AUTOMATED TEXT SUMMARIZER

A Thesis

B.Tech Project

by

Keshav Sharma , Akshat Dubey
(2019BITE017, 2019BITE049)

DEPARTMENT OF INFORMATION TECHNOLOGY
NATIONAL INSTITUTE OF TECHNOLOGY
SRINAGAR - 190006 (INDIA)

June 2023

AUTOMATED TEXT SUMMARIZER

A PROJECT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

by

Keshav Sharma , Akshat Dubey
(2019BITE017, 2019BITE049)

Under the guidance

of

Dr. Janibul Bashir

DEPARTMENT OF INFORMATION TECHNOLOGY
NATIONAL INSTITUTE OF TECHNOLOGY
SRINAGAR - 190006 (INDIA)

June 2023

Certificate

We/I hereby certify that the work which is being presented in the project titled **AUTOMATED TEXT SUMMARIZER** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology and submitted in the Department of Information Technology, National Institute of Technology Srinagar, is an authentic record of my own work carried out during a period from August 2022 to June 2023 under the supervision of Dr. Janibul Bashir, Assistant Professor, Department of Information Technology, National Institute of Technology Srinagar.

The matter presented in this project report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Keshav Sharma **Akshat Dubey**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: June 14, 2023

Dr. Janibul Bashir
Department of Information Technology

The project Viva-Voce Examination of Keshav Sharma and Akshat Dubey, has been held on June 14, 2023.

Signature of Supervisor Signature of External Examiner

Acknowledgements

We would like to express our sincere gratitude to our department head **Dr. Shabir Ahmad Sofi** for allowing us to work on this project **AUTOMATED TEXT SUMMARIZER** as it has been an enriching and fulfilling experience for us.

We deeply express our sincere thanks to our project guide **Dr. Janibul Bashir** for his invaluable guidance, constant support, and expert advice throughout the project.

We are also thankful to all of the department faculty members for their help, encouragement, valuable insights and feedback.

Lastly, we would like to acknowledge all other individuals, not mentioned by name, who have directly or indirectly contributed to the successful completion of my B.Tech project. Thank you all for being an essential part of this journey and for your contributions that have made this project possible.

Keshav Sharma

Akshat Dubey

Abstract

Information overload is one of the most important problems brought on by the Internet's explosive expansion. Because there is a wealth of information on any topic on the Internet, condensing the pertinent information into a summary will be helpful to many individuals. Massive quantities of text are difficult for people to manually summarise. Thus, there is now a greater need for summarizers that are more sophisticated and potent. Researchers have been attempting to enhance methods for developing summaries that are as close to human-generated summaries as possible.

This project offers a thorough, up-to-date information about text summarizing principles, including available methods of text summarization (both abstractive and extractive methods), proposed model, assessment measures, and potential future research areas.

We have designed a text summarizer tool which can be used to generate summaries for all types of text like long legal texts or for shorter reviews, etc.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Written language has been the most significant medium for preserving and transmitting knowledge for more than 5000 years. Furthermore, in the present digital era, the quantity of available texts, particularly online, is consistently increasing due to the decreasing costs associated with producing, storing, and reproducing digital texts. This exponential growth in data and information spans across various industries [4].

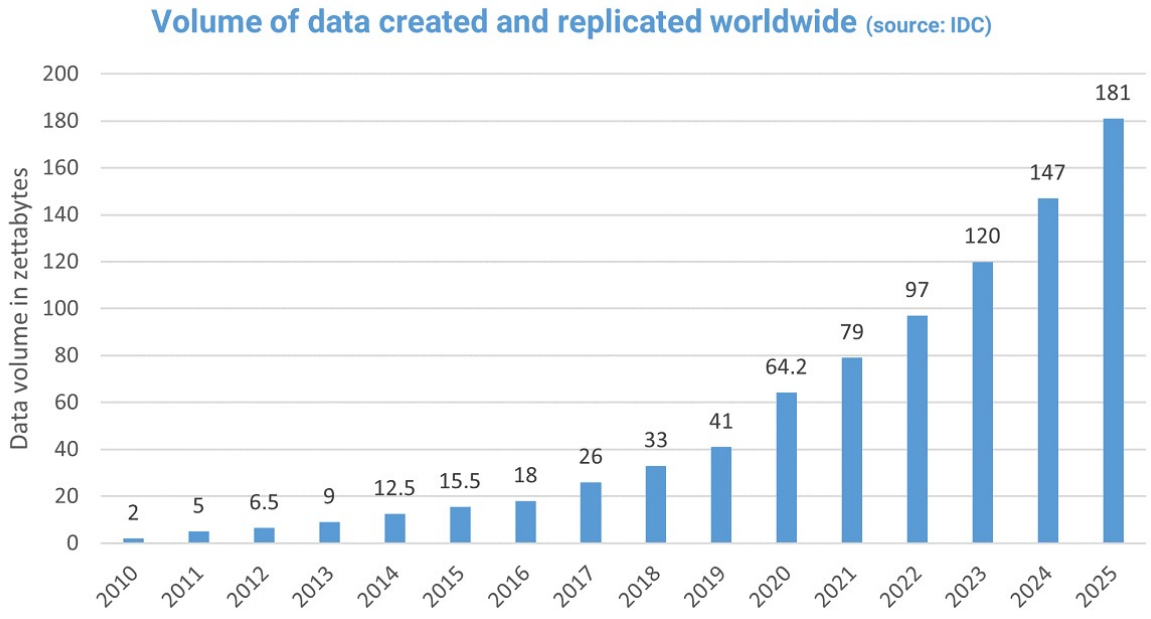


Figure 1.1: Data Surge: Alarming Growth at a Glance [1]

Let's examine some startling data-related statistics presented by Arne von See (2021) in [1], as depicted in the Fig. 1.1. Some statistics about it include the fact that 90% of the world's data

has been produced in the previous two years. Only 12% of the data that firms have access to is actually viewed. Inaccurate data costs the US $3.1 trillion annually. The total amount of data produced by 2025 will exceed 180 zettabytes. It would take a person around 181 million years to download all of the content available on the internet today.

In order to assist individuals in managing the overwhelming amount of information and enhance their comprehension of it, it becomes essential to extract valuable information from the vast volume of unstructured data. As a result, summarization plays a crucial role in this process.

There are several additional valid arguments supporting the use of automated document summarization, including:

- Summaries help reduce the time required for reading.
- Summaries assist in the process of document selection during research.
- The utilization of automatic summarization leads to improved indexing outcomes.
- Automatic summary systems exhibit reduced bias compared to human summarizers.
- Personalized summaries play a crucial role in question-answering systems as they deliver tailored information to individual users.

Text summary can be categorised into two categories:

1. Manual: The method of manual summarization entails the manual production of summaries by human experts. It would take a lot of time, effort, money, and worry for people to do this task. As a result, automation was required to make this task quicker, less expensive, and repeatable [5].
2. Automatic: Automatic text summarization refers to the automated process of condensing textual information or sources by retaining the crucial elements while reducing its overall length. It entails condensing a lengthy written document into a few words or a paragraph that captures the essence of the material.

So, the purpose of summarizing systems is to create a brief, cohesive summary that will enable users to grasp the input's content without having to read the complete text.

The practical application of automatic text summarization spans numerous industries. It can be used by businesses to create reports, by students and researchers to find the most significant

concepts that apply to their work, by doctors to summarise patients' medical information, or by journalists to cover a wide range of sources, identifying key facts and opposing viewpoints.

Furthermore, the classification of summarization techniques can be based on the specific method employed, resulting in two distinct types:

1. Extractive summarization: Extractive summarization techniques generate summaries by selecting and combining sentences directly from the original text. These methods focus on identifying the significant sentences that should be included in the summary. The goal is to determine the importance of each sentence in order to create an effective summary.

2. Abstractive summarization: Abstractive summarization techniques generate summaries by rephrasing the original text using new sentences, even if those sentences are not explicitly present in the input. This is different from extractive methods, which extract and concatenate important text units from the original text. Similar to abstractive summarization, humans also create summaries in their own words, capturing the essence of the text.

1.1 Problem Statement

Our problem statement includes developing an automated text summarizer which can be used for summarizing different types of text:

1. Long text: Some common examples of long text includes text containing legal information like terms and conditions, case proceedings, etc. These legal texts/documents are mostly very lengthy. Also, preparing a legal case summary is a challenging task that takes lawyers many hours but can be completed in a few minutes by a computer. Therefore, automatic text summarization is required in legal domain.

2. Short text: Short text do not generally require summaries. However, when available in abundance, they also need to be summarized for retrieving important information in less time and less words. For example, as electronic commerce continues to expand, there has been a significant increase in the volume of customer online reviews available on the internet. These reviews, which are typically shorter in length, provide valuable insights for product development and marketing purposes. However, due to a large volume of reviews, we also need summarizers for generating short (3 to 4 words) summary for each of these reviews so that they can be analyzed easily.

1.2 Objectives

Our aim is to develop a hybrid system which can be used for summarization of all types of text irrespective of their length or origin.

This system will use both extractive and abstractive based summarization techniques which will vary according to the input text and user requirements whether user wants extractive or abstractive summarization. So, a text either short or large in length, will be provided by the end user, and on the basis of its length, we want our model to select the appropriate summary generating method for it.

Additionally, we also want to develop a dataset comprising of judgements of various cases of Supreme Court of India along with their respected summaries. This dataset is used to evaluate our model as compared to human generated summaries. It can also be used for further improvement and research in the study of text summarization.

In our work, we utilized legal judgments as long textual inputs. Our objective was to generate a concise and informative legal brief as the output, which effectively represents the pertinent and valuable facts related to the case. The generated brief should encompass all the necessary information, including the crucial facts and events, allowing for a comprehensive understanding of the case without the need for assistance from legal professionals. Considering the substantial length of these legal texts, averaging around 20,000 to 30,000 words, our aim was to develop a model that is both efficient and succinct in producing summaries. We sought to ensure that the model can deliver an effective summary within an optimal timeframe.

Similarly, we also want our model to predict brief and concise summaries for shorter text like reviews. In this case, we want our model to take reviews as input and our model, after training on this, should generate summaries which are shorter in length (about 3 to 4 words). These generated summaries should clearly identify what the customers feel about a product (good or bad) in two or three words.

1.3 Organization

This report is organized as follows. We have discussed existing work and problems with existing techniques in Chapter 2. Our objectives that we wish to achieve are mentioned in Chapter 3. In Chapter 4, we disussed about the models and their theoretical working. Then, in Chapter 5, we have discussed about the evaluation of the generated summaries. The implementation

details of our model along with some code snippets and output are given in Chapter 6. Finally, in Chapter 7, we have summarized our project by giving conclusion of our work and possible future work.

Chapter 2

Review of Existing Work

2.1 Literature Review

Various general purpose and domain specific text summarizers have been proposed in the recent years. Some of the models which are designed for usage in legal domain are given below with a short brief about how they work.

2.1.1 FLEXICON

One of the pioneering systems in this field is the "Fast Legal EXpert CONsultant" (FLEXICON) developed by Gelbart and Smith (Gelbart and Smith, 1991a). FLEXICON is a keyword-based system that utilizes a extensive term database to identify significant sections of text (Gelbart and Smith, 1991b). It laid the foundation for subsequent text summarization approaches in the legal domain. The structured knowledge representation model created for the FLEXICON system, can be used as an external representation to summarise legal text for quick evaluation of search results as well as an internal knowledge representation scheme used in conjunction with statistical ranking.

2.1.2 SALOMON

SALOMON is a cosine similarity-based algorithm presented by Moens (Moens et al., 1999). Its goal is to automatically summarise criminal cases in Belgium in order to make the vast

amount of current and future cases easier to access. As a result, methods are created for finding and collecting pertinent data from the cases. When creating SALOMON, a dual methodology was used: on the one hand, the instances were processed using supplementary knowledge to interpret structural patterns and features, and on the other, they were treated using index term occurrence statistics. As a result, SALOMON first organises and categorises the instances before extracting the most pertinent text from the cases [6].

2.1.3 Letsum

A prototype system named LetSum, short for Legal Text Sum-marizer, structures the thematic framework of a judgment into four key themes: Introduction, Context, Juridical Analysis, and Conclusion. The pertinent sentences for each theme are then determined. It cover both the statistical and human evaluation of the provided summaries based on jurist judgement. The system appears to function well thus far when compared to previous summarization technologies, according to the results [7].

2.2 Problems with Existing Techniques

There have been numerous automatic text summarizing methods proposed up to this point. These include a large number of other suggested approaches and algorithms, such as LEXA, CaseSummarizer, SUMMARIST, and SMMRY. However, due to the intricacy of the cases and the sheer volume of them, summarizing lengthy legal cases is still difficult for some of the available systems.

Also, as dataset for legal text along with their summaries is not easily available, so building a new model and training it on this legal dataset is equally challenging. Additionally, there is no effective technique for generating three to four words summary for a short text.

Therefore, it is crucial to come up with a system that can generate effective summaries both for longer and shorter texts.

Chapter 3

Work Plan

3.1 Dataset

We have developed a dataset of judgment-summary pairs which contains decisions of the Supreme Court of India along with their summaries. This dataset was created with the help of Indian Supreme Court case proceedings available online.

Some of these case verdicts contain headnotes, so, we have used headnotes as summary and the remaining text as judgment for our dataset.

Our dataset contains total 7030 rows and total 3 columns. Each of the fields/columns in our dataset are described below:

1. **Judgment_ID**: This entry stores a number representing the identifier for a particular judgment. It is used for indexing each row in our dataset with respect to its source.

2. **Judgment**: It stores the original court decison/judgment, proceedings and verdicts related to the corresponding 'Judgment_ID'.

3. **Summary**: It stores the summary/headnotes for the text present in the 'Judgment' field for the respected 'Judgment_ID'.

Along with this legal dataset, we have also used anothe dataset which consists of customer reviews and their human-generated summaries.

3.2 Methodology

3.2.1 Text Summarization

Text summarization is the procedure of compressing a lengthy textual document into a coherent and succinct summary that effectively captures the essential points and concepts of the original text. This is achieved by giving prominence to the significant sentences within the text.

3.2.1.1 Types of text summarization

Text summarization can be broadly categorized into two main approaches:

1. **Extractive Summarization**: The extractive summarization approach involves identifying key sentences or phrases from the original text and extracting them to form the summary. The diagram below provides an illustration of the extractive summarization process.

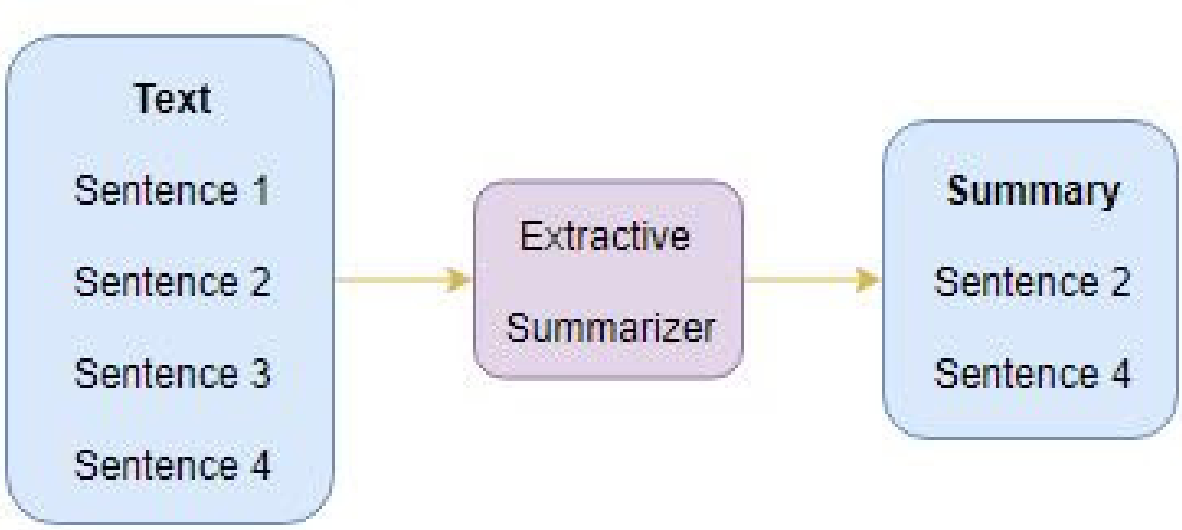


Figure 3.1: Extractive Summarization

2. **Abstractive Summarization**: This strategy is quite intriguing. Here, we take the original content and create new sentences. In contrast to the extractive approach where we only use existing phrases from the original text, abstractive summarization generates sentences that may not be directly present in the original text. The approach for this strategy is illustrated in the figure below:

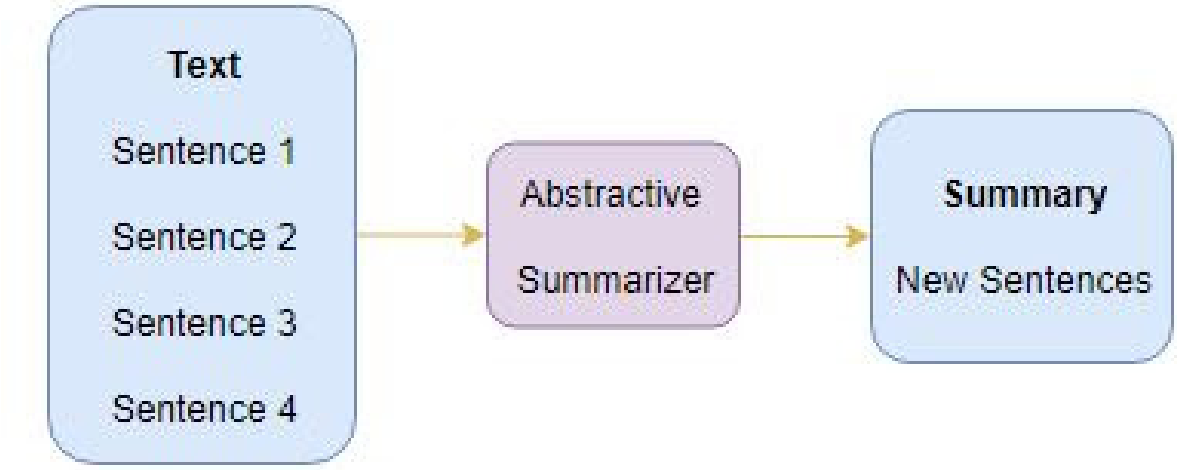


Figure 3.2: Abstractive Summarization

Before understanding the model used for abstractive text summarization, let us first understand the below mentioned concepts which are used in our model.

3.2.2 Introduction to Sequence-to-Sequence Modeling

A Sequence-to-Sequence model is a type of model that is capable of generating a new sequence of elements based on an input sequence of objects, such as words, letters, or time series data. The 'black box' shown in Fig. 3.3 is a complicated system made up of several Recurrent

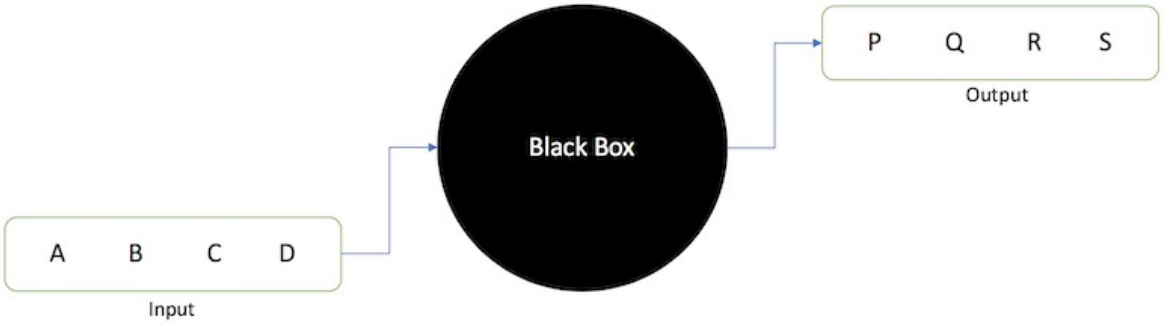


Figure 3.3: Black Box

Neural Networks (RNNs) or LSTMs. It first generates a fixed length vector representation of the variable length input string (in the case of Sequence-to-Sequence for transformation of text). Then, the output string is generated using this vector [8].

For each problem involving sequential information, we can develop a Seq2Seq model. Some extremely popular uses of sequential information include Sentiment analysis, Neural Machine Translation, and Identification of Named Entities.

For example, in neural machine translation, text in one language is the input and a text in some other language is the output. This can be visualized through the following illustration:

We love dancing $\longrightarrow$ Nous aimons danser

Our objective is to develop a text summarization system that takes a text in the form of sequence of tokens/words as input and generates a concise summary as output. This can be framed as a Many-to-Many Sequence-to-Sequence problem. The architecture of a typical Sequence-to-Sequence model is depicted in the diagram below:

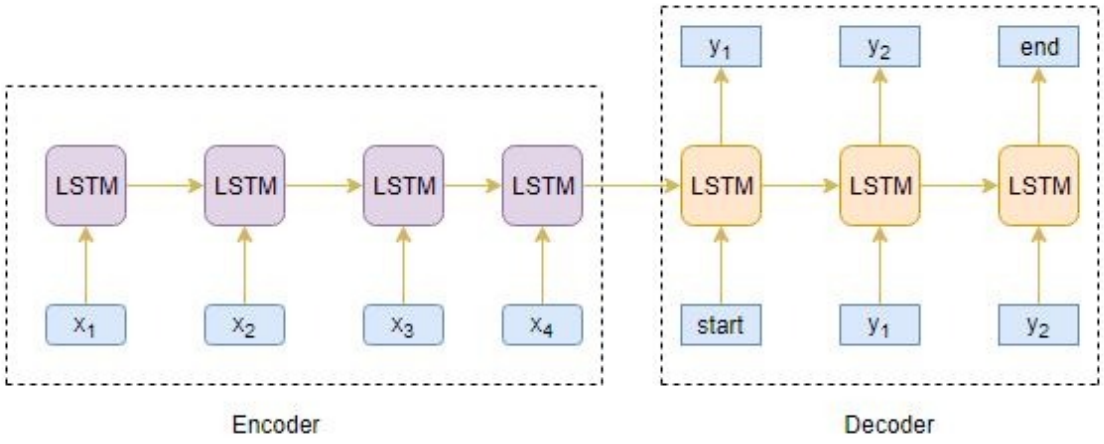


Figure 3.4: Architecture of Sequence-to-Sequence model

Two main components of a Sequence-to-Sequence model are:

1. Encoder
2. Decoder

3.2.3 Encoder-Decoder arcitecture

We can examine the "black box's" contents which are shown in the Fig. 3.3 above. The encoder and decoder architecture is made up of layered RNNs so that a word sequence can be easily encoded into an encoded sequence and the encoded sequence can be sent to a decoder network to

produce an output. This is what Sequence-to-Sequence models rely on [8]. The input sequence is fed word-for-word into the encoder after being tokenized, which changes it from a group of words into a group of integers. The encoder transforms the sequence into a new, abstract state before sending it to the decoder. This new state is used as a foundation for creating an output sequence (such as a shortened form of original text).

Typically, for selecting the encoder and decoder components, Gated Recurrent Unit or Long Short Term Memory (variants of Recurrent Neural Networks (RNNs)) are preferred. This is due to the fact that they can capture long-term dependency by getting around the vanishing gradient issue.

3.2.4 Long Short-Term Memory (LSTM)

Input, memory, forget, and output gates, form the LSTM architecture. These gates together form the repeating unit [2]. Despite sharing a similar structure with RNNs, LSTMs allow for long-term information flow through the exchange of information between the four gates.

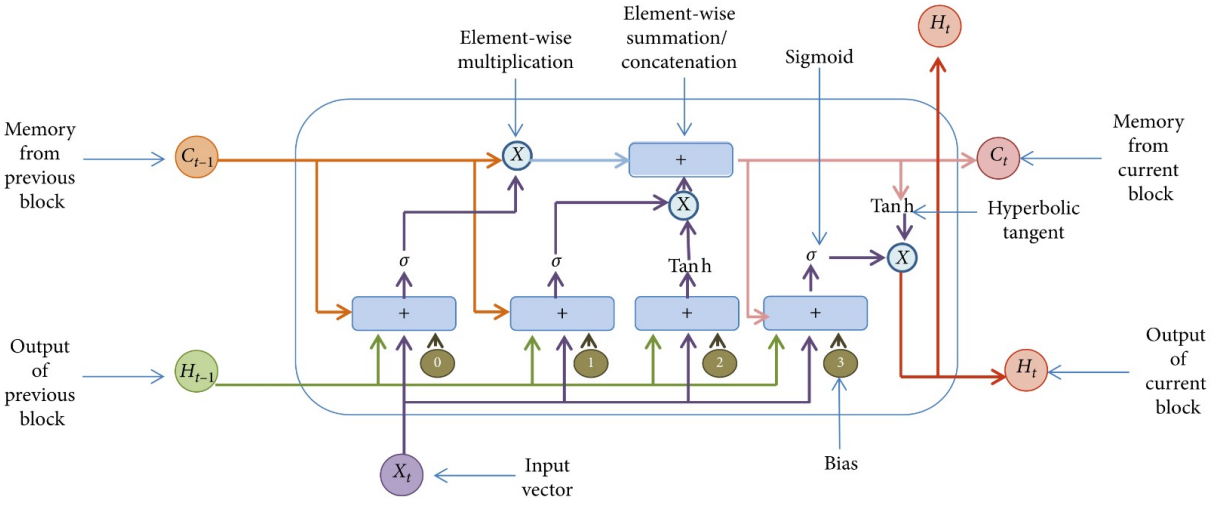


Figure 3.5: LSTM [2]

Each LSTM unit's four gates, which are seen in Fig. 3.5 above, are discussed below:

1. **Input Gate**: Unlike succeeding steps, where the input is the result (memory cell's content) of the preceding step, the input at first timestep is a vector. This vector has been randomly initialised. With the forget gate's output, the input is always multiplied element-by-element.. The present memory gate's output is increased by the multiplication result.

2. **Forget Gate**: The forget gate is implemented as a neural network with a single layer and utilizes a sigmoid activation function. The decision about whether or not the the previous state's information should be recalled, depends on the sigmoid function's result. The sigmoid value plays a crucial role in deciding whether the previous state is preserved or discarded. When the sigmoid fuction's value is 1, the last state is retained, while it is disregarded when the sigmoid value is 0.

3. **Memory Gate**: It regulates the influence of previously stored information on newly acquired knowledge. Two neural networks form this memory gate. Its second neural network, which generates new information, utilizes tanh activation function. The first network has a similar structure to the forget gate. However, this network has a different bias.

4. **Output Gate**: It control how much fresh data is sent to the following LSTM unit. The prior hidden state, the input vector, the bias, and the most recent information, and are all inputs to the output gate. It is also a neural network that uses a sigmoid activation function. For producing the output of the current block, the hyperbolic tangent tanh of the recently learned information is multiplied with the sigmoid function's output.

3.2.5 Encoder

An encoder long short-term memory model (LSTM) comprehensively reads the entire input sequence, where each word is sequentially fed into the encoder. At each timestep, the information is processed, allowing for the capture of contextual information from the input sequence.

Upon encountering a new token at each time step, the hidden state is refreshed with the latest information. Once the input sequence concludes, the encoder generates a fixed-length representation of the input, referred to as the encoder vector, irrespective of the input's length. For initializing the decoder, the final hidden state, given by the encoder vector, is used.

The below diagram illustrates the above described process:

The decoder is initialised with T=the cell state (c_i) and hidden state (h_i) from the prior time step. This approach is adopted because the decoder and encoder in the LSTM architecture operate as distinct components.

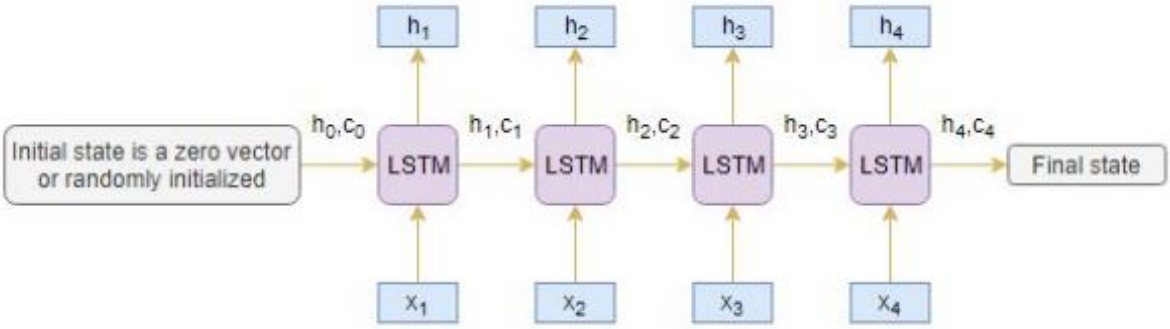


Figure 3.6: Working of encoder

3.2.6 Decoder

The decoder, consisting of an LSTM network, generates predictions for the target sequence. This is done by sequentially processing each word of the sequence, with a time delay of one timestep in the prediction process. When given the previous word in the sequence, The decoder is trained to predict the following word. The decoder is initialized by using the encoder vector

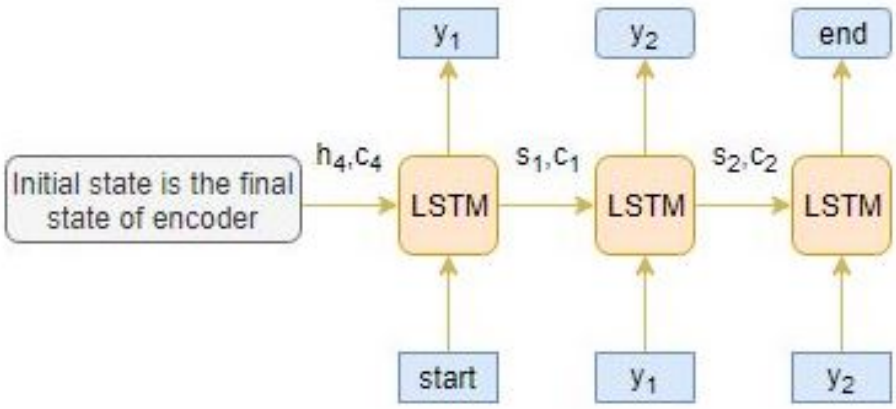


Figure 3.7: Working of decoder

as its initial hidden state and incorporating a <start> token, which signifies the beginning of the output sequence.

Before feeding the target sequence into the decoder, special tokens called <start> and <end> are added. During the decoding of the test sequence, the target sequence is unknown. Hence, the prediction process begins by providing the decoder with the initial word, which is always the <start> token. The sentence is then terminated with the <end> token.

Following the training phase, the model is evaluated using new source sequences that have

unknown target sequences. To decode a test sequence, it is necessary to configure the inference architecture, as illustrated in Fig. 3.8.

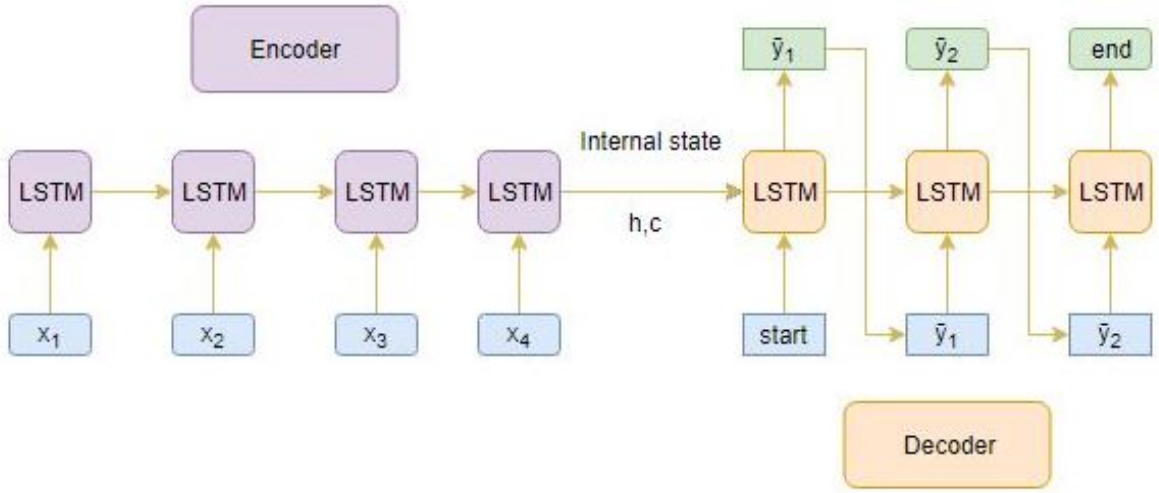


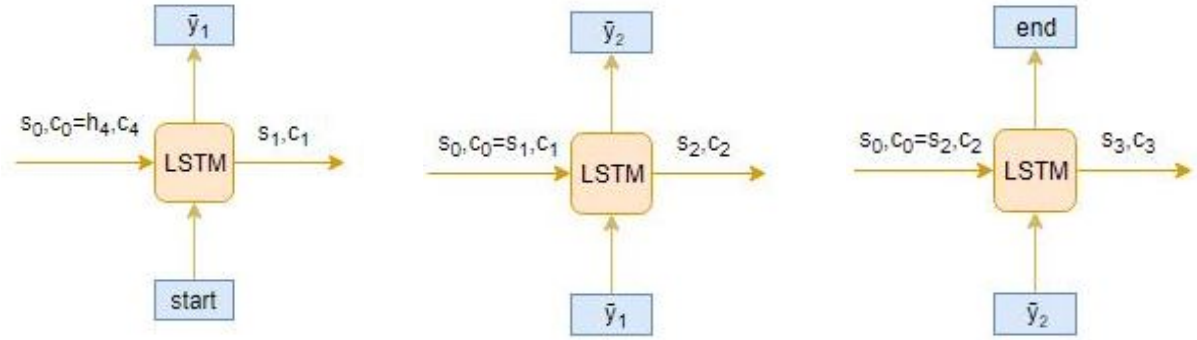
Figure 3.8: Inference architecture

For the test sequence to be decoded properly, the below mentioned steps need to be followed:

1. Process the complete input sequence through the encoder, and subsequently utilize the internal states of the encoder for initializing the decoder.
2. Provide the decoder with the input of the <start> token.
3. Execute the decoder for a single timestep using its internal states.
4. The result will be a probability distribution for the subsequent word. Out of all options, the one (word) which has the highest probability, word will be chosen.
5. For the following timestep, provide the decoder with the selected word as input, and update the internal states accordingly. The process from steps 3 to 5 is repeated until either the <end> token is generated or the maximum length of the target sequence is attained.

Consider the test sequence as $[x_1, x_2, x_3, x_4]$. For this test sequence, the inference procedure will function in the following manner:

1. Convert the test sequence into internal state vectors through the encoding process.
2. Observe the predictions made by the decoder (at each timestep) for the target sequence, as depicted in Fig. 3.9



(a) Timestep: t = 1 (b) Timestep: t=2 (c) Timestep: t=3

Figure 3.9: Target sequence at different timesteps

3.2.7 Limitations of the Encoder and Decoder Architecture

Even though GRU or LSTM units in the encoder-decoder architecture are effective in capturing contextual information from inputs, they tend to encounter difficulties as the input sequences become longer [2]. Lengthy input sequences can lead to the encoder losing vital context from earlier parts of the sequence in the resulting final state vector. The hidden states of the encoder RNN are updated by fresh data on each iteration, gradually drifting away from the initial inputs. The main issue here is that the lengthy input text's context has been reduced to a single state vector.

So how can we solve the issue of lengthy sequences? The idea of an attention mechanism enters the picture at this point. Instead of analysing the full sequence, it seeks to predict a word by focusing only on a few particular sequence elements.

3.2.8 Attention mechanism

Let us attempt to comprehend the functioning of the Attention Mechanism using the following example:

Source sequence: "What is your favorite sport?"
Target sequence: "I like playing basketball."

In this situation, there exists an association between the third word, "your", found in the source sequence and the first word, "I", present in the target sequence. Similarly, the second word of the target sequence, "like", is linked to the fourth word of the source sequence, "favorite". [8].

As a result, rather than concentrating on every word in the source sequence, we can concentrate specific source sequence components that lead to the target sequence. The attention mechanism is based on this fundamental concept.

The interface present between the encoder and decoder is termed as attention layer. The attention mechanism ensures that information from each encoder's hidden state is relayed to the decoder. Using this configuration, the model can concentrate on important segments of the input sequence and subsequently capture their alignment. This makes it easier for the model to handle lengthy input sentences.

The diagram for attention layer is shown below:

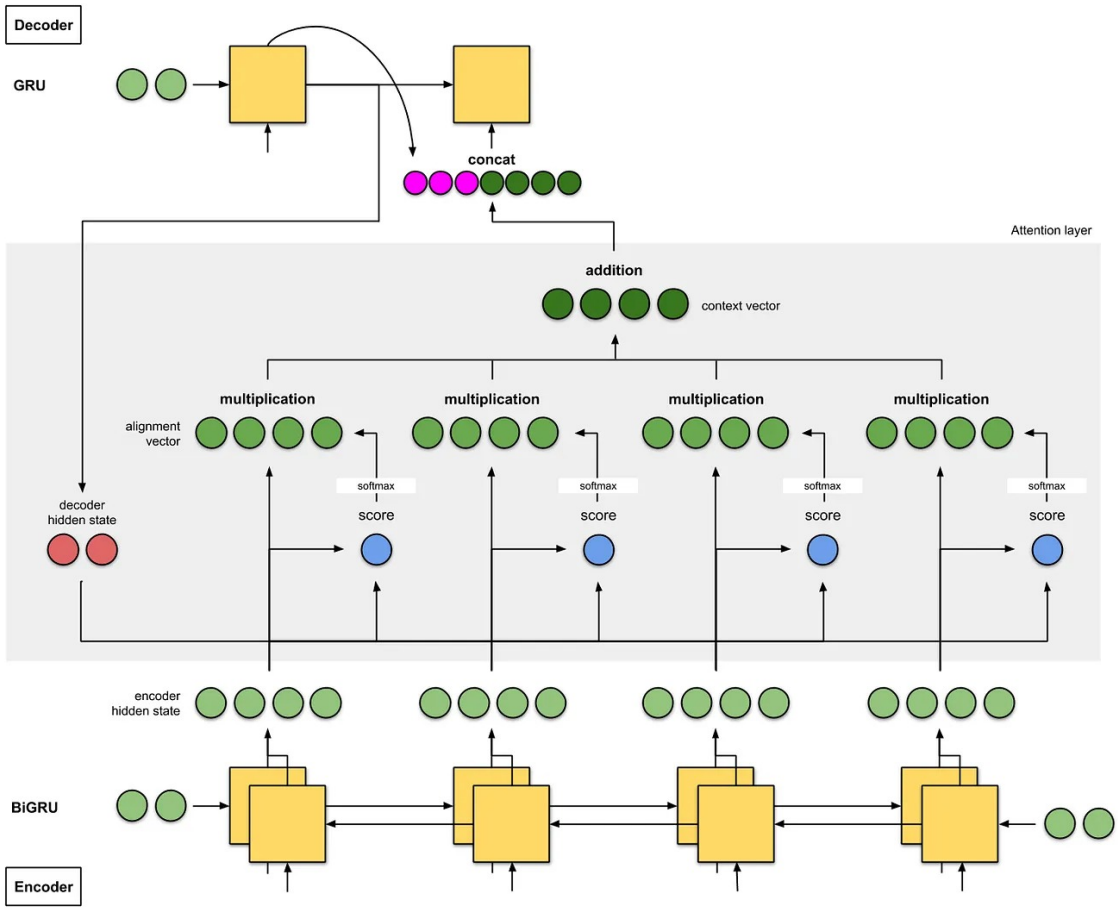


Figure 3.10: Attention layer

We have the current hidden state of the deocder in addition to the four encoder hidden states in our case. (Note: The initial time step of the decoder receives as input the most recent aggregated encoder hidden state. The first decoder hidden state is the result of the decoder's first time step).

The attention mechanism employs a score function, also known as the alignment score function or alignment model, to calculate a scalar score. In the given Fig. 3.10, the score function is implemented as a dot product between the hidden states of the decoder and the encoder. To ensure that the softmaxed scores (scalar) add up to 1, a softmax layer is applied to the scores. These softmaxed scores represent the attention distribution. Multiplying each encoder hidden state by its corresponding softmaxed score yields the alignment vector or annotation vector [9]. The attention distribution is depicted by these softmaxed scores. By summing the alignment vectors, the context vector is formed. Essentially, the context vector is derived from the information contained in the alignment vectors from the previous phase.

Then, the context vector is fed into the decoder.

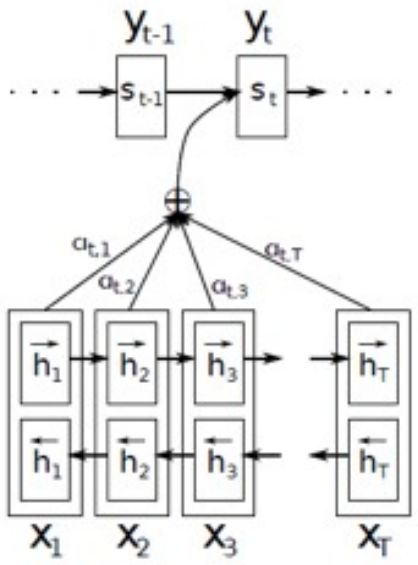


Figure 3.11: Feeding context vector into decoder [3]

The bidirectional LSTM employed in this approach generates a sequence of annotations represented as $(h_1, h_2, \ldots, h_{Tx})$ for each input sentence. Each vector, such as $h_1, h_2, \ldots$, is a concatenation of the hidden states from both the backward and forward directions in the encoder [3].

$$h_j = [\overrightarrow{h}_j^\top ; \overleftarrow{h}_j^\top]^\top$$

Next, the context vector c_i for the output word y_i is created by computing the weighted sum of the annotations.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

The weights α_{ij} are computed by a softmax function. The formula for the same is given below:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}$$

$$e_{ij} = a\left(s_{i-1}, h_j\right)$$

In this case, the output score e_{ij} is obtained from a feedforward neural network, which is defined by the function a, aiming to capture the alignment between the input at position j and the output at position i.

Before diving further into the actual implementation of our model for abstractive summarization, let us first discuss our proposed strategy for extractive summarization. For this, we first need to understand some important concepts that are explained below.

3.2.9 Term Frequency (TF)

It is calculated by dividing the number of occurrences of a specific word (w) in a document (d) by the total number of words in the document. This straightforward calculation allows us to quantify the frequency of a word within the document [10].

$$\mathrm{tf}(w,d) = \frac{\text{occurence of } w \text{ in document } d}{\text{total number of words in document } d}$$

For example, the term frequency ratio of the word "the" would be (3/7) in a sentence with total seven words and three instances of the word "the" in it.

3.2.10 Inverse Document Frequency (IDF)

IDF represents the significance of a word in a corpus D. It is calculated by dividing the overall corpus's document count by the proportion of those that contain a certain word [10].

$$\mathrm{idf(w,D)} = \log\left(\frac{\mathrm{N}}{\mathrm{f(w,D)}}\right)$$

In this context, N represents the total number of documents in the corpus D, and $\mathrm{f(w,D)}$ indicates the number of documents in which the word w is present.

3.2.11 Term Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF can be computed by multiplying the term frequency (TF) of a word with its inverse document frequency (IDF). This calculation assigns higher significance to words that are infrequent in the overall corpus but appear frequently within a specific document.

$$\text{tfidf}(w,d,D) = \text{tf}(w,d) \times \text{idf}(w,D)$$

3.2.12 Term Frequency - Inverse Document Frequency matrix

TF-IDF is a statistical formula used to convert text documents into vectors, representing the relevancy of each word. This transformation is based on the importance of the term within the document and its frequency in the entire corpus. The matrix holding the data on the least and most relevant terms in the document is created using the bag of words approach. Machine learning, topic modelling, and NLP jobs all benefit greatly from TF-IDF. Algorithms can better predict results by knowing about a word's importance.

3.2.13 Similarity Matrix

The similarity metric is a way to compare the similarity of two data objects. A distance with dimensions that represent object features serves as the similarity measure in a data mining or machine learning setting. The similarities between the features are greatest when there is little separation between them. On the other hand, a significant distance will produce little similarities.

We measure similarity matrix using cosine similarity.

3.2.14 Cosine Similarity

The similarity between two vectors is found using the cosine similarity metric [10]. In particular, it ignores variations in the magnitude or scale of the vectors and compares the similarity in their direction or orientation. To obtain a scalar using inner product multiplication, both vectors must belong to the same inner product space. The cosine of the angle between two vectors is used to determine how similar they are.

The mathematical definition of cosine similarity is the vectors' dot product divided by their magnitude. For instance, the following formula can be used to determine how similar two vectors A and B are:

$$similarity(A,B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

3.2.15 Summing Similarities

The similarity matrix is then summed along the rows (axis=1) to obtain a score for each sentence. This sum represents the cumulative similarity of a sentence with all other sentences. Higher scores indicate more important sentences. The resulting scores represent the importance or centrality of each sentence within the text. Sentences with higher scores are considered more significant.

3.2.16 Ranking Sentences

The sentences are ranked based on their scores. The higher the score, the higher the rank. This is done by sorting the scores in descending order and keeping track of the original indices.

3.2.17 Selecting Top Sentences

The top N sentences with the highest scores are selected as the summary sentences. These sentences are typically the most important and relevant ones based on their similarity to other sentences.

3.2.18 Generating Summary

The selected summary sentences are combined to form the final summary. They can be concatenated together using a space or any other desired separator.

For understanding these terms, lets take an example.
Document 1: "The cat jumped on the table"
Document 2: "The dog played in the park"
Document 3: "The cat chased the mouse"

Now, we will compute the TF-IDF scores for every word in each document. The calculated TF-IDF scores for all words present for each document is shown in Table 3.1.

Word	Document 1 (D_1)	Document 2 (D_2)	Document 3 (D_3)
cat	0.176	0	0.176
chased	0	0	0.176
dog	0	0.176	0
jumped	0.176	0	0
mouse	0	0	0.176
on	0.176	0	0
park	0	0.176	0
played	0	0.176	0
table	0.176	0	0

Table 3.1: TF-IDF Scores

The corresponding TF-IDF matrices can be found from table 3.2.

Document	TF-IDF matrix
D_1	TF-IDF(D_1) = [0, 0.176, 0, 0, 0, 0, 0.176, 0.176, 0]
D_2	TF-IDF(D_2) = [0, 0.176, 0, 0, 0, 0, 0.176, 0.176, 0]
D_3	TF-IDF(D_3) = [0.176, 0, 0.176, 0, 0.176, 0, 0, 0.176, 0]

Table 3.2: TF-IDF matrices

Let us assume that x be the cosine similarity between D1 and D2.

$$x = \text{Cosine Similarity}(D1, D2)$$

Then, it can be calculated as follows:

$$\begin{aligned} x &= \frac{\text{TF-IDF}(D1) \cdot \text{TF-IDF}(D2)}{||\text{TF-IDF}(D1)|| \cdot ||\text{TF-IDF}(D2)||} \\ &= \frac{(0.176 \cdot 0 + 0 \cdot 0.176 + 0 \cdot 0 + 0.176 \cdot 0 + 0 \cdot 0 + 0.176 \cdot 0 + 0 \cdot 0.176 + 0 \cdot 0.176 + 0.176 \cdot 0)}{||\text{TF-IDF}(D1)|| \cdot ||\text{TF-IDF}(D2)||} \\ &= 0 \end{aligned}$$

So, Cosine Similarity$(D1, D2) = x = 0$

The resulting cosine similarity matrix would be: $\begin{bmatrix} 1.0 & 0.0 & 0.577 \\ 0.0 & 1.0 & 0.0 \\ 0.577 & 0.0 & 1.0 \end{bmatrix}$

We will now calculate the sum of similarity scores for each document.

$$\begin{aligned} \text{Sum for D1} &= 1.0 + 0.0 + 0.577 &&= 1.577 \\ \text{Sum for D2} &= 0.0 + 1.0 + 0.0 &&= 1.0 \\ \text{Sum for D3} &= 0.577 + 0.0 + 1.0 &&= 1.577 \end{aligned}$$

At last, we will rank the documents based on their sum of similarity scores in descending order.

$$\begin{aligned} &\text{Rank 1:} \quad D1 \quad (\text{Sum} = 1.577) \\ &\text{Rank 2:} \quad D3 \quad (\text{Sum} = 1.577) \\ &\text{Rank 3:} \quad D2 \quad (\text{Sum} = 1.0) \end{aligned}$$

Chapter 4

Evaluation of Summaries

We assess the similarity of model-generated summaries to gold-standard annotated summaries using some evaluation criteria. In the research community, the accepted evaluation metric for summarising jobs is with the help of ROUGE score.

4.1 ROUGE

Recall-Oriented Understudy for Gisting Evaluation is referred to as ROUGE [11]. Although ROUGE is 'recall-oriented' as its name implies, in practise it evaluates both Recall and Precision between candidate (model-generated or forecast) and reference (golden-annotated or target) summaries.

Recall refers to the proportion of words present in the reference summary that are also present in the candidate summary. Precision, on the other hand, denotes the percentage of terms in the candidate summary that genuinely match those in the reference summary.

The above mentioned ROUGE scores are further subdivided into ROUGE-1, ROUGE-2, and ROUGE-L scores.

1. **ROUGE-1**: The similarity of uni-grams between reference and candidate summaries is compared using the ROUGE-1 Precision and Recall test. Simply put, we mean by unigrams that each unit of comparison is a single word.

2. **ROUGE-2**: In ROUGE-2 Precision and Recall, bi-gram similarity between reference

and candidate summaries is compared. Bi-grams are two words from the reference and candidate summaries that are used as comparison tokens.

3. **ROUGE-L**: The LCS (Longest Common Subsequence) terms between the predicted and actual summaries are measured by ROUGE-L Precision and Recall. When we use the term LCS, we mean word tokens that are sequential but not necessarily one after the other.

Chapter 5

Implementation

5.1 Implementation for abstractive summarization

First, we will implement our model for abstractive summarization. The following steps are involved for this:

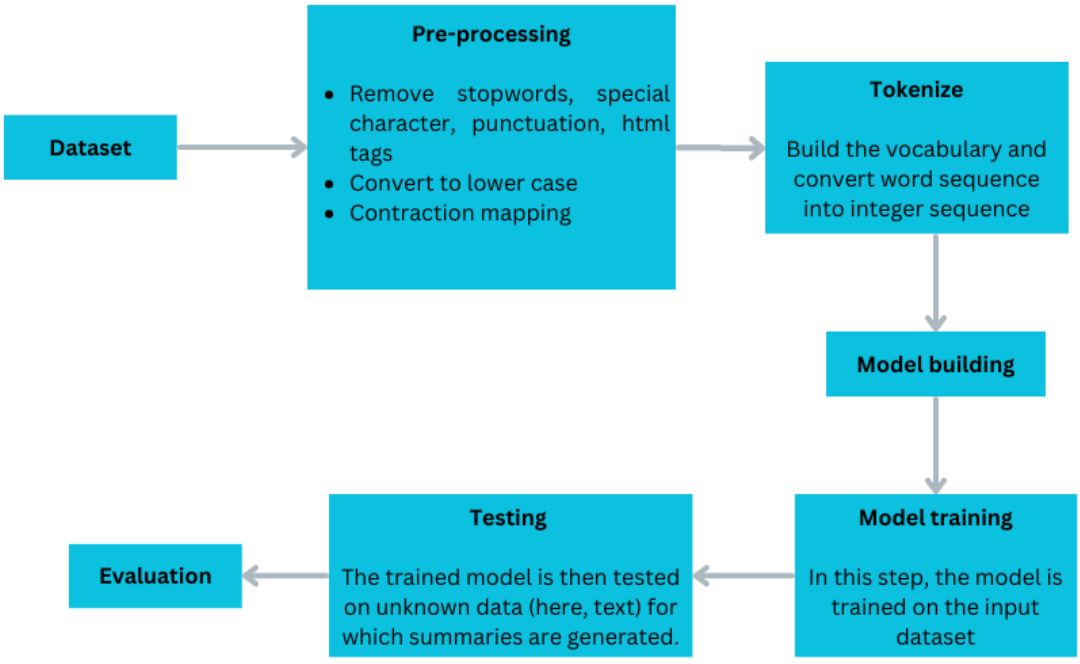


Figure 5.1: Workflow for generating abstractive summary

The dataset of customer online reviews used here has the following fields and entries :

```
data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 46250 entries, 0 to 49999
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      46250 non-null  int64
 1   ProductId               46250 non-null  object
 2   UserId                  46250 non-null  object
 3   ProfileName             46250 non-null  object
 4   HelpfulnessNumerator    46250 non-null  int64
 5   HelpfulnessDenominator  46250 non-null  int64
 6   Score                   46250 non-null  int64
 7   Time                    46250 non-null  int64
 8   Summary                 46250 non-null  object
 9   Text                    46250 non-null  object
dtypes: int64(5), object(5)
memory usage: 3.9+ MB
```

Figure 5.2: Reviews dataset

After preprocessing, the entries in '**Summary**' looks like below:

```
[17] data['Summary'][:10]

0                           Good Quality Dog Food
1                               Not as Advertised
2                           "Delight" says it all
3                                  Cough Medicine
4                                     Great taffy
5                                      Nice Taffy
6    Great!  Just as good as the expensive brands!
7                          Wonderful, tasty taffy
8                                      Yay Barley
9                                Healthy Dog Food
Name: Summary, dtype: object
```

Figure 5.3: Summaries after preprocessing

The below Fig. 5.4 shows the word count in text and summary.

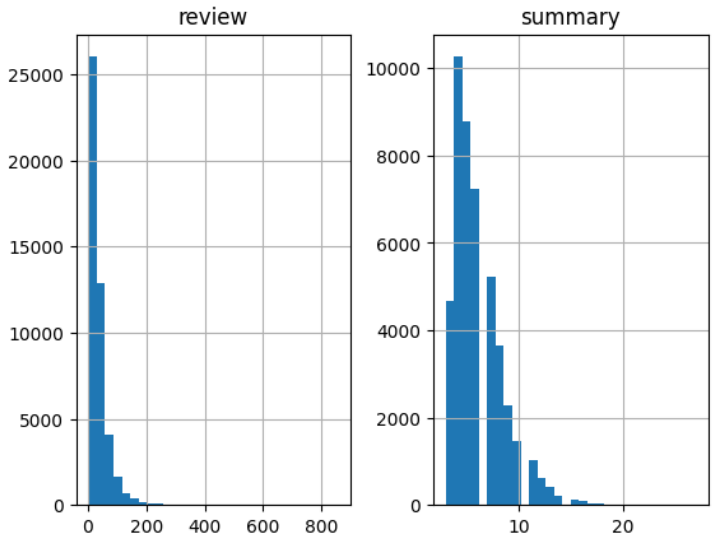


Figure 5.4: Word count in text and summary

Then, after building the model, we are training it on our dataset.

```
history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,1:] ,epochs=50,callbacks=[es],batch_size=32, validation_data=([x_val,y_val[:,:-1]], y_val.reshap
```

```
Epoch 1/50
1300/1300 [==============================] - 173s 132ms/step - loss: 1.9026 - val_loss: 1.7616
Epoch 2/50
1300/1300 [==============================] - 133s 102ms/step - loss: 1.8089 - val_loss: 1.6821
Epoch 3/50
1300/1300 [==============================] - 130s 100ms/step - loss: 1.7354 - val_loss: 1.6359
Epoch 4/50
1300/1300 [==============================] - 128s 99ms/step - loss: 1.6781 - val_loss: 1.6174
Epoch 5/50
1300/1300 [==============================] - 128s 99ms/step - loss: 1.6247 - val_loss: 1.5790
Epoch 6/50
1300/1300 [==============================] - 128s 98ms/step - loss: 1.5791 - val_loss: 1.5536
Epoch 7/50
1300/1300 [==============================] - 128s 98ms/step - loss: 1.5366 - val_loss: 1.5448
Epoch 8/50
1300/1300 [==============================] - 127s 97ms/step - loss: 1.4976 - val_loss: 1.5390
Epoch 9/50
1300/1300 [==============================] - 128s 99ms/step - loss: 1.4602 - val_loss: 1.5277
Epoch 10/50
1300/1300 [==============================] - 127s 97ms/step - loss: 1.4237 - val_loss: 1.5229
Epoch 11/50
1300/1300 [==============================] - 128s 98ms/step - loss: 1.3883 - val_loss: 1.5281
Epoch 11: early stopping
```

Figure 5.5: Training the model

Some of the actual and generated summaries that we got while testing our model are shown below:

```
Review: never typed review goes peas great wasabi peas non compare taste yesterday latest bag arrived
Original summary: awesome
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
Predicted summary:  great product


Review: ordered solely reviews found muddy bitter however question freshness cups boxes cups puffed l
Original summary: not real winner
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
Predicted summary:  not the best
```

Figure 5.6: Predicted and actual summaries

5.2 Implementation for extractive summarization

The steps involved in generating extractive summary are shown in Fig. 5.7.

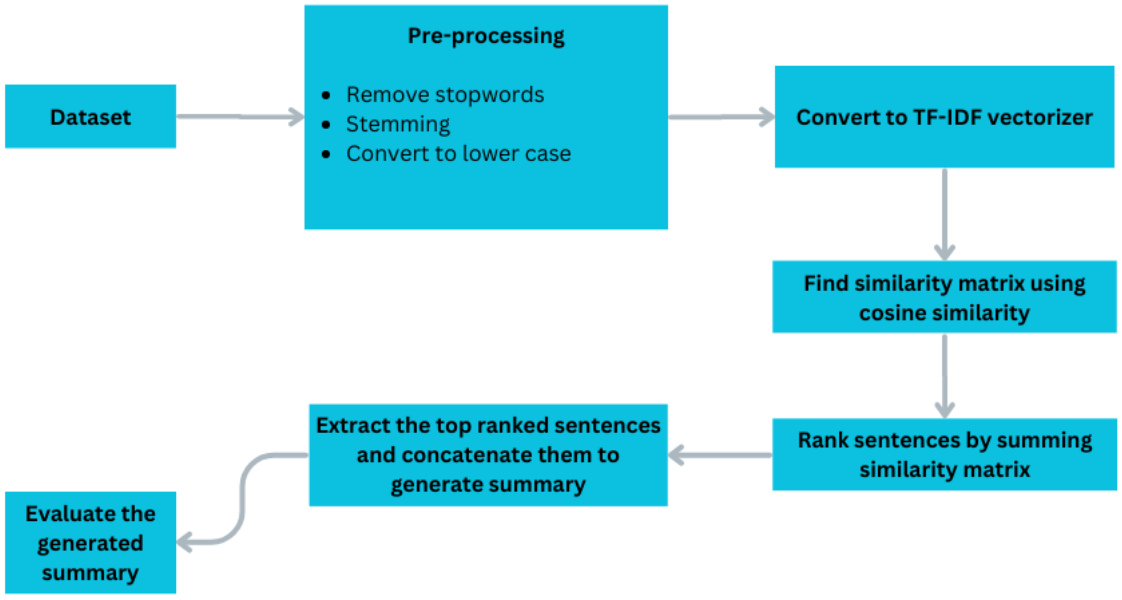


Figure 5.7: Workflow for generating extractive summary

For this, we have used judgment dataset which can be seen from Fig 5.8.

```
data
```

	Judgment_ID	Judgment	Summary
0	1	Appeal No. LXVI of 1949. Appeal from the High ...	The charge created in respect of municipal pro...
1	2	XXIX of 1950. Application under article 32 of ...	Section 7 (1) (c) of the East Punjab Public Sa...
2	3	XXXVII of 1950. Application under article 32 o...	Section 4, sub section (1) (c), of the East Pu...
3	4	No. XVI of 1950. Appli cation under article 32...	Held, by the Full Court (i) (overruling a prel...
4	5	Civil Appeal No. 8 of 1951. Appeal from the ju...	S and B were sons of two brothers respectively...
...	...	...	...
7025	7127	Appeals Nos. 722 and 723 of 1993. From the Jud...	On 25.2.91 the appellants except appellant No....
7026	7128	Appeal No. 1690 of 1993. From the Judgment and...	The Director General, Ordnance Factories (D.G....
7027	7129	Appeal Nos. 2919 20 of 1981. From the Judgment...	The appellant State issued a notification unde...
7028	7131	Appeal No. 228 (NT) of 1987. From the Judgment...	The respondent assessee challenged the best ju...
7029	7132	Appeal No. 2881 of 1993. From the Judgment are...	The appellant landlord filed an eviction suit ...

7030 rows × 3 columns

Figure 5.8: Dataset consisting of judgment-summary pairs

Now, let us test our model on a specific judgment. The actual and predicted summaries are as follows:

```
Actual summary:

Applying the principle enunciated in Atma Ram Mittal vs Ishwar Singh Punia; , , this Court dismissed
the special leave petition, and, HELD: 1.1 The exemption would apply for a period of ten years and
will continue to be available until suit is dis posed of or adjudicated. [121H] 1.2 If the petitioner
fails to file an undertaking on usual terms, the decree shall become executable forthwith. [122B]


Generated summary:

Learned counsel for the respondent Mr. S.C. Maheshwari states that the decree will not be executed
till 30th April, 1990 subject to an undertaking on usual terms being filed in this Court within four
weeks from today. ecial Leave Petition (C) No. The special leave petition is dismissed. Petition dis
missed. From the Judgment and Order dated 4.1.1988 of the Punjab and Haryana High Court in Regular
Second Appeal No. If the undertaking is not filed, the decree shall become executable forthwith.
```

Figure 5.9: Actual and predicted summaries

The ROUGE metrics for this actual and generated summary can be found from the Fig. 5.10.

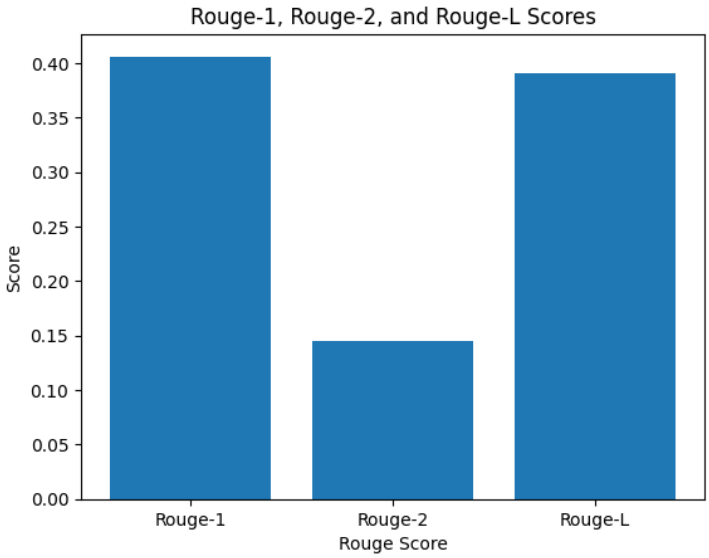


Figure 5.10: Rouge scores

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Automating the process of text summarization has been an active area of research for a considerable period of time. While extraction-based summarization has received significant attention, producing abstractive summaries from text sources is still a relatively new and less explored field. Legal texts, in particular, present unique challenges due to their lengthy and distinct nature compared to other genres. Even though, the demand for summarization of legal text automatically has long been recognized, but it has only recently garnered attention from computer scientists.

In this project, our aim was to provide an overview of various text summarization methods, with a particular emphasis on summarization of legal text. First, we began by providing a general definition of text summarization and a quick overview of current studies in the field. Thereafter we discussed about the model that we used during the course of our project. First we discussed the model for abstractive summarisation using deep learning and discussed about the extractive summarisation using TF-IDF and cosine similarity.

We also discussed about the dataset that we used. We collected some past years judgments of Supreme Court and their summaries and we used it as our dataset. Along with this, we also used an online customer review dataset. Then, we trained our model and tested it and predicted the outputs for both the cases.

6.2 Future Work

One of the future aims for this is to extend the work in the machine learning methodologies and apply the topic-focused summarization framework. The summaries that are topic-focused are much more accurate and useful to users. So, working on subject modelling and summarization in the social media space would be more intriguing in the future.

Its functionality can be increased by making it a multilingual summarizing system. If online lexical databases for other languages are available, this project would be helpful for working in that direction. By utilising very basic extensions, the work given in the thesis can also be applied to multiple document summarization.

Precision, Recall, and F-measure evaluation measures have been employed in the project to quantify the performance improvement over current systems. Future research can look into additional metrics that can be employed in an environment of automatic review to gauge the overall quality, such as grammar, readability, prominence, and relativeness. Despite the fact that ROUGE metrics have limitations and can only be utilised when the projected and real summaries are nearly identical since it assesses by word matching, new metrics can be developed that assess the actual and predicted summaries based on the meaning that they are specifying.

Bibliography

[1] Divakar Yadav, Jalpa J Desai, and Arun Kumar Yadav. Automatic text summarization methods: A comprehensive review. *ArXiv*, abs/2204.01849, 2022.

[2] Dima Suleiman and Arafat Awajan. Deep learning based abstractive text summarization: Approaches, datasets, evaluation measures, and challenges. *Mathematical Problems in Engineering*, 2020, 08 2020.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[4] Varun Pandya. Automatic Text Summarization of Legal Cases: A Hybrid Approach. *arXiv e-prints*, page arXiv:1908.09119, August 2019.

[5] Juan-Manuel Torres-Moreno. *Automatic Text Summarization*. 10 2014.

[6] Caroline Uyttendaele, Marie-Francine Moens, and Jos Dumortier. Salomon: Automatic abstracting of legal cases for effective access to court decisions. *Artif. Intell. Law*, 6:59–79, 03 1998.

[7] Atefeh Farzindar and Guy Lapalme. Letsum, an automatic legal text summarizing system. *Jurix*, pages 11–18, 01 2004.

[8] Ilya Sutskever, Oriol Vinyals, and Quoc Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4, 09 2014.

[9] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[10] Mohammad Alodadi and Vandana P. Janeja. Similarity in patient support forums using tf-idf and cosine similarity metrics. In *2015 International Conference on Healthcare Informatics*, pages 521–522, 2015.

[11] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.