

Project Report
On
Abstractive Summarization



Submitted
In partial fulfillment
For the award of the Degree of

PG-Diploma in Artificial Intelligence
(C-DAC, ACTS (Pune))

Guided By:

Dr. Krishnanjan Bhattacharjee

Submitted By:

Aakash Mahesha (210940128001)
Aakash Negi (210940128002)
Abhijit Das (210940128003)
Ganesh Eknathrao Panchal (210940128016)
Kesugade Aboli Rajendra (210940128024)

Center for Development of Advanced Computing

(C-DAC), ACTS (Pune- 411008)

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, **Dr. Krishnanjan Bhattacharjee**, C-DAC ACTS, Pune for her constant guidance and helpful suggestion for preparing this project **Abstractive Summarization**. We express our deep gratitude towards him for his inspiration, personal involvement, and constructive criticism that he provided us along with technical guidance during the course of this project.

We take this opportunity to thank the Head of the department **Mr. Gaur Sunder** for providing us with such a great infrastructure and environment for our overall development.

We express sincere thanks to **Ms. Namrata Ailawar**, Process Owner, for their kind cooperation and extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude toward **Dr. Priyanka Ranade** (Course Coordinator, PG-DAI) for their valuable guidance and constant support throughout this work and helps to pursue additional studies.

Also our warm thanks to **C-DAC ACTS Pune**, which provided us with this opportunity to carry out, this prestigious Project and enhance our learning in various technical fields.

Aakash Mahesha(210940128001)
Aakash Negi(210940128002)
Abhijit Das(210940128003)
Ganesh Panchal(210940128016)
Aboli Kesugade(210940128024)

ABSTRACT

Text Summarization is the task of extracting salient information from the original text document. In this process, the extracted information is generated as a condensed report and presented as a concise summary to the user. It is very difficult for humans to understand and interpret the content of the text. In this paper, an exhaustive survey on abstractive text summarization methods has been presented. The two broad abstractive summarization methods are the structured-based approach and the semantic-based approach. This paper collectively summarizes and deciphers the various methodologies, challenges, and issues of abstractive summarization. State of art benchmark datasets and their properties are being explored. This survey portrays that most abstractive summarization methods produce highly cohesive, coherent, less redundant summaries and are information-rich. Index Terms—Text Summarization, structure Based Approach, semantic Based Approach, Sentence Fusion, Abstraction Scheme, Sentence Revision, Abstractive Summary

Table of Contents

S. No	Title	Page No.
	Front Page	1
	Acknowledgement	2
	Abstract	3
	Table of Contents	4
1	Introduction	5-6
1.1	Introduction	5
1.2	Objective and Specifications	6
2	Methodology/ Techniques	7-14
2.1	Approach and Methodology/ Techniques	7
2.2	Literature Survey	9
2.3	Dataset	18
3	Implementation	20-35
3.1	Implementation	
4	Results	35-44
4.1	Results	35
5	Conclusion	45
5.1	Conclusion	45
6	References	46
6.1	References	46

Chapter 1

Introduction

1.1 Introduction

In recent times text summarization has gained its importance due to the data overflowing on the web. This information overload increases in great demand for more capable and dynamic text summarizers. It finds importance because of its variety of applications like summaries of newspaper articles, book, magazine, stories on the same topic, event, scientific paper, weather forecast, stock market, News, resume, books, music, plays, film, and speech. Due to its enormous growth, many top-notch universities like Aarhus University-Denmark, National Center for Text Mining (NaCTeM)-Manchester University, etc. have been staunchly working for its improvement. As the volume of information and published data on the World Wide Web is growing day by day, accessing and reading the required information in the shortest possible time is becoming constantly an open research issue. It is a tedious task to gather all the information and then give the output in a summarized form. The Internet is a platform that fetches information from databases. But still, this information is massive to handle. So text summarization came into a demand that condenses the document into shorter versions by preserving the meaning and the content. A summary is thus helpful as it saves time and retrieves massive documents data. Prior to this time, it was done by manual labor but nowadays automation has brought forth many advantages.

Text summarization approaches can be typically split into two groups: extractive summarization and abstractive summarization. Extractive summarization takes out the important sentences or phrases from the original documents and groups them to produce a text summary without any modification to the original text. Normally the sentences are in sequence as in the original text document. Nevertheless, abstractive summarization performs summarization by understanding the original text with the help of a linguistic method to understand and examine the text. The objective of abstractive summarization is to produce a generalized summary, which conveys information in a precise way that generally requires advanced language generation and compression techniques. Abstractive summarization is an efficient form of summarization compared to extractive summarization as it retrieves information from multiple documents to create a precise summary of information. This has

gained popularity due to the ability to develop new sentences to tell the important information from text documents. An abstractive summarizer displays the summarized information in a coherent form that is easily readable and grammatically correct. Readability or linguistic quality is an important catalyst for improving the quality of a summary.

1.2 Objective and specification:

The project aims to deliver a software program that exploits the latest Deep Learning libraries and functions. The program works towards the objective of text summarization.

Using the model thus created we show a short text summary of large text with a coherent and fluent summary having only the main points.

The different processes which can be used in this project are

- Abstractive Text Summarization
- Extractive Text Summarization

Chapter 2

Methodology and Techniques

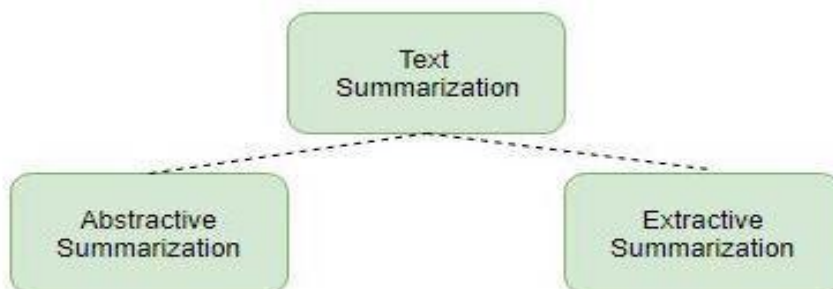
2.1 Approach & Methodology/Techniques:

Natural language processing (NLP) is a branch of artificial intelligence (AI) that assists computers in understanding and interpreting natural language. According to experts, natural language processing (NLP) may be used to organize and arrange knowledge in order to fulfill tasks such as translation and summarization.

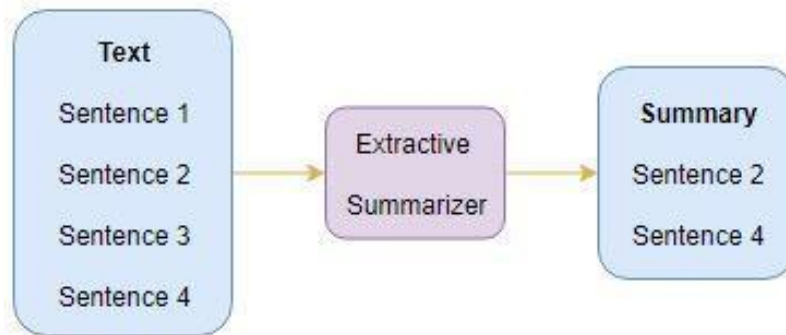
- The first step is to obtain the text data or conversational data from different sources.
- Import required packages for text summarization like Bart_sum ,LSTM, Pandas,NumPy,transformers, TF-IDF,spacy,summarizer.
- Apply different NLP algorithms to the gathered text data to get meaningful important summaries.

There are two fundamental approaches to text summarization:

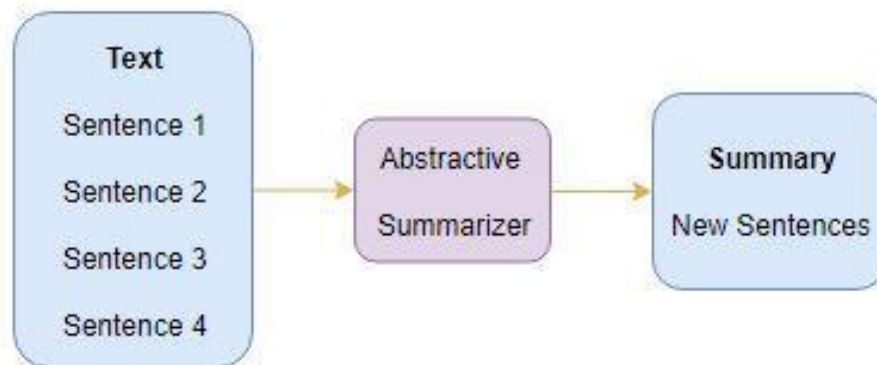
extractive and **abstractive**. The former extracts words and word phrases from the original text to create a summary. The latter learns an internal language representation to generate more human-like summaries, paraphrasing the intent of the original text.



1. **Extractive text summarization:** The name gives away what this approach does. We identify the important sentences or phrases from the original text and extract only those from the text. Those extracted sentences would be our summary. The below diagram illustrates extractive summarization



2. **Abstractive text summarization:** This is a very interesting approach. Here, we generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences that were present. The sentences generated through abstractive summarization might not be present in the original text:



2.2 Literature Survey:

1. EXTRACTIVE SUMMARIZATION APPROACHES FOR SINGLE DOCUMENT SUMMARIZATION

Extractive summarizers aim to select the most important sentences in the document and also maintain a low redundancy in the summary.

1.1 Term frequency-inverse document frequency-based approach:

Bag-of-words model is constructed at the sentence level, with the usually weighted term frequency and inverse sentence frequency standard, where sentence frequency is calculated by finding the number of sentences in the document that contain that term. These sentence vectors are then scored by similarity to the query and the highest-scoring sentences are taken as part of the summary. This is a use of the Information Retrieval concept. Summarization is query-specific but can be adapted to be generic. To generate a generic summary, nonstop words that occur most frequently in the document may be taken as the query words. Since these words represent the topic of the document, they can generate generic summaries. Term frequency is usually 0 or 1. If users create query words the way they create for information retrieval, then the query-based summary generation would become generic summarization.

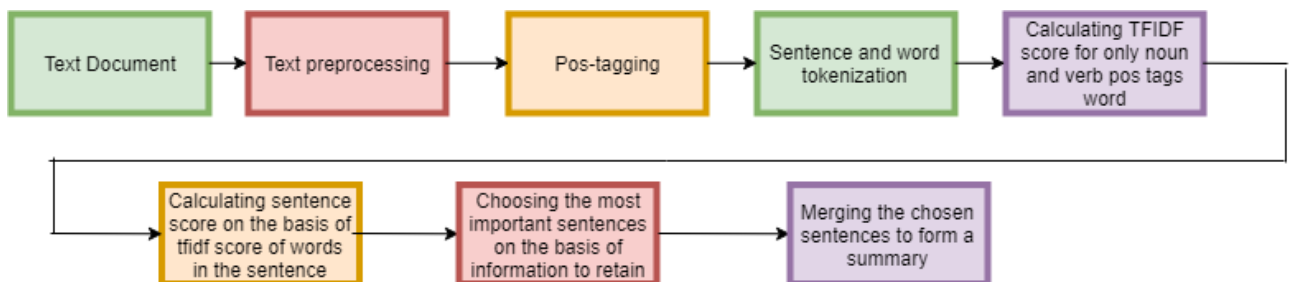


fig.Flow Of Extractive Summarization using tf-idf

Step1: Import the required library.

```
#Importing required libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from spacy.lang.en import English
import numpy as np
```

Step2: use the pipeline of nlp for preprocessing operations like Removing punctuations like . , ! \$() * % @, Removing URLs, Removing Stop words, Lower casing, Tokenization, Stemming, Lemmatization.

```
[ ] #Load spacy model for sentence tokenization
nlp = English()
nlp.add_pipe(nlp.create_pipe('sentencizer'))
```

Step 3: Assign numbers to the sentences according to the tokenization for maintaining the sequence of sentences for future use.

```
[ ] #Creating sentence organizer
# Let's create an organizer which will store the sentence ordering to later reorganize the
# scored sentences in their correct order
sentence_organizer = {k:v for v,k in enumerate(sentences)}

#Peeking into our sentence organizer
print("Our sentence organizer: \n", sentence_organizer)

Our sentence organizer:
{'The regulator, NHS Improvement, said it wanted more progress after the NHS had slipped behind schedule in its efforts to reduce the agency bi
```

Step 4: Create a tf-idf model and also calculate the score of the words.

```
#Creating TF-IDF model
# Let's now create a tf-idf (Term frequency Inverse Document Frequency) model
tf_idf_vectorizer = TfidfVectorizer(min_df=2, max_features=None,
strip_accents='unicode',
analyzer='word',
token_pattern=r'\w{1,}',
ngram_range=(1, 3),
use_idf=1, smooth_idf=1,
sublinear_tf=1,
stop_words = 'english')

[ ] # Passing our sentences treating each as one document to TF-IDF vectorizer
tf_idf_vectorizer.fit(sentences)

TfidfVectorizer(min_df=2, ngram_range=(1, 3), smooth_idf=1,
stop_words='english', strip_accents='unicode', sublinear_tf=1,
token_pattern='\\w{1,}', use_idf=1)
```

Step 5: Add the score and convert it into a vector.

```
[ ] # Transforming our sentences to TF-IDF vectors
sentence_vectors = tf_idf_vectorizer.transform(sentences)

sentence_vectors

<16x29 sparse matrix of type '<class 'numpy.float64''
with 92 stored elements in Compressed Sparse Row format>

[ ] # Getting sentence scores for each sentences
sentence_scores = np.array(sentence_vectors.sum(axis=1)).ravel()

# Sanity checkup
print(len(sentences) == len(sentence_scores))

True
```

Step 6: Select the top 3 sentences which have a high score and reorganize these sentences according to the sentence organizer.

```
[ ] # Getting top-n sentences
N = 3
top_n_sentences = [sentences[ind] for ind in np.argsort(sentence_scores, axis=0)[::-1][:N]]

# Let's now do the sentence ordering using our prebaked sentence_organizer
# Let's map the scored sentences with their indexes
mapped_top_n_sentences = [(sentence, sentence_organizer[sentence]) for sentence in top_n_sentences]
print("Our top_n_sentence with their index: \n")
for element in mapped_top_n_sentences:
    print(element)

# Ordering our top-n sentences in their original ordering
mapped_top_n_sentences = sorted(mapped_top_n_sentences, key = lambda x: x[1])
ordered_scored_sentences = [element[0] for element in mapped_top_n_sentences]

# Our final summary
summary = " ".join(ordered_scored_sentences)
```

Step 7: Summary As an output.

```
[23] summary

'The regulator, NHS Improvement, said it wanted more progress after the NHS had slipped behind schedule in its efforts to reduce the agency bill 1. The figures released by NHS Improvement do not cover NHS "bank" staff - effectively the health service's in-house agency - or money spent getting employed staff to do overtime. In one case a trust was quoted £130 an hour for a surgeon when the cap is meant to be £76, while another was quoted £70 an hour for a junior doctor when the fee is meant to be £35. NHS Improvement chief executive Jim Mackey said: "The NHS simply doesn't have the money to keep forking out for hugely expensive agency staff.'
```

1.2 Extractive Text summarization using BART_SUM:

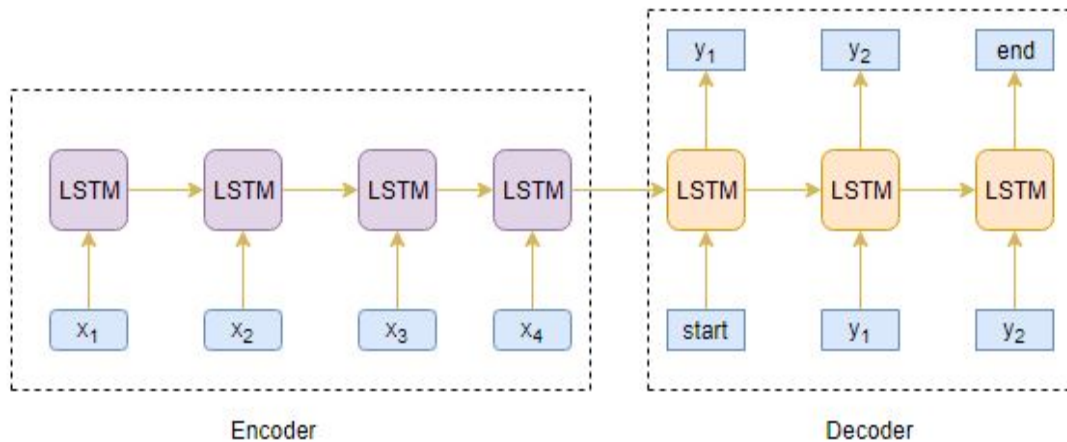
To use BERT for extractive summarization, we require it to output the representation for each sentence. However, since BERT is trained as a masked-language model, the output vectors are grounded to tokens instead of sentences. Meanwhile, although BERT has segmentation embeddings for indicating different sentences, it only has two labels (sentence A or sentence B), instead of multiple sentences as in extractive summarization. Therefore, we modify the input sequence and embeddings of BERT to make it possible for extracting summaries.

The BERTSUM model is an extension of the BERT model but in particular to the task of text summarization. The main difference between BERT and BERTSUM is the addition of inputting data with symbols to represent the start and end of sentences so that the model may learn sentence representations. Additionally, another difference is in the segment embeddings and how BERTSUM embeds pairs of sentences to learn adjacency patterns between each input sentence. The overall goal of this model is to give every sentence in the document a score representing the relevance of the sentence to the overall document, as a way to indicate to the model which sentence should be included in the summary.

2. Abstractive Summarization-

2.1 Abstractive Summarization using LSTM

Our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a short summary (which is a sequence as well). So, we can model this as a Many-to-Many Seq2Seq problem. Below is a typical Seq2Seq model architecture:



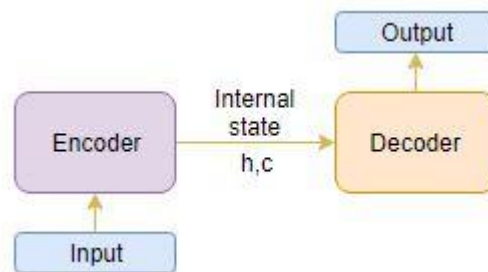
There are two major components of a Seq2Seq model:

- Encoder
- Decoder

Let's understand these two in detail. These are essential to understanding how text summarization works.

Understanding the Encoder-Decoder Architecture

The Encoder-Decoder architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problems where the input and output sequences are of different lengths. Let's understand this from the perspective of text summarization. The input is a long sequence of words and the output will be a short version of the input sequence.



Generally, variants of Recurrent Neural Networks (RNNs), i.e. Gated Recurrent Neural Network (GRU) or Long Short Term Memory (LSTM), are preferred as the encoder and decoder components. This is because they are capable of capturing long-term dependencies by overcoming the problem of vanishing gradient.

We can set up the Encoder-Decoder in 2 phases:

- Training phase

- Inference phase

Let's understand these concepts through the lens of an LSTM model.

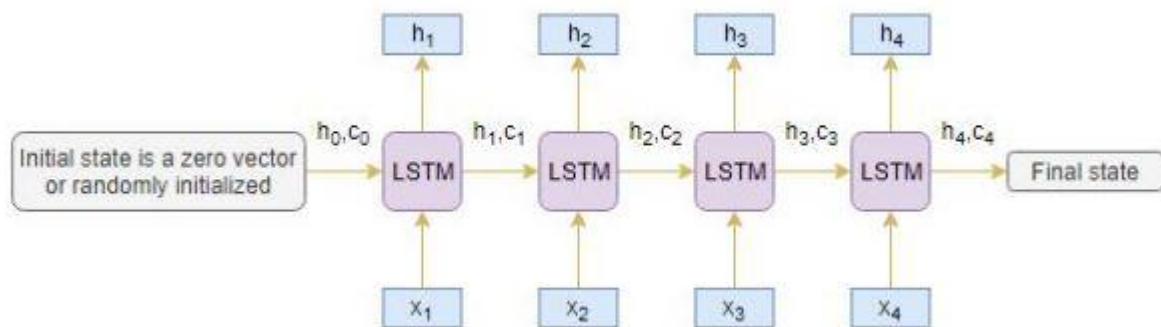
Training phase

In the training phase, we will first set up the encoder and decoder. We will then train the model to predict the target sequence offset by one timestep. Let us see in detail on how to set up the encoder and decoder.

Encoder

An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each timestep, one word is fed into the encoder. It then processes the information at every timestep and captures the contextual information present in the input sequence.

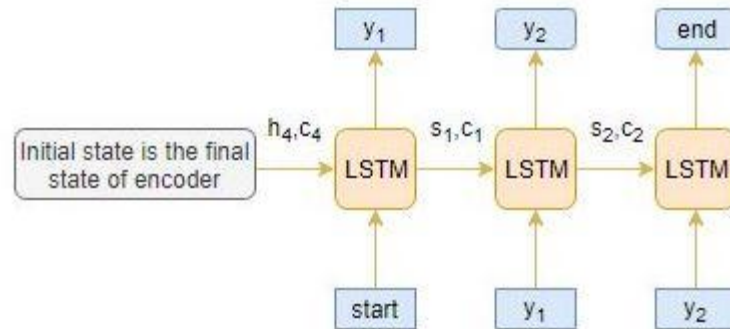
I've put together the below diagram which illustrates this process:



The hidden state (h_i) and cell state (c_i) of the last time step are used to initialize the decoder. Remember, this is because the encoder and decoder are two different sets of the LSTM architecture.

Decoder

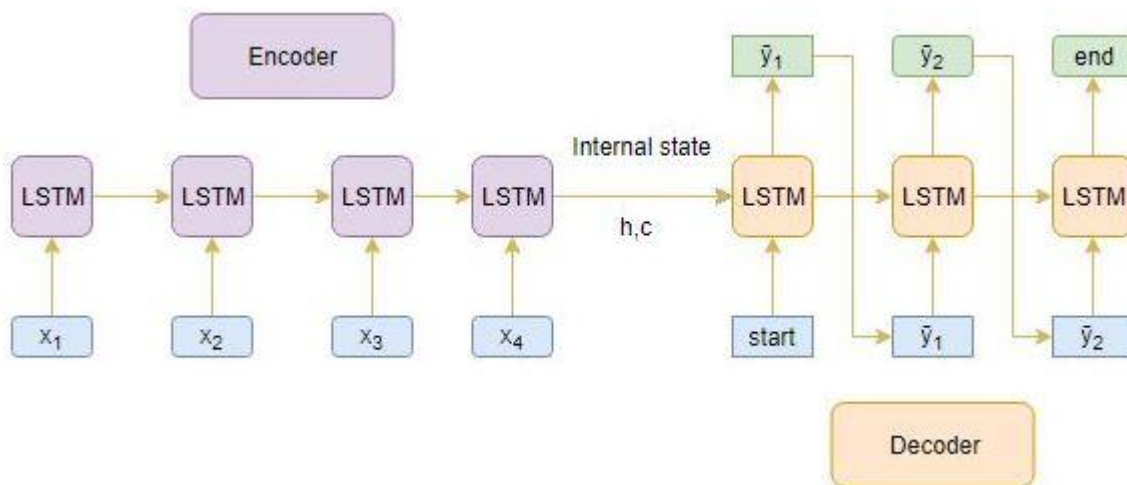
The decoder is also an LSTM network that reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. The decoder is trained to predict the next word in the sequence given the previous word.



$\langle \text{start} \rangle$ and $\langle \text{end} \rangle$ are the special tokens that are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would always be the $\langle \text{start} \rangle$ token. And the $\langle \text{end} \rangle$ token signals the end of the sentence.

Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence:



How does the inference process work?

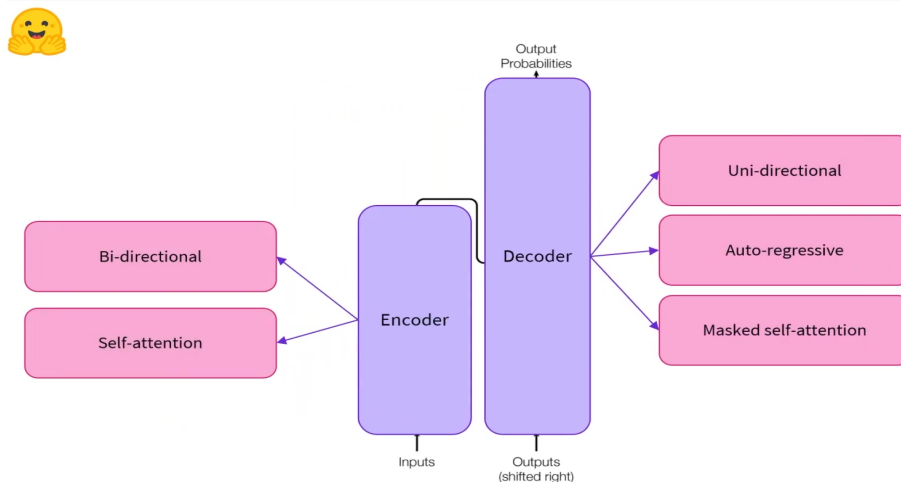
Here are the steps to decode the test sequence:

1. Encode the entire input sequence and initialize the decoder with the internal states of the encoder
2. Pass $\langle \text{start} \rangle$ token as an input to the decoder
3. Run the decoder for one timestep with the internal states
4. The output will be the probability for the next word. The word with the maximum probability will be selected

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

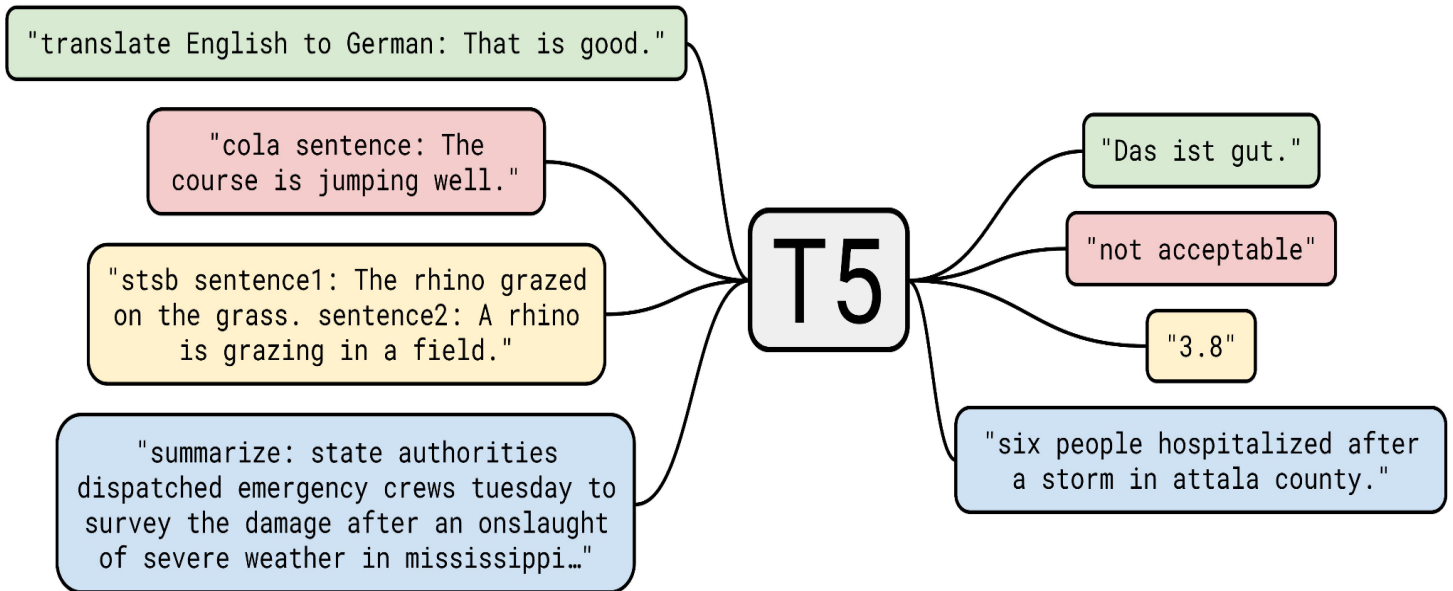
2.2 Abstractive Summarization using Transformers:

The Transformer architecture excels at handling text data which is inherently sequential. They take a text sequence as input and produce another text sequence as output. The Transformer architecture basically consists of two parts: Encoder and Decoder. These 2 can be used together but also independently.



- The encoder accepts input that represents texts and converts them into numerical representations which are known as embeddings. Every embedding generated also takes into account the words that are around it. It does this with the help of a self-attention mechanism. Encoders as stand-alone models are good at extracting meaningful information like sequence classification tasks, and question answering tasks. Example: BERT.
- The decoder can be used for most of the same tasks as the encoder albeit with generally a little loss of performance. The main difference between them is that when text is given as an input to the decoder, it generates an embedding(vector) but doesn't take into consideration the meaning or context of the word. It uses masked self-attention. Decoders are great at generating sequences. Example: GPT-2.
- The encoder-decoder model uses both of these architectures together. The encoder part takes care of understanding the sequence and the decoder part takes care of generating a sequence according to the understanding of the encoder. They are good for the sequence to sequence tasks like translation and summarization. Example: T5, BART, Pegasus.

We'll be using the T5-small model in our project. T5 stands for Text to Text Transfer Transformer. This model was presented in [Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer](#).



3. Dataset

3.1 XSum Dataset:

The Extreme Summarization (XSum) dataset is a dataset for the evaluation of abstractive single-document summarization systems. The goal is to create a short, one-sentence new summary answering the question “What is the article about?”. The dataset consists of 226,711 news articles accompanied by a one-sentence summary. The articles are collected from BBC articles (2010 to 2017) and cover a wide variety of domains (e.g., News, Politics, Sports, Weather, Business, Technology, Science, Health, Family, Education, Entertainment, and Arts). The official random split contains 204,045 (90%), 11,332 (5%) and 11,334 (5) documents in training, validation and test sets, respectively.

Extreme Summarization (XSum) Dataset.

There are three features:

- document: Input news article.
- summary: One sentence summary of the article.
- id: BBC ID of the article.

document (string)	summary (string)	id (string)
The full cost of damage in Newton Stewart, one of the areas worst affected, is still...	Clean-up operations are continuing across the Scottish Borders and Dumfries and...	35232142
A fire alarm went off at the Holiday Inn in Hope Street at about 04:20 BST on Saturday...	Two tourist buses have been destroyed by fire in a suspected arson attack in...	40143035
Ferrari appeared in a position to challenge until the final laps, when the Mercedes...	Lewis Hamilton stormed to pole position at the Bahrain Grand Prix ahead of Mercedes...	35951548
John Edward Bates, formerly of Spalding, Lincolnshire, but now living in London,...	A former Lincolnshire Police officer carried out a series of sex attacks on...	36266422
Patients and staff were evacuated from Cerahpasa hospital on Wednesday after a man...	An armed man who locked himself into a room at a psychiatric hospital in Istanbul...	38826984
Simone Favaro got the crucial try with the last move of the game, following earlier...	Defending Pro12 champions Glasgow Warriors bagged a late bonus-point victory over th...	34540833
Veronica Vanessa Chango-Alvarez, 31, was killed and another man injured when an Audi...	A man with links to a car that was involved in a fatal bus stop crash in...	20836172

3.2 SAMSum Dataset:

The SAMSum dataset contains about 16k messenger-like conversations with summaries. Conversations were created and written down by linguists fluent in English. Linguists were asked to create conversations similar to those they write on a daily basis, reflecting the proportion of topics in their real-life messenger conversations. The conversations were annotated with summaries. It was assumed that summaries should be a concise brief of what people talked about in the conversation in the third person. The SAMSum dataset was prepared by Samsung R&D Institute Poland.

Dataset Preview		
Subset		Split
samsun		train
id (string)	dialogue (string)	summary (string)
13729567	Leon: did you find the job yet? Arthur: no bro, still unemployed :D Leon: hahaha, LIVING LIFE Arthur: i love it, waking up at noon, watching sports - what else could a man want?...	Arthur is still unemployed. Leon sends him a job offer for junior project manager position. Arthur is interested.
13864634	Macca: i'm so exited today Adrien: why? Macca: I've never done ice climbing before Mark: Are you ready? Macca: think so Tobias: where are you doing this? Macca: not far from...	Macca has done ice climbing for the first time today, close to Reykjavik. He enjoyed it very much.
13815560	Isabella: fuck my life, I'm so not able to get up to work today Isabella: I need to call in sick :(Oscar: haha, well you certainly had a good time at the Christmas party...	Isabella feels bad after the Christmas party. She got drunk. She is ashamed to go back to work.
13731403	Tina: I'd only like to remind you that you owe me 50 bucks Lucy: Of course, I know. Lucy: I've already transferred the money but it's Sunday today so you'll have it in your bank...	Lucy owes Tina 50 dollars. She made a transfer but it is Sunday so the payment will be on Tina's account on Monday. Tina needs the money because she has been having expenses...
13729191	Betty: Please remind me next time that too much wine isn't good for me and me surrounding Betty: Hangover is one thing Betty: But I feel like never touching wine again Amber:...	Betty feels remorse she got drunk last night and went out of control.
13827937	Mary: Hi Mike! Mike: Hello :) Mary: do u have any plans for tonight? Mike: I'm going to visit my grandma. Mike: You can go with me. Mike: She likes u very much. Mary: Good idea...	Mike and Mary are going to visit Mike's grandma tonight. Mary will buy her some chocolate.
13828064	Laura: ok , I'm done for today-) Laura: let me know once u're free and we come back home together Kim: hmm.. ?? Laura: ok Kim: cool, wait for me at work, I'll call once I get...	Laura will pick up Kim from work around 7, and they will come back home together.

The dialogues look like what you would expect from a chat via SMS or WhatsApp, including emojis and placeholders for GIFs. The dialogue field contains the full text and the summary of the summarized dialogue.

Chapter 3

Implementation

1. Approach 1 (Using LSTM):

Our objective here is to generate a summary for the Extreme Summarization(XSum) dataset using the abstraction-based approach.

Import the Libraries:

```
import numpy as np
import pandas as pd
import re
import random
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import keras
from keras.layers import Input, LSTM, Embedding, Dense, concatenate, TimeDistributed, Add, dot, Activation
from keras.models import Model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from rouge import Rouge


import matplotlib.pyplot as plt
%matplotlib inline

import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

Load the XSum dataset:

```
from datasets import load_dataset
raw_datasets = load_dataset("xsum")

Using custom data configuration default
Reusing dataset xsum (C:\Users\Abhijit Das\.cache\huggingface\datasets\xsum\default\1.2.0\32c23220eaddb1149b16ed2e9430a05293768cffffbdf151058697d4c11f934)

100%  3/3 [00:00<00:00, 46.25it/s]
```

Preprocessing:

Performing basic preprocessing steps is very important before we get to the model building part. Using messy and unclean text data is a potentially disastrous move. So in this step, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem.

Here is the dictionary that we will use for expanding the contractions:

```
# define a dictionary of all possible contractions and their expanded forms
contraction_mapping = {'ain't': "is not", "aren't": "are not", "can't": "cannot", "'cause": "because", "could've": "could have", "couldn't": "could not", "didn't": "did not", "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not", "he'd": "he would", "he'll": "he will", "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "I'd": "I would", "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have", "I'm": "I am", "i'll": "i will", "i'll've": "i will have", "i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "it would", "it'll": "it will", "it'll've": "it will have", "it's": "it is", "let's": "let us", "ma'am": "madam", "might've": "might have", "mightn't": "might not", "mightn't've": "might not have", "must've": "must have", "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have", "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not", "shan't've": "shall not have", "she'd": "she would", "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have", "should've": "should have", "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have", "that's": "that is", "that'd": "that would", "that'd've": "that would have", "that's": "that is", "there'd": "there would", "there'd've": "there would have", "there's": "there is", "here's": "here is", "they'd": "they would", "they'll": "they will", "they'll've": "they will have", "they're": "they are", "they've": "they have", "wasn't": "was not", "we'd": "we would", "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've": "what will have", "what's": "what is", "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did", "where've": "where have", "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have", "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have", "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have", "y'all": "you all", "y'all'd": "you all would", "y'all'd've": "you all would have", "y'all're": "you all are", "y'all've": "you all have", "you'd": "you would", "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have", "you're": "you are", "you've": "you have"}
```

We have carried out the following preprocessing operations:

1. Convert text to lowercase
2. Expand the contractions ("isn't" to "is not")
3. Remove everything from the text except alphabets, '.' and ','
4. Remove single-character tokens

```
def text_cleaner(text):
    newString = text.lower()
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in newString.split()])
    newString = re.sub(r"'s\b|'b|'", "", newString)
    newString = re.sub("[^a-zA-Z.,]", " ", newString)

    # remove terms with length = 1
    long_tokens=[]

    for i in newString.split():
        if(len(i) > 1):
            long_tokens.append(i)

    # return preprocessed tweets
    return " ".join(long_tokens)
```

The distribution of the sequences:

Here, we will analyze the length of the reviews and the summary to get an overall idea about the distribution of the length of the text. This will help us fix the maximum length of the sequence:

```
text_word_count = []
summary_word_count = []

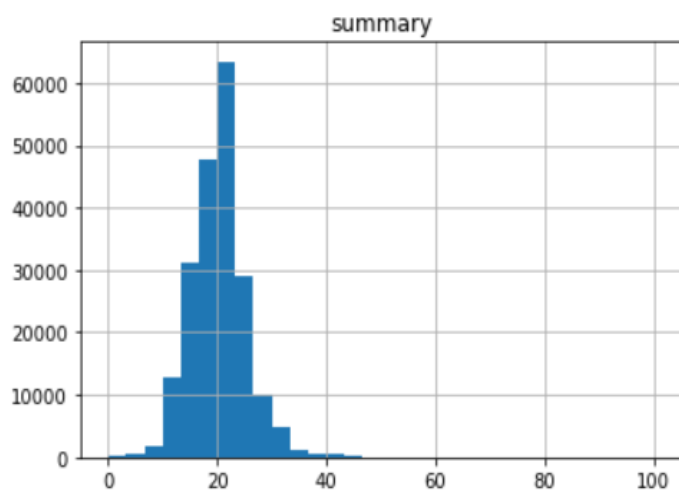
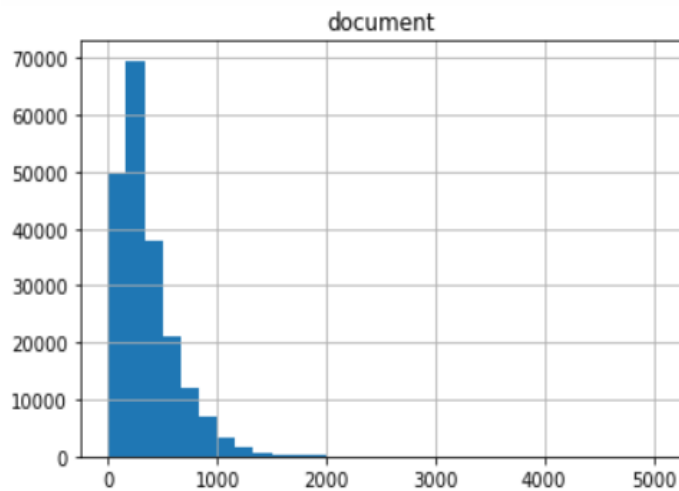
for i in train['document']:
    text_word_count.append(len(i.split()))

for i in train['summary']:
    summary_word_count.append(len(i.split()))

text_length_df = pd.DataFrame({'document':text_word_count})
text_length_df.hist(bins = 30,range=(0,5000))

summary_length_df = pd.DataFrame({'summary':summary_word_count})
summary_length_df.hist(bins = 30,range=(0,100))

plt.show()
```



Interesting. We can fix the maximum length of the documents to 1000 since that seems to be the majority of document length. Similarly, we can set the maximum summary length to 40:

```
# maximum length for document
max_text_len = 1000

# maximum length for summaries
max_summary_len = 40
```

Let's add the **start-of-sentence** and **end-of-sentence** tokens ("sostok" and "eostok") to the summaries.

```
train['summary'] = train['summary'].apply(lambda x : 'sostok ' + x + ' eostok')
test['summary'] = test['summary'].apply(lambda x : 'sostok ' + x + ' eostok')
```

Preparing the Tokenizer:

A tokenizer builds the vocabulary and converts a word sequence to an integer sequence. The tokenizers are built for document and summary::

```
x_tokenizer = Tokenizer(num_words=5000)
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr_seq     = x_tokenizer.texts_to_sequences(x_tr)
x_val_seq    = x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr         = pad_sequences(x_tr_seq, maxlen=max_text_len, padding='post')
x_val        = pad_sequences(x_val_seq, maxlen=max_text_len, padding='post')

#size of vocabulary
x_voc        = x_tokenizer.num_words + 1
```

Model building:

We are finally at the model-building part. But before we do that, we need to familiarize ourselves with a few terms which are required prior to building the model.

- Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep
- Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only
- Initial State: This is used to initialize the internal states of the LSTM for the first

timestep.

```
import keras
from keras.layers import Input, LSTM, Embedding, Dense, concatenate, TimeDistributed, Add, dot, Activation
from keras.models import Model
from keras.callbacks import EarlyStopping, ModelCheckpoint

# No. of hidden nodes
latent_dim = 300
# Dimension of embeddings
embedding_dim=100

## Encoder-Decoder Structure for model training

#Encoder
encoder_inputs=Input(shape=(max_text_len,))
encoder_embedding = Embedding(x_voc,embedding_dim, trainable=True, mask_zero=True)(encoder_inputs)

encoder_lstm = LSTM(latent_dim,return_sequences=True,return_state=True,go_backwards=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)

#Decoder
decoder_inputs=Input(shape=(None,))
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True,mask_zero=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim,return_sequences=True,return_state=True)
decoder_outputs, dec_state_h, dec_state_c = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

#Dense Layer
dense_layer = TimeDistributed(Dense(latent_dim, activation="tanh"))
dense_outputs=dense_layer(decoder_outputs)

#Output Layer
output_layer = TimeDistributed(Dense(y_voc, activation="softmax"))
decoder_outputs=output_layer(dense_outputs)

model = Model([encoder_inputs,decoder_inputs], decoder_outputs)
model.summary()
```

Output:

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 1000)]	0	[]
input_4 (InputLayer)	[(None, None)]	0	[]
embedding_2 (Embedding)	(None, 1000, 100)	500100	['input_3[0][0]']
embedding_3 (Embedding)	(None, None, 100)	300100	['input_4[0][0]']
lstm_2 (LSTM)	[(None, 1000, 300), (None, 300), (None, 300)]	481200	['embedding_2[0][0]']
lstm_3 (LSTM)	[(None, None, 300), (None, 300), (None, 300)]	481200	['embedding_3[0][0]', 'lstm_2[0][1]', 'lstm_2[0][2]']
time_distributed_2 (TimeDistri buted)	(None, None, 300)	90300	['lstm_3[0][0]']
time_distributed_3 (TimeDistri buted)	(None, None, 3001)	903301	['time_distributed_2[0][0]']
=====			
Total params: 2,756,201			
Trainable params: 2,756,201			
Non-trainable params: 0			

sparse categorical cross-entropy as the loss function since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
```

It is used to stop training the neural network at the right time by monitoring a user-specified metric. Here, I am monitoring the validation loss (val_loss). Our model will stop training once the validation loss increases:

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2, min_delta=0.0001)
mc = ModelCheckpoint('best_model_9.hdf5', monitor='val_loss', verbose=1, save_best_only=True, mode='min')
```

We'll train the model on a batch size of 32 and epoch of 50:

Initiate Model Training

```
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,mc], batch_si:
```

Inference:

Set up the inference for the encoder and decoder:

```
reverse_target_word_index=dict((v, k) for k, v in y_tokenizer.word_index.items())
reverse_source_word_index=dict((v, k) for k, v in x_tokenizer.word_index.items())

## Encoder-Decoder structure for model inference

# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb= dec_emb_layer(decoder_inputs)

# To predict the next word in the sequence, set the initial states to the states from the previous time step
dec_outputs, dec_h, dec_c = decoder_lstm(dec_emb, initial_state=[decoder_state_input_h, decoder_state_input_c])

#dense layer
dense_outputs=dense_layer(dec_outputs)

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = output_layer(dense_outputs)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
    [decoder_outputs2] + [dec_h, dec_c])
```

We are defining a function below which is the implementation of the inference process

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = y_tokenizer.word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token != 'eostok'):
            decoded_sentence = decoded_sentence + sampled_token + ' '

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eostok' or len(decoded_sentence.split()) >= (max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence.strip()
```

Let us define the functions to convert an integer sequence to a word sequence for summary as well as the reviews:

```
def seq2source(input_seq):
    newString = ''
    for i in input_seq:
        if(i != 0):
            newString = newString + reverse_source_word_index[i] + ' '
    return newString.strip()

source = []
for i in range(len(x_val)):
    source.append(seq2source(x_val[i]))

summary_val = ' '.join(i.split()[1:-1]) for i in summary_val

index = []
for i in range(len(source)):
    if(len(source[i].split()) >= 10):
        index.append(i)
```

2. Approach 2 (using pre-trained google/pegasus model):

PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization

Recent work pre-training Transformers with self-supervised objectives on large text corpora has shown great success when fine-tuned on downstream NLP tasks including text summarization. However, pre-training objectives tailored for abstractive text summarization have not been explored. Furthermore, there is a lack of systematic evaluation across diverse domains. In this work, we propose pre-training large Transformer-based encoder-decoder models on massive text corpora with a new self-supervised objective. In PEGASUS, important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. We evaluated our best PEGASUS model on 12 downstream summarization tasks spanning news, science, stories, instructions, emails, patents, and legislative bills. Experiments demonstrate it achieves state-of-the-art performance on all 12 downstream datasets measured by ROUGE scores. Our model also shows surprising performance on low-resource summarization, surpassing previous state-of-the-art results on 6 datasets with only 1000 examples. Finally, we validated our results using human evaluation and show that our model summaries achieve human performance on multiple datasets.

Pegasus pretrained model is used for abstractive summarization of some texts:

Pretrained Model

```
# Importing dependencies from transformers
from transformers import PegasusForConditionalGeneration, PegasusTokenizer
```

```
# Load tokenizer
tokenizer = PegasusTokenizer.from_pretrained("google/pegasus-xsum")
```

Downloading: 100%  3.36M/3.36M [00:05<00:00, 824kB/s]

```
# Load model
model = PegasusForConditionalGeneration.from_pretrained("google/pegasus-xsum")
```

3. Approach 3 (training t5-small model on SAMSum dataset)

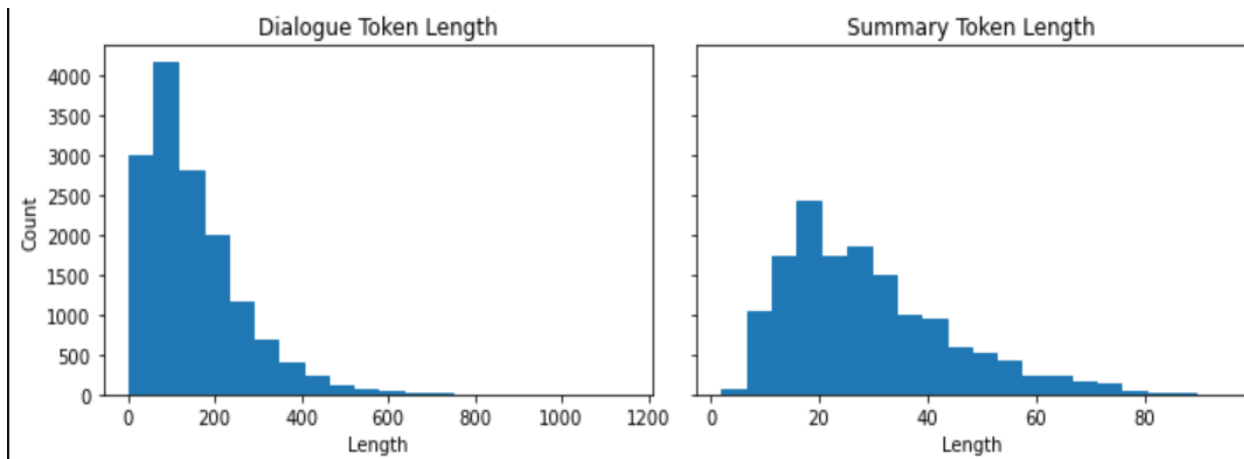
- **Step 1:** Let us import the tokenizer and model from the transformers library.

```
from transformers import T5Tokenizer, T5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("t5-base")
model = T5ForConditionalGeneration.from_pretrained("t5-base")
```

- **Step 2:** Before starting to train our model, let us check the length distribution of dialogue and summary.

```
d_len = [len(tokenizer.encode(text)) for text in samsum['train']['dialogue']]
s_len = [len(tokenizer.encode(text)) for text in samsum['train']['summary']]
```



Most of our dialogue lengths are between 100-200 tokens per dialogue and summary length is between 20-40 tokens per dialogue.

- **Step 3:** Now, we will tokenize our dataset. We'll set the maximum length to 1024 for dialogues and 128 for summaries.

```
def convert_examples_to_features(example_batch):
    input_encodings = tokenizer(example_batch["dialogue"],
                                max_length=1024,
                                truncation=True)

    with tokenizer.as_target_tokenizer():
        target_encodings = tokenizer(example_batch["summary"],
                                    max_length=128,
                                    truncation=True)

    return {"input_ids": input_encodings["input_ids"],
            "attention_mask": input_encodings["attention_mask"],
            "labels": target_encodings["input_ids"]}

dataset_samsum_pt = samsum.map(convert_examples_to_features,
                               batched=True)

columns = ["input_ids", "labels", "attention_mask"]
dataset_samsum_pt.set_format(type="torch", columns=columns)
```

- **Step 4:** Now, we need to create Data Collator. The main job of the Data Collator is to put together a list of samples into a single training mini-batch.

```
from transformers import DataCollatorForSeq2Seq

seq2seq_data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

- **Step 5:** Now we will call our Trainer API and define all the hyperparameters required. `trainer.train()` will start training our model.

```
[ ]: from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    't5-small-demo',
    num_train_epochs=1,
    warmup_steps=500,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    weight_decay=0.01,
    logging_steps=10,
    push_to_hub=True,
    push_to_hub_model_id='t5-small-demo',
    evaluation_strategy='steps',
    eval_steps=500,
    save_steps=1e6,
    gradient_accumulation_steps=16
)

[ ]: trainer = Trainer(model=model,
    args=training_args,
    tokenizer=tokenizer,
    data_collator=seq2seq_data_collator,
    train_dataset=dataset_samsum_pt["train"],
    eval_dataset=dataset_samsum_pt["validation"]
)

[ ]: trainer.train()
```

- **Step 6:** After our model is trained, we can log into the hugging face so that we can push our model to the hub.

```
[52]: from huggingface_hub import notebook_login
notebook_login()
```



Copy a token from your Hugging Face tokens page and paste it below.

Immediately click login after copying your token or it might be stored in plain text in this notebook file.

Token:

Login

Pro Tip: If you don't already have one, you can create a dedicated 'notebooks' token with 'write' access, that you can then easily reuse for all notebooks. |

Logging in with your username and password is deprecated and won't be possible anymore in the near future. You can still use them for now by clicking below.

Use password

4. Implementation

The models were implemented as a web application and a web API using the **Flask Framework**, a micro web framework written in python.

4.1 Developing a Web Application

Step 1: A basic HTML page, index.html is developed to enter the conversational data and display the respective summary.

```

1 <html>
2 <head>
3 <!-- CSS only -->
4 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BnE4kqBq781YhFIdvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
5 </head>
6 <body>
7 <div class="mt-4 p-5 bg-primary text-white rounded text-center">
8 <h1>Conversational Text Summarizer</h1>
9 <p>Implementation</p>
10 </div>
11 <div class="container">
12 <div class="row">
13 <div id="inputForm" action="/displaysummary" method="POST">
14 <div class="form-group row">
15 <label for="inputTextData" class="form-label col-form-label">Input text</label>
16 <div class="col-12">
17 <textarea class="form-control" id="inputText" name="inputText" placeholder="Write here....." rows="12"></textarea>
18 </div>
19 </div>
20 </div>
21 <div class="form-group row mt-2">
22 <div class="col-12">
23 <button type="submit" class="btn btn-success">Summarize</button>
24 <button type="reset" class="btn btn-secondary md-2">Clear</button>
25 </div>
26 </div>
27 </form>
28 </div>
29 </div>
30 <div class="row mt-2">
31 <div class="col-12 col-md-6">
32 <div class="col-form-label"><h2>Pre-trained BART Abstractive Summary</h2></div><br></div>
33 <div class="col-12 col-md-6">
34 <div class="col-form-label"><h2>Custom Trained T5 Abstractive Summary</h2></div><br></div>
35 <div class="col-12 col-md-6">
36 <div class="col-form-label"><h2>Pre-trained BART Abstractive Summary</h2></div><br></div>
37 <div class="col-12 col-md-6">
38 <div class="col-form-label"><h2>Custom Trained T5 Abstractive Summary</h2></div><br></div>
39 </div>
40 </div>
41 </div>
42 </div>
43 </div>
44 </div>

```

Step 2: created an app.py file, where the code for the flask app is written. The required packages and the respective saved models are downloaded from the local directory (the models can be loaded directly from the HuggingFace Hub too).

```

app.py > renderDisplayForm
1 from logging import debug
2 from flask import Flask,render_template,request
3 from transformers import pipeline
4 import os
5
6 bart_summarizer= pipeline('summarization',model=os.path.join(os.getcwd(),os.path.join('models','bart-model')))
7 t5_summarizer=pipeline('summarization',model=os.path.join(os.getcwd(),os.path.join('models','custom-t5-model')))

```

Step 3: Starting the Flask app and providing a default route('/') to the html page and a /displaysummary route for displaying the respective summary of the input data on the html page.

```
app=Flask(__name__)

@app.route('/')
def main_page():
    return render_template('index.html')

@app.route('/displaysummary',methods=['GET','POST'])
def renderDisplayForm():
    if request.method=='GET':
        return f'The url /data is accessed directly'
    if request.method=='POST':
        form_data=request.form

        bart_summary=bart_summarizer(form_data['inputText'])[0]['summary_text']
        t5_summary=t5_summarizer(form_data['inputText'])[0]['summary_text']
        # return form_data
        return render_template('index.html',bart_summary=bart_summary,t5_summary=t5_summary)
```

Step 4: Finally we assign the 5000 port number on which the flask server should run on. Additionally the host ip is set to 0.0.0.0 so as to get access from any ip address (this is done for deployment over docker)

```
if __name__=='__main__':
    port=int(os.environ.get('PORT',5000))
    app.run(debug=True,host='0.0.0.0',port=port)
```

4.2 Developing a Web API

The corresponding API is a **REST API (REpresentational State Transfer)**. REST is an architectural style that defines a set of constraints to be used for creating web services. REST API is a way of accessing web services in a simple and flexible way without having any processing.

To develop a REST API we used the **Flask_RESTful**, which is an extension for Flask that adds support for quickly building REST API.

Step 1: Loading the required packages and the saved trained model from the HuggingFace hub. We have initialized the flask app and then assigned the flask app as an API.

```
main.py > ...
1  from flask import Flask, jsonify, request
2  from flask_restful import Resource, Api, reqparse
3  from transformers import pipeline
4  from logging import debug
5
6  # creating the flask app
7
8  # bart_summarizer= pipeline('summarization',model='lidiya/bart-large-xsum-samsum')
9  # print('bart success')
10 t5_summarizer=pipeline('summarization',model='anegi/t5smallmodel')
11 print('t5 success')
12
13 app=Flask(__name__)
14 api=Api(app)
15
```

Step 2: Defining a parser object to parse the json object consisting of the details which will be sent to the API. Using the parser we will extract the required data from the json object.

```
parser=reqparse.RequestParser()
parser.add_argument('inputText',action='append',required=True)
```


Step 3: We defined a **TextSummarization** class that acts as an API resource. The function **get()** acts as a get request.

```
class TextSummarization(Resource):
    def get(self):

        args=parser.parse_args()

        # bart_summary=bart_summarizer(args['inputText'])

        t5_summary=t5_summarizer(args['inputText'])

        return jsonify({'t5_summary':t5_summary[0]['summary_text']})
```

Step 4: then we make the TextSummarization class as a resource with the URL **‘/summarize’** for the API. Then we run the server with the **app.run()** method.

```
api.add_resource(TextSummarization, "/summarize")

if __name__ == "__main__":
    app.run()
```

4.3 Developing a Docker Image for the Web Application.

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

Step 1: create a **requirements.txt** file which consists of all the packages and library names along with their versions which are needed for the running of the web application.

```
requirements.txt
1 torch
2 torchaudio
3 torchvision
4 flask
5 huggingface-hub
6 transformers
7 gunicorn
```

Step 2: create a Docker file. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

```
Dockerfile
1 FROM python:3.10-buster
2 RUN mkdir webapp
3
4 COPY ./css/ /webapp/css/
5 COPY ./models/ /webapp/models/
6 COPY ./templates/ /webapp/templates/
7 COPY ./app.py /webapp/
8 COPY ./model_downloading.py /webapp/
9 COPY ./requirements.txt /webapp/
10
11 WORKDIR webapp
12 RUN ls -a
13
14 RUN pip install -r requirements.txt
15 ENTRYPOINT [ "python" ]
16 CMD [ "app.py" ]
17
18
19
```

FROM: is used to mention the base image on which our container has to be developed.

RUN: used to run shell commands on the container

COPY: used to copy files and directories from the host machine to the docker container

WORKDIR: used to define a working directory of a docker container at any given time.

ENTRYPOINT: is preferred when you want to define a container with a specific executable

CMD: defines default commands and/or parameters for a container.

Step 3: Build the docker image. after the image is built, the image is pushed to the Docker Hub.

```
(base) aakash@aakash-Inspiron-7501:~/CDAC content/Final Project/Text Sum Rest API$ sudo docker build -t app-image .
```

```
(base) aakash@aakash-Inspiron-7501:~/CDAC content/Final Project/Text Sum Rest API$ sudo docker tag app-image aakash280500/abstractiv-conversation-summarization
```

```
(base) aakash@aakash-Inspiron-7501:~/CDAC content/Final Project/Text Sum Rest API$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
registry.heroku.com/abst-convo-summ/web	latest	af5b31213d53	3 days ago	5.41GB
aakash280500/abstractive_conversation_summarization	1.0.0	17cc0711ee22	3 days ago	5.41GB
aakash280500/abstractive_conversational_summarization	1.0.0	17cc0711ee22	3 days ago	5.41GB
app-image	latest	17cc0711ee22	3 days ago	5.41GB
registry.heroku.com/abst-convo-summ/web	<none>	17cc0711ee22	3 days ago	5.41GB
python	3.10-buster	fe9883b0ab92	6 days ago	892MB

Chapter 4

Results

1. Abstractive summarization results using deep learning LSTM model:

```
for i in random.sample(range(0,len(text_val)+1),20):
    print("Review:",text_val[index[i]],"\n")
    actual = summary_val[index[i]]
    print("Actual summary:",actual)
    model_out=decode_sequence(x_val[index[i]].reshape(1,max_text_len))
    print("Predicted summary:",model_out)
    print("\n")
    print("Eval Metrics:")
    print(rouge.get_scores(model_out, actual, avg=True))
    print("\n")
```

Here are a few summaries generated by the model with ROUGE evaluation metrics:

Review: Oil prices suffered a second year of steep losses and are expected to take at least another year to clear as the international surplus continues.
 The Dow Jones was down 178.84 points or 1.03%, at 17,425.03.
 The S&P 500 was down 0.95% at 2,043.86, while the tech-heavy Nasdaq composite was 1.15% lower at 5,007.41.
 The oil price collapse sent global markets reeling throughout 2015.
 Shares of US oil giants Chevron and Exxon Mobil were down 0.17% and 0.22% respectively for the day.
 Energy stocks have taken a beating this year, with the S&P energy sector losing nearly 24% in the last twelve months.
 For the year the S&P 500 was down 0.7% while the Dow Jones ended 2.2% lower.
 The Nasdaq, however was a bright spot closing 5.7% higher for 2015.
 Trading volumes were thin on the last day of the year.
 Apple was down 1.92% weighing on the Nasdaq.
 McDonald's was down 1.08% at \$118 and weighed on the Dow the most.
 Stocks were led lower as US jobless claims increased by 20,000 to 287,000 last week, wildly missing forecasts of 270,000.
 Brent crude oil was up 3% at \$37.60 per barrel for the day but down 35% over the year. US light crude was 1.2% higher at \$37.04 but down 30% for the year.

Actual summary: wall street finished its final day of down, marking its worst annual performance in seven
 Predicted summary: oil prices rose by in the third quarter of the year after oil prices rose

Eval Metrics:
 {'rouge-1': {'r': 0.14285714285714285, 'p': 0.18181818181818182, 'f': 0.15999999507200013}, 'rouge-2': {'r': 0.0, 'p': 0.0, 'f': 0.0}, 'rouge-l': {'r': 0.07142857142857142, 'p': 0.09090909090909091, 'f': 0.0799999950720003}}

Review: Bath and North East Somerset (Banes) Council said a pay dispute between the Unite union and the contractor Kier was over following successful talks.
The council said a backlog of waste would be cleared by the weekend.
Only half of scheduled collections have taken place recently due to industrial action over what Unite called a "glaring pay ine quality".

Actual summary: planned strikes by waste collection workers around bath have been called

Predicted summary: two more strikes have been cancelled after the collapse of an emergency department

Eval Metrics:

```
{'rouge-1': {'r': 0.2727272727272727, 'p': 0.23076923076923078, 'f': 0.24999999503472223}, 'rouge-2': {'r': 0.1, 'p': 0.08333333333333333, 'f': 0.09090908595041348}, 'rouge-l': {'r': 0.2727272727272727, 'p': 0.23076923076923078, 'f': 0.24999999503472223}}
```

2. Abstractive summarization result using pre-trained google/pegasus model

```
from rouge import Rouge
rouge = Rouge()
```

```
text = """
Kerry Reeves died two days after being shot in Abingdon on 3 November.
Billy Johnson, 20, from Ripon Court, Corby, Northamptonshire, and Charles Noble, 20, from Kempton Avenue, Northolt, Ealing, have
They appeared at Oxford Magistrates' Court and will next appear at Oxford Crown Court on Monday.
Ms Reeves, 26, was found in Thornhill Walk. She was taken to the John Radcliffe Hospital in a critical condition where she later
A post-mortem examination was carried out on Monday, where the preliminary cause of death was given as a gunshot wound to the head.
"""
```

```
actual = '''
men have appeared in court charged with the murder of woman who was shot in the head in
'''
```

```
# Create tokens - number representation of our text
tokens = tokenizer(text, truncation=True, padding="longest", return_tensors="pt")
# Summarize
summary = model.generate(**tokens)
# Decode summary
predicted = tokenizer.decode(summary[0])
print("TEXT")
print(text)
print("ACTUAL SUMMARY")
print(actual)
print("PREDICTED SUMMARY",end="\n\n")
print(predicted,end="\n\n")
print("EVALUATION METRICS",end="\n\n")
print(rouge.get_scores(predicted, actual, avg=True))
```

Below shows the Text, Actual Summary, Predicted Summary by Pegasus model, and the ROUGE metrics for model performance evaluation:

TEXT

Kerry Reeves died two days after being shot in Abingdon on 3 November.
 Billy Johnson, 20, from Ripon Court, Corby, Northamptonshire, and Charles Noble, 20, from Kempton Avenue, Northolt, Ealing, have both been charged with her murder.
 They appeared at Oxford Magistrates' Court and will next appear at Oxford Crown Court on Monday.
 Ms Reeves, 26, was found in Thornhill Walk. She was taken to the John Radcliffe Hospital in a critical condition where she later died.
 A post-mortem examination was carried out on Monday, where the preliminary cause of death was given as a gunshot wound to the head.

ACTUAL SUMMARY

Two men have appeared in court charged with the murder of woman who was shot in the head in Oxfordshire.

PREDICTED SUMMARY

Two men have appeared in court charged with the murder of a woman who was shot in the head in Oxfordshire.

EVALUATION METRICS

```
{'rouge-1': {'r': 1.0, 'p': 0.8333333333333334, 'f': 0.9090909041322315}, 'rouge-2': {'r': 0.9411764705882353, 'p': 0.8, 'f': 0.8648648598977356}, 'rouge-l': {'r': 1.0, 'p': 0.8333333333333334, 'f': 0.9090909041322315}}
```

3. Abstractive summarization using our trained t5-small model

To see how well our model has been summarized, we need to generate a random conversation and give supply it to our model.

Testing our model on random conversation

```
[11]: conversation = '''
Aakash: Do you watched last night match?
Vikas: No, I was busy. What happened?
Aakash: Barcelona won their game against Sevilla 1-0. Pedri scored an sensational goal.
Vikas: Oh really!!! I have to watch the highlights at home.
Aakash: Yes, I am so excited for the new upcoming season. Xavi really doing wonders.
Vikas: Exactly!! Xavi has changed the way Barcelona play. The good days are coming back.
'''

[12]: from transformers import pipeline

[13]: summarizer = pipeline('summarization', model = 'anagi/t5smallmodel')

[14]: summarizer(conversation, max_length=90, min_length = 35)

[14]: [{'summary_text': 'Pedri scored a sensational goal for Barcelona against Sevilla . Xavi has changed the way they play. Aakash is excited for the new season.'}]
```

We can clearly see it gives a good summary of the above-provided conversation. Now, we would like to check the quality of the summary generated from the above model. For this, we'll be using the ROUGE metric. The basic idea behind it is to assign a single numerical score to a summary that tells us how good it is compared to the reference provided.

The different types of ROUGE are

- ROUGE-N: It measures the number of matching 'n-grams' between our model-generated summary and a human-generated reference. ROUGE-1 means we would be measuring the match rate of unigrams between our model output and reference. ROUGE-2 would use bigrams.
- ROUGE-L: It measures the longest common subsequence(LCS) between our model output and reference which means we count the longest sequence of tokens that are shared between both.

Let us now create a humanly generated summary of the above-provided conversation and then calculate the ROUGE metrics.

ROUGE metric

- On our trained model(t5-small)

```
[32]: from rouge_score import rouge_scorer

[43]: scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'],
    use_stemmer=True)

# Adding a summary that a human would generate when the above conversation is provided to them.
scores = scorer.score('Pedri scored a sensational goal for Barcelona against Sevilla . Xavi has changed the way they play. Aakash is excited for the new season.',
    'Vikas asked Aakash about the match where Barcelona played against Sevilla and won. Pedri scored the only goal. Both are excited for the upcoming season.'')
```

- Converting the scores to dataframe

```
[44]: df_score = pd.DataFrame(scores)
```

```
[45]: df_score.index = ['precision', 'recall', 'fmeasure']
```

```
[46]: df_score
```

```
[46]:
```

	rouge1	rouge2	rougeL
precision	0.480000	0.083333	0.280000
recall	0.521739	0.090909	0.304348
fmeasure	0.500000	0.086957	0.291667

We can also create a demo for our created model with the help of hugging face space. Here, we can compare our model's summary with other pre-trained model summaries.

[Models](#)
[Datasets](#)
[Spaces](#)
[Docs](#)
[Solutions](#)
[Pricing](#)

[Spaces:](#)
anegi/Comparing-dialogue-summarization-models
like 1
See logs
Running

[App](#)
[Files and versions](#)
[Settings](#)
[Linked Models](#)

Comparing different dialogue summarization models

Here we are going to compare different summarization model namely BART-LARGE model which is trained on samsum data and t5small model which is also trained on the same dataset.

Input Text

A: Hello B. How are you?
 B: I'm good. How's your course going on these days?
 A: It's going well. Pretty tiring but overall fun.
 B: That's great to hear. If you are free wanna grab a cup of coffee?
 A: Sure!!!! I recently discovered a new cafe near my home.
 B: Awesome. Let's go.

anegi/t5smallmodel: Generated Text

A and B are going to a new cafe near their home.

philschmid/bart-large-cnn-samsum: Summary

A and B are going to grab a coffee at a new cafe near A's home. A's course is going well and he finds it tiring but fun. B is also enjoying his course. A and B will meet at the cafe soon.

lidiya/bart-large-xsum-samsum: Summary

B and A are going to grab a coffee at a new cafe near A's home.

Clear

Submit

Examples

A: Hello B. How are you? B: I'm good. How's your course going on these days? A: It's going well. Pretty tiring but overall fun. B: That's great to hear. If you are free wanna grab a cup of coffee? A: Sure!!!! I recently discovered a new cafe near my home. B: Awesome. Let's go.

Rohit: Hi, how're you? Mahesh: I'm fine. What about you? Rohit: Good. How's your work going on? Mahesh: Not great. Rohit: Why? What happened? Mahesh: My workplace is far from my home. Most of my time is spent commuting and I'm not able to give time to my family. Rohit: Oh!! That sounds taxing. What are you planning to do now? Mahesh: I will again start applying for jobs near my home. Rohit: Best of luck man!!

[view the api](#)
[built with gradio](#)

4. Extractive summarization using BART_SUM:

```
import datetime
import argparse
import bart_sum
import logging
import presumm.presumm as presumm

logger = logging.getLogger(__name__)

def do_summarize(contents):
    document = str(contents)
    logger.info("Document Created")

    doc_length = len(document.split())
    logger.info("Document Length: " + str(doc_length))

    min_length = int(doc_length/6)
    logger.info("min_length: " + str(min_length))
    max_length = min_length+200
    logger.info("max_length: " + str(max_length))

    transcript_summarized = summarizer.summarize_string(document, min_length=min_length, max_length=max_length)
    with open("summarized.txt", 'a+') as file:
        file.write("\n" + str(datetime.datetime.now()) + ":\n")
```

Summarization aims to condense a document into a shorter version while preserving most of its meaning. Abstractive summarization task requires language generation capabilities to create summaries containing novel words and phrases not featured in the source document. Extractive summarization is often defined as a binary classification task with labels indicating whether a text span (typically a sentence) should be included in the summary.

Input:

```
Google CEO Sundar Pichai spoke exclusively to NDTV at the Google headquarters in California.

4
Mountain View, California: As Google geared up to launch a new wave of products and services including the much-awaited
NDTV: Sundar Pichai, God Bless you, thank you very much for joining us.
Sundar Pichai: Pleasure seeing you as well.
NDTV: Your success is an absolute, absolute inspiration for all of us.
Sundar Pichai: Thank you, you know I grew up watching you on the news channel as well
NDTV: Kindergarten?
Sundar Pichai: Not quite kindergarten, I was in college by then.
NDTV: You know I say that we are really inspired by you but there is a real puzzle to fix, one you are not a dropout, he
Sundar Pichai: First of all, IIT was too much fun to actually drop out, so wasn't there a movie about it or something?
NDTV: I totally believe that. And I think, you reflected in your handling of Google, how you changed the ethos here, the
Sundar Pichai: Yes. I always felt that even if you work 40 hours a week, you spend more time at work than the rest of yo
NDTV: We've talked to a lot of Google people and they all say it starts from you. Now Androids, look at that guy there,
Sundar Pichai: You are close to the place where Android gets built.
```


Output:

2022-04-09 09:10:44.141969:

Google CEO Sundar Pichai spoke exclusively to NDTV at the Google headquarters in California. Sundar: IIT was too much fun to actually drop out, so was not there a movie about it or something? It was more like that. I have always felt being a good guy and doing well are not necessarily at odds with each other so always felt that applied. I love using Google Maps and in India it actually works well, we are constantly trying to make it better, but I think the traffic in India is a challenge, so hopefully that Maps makes a small difference. The best food in the world is here, so nice most of the time to do that.

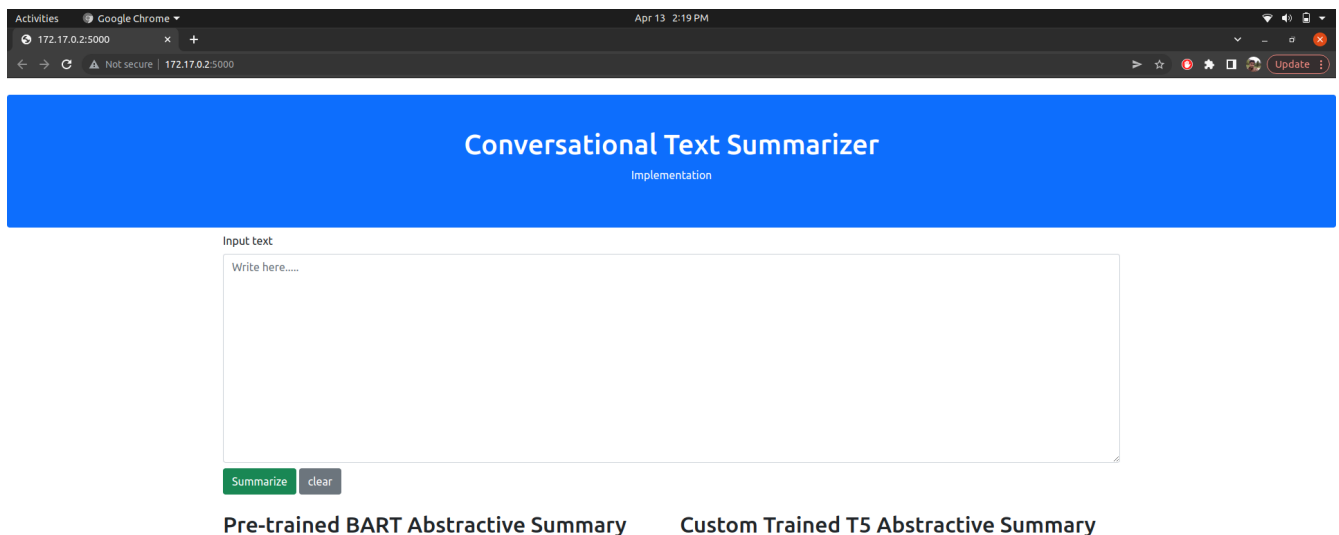
5. Deployment

5.1 Working of web application on Docker

The docker container called 't1' is created and run using the 'app-image' of our flask web application.

```
(base) aakash@aakash-Inspiron-7501:~/CDAC content/Final Project/Text Sum Rest API$ sudo docker run -ti --name t1 -p 5000:5000 app-image
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses (0.0.0.0)
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000 (Press CTRL+C to quit)
* Restarting with stat
```

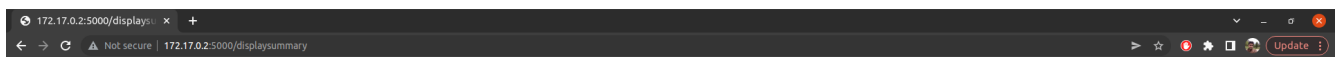
The respective web application is running on '172.17.0.2:5000'.



A sample conversation data is given to it and the respective predictions for the pre-trained BART model and our custom-trained T5 model are produced and displayed on the web page.

Input text

```
Nathan: Hey, Alicia?
Alicia: Oh hey, I didn't see you there. Did you already get a table?
Nathan: Yeah, right over here.
Alicia: I'm glad we had time to meet up.
Nathan: Me too. So, what's going on?
Alicia: Oh, not much. You?
Nathan: Not much. Hey, how did your interview go? Wasn't that today?
Alicia: Oh, yeah. I think it went well. I don't know if I got the job yet, but they said they would call in a few days.
Nathan: Well, I'm sure you did great. Good luck.
Alicia: Thanks. I'm just happy that it's over. I was really nervous about it.
Nathan: I can understand that. I get nervous before interviews, too.
Alicia: Well, thanks for being supportive. I appreciate it.
```



Conversational Text Summarizer

Implementation

Input text

Write here.....

Summarize

clear

Pre-trained BART Abstractive Summary

Alicia had an interview today. It went well, but she doesn't know if she got the job yet.

Custom Trained T5 Abstractive Summary

Alicia didn't know if she got the job yet, but they said they would call in a few days . Alicia was nervous about the interview before the interview.

5.2 Working of REST API

We run the flask_restful application on our system.

```
(project_env) aakash@aakash-Inspiron-7501:~/CDAC content/Final Project/Text Sum Rest API$ /home/a
est API/main.py"
t5 success
* Serving Flask app 'main' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

the API is running on '127.0.0.1:5000' (localhost:5000)

We use the respective code for testing our web API. The input data is sent to the API as a JSON object.

```
rest-api-test.py > ...
1  import requests
2
3  url='http://127.0.0.1:5000/summarize'
4
5
6  data={'inputText':"""
7  Sam: Oh? Bob!
8
9  Bob: Hey Sam! Good to see you!
10
11  Sam: How's it going?
12
13  Bob: Yeah, good. Working a lot. And you?
14
15  Sam: I went back to school.
16
17  Bob: Good for you!
18
19  Mike and Jim
20
21  Jim: Mike?
22
23  Mike: Jim?
24
25  Jim: What have you been up to?
26
27  Mike: Working a lot.
28
29  Jim: That sounds hard.
30
31  Mike: How's the family?
32
33  Jim: Everyone is good. Thanks!
34  """}
35
36  summaries=requests.get(url,json=data)
37
38  print(summaries.json())
```

and from the API we get the summary in the form of JSON

```
(project_env) aakash@aakash-Inspiron-7501:~/CDAC content/Final Project/Text Sum
Rest API$ python rest-api-test.py
{'t5_summary': "Bob and Jim are working a lot. Jim went back to school and he's
going to get back to the family. Jim and Mike are going to meet at the school."}
```

Chapter 5

Conclusion

- Text summarization is an interesting research topic among the NLP community that helps produce concise information.
- Abstractive summarization as of now requires heavy machinery for language generation and is difficult to replicate in the domain-specific areas.
- The important thing that is considered interesting from the review that has been done is the results of the analysis which states that extractive summaries are relatively easier than abstractive summaries which are very complex, extractive summaries are still the topic of current favorite trends. This is because there are still many things that are a challenge for researchers to do. It also can be seen that the most important features to produce a good summary are keywords, frequency, and similarity.
- Automatic text summarization only allows people to cut down on the reading necessary but also frees up time to read and understand otherwise overlooked written works. It is only a matter of time before such summarizers get integrated so well that they create summaries indistinguishable from those written by humans.
- Transformers are models that can translate text, write stories and summarize large text data. Transformers could be very efficiently parallelized which meant we could train some really big models.
- We need high GPU processing to train large models such as BERT, and PEGASUS. We could also use pre-trained models available on the Huggingface site to check how good these work.

Chapter 6

References

- <https://huggingface.co/datasets/samsum>
- <https://huggingface.co/philschmid/bart-large-cnn-samsum>
- <https://www.youtube.com/c/HuggingFace/playlists>
- <https://www.oreilly.com/library/view/natural-language-processing/9781098103231/>
- <https://towardsdatascience.com/text-summarization-using-tf-idf-e64a0644ace3>
- <https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/>
- <https://medium.com/voice-tech-podcast/automatic-extractive-text-summarization-using-tfidf-3fc9a7b26f5>
- <https://analyticsindiamag.com/hands-on-guide-to-extractive-text-summarization-with-bertsum/>