Abstractive Summarization

Mandate IV

Project Report

- **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.**

- **Work Done:**

<I> Dataset Collection for Fine Tuning
<II> Data Cleaning & Pre Processing
<III>Summary Generation for our data points using chatgpt and classical NLP.
<IV>Finetuning various pretrained large language models using our custom dataset

- **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.**

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

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'** library.
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.

● 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 OpenAI's API to automate Chatgpt for generating summaries of our data points .
2. I also tried to explore various classical nlp approaches like - Text Ranking Using Google Page Rank algorithm, POS Tagging using Hidden Markov Models, Data generation using Markov chains to generate summaries for our datapoints.

Approach:
I. At first given a preprocessed and clean datapoint (A group of sentences) I performed pos tagging on each of them using nltk's pos_tag() function which uses a rule-based approach, which is based on a pre-trained set of hand-crafted rules to assign Part-of-Speech (POS) tags to words in a given text.
II. Then I made a list of important pos tags as per my knowledge
important_pos_tags = ["NN", "NNS", "VB", "VBD", "VBG", "VBN", "VBP", "VBZ", "JJ", "JJR", "JJS"]
And stored those words of our datapoint into a list whose pos tag matches any of our important pos tags.
III. Then I made a markov chain model for next word generation (got the transition probabilities of important words using our original datapoints) which randomly chooses important words from the list and generates next word based on transition probabilities.
IV. Doing this I generated a group of new sentences from our original datapoint.
V. Finally I used google page rank algorithm to rank the newly generated sentences and took the top 50% sentences of my datapoint's length as a summary of my datapoint.
*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** e**mbeddings** (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.

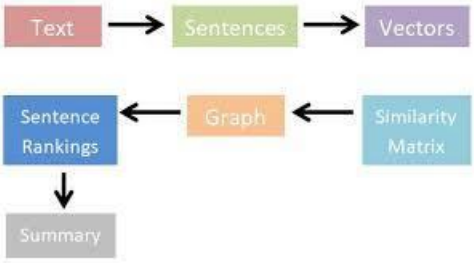
Creating a Similarity Matrix: Then I created a similarity matrix of size (n*n) where n is the no of sentences in each data point. Each entry of this matrix I.e. sim_mat[i][j] represents cosine similarity of ith and jth sentence. I used sklearn's cosine similarity function for this.

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 (top 50% sentences of my datapoint's length) sentences as the summary sentences.

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



Overview Of TextRank Algorithm

VI. I also tried to do POS tagging using spacy which uses Hidden Markov Models.

VII. I also tried to do summary generation using POS tagging .At first I did POS tagging on each of the sentences. Then counted how many important POS tags appeared on each of the sentences. Then ranked the sentences in a order that sentence with highest no. of important POS tags gets the top rank

and sentences with lowest no. of POS tags gets the low rank.Then picked up the top (50% of the total length of the original datapoint) ranked sentences as summary of the given datapoint.

- **Finetuning various pretrained large language models using our custom dataset:**

As I mentioned in the previous mandate I used various pretrained attention based seq2seq models from Huggingface's Transformer Library namely Pegasus,T5,BART for abstractive text summarization on our custom dataset.

- **Results On Unseen Test Data:**

Model	#Epochs	Rouge1	Rouge2	RougeL	RougeLsum	Bert Score		
						Precision	Recall	F1
Pretrained t5		0.2101	0.082602	0.1554	0.1555	0.83760	0.8339	0.8351
Finetuned t5	10	0.4195	0.2052	0.3173	0.3176	0.8887	0.8787	0.8833
Finetuned BART	3	.2609	.1600	.2323	.2476	0.9151	0.8415	0.8765
Pretrained Pegasus		.316229	0.154846	0.23563	0.234885	0.8389	0.8437	0.8409
Finetuned Pegaus	10	0.508948	0.28972	0.384435	0.383319	0.9415	0.9350	0.9381

- **Observations:**

1. I observed the summaries generated using classical NLP approaches, even though they are syntactically and grammatically correct they are not always semantically meaningful.

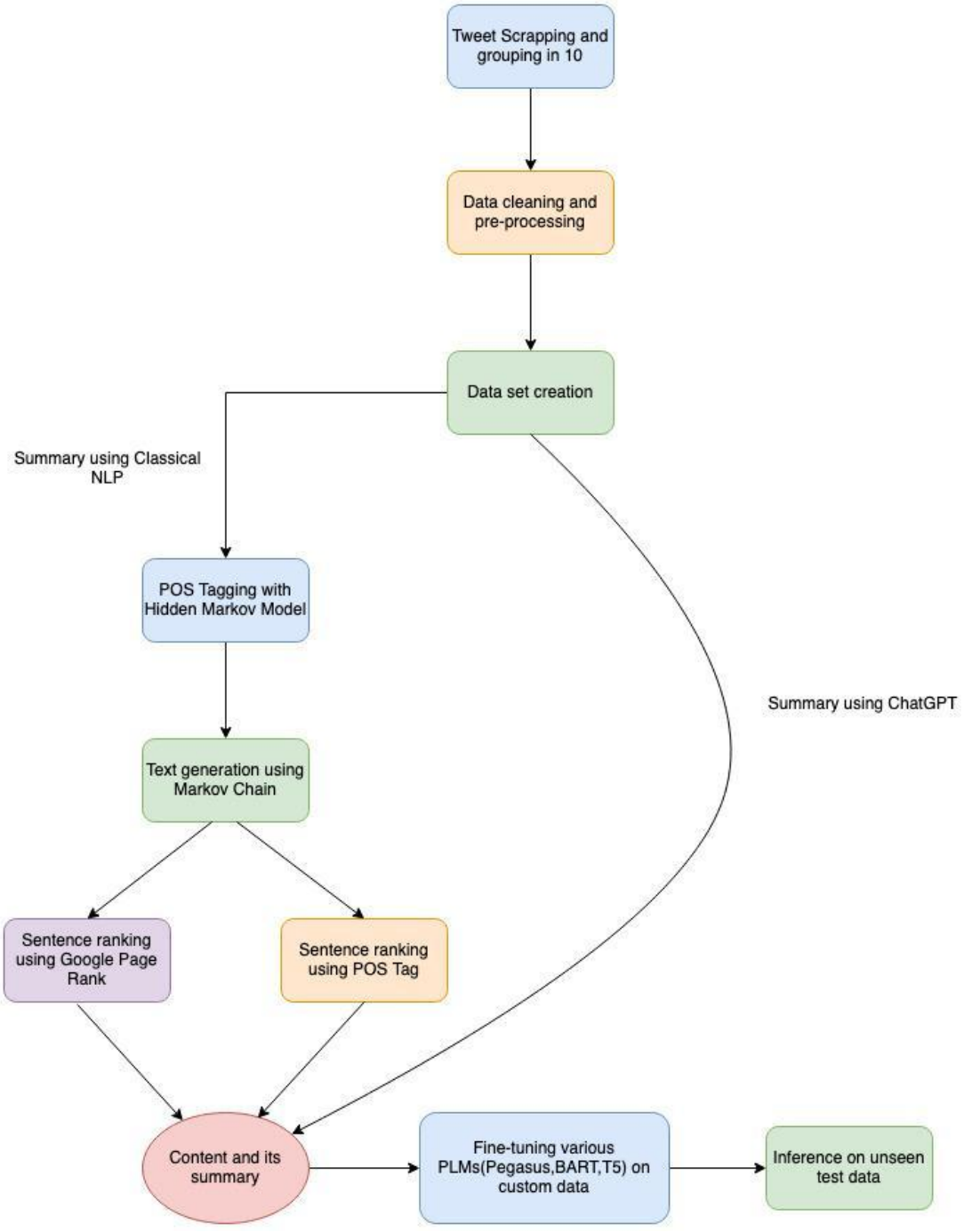
2.ROGUE measures the similarity between the generated summary and the reference summary in terms of overlapping n-grams (contiguous sequences of n words) or word sequences.It calculates the percentage of overlapping n-grams (typically, 1-gram, 2-gram, or 3-gram) between the generated summary and the reference summaries. Higher overlap indicates greater similarity.So when the no. of tokens increases in the reference and generated summaries the rogue score between them tends to decrease even though both the texts have semantically

similar meaning.In my datapoints summary has 300 tokens. That's why we tend to observe pretty low rogue score almost in all the models.
3.BERT scores sentences based on their semantic similarity to the summary generated so far, their saliency or importance in the source text, or other relevant criteria. These scoring criteria can be determined based on the specific requirements of the summarization task and the design of the summarization model.BERT scoring can be integrated into the overall architecture of the abstractive summarization model. BERT's relevance scores can be used to guide the selection or generation of sentences or phrases in the summary, helping to ensure that the generated summary is coherent, relevant, and representative of the source text which can not be done by rouge score. High Bert scores by our fine tuned models indicate good quality summary generation given a set of tweets.

<u>Comparison Between Rouge and Bert Score</u> :1.The book is written by me 2.I am the author of that book . We can see that both the sentences carry same meaning.But Rouge score indicates low similarity.

	Rogue1	**Rouge2**	**RogueL**	**RogueLsum**
scores	**0.3076**	**0.0**	**0.3076**	**0.3076**

While Bert Score indicates high similarity:Precision = 0.9171 ,Recall=0.9120,F1-Score=0.9146.



Full Project Overview

References:

1.https://towardsdatascience.com/basic-tweet-preprocessing-in-python-efd8360d529e
2.https://analyticsindiamag.com/guide-to-nlps-textrank-algorithm/
3.https://www.kaggle.com/code/orion99/markov-chain-nlp/notebook
4.https://www.youtube.com/watch?v=00GKzGyWFEs&list=PLo2EIpI_JMQvWfQndUesu0nPBAtZ9gP1o