A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP

Akila D.
Saveetha College of Liberal Arts and Sciences
103 PUBLICATIONS **264** CITATIONS

A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP

Krishna Prakash Kalyanathaya

Research Scholar, Department of Computer Science, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai, Tamilnadu.

E-mail: krishna.prakash.kk@gmail.com

Dr.D. Akila

Associate Professor, Department of Information Technology, School of Computing Sciences, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai, Tamilnadu.

E-mail: akiindia@yahoo.com

Dr.G. Suseendren

Assistant Professor, Department of Information Technology, School of Computing Sciences, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai, Tamilnadu.

E-mail: suseendar_1234@yahoo.co.in

Abstract

Approximate string matching has many applications in Natural Language Processing. This paper provides a comparison of various algorithms for approximate string matching. Most of the algorithms are based on the edit distance between characters in the two strings. It also covers the challenges in using these algorithms for the purpose of text retrieval. The authors propose an alternative approach for approximate string matching which are better suited for text retrieval. In this study we are comparing two strings to identify similarities using a matrix. The matrix will be updated for each overlap character between two strings. An overlap counter is maintained to increment value for each overlap character position and reset position to 0 when no overlap position is encountered. The maximum counter value is then used in a ratio to calculate the degree of similarity. The algorithm implemented using Python language. The results indicate the proposed approach can be used for identifying lexically similar words. This type of approach will find it use in lemmatization, text summarization, topic modelling and data mining solutions.

Keywords: Natural Language Processing, Data mining, Semantic Similarity, WordNet.

1. Introduction

Approximate string matching is a method of finding a string that closely matches with the given string if there is no exact match [1][7]. It is method of comparing two strings and estimating a measure (metrics) of match between two strings [11]. The basis for all problems involving string comparisons is the classical pattern matching problem wherein one searches for a pattern p in a given text[8][10] Some of the application areas of approximate string matching are Image processing, word stemming, genetic algorithms, knowledge discovery, information retrieval, and pattern recognition[1][8].

This paper will cover a comparison of commonly available fuzzy based approximate string matching algorithms for the purpose of text retrieval. It also covers the challenges in using these algorithms for the purpose of identifying lexically similar words. In this context, lexical similarity is the degree of similarity between two words.

Received: 03 Mar 2019/Accepted: 04 Apr 2019

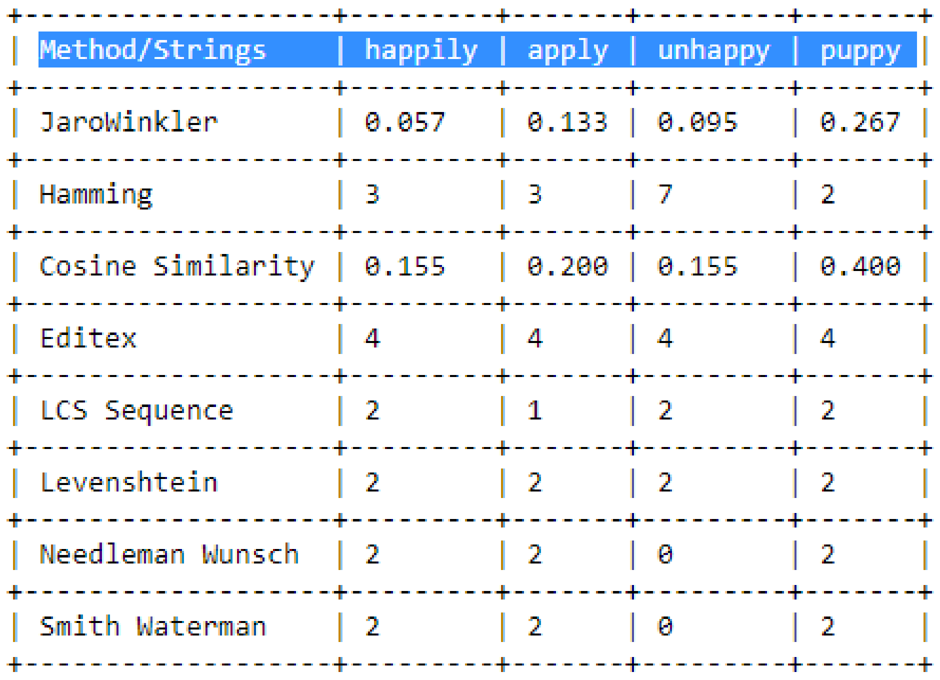
The author investigates various alternative methods to identify matching ratio for best approximate string using fuzzy matching methods that can be utilized in lexical analysis.Further, the author wants to use this approach to demonstrate an alternative approach to stemming.

2. The Issues

For the purpose of the study, the author has compared the results of implementation of 8 popular methods. The implementations are taken from the Python language package text distance [3].

Comparison of existing approximate string matching approaches

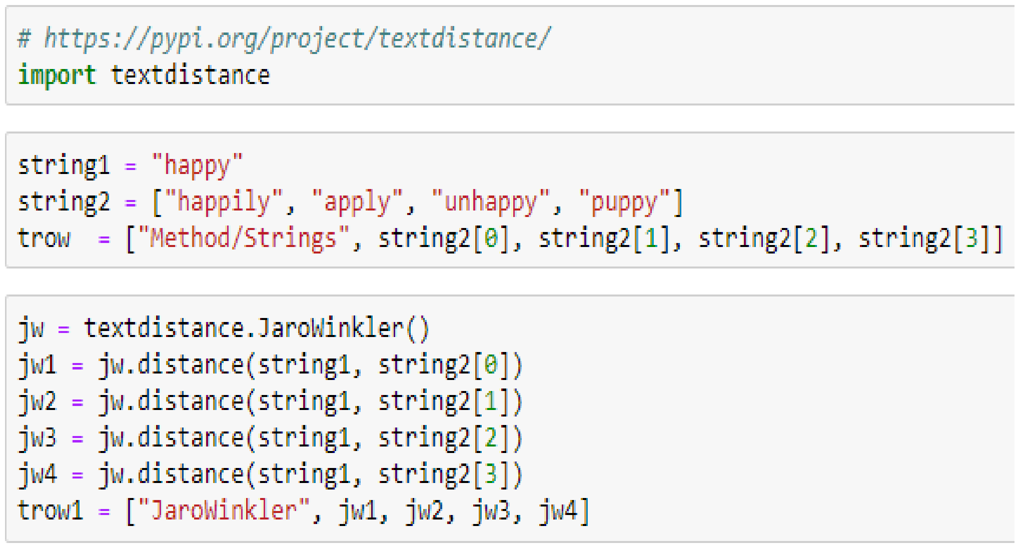
Table 1: Execution results of approximate string matching for the word “happy”. Python package text distance is used for execution



```
+-------------------+---------+-------+---------+-------+
| Method/Strings    | happily | apply | unhappy | puppy |
+-------------------+---------+-------+---------+-------+
| JaroWinkler       | 0.057   | 0.133 | 0.095   | 0.267 |
+-------------------+---------+-------+---------+-------+
| Hamming           | 3       | 3     | 7       | 2     |
+-------------------+---------+-------+---------+-------+
| Cosine Similarity | 0.155   | 0.200 | 0.155   | 0.400 |
+-------------------+---------+-------+---------+-------+
| Editex            | 4       | 4     | 4       | 4     |
+-------------------+---------+-------+---------+-------+
| LCS Sequence      | 2       | 1     | 2       | 2     |
+-------------------+---------+-------+---------+-------+
| Levenshtein       | 2       | 2     | 2       | 2     |
+-------------------+---------+-------+---------+-------+
| Needleman Wunsch  | 2       | 2     | 0       | 2     |
+-------------------+---------+-------+---------+-------+
| Smith Waterman    | 2       | 2     | 0       | 2     |
+-------------------+---------+-------+---------+-------+
```

Sample execution of text distance package for generating the data in TABLE 1.

Table 2: Sample execution of text distance package for generating the data



```python
# https://pypi.org/project/textdistance/
import textdistance
```

```python
string1 = "happy"
string2 = ["happily", "apply", "unhappy", "puppy"]
trow  = ["Method/Strings", string2[0], string2[1], string2[2], string2[3]]
```

```python
jw = textdistance.JaroWinkler()
jw1 = jw.distance(string1, string2[0])
jw2 = jw.distance(string1, string2[1])
jw3 = jw.distance(string1, string2[2])
jw4 = jw.distance(string1, string2[3])
trow1 = ["JaroWinkler", jw1, jw2, jw3, jw4]
```

Received: 03 Mar 2019/Accepted: 04 Apr 2019

Challenge posed by the issue

The results of evaluation shows that the current approaches to approximate string matching faced a challenge in identifying lexically similar words as compared to lexically different words. This is due to the fact that most of the algorithms are based on the edit distance between characters in the two strings[1].

In the above example, the strings “apply” and “happy” are lexically dissimilar words. Again, the strings “happy” and “unhappy” are lexically very close match. But the result from the experiments does not conclude the same. The evaluations of the algorithms are as follows:

In Jaro Winkler algorithm: “happy” and “happily” scored closely matching. The word “apply” and “happy” are lexically dissimilar words but the resulting score of 0.133 is not satisfactory to show the dissimilarity.

In Hamming distance: “apply” and “happily” scored same though they are lexically dissimilar.In consine similarity, the word “apply” and “happy” are lexically dissimilar words but the resulting score of 0.2 is not satisfactory to show the dissimilarity.

All other algorithms also gave similar results that “apply” and “happily” scored same though they are lexically dissimilar. The two strings “apply” and “happily” both are matching equally closer to the string “happy” is an issue and has serious impact on natural language processing (NLP) in general and lexical analysis and semantic analysis and summarization in particular. As NLP is used in several areas of data analytics, there is need to research on this issue mentioned above.

Currently lexical analysis and semantic analysis are performed by complex tools such as WordNet, Genesis and many other tools are being developed to make lexical analysis and summarization to achieve the better accuracy.

3. Literature Review

The following section summarises the text distance metrics used in the algorithms referred in this paper.

- Jaro Winkler distance[1]: This method measures metrics which is a function of number of matching characters and number of transpositions between the strings. Lower the value the more similar strings are. 0 means full match and 1 means no match.
- Hamming distance[1]: This method measures the distance function which is a minimum number of substitutions required to make both strings equal. Lower the value the more similar strings are.
- Cosine similarity[1][4]: Measures similarity between two non-zero vectors by using cosine of the angle between them. Vectors are similar if they are parallel and cosine value is 1 (Cos 0 = 1) and dissimilar if they are perpendicular and cosine value is 0 (Cos 90 = 0). So the value of measure lies between 0 and 1 determines the match.
- Editexalgorithm[9]: Phonetic matching distance is used here to identify the strings that are of identical pronunciations. Distance measure 0 means match and non-zero is degree of no match.
- Longest Common Sub Sequence distance (LCS Sequence)[1]: It measures distance as a function of number of unpaired characters that can be formed from the two strings. Lower the value the more similar strings are.
- Levenshtein distance[1]: This method measures the edit distance between two strings[1] to identify how different is one string to the other. The edit distance between two strings S1 and S2 is the minimum number of edit operations required to transform S1 to S2. The edit operations include insertions and deletions[8].
- Needleman Wunsch[5]: This method uses a scoring system giving a score with 1 for each match, -1 for each mismatch and -1 for deletionfor gaps) and calculate the edit distance. Higher the edit distance indicates more similarity.

Received: 03 Mar 2019/Accepted: 04 Apr 2019
1553-9105 / Copyright © 2019 Binary Information Press

- Smith Waterman[6]: This algorithm performs local sequence alignment to determine similar regions between two strings. A scoring system is used to measure the local sequence alignment. Higher the score the better is the alignment with two strings.

Essentially, all of the above methods follow one common which is basically identifying measure of how much one string differ from the other. None of these methods lay importance on the lexical or semantic property of the strings. As a result, these methods may not fully fit for solving NLP problems[12].Hence, we look at some alternative approaches to retrieve lexically or semantically matching strings for the given string. This type of approach will find it use in lemmatization, text summarization, topic modelling and data mining solutions.

4. Empirical study of the New Approach

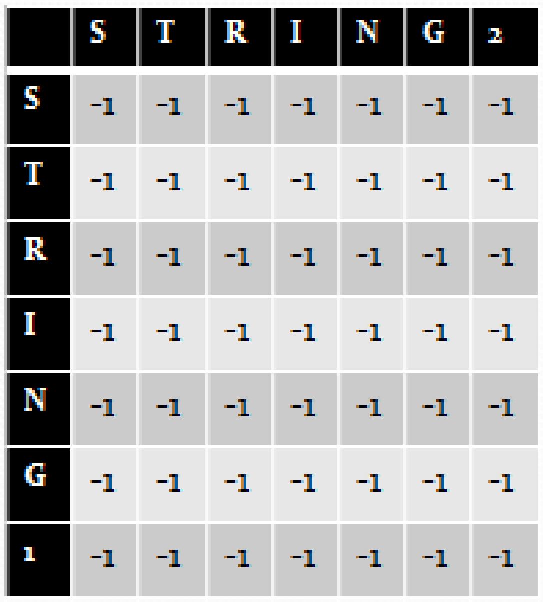
In this study we are comparing two strings to identify similarities using a matrix. The matrix will be updated for each overlap character position between the strings. Further, we apply row wise comparison of each character position. Anoverlap counter is maintained to increment value for each overlap character position and reset position to 0 when no overlap position is encountered. This process is repeated for each row and terminated when no overlaps found in a single row. Finally, max value in the matrix is taken in a formula to calculate the ratio as follows:

$$\text{Ratio} = \frac{2 * \text{Max}(A)}{\text{Len}(S1) + \text{Len}(S2)}$$

Ratio is the measure of degree of similarity between two strings with a value between 0 and 1. The two strings completely overlap (100 % similar) if value is 1. The two strings have no overlap (or 0% similar) if value is 0.

S1 is the base form of the string or a shorter word between s1 and S2.S2 is the word to which degree of similarity to be determined. A is the matrix representing the state of similarity between s1 and s2.

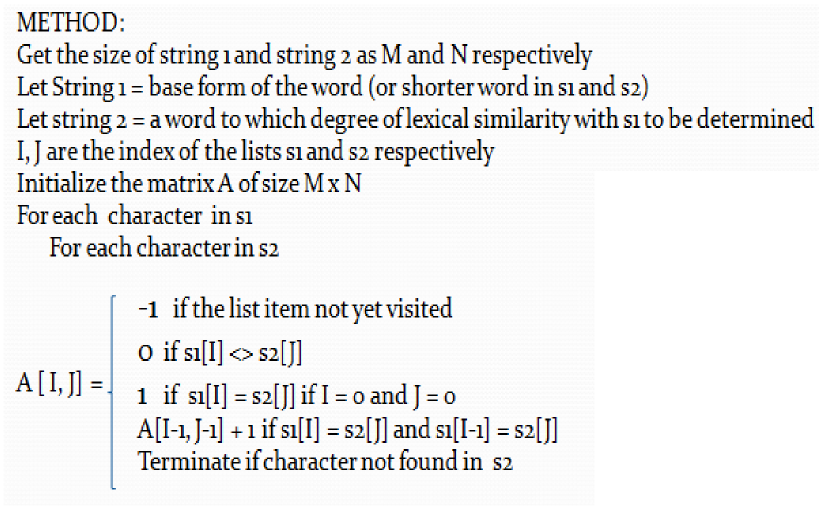
Len is the length function to return the length of the string. Max is matrix function to return maximum value in the matrix



	S	T	R	I	N	G	2
S	-1	-1	-1	-1	-1	-1	-1
T	-1	-1	-1	-1	-1	-1	-1
R	-1	-1	-1	-1	-1	-1	-1
I	-1	-1	-1	-1	-1	-1	-1
N	-1	-1	-1	-1	-1	-1	-1
G	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1

Initialization of matrix A with MxN size

Received: 03 Mar 2019/Accepted: 04 Apr 2019

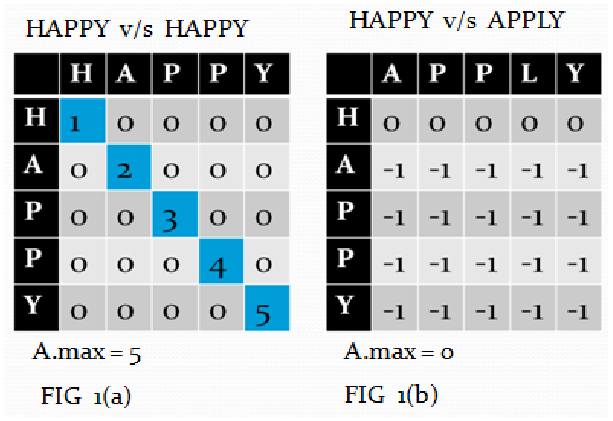


```
METHOD:
Get the size of string 1 and string 2 as M and N respectively
Let String 1 = base form of the word (or shorter word in s1 and s2)
Let string 2 = a word to which degree of lexical similarity with s1 to be determined
I, J are the index of the lists s1 and s2 respectively
Initialize the matrix A of size M x N
For each  character  in s1
    For each character in s2

                -1  if the list item not yet visited
                0  if s1[I] <> s2[J]
    A [ I, J] = 1  if  s1[I] = s2[J] if I = 0 and J = 0
                A[I-1, J-1] + 1 if s1[I] = s2[J] and s1[I-1] = s2[J]
                Terminate if character not found in  s2
```

Illustration

The following two examples illustrate the approach used in this analysis:



HAPPY v/s HAPPY

	H	A	P	P	Y
H	1	0	0	0	0
A	0	2	0	0	0
P	0	0	3	0	0
P	0	0	0	4	0
Y	0	0	0	0	5

A.max = 5

FIG 1(a)

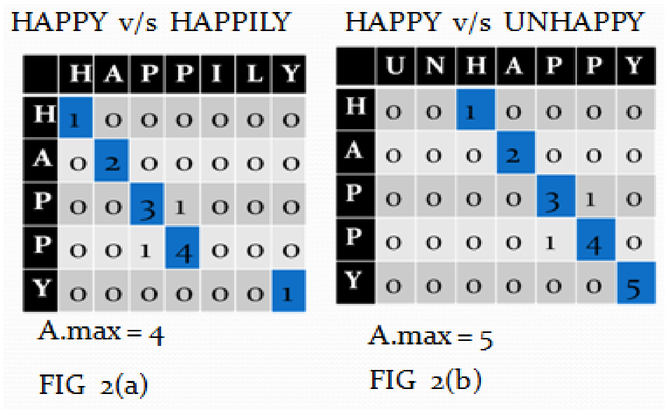
HAPPY v/s APPLY

	A	P	P	L	Y
H	0	0	0	0	0
A	-1	-1	-1	-1	-1
P	-1	-1	-1	-1	-1
P	-1	-1	-1	-1	-1
Y	-1	-1	-1	-1	-1

A.max = 0

FIG 1(b)

Fig 1: An illustration of comparison of two strings
Ratio calculation For 1(a), is as follows:
((2 * 5))/(5+5) = 10/10 = 1
Ratio calculation For 1(b), is as follows:
((2 * 0))/(5+5) = 0/10 = 0



HAPPY v/s HAPPILY

	H	A	P	P	I	L	Y
H	1	0	0	0	0	0	0
A	0	2	0	0	0	0	0
P	0	0	3	1	0	0	0
P	0	0	1	4	0	0	0
Y	0	0	0	0	0	0	1

A.max = 4

FIG 2(a)

HAPPY v/s UNHAPPY

	U	N	H	A	P	P	Y
H	0	0	1	0	0	0	0
A	0	0	0	2	0	0	0
P	0	0	0	0	3	1	0
P	0	0	0	0	1	4	0
Y	0	0	0	0	0	0	5

A.max = 5

FIG 2(b)

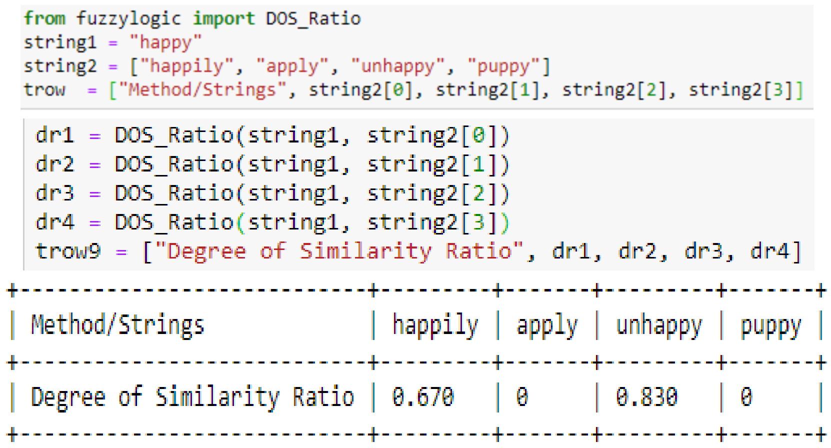
Fig 2: An illustration of comparison of two strings “happily” with “unhappy”
Ratio calculation For 2(a), is as follows:
((2 * 4))/(5+7) = 8/12 = 0.67

Received: 03 Mar 2019/Accepted: 04 Apr 2019

Ratio calculation For 2(b), is as follows:
((2 * 5))/(5+7) = 10/12 = 0.83

Summary of implementation in Python

Table 2: Execution results of proposed approach



```
from fuzzylogic import DOS_Ratio
string1 = "happy"
string2 = ["happily", "apply", "unhappy", "puppy"]
trow  = ["Method/Strings", string2[0], string2[1], string2[2], string2[3]]
```

```
dr1 = DOS_Ratio(string1, string2[0])
dr2 = DOS_Ratio(string1, string2[1])
dr3 = DOS_Ratio(string1, string2[2])
dr4 = DOS_Ratio(string1, string2[3])
trow9 = ["Degree of Similarity Ratio", dr1, dr2, dr3, dr4]
```

```
+----------------------------+---------+-------+---------+-------+
| Method/Strings             | happily | apply | unhappy | puppy |
+----------------------------+---------+-------+---------+-------+
| Degree of Similarity Ratio | 0.670   | 0     | 0.830   | 0     |
+----------------------------+---------+-------+---------+-------+
```

The above findings shown in the execution results (TABLE 2) obtained for proposed solution as follows:

- “happy” and “happily” will match 67%
- “happy” and “apply” will match 0%
- “happy” and “unhappy” will match 83%
- “happy” and “puppy” will match 0%

Thus, it is evident from the results that “happily” and “happy” are close match and “happily” and “apply” does not match lexically.

5. Recommendations

The above findings shows that proposed approach for approximate string matching works better than existing algorithms for extracting lexically similar words.

6. Conclusion

The results of evaluation shows that the current approaches to approximate string matching faced a challenge in identifying lexically similar words. In the proposed approach we have used new metric to measure the degree of similarity. This metric is called as degree of similarities ratio whose value lies between 0 and 1 indicating the degree of match.

References

[1] Navarro, Gonzalo. (2000). A Guided Tour to Approximate String Matching. ACM Computing Surveys. 33. 10.1145/375360.375365. p36-38
[2] Baeza-Yates, Ricardo & Navarro, Gonzalo. (1999). Fast Approximate String Matching in a Dictionary. 10.1109/SPIRE.1998.712978. p2-3
[3] textdistance 4.0.0 https://pypi.org/project/textdistance/
[4] Cosine similarity https://en.wikipedia.org/wiki/Cosine_similarity
[5] Needleman–Wunsch algorithm https://en.wikipedia.org /wiki/ Needleman% E2%80%93 Wunsch_ algorithm
[6] Smith–Waterman algorithm https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm
[7] Nattachai Watcharapinchai, Sitapa Rujikietgumjorn. "Approximate license plate string matching for vehicle re-identification", 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance(AVSS), 2017 p3-4

Received: 03 Mar 2019/Accepted: 04 Apr 2019

[8] Sreenivas Gollapudi, Rina Panigrahy. "A dictionary for approximate string search and longest prefix search", Proceedings of the 15th ACM international conference on Information and knowledge management - CIKM '06, 2006 p768-770.
[9] Recchia, Gabriel & Louwerse, Max. (2013). A Comparison of String Similarity Measures for Toponym Matching. COMP 2013 - ACM SIGSPATIAL International Workshop on Computational Models of Place. 54-61. 10.1145/2534848.2534850. p3.
[10] Dr.D.Akila,S.Sathya,Dr.G.Suseendran,"Survey on Query Expansion Techniques in Word Net Application", Journal of Advanced Research in Dynamical and Control Systems, Vol.10(4), May, 2018 (UGC Approved Journal No. 26301)
[11] Dr.D.Akila, Dr.C.Jayakumar, "Acquiring Evolving Semantic Relationships for WordNet to Enhance Information Retrieval", International Journal of Engineering and Technology, Volume 6, November 5,Pages 2115-2128,2014
[12] G. Suseendran, E. Chandrasekaran and Anand Nayyar ,"Defending Jellyfish Attack in Mobile Ad hoc Networks via Novel Fuzzy System Rule G.," Data Management, Analytics and Innovation, Advances in Intelligent Systems and Computing, vol. 839, pp.437-455, 2019.

Received: 03 Mar 2019/Accepted: 04 Apr 2019
1553-9105 / Copyright © 2019 Binary Information Press