Fast Language-Independent Correction of Interconnected Typos to Finding Longest Terms

Behzad Soleimani Neysiani
Department of Software Engineering, Faculty of Computer & Electrical Engineering,
University of Kashan,
Kashan, Esfahan, Iran,
B.Soleimani@grad.kashanu.ac.ir

Seyed Morteza Babamir
Department of Software Engineering, Faculty of Computer & Electrical Engineering,
University of Kashan,
Kashan, Esfahan, Iran,
Babamir@kashanu.ac.ir

***Abstract*— Triagers deal with bug reports in software triage systems like Bugzilla to prioritizing, finding duplicates, and assigning those to developers, which these processes should be automated, especially for substantial open source projects. These bug reports must be mined by text mining, information retrieval, and natural language processing techniques for automation processes. There are many typos in user bug reports which cause low accuracy for artificial intelligence techniques. These typos can be detected based on standard dictionaries, but correction of these typos needs human knowledge based on the context of bug reports. It is essential which neither Google Translator nor Microsoft Office Word can detect interconnected terms –a common type of typos in bug reports- having more than two meaningful terms. This research provides a novel language-independent approach for fast correction of interconnected typos based on natural language processing and human neural network structure to detect and correct interconnected typos — a new tree-based method proposed for term matching. Also, two algorithms proposed for a fast finding the longest meaningful term in an interconnected typo. A dataset is used including 180-kilo typos based on four famous bug report dataset of Android, Eclipse, Mozilla Firefox, and Open Office projects. Then proposed method evaluated on typos versus state of the art. The results show the runtime performance of the proposed method is as same as the related works, but the average length of words is improved and at least more than 57% of typos in the dataset can be classified as interconnected typos.**

Keywords*— Information Retrieval, Natural Language Processing, Duplicate Detection, Bug Reports, Typo Correction, Lexical Interconnected Typo, Trie

I. Introduction

Many massive projects, especially open source projects have a large range of analyzers, designers, developers, testers and end users, which after each new release, all of them may find some issues or bugs and/or have some suggestions to improve the software. Software triage systems such as Bugzilla are software which usually gets these reports online and then the Triagers will deal with these bug reports to evaluate the importance and priority of each report, finding duplicate reports based on their contents, assign bug reports to developers for checking bugs and planning to modify the project in future [1]. Because of the large amount and volume of bug reports, many researchers have tried to automate these processes since 2004 by artificial intelligence techniques and algorithms[1]. Duplicate bug reports detection is an essential problem in this research area [2, 3]. The algorithms and techniques of duplicate bug report detection such as Term Frequency and Inverse Document Frequency in information retrieval technique need to check the similarity of two bug reports to each other word by word, so the lexical correctness of words and terms is essential for these techniques [4]. There are many typos in bug reports, e.g., more than 50% of bug reports have typos, and more than 2.5% of bug reports have more than 50% typos [4]. These typos distort similarity detection process in duplicate detection. It is vital to detect and correct these typos automatically because there are more than 1.5 million typos [4] in Mozilla Firefox, Android, Open Office and Eclipse datasets [5] and about 390-kilo unique typos in those. A scientific semi-dictionary is made for typo detection in bug reports to detect typos automatically [4] including general English words and many scientific words like abbreviations or proper nouns. This semi-dictionary can be made for every language based on some valid reference like computer dictionaries or reference websites.

There are many types of typos in texts such as additional, removal, or substitute characters. Interconnected terms are a regular typo in the software context because there are many method or class names in this context which contains interconnected terms like ‘getItemById’ or ‘printAllMembers’. Sometimes these words are camel case, and sometimes users typed them and have not any specific case sensitivity. Also, sometimes typists forgot to press space between words so there will be many interconnected terms in the software bug reports or even other contexts too. Also, it is possible to find some interconnected typos in optical character recognition (OCR) output too. These interconnected terms must be separated otherwise humans and/or computers algorithms, and methods like term frequency of information retrieval techniques cannot recognize the text or detect similarities for duplicate bug report detection problem. The primary purpose of this research is to figure out how does correct these typos rapidly.

The organization of the paper is as follows: section 2 explains the literature view and related works. Section 3 describes the methodology of interconnected typo correction, section 4 will discuss evaluation results in experiments, and section 5 will conclude the research.

II. Literature Review

Typo detection and correction is a regular and an ancient issue in text mining and natural language processing [6, 7]. There are many efforts on typo detection and correction in a scientific context like clinical records, which uses Shannon's noisy channel model to predict next words based on the previous word sequence [8]. In some case, there is less previous word sequence like web query, so the log of web query can be used as a baseline, and maximum entropy model can help for rare queries to conquer the sparseness problem of prior data [9].

Some researchers focus on correction of misspelled typos by different kind of machine learning and natural language processing models, e.g., creating a confusion matrix for a different type of misspellings like additional or removal or transposal or replaced characters to searching these patterns in terms and predict the correction [10]. Also, phonetic, language, and keyboard models can be useful for correction prediction by decision tree as a machine learning based technique [11, 12]. Another approach can be creating a model based on machine learning techniques to detect typos and predict the correction according to context and domain knowledge [13, 14]. String transduction tries to map one string to another and can be used for misspelled typo corrections too [15]. Also, machine learning is used in character scale to typo detection and corrections, but the recall rate is low (about 30%) [16].

Some other researchers focus on using tree structure for typo correction. It is possible to make a tree based on a probabilistic model of the relationship between characters of words which what characters can become after a particular character and in advance mode, after a sequence of characters. So, these models use Bayes theory to make a prediction model on a tree called Trie and use it for typo correction as the user is typing [17, 18]. The tree structure can be used for grammatical checking and translating too by merging several grammatical trees in a Trie [19]. The simple Trie (without probability) is used for spell checking too [20]. The acyclic deterministic finite automata is a graph with a similar structure which can be used for spell checking and typo correction [21]. There are some methods for query in Trie by wild characters, too [22]. Trie-based index structure can be used for real-time interaction like search recommendation and query completion [23].

The interconnected terms problem was not significant a lot in other contexts, and there is no specific method for correction of interconnected terms. As it was tested, the google translate, and Microsoft office word can detect two parts interconnected terms and suggest a correction for them, but if there are more than two meaningful terms, they cannot detect and suggest any correction. It shows that even huge companies have not been investigated with this problem. So, a divide and conquer algorithm based on the most extended common sequence algorithm have been made, as shown in Fig. 1 to find meaningful terms in an interconnected term. It is a simple brute force algorithm which will consider all combinations of start and end index of a substring in interconnected term to find a meaningful term. Meaningfully checking needs a dictionary. Luckily an excellent trustful dictionary for computer context have been made in past research [24] and can be used for this purpose too.

Checking a word in the dictionary usually is a daily operation, especially in meaningful word detection; so the time complexity of this process is significant. Usually, dictionaries sort their terms to use the binary search with $\log_2$ (N) time complexity for term checking which N is the number of terms in the dictionary. Also, every word needs to be compared with a suspicious meaningful word which complexity of this operation is based on the length of terms even though almost string comparer method uses short circuit idea for time reduction, in other words when they find first different character between two words, and they will cut the comparison operation. So, the meaningful word detection takes the logarithm (N) operation in the worst case to find out the result, and it is in the worst case usually in this procedure because many substrings are meaningless and they are not in dictionaries.

```
Algorithm: Meaningful Word Finding
Input: a connected term with index of 1 to L
Output: a list of meaningful words with start and end
index in connected term
For I in range of 1 to L
   For J in range of I+1 to L
      If substring of term from I to J is in dictionary
         Put the (I, J, substring) in the output
```

Fig. 1. Algorithm of finding meaningful words in an interconnected term

The selected dictionary contains more than 600,000 terms, so it needs 20 comparing ($\log_2$ (600,000)) each time. Also, the above algorithm has two for loop, which takes Combination (n, 2) operations equal to n×(n-1)/2 time complexity. Each iteration needs a dictionary term checking, so the total complexity of this algorithm is in the equation (1) which N is the number of terms in dictionaries, L is the average length of each term and *n* is the length of the interconnected term. Also, this algorithm can be parallel easily by dispatching combinations between some threads or processes, and two threads or processes can be made at least to parallelize this algorithm, which everyone uses half combinations.

$$t(N, L, n)_{Alg1} = log_2^N \times L \times \frac{n\times(n-1)}{2} \qquad (1)$$

Meaningful substring can be everywhere in interconnected term and have overlap, e.g. 'hishe' can be 'hi' and 'she' or 'his' and 'he', so the next step is to find the meaningfulness combination between substrings which have no overlap (e.g. 'his' and 'she' which is not possible according to 's' overlap in primary interconnected term). This algorithm uses a recursive depth-first search approach to find all non-overlap combination shown in Fig. 2. It takes four inputs containing the output list of the previous algorithm, a start index based on the list of meaningful words, a list of the selected index in meaningfulness combination, and the length of the interconnected term. These parameters are the meaningful words, start search index for next combinations, considered in the current path of depth search, and can be considered as a constant in this algorithm respectively. Also, the output of this algorithm is a list of combination too. This final list should be evaluated based on the context of interconnected terms, and the best combination is picked semantically. This algorithm will consider all combinations of meaningful words and choose those combinations with no overlap. So, if there are N words in the meaningful words list, the time complexity of this algorithm equals 2^N, which is exponential, and it is a non-polynomial problem. It takes a long time, and it is not suitable for a real-time situation like correction suggestion as the user is typing in text editors which is very important; because if the user looks the suggestion and correct this typo, it is not necessary to evaluate the result combination semantically by artificial intelligence techniques. It is enough to sort the output list based on a metric and show the top-10 suggestion to the user; then the user will pick the best one. The average length of words can be a useful metric because every much the average length of words be high, the combination contains the largest meaningful component in interconnected terms, and the possibility of meaningfulness is more.

```
Algorithm: Finding meaningfulness combination
between substrings
Input: MW as a list of meaningful words from 1 to n
indexes, SI as start index of searching in meaningful
words with 0 initial value, SC as selected combination
with empty initial value, LCT as length of connected term
Output: a list of meaningfulness combination of words
If start index is 0
    Consider end point equal to 0
Else
    Consider end point equal to end index in SI-th of MW
    If end point equals to LCT
        Return SC
For each index J which start index of J-th of MW equals
to end point
    Consider SCn as new list containing SC plus J as
appended value
    Call this algorithm with MW, J, SCn , LCT ...
        and put the result in output list if it is not empty
```

Fig. 2. Algorithm of finding meaningfulness combination between substrings

III. Proposed Method

In the middle procedure of the process of meaningful word finding, it has been considered that neural networks of the human brain look at a word and predict the next letters based on priors and it seems the human brain uses a tree-like algorithm to find the correctness of a word. So, a binary like a tree proposed to be made for meaningful word checking. This process needs two steps: creating the tree, parsing the tree for checking the existence of a term in the dictionary. Also after making this tree, it was found that this tree can be used to find meaningful words more efficient than brute force algorithm, so in step 3 this tree should be used for finding the meaningful terms. Then these meaningful terms should be checked where which one is much possible in primary interconnected term to be meaningful. Thus, there are four main steps to separate interconnected terms which shown in Fig. 3 which every step will be explained in the next sections with an example.

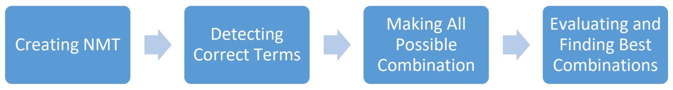
Creating NMT
Detecting Correct Terms
Making All Possible Combination
Evaluating and Finding Best Combinations

Fig. 3. The 4 steps of finding meaningful words in an interconnected term

Suppose that there is a dictionary with these words: 'hello', 'book', 'help', 'his', 'hiss', 'she'. Also, 'hellohelphissbookhishel' is considered as multiple interconnected terms with a typo in the last term. Now the process of neural-like tree making will be explained for matching the input terms which this tree has called a neural matching tree (NMT).

1.1 Neural Matching Tree Creation Process

This tree is like the binary tree, but it has more than two children, so it is an n-ary like a tree. It has a root, and every word in the dictionary will have appeared as a path below of the root. Every letter in words will be put in a node in the tree. Also, every node will contain a flag for showing the end of the word, and if a node contains a letter which is the end of a word, the flag will be true; otherwise it will be false. Every node can be implemented by a map or dictionary data structure in programming languages. So, for the supposed example, this tree will be like the Fig. 4. In this tree, the flag of end letter of every word is T (true) and has different color.

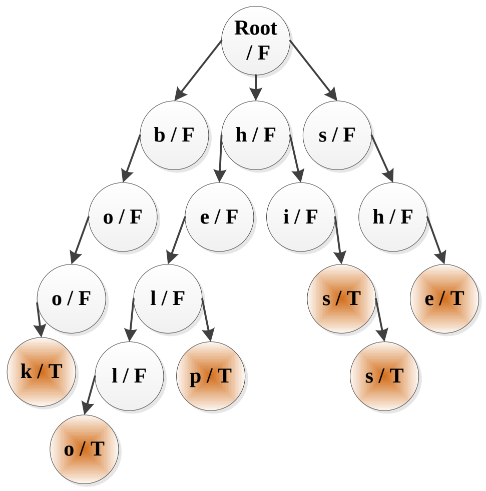

Fig. 4. Neural Match Tree example for supposed dictionary

It is interesting that a path can have multiple final nodes, for example, both words 'his' and 'hiss' have the same prefix and in this tree, have the same path. It should be mentioned that this neural matching tree, can compress the dictionary too. The human mind is like this tree as to when we look at a word, some path in our brain will be activated and we can predict next letters. The procedure of creating NMT is explained in Fig. 5. This algorithm takes a list of words of a dictionary and returns the root node for NMT. This procedure is very simple and for every word, parse the tree once time from the root node and check the child nodes to have the letters of a word, otherwise create a child node for each letter as shown in Fig. 3. Also, for the last node, which contains the last letter of a word, the flag must be true to show the end of a word.

```
Algorithm: Creating Neural Match Tree
Input: a list of words in a dictionary
Output: a root node to a neural match tree
Create a Root node containing no letter with false flag
as end character
For each word in words list of dictionary
        Consider n as a node pointed to Root node
        For each letter in word
                If letter is not in child node of n
                        Create a child node for n with
letter and false flag
     Consider child node containing the letter as new n
  Put true flag for node n (the last one)
```

Fig. 5. Algorithm of creating neural match tree

Time and memory complexity of NMT are essential too. As explained before, the NMT memory is less than a simple dictionary, and it can be used for compression too. Also, the creation procedure of NMT need a parse on the whole dictionary just one time, so it depends on the number of words in the dictionary and the length of each word. If every word has an average length of L and there are N words in the dictionary (as denoted before), the time complexity will be N×L.

1.2 Neural Matching Tree Using Process

The next step is to use the NMT for checking a new term is valid or not; in other words, is it in the dictionary or not. This procedure is like the NMT creation process, which had described in Fig. 6. It will check every letter of the suspicious term is in NMT or not. If the first letter is in child nodes of the root node, then the next letter will check with the child node of that selected child node. Unlike the checking process in regular dictionaries, which has $Log_2(N)\times L$ time complexity, this process has L time complexity.

1.3 Finding Meaningful Words in an Interconnected Term by Neural Matching Term

Now, it is time to do the main procedure instead of the brute force algorithm in Fig. 1. The first loop step of brute force algorithm cannot be connivance because there may be some lexical mistake in interconnected term and some substring is useless, so every substring maybe meaningful.

```
Algorithm: Checking the existence of word in NMT as a
dictionary
Input: a Root node for NMT and a term for checking
Output: a Boolean value indicating the existence of the ...
        word in the NMT
Consider n as a node pointed to Root node
For each letter in word
  If letter is not in child node of n
    Return False
  Consider child node containing the letter as new n
Return True
```

Fig. 6. Algorithm of checking the existence of a word in NMT

The second loop in the brute force algorithm can be more intelligence based on NMT by cutting the searching existence of substring as soon as finding a letter is that substring is not in the next node of NMT. The algorithm of finding meaningful words in interconnected term by NMT is shown in Fig. 7. The time complexity of searching in NMT is less than regular dictionaries. So, the time complexity of this algorithm will be L×L.

```
Algorithm: Finding Meaningful Words of a Connected
Term by NMT
Input: Root of NMT and a Connected Term with index of 1
to L
Output: the list of meaningful substring of connected term
An empty list for connected terms as output
For I in range of 1 to L
  J = I
  Consider n as a node pointed to Root node
  While J <= L
    If n has a final flag (is the last letter of a word)
      Put substring of connected term from I to J ...
                     index into output list
    If letter is not in child node of n
      Break out from while loop
  Consider child node containing the letter as new n
```

Fig. 7. Algorithm of finding meaningful substrings in an interconnected term by NMT

1.4 Finding meaningfulness combination

As mentioned before, the recursive algorithm of finding meaningfulness combinations has high time complexity, so, a new iteration based algorithm has created for this purpose, as shown in Fig. 8. There is a new input in this algorithm to limit the search based on average word length metric as mentioned and select just top combination with most ranks. This algorithm needs a priority queue which can be implemented by heap algorithm as heap queue to contain every combination with its rank. Every time this algorithm chooses the highest rank combination. If this combination is completed and considered all non-overlap words, the combination will be added to the output list. Otherwise, it will be progressed to choose the next word and add to its combination, and calculate the rank again based on the new combination. This algorithm sort the non-overlap combination by the length of them descending and then choose combination as noted in the algorithm by underline. It causes the longest combination of chose. The time complexity of this algorithm depends on several words in a combination and parameter N which in the worst case if every letter of the interconnected term has considered as a meaningful word, the time complexity of this algorithm will be LCT×N and it is polynomial.

```
Algorithm: Finding meaningfulness combination between
substrings by NMT
Input: MW as a list of meaningful words from 1 to n
indexes, LCT as length of connected term, N as number of
top most meaningfulness combination
Output: a list of meaningfulness combination of words
Consider LS as a list of search states …
    with a combination containing 0 as index of selected …
    words with 0 rank
While LS is not empty and has not N output
    Pop highest rank combination in LS as HRC
    Choose last index of combination in HRC as end point
    For each index J which start index of J-th of MW …
     equals to end point Sorted by J descending
        Consider HRCn as new combination containing …
            HRC combination plus J as appended value …
            and calculate rank based on HRCn
              If the end point of J-th of MW equals to LCT
                  Put the combination of HRCn in output list
              Else
                  Put HRCn in LS
```

Fig. 8. Algorithm of finding meaningfulness combination between substrings

IV. Evaluation Method

The new scientific semi dictionary as word list and unique typo dataset of bug reports have picked for evaluating proposed algorithms [4]. The implementation of proposed algorithms done in Python 3.6 programming language and a Core i5 1.8 GHz computer with 12GB memory having windows 8.1 x64. In the first step, the dataset has analyzed, and it was denoted that it has 391,807 suspicious typos, but it was detected that 149,749 typos are numeric values which are hexadecimal or have semi-hexadecimal form. Also, some of them was newly devised words, and some others have another type of typos. So the terms with length of more than five characters have been choosing for evaluating proposed algorithms which contain 182,402 terms. The evaluation designed to test algorithms based on selected typos of a dataset containing interconnected terms and considering the scientific semi dictionary. Also, two algorithms considered for finding meaningfulness combination between substrings by NMT based on [24] as Algorithm 1 (Alg.1) and new proposed algorithm Fig. 8. As Algorithm 2 (Alg.2). The results of the number of detected terms based on the average word length per character of detected terms are shown in Fig. 9 using a logarithmic scale in base two.

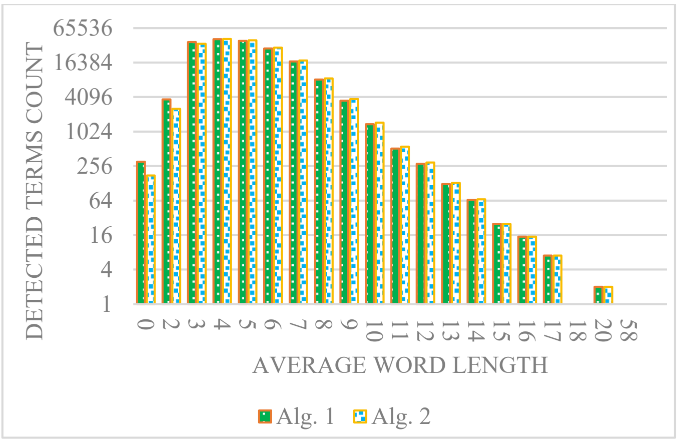

Fig. 9. Average Word Length of Detected Terms versus Alg.1 [24]

The detail of Fig.9 is tabulated in Table 1. As it is evident in this table, the number of detected terms with lower length is less than the higher length, and it is because the new proposed algorithm chooses the longest combination which is more relevant based on our observations.

TABLE I. Average Word Length of Detected Terms

Average Word Length	Number of Detected Terms	
	Algorithm 1	Algorithm 2
0	304	175
2	3,727	2,550
3	37,348	34,697
4	41,815	42,232
5	39,003	40,063
6	28,851	29,809
7	17,140	17,763
8	8,255	8,646
9	3,548	3,776
10	1,371	1,469
11	516	558
12	280	295
13	124	130
14	66	67
15	25	25
16	15	15
17	7	7
18	1	1
20	2	2
58	1	1

The speed of both algorithms was almost the same. The algorithm 1 and proposed algorithm are tested for ‘hellol654shelphissbookhishel’ too. The return value of algorithm 1 was ‘hel lol 6 54 shel phis sbo ok hishe l’ but the return value of proposed algorithm is ‘hello l654s help hiss book hishe l’ and the seventh answer of the proposed algorithm is ‘hello l654s help hiss book his hel’ which is the proper answer.

V. Conclusion

This research focuses on the correction of interconnected terms typos by natural language processing based on a reliable word list like a formal dictionary to build an n-ary tree inspired from human neural network to recall the memory. This tree was called Neural Matching Tree (NMT), which is created based on the word list. Then the interconnected term will be parsed based on NMT, and the meaningful substrings in interconnected term will be extracted. Then non-overlap combinations of meaningful substrings had picked as correction output words. The proposed algorithms are elementary, and their time complexity is negligible. Another achievement of this research is showing that there are many interconnected terms in the software context, especially bug reports. So the correction of interconnected typos can be useful for other goals like duplicate bug report detection which use information retrieval techniques like term frequency that use the lexical form of words and depends on having non-typo in bug reports.

In the next step, any improvements can be used for meaningfulness combination extraction process to achieve the best one between other combinations and also based on the main context. Also, other metrics can be introduced for this purpose instead of the average length of words, which this research had introduced and had used.

References

[1] B. Soleimani Neysiani and S. M. Babamir, "Methods of Feature Extraction for Detecting the Duplicate Bug Reports in Software Triage Systems," presented at the International Conference on Information Technology, Communications and Telecommunications (IRICT), Tehran, Iran, 2016, 2016. Available: http://www.sid.ir/En/Seminar/ViewPaper.aspx?ID=7677

[2] B. Soleimani Neysiani and S. M. Babamir, "Improving Performance of Automatic Duplicate Bug Reports Detection Using Longest Common Sequence," in IEEE 5th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, 2019, vol. 5.

[3] B. Soleimani Neysiani and S. M. Babamir, "New Methodology of Contextual Features Usage in Duplicate Bug Reports Detection," in IEEE 5th International Conference on Web Research (ICWR), Tehran, Iran, 2019, vol. 5.

[4] B. Soleimani Neysiani and S. M. Babamir, "Automatic Typos Detection in Bug Reports," presented at the IEEE 12th International Conference Application of Information and Communication Technologies, Kazakhstan, 2018.

[5] A. Alipour, A. Hindle, T. Rutgers, R. Dawson, F. Timbers, and K. Aggarwal. (2013). Bug Reports Dataset. Available: https://github.com/kaggarwal/Dedup

[6] L. Zhuang, F. Jing, and X.-Y. Zhu, "Movie review mining and summarization," in Proceedings of the 15th ACM international conference on Information and knowledge management, 2006, pp. 43-50: ACM.

[7] K. Kukich, "Techniques for automatically correcting words in text," Acm Computing Surveys (CSUR), vol. 24, no. 4, pp. 377-439, 1992.

[8] K. H. Lai, M. Topaz, F. R. Goss, and L. Zhou, "Automated misspelling detection and correction in clinical free-text records," Journal of biomedical informatics, vol. 55, pp. 188-195, 2015.

[9] Q. Chen, M. Li, and M. Zhou, "Improving query spelling correction using web search results," in Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 2007.

[10] H. M. Noaman, S. S. Sarhan, and M. Rashwan, "Automatic Arabic spelling errors detection and correction based on confusion matrix-noisy channel hybrid system," Egypt Comput Sci J, vol. 40, no. 2, p. 2016, 2016.

[11] G. A. d. M. Almeida, "Using phonetic knowledge in tools and resources for Natural Language Processing and Pronunciation Evaluation," Master, Universidade de São Paulo, 2016.

[12] G. A. de Mendonça Almeida, L. Avanço, M. S. Duran, E. R. Fonseca, M. d. G. V. Nunes, and S. M. Aluísio, "Evaluating phonetic spellers for user-generated content in brazilian portuguese," in International Conference on Computational Processing of the Portuguese Language, 2016, pp. 361-373: Springer.

[13] Y. Huang, Y. L. Murphey, and Y. Ge, "Intelligent typo correction for text mining through machine learning," International Journal of Knowledge Engineering and Data Mining, vol. 3, no. 2, pp. 115-142, 2015.

[14] Y. Huang, Y. L. Murphey, and Y. Ge, "Automotive diagnosis typo correction using domain knowledge and machine learning," in IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2013, pp. 267-274: IEEE.

[15] J. Ribeiro, S. Narayan, S. B. Cohen, and X. Carreras, "Local String Transduction as Sequence Labeling," in Proceedings of the 27th International Conference on Computational Linguistics, 2018, pp. 1360-1371.

[16] M. Korpusik, Z. Collins, and J. Glass, "Character-based embedding models and reranking strategies for understanding natural language meal descriptions," Proc. Interspeech, pp. 3320-3324, 2017.

[17] H. Duan and B.-J. P. Hsu, "Online spelling correction for query completion," in Proceedings of the 20th international conference on World wide web, 2011, pp. 117-126: ACM.

[18] B.-J. Hsu, K. Wang, and H. Duan, "Online spelling correction/phrase completion system," ed: Google Patents, 2012.

[19] K. Oflazer, "Error-tolerant tree matching," in Proceedings of the 16th conference on Computational linguistics-Volume 2, 1996, pp. 860-864: Association for Computational Linguistics.

[20] H. Shang and T. Merrettal, "Tries for approximate string matching," IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 4, pp. 540-547, 1996.

[21] S. Deorowicz and M. G. Ciura, "Correcting spelling errors by modeling their causes," International journal of applied mathematics and computer science, vol. 15, pp. 275-285, 2005.

[22] N. Ito, "Character-string retrieval system and method," ed: Google Patents, 1997.

[23] P. Fafalios and Y. Tzitzikas, "Type-Ahead Exploratory Search through Typo and Word Order Tolerant Autocompletion," J. Web Eng., vol. 14, no. 1&2, pp. 80-116, 2015.

[24] B. Soleimani Neysiani and S. M. Babamir, "Automatic Interconnected Lexical Typo Correction in Bug Reports of Software Triage Systems," presented at the International Conference on Contemporary Issues in Data Science, Zanjan, Iran, 2019.

[25] Napoli, C., Tramontana, E., Sciuto, G. L., Wozniak, M., Damaevicius, R., & Borowik, G. (2015, July). Authorship semantical identification using holomorphic Chebyshev projectors. In 2015 Asia-Pacific Conference on Computer Aided System Engineering (pp. 232-237). IEEE.

[26] Venckauskas, A., Karpavicius, A., Damaševičius, R., Marcinkevičius, R., Kapočiūte-Dzikienė, J., & Napoli, C. (2017, September). Open class authorship attribution of lithuanian internet comments using one-class classifier. In 2017 Federated Conference on Computer Science and Information Systems (FedCSIS) (pp. 373-382). IEEE.