NLP PROJECT

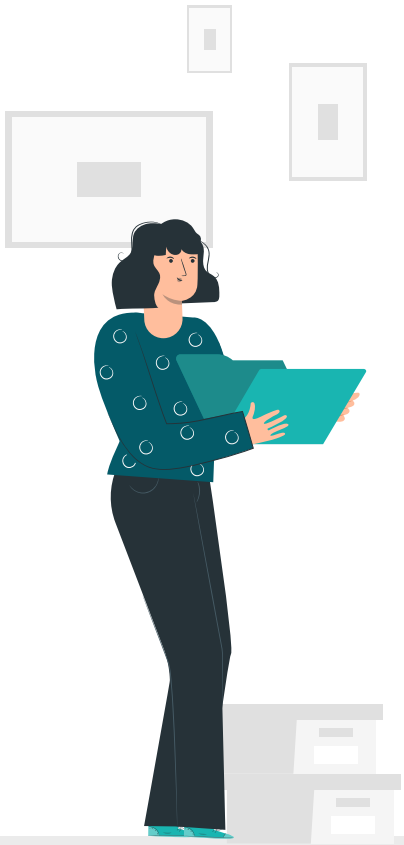
TEXT SUMMARISER

IIT2018125 (Raktim Bijoypuri)

WHAT IS TEXT SUMMARISER ???

Text summarisation is an effective process of converting a large monotonous document into a short yet coherent version of it. Automatic text summarization methods are required to address the ever-growing amount of text data available online to both better help discover relevant information and to consume relevant information faster.

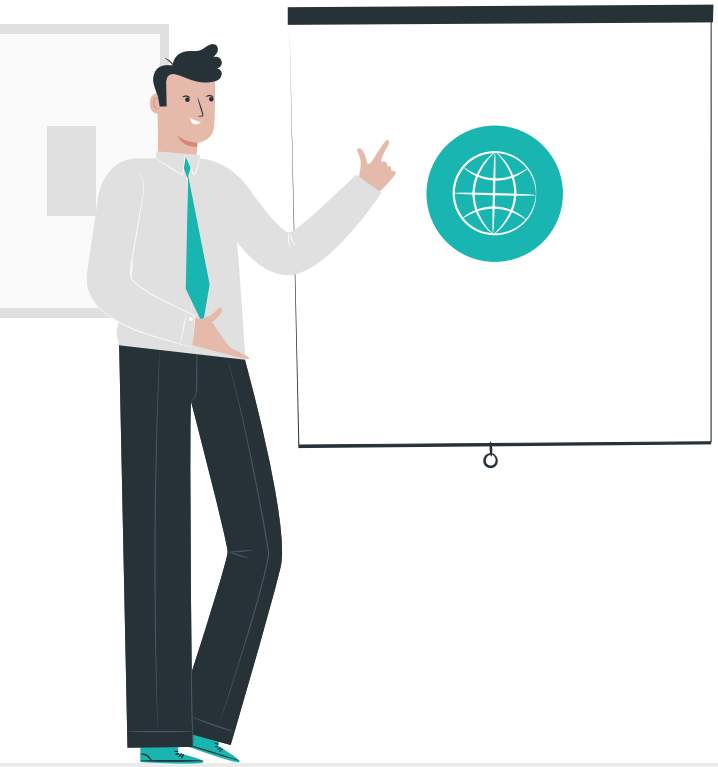
WHY WE NEED TEXT SUMMARISER ??



- Summarised text is easier to read and consumes less time.
- Whilst researching documents, summarization makes the selection process less cumbersome.
- Automatic summarisers are less biased than human summarisers.
- Automatic summarisation can be used to perform personal summarisation, and thus can be used as an effective chatbot system.

TWO MAIN APPROACH FOR SUMMARISATION

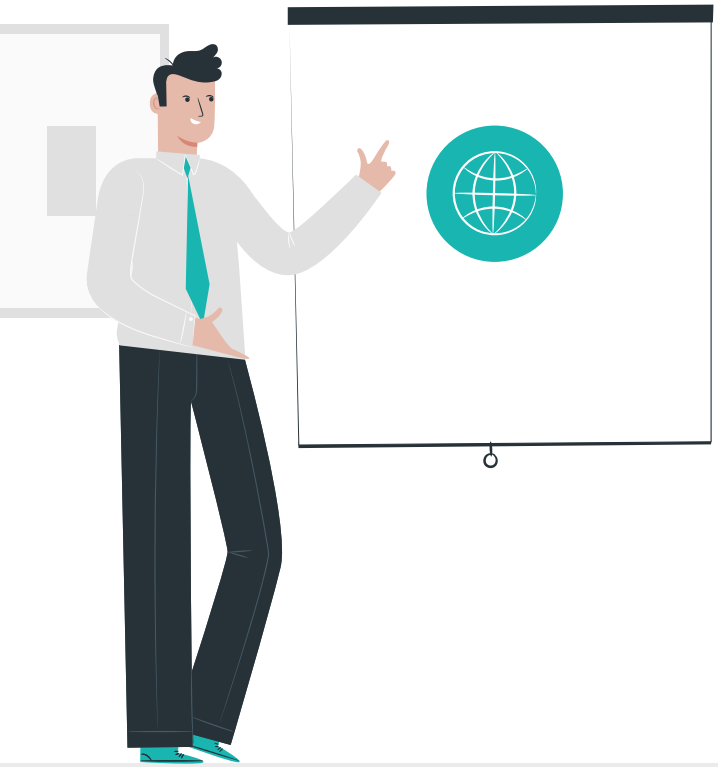
1. EXTRACTIVE SUMMARISATION
2. ABSTRACTIVE SUMMARISATION



EXTRACTIVE SUMMARISATION

It does what the name suggests, it extracts the most relevant sentences and phrases and combines them to produce a summary. Extractive strategies select the top *N* sentences that best represent the key points of the article

ABSTRACTIVE SUMMARISATION



This technique involves generating entirely new sentences, words, to capture an appropriate summary of the document.

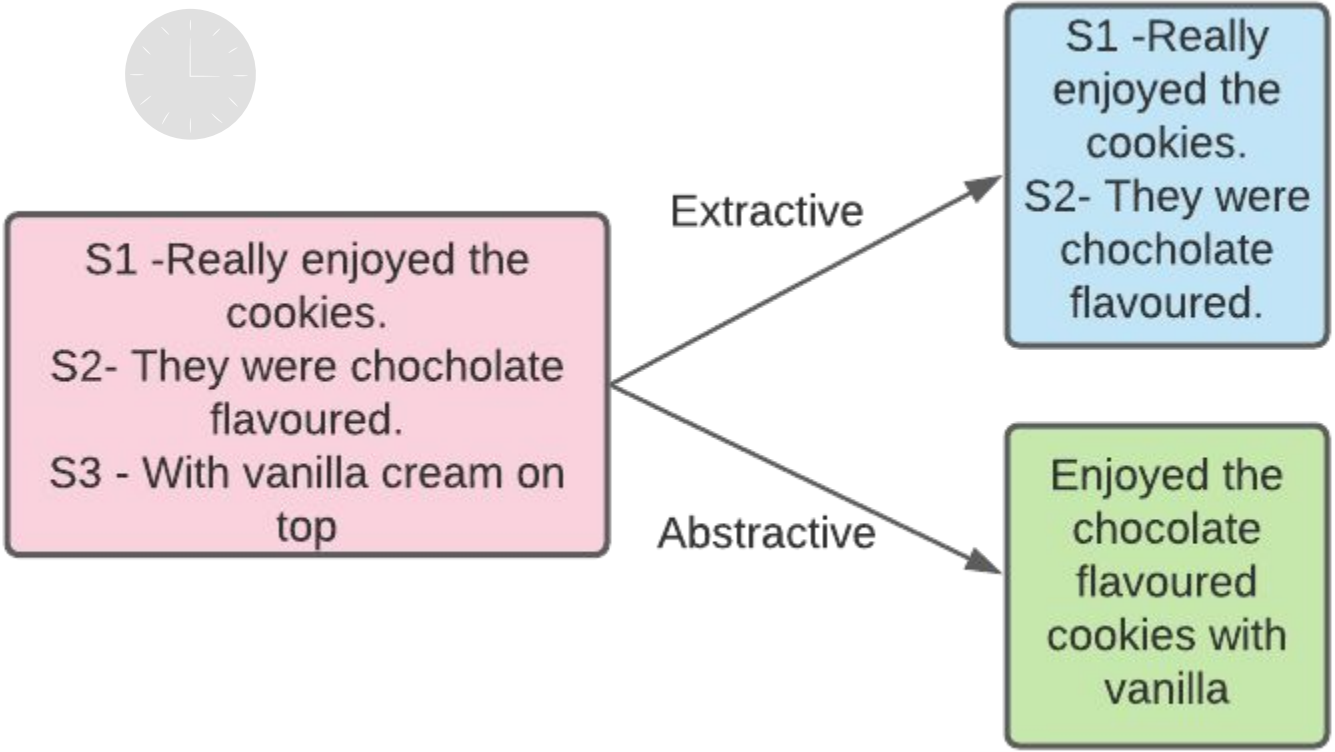


Fig 1. Text Summariser variants [1]

EXTRACTIVE TEXT SUMMARISER

METHODS FOR EXTRACTIVE SUMMARISER

We used Bert Base Model coupled with K-Means clustering algorithm, to perform extractive summariser.

REASON TO USE BERT MODEL

We have used the bert model to create a sentence embedder, this is done because it has increased accuracy in most algorithms.

- The bert model has been trained on a vast dataset
- It overcome the limitations of RNN and other neural networks
- It uses a powerful flat architecture with inter sentence transform layers so as to get the best results in summarization.
- Faster than RNN.

WORKFLOW

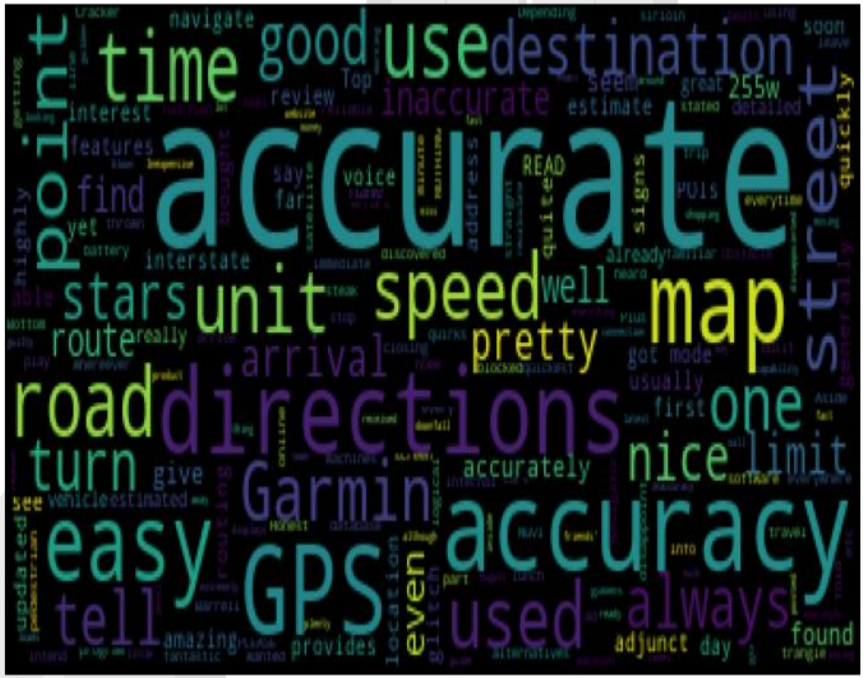
Fig. 2 Workflow of BERT [1]

STEPS INVOLVED

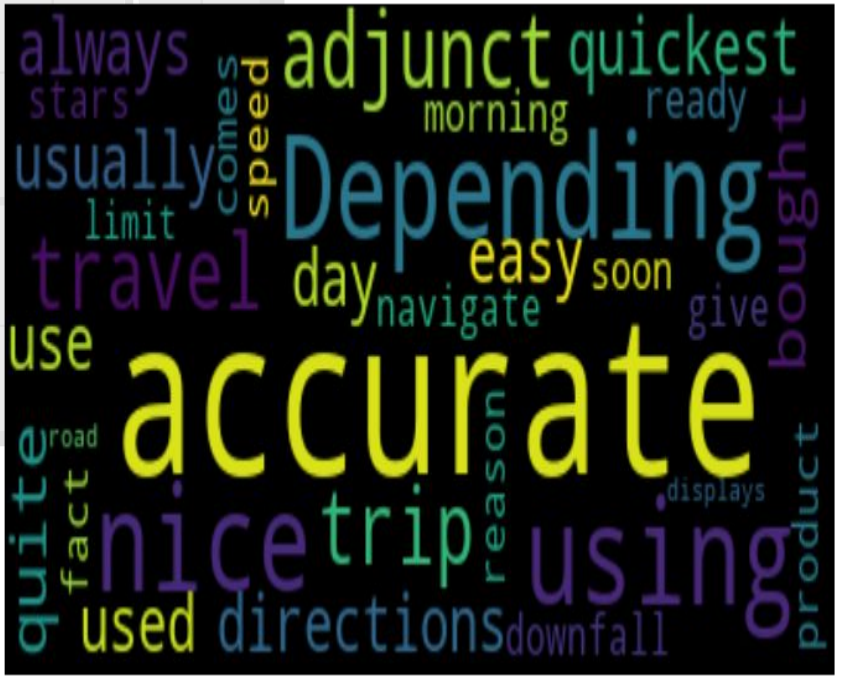
- We have to tokenize paragraphs into sentences first .
- After that modify sentences as per BERT input format
- After converting in BERT input form, tokenize each sentence into words.
- Now, we have to find embedded words for each sentence , for this we have loaded the BERT base model and evaluated it.
- Before finding embedded word for each sentence, we have to add starting and ending token before each sentence
- The BERT model produces 12 layers of latent vector. Here it depends upon the use case that whether you can use the last output layer or take average of all layers.
- After that pooling strategy is applied to obtain sentence embedding from word embeddings , here you can use mean or max of all word embedding.
- So we have obtained a sentence vector for each sentence.
- After that, for grouping similar sentences we can use the K-Means algorithm to cluster similar sentences together.

USING K-MEANS FOR CLUSTERING SENTENCE

After that bert sentence embedder method is used to to find sentence embedding which is then passed to KMSI which uses Kmeans clustering to cluster the sentences which are closest to each other (using euclidean distance) and returns cluster of sentences.

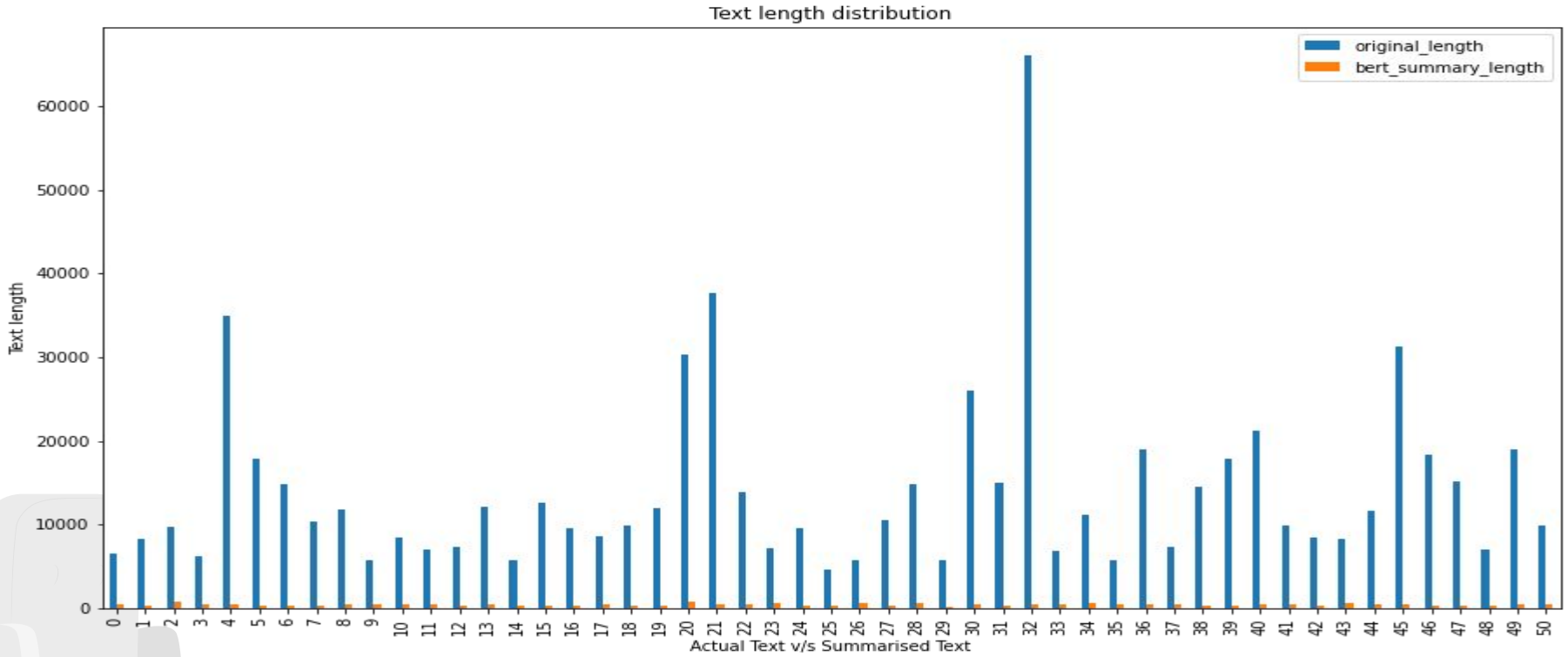


WORDCLOUD OF INPUT



WORDCLOUD OF PREDICTED

Fig. 3 Wordcloud



ORIGINAL TEXT LENGTH
V/S
PREDICTED LENGTH

Fig 4. Comparison of length

ABSTRACTIVE TEXT SUMMARISER

ENCODER DECODER ARCHITECTURE

As our problem is essentially a sequence to sequence problem because the input(text) and output sequences(summary) are of different length and we need the context of the whole input sequence in order to generate the output. For these seq2seq problems we use the encoder decoder architecture.

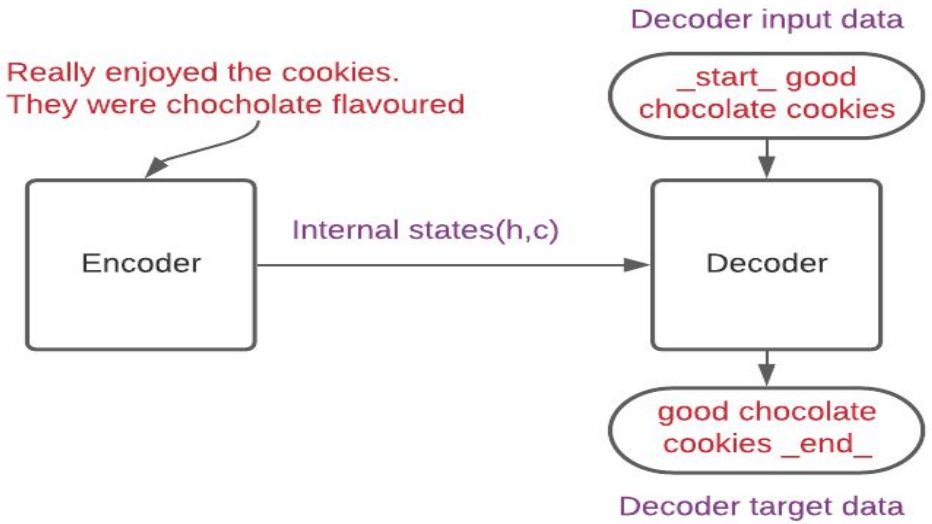


Fig 5. Encoder Decoder Example [2]

ENCODER

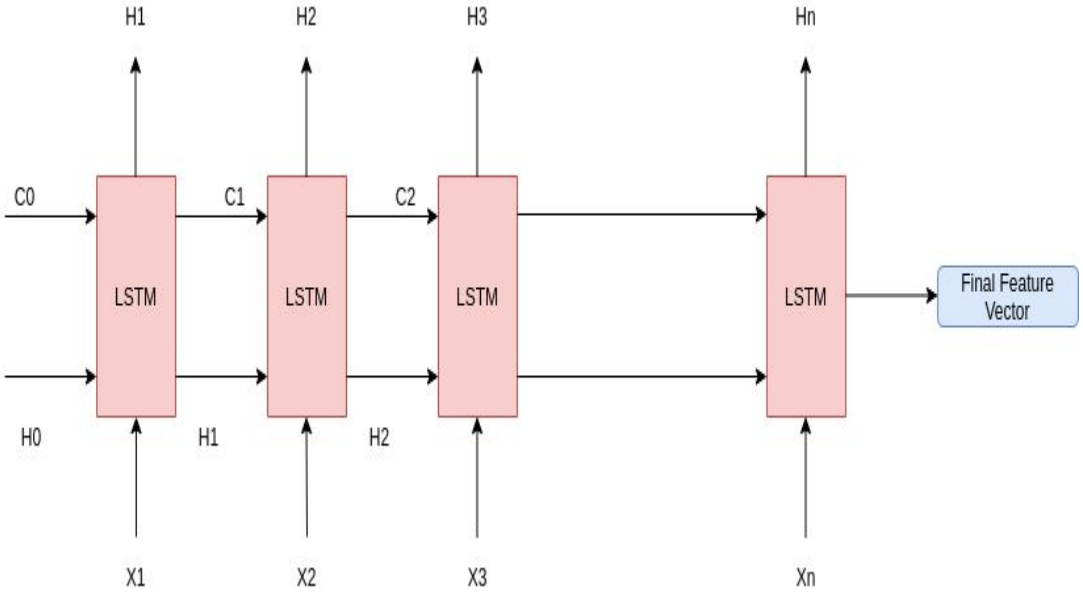


Fig 6. Encoder [2]

It is either a simple RNN or LSTM cell or a sequence of stacked LSTM cells. We feed the input sequence to the encoder and obtain the final hidden states of the encoder(this serves as our context).

DECODER

We initialise decoder internal states with the encoder final states.

- After this we feed the start token(*_start_*) into the decoder to start the decoding process.
- We then select the word with maximum probability as our predicted word.
- In the next time step we will now feed this predicted word to get our second word, we continue this process until we hit the end token(*_end_*) or reach the maximum specified length.

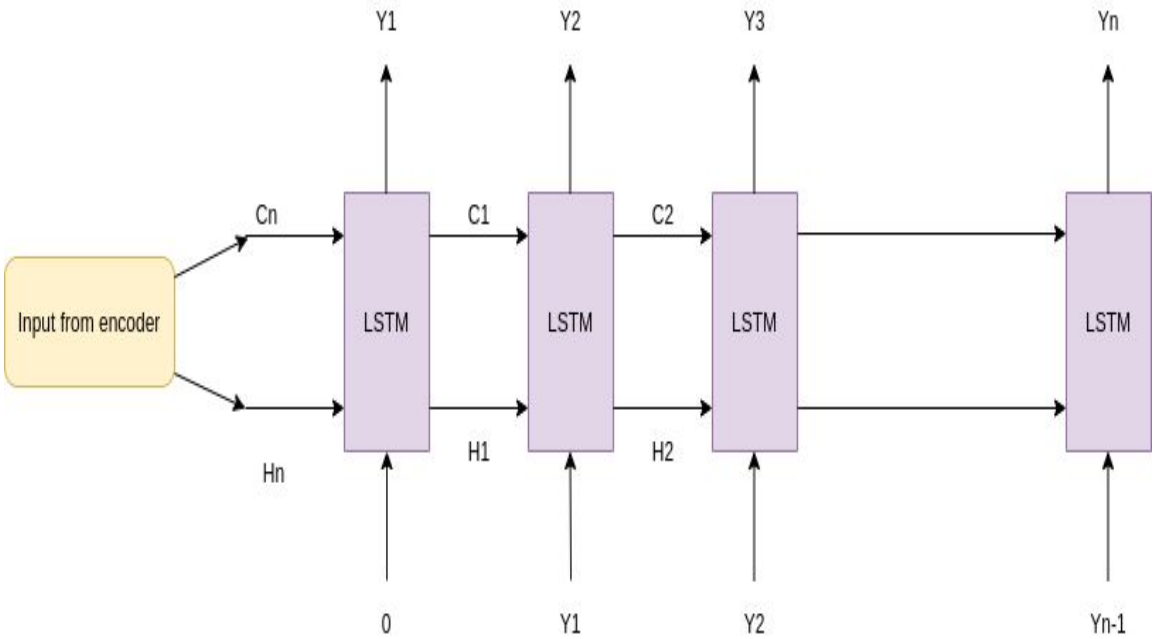


Fig 7. Decoder [2]

ATTENTION LAYER

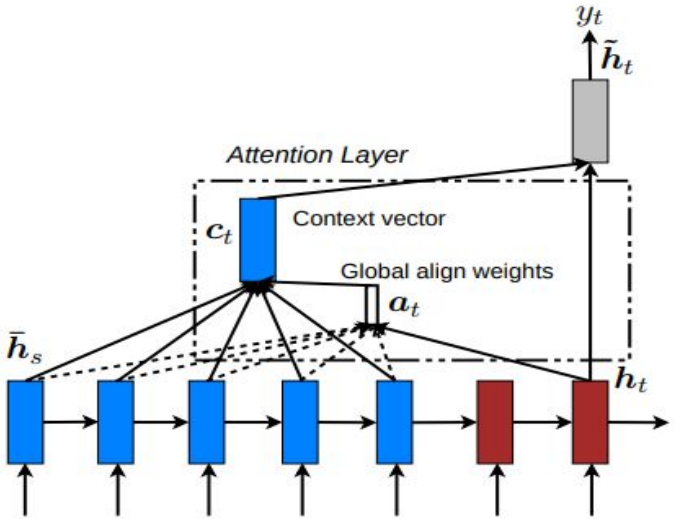


Fig 8. Attention Layer [3]

The whole input sequence when passed through the encoder gives us a vector of fixed length(i.e hidden states h,c) this is a problem of long texts as it leads to loss of information. So to overcome this we have also used an attention layer to increase accuracy of our model.

This layer focuses on essential features and ignores less relevant features. Another problem of the encoder-decoder model was that if the encoder gives a wrong summary, the decoder will just translate it, however, with the attention layer, we can ignore non-relevant information and focus on the more relevant ones.

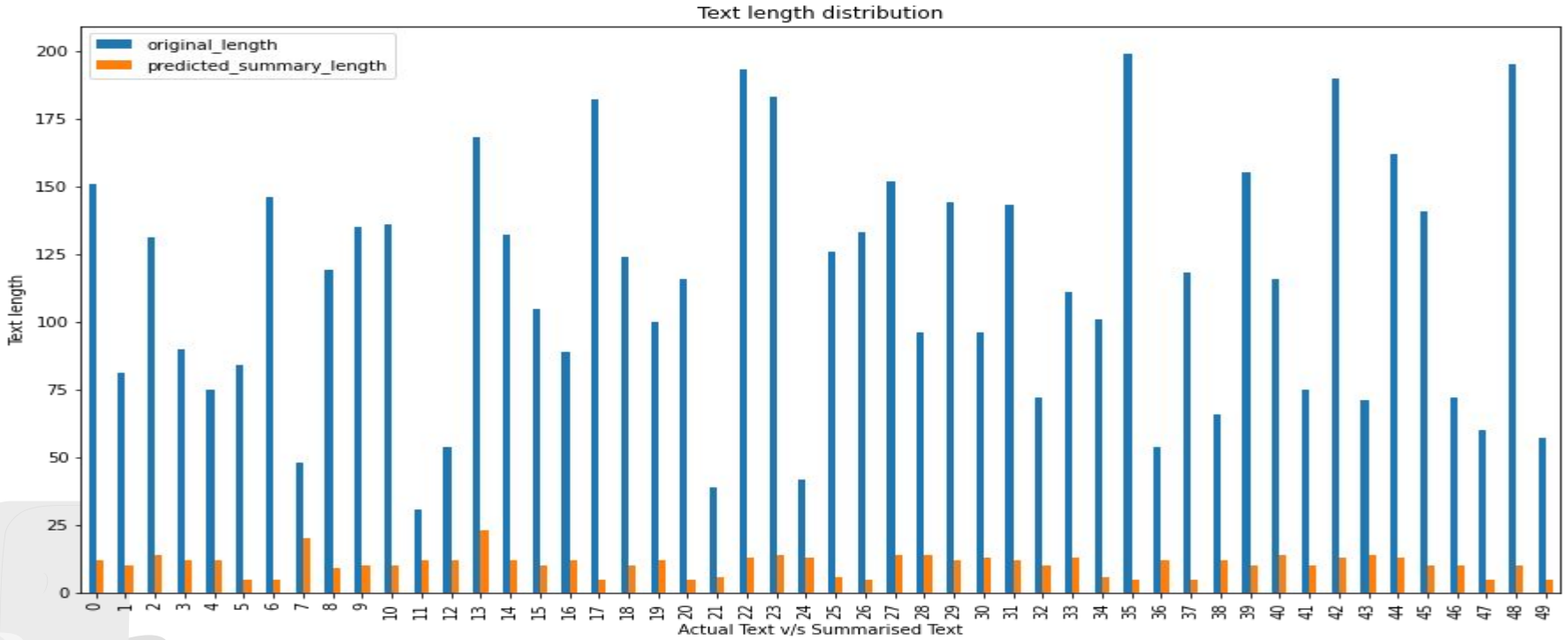


Fig 9. Comparison of length

ORIGINAL TEXT LENGTH
V/S
PREDICTED LENGTH

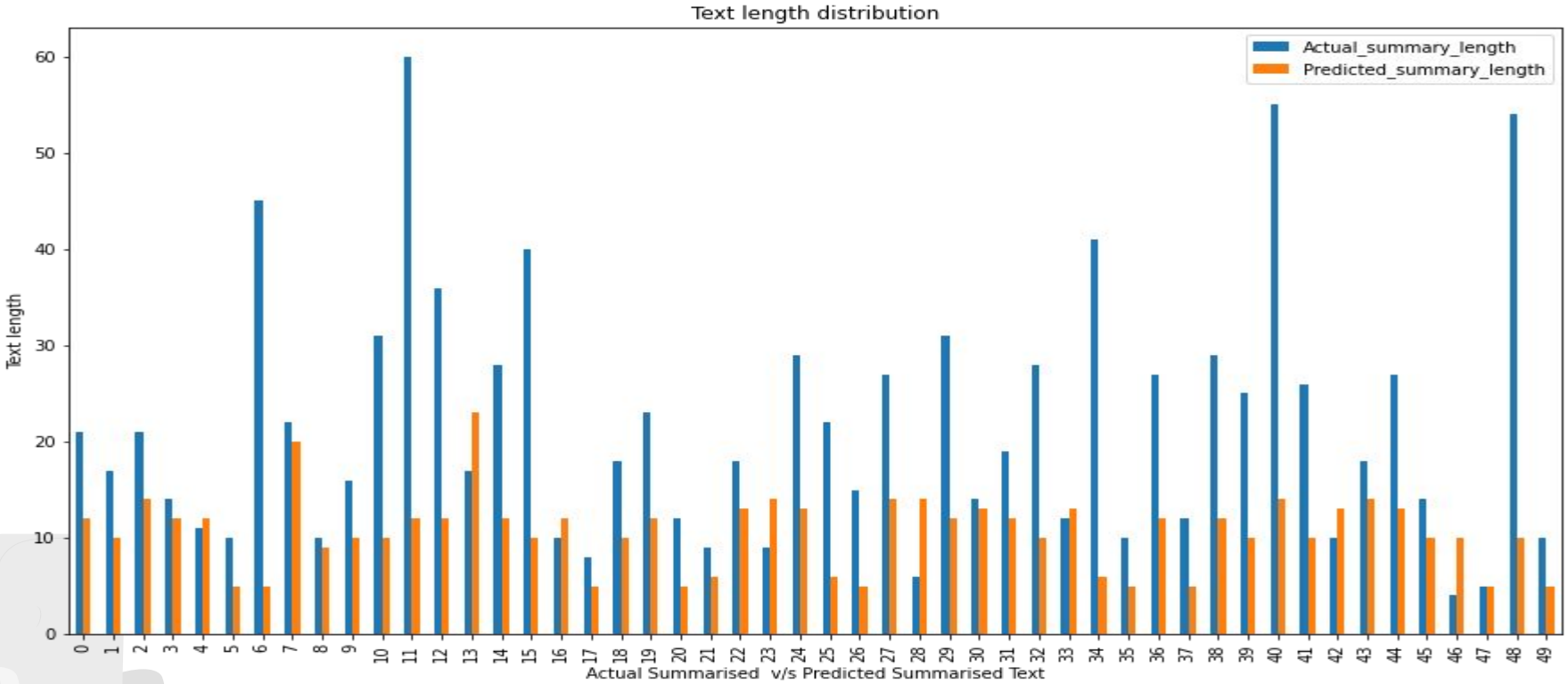
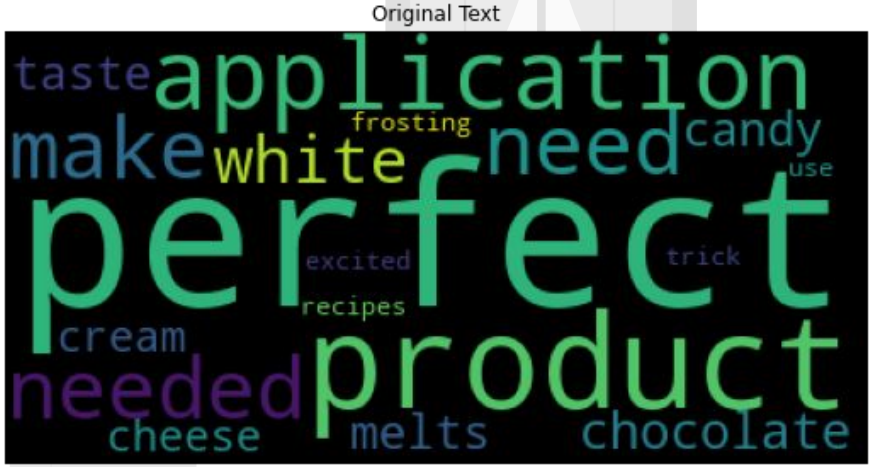


Fig 10. Comparison of length

ORIGINAL SUMMARY LENGTH
V/S
PREDICTED SUMMARY LENGTH



WORDCLOUD OF INPUT

WORDCLOUD OF PREDICTED

Fig 11. Wordcloud

REFERENCES

1. https://iq.opengenus.org/bert-for-text-summarization/

2. Leveraging BERT for Extractive Text Summarization on Lectures Derek Miller, Georgia Institute of Technology Atlanta, Georgia

3. Attention Is All You Need by Ashish Vaswani , Noam Shazeer , Niki Parmar , Jakob Uszkoreit , Llion Jones, Aidan N. Gomez ,Łukasz Kaiser

4.https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

5.https://blog.paperspace.com/implement-seq2seq-for-text-summarization-keras/

END