Spell Corrector to Social Media Datasets in Message Filtering Systems

Zar Zar Wint[*], Théo Ducros[**], Masayoshi Aritsugi[§]

[*§]*Computer Science and Electrical Engineering*
Graduate School of Science and Technology, Kumamoto University
Kumamoto 860-8555, Japan
[*]zarwint12@gmail.com, [§]aritsugi@cs.kumamoto-u.ac.jp

[**]*Polytech Clermont-Ferrand*
University Clermont Auvergne
Clermont-Ferrand, France
[**]ducros.t64@gmail.com

***Abstract*—We develop a spell checker and corrector to check word errors in the social media datasets, which will be used in message filtering systems specially for cyberbullying detection. We use the dictionary techniques to check words and there are ten word spell error checking and correction approaches. If there are more than one corrected word we get from each approach, we use Levenshtein distance to choose the corrected word from the words in the dictionary. The spell correction results were around 90%. Moreover the percentage of each approach highlighted the efficiency of adding letters in the word.**

***Index Terms*—word error, spell checker, spell corrector, approach, dataset**

I. Introduction

There are two kinds of word errors in the real world text processing applications and websites: real word errors and non-word errors. Real word errors are the words that can be found in the dictionary and the users misspell them. The non-word errors are the words that cannot be found in the dictionary. The non-word spell errors happen when the users pressed the nearby key on the keyboard by accidentally, when the users spelled the words as they speak, etc. There have been many studies to check these kinds of word errors in the real world text processing applications and websites [1].

In this paper, we develop another spell checker and corrector to check and correct the words, which have already existed in social media datasets. We develop this to classify the bullied messages exactly. Sometimes, the social media use bullied or abusive words filter in their systems for example, Twitter uses the "mute word" to block unwanted words by the users. At that time, some users use misspelled words intentionally to bully others and avoid the filter. For example they use "bi+(h" instead of the bullied or abusive word "bitch". We have found that there are so many word errors in the social media datasets. The users intentionally did most of these word errors for fun and avoiding censorship from the filter. The word errors occur in some reasons, e.g., (1) Sometimes, people use characters to replace letters like "@" for "a", "0" for "o" , "(" for "c" , "3" for "e" , "$" for "s" , "+" for "t" and "!" for "i" (example: "bi+(h" instead of "bitch") and (2) Sometimes, users use repeated characters at the end of the word or middle the word like "hellooooooooo" instead of "hello" and "weeeekeeend" instead of "weekend".

Most of the word spell checkers and correctors rely on the dictionaries, which contain the corrected spelled words. By using the dictionaries, word spell mistakes are checked and corrected. Trying to find the word by looking in the entire dictionary can blow up in the time of complexity. There are some approaches which have tried to reduce the time complexity of spell correction. In our proposed system, we do not look in the whole dictionary to find if the word is correct or not. We only check the word starting with the same letter as the one we are testing. Sometimes, cyberbully detectors cannot detect well the bullied words because of the intentional misspelling. We have found these kinds of misspelled words in our testing datasets. The patterns are different so we develop different approaches to correct them as shown in Table I.

By adopting such approaches, we can investigate more precisely the characteristics of datasets, thereby allowing us to improve the detection with spell correction. Moreover, our proposed system uses the Levenshtein distance [2] to find how close a dictionary word to the misspelled word.

The remainder of this paper is organized as follows: In Section II, we review similar research work on spell error checking and correction. In Section III we present our proposed spell checker and corrector. In Section IV we present the experimental results using Twitter and Formspring.me datasets. Section V concludes the paper and mentions some future work.

TABLE I
EXAMPLE OF MISSPELLING PATTERNS IN DATASET

Misspelled Words	Corrected Words
uqly	ugly
Trashhy	Trashy
bitchezz	bitches
f**kin	f**king
siht	shit
slutz	slut
ihateyou	i hate you
p**syyyy	p**sy

[**] Work conducted while interning at Kumamoto University

978-1-5386-0664-3/17/$31.00 ©2017 IEEE

II. RELATED WORK

There have been many spell checkers and correctors in the real world applications. Some researchers are still trying to improve the spell checkers and correctors in every text processor by using modern techniques based on their needs.

Pande [3] presented a novel, unsupervised, and distance measure agnostic method for search space reduction in spell correction using neural character embedding. The embedding is learned by skip-gram word2vec training on sequences generated from dictionary words in a phonetic information-retentive manner. He tried to reduce the time complexity of spell correction by reducing the search space of words over which the search for correct spelling is to be done. He also used the dictionary of correctly spelled words during the training process. It is distance measure agnostic because once the search space is reduced then any distance measure of spell correction is used. His method is used as a filter before a spell correction algorithm. His method outperforms one of the recent methods, in terms of both extent of search space reduction and success rates. Our proposed system checks the same first letter word in the dictionary. We also use the Levenshtein distance [2] to find how close a dictionary word is to the corrected misspelled word.

A language-independent spell-checker based on an enhancement of the n-gram model is presented [4]. Their spell checker proposed correction suggestions by selecting the most promising candidates from a ranked list of correction candidates that was derived based on n-gram statistics and lexical resources. Their proposed approach in an application is for keyword and semantic-based search support. They concentrated on the correction of the non-word errors with revised n-gram based approaches because the pure n-gram based approach to compute the similarity coefficient did not consider the order of the n-grams. This kind of problem can be solved by our proposed approaches, too.

Yannakoudakis and Fawthrop [5] found that in most cases the first letter in a misspelled word is usually correct and the misspelled and real word would be either the same length or the length differs just by one. However, in social media datasets, there are so many repeated letters in the middle of the word and at the end of the word. Hence, the word length is too much larger than the real word. This kind of problem can be solved by our proposed approaches: deletion (removing end character approach and removing middle characters approach).

Smart Spell Checker System (SSCS) can adapt itself to a particular user by using user's feedback for adjusting its behavior [6]. The result of the adjustment is manifested in a different ordering of the suggestions to the user on how a particular spelling mistake should be corrected. SSCS uses the Adaptive Software Architecture (ASA) which contains number of components called Knowledge Sources, i.e., Left/Right Character Shifter, Character Doubler, End Character Appender, Character Remover and Subsequent Character Switcher. Each Knowledge Source is responsible for generating suggestions for correcting a specific type of error. Feedback is propagated to Knowledge Sources after the user makes a selection of the correction. In response to feedback, Knowledge Sources adjust their algorithms. Similar to this system, our system also uses these kinds of knowledge sources but our system does not take back the users suggestions. We use the Levenshtein distance [2] to find how close a dictionary word is to corrected misspelled word and get the corrected word among the suggested words.

Programs that do not check what you are typing as you type it are becoming increasingly annoying and the surprising number of programs that would not check your spelling at all are even worse. Despite this, editors and other tools used by programmers are least likely to offer spelling checking. Hughes [7] described the addition of spelling checking to a programmers' editor. The system not only operates when the author requests it but it is always checking in background. Ours only operates upon request but checks every single word in the given text.

III. SPELL CHECKER AND CORRECTOR

There are four steps in our word spell checker and corrector: (i) preprocessing step, (ii) dictionary look up, (iii) word spell checking and correction approaches, and (iv) spell corrector as shown in Fig. 1.

A. Preprocessing Step

When we look at social media datasets, they include various word patterns: chat abbreviation, syntax, short-term words, emotional icons, etc. Before doing the word error checking and correction, we transform these words to ordinary words. The first one is to remove all the chat abbreviation and to

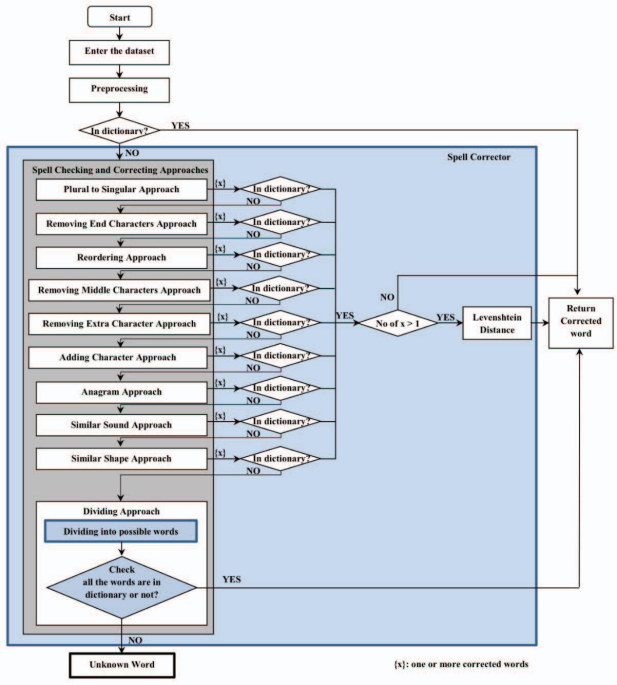

Fig. 1. Overview flow chart for our proposed system.

replace them by their meaning (e.g., “ppl” to “people”), syntax (e.g.,“&#39 ” to “ ’ ”), transform short-term to long-term word (e.g., “ltns” to “long time no see”)by using chat abbreviations file [8], and emotional icon to word (e.g., “<3” to “love”) by using smiley dictionary [9]. We will use some opinion mining files (which consist of positive and negative words) and we will assign weight value in our cyberbullying detection system. So we do these kinds of prepossessing steps.

B. Dictionary Look Up

After the preprocessing step, we use the dictionary file to check if the word exists in the dictionary or not. The dictionary we used in the proposed system is based on a file composed of 194,438 words [10]. We read this file and keep in memory the different words. Nevertheless, we do not only import the resources but we also create a list containing the beginning index of the word with the same first letter. We take a list of 26 slots and we put in the starting point in the dictionary. This list has 26 elements for the 26 letters of the alphabet. Therefore, element 0 corresponds to “a”, element 1 corresponds to “b”, ... , element 25 corresponds to “z”. In order to create this index we cross the dictionary once and, when the letter changes, we add the actual position to the list of index.

To do that, we need to check if it starts with a letter or not then we get the starting point of this letter in the list of index. For the ending point, we need to check if it starts with a “z” because there are two cases:

- If yes, the ending point is the end of the dictionary.
- If no, the ending point is the beginning of the next letter therefore we can obtain it in the list of index.

Once we have the beginning and ending points we can cross the dictionary between those two points in order to know if the word is inside. If it is, we return “true”. After looking at the dictionary file, the unknown words go through the new lists to word spell check and correct.

C. Word Spell Checking and Correction Approaches

Our proposed system consists of total 10 word spell correction approaches and each approach is described details in the following.

a) Plural to Singular Approach (PSA): This approach is mainly motivated by the fact that a dictionary does not necessarily contain all the plural and singular forms. Indeed, that would make the dictionary bigger, which means more time to get through it. That is why we developed that step which consists in checking the last character and if it is an “s” we remove it and check the word without. Moreover, it can also correct when people use wrong plural forms for example “puppys” for “puppies”.

b) Removing End Characters Approach (RECA): This approach is used to check and correct the word errors which are characterized by repeating the same characters at the end of the word (e.g.,“hellooooo” for “hello”)as shown in Fig. 2.

This kind of error can be solved by finding the first occurence of the repeated character. The program keeps the index and takes a substring of the original word from the first character to the first appearance of the repeated character. Nevertheless, some words have two occurrences of a character at the end like “tree”. That is why this method returns a list of Strings composed of the two possibilities in order to be checked. To determine the last repetition of the character, the program picks the last character up. Consequently it analyses from the end to the beginning until it finds a different character. Then it saves and returns the two possibilities. Nevertheless, if the word length is 1, it does not check the word.

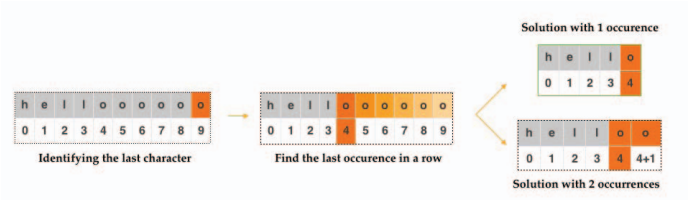

Fig. 2. Removing end characters.

c) Removing Middle Characters Approach (RMCA): Furthermore, some users add letters to accentuate some syllables in such words as “weeeekeeend” for “weekend”. To solve this kind of error, we have to delete the repeated characters. The first thing to do is to determine when a character is repeated. To do this step it crosses the word from the beginning to the end. Each time the character is similar, it increases a counter by one. When the character is different, if the counter is at 2 or more it creates 2 words: one with 1 occurrence and the other with 2 occurrences. Moreover, as the words are no more the same, it keeps in memory the different index of each word. After that we keep crossing words until it arrives at the end of all the new words as shown in Fig. 3.

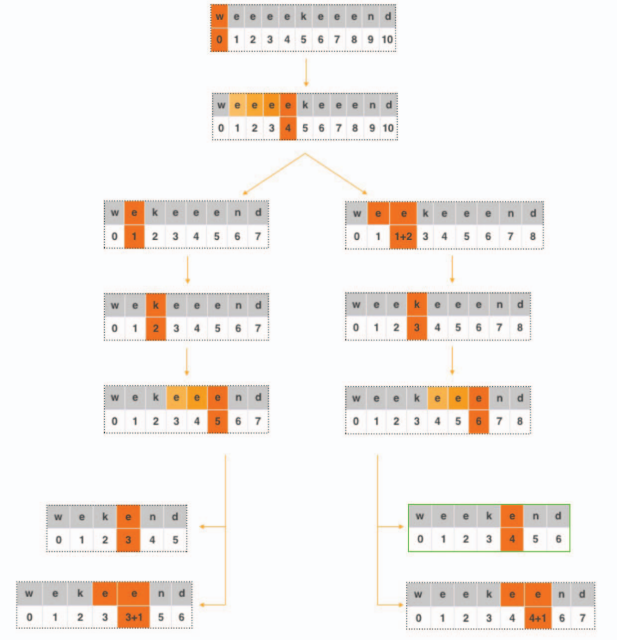

Fig. 3. Removing middle characters.

In order to make it more efficient, we suppose that the words with 3 occurrences or more are wrong then we do not keep them in memory anymore and replace them by the solution with one occurrence. The other new solution is added at the end of the list.

d) Reordering Approach (RA): The other frequent mistakes we found in the social media datasets are switching two characters like “spiol” instead of “spoil”.

For this problem, we suppose that the first letter of the word is the right one. Then we create different possibilities by exchanging all letters with its neighbor on the right until we reach the end of the word. Once it has done it returns a list with the possibilities as shown in Fig. 4.

e) Similar Shape Approach (SShA): Sometimes, people use similar character from a shape point of view in order to write their words. The program developed for this part was recursive but we find an iterative way to do it as shown in Fig. 5. The first character is considered as right and the others are put in a list of characters because it is easier to manipulate. The second step is to create the different groups for the letter with a similar shape as the following groups:

- Group 0: “c”, “e”, “o”, “a”
- Group 1: “I”, “j”, “l”, “t”, “f”
- Group 2: “u”, “v”
- Group 3: “m”, “n”, “r”
- Group 4: “p”, “q”, “g”, “y”
- Group 5: “h”, “k”, “b”

It takes the first character of the list. We obtain the index of the letter to know which group it belongs to. Once we have it, we get the list with all letters with similar shape. For the first character, we just copy the group in the solution list. For the other characters of the character list we take them one by one and we duplicated the list regarding the size of the group. For example, if two letters compose the group, we need to duplicate it once to have enough copy. After duplicating, we add to each copy one of the letters of the groups until we have done it for every possibility.

f) Similar Sound Approach (SSoA): Sometimes, the users mistype as they speak. This kind of mistake is common because we speak more than we write and read and as a consequence we can confuse how we have to write the word as we speak. The way used to resolve that problem follows exactly the same path as the previous one. Indeed, we create groups and index and duplicate it regarding the number of the

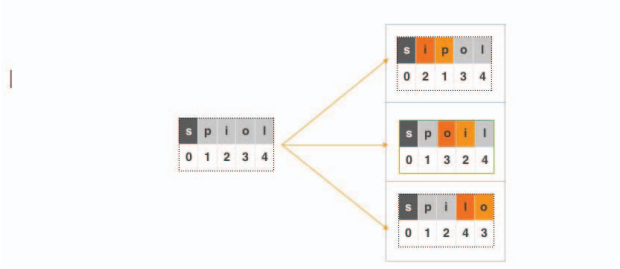

Fig. 4. Exchanging two consecutive characters.

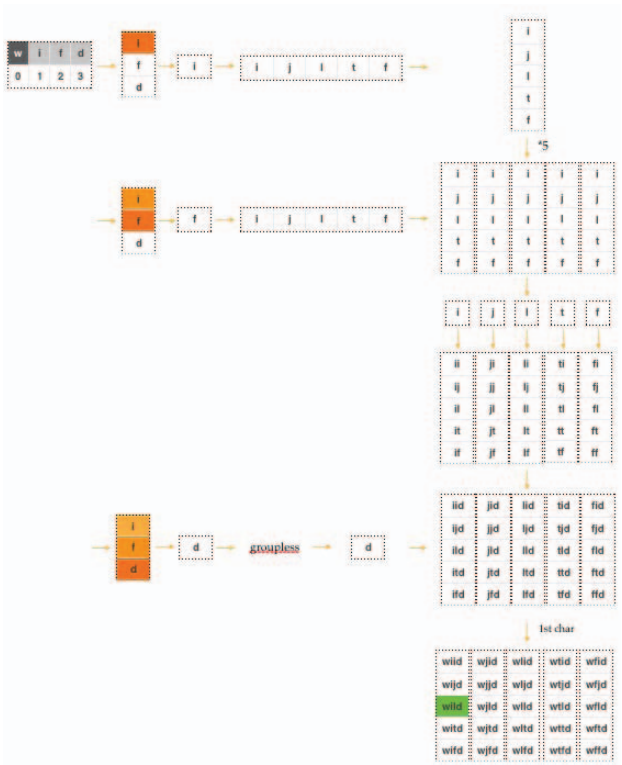

Fig. 5. Reforming the same shape character.

characters in the group. This time the groups were composed with one-letter or two-letter words in order to change them. However, we never check a two-character sound in the words.

- Group 0: “k”, “c”, “ck”, “ke”
- Group 1: “y”, “I”
- Group 2: “d”, “th”
- Group 3: “u”, “ou”, “o”
- Group 4: “v”, “ve”
- Group 5: “z”, “s”
- Group 6: “n”, “ne”

g) Dividing Approach (DA): Furthermore, it is not rare to write a phrase in one word from a grammatical point of view. For example, “alittle” for “a little” as there is no space between those two words and thus they become one for the machine. In order to resolve this, we have developed the following program as shown in Fig. 6. The main concern when we do that is to do not create new words. That is why we take the whole word and look if this word without the first character is an existing word. We keep removing the first character until we find a

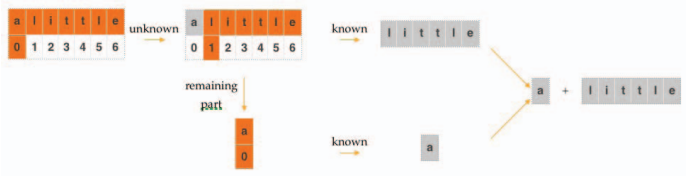

Fig. 6. Dividing the combined words.

word. Once we have one, we put it aside and keep looking for a new word in the remaining part. The dictionary contains one-letter words and some two-letter words, which can create non-sense. That is why we require at least a four-letter word. The dictionary has to contain all new separated words. As a consequence if it does not fit this by having an unknown word in it the word will remain unchanged.

h) Removing Extra Character Approach (RExCA): There are a lot of words that have extra character in them like "infereior" as shown in Fig. 7. The idea of how our system operates is really basic we take the second letter (we assume the first one to be corrected), remove it and stock the new created word. And we do it with the next characters as long as we get through the word. Moreover, after implementing it we noticed that we should not use it for words shorter than 5 characters. Indeed, it corrected a lot of them for something else than their meaning that is why we established it.

i) Adding Character Approach (ACA): Sometimes, the users remove all the vowels in some word. That is why, we separate the words in two categories, those with only consonants and those with both consonants and vowels as shown in Figs. 8 and 9. Indeed, we do not operate the same way according to the type of the words. When there are no vowels, regarding the length of the word, you might need to add up to two of vowels and not only one. Moreover, you should not add more consonants. That is why we add a vowel at every spot available and add another to those new created words. For the other kind of words, we try to add all the letters of the alphabet at every spot in the word. As a non-vowel word, for example we can take "n" for "no" and "goin" for "going". "n" is relevant only to explain how the program operates because otherwise the number of words becomes really important.

j) Anagram Approach (AA): Sometimes, people mistype the word with the characters in unorderly. To correct that kind

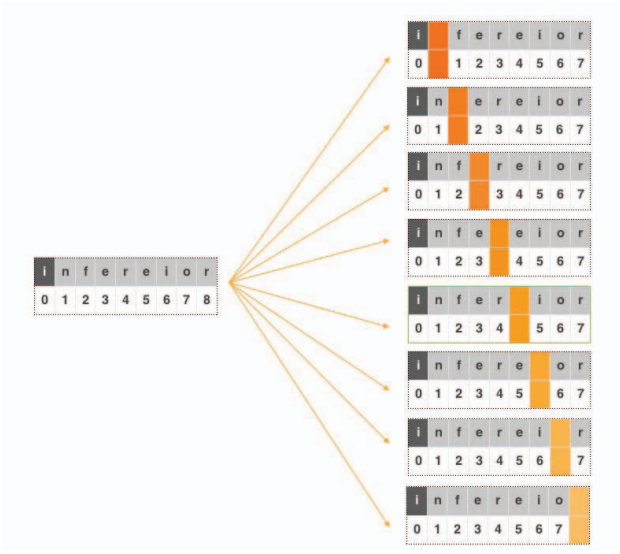

Fig. 7. Removing extra character.

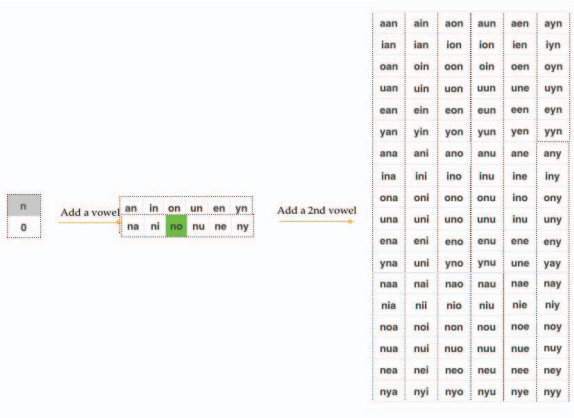

Fig. 8. Adding vowels.

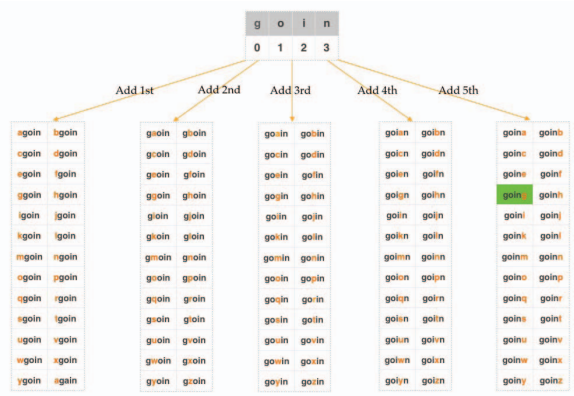

Fig. 9. Adding a character.

of mistake we create and check all the different combinations with the letter in the word as shown in Fig. 10.

We put all the character of the word in a list except the list one that we assume correct. Then we use a recursive algorithm to create the possibilities. We get through the list of the characters one by one and launch the same algorithm with the same list minus the used character until the size list drops to one (which is the shutoff parameter) and return it. Going back to the previous step will make us add the letter and return the new solutions. In order to avoid duplication we use a list that contains the letters already used at that position. If the list contains the character we are going to use, we skip it otherwise we operate as explained as above and add it to that list.

According to how we operate with the different approaches we can divide them in 5 categories as follows:

- Deletion when we remove letters: PSA, RECA, RMCA and RExA.
- Insertion when we add characters: ACA.
- Transformation when we change characters to another: SSoA and SShA.
- Separation when we split a word: DA.
- Reordering when we modify the order of the characters: RA and AA.

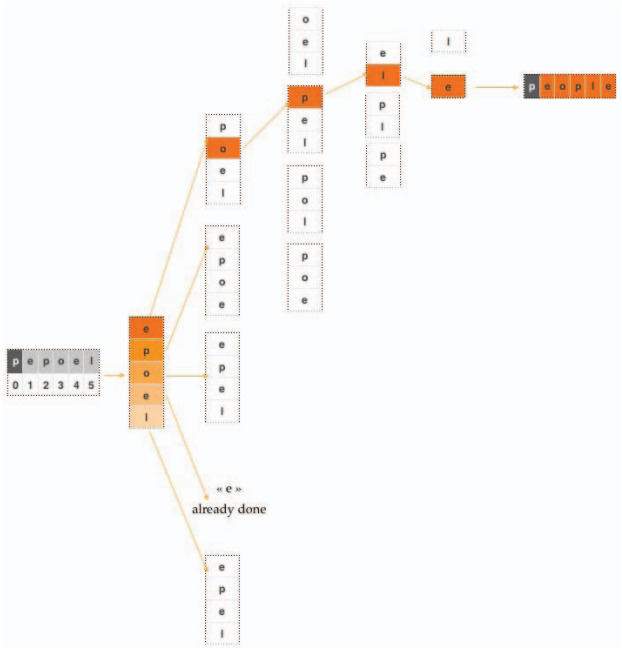

Fig. 10. Reordering the character in the word.

D. Spell Corrector

The proposed spell corrector is designed to correct the words that have two or more characters. Moreover, in the social media datasets, there are hashtags, pseudonyms, websites links and numbers and we escape these to check and correct. The proposed spell corrector does the following approaches in orderly to get the best correction results:

- Plural to Singular Approach (PSA)
- Removing End Characters Approach (RECA)
- Reordering Approach (RA)
- Removing Middle Characters Approach (RMCA)
- Removing Extra Character Approach (RExCA)
- Adding Character Approach (ACA)
- Anagram Approach (AA)
- Similar Sound Approach (SSoA)
- Similar Shape Approach (SShA)
- Dividing Approach (DA)

If there are more than one corrected word in one approach, the proposed system uses Levenshtein distance to choose the suitable word.

IV. EXPERIMENTAL RESULTS

In this section, an experiment of the proposed system and performance results are discussed. We use the Formspring.me dataset [11] and Twitter Sentiment Analysis Dataset [12] in our experiment. Each dataset has two files: positive and negative files. By using the spell corrector with the orderly approaches which were mentioned in the previous section, we got better correction rate than other orders. The word correction rate for Twitter dataset in Table II is 89.5% on average. The word correction rate for Formspring.me dataset in Table III is 90.5% on average.

TABLE II
WORD CORRECTED RESULTS FOR TWITTER DATASET

	Positive file			Negative file		
	1st Test	*2nd Test*	*3rd Test*	*1st Test*	*2nd Test*	*3rd Test*
Total Words	90 761			79 992		
Unknown Words	3 556			2 907		
SPA		182	182		150	150
RECA	400	381	381	348	338	338
RA	163	154	154	91	89	89
RMCA1	198	190	190	213	211	211
RExCA			498			379
ACA	1 430	1 365	1 079	1 019	910	728
AA	82	79	39	92	87	46
SSoA		97	71		85	80
SShA	425	357	286	346	321	253
DA		399	362		335	299
Corrected	2 698 (76%)	3 204 (90%)	2 109 (91%)	3 242 (72%)	2 526 (86%)	2 573 (88%)

TABLE III
WORD CORRECTED RESULTS FOR FORMSPRING.ME DATASET

	Positive file			Negative file		
	1st Test	*2nd Test*	*3rd Test*	*1st Test*	*2nd Test*	*3rd Test*
Total Words	19 699			258 356		
Unknown Words	2 313			15 477		
SPA		100	100		831	831
RECA	486	448	448	2 768	2 642	2 642
RA	108	105	105	792	779	779
RMCA1	126	122	122	990	977	977
RExCA			306			2016
ACA	788	743	625	5 224	4 657	3 779
AA	44	44	24	389	368	174
SSoA		130	101		471	389
SShA	265	236	168	1 385	1 206	871
DA		135	120		1 706	1583
Corrected	1 817 (78%)	2 053 (89%)	2 119 (91%)	11 548 (75%)	13 637 (88%)	14 041 (90%)

After reaching 90% of the unknown word corrected, we looked at the remaining unknown words to try to understand why they were unchanged.

One of the reasons is the use of pseudonym without the regular @ to symbolize it. Furthermore, some of these words contain sound spelling and shape spelling mistakes at the same time. However, it would need too many memory and time to combine those two methods and correct such words. Another pattern we noticed is when the user mistyped abbreviations (e.g., “omgggggggz”), which make them unrecognized during the preprocessing step. As the dictionary does not know the abbreviations, it does not reach a solution during the correcting step. Finally, some of them are really unique as they do not follow a pattern (e.g., “zobliiously”) or they are no understandable even by a human being (e.g., “uooacuooac”).

According to Table IV, we can say that the deletion category is the most effective. However, as shown in Table V, the best approach is ACA.

We also tested the corrected datasets to classify the bullied and no bullied message by using the Naive Bayes Classifier and Convolutional Neural Network (CNN). For these tests, we used Formspring.me and Twitter datasets. CNN text classifier gets better accuracy rate with the corrected Twitter dataset

TABLE IV
WORD CORRECTED PERCENTAGE FOR CATEGORIES

Category	Twitter	Formspring.me
Deletion	28.89%	36.60%
Insertion	27.96%	24.75%
Transformation	10.68%	8.59%
Separation	10.23%	9.57%
Reordering	5.07%	6.07%

TABLE V
WORD CORRECTED PERCENTAGE FOR APPROACHES

Approach	Twitter	Formspring.me
ACA	27.96%	24.75%
RExCA	12.57%	13.05%
RECA	11.12%	17.36%
DA	10.23%	9.57%
SShA	8.34%	5.84%
RMCA	6.20%	6.19%
SPA	5.13%	5.23%
RA	3.76%	4.96%
SSoA	2.34%	2.75%
AA	1.31%	1.11%

but not with the corrected Formspring.me dataset as shown in Table VI. However, Naive Bayes classifier gets better accuracy rate with the corrected Formspring.me dataset but not with the corrected Twitter dataset as shown in Table VII. This would be mainly because the datasets have different characteristics as shown in Tables IV and V. Moreover, each classifier has its own way to operate. Naive Bayes classifier uses one word as one feature and CNN uses the three words as one feature. Our proposed system changes short-term words to long-term word such as "ltns" to "long time no see" in the preprocessing step and also divides combined words such as "thatawkwardmoment" into "that awkward moment" in the dividing approach. In Twitter dataset, users used short-term words and combined words more than Formspring.me users. Therefore these kinds of short-term word change to long-term word are not much effective in Naive Bayes classifier because it uses one word as one feature. CNN is more effective because they use the 3 words as one feature. Our experimental results indicate that it would be valuable to take such characteristics of datasets as shown in Tables IV and V into account for cyberbullying detection.

TABLE VI
ACCURACY RATE WITH CNN CLASSIFIER

Datasets	Original	Corrected
Twitter	95.14%	96.10%
Formspring.me	91.13%	89.38%

TABLE VII
ACCURACY RATE WITH NAIVE BAYES CLASSIFIER

Datasets	Original	Corrected
Twitter	97.21%	97.02%
Formspring.me	89.38%	90.00%

V. CONCLUSION

In this paper, we discussed the spell checker and corrector to correct the various kinds of misspelled words, which are included in social media datasets, e.g., Twitter and Formspring.me. We were able to correct on average 89.5% for Twitter dataset and 90.5% for Formspring.me dataset. When we used the corrected Twitter dataset to classify the positive and negative sentences, we got the more accuracy rate than the original dataset we used. According to the experimental results, the proposed system is working well to correct when users miss one to two characters in the words. Furthermore, it would be interesting to be able to predict if it is better to use the corrected file or the original one from the text's behavior. In the future, we should work on finding out the characteristics of the datasets that would have a better classifying rate after correcting. We will also consider the synonyms words in spell corrector.

REFERENCES

[1] http://www.onlinecorrection.com
[2] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", Soviet Physics Doklady, vol.10. Soviet Union, pp.707–710, February 1966.
[3] H. Pande, "Effective search space reduction for spell correction using character neural embeddings", Proc. of the 15th Conference of the European Chapter of the Association for Computational Linguistics, pp.170–174, 2017.
[4] F. Ahmed, E. W. D. Luca, and A. Nurnberger, "Revised N-Gram based Automatic Spelling Correction Tool to Improve Retrieval Effectiveness", Polibits., pp.39–48, 2009.
[5] E. J. Yannakoudakis and D. Fawthrop, "An intelligent spelling error corrector", Information Processing and Management, vol. 19, no.1, pp.101–108, 1983.
[6] D.Seth and M.M. Kokar, "SSCS: A Smart Spell Checker System Implementation Using Adaptive Software Architecture", In: Laddaga R., Shrobe H., Robertson P. (eds) "Self-Adaptive Software: Applications", IWSAS 2001. Lecture Notes in Computer Science, vol. 2614. Springer, Berlin, Heidelberg, 2003.
[7] E. Hughes, "Checking Spelling in Source Code", ACM SIGPLAN Notices, vol.39, no.12, pp.32–38, December 2004.
[8] http://web.ncf.ca/aj624/chatabbreviations.html
[9] http://www.planetpals.com/smiley.html
[10] http://www.gwicks.net/dictionaries.htm
[11] K. Reynolds, "Using Machine Learning to Detect Cyberbullying", Proc. of the 2011 10th Conference on Machine Learning and Applications Workshops, vol. 2, pp. 241–244, December 2011.
[12] http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/