

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By

Lakkaraju Alekhya
(Reg.no: 40731052)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU

APRIL2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **LAKKARAJU ALEKHYA(40731052)** who carried out the project entitled "**TEXT SUMMARIZATION USING DEEP LEARNING**" under my supervision from FEB 2023 to APRIL 2023.

Internal Guide

Ms G.Anbu Selvi M.E.,(Ph.D)

Head of the Department

Dr.S.Vigneshwari M.E., Ph.D

Submitted for Viva voce Examination held on

17-04-2023

Internal Examiner

External Examiner

DECLARATION

I **Lakkaraju Alekhya** hereby declare that the Project Report entitled **TEXT SUMMARIZATION USING DEEP LEARNING**. Done by me under the guidance of **Ms G.Anbu Selvi M.E.,(Ph.D)** submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 17-04-2023

PLACE: Chennai



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing, **Dr. S.Vigneshwari M.E., Ph.D.**, and **Dr.L.Lakshmanan M.E., Ph.D.**, Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Ms G.Anbu Selvi M.E(Ph.D)** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

I wish to express my thanks to all Teaching and Non-teaching staff Members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

In recent years, deep learning has achieved great success in many fields, such as computer vision and natural language processing. Compared to traditional machine learning methods, deep learning has a strong learning ability and can make better use of datasets for feature extraction. Because of its practicability, deep learning becomes more and more popular for many researchers to do research works. In this paper, we mainly introduce some advanced neural networks of deep learning and their applications. Besides, we also discuss the limitations and prospects of deep learning.

The availability of huge volume of Image and Video data over the internet has made the problem of data analysis and interpretation a really challenging task. Deep Learning has proved itself to be a possible solution to such Computer Vision tasks. Not only in Computer Vision, Deep Learning techniques are also widely applied in Natural Language Processing tasks. In this course we will start with traditional Machine Learning approaches, e.g. Bayesian Classification, Multilayer Perceptron etc. and then move to modern Deep Learning architectures like Convolutional Neural Networks, Autoencoders etc. On completion of the course students will acquire the knowledge of applying Deep Learning techniques to solve various real life problems.

Text summarization is a process of extracting the context of a large document and summarize it into a smaller paragraph or a few sentences. Text summarization plays a vital role in saving time in our day to day life. It is also used in many bigger project implementations of classification of documents or in search engines. This paper presents a method of achieving text summaries accurately using deep learning methods.

Table of Contents

TITLE NO	PAGE
ABSTRACT	i
LIST OF FIGURES	iii
1. Introduction	1
1.1 Extractive Summarization	2
1.2 Abstractive Summarization	3
1.3 Natural Language Processing	4
1.4 Neural Machine Translation	4
1.5 Deep Learning	6
1.6 Artificial Neural Network	6
1.6.1 Single Layer Perceptron	7
1.6.2 Multi Layered Perceptron	7
1.6.3 Recurrent neural network	9
1.6.4 Convolutional neural network	11
2. Aim and Scope	13
2.1 Summarization method using Fuzzy Rules	13
2.2 Deep Learning approaches for Summarization	14
2.3 Summarization using Multi-Layered network	14
2.4 Methods Using Deep Learning	15
2.5 Existing Methods of Text Summarization	16
3. Methodology	18
3.1 Proposed System	18
3.1.1 Architecture	18
3.1.2 Detailed understanding of Architecture	19
4. Experimental Analysis and Results	24
4.1 System Configuration	24
4.1.1 Software Requirements	24
4.1.2 Hardware Requirements	25
4.2 Sample Code	27
4.3 Screenshots	
5. Summary and Conclusions	34
REFERENCES	36

LIST IF FIGURES

Fig no no	Topic name	Page
Fig 1.1	Extractive Summarization	2
Fig 1.2	Abstractive Summarization	3
Fig 1.3	Neural Machine Translation	5
Fig 1.4	Single Layer Perceptron	7
Fig 1.5	Architecture of multi Layer perceptron	9
Fig 1.6	Architecture of LSTM	11
Fig 1.7	Convolutional neural network	12
Fig 3.1	Neural network Architecture	22

CHAPTER 1

INTRODUCTION

Text summarization is a process of producing brief and concise summary by capturing the vital information and the comprehensive meaning. Text summarization is achieved by natural language processing techniques by using algorithms like page rank algorithms etc. While these algorithms fulfil the objective of text summarization, they cannot generate new sentences which are not in the document like humans. They can also have grammatical errors. This is where Deep Learning comes to our rescue. The use of deep learning builds an efficient and fast model for text summarization. The use of deep learning methods helps us generate summaries which can be formed with new phrases and sentences and also which are grammatically correct.

There is an enormous amount of textual material, and it is only growing every single day.

Think of the internet, comprised of web pages, news articles, status updates, blogs and so much more. The data is unstructured and the best that we can do to navigate it is to use search and skim the results.

There is a great need to reduce much of this text data to shorter, focused summaries that capture the salient details, both so we can navigate it more effectively as well as check whether the larger documents contain the information that we are looking for.

6 reasons why we need automatic text summarization tools.:

1. Summaries reduce reading time.
2. When researching documents, summaries make the selection process easier.
3. Automatic summarization improves the effectiveness of indexing.
4. Automatic summarization algorithms are less biased than human summarizers.
5. Personalized summaries are useful in question-answering systems as they provide personalized information.
6. Using automatic or semi-automatic summarization systems enables commercial abstract services to increase the number of texts they are able to process.

Text Summarization is broadly classified into two types:

1. Abstractive Summarization
2. Extractive Summarization

1.1 Extractive Summarization

The extractive text summarization involves pulling key phrases from the source document and combining them to make a summary. We identify important words or phrases from the text and extract only those for the summary.

Techniques involve ranking the relevance of phrases in order to choose only those most relevant to the meaning of the source.



Fig-1.1: Extractive Text Summarization

Classically, most successful text summarization methods are extractive because it is an easier approach, but abstractive approaches hold the hope of more general solutions to the problem.

Example :

Source text: DataFlair is an online, immersive, instructor-led, self-paced technology school for students around the world. DataFlair offers lifetime support, quizzes to sharpen student's knowledge, and various live project participation. DataFlair machine learning projects are best for students to gain practical knowledge for realworld problems.

Summary: DataFlair is an online school for students around the world. DataFlair offers lifetime support, quizzes, and live projects. DataFlair machine Learning projects are best to gain knowledge for real-world problems.

1.2 Abstractive Summarization

Abstractive text summarization involves generating entirely new phrases and sentences to capture the meaning of the source document. This is a more challenging approach, but is also the approach ultimately used by humans. Classical methods operate by selecting and compressing content from the source document.

The abstractive text summarization can create new phrases and sentences that relay the most useful information from the original text. The sentences generated through this method may not be present in the original document.

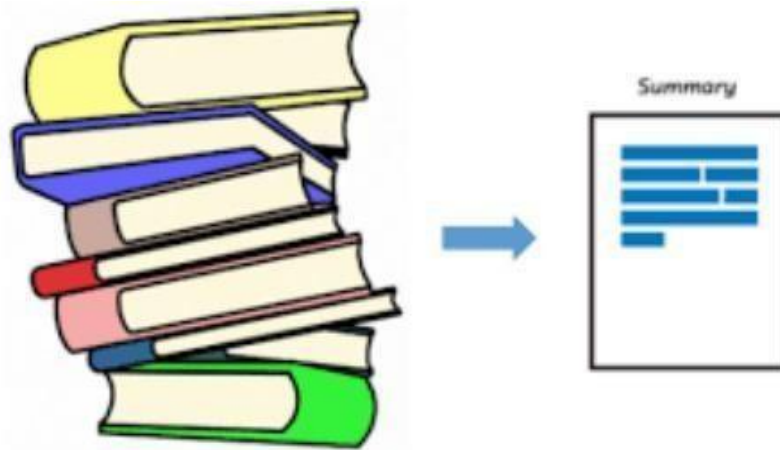


Fig-1.2: Abstractive Text Summarization

Example :

Source text: DataFlair is an online, immersive, instructor-led, self-paced technology school for students around the world. DataFlair offers lifetime support, quizzes to sharpen student's knowledge, and various live project participation. DataFlair machine learning projects are best for students to gain practical knowledge for realworld problems.

Summary: DataFlair is an online school where students are offered various quizzes and projects including machine learning to solve real-world problems.

1.3 NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is all about computer and human languages interactions. A finest procedure that helps the system in reading and understanding the given text. Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software. Data generated from conversations, declarations or even tweets are examples of unstructured data. Unstructured data doesn't fit neatly into the traditional row and column structure of relational databases and represent most data available in the actual world. It is messy and hard to manipulate. Nevertheless, thanks to the advances in disciplines like machine learning a big revolution is going on regarding this topic.

Nowadays it is no longer about trying to interpret a text or speech based on its keywords (the old-fashioned mechanical way), but about understanding the meaning behind those words (the cognitive way). This way it is possible to detect figures of speech like irony, or even perform sentiment analysis. It is a discipline that focuses on the interaction between data science and human language and is scaling to lots of industries. Today NLP is booming thanks to the huge improvements in the access to data and the increase in computational power, which are allowing practitioners to achieve meaningful results in areas like healthcare, media, finance and human resources, among others.

With the rapid increase in data the text summarization manually consumes a lot of time. That resulted in development of various summarization methods. Using them, the system should deliver a précis with the main essence of the document in reduced size. Text summarization is of two types: i) extractive ii) abstractive. Firstly, the extractive text summarization where important sentences and words from the given text document are identified and those are combined into the summary in a meaningful way. Finally, in abstractive text summarization, new sentences are fabricated, and the summary is given without the loss of information which is understandable. This can be done using different methods that include statistical measures, deep learning techniques with supervised or unsupervised learning and word vector embedding.

1.4 NEURAL MACHINE TRANSLATION

Neural Machine Translation[13] is the approach of modelling this entire process via one big artificial neural network, known as a Recurrent Neural Network (RNN). RNN is a stateful neural network, in which it has connections between passes, connections through time. Neurons are fed information not just from the previous layer but also from themselves from the previous pass. This means that the order in which we feed the input and train the network matters: feeding it "Donald" and then "Trump" may yield different results compared to feeding it "Trump" and then "Donald".

Neural Machine Translation is a machine translation approach that applies a large artificial neural network toward predicting the likelihood of a sequence of words, often in the form of whole sentences. Unlike statistical machine translation, which consumes more memory and time, neural machine translation, NMT, trains its parts end-to-end to maximize performance. NMT systems are quickly moving to the forefront of machine translation, recently outcompeting traditional forms of translation systems.

Machine translation is the task of automatically converting source text in one language to text in another language. Given a sequence of text in a source language, there is no one single best translation of that text to another language. This is because of the natural ambiguity and flexibility of human language. This makes the challenge of automatic machine translation difficult, perhaps one of the most difficult in artificial intelligence. Classical machine translation methods often involve rules for converting text in the source language to the target language. The rules are often developed by linguists and may operate at the lexical, syntactic, or semantic level. This focus on rules gives the name to this area of study: Rule-based Machine Translation, or RBMT. The key limitations of the classical machine translation approaches are both the expertise required to develop the rules, and the vast number of rules and exceptions required.

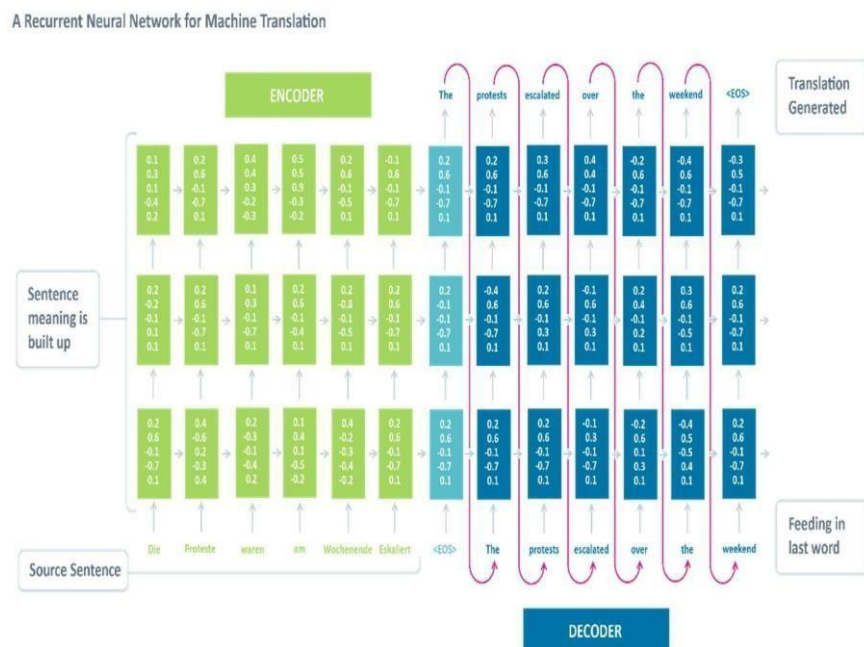


Fig-1.3 A Recurrent neural network for Machine Translation

1.5 DEEP LEARNING

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers.

There are two main reasons it has only recently become useful:

- Deep learning requires large amount of labelled data. For example, driverless car development requires millions of images and thousands of hours of video.
- Deep learning requires substantial computing power. High performance GPU have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

There are many applications of deep learning in different fields:

- Automatic driving
- Aerospace and defence
- Medical research
- Industrial automation

1.6 Artificial Neural Network

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites.

The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.

ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The

nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value.

Each link is associated with weight. ANNs are capable of learning, which takes place by altering weight values.

1.6.1 Single Layer Perceptron

Single layer perceptron is the first proposed neural model created. The content of the local memory of the neuron consists of a vector of weights. The computation of a single layer perceptron is performed over the calculation of sum of the input vector each with the value multiplied by corresponding element of vector of the weights. The value which is displayed in the output will be the input of an activation function.

Neural networks are the core of deep learning, a field which has practical applications in many different areas. Today neural networks are used for image classification, speech recognition, object detection etc. Now, Let's try to understand the basic unit behind all this state of art technique.

A single neuron transforms given input into some output. Depending on the given input and weights assigned to each input, decide whether the neuron fired or not. Let's assume the neuron has 3 input connections and one output.

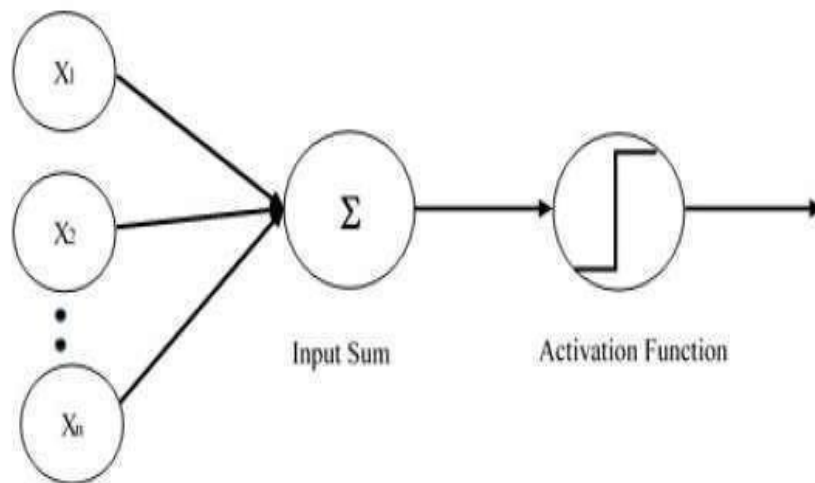


Fig 1.4 Single Layer Perceptron

1.6.2 Multi Layered Perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to refer to any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptron (with threshold activation); see &Terminology. Multilayer perceptron are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

A multilayer perceptron (MLP) is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that decides or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer can approximate any continuous function.

Multilayer perceptron's are often applied to supervised learning problems³: they train on a set of input output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (RMSE).

Feedforward networks such as MLPs are like tennis, or ping pong. They are mainly involved in two motions, a constant back and forth. You can think of this ping pong of guesses and answers as a kind of accelerated science, since each guess is a test of what we think we know, and each response is feedback letting us know how wrong we are.

In the forward pass, the signal flow moves from the input layer through the hidden layers to the output layer, and the decision of the output layer is measured against the ground truth labels.

In the backward pass, using backpropagation and the chain rule of calculus, partial derivatives of the error function with respect to the various weights and biases are backpropagated through the MLP. That act of differentiation gives us a gradient, or a landscape of error, along which the parameters may be adjusted as they move the MLP one step closer to the error minimum. This can be done with any gradientbased optimisation algorithm such as stochastic gradient descent. The network keeps playing that game of tennis until the error can go no lower. This state is known as convergence.

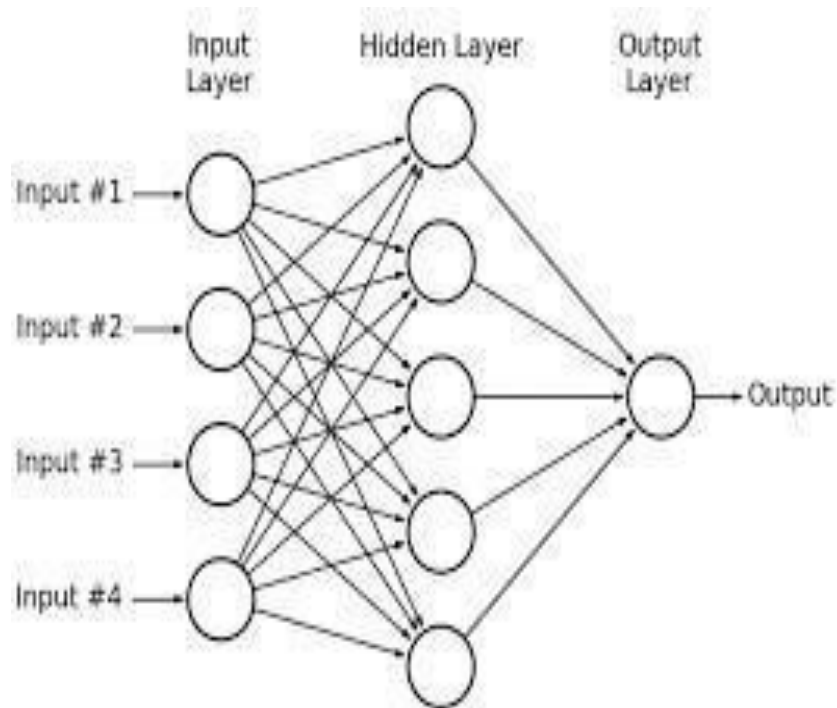


Fig 1.5 Architecture of multi layered perceptron

1.6.3 Recurrent neural network

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs.[1] This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behaviour. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.

Recurrent Neural Network remembers the past and it’ s decisions are influenced by what it has learnt from the past. Note: Basic feed forward networks “remember” things too, but they remember things they learnt during training. For example, an image classifier learns what a “1” looks like during training and then uses that knowledge to classify things in production.

It's part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series.

In summary, in a vanilla neural network, a fixed size input vector is transformed into a fixed size output vector. Such a network becomes "recurrent" when you repeatedly apply the transformations to a series of given input and produce a series of output vectors. There is no pre-set limitation to the size of the vector. And, in addition to generating the output which is a function of the input and hidden state, we update the hidden state itself based on the input and use it in processing the next input.

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So, if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning. During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural network's weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning. So, in recurrent neural networks, layers that get a small gradient update stop learning. Those are usually the earlier layers. So, because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory.

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behaviour required in complex problem domains like machine translation, speech recognition, and more. LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field. There are few that are better at clearly and precisely articulating both the promise of LSTMs and how they work than the experts that developed them. So, in recurrent neural networks, layers that get a small gradient update stop learning. Those are usually the earlier layers. So because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory.

These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state-of-the-art results based on recurrent neural

networks are achieved with these two networks. LSTM's and GRU's can be found in speech recognition, speech synthesis, and text generation. You can even use them to generate captions for videos.

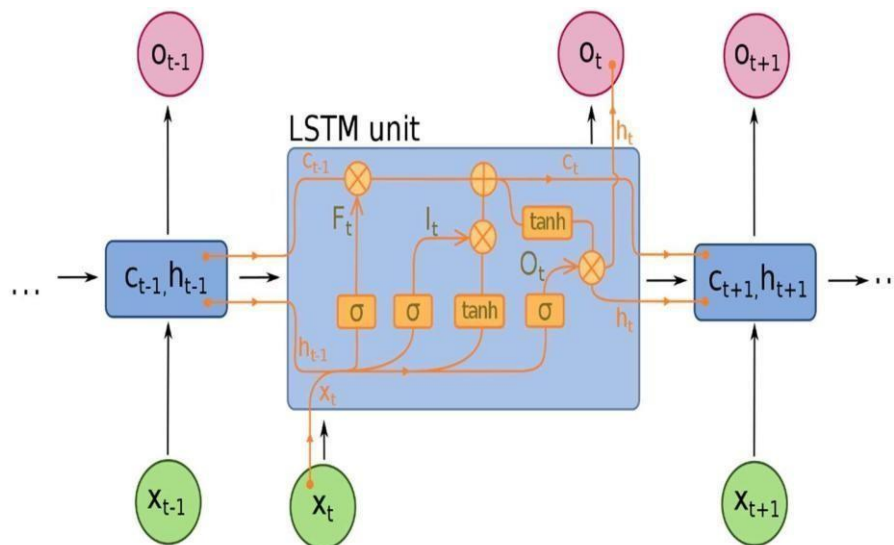


Fig 1.6 Architecture of LSTM

1.6.4 Convolutional neural network

CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually means fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fullyconnectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes[6][7][8][9] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification,

segmentation and for other auto correlated data. A convolution is essentially sliding a filter over the input.

The most common use for CNNs is image classification, for example identifying satellite images that contain roads or classifying hand-written letters and digits. There are other quite mainstream tasks such as image segmentation and signal processing, for which CNNs perform well at. A CNN can also be implemented as a U-Net architecture, which are essentially two almost mirrored CNNs resulting in a CNN whose architecture can be presented in a U shape. U-nets are used where the output needs to be of similar size to the input such as segmentation and image improvement.

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. These are linear transformations; each convolution is a type of affine function.

In computer vision the input is often a 3 channel RGB image. For simplicity, if we take a greyscale image that has one channel (a two-dimensional matrix) and a 3x3 convolutional kernel (a two-dimensional matrix). The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the images pixel values. Then the kernel strides down vertically to subsequent rows. Note, the filter may stride over one or several pixels at a time, this is detailed further below. In other non-vision applications, a onedimensional convolution may slide vertically over an input matrix.

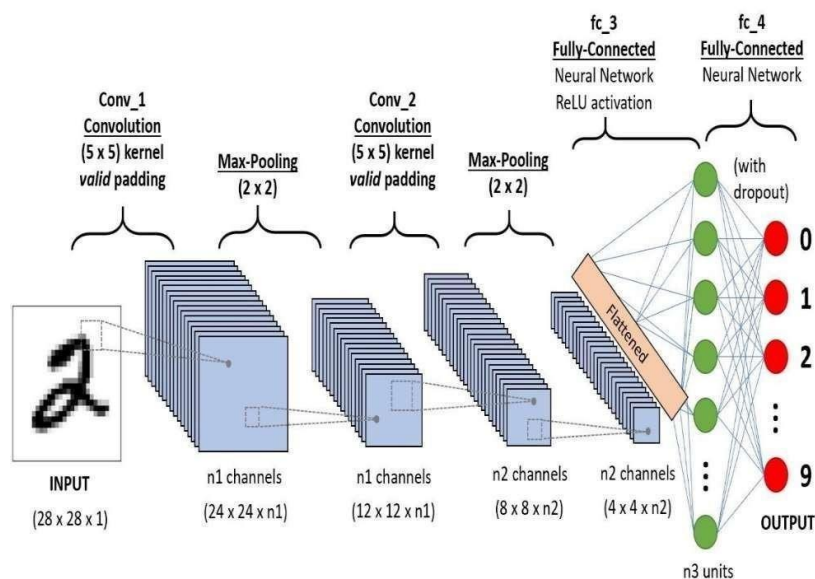


Fig 1.7 Architecture of Convolutional neural network

CHAPTER 2

AIM AND SCOPE

Text summarization has become one of the hottest topics in Text mining as there is more amount of text that is produced on the internet. So, has become so difficult to read all the content due to the busy schedules and as the time has become the more important constraint in everyone' s life. To make the text content simpler to read, there came an idea of summarizing the text by giving only the important information from the available text articles. The sports are the most admiring topics in news, and everyone wants to know about the sports news and due to the time constraint, they couldn' t read the whole news which contains only a few important information. So, we have chosen to summarize the sports articles.

Generating a summary from a text document. So that it helps in understanding a huge text data. With the availability of internet at every corner of the world, the amount of information is growing rapidly at certain rate. Considering the hectic schedule of the people and the need for information abstraction, here comes the summarization of the News Articles. Every domain in the news has its own type of information important. So, as it is difficult to summarize all types of news data. We are limiting our problem to extractive summarize the sports genre in news articles.

2.1 SUMMARIZATION METHOD USING FUZZY RULES

A method using fuzzy rules to retrieve the most important sentences in a document. The text summarization task has gained much importance because of the large amount of online data, and its potential to extract useful information and knowledge in a way that could be easily handled by humans and used for a myriad of purposes, including expert systems for text assessment. This paper presents an automatic process for text assessment that relies on fuzzy rules on a variety of extracted features to find the most important information in the assessed texts. The automatically produced summaries of these texts are compared with reference summaries created by domain experts. Differently from other proposals in the literature, our method summarizes text by investigating correlated features to reduce dimensionality, and consequently the number of fuzzy rules used for text summarization. Thus, the proposed approach for text summarization with a relatively small number of fuzzy rules can benefit development and use of future expert systems able to automatically assess writing. The proposed summarization method has been trained and tested in experiments using a dataset of Brazilian Portuguese texts provided by students in response to tasks assigned to them in a Virtual Learning Environment (VLE). The proposed approach was compared with other methods including a naive baseline, Score, Model and Sentence, using ROUGE measures. The results show that the proposal provides better f-measure (with 95%

CI) than methods. This paper presented a method that uses fuzzy metrics to determine the most informative sentences in the text, and then compare and classify text written in response to learning tasks with reference responses for these tasks. Fuzzy logic was used because it is a widely recommended approach for applications that handle uncertain information (e.g., ambiguity in linguistic terms). We show that fuzzy logic associated with text summarization can represent the main concepts in natural language on student's assessment. The observed percentile is 73 percentiles.

2.2 DEEP LEARNING APPROACHES FOR SUMMARIZATION OF LEGAL DOCUMENTS

A method for summarizing the Indian legal judgment documents for this the author has used a neural network approach that is semi-supervised. The availability of legal judgment documents in digital form offers numerous opportunities for information extraction and application. Automatic summarization of these legal texts is a crucial and a challenging task due to the unusual structure and high complexity of these documents. Previous approaches in this direction have relied on huge labelled datasets, using hand engineered features, leveraging on domain knowledge and focussed their attention on a narrow subdomain for increased effectiveness. In this paper, we propose simple generic techniques using neural network for the summarization task for Indian legal judgment documents. We explore two neural network architectures for this task utilizing the word and sentence embeddings for capturing the semantics. The main advantage of the proposed approaches is that they do not rely on hand crafted features, or domain specific knowledge, nor is their application restricted to a particular sub-domain thus making them suitable to be extended to other domains as well. We tackle the problem of unavailability of labelled data for the task by assigning classes/scores to sentences in the training set, based on their match with reference summary produced by humans. The experimental evaluations establish the effectiveness of our proposed approaches as compared with other baselines. A full Python Implementation of the ROUGE Metric (not a wrapper). Retrieved from in this paper we presented a data driven semi-supervised approach to extractive legal document summarization using. The observed percentile is 63 percentage.

2.5 SUMMARIZATION USING MULTI-LAYERED NETWORK

A multi-layered neural network model to generate the extractive summary based on the sentences having high importance in the context for multi-documents. A graph which consists of nodes and edges is built with sentences and words respectively. The datasets used are CST News for Portuguese multi-document summarization and DUC-2002, DUC-2004 for English multidocument summarization. The word embedding feature is lacking in this work for generating a better summary. Huge

volumes of textual information have been produced every single day. In order to organize and understand such large datasets, in recent years, summarization techniques have become popular. These techniques aim at finding relevant, concise and non-redundant content from such a big data. While network methods have been adopted to model texts in some scenarios, a systematic evaluation of multilayer network models in the multi-document summarization task has been limited to a few studies. Here, we evaluate the performance of a multilayer-based method to select the most relevant sentences in the context of an extractive multi document summarization (MDS) task. In the adopted model, nodes represent sentences and edges are created based on the number of shared words between sentences. Differently from previous studies in multi- document summarization, we make a distinction between edges linking sentences from different documents (inter-layer) and those connecting sentences from the same document (intra-layer). As a proof of principle, our results reveal that such a discrimination between intra- and interlayer in a multi-layered representation is able to improve the quality of the generated summaries. This piece of information could be used to improve current statistical methods and related textual models. The main contribution of this paper consisted in using a multilayer representation to address the multi- document summarization problem. While networked representations of texts have been used to model documents, to our knowledge, no previous study has used the multilayer approach to address this type of problem. The main advantage of the proposed technique is that it does not depend on any source of external information, such as lexical datasets or other deeper resources. As the main result, we showed – as proof of principle – that different weighting schemes for inter-layer edges can better represent a set of documents. Such an improvement in the model turned out to increase the informativeness of the generated summaries.

2.3 METHODS USING DEEP LEARNING

An approach using both Restricted Boltzmann machine and Fuzzy logic to identify the important sentences. An approach for generating short and precise summaries for long text documents is proposed. Lately, the size of information on the internet is increasing. It has become tough for the users to dig into the loads of information to analyse it and draw conclusions. Text summarization solves this problem by generating a summary, selecting sentences which are most important from the document without losing the information. In this work, an approach for Extractive text summarization is designed and implemented for single document summarization. It uses a combination of Restricted Boltzmann Machine and Fuzzy Logic to select important sentences from the text keeping the summary meaningful and lossless. The text documents used for summarization are in English language. Various sentence and word level features are used to provide meaningful sentences. Two summaries for each document are generated using Restricted Boltzmann Machine and Fuzzy logic. Both summaries are then combined and processed using a set of

operations to get the final summary of the document. The results show that the designed approach overcomes the problem of text overloading by generating an effective summary. In this work, RBM is used as an unsupervised learning algorithm along with fuzzy logic for improving the accuracy of the summary. It is observed that the proposed approach generates short and precise summaries without any irrelevant text. Using features like Sentence-Centroid similarity and thematic words has improved the connectivity of the sentences. Using the proposed method, on an average 88% precision, 80% recall and 84% F measure is obtained. The results produced using the proposed method give better evaluation parameters in comparison with prevailing RBM method. The proposed method can be extended for multi document summarization. Documents in different languages can be summarized. Various other features can be used, and the method can be combined with other methods for improving the nature of the summary. Also, it can be used in Abstractive text summarization.

2.4 EXISTING METHODS OF TEXT SUMMARIZATION

2.4.1 Ranking of sentences using Cosine similarity

Cosine similarity is a metric used to determine how similar the documents are irrespective of size. Mathematically, it measures the cosine of the angle between the two vectors projected in a multi-dimensional space.

This is advantageous because even if two similar documents are far apart by Euclidian distance, they could have still had a similar smaller angle between them. Smaller angle means high similarity.

$$\text{cosine similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Example:

D1: the best Italian restaurant enjoy the best pasta

D2: American restaurant enjoy the best hamburger D3: Korean restaurant enjoy the best bibimbap D4: the best the best American restaurant

Using Cosine similarity: (comparing similarity with D4)

D1: 0.82

D2: 0.77

D3: 0.65

D4: 1

2.4.2 Multi document text summarization using TF-IDF

TF-IDF stands for term frequency-inverse document frequency. This is a statistical measure used to evaluate how important a word is to a

document in collection of corpora. The importance increases proportionally to the number of times a word appears in the document but the offset by the frequency of the word in the corpus.

- TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization.

$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$.

- IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$IDF(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it})$. D1:
0

D2: 0.5

D3: 0

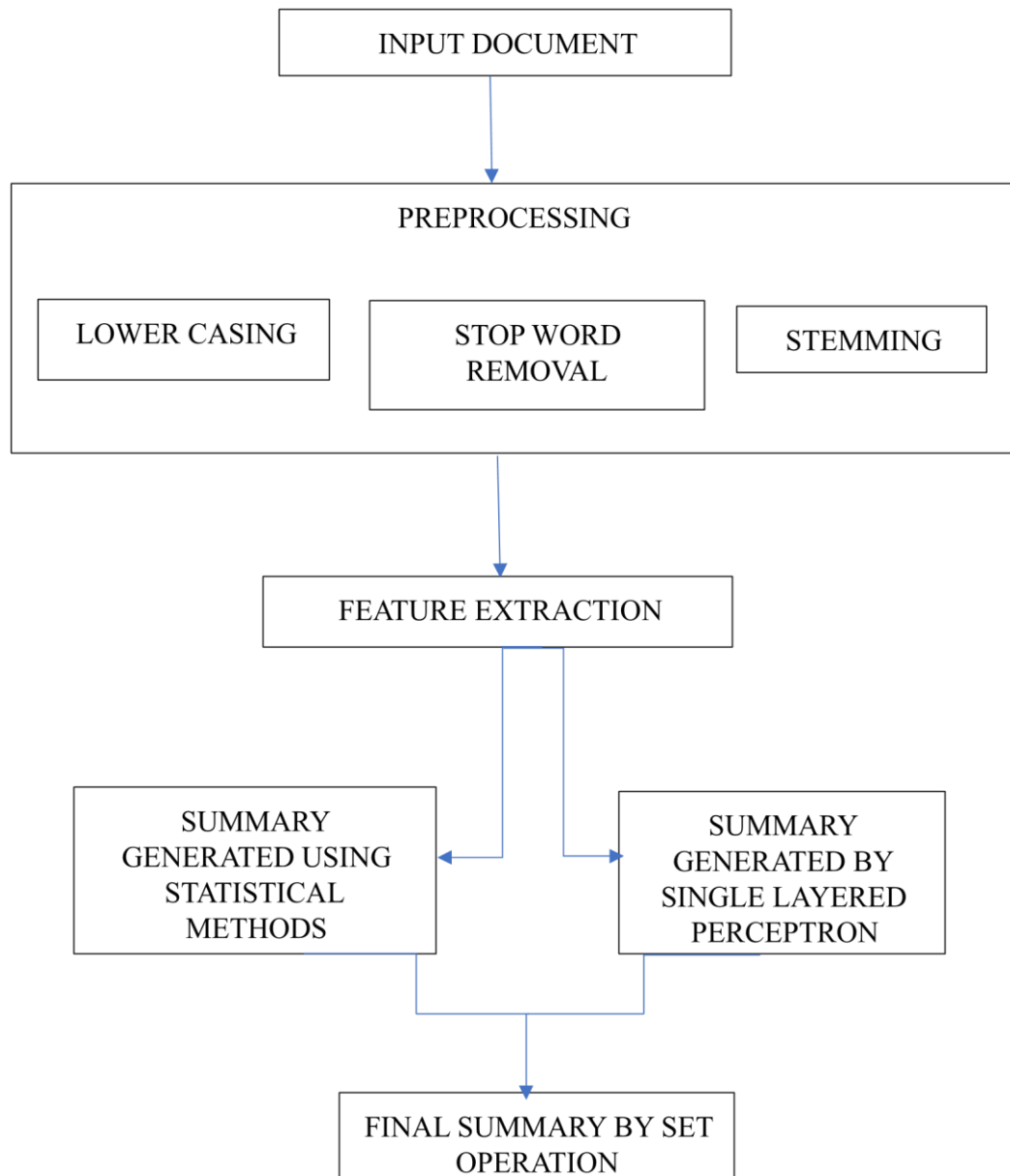
D4: 1

CHAPTER 3

METHODOLOGY

3.1 PROPOSED SYSTEM

3.1.1 ARCHITECTURE



3.1.2 Detailed Understanding of architecture:

1) Input Document:

To generate a summary some input should be given in the file format “.txt” . The text documents of sports articles are considered for summarization and are valid since the author has proposed a method for summarizing sports articles. The input document should only be in English language. The size of the input document can be large as well.

2) Pre-processing:

The input text is divided into sentences based on the sentence terminator. These sentences are individually preprocessed using the below three techniques: a. Lower casing:

In the lower casing, the whole document is converted into lowercase alphabetical letters. This is done to remove the ambiguity of the words with different casing. This is done using the lower() of python language which is available inbuilt. There is another method also, considering each character and its ascii values. Add the required value to convert to lower case. b. Stop word removal:

The stop words in the document are removed from the document because these are the most frequent words like articles, prepositions, conjunctions which don't add any value in defining the importance of a sentence. The stop words are removed using nltk (natural language tool kit) library. All the stop words are stored in a list and each word is compared with the words present in the list. If the word is present in the list it is removed from the document, else it is remained unaltered. c. Stemming:

Stemming is referred as getting the words reduced to their root form. It removes the different forms of the same word present in different sentences. In this porters stemmer is used.

3) Feature Extraction: a. TF-ISF:

Term frequency and inverse document frequency (TF-IDF) are used for information retrieval systems. In this paper, the summarization is done for a single document of sports article and Term frequency inverse sentence frequency (TF-ISF) is calculated as follows.

$$TF - ISF(i, j) = TF(j) * \log () \quad (1) ()$$

(,)

$$TF - ISF(i) = \quad (2)$$

Where TF_j indicates the term frequency of a word in a sentence, ISF_j is the inverse sentence frequency of a jth word in a sentence, TF-ISF_i(2) is the Term frequency and inverse sentence frequency of ith sentence in a document, n cites to the number of words in a sentence, N cites to the number of sentences in a document.

b. Sentence length:

This feature provides the less weightage to short sentences as short sentences are relatively less important when compared to long sentences. It is measured by calculating the ratio of no of words in the sentence to the no of words in the longest sentence of the document.

This feature's value is calculated using eq. (3).

$$sentence\ length = \quad (3)$$

c. Sentence position:

In a document the numerical value of the sentence's position is gauged (4). The position is evaluated as the normalized percentile score in the range of 0 to 1.

$$position = \begin{cases} 0, & \text{if first sentence} \\ 1, & \text{if last sentence} \end{cases} \quad sentence \quad (4)$$

$$, \quad \text{if } 0 < p < n$$

p is the position of the current sentence and n is

the total number of sentences. d. Cosine similarity:

Analogy between sentences is calculated using this property. This briefs how much similar this sentence with other sentences in a document is. A vector for each sentence is assessed and (5) is practiced obtaining the score of the feature.

$$cosine\ similarity = \quad (5)$$

A is the first vector and B is the second vector with which it is compared. e.

Existence of proper noun:

Proper names broaching to places and people might be useful to decide the relevance of a sentence. This binary feature, with value '1' if a sentence contains these proper names and 0 otherwise.

For Example: New York, Monday, Johnathan, etc. f.

Existence of pronoun:

Pronouns cites to the persons which are described in the previous sentences. This binary feature, with value '1' if a sentence contains these proper names and 0 otherwise.

For Example: He, she, it, they.

g. Sentence containing scores:

Since in sports the score of the team or the individual participant is important the score is taken into consideration. This binary feature, with value '1' if a sentence contains these proper names and 0 otherwise.

4) Summary generated using Statistical methods: The mean, median and mode are generally some statistical tools used. The mean is used in many cases. Firstly, the input document is divided in to sentences and then the sentences are divided into words. Secondly, the preprocessing which consists of lower casing, stop word removal and stemming is done and stored in a list.

Then TF-ISF for each word is calculated using equation (1) and (2). But we need the TF-ISF for each sentence. So, for a sentence there may be 'n' words then we get 'n' different values, we calculate the mean of the 'n' values to obtain total TF-ISF for the sentence.

Sentence length is calculated using (3). We get a value for each sentence.

Sentence position is calculated using (4). We get a value for each sentence.

Cosine similarity is calculated by converting the words to vectors and each word has a unique vector. The similarity between the sentence is calculated using equation (5). Like TF-ISF each word has its own similarity value. To obtain a value for each sentence we calculate the mean of all the words of a sentence.

Based on all these seven-feature values for each sentence. We take the mean of TF-ISF and cosine similarity of each sentence and considering the sentence position, sentence length, existence of proper noun, existence of pronoun, and a sentence with score. We give each sentence a predictive score which is also called as the rank of the sentence.

Compression ratio is defined as the percentage of the original input should be present in the summary.

All the predictive scores of sentences are sorted in descending order. The sentence with highest predictive score is taken and added to the summary in the order of the original document to not loose the context of what author is trying to convey.

The final summary for the compression ratio is stored and the sentence number is also stored along with it for easy identification of the sentence.

5) Summary generated using Single layered perceptron:

The single layered perceptron is used as a neural network. It consists of one input layer, one hidden layer and an output layer. Input layer has 7 nodes and output layer has one node. The values of the features become the input to the neural network which is multiplied by weights and it is passed through activation function, sigmoid is chosen and we get a value for each sentence and it becomes the predictive score of a sentence.

Weight matrix= [0.8, 0.6, 0.4, 0.2, 0.3, 0.5, 0.7] For each sentence:

Input: [x1, x2, x3, x4, x5, x6, x7] Weight: [w1, w2, w3, w4, w5, w6, w7] Sum:

$$(x1*w1+x2*w2+x3*w3+x4*w4+x5*w5+x6*w6+x7*w7)$$

1

$a =$

$$1 + \text{pow}(e, -\text{sum})$$

The value of 'a' is sorted and the sentences with highest value for given compression ratio is given as summary.

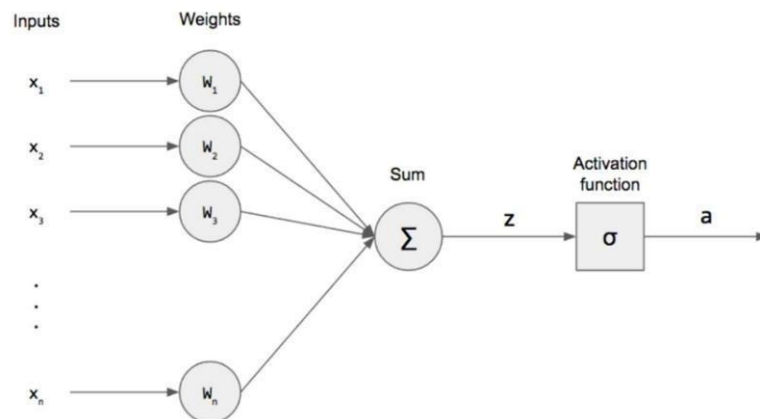


Fig 3.1 Neural network architecture (Single Layered perceptron)

The summary is stored along with the number of the sentence stored.

6) Final summary by set operation:

The common sentences are taken from both the techniques and the summary is given based on the compression ratio considering it to be 50%.

CHAPTER 4

EXPERIMENTAL ANALYSIS AND RESULTS

4.1 System Configuration 4.1.1

Software Requirements

a. Python:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable.

Characteristics of Python:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to bytecode for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.

Reasons why python is used:

- Readable and Maintainable code
- Multiple programming paradigms
- Compatible with major platforms and systems
- Robust standard library
- Many open source frameworks and tools
- Simplify complex software development
- Adopt test driven development
- Highly productive compared to other programming languages like C++ and Java.

b. Jupyter Notebook:

Jupyter Notebook (formerly IPython Notebook) is a web-based interactive computational environment for creating notebook documents. Jupyter Notebook is built using several open-source libraries, including IPython, ZeroMQ, Tornado, jQuery, Bootstrap, and MathJax. A Jupyter Notebook application is a browser-based REPL containing an ordered list of input/output cells which can contain code, text (using Github Flavored Markdown), mathematics, plots and rich media.

Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s. Jupyter interest overtook the popularity of the Mathematica notebook interface in early 2018.

JupyterLab is a newer user interface for Project Jupyter, offering a flexible user interface and more features than the classic notebook UI. The first stable release was announced on February 20, 2018. In 2015, a joint \$6 million grant from The Leona M. and Harry B. Helmsley Charitable Trust, The Gordon and Betty Moore Foundation, and The Alfred P. Sloan Foundation funded work that led to expanded capabilities of the core Jupyter tools, as well as to the creation of JupyterLab.

GitHub announced in November 2022 that JupyterLab would be available in its online Coding platform called Codespace.

JupyterHub is a multi-user server for Jupyter Notebooks. It is designed to support many users by spawning, managing, and proxying many singular Jupyter Notebook servers.

4.1.2 Hardware Requirements

a. Processor:

The first trade-off we came across was which chip to buy? i7 processors seemed to have lesser cores compared to Xeon processors, but higher frequency. i7 processors are also very easy to overclock, which is not the case with Xeon. Xeon on the other hand, supported much higher RAM – i7 maxed out at 128 GB RAM (that too was available on a single model). Also, the power consumption of Xeon is lower compared to an i7. We started shortlisting possible processor chips, which would fit our requirement. We came down to a shortlist of a few i7 and Xeon processors. We decided to go for higher cores, power efficiency and possibility to expand more RAM in future over high frequency i7 chips. After searching for prices of various chips and their benchmarks, we finally zeroed in on Intel Xeon E2630 v4

. b. RAM:

We bought 128 GB RAM for now. We can obviously add more, if required. This should be good for now. Needless to say DDR4 is better than DDR3 and 2400 MHz is better than 2133 MHz. We ended up with 2133 MHz to save a few bucks.

c. Motherboard:

Once the processor is finalized, we looked for compatible motherboards. Since we wanted to use this machine for deep learning, we had to make sure that the motherboard had enough PCIe slots. We wanted to have the option to add up to 2 GPUs in future. Additional things to look out for included the number of RAM slots, which would directly impact the amount of RAM you can have. You can also look at whether the motherboard supports one socket or 2 sockets. Having 2 sockets is more desirable as you can add one more processor to the box later. We zeroed in on

ASRock EPC612D8A – while it had only one socket, it gave us option to expand RAM further and add more GPUs, if needed in future. d. GPU:

Given the evolution in deep learning, we knew that we had to invest in the best in class GPU. The options include NVidia GTX 1080, NVidia Tesla K40. Again, we went to the basics on why we need GPUs. For deep learning, we do not need high precision computations, so the expensive Tesla K series went out of consideration. Among the two remaining Titan X Pascal clearly looked far superior for deep learning. There was only one problem – this card was not available in India yet. I had to ask one of my friends in the US to get it for us. e. Hard disk:

The first thing we did was to buy a SSD and a hard disk. The idea was to do the frequent read write operations on the SSD, while keep the large datasets on the normal hard disk. We also decided to not go with RAID based hard disks as they were adding to the costs and most of the time, we have backups of the data anyways. So, we took a 512 GB SSD + 2 TB HDD (7200 RPM) f. Power:

We added a 1000W Corsair SMPS to make sure that the GPU and the motherboard get ample power. g. Cooling:

Cooling is very important for a workstation like this. Ideally, liquid based cooling systems are the most effective. However, we could not find a compatible model. So, we went with Intel Heatsink which was the next best option. I was a bit sceptical whether this would be sufficient or not. However, we have not faced any heating problems till now. The Fan on GPU, heat sink and 2 of them on CPU have been effective to keep the temperature under control.

4.2 SAMPLE CODE

Importing required libraries

```
from sklearn.feature_extraction.text import  
TfidfVectorizer from nltk.tokenize import word_tokenize,  
sent_tokenize from spacy.lang.en import English import  
numpy as np
```

Load spacy model for sentence tokenization

```
nlp = English() nlp.add_pipe('sentencizer')  
#nlp.add_pipe(nlp.create_pipe('sentencizer'))  
<spacy.pipeline.sentencizer.Sentencizer at 0x1ee71361ec0> text_corpus  
= """
```

One day, a fox became very hungry as he went to search for some food. He searched high and low, but couldn't find something that he could eat. Finally, as his stomach rumbled, he stumbled upon a farmer's wall. At the top of the wall, he saw the biggest, juiciest grapes he'd ever seen. They had a rich, purple color, telling the fox they were ready to be eaten. To reach the grapes, the fox had to jump high in the air. As he jumped, he opened his mouth to catch the grapes, but he missed. The fox tried again but missed yet again. He tried a few more times but kept failing. Finally, the fox decided it was time to give up and go home. While he walked away, he muttered, "I'm sure the grapes were sour anyway."

```
"""
```

Create spacy document for further sentence level tokenization

```
doc = nlp(text_corpus.replace("\n", "")) doc_sentences =  
sent_tokenize(text_corpus)  
#print(doc_sentences) def summarizer(text, tokenizer,  
max_sent_in_summary=3): # Create spacy document for  
further sentence level tokenization doc =  
nlp(text_corpus.replace("\n", "")) sentences =
```

```
[sent.string.strip() for sent in doc.sents]
```

Peeking into our tokenized sentences

```
print("Senetence are: \n",doc_sentences) Senetence are:
```

```
['\nOne day, a fox became very hungry as he went to search for some food.', 'He searched high and low, but couldn\' t find something that he could eat.Finally, as his stomach rumbled, he stumbled upon a farmer\' s wall.', 'At the top of the wall, he saw the biggest, juiciest grapes he\' d ever seen.', 'They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the fox had to jump high in the air.', 'As he jumped, he opened his mouth to catch the grapes, but he missed.', 'The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fox decided it was time to give up and go home.', 'While he walked away, he muttered, "I\' m sure the grapes were sour anyway." ']
```

Creating sentence organizer

```
# Let's create an organizer which will store the sentence ordering to later reorganize the
```

```
# scored sentences in their correct order
```

```
sentence_organizer = {k:v for v,k in enumerate(doc_sentences)}
```

Peeking into our sentence organizer

```
print("Our sentence organizer: \n", sentence_organizer) Our
```

sentence organizer:

```
{'\nOne day, a fox became very hungry as he went to search for some food.': 0, 'He searched high and low, but couldn\' t find something that he could eat.Finally, as his stomach rumbled, he stumbled upon a farmer\' s wall.': 1, 'At the top of the wall, he saw the biggest, juiciest grapes he\' d ever seen.': 2, 'They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the fox had to jump high in the air.': 3, 'As he jumped, he opened his mouth to catch the grapes, but he missed.': 4, 'The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fox decided it was time to give up and go home.': 5, 'While he walked away, he muttered, "I\' m sure the grapes were sour anyway." ': 6}
```

Creating TF-IDF model

```

# Let's now create a tf-idf (Term frequency Inverse Document Frequency) model
tf_idf_vectorizer = TfidfVectorizer(min_df=2, max_features=None,
strip_accents='unicode',

analyzer='word',

token_pattern=r'\w{1,}',

ngram_range=(1, 3),

use_idf=1,smooth_idf=1,

sublinear_tf=1, stop_words

= 'english')

# Passing our sentences treating each as one document to TF-IDF vectorizer
tf_idf_vectorizer.fit(doc_sentences)

# Passing our sentences treating each as one document to TF-IDF vectorizer
tf_idf_vectorizer.fit(doc_sentences)

TfidfVectorizer(min_df=2, ngram_range=(1, 3), smooth_idf=1,

stop_words='english', strip_accents='unicode', sublinear_tf=1,

token_pattern=r'\w{1,}', use_idf=1)

# Transforming our sentences to TF-IDF vectors
sentence_vectors = tf_idf_vectorizer.transform(doc_sentences)

Performing sentence scoring

# Getting sentence scores for each sentences sentence_scores =

np.array(sentence_vectors.sum(axis=1)).ravel()

# Sanity checkup
print(len(doc_sentences) == len(sentence_scores))

True

# Getting top-n sentences N = 3 top_n_sentences = [doc_sentences[ind] for ind in

np.argsort(sentence_scores, axis=0)[-1][:N]]

Performing final summarization

```

```
# Let's now do the sentence ordering using our prebaked sentence_organizer
# Let's map the scored sentences with their indexes mapped_top_n_sentences =
[(sentence,sentence_organizer[sentence]) for sentence in top_n_sentences]
print("Our top_n_sentence with their index: \n") for element in
mapped_top_n_sentences:
    print(element)
```

```
# Ordering our top-n sentences in their original ordering mapped_top_n_sentences
= sorted(mapped_top_n_sentences, key = lambda x: x[1])
ordered_scored_sentences = [element[0] for element in mapped_top_n_sentences]
```

```
# Our final summary summary = ""
"".join(ordered_scored_sentences) Our
top_n_sentence with their index:
```

('He searched high and low, but couldn't find something that he could eat.Finally, as his stomach rumbled, he stumbled upon a farmer's wall.', 1)

('The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fox decided it was time to give up and go home.', 5)

('They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the fox had to jump high in the air.', 3)

Result / Summary

```
print("Summary: \n", summary) Summary:
```

He searched high and low, but couldn't find something that he could eat.Finally, as his stomach rumbled, he stumbled upon a farmer's wall. They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the fox had to jump high in the air. The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fox decided it was time to give up and go home.

4.3 SCREENSHOTS

Topic: Building a Text Summarizer

Importing required libraries

```
In [1]: from sklearn.feature_extraction.text import TfidfVectorizer
        from nltk.tokenize import word_tokenize, sent_tokenize
        from spacy.lang.en import English
        import numpy as np
```

Load spacy model for sentence tokenization

```
In [2]: nlp = English()
        nlp.add_pipe('sentencizer')
        #nlp.add_pipe(nlp.create_pipe('sentencizer'))
```

```
Out[2]: <spacy.pipeline.sentencizer.Sentencizer at 0x1c5be8cec40>
```

```
In [3]: text_corpus = """
        One day, a fox became very hungry as he went to search for some food. He searched high and low, but couldn't find something that he could eat. Finally, as his stomach rumbled, he stumbled upon a farmer's wall. At the top of the wall, he saw the biggest, juiciest grapes he'd ever seen. They had a rich, purple color, telling the fox they were ready to be eaten. To reach the grapes, the fox had to jump high in the air. As he jumped, he opened his mouth to catch the grapes, but he missed. The fox tried again but missed yet again. He tried a few more times but kept failing. Finally, the fox decided it was time to give up and go home. While he walked away, he muttered, "I'm sure the grapes were sour anyway."
        """
```

Create spacy document for further sentence level tokenization

```
In [4]: doc = nlp(text_corpus.replace("\n", ""))
        doc_sentences = sent_tokenize(text_corpus)
        #print(doc_sentences)
```

```
In [5]: def summarizer(text, tokenizer, max_sent_in_summary=3):
        # Create spacy document for further sentence level tokenization
        doc = nlp(text_corpus.replace("\n", ""))
        sentences = [sent.string.strip() for sent in doc.sents]
```

```
In [6]: print("Sentence are: \n", doc_sentences)
```

Sentence are:

```
['\nOne day, a fox became very hungry as he went to search for some food.', 'He searched high and low, but couldn't find something that he could eat. Finally, as his stomach rumbled, he stumbled upon a farmer's wall.', 'At the top of the wall, he saw the biggest, juiciest grapes he'd ever seen. They had a rich, purple color, telling the fox they were ready to be eaten. To reach the grapes, the fox had to jump high in the air.', 'As he jumped, he opened his mouth to catch the grapes, but he missed.', 'The fox tried again but missed yet again. He tried a few more times but kept failing. Finally, the fox decided it was time to give up and go home.', 'While he walked away, he muttered, "I'm sure the grapes were sour anyway."']
```

Creating sentence organizer

```
In [7]: # Let's create an organizer which will store the sentence ordering to later reorganize the
        # scored sentences in their correct order
        sentence_organizer = {k:v for v,k in enumerate(doc_sentences)}
```

```
In [6]: print("Sentence are: \n", doc_sentences)
```

Sentence are:

```
['\nOne day, a fox became very hungry as he went to search for some food.', 'He searched high and low, but couldn't find something that he could eat. Finally, as his stomach rumbled, he stumbled upon a farmer's wall.', 'At the top of the wall, he saw the biggest, juiciest grapes he'd ever seen. They had a rich, purple color, telling the fox they were ready to be eaten. To reach the grapes, the fox had to jump high in the air.', 'As he jumped, he opened his mouth to catch the grapes, but he missed.', 'The fox tried again but missed yet again. He tried a few more times but kept failing. Finally, the fox decided it was time to give up and go home.', 'While he walked away, he muttered, "I'm sure the grapes were sour anyway."']
```

Creating sentence organizer

Creating sentence organizer

```
In [7]: # Let's create an organizer which will store the sentence ordering to later reorganize the
# scored sentences in their correct order
sentence_organizer = {k:v for v,k in enumerate(doc_sentences)}
```

Peeking into our sentence organizer

```
In [8]: print("Our sentence organizer: \n", sentence_organizer)
```

```
Our sentence organizer:
{'\nOne day, a fox became very hungry as he went to search for some food.': 0, 'He searched high and
low, but couldn't find something that he could eat.Finally, as his stomach rumbled, he stumbled upon
a farmer's wall.': 1, 'At the top of the wall, he saw the biggest, juiciest grapes he'd ever seen.':
2, 'They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, t
he fox had to jump high in the air.': 3, 'As he jumped, he opened his mouth to catch the grapes, but
he missed.': 4, 'The fox tried again but missed yet again.He tried a few more times but kept failing.
Finally, the fox decided it was time to give up and go home.': 5, 'While he walked away, he muttered,
"I'm sure the grapes were sour anyway."'': 6}
```

Creating TF-IDF model

```
In [9]: # Let's now create a tf-idf (Term frequency Inverse Document Frequency) model
tf_idf_vectorizer = TfidfVectorizer(min_df=2, max_features=None,
                                   strip_accents='unicode',
                                   analyzer='word',
                                   token_pattern=r'\w{1,}',
                                   ngram_range=(1, 3),
                                   use_idf=1,smooth_idf=1,
                                   sublinear_tf=1,
                                   stop_words = 'english')
```

```
In [10]: # Passing our sentences treating each as one document to TF-IDF vectorizer
tf_idf_vectorizer.fit(doc_sentences)
```

```
Out[10]: TfidfVectorizer(min_df=2, ngram_range=(1, 3), smooth_idf=1,
                        stop_words='english', strip_accents='unicode', sublinear_tf=1,
                        token_pattern='\\w{1,}', use_idf=1)
```

```
In [11]: # Transforming our sentences to TF-IDF vectors
sentence_vectors = tf_idf_vectorizer.transform(doc_sentences)
```

Performing sentence scoring

```
In [12]: # Getting sentence scores for each sentences
sentence_scores = np.array(sentence_vectors.sum(axis=1)).ravel()

# Sanity checkup
print(len(doc_sentences) == len(sentence_scores))
```

True

```
In [13]: # Getting top-n sentences
N = 3
top_n_sentences = [doc_sentences[ind] for ind in np.argsort(sentence_scores, axis=0)[::-1][:N]]
```

Performing final summarization

```
In [14]: # Let's now do the sentence ordering using our prebaked sentence_organizer
# Let's map the scored sentences with their indexes
mapped_top_n_sentences = [(sentence, sentence_organizer[sentence]) for sentence in top_n_sentences]
print("Our top_n_sentence with their index: \n")
for element in mapped_top_n_sentences:
    print(element)

# Ordering our top-n sentences in their original ordering
mapped_top_n_sentences = sorted(mapped_top_n_sentences, key = lambda x: x[1])
ordered_scored_sentences = [element[0] for element in mapped_top_n_sentences]

# Our final summary
summary = " ".join(ordered_scored_sentences)
```

Our top_n_sentence with their index:

```
('He searched high and low, but couldn't find something that he could eat.Finally, as his stomach rum
bled, he stumbled upon a farmer's wall.', 1)
('The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fo
x decided it was time to give up and go home.', 5)
('They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the
fox had to jump high in the air.', 3)
```

Result / Summary

```
In [15]: print("Summary: \n", summary)
```

Summary:

He searched high and low, but couldn't find something that he could eat.Finally, as his stomach rumb led, he stumbled upon a farmer's wall. They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the fox had to jump high in the air. The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fox decided it was time to give up and go home.

Creating a function template using above steps:

```
In [16]: def summarizer(text, tokenizer, max_sent_in_summary=3):
# Create spacy document for further sentence level tokenization
doc = nlp(text.corpus.replace("\n", ""))
sentences = [sent.string.strip() for sent in doc.sents]
# Let's create an organizer which will store the sentence ordering to later reorganize the
# scored sentences in their correct order
sentence_organizer = {k:v for v,k in enumerate(sentences)}
# Let's now create a tf-idf (Term frequency Inverse Document Frequency) model
tf_idf_vectorizer = TfidfVectorizer(min_df=2, max_features=None,
token_pattern=r'\w{1,}',
ngram_range=(1, 3),
use_idf=1,smooth_idf=1,
sublinear_tf=1,
stop_words = 'english')

# Passing our sentences treating each as one document to TF-IDF vectorizer
tf_idf_vectorizer.fit(sentences)
# Transforming our sentences to TF-IDF vectors
sentence_vectors = tf_idf_vectorizer.transform(sentences)
# Getting sentence scores for each sentences
sentence_scores = np.array(sentence_vectors.sum(axis=1)).ravel()
# Getting top-n sentences
N = max_sent_in_summary
top_n_sentences = [sentences[ind] for ind in np.argsort(sentence_scores, axis=0)[::-1][:N]]
# Let's now do the sentence ordering using our prebaked sentence_organizer
# Let's map the scored sentences with their indexes
mapped_top_n_sentences = [(sentence, sentence_organizer[sentence]) for sentence in top_n_sentences]
# Ordering our top-n sentences in their original ordering
mapped_top_n_sentences = sorted(mapped_top_n_sentences, key = lambda x: x[1])
ordered_scored_sentences = [element[0] for element in mapped_top_n_sentences]
# Our final summary
summary = " ".join(ordered_scored_sentences)
return summary
```

```
In [17]: print("Summarizer Result: \n", summary)
```

Summarizer Result:

He searched high and low, but couldn't find something that he could eat.Finally, as his stomach rumb led, he stumbled upon a farmer's wall. They had a rich, purple color, telling the fox they were ready to be eaten.To reach the grapes, the fox had to jump high in the air. The fox tried again but missed yet again.He tried a few more times but kept failing.Finally, the fox decided it was time to give up and go home.

CHAPTER 5

SUMMARY AND CONCLUSIONS

Text summarization is an interesting research topic among the NLP community that helps produce concise information. The idea of this paper is to present the latest research and progress made in this field with the SLR method. With the SLR method, it is proven that it can provide a more structured, broad, and diverse review, ranging from trends/topics, datasets, preprocessing, features, approach techniques, problems, methods to evaluation that are available as a guide for future work, the relationship between trends/topics, problems and challenges in each topic, technique, and method used is summarized into one to make it easier to explore and re-analyze.

The important thing that is considered interesting from the review that has been done is the results of the analysis which states that extractive summaries are relatively easier than abstractive summaries which are very complex, extractive summaries are still the topic of current favorite trends. This is because there are still many things that are a challenge for researchers to do. It also can be seen that the most important features to produce a good summary are keywords, frequency, similarity, sentence position, sentence length, and semantics. The machine learning approach is a favorite technique because of automatic machine learning performance and learning to enhance the experience without being explicitly programmed. Even though machine learning is a favorite, machine learning is not the only best approach. An approach that is easily combined with other approaches is statistics. Statistics can be combined with machine learning, or statistics with fuzzy based. And the problem most often solved with a statistical approach is combined with other approaches such as determining frequency, determining keywords, and similarity. In the topic of text summarization research, future work that can be done includes i) solving feature problems, namely determining the most appropriate features to use in summarizing by the dataset by selecting features, discovering new features, optimizing commonly used features, feature engineering, using features for semantics.

linguistic features, finding features to produce coherence sentences, and add grammatical features. ii) Preprocessing by the problem dataset using appropriate stemming, besides, to stop word removal and tokenizing, POS Tagging is also needed, namely to categorize word classes, such as nouns, verbs, adjectives, etc. iii) For extractive summaries, collaborating statistical techniques, fuzzy-based techniques, and machine learning are very challenging to try. Or develop MDS (Patel et al., 2019) by combining the Fuzzy Logic and TF-IDF methods with a machine learning approach to produce a better summary. iv) For abstractive summarizing, we can develop the latest method, namely NATSUM in other cases or improve the performance of NATSUM by increasing coherence. v) Use rarely used datasets such

as legal documents, summarizing tourist attractions, and summarizing documents for assessment. Or you can use favorite datasets such as DUC 2002.

The extractive text summarization has many tasks involved in generating the précis. The author proposed a statistical approach for generating an extractive précis of a sports articles that uses sentence ranking based on the selected features for the sentences. The most critical part is identifying the important sentences without loss of meaning from the given document. Using the proposed method on an average, 73% precision, 76% recall, and 74% f-measure is obtained which when compared with a manual generated summary and an online summarizer tool.

In nearly future, the proposed model can also be developed to a neural network model that takes the values of different features as input for generating an accurate précis by considering the contextual meaning. To increase the accuracy of the précis, various features can also be added.

REFERENCES

- [1] U. Hahn, I. Mani, "of Automatic Researchers are investigating summarization tools and methods that", IEEE Computer 33. 11, pp. 29-36, November 2000.
- [2] E. Lloret, M. Palomar, "Text summarization in progress: a literature review" in Springer, Springer, pp. 1-41, 2012
- [3] K. Spärck Jones, "Automatic summarizing: The state of the art", Information Processing & Management, vol. 43, pp. 1449-1481, nov 2007
- [4] Khan, N. Salim, "A review on abstractive summarization methods", Journal of Theoretical and Applied Information Technology, vol. 59, no. 1, pp. 64- 72, 2014
- [5] E. Baralis, L. Cagliero, A. Fiori, P. Garza, "MWI-Sum: A Multilingual Summarizer Based on Frequent Weighted Itemsets", ACM Transactions on Information Systems, vol. 34, no. 1, pp. 1-35, 2015
- [6] M. Kamal, "Text Summarization in Bioinformatics", 15th International Conference on Computer and Information Technology, pp. 592-597, 2012
- [7] Jo, Duke Taeho. (2017). K nearest neighbor for text summarization using feature similarity. 1-5. 10.1109/ICCCCEE.2017.7866705.
- [8] N. S. Shirwandkar and S. Kulkarni, "Extractive Text Summarization Using Deep Learning," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-5. doi: 10.1109/ICCUBEA.2018.8697465
- [9] P. Krishnaveni and S. R. Balasundaram, "Automatic text summarization by local scoring and ranking for improving coherence," 2017 International Conference on Computing Methodologies and Communication (ICCMC), Erode, 2017, pp. 59-64. doi: 10.1109/ICCMC.2017.8282539
- [10] Valverde Tohalino, Jorge & Amancio, Diego. (2017). Extractive Multidocument Summarization Using Multilayer Networks. Physica A: Statistical Mechanics and its Applications. 503. 10.1016/j.physa.2018.03.013.