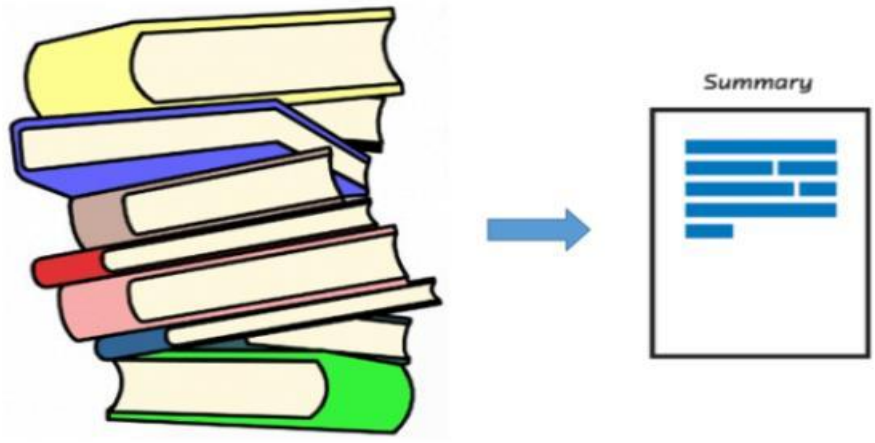
Abstractive summarization



Mandate-II

- ***Problem Statement:***

Given a set of tweets pertaining to a trending topic, create an abstractive prose summary of the tweets. Do not just string the tweets together to form the summary. The summary will need to paraphrase and/or say more than what is directly said in the tweets. Propose a rubric to evaluate the accuracy of your summarization.

- ***The code for this mandate:*** ***https://github.com/rittik9/Abstractive-Summarization-Of-Tweets***

(For now this repository is private)

- ***Work Done:***

<I> Dataset Collection for Fine Tuning
<II> Data Cleaning & Pre Processing
<III> Exploring various summarization techniques (extractive and abstractive)
<IV> Summary Generation for our data points.

</> Dataset Collection for Fine Tuning: *I used 'snscrape' to scrape tweets on trending topics.I selected 10 trending topics and scraped 1000 tweets on each of the trending topics. After that, I grouped 10 tweets in a single data point of our data set. So a tweet data set of 1000 data points got created.But there were no summaries with respect to the data points.*

I also got a tweet data set of 977 data points with their respective summaries from the internet.

<II>Data Cleaning & Pre Processing: *After collecting the tweets the following data cleaning and pre processing steps were applied on them:*

1.Lower casing
2.Traslating the tweets of various languages to English
3.Expansion of words
4.Removing URL
5.Removing RT & Emojis
6.Removing HTML tags
7.Removing Special Characters and Punctuation

8.Removal of Accented Characters
9.Spelling Correction

1. Lower Casing: *At first, I converted each set of tweets in to lower case .*

2. ***Translating the tweets of various languages to English:*** *To achieve this task I used Google Translator's API which supports 107 different languages.*

3.***Expansion of words:*** *I made a dictionary of 80 key value pairs, where key is the contracted form of the words or phrase and value is their expanded form. Ex: "he'd've": "he would have", "hadn't've": "had not have".*

*4.***Removing URL:** *To remove url from the tweets I used python's 're' module which provides support for regular expressions.*
re.sub(r'(http|ftp|https)://([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:/~+#-]*[\w@?^=%&/~+#-])?', '', x)
I used the re.sub() function to replace all occurrences of URLs in the tweet with an empty string, effectively removing them from the tweet.
Ex. "Check out this cool website: https://www.abc.com "
After removal of URL: "Check out this cool website: "

5. ***Removing RT & Emojis:*** *To remove RT & emojis as well I used python's 're' module. I replaced both RT and emojis with an empty string.*

6. Removing HTML tags: *To remove HTML tags I used* ***'BeautifulSoup'*** *package.*

7. Removing Special Characters and Punctuation: *Then I replaced* ***"!$%&'()*+-/:;<=>[]^_`{|}~•@#\n&"*** *these special characters with an empty string.*

8. Removing Accented Characters: *Accented characters are* ***those characters that consist of accent marks****. Accent marks are diacritic marks placed above or below a letter in a word to represent a specific pronunciation. I replaced them with normal characters.*

9. Spelling Correction: *Finally I did spelling correction in each of our data points using* ***'textblob'*** *library.*

<////> Exploring various summarization technique (extractive and abstractive):

Extractive Summarization: *I explored various summarization techniques, extractive summarization is one of them.Extractive summarization is a technique used in natural language processing to create summary of a longer text document by extracting and condensing the most important information from the original text. I implemented extractive summarization using* ***' TextRank algorithm'.***

The TextRank algorithm is an unsupervised approach that uses ***graph-based methods*** *to determine The importance of words or sentences in a text****. It uses co-occurrence measures and PageRank algorithm to identify the most important sentences in a document.***
Here are the steps for implementing the TextRank algorithm for extractive summarization:
Preprocess the text:

- *At first, I* ***tokenized*** *the input text into sentences and words & remove* ***stop words*** *(****using NLTK****).*
- *Then I* ***lemmatized*** *the words into their base form using* ***spacy*** *.*
- *I also removed the most frequently occurring (****language builder terms****) 10 terms and rarely occurring 10 terms from our documents.*
- *Then I used* ***GloVe embeddings*** *(GloVe stands for Global Vectors for word representation. It is an unsupervised learning algorithm developed by researchers at Stanford University aiming to generate word embeddings by aggregating global word co-occurrence matrices from a given corpus.) for representing the words in our sentences. Then I* ***vectorized*** *the sentences by summing up the all the word embeddings and dividing it by the no. of words in the sentence.*

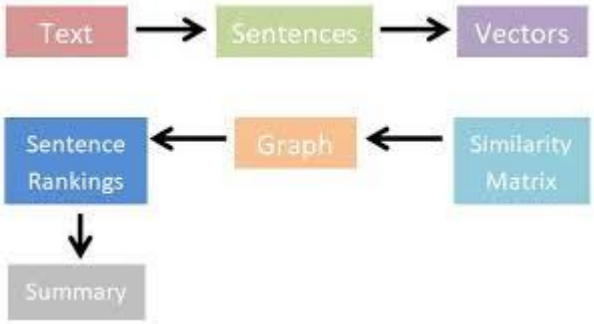
Creating a Similarity Matrix: *Then I created a similarity matrix of size (n*n) where n is the no of sentences in each data point. Each entry of this matrix I.e. sim_mat[i][j] represents cosine similarity of ith and jth sentence. I used sklearn's cosine similarity function for this.*

Creating a graph: *Then I created a graph where each node represents a sentence, and each edge represents the cosine similarity between sentences. I did this using* ***networkx*** *library.*

Applying PageRank: *Then I applied the PageRank algorithm of Google to the graph using* ***networkx*** *library to determine the importance of each sentence in the graph. The PageRank algorithm assigns a score to each sentence based on its connections to other sentences in the graph.*

Select the top-ranked sentences: *I selected the top-ranked sentences as the summary sentences. The number of sentences to select depends on the desired level of summarization.I took it as 3.*

Generate the summary: *Finally, generate the summary by concatenating the selected sentences.*



Overview Of TextRank Algorithm

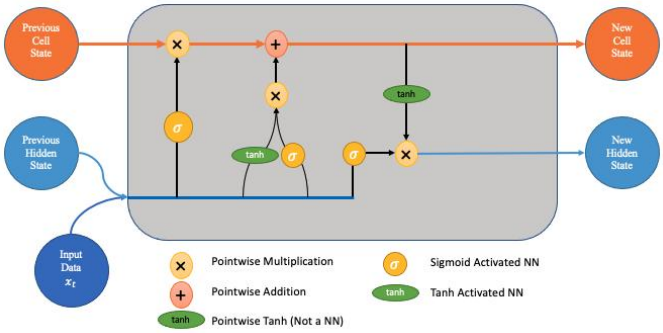
Abstractive Text Summarization*:An abstractive summarization model creates a summary that is not just a subset of the original sentences, but rather a new text that captures the essential meaning of the original content in fewer words.To capture the inner meaning of the we have to take care of various problems.One of them are*

Long Range Dependency Resolution *which refers to the difficulty in capturing the relationship between two words that are far apart in a sentence. The problem arises because the meaning of a word can be influenced by other words that are several positions away in the sentence.*

*Ex:****"The man who Bill met yesterday at the restaurant, and who was wearing a blue shirt, is a famous actor."***

In this sentence, the subject ***"man"*** *is linked to the object* ***"actor"*** *through a long distance of several words and clauses.*

The long range dependency problem is particularly challenging for traditional machine learning models, such as ***BOW,Tf-Idf****,* ***n-gram*** *language models like* ***CBOW W2V(Which tries to predict a word given neighbouring context words)*** *,****Skipgram W2V(which tries to predict the context words given a word)*** *or* ***feedforward neural networks,LSTM(in case of large documents)****, which are limited in their ability to capture long-term dependencies. This is because these models* ***typically process input sequentially and can only consider a fixed context window around each word****.Several other complex problems like* ***polysemy resolution, co-reference resolution****, etc. appear due to long range dependency (LRDs) in sentence structures.*



Overview Of a LSTM Cell

To address the long range dependency problem, researchers have developed more sophisticated ***self attention based Masked Language Models*** *which are called* ***transformers****, which are specifically designed to handle sequential data and can capture longer-term dependencies. Masked Language Model (MLM) is an generalization over Skipgrams, where inputs are provided as sentences with some elements missing (masked). A neural network is then trained to predict the missing term .Typically Typical architectures for MLM include attention layer & deep neural networks which can be* ***Unidirectiona****l (Process the input text in only one direction (typically left to right) and* ***Bidirectional*** *(Process input sentence from both directions to predict the masked word) .Unidirectional language models perform poorly on polysemy problem. Ex:*

"I stood by the bank for 2 hours to get to the ATM"

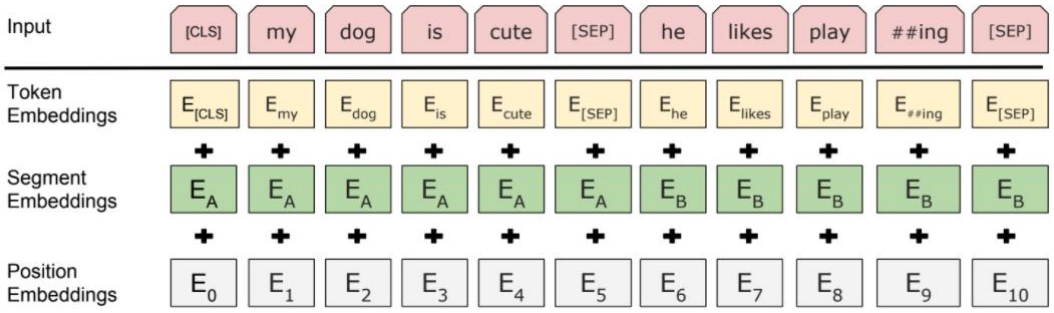
"I stood by the bank for 2 hours watching the river flow."

A unidirectional model like LSTM encodes the term 'bank' with the same context "I stood by the" for both sentences. A bidirectional model would encode the term "bank" differently for both sentences.

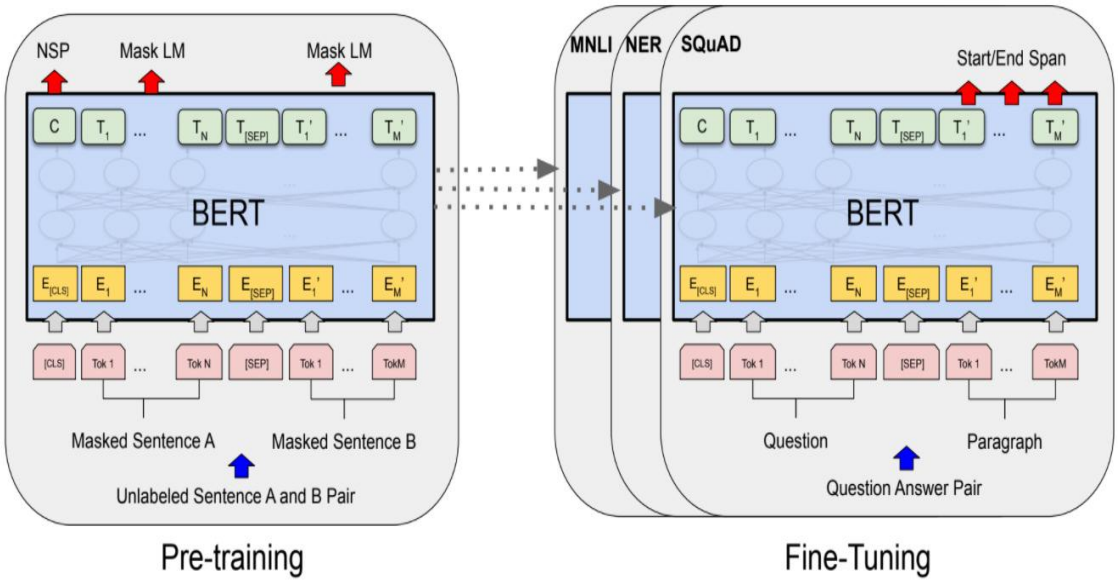
I explored various pretrained attention based seq2seq models from ***Huggingface's Transformer Library*** *for abstractive text summarization. Some of them are-*

- ***T5 (Text-to-Text Transfer Transformer****) - developed by Google AI.*
- ***BART (Bidirectional and Auto-Regressive Transformer)*** *- developed by Facebook AI Research (FAIR),*
- ***BERT (Bidirectional Encoder Representations from Transformers)-****developed by Google AI.BERT is a purely encoder-based model, meaning it only uses the encoder part of the transformer architecture.*

  The encoder is a stack of transformer layers that takes in a sequence of input tokens and generates a sequence of contextualized output representations for each token in the input sequence. BERT uses a multi-layer bidirectional transformer encoder, which means that the input tokens are processed in both forward and backward directions through the network, allowing the model to capture contextual information from both past and future tokens.The pre-training objectives used to train BERT are masked language modeling (MLM) and next sentence prediction (NSP). MLM involves randomly masking some of the input tokens and training the model to predict the original tokens from the masked ones. NSP involves predicting whether two input sentences are consecutive or not.
- ***Pegasus(Pre-training with Extracted Gap-sentences for Abstractive Summarization) -*** *developed by Google AI, Pegasus is a transformer-based model that was specifically designed for abstractive text summarization.Instead of the standard masked language modeling or next sentence prediction tasks, PEGASUS is trained using a gap-sentence generation task, where it is tasked with generating missing sentences in a document.*



We can add layers (like normal neural network layers, LSTM layers etc) on top of it to fine tune for specific tasks like - text summarization , sentiment analysis, question answering etc.

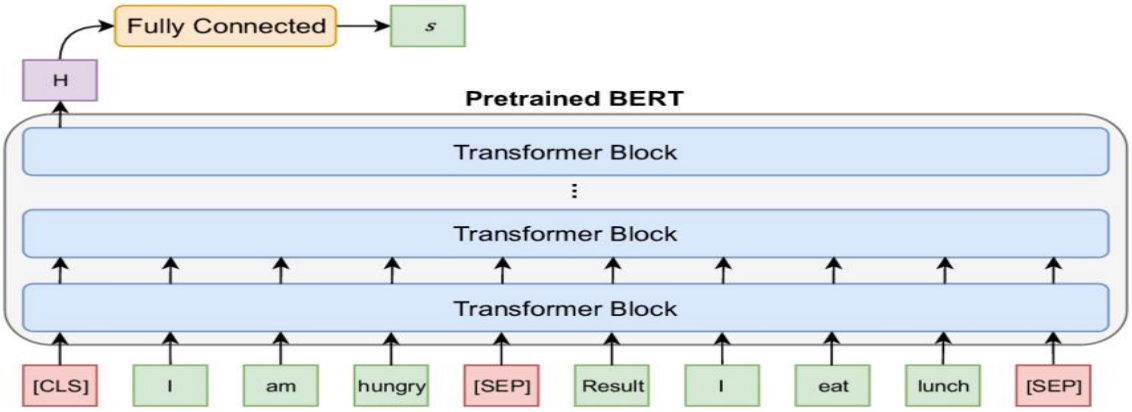


Summary Generation For Our Data Points: *After collecting the tweets and cleaning & pre processing them I took two approaches to create summaries for the data points in our data set.*

1. I used state-of-the art Pegasus model for summary generation of few data points.
2. I used OpenAI's API to automate Chatgpt for generating summaries of our data points .

In total I created a cleaned data set of 1960 data points for training and 110 data points for testing. Each data point contains a set of tweets and their respective summary.

- ***Future Work:*** *In mandate III and IV I will fine tune (at first I will freeze the parameters of pretrained model and add extra layers of simple feed forward neural network or LSTM or Transformer and train those layers) various pre trained models (as mentioned above) using my custom data set and evaluate their performances using various metrics like BLEU & ROGUE .*


Fully Connected
s
H
Pretrained BERT
Transformer Block
Transformer Block
Transformer Block
[CLS]
I
am
hungry
[SEP]
Result
I
eat
lunch
[SEP]