TEXT SUMMARIZATION USING NATURAL LANGUAGE PROCESSING

A Report submitted in partial fulfilment of the requirement for the award of degree of

Bachelor of Technology

In

Electronics and Communication Engineering

Under the Supervision of

Dr. Sudesh Pahal

Submitted By:

Ayan Ansar (00815002819)

Sitanshu Deb (01215002819)

Srajit Rastogi (04815002819)

MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY

C4, JANAKPURI, NEW DELHI – 110058

(GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DWARKA, NEW DELHI)

June 2023

DECLARATION

We, student of B.Tech (Electronics & Communication Engineering) hereby declare that the project work done on "Text Summarization Using NLP" submitted to Maharaja Surajmal Institute of Technology, Janakpuri Delhi in partial fulfilment of the requirement for the award of degree of Bachelor of Technology comprises of our original work and has not been submitted anywhere else for any other degree to the best of our knowledge.

*Ayan Ansar (*00815002819*)*

*Sitanshu Deb (*01215002819*)*

Srajit Rastogi (04815002819)

CERTIFICATE

This is to certify that the project work done on "Text Summarization Using NLP" submitted to Maharaja Surajmal Institute of Technology, Janakpuri Delhi by "Ayan Ansar, Sitanshu Deb and Srajit Rastogi" in partial fulfillment of the requirement for the award of degree of Bachelor of Technology, is a bonafide work carried out by them under my supervision and guidance. This project work comprises of original work and has not been submitted anywhere else for any other degree to the best of my knowledge.

Signature of Supervisor
(Dr. Sudesh Pahal)

Signature of HOD
(Prof. Archana Balyan HOD, ECE)

ACKNOWLEDGEMENT

We would like to express our sincere appreciation to the individuals and organizations who have contributed to the successful completion of this group project report.

First and foremost, we extend our deepest gratitude to Prof. Archana Balyan, HOD, ECE, for their invaluable guidance, expertise, and continuous support throughout this project. Their insightful feedback and unwavering commitment to our development have been crucial in shaping this report.

We are immensely grateful Dr. Sudesh Pahal, ECE, for providing the necessary resources, facilities, and research opportunities that have facilitated the execution of this project. Their commitment to academic excellence and creating an environment conducive to learning and innovation has been pivotal in our growth as engineers.

We would like to acknowledge and thank our fellow group members for their dedication, hard work, and collaboration. Each member's unique skills, perspectives, and unwavering commitment to excellence have significantly contributed to the overall success of this project.

Our deepest appreciation goes to our families and friends for their unwavering support, understanding, and encouragement throughout this project. Their belief in our abilities and constant motivation have been instrumental in our progress.

Furthermore, we would like to acknowledge the authors, researchers, and organizations whose scholarly works, publications, and online resources have served as valuable references for this report. Their contributions to the field of engineering have provided a solid foundation for our research.

Finally, we would like to acknowledge anyone else who has directly or indirectly contributed to this project but may not have been mentioned individually. Your support, feedback, and contributions have been deeply appreciated.

Ayan Ansar (00815002819)

Sitanshu Deb (01215002819)

Srajit Rastogi (04815002819)

CONTENTS

ABSTRACT

Text summarization plays a crucial role in effectively extracting important information from large volumes of text. With the exponential growth of digital content, there is an increasing need for automated methods to summarize text and provide concise representations of the main ideas. In this group project report, we explore the application of Natural Language Processing (NLP) techniques for text summarization.

Our project aims to develop a robust and accurate text summarization system that leverages the power of NLP. We begin by gathering a diverse dataset of textual documents from various domains, including news articles, research papers, and online blogs. We preprocess the data by cleaning, tokenizing, and removing noise to ensure high-quality input for the summarization model.

Next, we implement and experiment with different state-of-the-art NLP techniques for text summarization, including extractive and abstractive methods. Extractive approaches involve selecting and rearranging the most important sentences or phrases from the source text, while abstractive methods generate new sentences that capture the essence of the original text. We evaluate the performance of these techniques using metrics such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores and human evaluation.

To enhance the quality and coherence of the generated summaries, we incorporate advanced NLP components such as named entity recognition, part-of-speech tagging, and syntactic parsing. These components help in identifying key entities, determining sentence importance, and maintaining grammatical correctness in the summarized text.

Finally, we discuss the limitations and challenges faced during the project, including issues related to dataset biases, handling long and complex documents, and generating concise yet informative summaries. We also provide insights into potential future directions for improving text summarization using NLP, such as incorporating multi-modal inputs and exploring reinforcement learning techniques.

Overall, our group project showcases the power and versatility of NLP in the field of text summarization. By leveraging advanced techniques and pretraining models, we demonstrate the effectiveness of automated approaches in condensing large volumes of text into concise and meaningful summaries, thereby facilitating efficient information retrieval and consumption in various domains.

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	FULL FORM OF ABBREVIATION
ANN	Artificial Neural Network
NLP	Natural Language Processing
BERT	Bidirectional Encoder Representations from Transformers

CHAPTER 1 –INTRODUCTION

1.1 Introduction

Introducing our revolutionary text summarizer—a powerful tool designed to simplify the way we consume information in our fast-paced world. As we navigate through a vast sea of textual data, be it news articles, research papers, or lengthy documents, extracting the most pertinent details efficiently becomes an essential skill. Our text summarizer emerges as the ultimate solution, enabling users to distill lengthy texts into concise summaries, saving time and enhancing productivity.

In today's information-driven society, the ability to quickly grasp the core essence of a text is invaluable. Whether you're a student seeking to comprehend complex subject matter, a professional staying updated on industry trends, or simply an individual keeping up with current events, our text summarizer streamlines the process by extracting key ideas, crucial facts, and essential arguments from any given text.

Gone are the days of skimming through pages upon pages of information, desperately searching for relevant content. Our text summarizer leverages cutting-edge natural language processing algorithms to analyze and understand the context, identifying the most salient points within a document. It then generates concise summaries that encapsulate the main concepts, allowing users to digest information quickly and effortlessly.

Imagine preparing for an important presentation—instead of spending hours meticulously reading through a multitude of research papers and articles, our text summarizer condenses the essential content into a comprehensive overview. With the most critical points at your fingertips, you can focus on crafting a compelling narrative, confident in your understanding of the subject matter.

Moreover, our text summarizer finds its utility in the realm of news consumption. In an era where news stories break and evolve rapidly, staying informed can be overwhelming. By employing our tool, you can effortlessly stay up-to-date with the latest events. Our summarizer sifts through extensive news articles, extracts the core details, and presents them concisely, allowing you to grasp the crucial information in a fraction of the time.

Our text summarizer extends beyond personal use; it serves as an invaluable asset in various professional domains. From research and development to legal and financial sectors, the ability to analyze and synthesize large volumes of text swiftly is a game-changer. Our tool empowers professionals to navigate complex information landscapes effectively, making informed decisions and driving progress with confidence.

In conclusion, our text summarizer revolutionizes the way we approach textual information, providing a swift and efficient solution to the challenges of the modern world. By condensing lengthy texts into concise summaries, it saves time, enhances comprehension, and boosts productivity. From students and professionals to avid readers and news enthusiasts, our text summarizer is the ultimate companion in the quest for

1.2 Methodology

The methodology for a text summarizer involves several key steps to analyze and extract essential information from a given text. Here's a high-level overview of the methodology along with a flow diagram illustrating the process:

Text Preprocessing: The first step involves preprocessing the input text to ensure it is in a suitable format for analysis. This may include removing unnecessary characters, converting the text to lowercase, and handling special cases like abbreviations and acronyms.

Sentence Tokenization: The text is then segmented into individual sentences to facilitate further analysis at the sentence level. This step helps in breaking down the text into smaller, more manageable units.

Text Analysis: In this step, the text is analyzed using natural language processing (NLP) techniques. This may include tasks such as part-of-speech tagging, named entity recognition, syntactic parsing, and sentiment analysis. These analyses provide valuable insights into the structure, entities, and sentiment expressed in the text.

Sentence Scoring: Each sentence in the text is assigned a score based on its importance or relevance to the overall meaning of the document. Various approaches can be employed to calculate these scores, such as term frequency-inverse document frequency (TF-IDF), graph-based algorithms like PageRank, or deep learning models like BERT (Bidirectional Encoder Representations from Transformers).

Sentence Selection: The sentences with the highest scores are selected for inclusion in the summary. The number of selected sentences can be predefined or dynamically determined based on the desired length or compression ratio.

Summary Generation: Finally, the selected sentences are concatenated to form the summary of the original text. Additional post-processing steps, such as ensuring coherence and readability, can also be applied to refine the summary further.

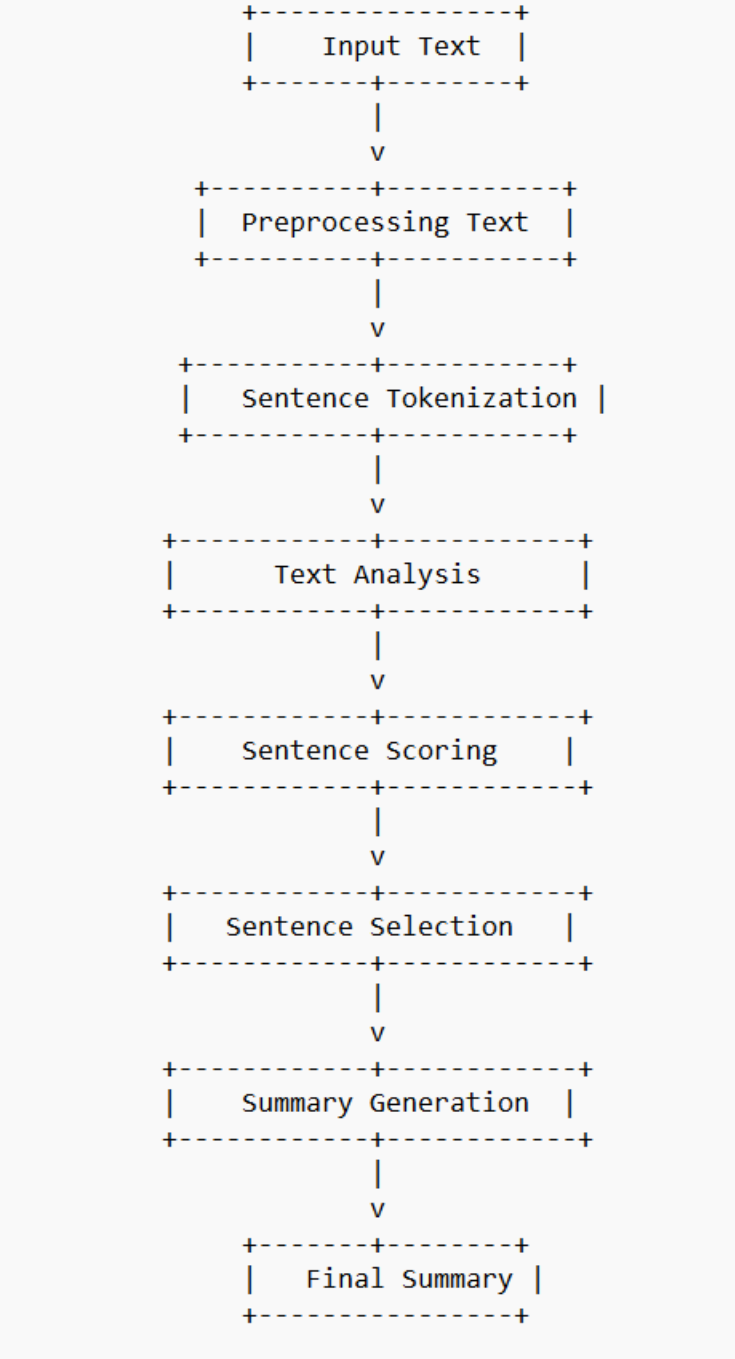

Figure 1: Flowchart for text summarizer

1.3 Software Used

1.3.1 Python

Python is a versatile and widely used programming language that has gained immense popularity among developers and enthusiasts alike. Known for its simplicity and readability, Python offers an elegant and concise syntax that makes it easy to write and understand code. It is a high-level language that prioritizes code readability, emphasizing the use of whitespace indentation to define code blocks. Python's extensive standard library and vast ecosystem of third-party packages make it suitable for a wide range of applications, from web development and data analysis to artificial intelligence and scientific computing. With its rich set of tools and libraries, Python empowers developers to quickly prototype ideas, build robust applications, and solve complex problems efficiently. Its versatility, ease of use, and thriving community have solidified Python's position as a go-to language for both beginners and seasoned professionals in the world of programming.

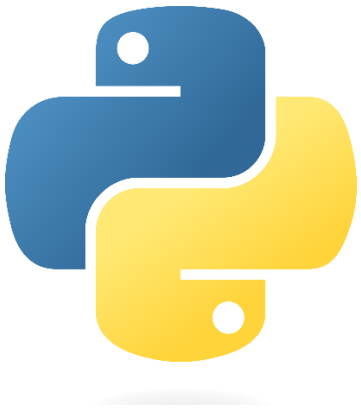

Figure 2: Python logo

1.3.2 TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. TensorFlow is a rich system for managing all aspects of a machine learning system; however, this class focuses on using a particular TensorFlow API to develop and train machine learning models.

Figure 3: TensorFlow logo

1.3.3 Streamlit

Streamlit is an open-source Python library that enables developers to create interactive web applications and data dashboards with ease. It simplifies the process of building and deploying data-driven applications by providing a straightforward and intuitive interface. With Streamlit, developers can transform their Python scripts into interactive web apps using a single codebase.

Streamlit focuses on simplicity and allows developers to quickly prototype and share their ideas. It provides a range of built-in components and widgets that enable the creation of interactive elements such as sliders, buttons, checkboxes, and plots. Developers can use these components to easily add interactivity and dynamic behavior to their applications.

Figure 4: Streamlit logo

1.3.4 Neural Network

A neural network, also known as an artificial neural network (ANN) or simply a "neural net," is a computational model inspired by the structure and functioning of the biological brain. It is a powerful machine learning technique that can learn from data, identify patterns, and make predictions or decisions.

Neural networks consist of interconnected nodes, called artificial neurons or "neurons," organized in layers. These layers typically include an input layer, one or more hidden layers, and an output layer. The connections between neurons, represented by weighted edges, allow information to flow through the network.

Each neuron receives inputs, performs a computation, and produces an output. The inputs are multiplied by corresponding weights, summed together, and then passed through an activation function that introduces non-linearity. The output of each neuron becomes the input for the next layer, and this process continues until the final output is produced.

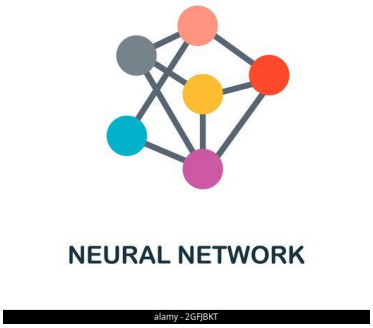

Figure 5: Neural Network logo

1.3.5 Long Short-Term Memory

LSTM stands for Long Short-Term Memory, and it is a type of recurrent neural network (RNN) architecture that is widely used in machine learning and natural language processing tasks. LSTMs are specifically designed to overcome the limitations of traditional RNNs in capturing and retaining long-term dependencies in sequential data.

In many sequential data tasks, such as language modeling, speech recognition, and sentiment analysis, understanding the context and preserving information from distant past inputs becomes crucial. Traditional RNNs suffer from the "vanishing gradient" problem, which hampers their ability to capture long-term dependencies. LSTMs were developed to address this issue.

LSTMs incorporate a more complex memory cell structure compared to traditional RNNs. The key component of an LSTM cell is the memory cell, which allows the network to selectively remember or forget information over time. It consists of three main components:

Forget Gate: This gate determines which information to discard from the memory cell. It takes inputs from the previous hidden state and the current input and produces a forget gate activation value between 0 and 1 for each element in the memory cell.

Input Gate: The input gate decides which new information needs to be stored in the memory cell. It consists of two sub-components: a sigmoid activation layer that determines the relevance of new information and a tanh activation layer that creates a vector of candidate values to be added to the memory cell.

Output Gate: The output gate determines what information to output from the current hidden state. It takes inputs from the previous hidden state and the current input and combines them to produce an output value between -1 and 1.

LSTMs effectively handle the vanishing gradient problem by allowing information to flow unchanged through the memory cell using a mechanism called the "constant error carousel." This enables the network to retain information over long sequences and makes them well-suited for tasks involving long-term dependencies.

LSTM networks have been successfully applied to various tasks, including machine

translation, speech recognition, text generation, sentiment analysis, and more. They have proven to be effective in capturing complex patterns and dependencies in sequential data and have become a fundamental component in many state-of-the-art models within the field of deep learning.

1.3.6 Seq2Seq Model

A Seq2Seq model, short for Sequence-to-Sequence model, is a type of neural network architecture used for tasks involving sequential data, such as machine translation, text summarization, chatbot development, and speech recognition. It consists of two main components: an encoder and a decoder.

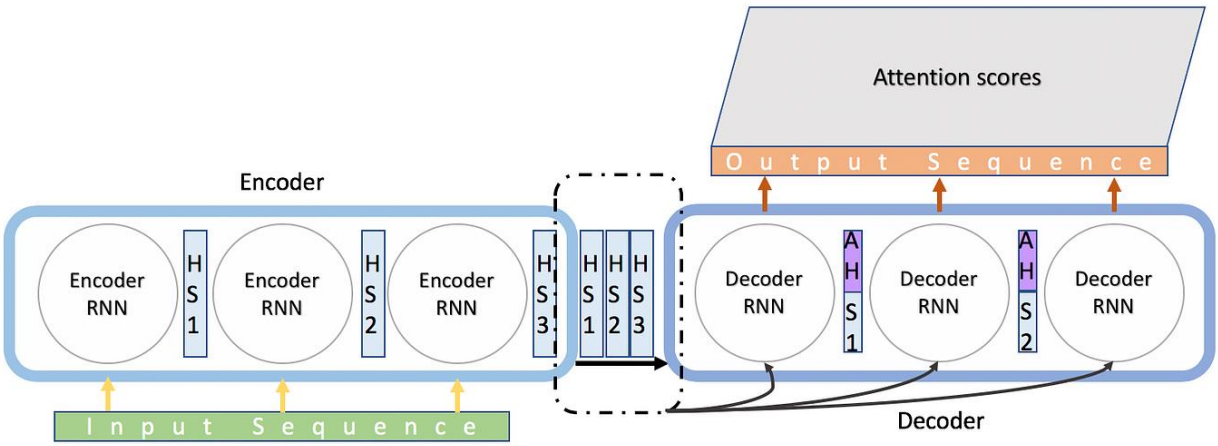

Figure 6: Seq2Seq Mode

CHAPTER 2 – Working of Seq2Seq model

2.1 Working of seq2seq model

A seq2seq (sequence-to-sequence) model for text summarization is a deep learning architecture designed to generate concise summaries from longer pieces of text. It consists of two main components: an encoder and a decoder. The purpose of the encoder is to process the input text and convert it into a fixed-length vector representation, capturing the important information. The decoder then takes this representation and generates a summary based on it.

At the heart of the seq2seq model is a recurrent neural network (RNN), typically a type of RNN called long short-term memory (LSTM). The input text is tokenized and fed into the encoder LSTM, one token at a time. The LSTM updates its internal state at each time step, gradually building a contextual understanding of the input. The final hidden state of the encoder LSTM serves as the condensed representation of the input text.

Once the encoder has processed the entire input, the decoder LSTM takes over. It starts with an initial hidden state, which is set to the final hidden state of the encoder LSTM. The decoder generates the summary by predicting one token at a time, conditioned on both its previous predictions and the encoded representation of the input text. At each time step, the decoder LSTM updates its hidden state based on the previous hidden state, the previously generated token, and the encoded representation. It then predicts the next token using a softmax layer, which outputs a probability distribution over the vocabulary.

During training, the seq2seq model is optimized to minimize the discrepancy between the predicted summary and the target summary. This is typically done using a loss function such as cross-entropy loss, which measures the dissimilarity between the predicted and target probability distributions over the vocabulary.

To generate summaries during inference, a technique called beam search is often employed. Beam

search maintains a set of the most likely summary candidates at each time step, expanding the search space as new tokens are generated. This helps to explore multiple potential summary paths and produce more diverse and coherent summaries.

Overall, the seq2seq model for text summarization leverages the power of recurrent neural networks to capture the contextual information in the input text and generate concise summaries based on that information. It has been widely used in various natural language processing tasks and has shown promising results in automatic text summarization.

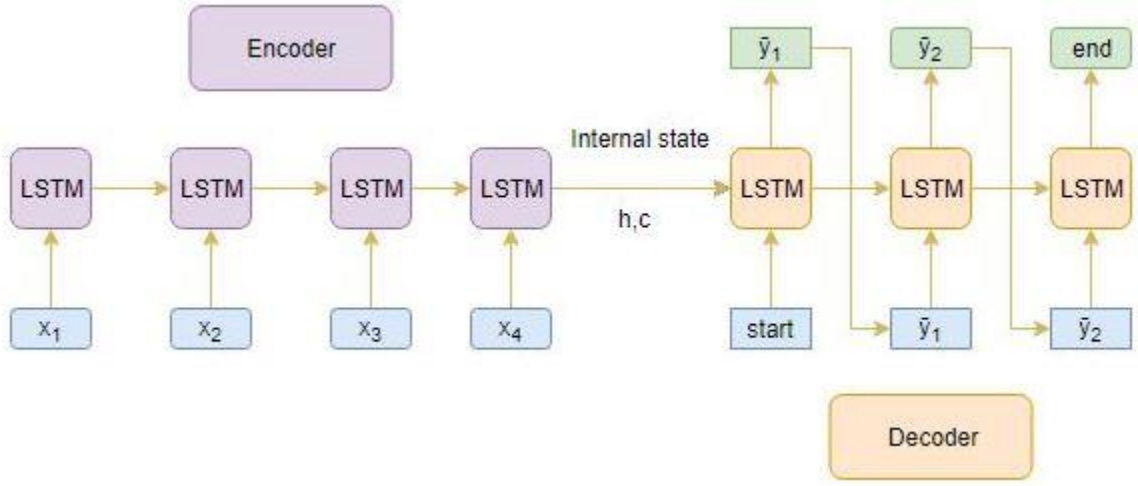

Fig 7: seq2seq Model indicating encoder and decoder

2.2 How does the inference process work in seq2seq Model?

Here are the steps to decode the test sequence:

- Encode the entire input sequence and initialize the decoder with internal states of the encoder
- Pass <**start**> token as an input to the decoder
- Run the decoder for one timestep with the internal states
- The output will be the probability for the next word. The word with the maximum probability will be selected

- Pass the sampled word as an input to the decoder in the next timestep and update the internal states with the current time step
- Repeat steps 3 – 5 until we generate <**end**> token or hit the maximum length of the target sequence

Let's take an example where the test sequence is given by $[x_1, x_2, x_3, x_4]$. How will the inference process work for this test sequence? I want you to think about it before you look at my thoughts below.

Encode the test sequence into internal state vectors

Observe how the decoder predicts the target sequence at each timestep:

Timestep: t=1

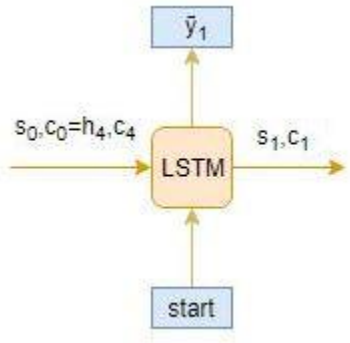

Figure 8: Decoder at t=1

Timestep: t=2

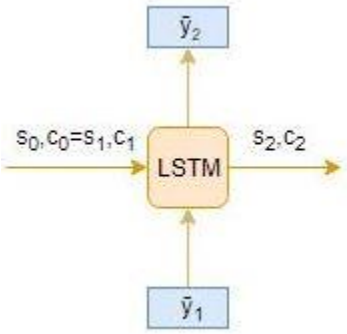

Figure 9: Decoder at t=2

And, Timestep: t=3

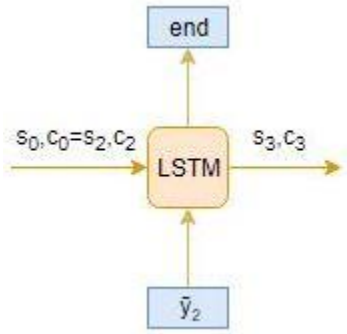

Figure 10: Decoder at t=3

2.3 Limitations of the Encoder – Decoder Architecture

As useful as this encoder-decoder architecture is, there are certain limitations that come with it. The encoder converts the entire input sequence into a fixed length vector and then the decoder predicts the output sequence. **This works only for short sequences** since the decoder is looking at the entire input sequence for the prediction

Here comes the problem with long sequences. **It is difficult for the encoder to memorize long sequences into a fixed length vector**

"A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences. The performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases."
-Neural Machine Translation by Jointly Learning to Align and Translate

So how do we overcome this problem of long sequences? This is where the concept of **attention**

mechanism comes into the picture. It aims to predict a word by looking at a few specific parts of the sequence only, rather than the entire sequence.

2.4 The Intuition behind the Attention Mechanism

How much attention do we need to pay to every word in the input sequence for generating a word at timestep **t**? That's the key intuition behind this attention mechanism concept.

Let's consider a simple example to understand how Attention Mechanism works:

Source sequence: "Which sport do you like the most?

Target sequence: "I love cricket"

The first word **'I'** in the target sequence is connected to the fourth word **'you'** in the source sequence, right? Similarly, the second-word **'love'** in the target sequence is associated with the fifth word **'like'** in the source sequence.

So, instead of looking at all the words in the source sequence, we can increase the importance of specific parts of the source sequence that result in the target sequence. This is the basic idea behind the attention mechanism.

There are 2 different classes of attention mechanism depending on the way the attended context vector is derived:

1. Global Attention
2. Local Attention

Let's briefly touch on these classes.

2.4.1 Global Attention

Here, the attention is placed on all the source positions. In other words, **all the hidden states of the encoder are considered for deriving the attended context vector:**

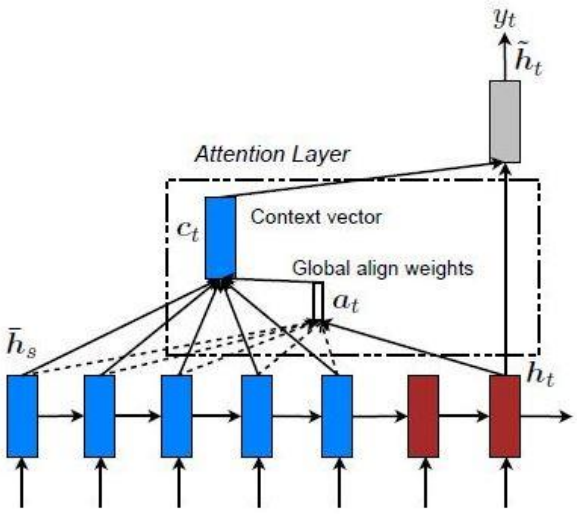

Fig 11: Working of Global Attention Mechanism

2.4.2 Local Attention

Here, the attention is placed on only a few source positions. **Only a few hidden states of the encoder are considered for deriving the attended context vector:**

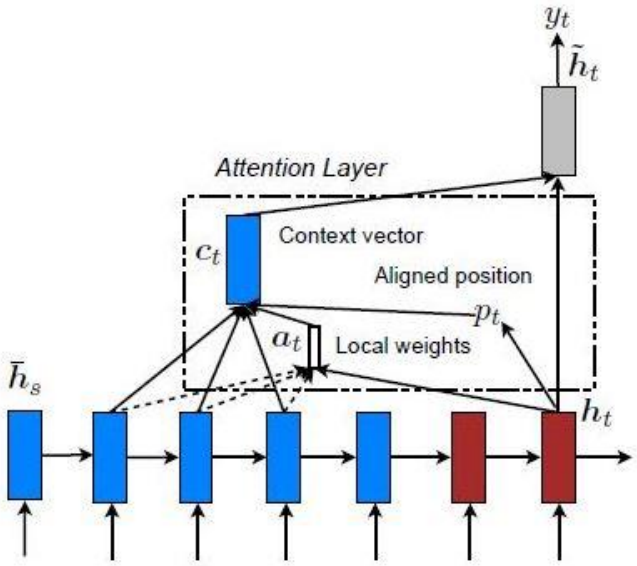

Fig 12: Working of Local Attention Mechanism

Chapter 3 - Problem Statement and Dataset

3.1 Understanding the Problem Statement

Customer reviews can often be long and descriptive. Analyzing these reviews manually, as you can imagine, is really time-consuming. This is where the brilliance of Natural Language Processing can be applied to generate a summary for long reviews.

We will be working on a really cool dataset.

Our objective here is to generate a summary for the Amazon Fine Food reviews using the abstraction-based approach we learned about above.

3.2 Read the dataset

This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all **~500,000** reviews up to October 2012. These reviews include product and user information, ratings, plain text review, and summary. It also includes reviews from all other Amazon categories.

We'll take a sample of **100,000** reviews to reduce the training time of our model. Feel free to use the entire dataset for training your model if your machine has that kind of computational power.

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Fig 13: Example of Dataset

3.3 Preprocess the Data

Preprocessing text data is a crucial step in preparing it for a text summarization model. Several techniques can be applied to clean and transform the raw text into a more suitable format for the model.

The first step is often tokenization, where the text is split into individual words or subword units. Tokenization helps in breaking down the text into meaningful units for further analysis. Additionally, the text is often lowercased to ensure consistency and reduce the vocabulary size.

Next, stop words, which are common words that do not carry significant meaning, are typically removed. These words include articles, prepositions, and conjunctions. Removing stop words can help reduce noise and focus on more informative content.

Another important preprocessing step is handling punctuation and special characters. Depending on the requirements of the summarization task, certain punctuation marks may be preserved, while others may be removed. Special characters such as URLs, email addresses, or numbers can also be replaced or removed, depending on their relevance to the summarization process.

Text data often contains noise in the form of HTML tags, non-ASCII characters, or extra whitespaces. Cleaning these artifacts is essential to ensure the accuracy of the summarization model. Techniques like regular expressions or library-specific methods can be used to remove unwanted patterns and special characters.

In some cases, stemming or lemmatization can be applied to reduce words to their base form. This helps in grouping together variations of the same word and reducing vocabulary size, thereby improving generalization.

Finally, the preprocessed text may undergo additional steps such as removing rare words, handling out-of-vocabulary (OOV) words, or applying word embeddings for representation. These steps depend on the specific requirements and limitations of the text summarization model being used.
By performing these preprocessing steps, the text data is transformed into a more structured and consistent format that can be fed into a text summarization model, facilitating better understanding and extraction of key information from the input text.

3.4 Distribution of length of text and summary

To calculate the distribution of the length of summaries and texts, you would typically iterate through your dataset and record the length of each summary and text. This can be done by tokenizing the text and counting the number of tokens or by measuring the number of characters in the text. By collecting this information, you can analyze the distribution to gain insights into the typical lengths of summaries and texts in your dataset. This analysis can help you determine appropriate model architecture choices, such as setting appropriate sequence lengths or choosing an appropriate beam size during inference. Additionally, understanding the distribution can guide you in preprocessing steps like truncating or padding the input sequences to ensure consistency and optimize model performance.

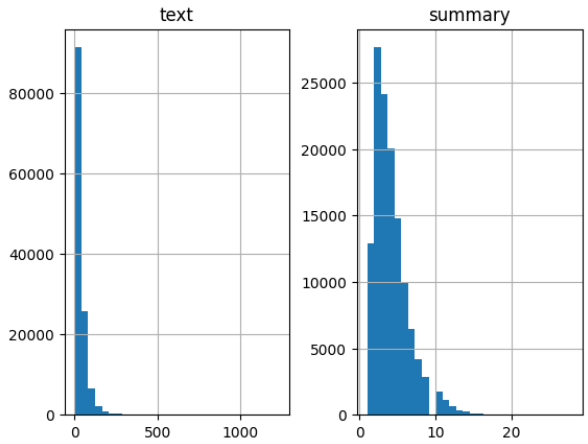

Fig 14: Distribution of length of summaries and text

Chapter 4 – Model Building

4.1 Preparing Tokenizer

Preparing a tokenizer is a fundamental step in text preprocessing for various natural language processing tasks, including text summarization. Tokenization involves dividing the text into smaller, meaningful units such as words, subwords, or characters, which serve as the basic building blocks for further analysis. The process of preparing a tokenizer typically begins with collecting a representative corpus of text data. This corpus can consist of a large collection of documents, sentences, or paragraphs that cover the domain or language of interest.

Once the corpus is collected, the next step is to train the tokenizer. Training a tokenizer involves analyzing the statistical properties of the text and assigning unique tokens to different units based on their frequency, context, or linguistic rules. There are different strategies for tokenization, such as word-based tokenization, where each word in the text becomes a token, and subword-based tokenization, which splits words into smaller subword units to handle out-of-vocabulary (OOV) words or handle morphologically rich languages.

During training, the tokenizer builds a vocabulary, which is essentially a mapping of tokens to numerical indices. The tokens are typically sorted based on their frequency, with more frequent tokens assigned lower indices. This vocabulary is then used to convert the raw text into a sequence of tokens. The tokenizer can also handle special tokens like start-of-sentence (SOS) and end-of-sentence (EOS) markers, as well as handle unknown words by assigning a special OOV token.

The prepared tokenizer plays a crucial role in subsequent steps of the text summarization pipeline. It enables efficient processing and modeling of the text data. For instance, the tokenized sequences can be fed into a deep learning model, such as a seq2seq model, for training or inference. Additionally, the tokenizer assists in standardizing the input text representation, allowing the model to generalize well across different inputs.

Furthermore, the tokenizer facilitates other preprocessing steps such as padding or truncating

sequences. In cases where the input sequences have variable lengths, padding can be applied to make them of equal length by adding a special padding token. This ensures that all input sequences are of the same length, which is often necessary for batch processing in neural networks. Truncation, on the other hand, involves removing tokens from longer sequences to match a desired maximum length.

4.2 Model Building

In the context of model building for text summarization, there are several terms that we need to familiarize ourselves with. These terms are important for understanding the configuration and behavior of the LSTM (Long Short-Term Memory) layers commonly used in sequence-to-sequence models.

The first term is "Return Sequences = True." In LSTM, the return sequences parameter determines whether the layer should return the hidden state and cell state for every timestep in the input sequence. By setting return sequences to True, the LSTM layer produces output sequences of hidden states and cell states that correspond to each input timestep. This is useful when we want to capture and utilize the information from each step in the input sequence.

The second term is "Return State = True." When return state is set to True, the LSTM layer provides access to the hidden state and cell state of the last timestep only. This means that instead of returning output sequences for all timesteps, the layer only returns the final hidden state and cell state. This can be beneficial when we are interested in obtaining the summarization or representation of the entire input sequence in a condensed form.

The next term is "Initial State." In LSTM, the initial state is used to initialize the internal states (hidden state and cell state) of the LSTM layer for the first timestep of the input sequence. It allows us to set a specific initial state, which can be useful in certain scenarios, such as when we want to control the starting point of the LSTM's memory.

Lastly, we have "Stacked LSTM." Stacked LSTM refers to having multiple layers of LSTM stacked

on top of each other. This configuration allows for a more expressive and deeper representation of the input sequence. Each LSTM layer in the stack receives input from the previous layer, thereby enabling the model to learn hierarchical representations and capture more complex patterns in the data. Stacked LSTM can be advantageous for text summarization tasks as it allows the model to capture both local and global dependencies in the text, potentially leading to improved performance.

By understanding these terms and experimenting with different configurations, such as enabling return sequences, return state, initializing states, and exploring stacked LSTM architectures, we can gain a deeper understanding of the LSTM-based model's behavior and leverage its capabilities to build effective text summarization models.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 30)]              0         []

embedding (Embedding)        (None, 30, 100)           1016200   ['input_1[0][0]']

lstm (LSTM)                  [(None, 30, 300),         481200    ['embedding[0][0]']
                              (None, 300),
                              (None, 300)]

input_2 (InputLayer)         [(None, None)]            0         []

lstm_1 (LSTM)                [(None, 30, 300),         721200    ['lstm[0][0]']
                              (None, 300),
                              (None, 300)]

embedding_1 (Embedding)      (None, None, 100)         251800    ['input_2[0][0]']

lstm_2 (LSTM)                [(None, 30, 300),         721200    ['lstm_1[0][0]']
                              (None, 300),
                              (None, 300)]

lstm_3 (LSTM)                [(None, None, 300),       481200    ['embedding_1[0][0]',
                              (None, 300),                        'lstm_2[0][1]',
                              (None, 300)]                        'lstm_2[0][2]']

attention_layer (AttentionLaye  ((None, None, 300),    180300    ['lstm_2[0][0]',
r)                           (None, None, 30))                   'lstm_3[0][0]']

concat_layer (Concatenate)   (None, None, 600)         0         ['lstm_3[0][0]',
                                                                  'attention_layer[0][0]']

time_distributed (TimeDistribu  (None, None, 2518)     1513318   ['concat_layer[0][0]']
ted)

==================================================================================================
Total params: 5,366,418
Trainable params: 5,366,418
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig 15: Model Summary

Chapter 5 – Training

5.1 Train Test Split

Train-test split is a common practice in machine learning and is crucial for evaluating the performance of a model. In our case, we are using a train-test split to divide our data into two sets: a training set and a test set. The training set is used to train the model, while the test set is used to evaluate its performance on unseen data. The train-test split helps assess how well the model generalizes to new, unseen examples. In this scenario, we are using 10% of our data for training, implying that 90% of the data is reserved for testing. This split ensures that the model is trained on a sufficient amount of data while still leaving a substantial portion for evaluation, enabling us to assess its effectiveness in summarizing text accurately and we train our model for till the validation loss increases on a epoch.

5.2 Optimizer and Metric

When compiling our model, we are using the Adam optimizer and sparse categorical entropy as the evaluation metric. The optimizer is a crucial component that determines how the model learns and updates its parameters during training.

5.2.1 Adam Optimizer :

Adam stands for Adaptive Moment Estimation and is an optimization algorithm that combines the benefits of both AdaGrad and RMSProp optimizers. It adapts the learning rate for each parameter based on its historical gradients, allowing for efficient and adaptive optimization of the model's weights. Adam is widely used due to its effectiveness in handling sparse gradients and accelerating the convergence of the training process.

5.2.2 Metric :

On the other hand, the evaluation metric, sparse categorical entropy, is used to measure the dissimilarity between the predicted and true labels in multi-class classification tasks. Sparse

categorical entropy is suitable when the target labels are integers and not one-hot encoded. It calculates the cross-entropy loss between the predicted probability distribution and the true labels, taking into account the sparsity of the target representation. By using sparse categorical entropy as the evaluation metric, we can effectively assess the performance of our text summarization model in correctly predicting the class labels or summaries, considering the specific nature of our problem.

By compiling our model with the Adam optimizer and utilizing sparse categorical entropy as the evaluation metric, we aim to optimize the model's learning process and assess its accuracy in predicting the summaries accurately. These choices help ensure that our model is trained efficiently and enables us to monitor its performance effectively during training and evaluation phases.

5.3 Training loss and Validation loss

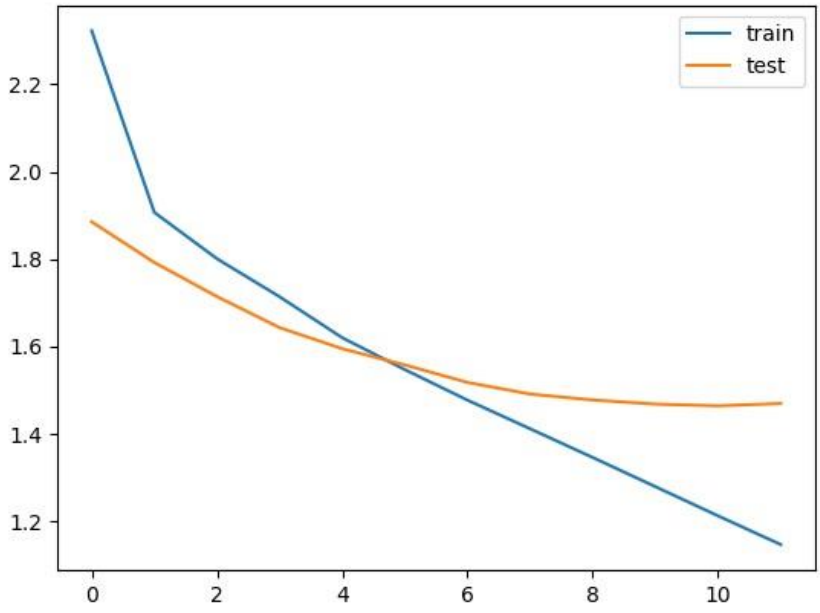

Fig 16: Training and Test loss for 12 epochs

5.4 Training and Testing Accuracy

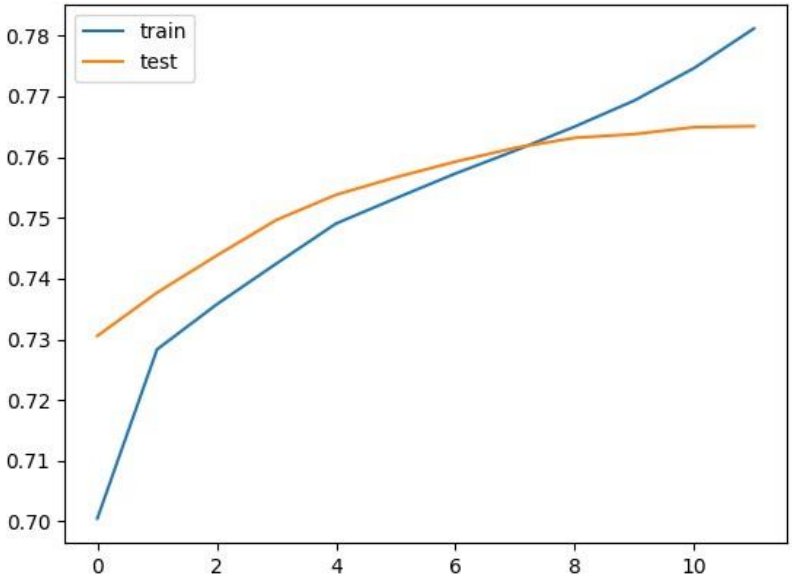

Fig 17: Training and Testing accuracy for 12 epochs

Chapter 6 – Conclusion

6.1 Conclusions from Accuracy graph (fig 17)

The training accuracy of our model is reported to be 0.78, indicating that during the training phase, the model correctly predicts the summaries or labels for approximately 78% of the training examples. This metric provides an insight into how well the model has learned from the training data. On the other hand, the testing accuracy is recorded as 0.76, suggesting that when presented with unseen data from the test set, the model accurately predicts the summaries or labels for approximately 76% of the test examples. The testing accuracy serves as an evaluation metric for assessing how well the model generalizes to new, unseen data. Comparing the training and testing accuracies allows us to gauge if the model is overfitting (performing significantly better on the training data but worse on the test data) or underfitting (performing poorly on both training and test data). In this case, the model seems to achieve similar accuracies on both the training and testing sets, indicating a reasonable level of generalization without overfitting or underfitting concerns.

6.2 Web Application

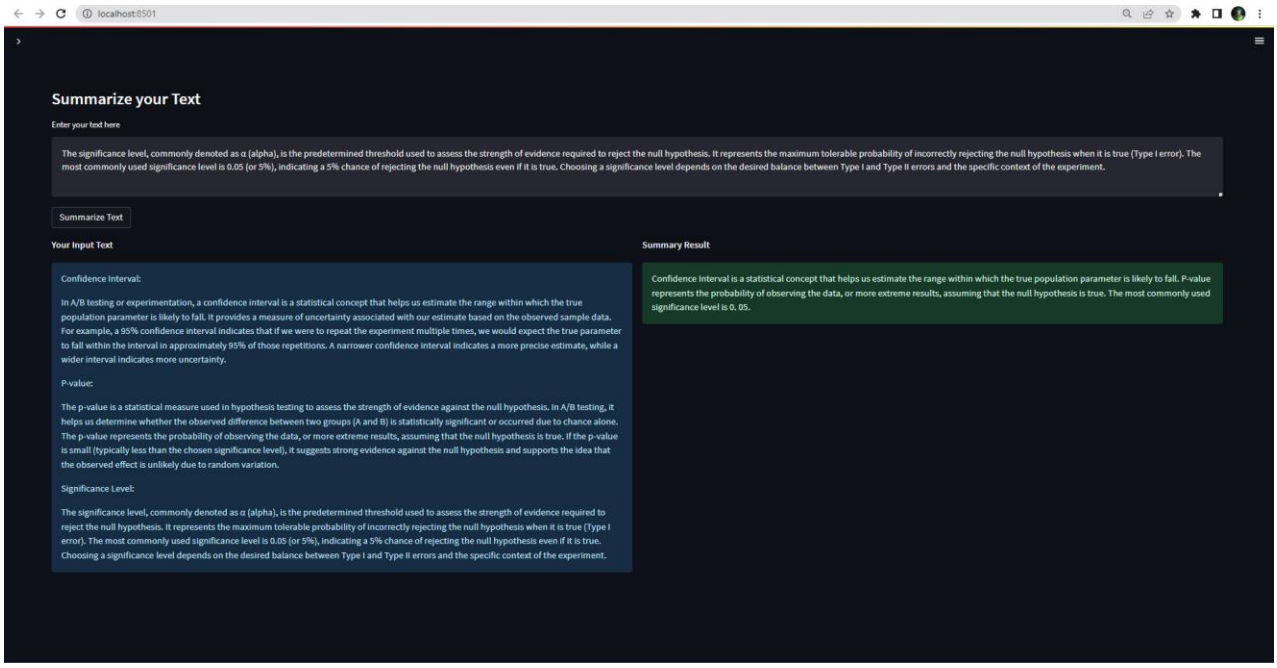

Fig 18. Web App Layout

Input:

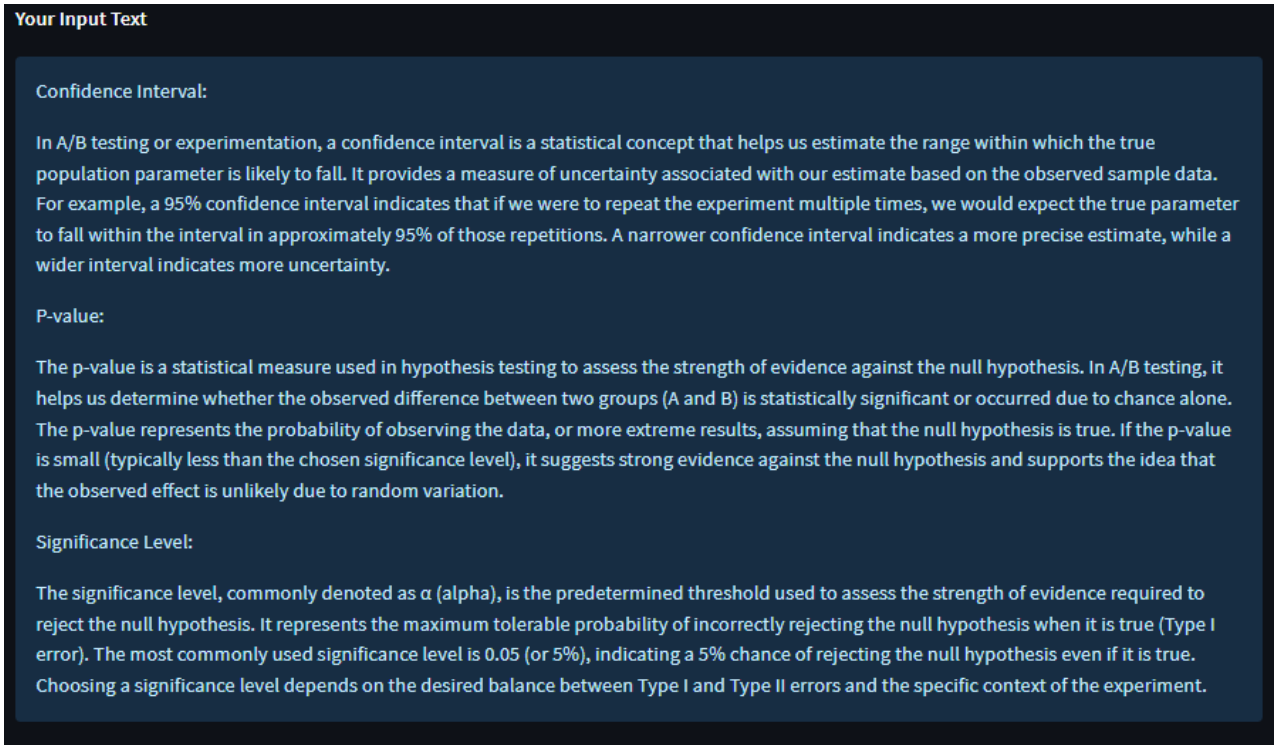

Fig 19. Demo Input Text

Output:

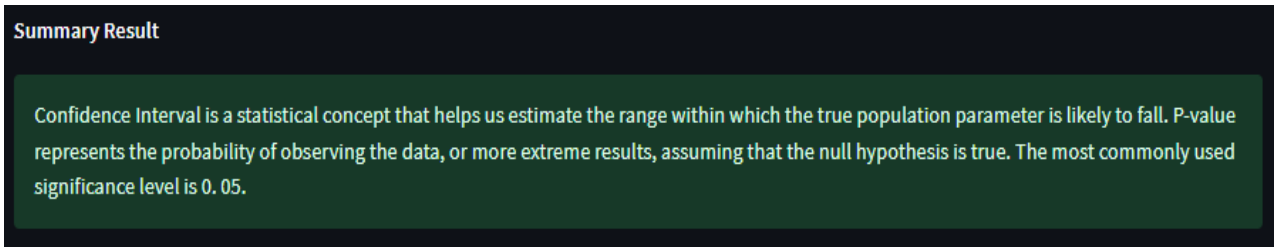

Fig 20. Summary Results

REFERENCES

[1]Anand, Deepa & Wagh, Rupali. (2019). Effective Deep Learning Approaches for Summarization of Legal Texts. Journal of King Saud University - Computer and Information Sciences. 10.1016/j.jksuci.2019.11.015

[2]Berg-Kirkpatrick, T., Gillick, D., & Klein, D. (2011) "Jointly optimizing readability and summarization. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies", Volume 1, 546-556.

[3]Brill, E. Part-of-speech Tagging. Handbook of Natural Language Processing, pages 403-414, 2000

[4]Brownlee, J. "A Gentle Introduction to Text Summarization.": https://machinelearningmastery.com/gentle-introduction-text-summarization/ , [November 29, 2017].

[5]Brownlee, J. "How to Use Metrics for Deep Learning with Keras in Python.": https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/ , [August 09, 2017]

[6]Chen, Y., Zhu, L., Ling, Z., Wei, S., Jiang, H., & Inkpen, D. (2016) "Distraction-based neural networks for document summarization. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies", 1288-1298.

[7]Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019) "BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1" (Long and Short Papers), 4171-4186.

[8]Erkan, G., & Radev, D. R. (2004) "LexRank: Graph-based lexical centrality as salience in text summarization. Journal of Artificial Intelligence Research", 22, 457-479.

[9]Farooq, M. S., Khan, S. A., Abid, K., Ahmad, F., Naeem, M. A., Shafiq3a, M., & Abid, A. (2015). Taxonomy and design considerations for comments in programming languages: a quality perspective. Journal of Quality and Technology Management, 10(2), 167-182.

[10]Farooq, M. S., Riaz, S., Abid, A., Umer, T., & Zikria, Y. B. (2020). Role of IoT technology in agriculture: A systematic literature review. Electronics, 9(2), 319.

[11]Gehrmann, S., Deng, Y., & Rush, A. M. (2018) "Bottom-up abstractive summarization. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing", 4098-4109.

[12]Ketkar, N. "Introduction to Keras. In Deep Learning with Python" : https://link.springer.com/chapter/10.1007/978-1-4842-2766-4_7/ , [February 26, 2018]

[13]Liu, Y., Lapata, M., & Keller, F. (2019) "Text summarization with pretrained encoders. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing" (EMNLP-IJCNLP), 3723-3733.

[14]Nallapati, R., Zhou, B., dos Santos, C. N., Gulcehre, C., & Xiang, B. (2016). "Abstractive text summarization using sequence-to-sequence RNNs and beyond" arXiv preprint arXiv:1602.06023.

[15]Nenkova, A., & McKeown, K. (2012) "A survey of text summarization techniques. Mining Text Data", 43-76.

[16]Paulus, R., Xiong, C., & Socher, R. (2018) "A deep reinforced model for abstractive summarization. Proceedings of the 6th International Conference on Learning Representations (ICLR)"

[17]Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019) "Language models are unsupervised multitask learners" OpenAI Blog, 1(8).

[18]See, A., Liu, P. J., & Manning, C. D. (2017) "Get to the point: Summarization with pointer-generator networks. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics" (Volume 1: Long Papers), 1073-1083.

[19]Tehseen, R., Farooq, M. S., & Abid, A. (2020). Earthquake prediction using expert systems: a systematic mapping study. Sustainability, 12(6), 2420.

[20]Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). "XLNet: Generalized autoregressive pretraining for language understanding. Advances in Neural Informati