching in English and Arabic: a one to many mapping extension of the Levenshtein edit distance algorithm. In Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of

Computational Linguistics (pp. 471-478). Association for Computational Linguistics.
[5] Cavnar, W. B., & Trenkle, J. M. (1994). N-gram-based text categorization. Ann Arbor MI, 48113(2), 161-175.