Sentence Similarity System

Darshan Kadam[1], Nayan Joshi[2], Sahil Kadu[3], Shubhangi Chavan[4]

[1-4]*Department of Information Technology, Pillai College of Engineering, New Panvel, India*

---***---

Abstract - *The measure of how similar the given sentences are can be called as Sentence similarity, which plays an important role in text-related research and application in area such as text-mining. In this system we Pre-process the given sentences in a bag of words using Tokenization, Stemming and other Natural language techniques. Then we apply syntax similarity techniques and semantics similarity techniques. The syntax similarity technique finds the grammatical syntax similarity between sentences. The Semantic similarity technique finds the Semantic similarity between words, it creates a relationship between words and sentences through the meanings of the words. The technique used to calculate Syntactic similarity are Cosine similarity, Word order similarity and Jaccard similarity. In Cosine similarity we find the cosine similarities between sentences, in Word order similarity we find the Intersection word set of them which contains common words between the sentence. The sentence similarity is used in Plagiarism detection system, Question-Answering system, etc.*

Key Words: **Semantic similarity, syntactic similarity, WordNet, Hindi similarity, Text similarity**

1. INTRODUCTION

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. Most NLP techniques rely on machine learning to derive meaning from human languages. NLP is commonly used in many applications such as.

- Language translation applications such as Google Translate
- Personal assistant applications such as Google Assistant, Siri, Cortana, and Alexa.
- And many more...

NLP entails applying algorithms to identify and extract the natural language rules such that the unstructured language data is converted into a form that computers can understand. When the text has been provided, the computer will utilize algorithms to extract meaning associated with every sentence and collect the essential data from them. Sometimes, the computer may fail to understand the meaning of a sentence well, leading to obscure results. In current times the Sentence Similarity measures are used more and needed in the Text-based Research and other areas. Some similarity measure calculates the similarity between 2 sentences, thoroughly using Word-net semantic dictionary. The Sentence Similarity is the one of the core functions in NLP tasks such as Paraphrase detection, etc. Given the two sentences, the task of calculating the similarity is defined how similar is the meaning of two sentences. The higher the similarity, the more similar the meaning of two sentences are.

2. PROPOSED MODEL

2.1 Overview

This section presents an Overview and Description of techniques used for the system. The Semantic similarity is calculated using Word-net as the corpus.

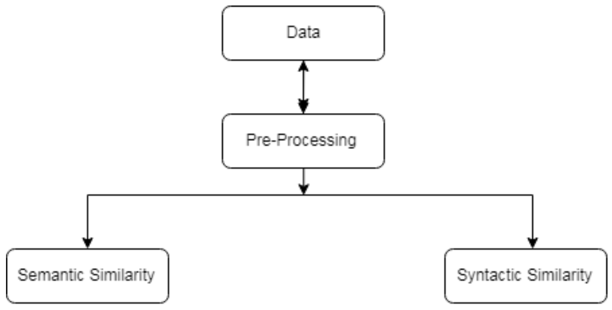


Fig -1: Overview of System

This system checks the similarity between sentences, giving a similarity output which presents how much percent of the sentences are similar. This System can be used to detect plagiarized documents.

Processing Steps :

1. **Tokenization:** A given Sentence is divided into array/list of separate words called Tokens.
2. **Lemmatization:** It brings the word to its base form using the proper vocabulary.
3. **Stop-Word Removal:** The removal of the Stopwords (such as "is","the".etc) is called Stop Word Removal.

4. **Semantic Similarity:** Similarity returns a score denoting how similar two word or sentence senses are, based on some measure that connects the senses in is-a taxonomy.

5. **Syntactic Similarity:** Syntax similarity is a measure of the degree to which the word sets of two given sentences are similar on the basis of their syntax.

2.2 Existing System

The presented System Architecture is using the syntactic approach and semantic with only one syntactic technique:

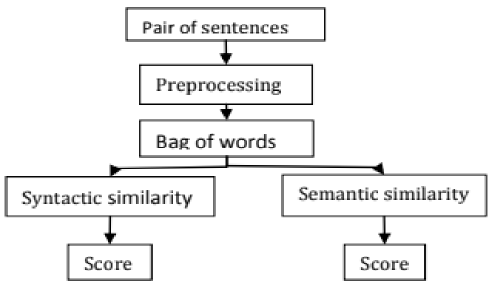


Fig -2: Existing System[1]

This approach uses only both Syntactic and Semantic way of the similarity measure. In the Semantic approach the techniques used:

1. Wu-Palmer similarity
2. Feature based measure
3. Short Path distance

The Syntactic approach only uses a single technique for measurement between text:

1. Cosine Similarity

The system checks the only uses one similarity technique for syntactic level of similarity calculation. This makes the system not so accurate on the level of Syntactic similarity.[1]

3. PROPOSED SYSTEM

As discussed above, architecture does not have the more accurate syntactic score and all the scores aren't calculated in a single score for the system.

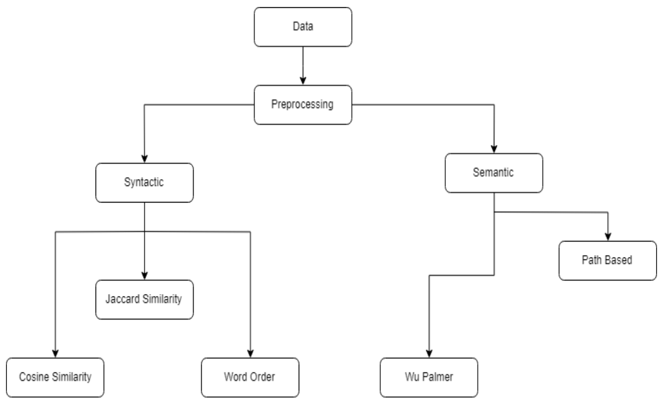


Fig -3: Proposed System

In the proposed system, the data is first passed through the Pre-Processing stage where the data get processed using the NLP (Natural Language Processing) techniques. Then the processed data is created in BOW (Bag of words). This processed data is used in different Syntactic and Semantic techniques.

The techniques gives a score as per the algorithm presented. This score is calculated as per a semantic level and syntactic level as a single score for each level. This score is then presented to the user.

3.1 Hardware and Software Specifications

Table -1: Hardware Details

Processor	1.6 Ghz Intel/AMD
Graphics	512 MB
RAM	4GB

Table -2: Software Details

Programming language	Python 3.6, HTML, CSS
IDE	IDLE/Pycharm
Operating System	Windows 7 and up
Database	MySQL
Browser	Chrome

3.2 Input Details

The project takes two sentences as input. The input is in Hindi language, this input is processed through in back-end for display of results.

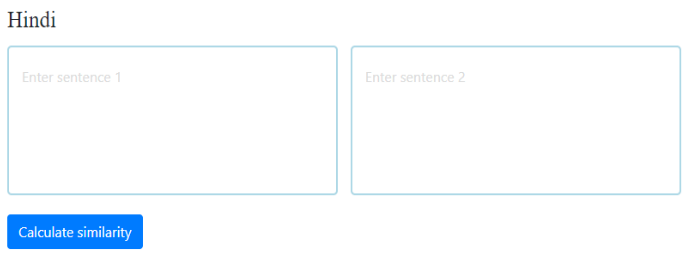


Fig -4: Input of the system

3.3 Data Processing and Evaluation

The data is given from the is Input is Pre-processed using the Pre-processing techniques. The data is translated from the multiple language to English as the corpus for other languages is not much developed.

In Pre-Processing we use tokenization, Stemming, and Stop Word removal. The Pre-Processed data is used for calculation of sentence similarity.

In Sentence Similarity, we apply Syntactic and Semantics Similarity. The Syntactic Similarity approach uses the following algorithms:

1. Cosine Similarity
2. Jaccard Similarity
3. Word-Order Similarity

1. Cosine Similarity

We convert the preprocessed data into vector array then find the cosine angle between the vector array of he both sentences.[1]

2. Jaccard Similarity

In Jaccard Similarity, we find the union of data between the 2 sentences and average it out with the total words in both sentences.[4]

3. Word Order Similarity

We take bigrams of the sentences and hash them then find the similarity of the hash between the both sentences.[8]

The Semantic approach uses the following algorithm:

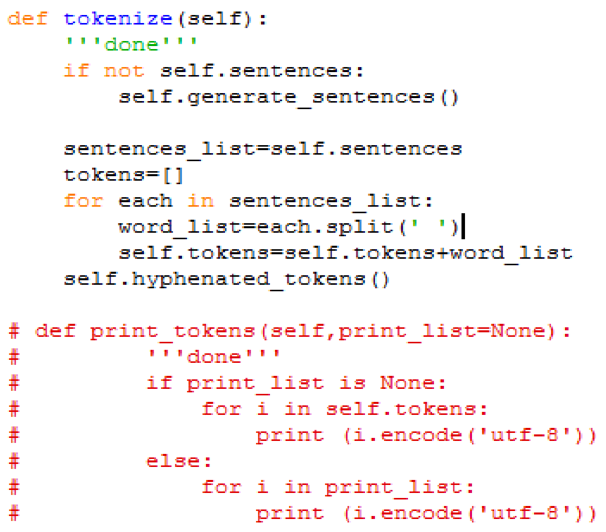
1. Wu- Palmer
2. Path Based

1. Wu - Palmer

In this we use wordnet corpus to get synsets of the provided words and then use wu palmer to find the closest matching synset. Then create array of there synset for both sentences then take their dot product for determining the similarity.[1]

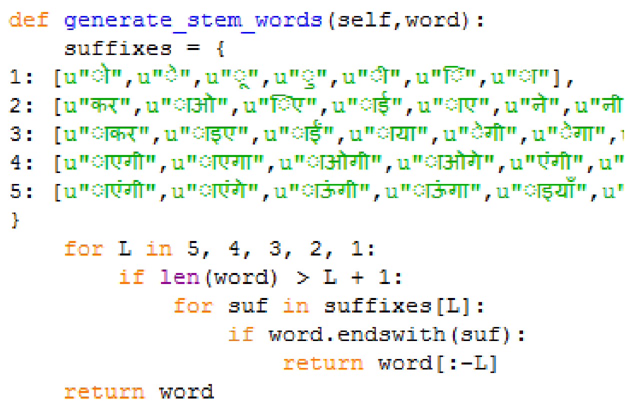
2. Path Based

In this we use wordnet corpus to get synsets of the provided words and then use path based nltk algorithm to find the closest matching synset. Then create array of there synset for both sentences then take their dot product for determining the similarity



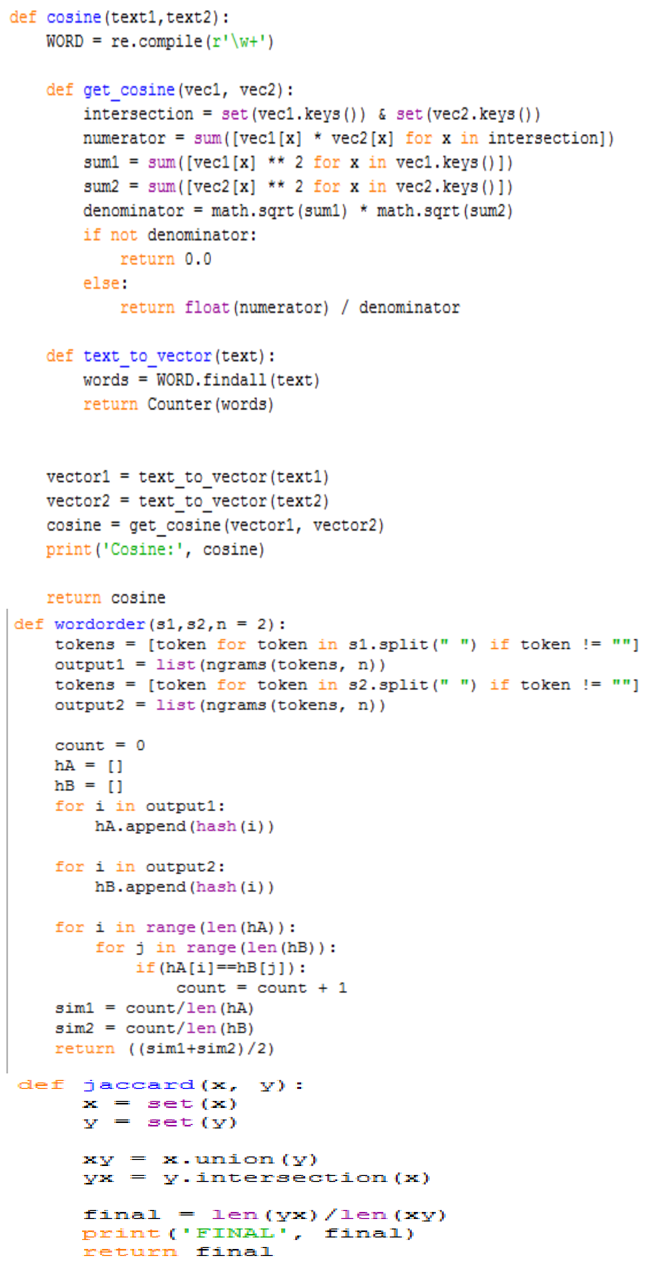
```
def tokenize(self):
    '''done'''
    if not self.sentences:
        self.generate_sentences()

    sentences_list=self.sentences
    tokens=[]
    for each in sentences_list:
        word_list=each.split(' ')
        self.tokens=self.tokens+word_list
    self.hyphenated_tokens()

# def print_tokens(self,print_list=None):
#         '''done'''
#         if print_list is None:
#             for i in self.tokens:
#                 print (i.encode('utf-8'))
#         else:
#             for i in print_list:
#                 print (i.encode('utf-8'))
```

Fig -5: Pre-processing of Data (1)



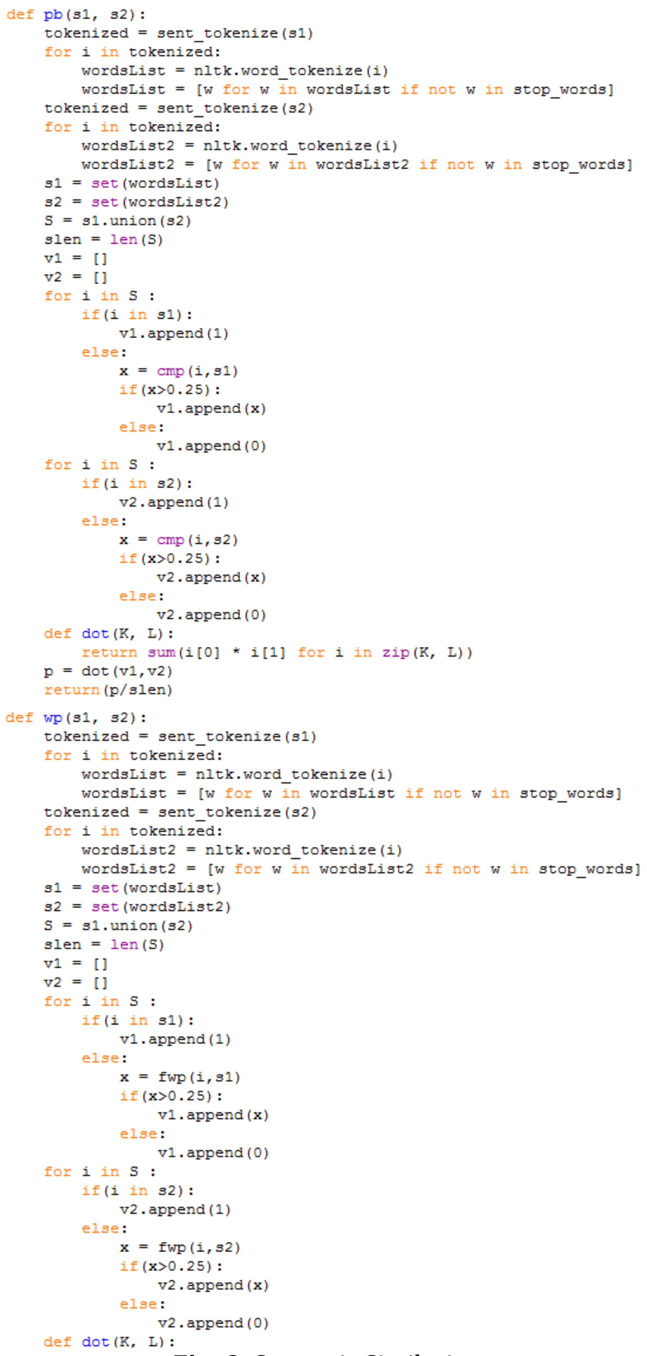
```
def generate_stem_words(self,word):
    suffixes = {
1: [u"ो",u"े",u"ू",u"ु",u"ी",u"ि",u"ा"],
2: [u"कर",u"ाओ",u"िए",u"ाई",u"ाए",u"ने",u"नी
3: [u"ाकर",u"ाइए",u"ाईं",u"ाया",u"ेगी",u"ेगा",
4: [u"ाएगी",u"ाएगा",u"ाओगी",u"ाओगे",u"एंगी",u"
5: [u"ाएंगी",u"ाएंगे",u"ाऊंगी",u"ाऊंगा",u"ाइयाँ",u'
}
    for L in 5, 4, 3, 2, 1:
        if len(word) > L + 1:
            for suf in suffixes[L]:
                if word.endswith(suf):
                    return word[:-L]
    return word
```

Fig -6: Pre-Processing of Data (2)



```
def cosine(text1,text2):
    WORD = re.compile(r'\w+')

    def get_cosine(vec1, vec2):
        intersection = set(vec1.keys()) & set(vec2.keys())
        numerator = sum([vec1[x] * vec2[x] for x in intersection])
        sum1 = sum([vec1[x] ** 2 for x in vec1.keys()])
        sum2 = sum([vec2[x] ** 2 for x in vec2.keys()])
        denominator = math.sqrt(sum1) * math.sqrt(sum2)
        if not denominator:
            return 0.0
        else:
            return float(numerator) / denominator

    def text_to_vector(text):
        words = WORD.findall(text)
        return Counter(words)


    vector1 = text_to_vector(text1)
    vector2 = text_to_vector(text2)
    cosine = get_cosine(vector1, vector2)
    print('Cosine:', cosine)

    return cosine
def wordorder(s1,s2,n = 2):
    tokens = [token for token in s1.split(" ") if token != ""]
    output1 = list(ngrams(tokens, n))
    tokens = [token for token in s2.split(" ") if token != ""]
    output2 = list(ngrams(tokens, n))

    count = 0
    hA = []
    hB = []
    for i in output1:
        hA.append(hash(i))

    for i in output2:
        hB.append(hash(i))

    for i in range(len(hA)):
        for j in range(len(hB)):
            if(hA[i]==hB[j]):
                count = count + 1
    sim1 = count/len(hA)
    sim2 = count/len(hB)
    return ((sim1+sim2)/2)
def jaccard(x, y):
    x = set(x)
    y = set(y)

    xy = x.union(y)
    yx = y.intersection(x)

    final = len(yx)/len(xy)
    print('FINAL', final)
    return final
```

Fig -7: Syntactic Similarity

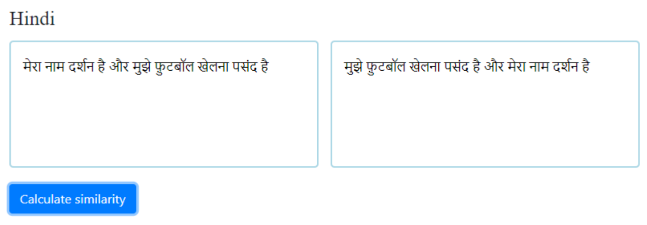


```
def pb(s1, s2):
    tokenized = sent_tokenize(s1)
    for i in tokenized:
        wordsList = nltk.word_tokenize(i)
        wordsList = [w for w in wordsList if not w in stop_words]
    tokenized = sent_tokenize(s2)
    for i in tokenized:
        wordsList2 = nltk.word_tokenize(i)
        wordsList2 = [w for w in wordsList2 if not w in stop_words]
    s1 = set(wordsList)
    s2 = set(wordsList2)
    S = s1.union(s2)
    slen = len(S)
    v1 = []
    v2 = []
    for i in S :
        if(i in s1):
            v1.append(1)
        else:
            x = cmp(i,s1)
            if(x>0.25):
                v1.append(x)
            else:
                v1.append(0)
    for i in S :
        if(i in s2):
            v2.append(1)
        else:
            x = cmp(i,s2)
            if(x>0.25):
                v2.append(x)
            else:
                v2.append(0)
    def dot(K, L):
        return sum(i[0] * i[1] for i in zip(K, L))
    p = dot(v1,v2)
    return(p/slen)
def wp(s1, s2):
    tokenized = sent_tokenize(s1)
    for i in tokenized:
        wordsList = nltk.word_tokenize(i)
        wordsList = [w for w in wordsList if not w in stop_words]
    tokenized = sent_tokenize(s2)
    for i in tokenized:
        wordsList2 = nltk.word_tokenize(i)
        wordsList2 = [w for w in wordsList2 if not w in stop_words]
    s1 = set(wordsList)
    s2 = set(wordsList2)
    S = s1.union(s2)
    slen = len(S)
    v1 = []
    v2 = []
    for i in S :
        if(i in s1):
            v1.append(1)
        else:
            x = fwp(i,s1)
            if(x>0.25):
                v1.append(x)
            else:
                v1.append(0)
    for i in S :
        if(i in s2):
            v2.append(1)
        else:
            x = fwp(i,s2)
            if(x>0.25):
                v2.append(x)
            else:
                v2.append(0)
    def dot(K, L):
```

Fig -8: Semantic Similarity

3.4 Output Details

The output is displayed on a Web Application using a Bar Chart. This chart displays the similarity of the given data with "0" being the lowest and "1" being the highest similarity of the sentences provided as input.


Hindi

मेरा नाम दर्शन है और मुझे फुटबॉल खेलना पसंद है

मुझे फुटबॉल खेलना पसंद है और मेरा नाम दर्शन है

Fig -9: Input

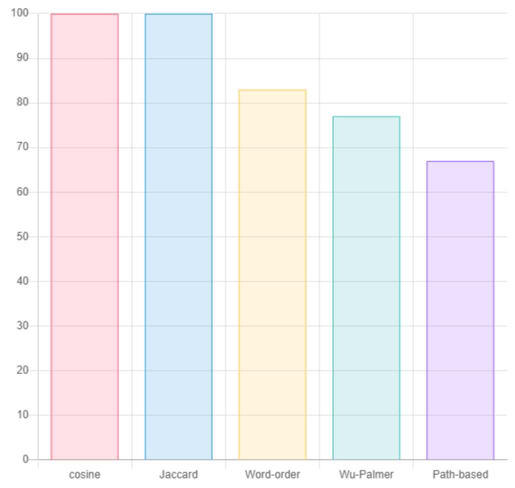


Fig -10: Output

4. CONCLUSIONS

In this paper, the study of different Natural Language Processing techniques is presented. The different Preprocessing techniques such as Stop Word Removal, Tokenization, Stemming are explained. Different semantic measures such as Cosine similarity, Word Order, Jaccard Similarity are explained. The different syntactic approaches like Wu Palmer and Path Based Similarity are also described.

The comparative study of various techniques mentioned above is presented in this report. The performance measures the accuracy of the similarity of the given different data. The applications of this domain is identified and presented. It overcomes the limitations of previous system on the syntactic approach, which can give us more accurate results.

5. FUTURE SCOPE

The accuracy of the whole system can increased by applying a Machine Learning model, which can be trained with large dataset consisting of various sentences. The model can be applied for each algorithm in the system to get more accurate results.

The semantic similarity score between sentences can be further improved by adding more semantic similarity algorithms in the system.

REFERENCES

[1] Pantulkar Sravanthi, Dr. B. Srinivasu, “Semantic Similarity Between Sentences”. International Research Journal of Engineering and Technology (IRJET), 2017.

[2] Atoum, Issa & Otoom, Ahmed. (2016). Efficient Hybrid Semantic Text Similarity using Wordnet and a Corpus. International Journal of Advanced Computer Science and Applications. 7. 10.14569/IJACSA.2016.070917.

[3] H. Ruan, Y. Li, Q. Wang and Y. Liu, "A Research on Sentence Similarity for Question Answering System Based on Multi-feature Fusion," *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, Omaha, NE, 2016, pp. 507-510.

[4] Sarkar, Sandip & Saha, Saurav & Bentham, Jereemi & Pakray, Dr. Partha & Gelbukh, Alexander. (2016). NLP-NITMZ@DPIL-FIRE2016: Language Independent Paraphrases Detection.

[5] Sneha B., Mohit D., Zorawar Singh V. (2016) Comparison of Different Similarity Functions on Hindi QA System. In: Satapathy S., Joshi A., Modi N., Pathak N. (eds) Proceedings of International Conference on ICT for Sustainable Development. Advances in Intelligent Systems and Computing, vol 408. Springer, Singapore

[6] Sultan, M.A., Bethard, S. and Sumner, T., 2015. DLS $@ $ CU: Sentence Similarity from Word Alignment and Semantic Vector Composition. In Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015) (pp. 148-153).

[7] Hatzlvassiloglou, V., Klavans, J.L. and Eskin, E., 1999. Detecting text similarity over short passages: Exploring linguistic feature mbinations via machine learning. In 1999 Joint SIGDAT conference on empirical methods in natural language processing and very large corpora.

[8] Elkhidir, M., Ibrahim, M.M., Khalid, T.A., Ibrahim, S. and Awadalla, M., 2015, September. Plagiarism detection using free-text fingerprint analysis. In 2015 World Symposium on Computer Networks and Information Security (WSCNIS) (pp. 1-4). IEEE.

[9] Jingling, Z., Huiyun, Z. and Baojiang, C., 2014, November. Sentence similarity based on semantic vector model. In 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (pp. 499-503). IEEE.

[10] Gupta, D., Vani, K. and Singh, C.K., 2014, September. Using Natural Language Processing techniques and fuzzy-semantic similarity for automatic external plagiarism detection. In 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 2694-2699). IEEE.

[11] Abdi, A., Idris, N., Alguliyev, R.M. and Aliguliyev, R.M., 2015. PDLK: Plagiarism detection using linguistic knowledge. Expert Systems with Applications, 42(22), pp.8936-8946.

[12] Joshi, N., Kadam, D., Kadu, S. and Chavan, S., Survey on Sentence Similarity System.