Abstractive Text Summarization: Enhanced Using Feature Fusion-Based Neural Network Model

Dr. S.Vijayarani[1], Dr. R. Janani[2], Ms. A.Revathi[3]
[1]Associate Professor, Dept of Computer Science
[1, 2, 3] Bharathiar University, Coimbatore, Tamilnadu, India

Abstract- *Text summarization refers to the process of condensing a given text, like an article, document, or web page, into a shorter version that maintains the essential information and main concepts. Text summarization techniques can be broadly categorized into two main approaches: extractive methods and abstractive methods. Extractive summarization extracts important information from the source text, while abstractive summarization generates new sentences that capture the essence of the text. Summaries can be classified into two types: indicative, which represents only the main idea, and informative, which provides concise information from the document. To achieve abstractive text summarization, this research work proposes the Neural Network based Abstractive Text Summarization (NN_ATS) algorithm, which is compared against commonly used techniques such as Decision Tree and Naïve Bayes algorithms. The system accepts inputs in the form of URLs, local file paths, or plain texts. Experimental results demonstrate that the proposed NN_ATS algorithm outperforms other techniques in summarizing both single documents and multiple documents.*

Keywords- Text Summarization, Machine Learning, Neural Network, Abstractive, Extractive

I. INTRODUCTION

Text mining, also known as knowledge discovery in text, is an important component of data mining. It involves the extraction of concealed patterns and valuable insights from vast amount of unstructured data. Text mining techniques have wide-ranging applications across diverse domains, including industry, academia, web applications, the internet, and beyond. These techniques are employed to extract valuable knowledge and insights from unstructured data. These techniques encompass information extraction, information retrieval, natural language processing (NLP), classification, clustering, and summarization. Text summarization is the procedure of extracting essential and significant information from a given text and presenting it in the form of a concise summary. This approach involves the selection of necessary details while eliminating redundant information, thus enabling a more efficient understanding of the document's main concept within a shorter timeframe. Text summarization serves as a valuable technique for comprehending the primary essence of a document in a time-efficient manner. Automatic text summarization refers to the process in which a computer program generates a summary encapsulating the key points of the text. By leveraging automatic text summarization, readers can efficiently evaluate the relevance of available documents and make informed decisions about which ones to explore further.

II. RELATED WORKS

Ahmad T. Al-Taani [1] has evaluated automatic text summarization. Automatic Text Summarization (ATS) is a computational technique that aims to generate concise yet informative summaries of source text, condensing the essential information while preserving its key elements. This summarization process aids in rapidly identifying the necessary details within the original text, facilitating efficient information retrieval and comprehension. Text summarization can be classified according to the source which can be single or multi documents. Single document can provide the summary of one document and multi-document summarization can provide the summary that should relate all the documents. In text summarization, there are two approaches namely extractive and abstractive text summarization. Extractive text summarization can extract the most highlighted information and abstractive text summarization can have deeper analysis and which provides summary in new way. This paper mainly evaluates methods of extractive based approach namely Statistical approaches, Graph-based approaches, Machine learning approaches, Clustering methods and Meta-heuristic search approach.

Arpita Sahoo, Dr.Ajit Kumar Nayak [2] surveyed Automatic Text summarization and the approaches like abstractive and extractive text summarization and analyzes which provides more efficient and accurate summary of the original document. The abundance of intricate facts, figures, and data on information servers has led to a concerning rise in "information overload," posing a significant challenge for individuals seeking to process and manage such overwhelming amounts of information. It has always been a task to

summarize and sort mountains of documents manually and a time-consuming job to generate a summary keeping all semantics in consideration. Hence, automatic text summarization can be a key solution for this problem. Text summarization serves as a valuable tool for facilitating rapid comprehension of extensive collections of text documents and finds numerous practical applications in real-life scenarios.This solution of summarization facility will help users to see at a glance what a collection of text document is about and provides a new way of managing a huge accumulation of information.

Rajat K. Kulshreshtha, Anuranjan Srivastava, Mayank Bhardwaj [3] has discussed Text summarization methods. From the last decade, it is observed that there is a never-before-seen positive flux in the quantity of textual information that a single document or even multi documents presented. This survey tends to present a go through upon some of the most relevant approaches for summarization from surface and also classifying techniques based on single and multiple documents used as an input, the aim is to present a good read to readers, young and budding linguistics about various existing methods used for text summarization.

Diksha Harbola, Shankersinh Vaghela bapu [4] have reviewed the basics of multi-document summarization, and analysis the several methods of extractive approaches and extractive methods and then evaluation methods of estimating the performance of the summarization techniques. Text summarization is a technique for creating short but useful information from a text document. Automatic text summarization plays a vital role in producing summarized information from a large corpus. Extractive summarization is a technique that revolves around extracting relevant sentences directly from the source documents to form a concise and informative summary.Various systems are based on feature extraction method. The main goal of all systems is to obtain relevant data and summarize them as per the user's input.

Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Safeid, Elizabeth D. Trippe [5] the review thoroughly examines the primary approaches employed in automatic text summarization and delves into the distinct processes utilized for summarization. It provides insights into the effectiveness and limitations of these diverse methods. Over the past years, the colossal surge in text data from various sources has been remarkable. This extensive volume of textual information represents a valuable resource of knowledge, necessitating efficient summarization techniques to unlock its true utility.

Oguzhan Tas1, Farzad Kiyani [6] has compared the various text summarization techniques such as extractive and abstractive text summarization. Text summarization is compressing the source text into a diminished version conserving its information content and overall meaning. Because of the great amount of information, we are provided it and thanks to development of internet technologies, text summarization is a tool for interpreting text information. In extractive summarization, the method entails carefully selecting sentences with eminent rank from the original document, leveraging both word and sentence features. These chosen sentences are then intelligently assembled to form a coherent and informative summary.The importance of the sentences is decided based on statistical and linguistics features of sentences. An abstractive summarization is used to understand the main concepts in a given document and then expresses those concepts in clear natural language.

N.Nazari and M.A. Mahdavi [7] analyzed an in-depth introduction to automatic text summarization and also mention some evaluation techniques to evaluate the quality of automatic text summarization. Text summarization endeavors to produce a summary version of a text while maintaining original ideas. It requires automatic text summarization mechanism to extract important information. This system is intended to assist with content reduction by keeping the relevant information and filtering the non-relevant parts of the text. In that there are two types of input namely single documents and multi documents. Regarding the output, summarization systems can be broadly categorized into two major types. One approach would extract exact sentences from the original document to build the summary. An alternative approach involves a more intricate process, where the generated text is a rephrased rendition of the original document.

III. METHODS

A. Documents

At first, various document's URL or internal files path or texts are fetched as input. It can be unstructured or semi-structured data. The input may be single document or multi document. Python Libraries are used to fetch the path of the documents as input. After the input is given, first it checks whether the input is URL or local files path or texts through conditional statements such as if-else statement. The function called validators, which is used to check the given URL is valid or not. Next the library called os.path.isfile is used to check the given input is local files path or not and also check if it is present or not. Finally os.path.isdir library is used to check the given input is directory or not and also check its presence. If the given input is URL then the library called beautifulsoup is used, which is very useful python utility for

web scraping. Another important library is lxml which is a combination of XML and HTML.

In this script, first import the important libraries required for scraping the data from the web. The function is used called urlopen from the urllib.request utility to scrape the data. Next, to call the read function on the object returned by urlopen function in order to read the data. To parse the data, it use beautifulsoup object and pass it the scraped data object. If the fetched input is internal document then, open the document with the help of open () function and in 'r' mode for reading the document and stores the opened document in text_summ variable. It is for single document summarization. If the given input is directory then through looping statement it opens the document from the directory in one-by-one with the help of open () function and read the opened document with the help of 'r' mode. Then, it stores all the opened documents in text_summ variable with filename and its content. It is for multi document summarization.

In that, all the text is enclosed inside the <p> tags. To retrieve the text, there is a need to call find_all function on the object returned by the beautiful soup. The tag name is passed as a parameter to the function. The find_all function returns all the paragraphs in the form of a list. All the paragraphs have been combined to recreate the summary. After that, it reads each and every word from the document or URL with the help of looping statement and stores it into all Paragraph Content variable.

```
from urllib import request
from bs4 import BeautifulSoup as bs
import os
import validators
import glob

summ = input("Enter directory or url or localfile path or some text : ")
allParagraphContent = ""
text_summ = ""

if validators.url(summ) is True:
  text_summ = request.urlopen(summ)
elif os.path.isfile(summ) is True:
  text_summ = open(summ, 'r').read()
elif os.path.isdir(summ) is True:
  for filename in glob.glob(os.path.join(summ, '*.txt')):
    with open(filename, 'r') as f:
      text = f.read()
      print (filename)
      text_summ = open(filename, 'r').read()
else:
  text_summ = summ


soupObject = bs(text_summ, 'lxml')
paragraphContents = soupObject.findAll('p')

for paragraphContent in paragraphContents:
  allParagraphContent += paragraphContent.text
```

Code of fetching input

B. Converting Paragraphs into Sentences:

It is the beginning step to convert the paragraphs from the given document into sentences. The standard way of converting into sentences from the paragraphs is to split or break the paragraph whenever a period is encountered. The following script is used to removes the square brackets and extra spaces then replaces the resulting multiple spaces by a single space.

```
allParagraphContent_cleanerData = re.sub(r'\[[0-9]*\]',' ', allParagraphContent)
allParagraphContent_cleanerData = re.sub(r'\s+',' ', allParagraphContent_cleanerData)
```

Code of removing brackets

The all Paragraph Content_cleanerData object contains text without brackets. However, other than brackets nothing will be removed since this is the original document. Still it contains numbers, punctuation marks and special characters. Then this text is used to create summaries and weighted word frequencies. At this point, there is a need to tokenize all the paragraphs from the document into sentences. The allParagraphContent_cleanerData object is used for tokenizing the paragraphs from the document into sentences since it contains full stops. Without using the full stop as a parameter, it is not possible to tokenize all the paragraphs from the documents into sentences. The following script performs sentence tokenization

```
from nltk.tokenize import sent_tokenize
sentences_tokens = nltk.sent_tokenize(allParagraphContent_cleanedData)
```

Pseudo Code of sentence tokenization

C. Text Preprocessing:

After converting into sentences, Preprocessing are done by removing all the special characters, stop words and numbers from all the sentences, then Tokenization is required for all the sentences to get all the words that exist in the sentences and Lemmatization is performed for getting root form of the inflected words.

1. Removal of special characters, punctuation and numbers:

The allParagraphContent_cleanedData object is created for clean the text by removing numbers, punctuation marks and special characters.

```
allParagraphContent_cleanedData = re.sub(r'[^a-zA-Z]', ' ', allParagraphContent_cleanerData)
allParagraphContent_cleanedData = re.sub(r'\s+', ' ', allParagraphContent_cleanedData)
```

Pseudo Code of removing punctuation, special characters and numbers

Now we have two objects all Paragraph Content_cleaner Data which contains the original document and all Paragraph Content_cleaned Data which contains the formatted document. The all Paragraph Content_cleaned Data is used to create weighted frequency histograms for the words and will replace these weighted frequencies with the words in the all Paragraph Content_cleanerData object.

2. Tokenization

Tokenization is the essential process of breaking down a string or text into a list of individual tokens or meaningful units.Now perform the word tokenization, to get all the words that exist in the sentences from the allParagraphContent_cleanedData object with using the inbuilt method from the python NLTK Library.

```
from nltk.tokenize import word_tokenize
words_tokens = nltk.word_tokenize(allParagraphContent_cleanedData)
```

Pseudo Code of word tokenization

3. Converting text into lowercase

It is regarded as one of the most straightforward and highly efficient forms of text preprocessing. The function called lower () is used to convert all the words into lowercase.

4. Stop word Removal

Stop words comprise a collection of frequently used words in a given language. Examples of stop words in English include "a," "the," "is," "are," and others. These words are considered insignificant as they do not contribute significantly to the overall meaning of a sentence.Hence, it can be safely removed without causing any change in the meaning of the sentence. The aim behind using stop words is that, by removing low information words from text and focus on the important words instead. The NLTK library has a set of stop words and it can use to remove stop words from the documents and return a list of word tokens. The following script is used to remove the stop words with using the inbuilt method from the python NLTK Library.

```
from nltk.corpus import stopwords
stopwords = set(stopwords.words("english"))
```

Pseudo Code of removing Stop words

5. Lemmatization

Lemmatization shares remarkable similarities with stemming as they both aim to reduce words to their root forms by removing inflections. However, the key distinction lies in lemmatization's approach, as it seeks to achieve this transformation in a linguistically accurate and contextually appropriate manner.It will use the dictionary such as WordNet Lemmatizer for mappings which is used to get the lemmas of words. The following script is used to perform lemmatization with using the inbuilt method from the python nltk Library.

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
word = lemmatizer.lemmatize(word, pos="v")
```

Pseudo Code of lemmatization

6. Find Weighted Frequency of Occurrence:

After the Preprocessing is done, it finds the weighted frequency of occurrences of all the words. Weighted frequency is calculated of each word by dividing its frequency by the frequency of the most occurring word. The variable words_tokens is used to find the frequency of occurrence each word since it doesn't have punctuation, digits, or other special characters. Here word frequency is used which is, word counting technique in which a sorted list of words and their frequency is generated. Where, the frequency means occurrences of each word.

```
word_frequencies = {}

for word in words_tokens:
    if word not in stopwords:
        if word not in word_frequencies.keys():
            word_frequencies[word] = 1
        else:
            word_frequencies[word] += 1
```

Pseudo Code of finding word frequency

In this script, first create an empty word_frequencies dictionary is created. In that, only use the words that are not part of the stop words. Then through loop it checks if the words are stop words. If not, it proceeds to check whether the words exist in word_frequencies dictionary or not. If the word is encountered for the first time, it is added to the dictionary as a key and its value is set to 1. Alternatively, if the word already exists in the dictionary, its corresponding value is updated by 1. In the end, to determine the weighted frequency, one can compute the ratio of the occurrences of each word to

the frequency of the most frequently occurring word in the collection.

```
maximum_frequency_word = max(word_frequencies.values())

for word in word_frequencies.keys():
  word_frequencies[word] = (word_frequencies[word]/maximum_frequency_word)
```

Pseudo Code of calculating weighted frequency

7. Replace words by Weighted Frequency in Original Sentences:

This step is for apply the weighted frequency value in the place of corresponding words in original sentences then sentence score is calculated by adding weighted frequency value of each word that occur in that particular sentence.

```
sentences_scores = {}

for sentence in sentences_tokens:
   for word in nltk.word_tokenize(sentence.lower()):
      if word in word_frequencies.keys():
         if (len(sentence.split(' '))) < 30:
            if sentence not in sentences_scores.keys():
               sentences_scores[sentence] = word_frequencies[word]
            else:
               sentences_scores[sentence] += word_frequencies[word]
```

Pseudo Code of calculating sentence score

In this script, first create an empty sentences_scores dictionary. In this dictionary, the sentences themselves serve as the keys, while their corresponding scores act as the values. Next, using loop each sentence in the sentences_tokens and tokenize the sentence into words. Next check if the word exists in the word_frequencies dictionary. This check is performed since created the sentences_tokens list from the allParagraphContent_cleanedData object. On the other hand, the word frequencies were calculated using the words_tokens object, which doesn't contain any stop words, numbers, etc.

In this script, all the sentences occurs in summary can have less than 30 words for specifying parameter but it can change as per user's need. Next, it checks whether the sentence exists in the sentences_scores dictionary or not. If the sentence doesn't exist, it can add it to the sentences_scores dictionary as a key and assign it the weighted frequency of the first word in the sentence, as its value. On the contrary, if the sentence exists in the dictionary, it can simply add the weighted frequency of the word to the existing value.

D. Text Summarization

Text summarization is a sophisticated process that involves the extraction and compilation of pertinent information from a given text, presenting the collected data in a concise and coherent summary.It is a way of selecting important information and removing redundant information in less time.

A. Neural Network based Abstractive Text Summarization (NN_ATS)

The Neural Network based Abstractive Text Summarization (NN_ATS) is proposed to summarize the documents in abstractive approach. There are three phases of proposed algorithm consists, network training, feature fusion, and sentence selection. The training step includes training a neural network to identify the sentence types with its description. The feature fusion step cuts the neural networks and breaks down the activations of the hidden layer unit to discrete frequency values. It simplifies the essential characteristics that must exist in sentence summary by fusing the characteristics and discovering patterns in summary sentences. The final step is selection of sentences, which uses the modified neural network to filter the text and select only the highly ranked phrases.

i. Training of Neural Networks

The training step includes training a neural network to identify the sentence types with its description. The proposed model consists of seven input layer and one output layer. Also, it consists of six hidden layers for text summarization. The main of network training is to achieve the global minimum of energy function. This training is accomplished with different set of sentences and several paragraphs for summarization. The network learns the pattern which is associated by the sentences and paragraphs inherently. The total energy function is calculated as:

$$\theta(x,y) = E(x,y) + P(x,y) \qquad \text{Eq.1}$$

Where x, y be the layers of neural network and E, P are the global minimum energy function.

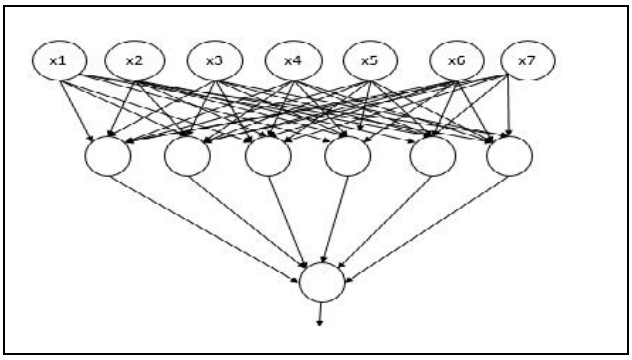

Figure 1: Network Training

ii. Feature Fusion

Once the network is trained, there is a need to determine the relationship between sentences and paragraph features. The main aim of this feature fusion is to remove the uncommon features and collapse the common features. The network tree is pruned by the sentence weights and the hidden neuron is discovered with its features. The connections of hidden layer also pruned, and the entire network is trained again. Then the activation ReLu function is introduced with the hidden neurons. This can be expressed as,

$$R(z) = Z \quad z > 0$$
$$R(z) = 0 \; z \leq 0 \qquad \text{Eq.2}$$

Based on the activation function, the features are re arranged and the network model has been trained.

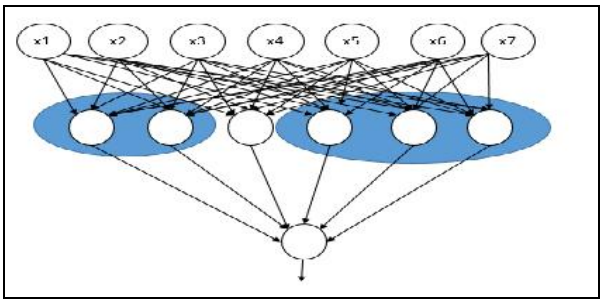

Figure 2: Feature Fusion

Algorithm

Step 1: Data Preparation
Gather a dataset of text documents and their corresponding human-generated summaries
dataset = gather_dataset()
Step 2: Preprocess the data
preprocessed_data = preprocess_dataset(dataset)
Split the dataset into training, validation, and test sets
train_set, val_set, test_set =
split_dataset(preprocessed_data)
Step 3: Network Training
Define the neural network architecture for abstractive text summarization
model = define_neural_network()
Define the optimizer and loss function
optimizer = initialize_optimizer(model)
loss_function = initialize_loss_function()
Step 4: Train the model
for epoch in range(num_epochs):
for batch in train_set:
Forward pass
outputs = model(batch.input)
Step 5: Feature Fusion
Perform feature fusion to capture important information from multiple sources
fused_features = perform_feature_fusion(model, dataset)
Step 6: Sentence Selection
Select important sentences for summarization based on the fused features
selected_sentences = select_sentences(fused_features)
Step 7: Summarization
Generate summaries based on the selected sentences
summary = generate_summary(selected_sentences)
Step 8: Evaluation
Evaluate the generated summaries using appropriate evaluation metrics
evaluation_metrics = evaluate_summary(summary, dataset.gold_summaries)

iii. Sentence Selection

The training and pruning stage is over, then next step is to select the most weighted sentences. This phase is accomplished by providing control parameters for the radius and frequency of hidden layer activation clusters to select highly ranked sentences. The sentence ranking is directly proportional to cluster frequency and inversely proportional to cluster radius. The sentences which are satisfying the radius condition, that sentences are selected for summarization.

iv. Summary

Finally, the summary is produced. It has most consequential points of the original document. The produced summary is based on Abstractive text summarization technique which supports both single document and multi documents.

IV. RESULTS AND DISCUSSION

The experiments are carried out on a 2GHz Intel CPU with 1 GB of memory and running on windows 10. Implemented the existing and proposed algorithm using Python to obtain the abstractive summarization of text documents.

A. Performance Measures

In order to perform this task, there are four performance measures used in this project work. They are precision, recall, f-measure and accuracy of the text summarization. To estimate the performance, the confusion matrix is essential. The confusion matrix as follows,

TABLEI: Confusion Matrix

	A Document which belongs to the particular category	**A Document which does not belong to the particular category**
Document category accepted by the classifier	TP	FP
Document category rejected by the classifier	FN	TN

This matrix is established in the terms,

- True Positives (TP) – It is defined; the similar documents are classified in the same category.

- True Negatives (TN) – It is defined; the dissimilar documents are classified in the different category.
- False Positives (FP) – It is defined; the dissimilar documents are classified in the same category.
- False Negatives (FN) – t is essential to define the criteria for classification accurately, as similar documents should ideally be placed in the same category rather than being classified differently.

$$Precision = \frac{TP}{TP+FP} \quad \text{Eq.3}$$

$$Recall = \frac{TP}{TP+FN} \quad \text{Eq.4}$$

$$F-Measure = \frac{2TP}{2TP+FP+FN} \quad \text{Eq.5}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad \text{Eq.6}$$

B. Results

Table II shows that the performance of existing and proposed text summarization for single document input. From this, it is inferred that the proposed NN_ATS algorithm summarizes the more specifically than other existing algorithms. Figure 3 illustrates the comparison of existing and proposed methods for single document summarization.

Table II: Performance Measures for Single Document

	Precision	Recall	F-Measure	Accuracy
Decision Tree	0.719	0.722	0.720	0.804
Naïve Bayes	0.724	0.731	0.730	0.811
NN_ATS	**0.744**	**0.759**	**0.755**	**0.820**

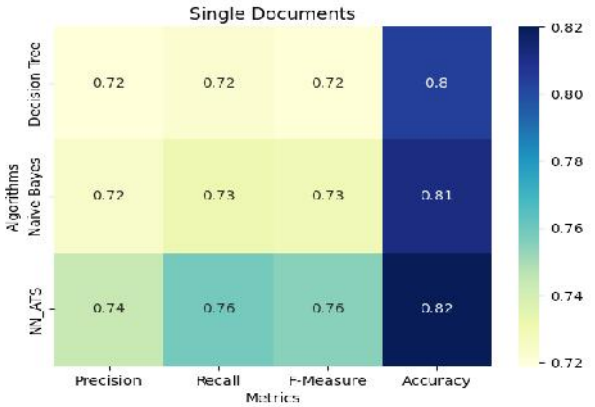

Figure 3: Performance Measures for Single Document

Table III shows that the performance of existing and proposed text summarization for multiple documents input. From this, it is inferred that the proposed NN_ATS algorithm summarizes the more specifically than other existing algorithms. Figure 4 clarifies the comparison of existing and proposed methods for multiple documents summarization.

Table III: Performance Measures for Multi Documents

	Precision	Recall	F-Measure	Accuracy
Decision Tree	0.749	0.753	0.751	0.839
Naïve Bayes	0.751	0.766	0.759	0.840
NN_ATS	**0.762**	**0.779**	**0.769**	**0.861**

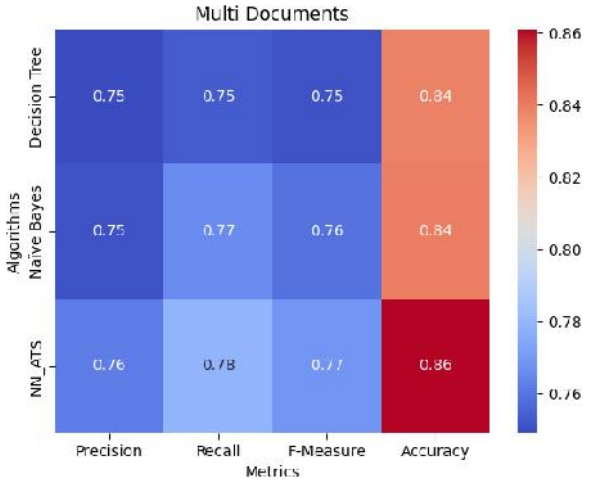

Figure 4. Performance Measures for Multi Document

Table IV shows that the performance of existing and proposed text summarization for URL input. From this, it is inferred that the proposed NN_ATS algorithm summarizes the more specifically than other existing algorithms. Figure 5 demonstrates the comparison of existing and proposed methods for URL summarization.

Table IV: Performance Measures for URL

	Precision	Recall	F-Measure	Accuracy
Decision Tree	0.721	0.733	0.727	0.811
Naïve Bayes	0.726	0.741	0.734	0.820
NN_ATS	**0.735**	**0.752**	**0.745**	**0.832**

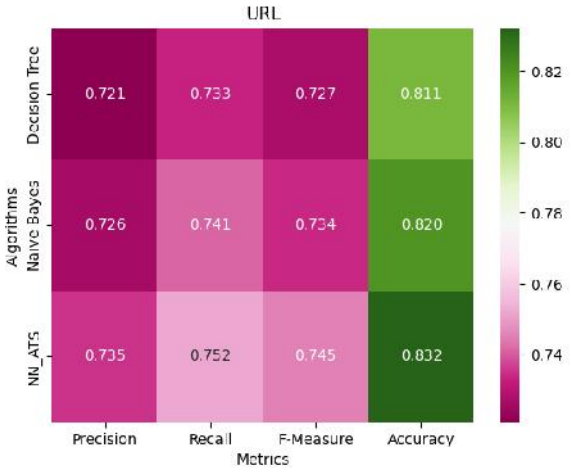

Figure 5: Performance Measures for URL

V. CONCLUSION

Text Summarization is a branch of Natural Language Processing (NLP) as the demand for compression due to rapid growth of information on the net. It is difficult for humans to summarize large amount of text in document. Accurate information helps to search more effectively and efficiently. Thus, text summarization is need and used by business analyst, teachers and students also. It requires minimum time to produce a summary. Summaries can reduce reading time. Automatic summarization helps to improve the effectiveness of indexing. Without the use of NLP, the generated summary may suffer from lack of semantics and cohesion. As abstractive summarization needs more learning and reasoning, it is bit more complex than extractive approach, but abstractive approach provides more meaningful and accurate summary when compared to extractive approach. The main goal of this research work is to develop a text summarization system based on Abstractive Text Summarization and it was implemented successfully. Extensive research has been conducted on both extractive and abstractive methods of text summarization. The developed text summarization system gives the best result when compared to existing tools. This system requires minimum time to produce a summary. The existing tools can summarize only single document, but this system can support both single document and multi documents. But this system can support only text document (‘.txt’) format as input not pdf document (‘.pdf)’ and word document (‘.docx’) and also this system can support only English language.

In future, this work can be enhanced which support other languages like French, German, Italian and Spanish. And also in future, there is a need to develop the text summarization system which support ‘.pdf’ and ‘.docx’ format as input.

REFERENCES

[1] Ahmad T. Al-Taani,” Automatic Text Summarization Approaches”, Internatioanl Conference on Infocom Technologies and Unmanned Systems (ICTUS’2017).

[2] Arpita Sahoo, Dr.Ajit Kumar Nayak, “Review Paper on Extractive Text Summariztion”, International Journal of Engineering Research in computer science and Engineering (IJERCSE) vol 5, Isuue 4, April 2018.

[3] Rajat K. Kulshreshtha, Anuranjan Srivastava, Mayank Bhardwaj, “ A survey paper on Text summarization Methods”, International Research Journal of Engineering and Technology(IRJET) vol 5, Issue 11, Nov 2018.

[4] Diksha Harbola, Shankersinh Vaghela bapu, “A survey on Extractive Text Summarization Techniques”, international journal of Scientific and research development, vol : 6, Issue: 2.

[5] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Safeid, Elizabeth D. Trippe, “Text Summarization Techniques: A Brief Survey”, (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 8, no 10, 2017.

[6] Oguzhan Tas1, Farzad Kiyani2, “A survey on Automatic Text Summarization”, 2nd World Conference on technology, Innovation and Enterpreneurship.

[7] N.Nazari and M.A. Mahdavi, “A Survey on Automatic Text Summarization”, Journal of AI and Data Mining, vol 7, no 1, 2019, 121-135.

[8] Jin-ge Yao, Xiaojun Wan, Jianugo Xiao, “Recent Advances in Document Summarization”, National Natural Science Foundation of china.

[9] Aysa Siddika Asa, Sumya Akter, Md.Palash Uddin, Md. Delowar Hossain, Shikhor Kumer Roy, Masud Ibn Afjal , “A Comprehensive Survey on Extractive Text Summarization Techniques”, American Journal of Engineering Research (AJER).

[10] Lavanya K C, C,Sivamani, Linnet Tomy, Ann Rija Paul, “ Two-Level Text Summarization with sentiment Analysis for Multi-Document Summarization”, International Journal of Recent Technology and Engineering(IJRTE) ISSN: 2277-3878, vol-7, Issue-5, January 2019.

[11] Jiwei Tan, Xiaojun Wan, Jianuguo Xiao Institute of Computer Science and Technology, Peking University, “Abstractive document summarization with a Graph-Based attentional neural model”,2017.

[12] Pratibha Devihosur, Naseer, “Automatic Text Summarization using Natural Language Processing”, Intenational Research Journal of Engineering and Technology (IRJET), vol: 4, issue :08, Aug 2017.

[13] Sabina Yeasmin, Priyanka Basak Tumpa, Adiba Mahjabin Nitu, Md. Palash Uddin, Emran Aliand Masud Ibn Afjal,

"Study of Abstractive Text Summarization Techniques", American Journal of Engineering Research (AJER).

[14] Deepali K. Gaikwad and C. Namarta Mahender "A Review paper on Text Summarization", International journal of Advanced Research in Computer and Communication Engineering, vol 5, Issue 3,March 2016.

[15] Hamza Shabbir Moiyadi, Harsh Desai, Dhairya Pawar, Geet Agarwal, Nilesh M.Patil, "NLP Based Text Summarization using Semantic Analysis", International Journal of Advanced Engineering Management and Science (IJAEMS) [vol-2, Issue-10, oct-2016].

[16] Knitha D.K and D. Muhammad Noorul Mubarak, "An overview of Extractive based Automatic text summarization systems", International Journal of computer science & Information Technology (IJCSIT) vol 8, no 5, oct 2016.

[17] N.Morantanch, S. Chitrakala, " A Survey on Abstractive Text Summarization", International Conference on circuit, Power and Computing Technologies [ICCPCT].

[18] Neelima Bhatia, Arunima Jaiswal, "Automatic Text Summarization:Single and Multiple Summarizations", International Journal of Computer Applications.

[19] Saranyamol C S, Sindhu L, "A Survey on Automatic Text Summarization", International Journal of Computer Science and Information Technologies, 2014, vol, 5 Issue 6.

[20] Niharika verma, prof. Ashish Tiwari, "A survey of Automatic Text Summarization", International Journal of Engineering & Technology (IJERT), vol. 3 Issue 6, june 2014.

[21] https://stackabuse.com/text-summarization-with-nltk-in-python/