Dialogue Text Summarization

Final Project Report

Name: Jessica Anna James

Phone Number: 4256558646

james.jess@northeastern.edu

Signature of Student: Jessica Anna James

Submission Date: 12/15/2023

Introduction

The rapid evolution of natural language processing (NLP), notably in text summarization, has been catalyzed by neural network models and richly labeled datasets. However, despite the progress in monologic text summarization, dialogue summarization remains a relatively underexplored domain within the NLP research landscape. This gap primarily stems from the scarcity of datasets explicitly tailored for dialogue texts, posing significant limitations on the development of robust dialogue summarization models.

The current dialogue summarization research heavily relies on datasets like the AMI meeting corpus, which, although rich in content, presents constraints in scale and predominantly focuses on virtual multi-party meeting records. This limitation obstructs substantial advancements in the field, considering the inherent differences between existing datasets and real-life spoken dialogues. The SAMSum dataset, while a recent addition, features short conversations from messenger apps, exhibiting distinct language styles and topics that differ from spontaneous spoken daily dialogues.

The absence of a comprehensive and diverse dialogue summarization dataset introduces several challenges. The disparity in language style, topic diversity, and scenario complexity between existing corpora and real-life dialogues hampers the development of robust models. Moreover, the inherent characteristics of daily conversations, including varied communication patterns, task-oriented nature, and longer dialogue lengths, pose unique hurdles in the summarization process. The complexity is further amplified by the need to comprehend and summarize multi-turn dialogues while tracking entities and events across turns.

To bridge these gaps, the DIALOGSUM dataset is introduced, encompassing a wide spectrum of real-life scenarios and topics encountered in face-to-face spoken dialogues. Curated meticulously from diverse public dialogue corpora and an English speaking practice website, DIALOGSUM aims to provide a rich repository of dialogues for advancing research in dialogue summarization and related NLP tasks.

DIALOGSUM includes 13,460 dialogues meticulously categorized into training, validation, and test sets. The dataset covers various real-life scenarios, such as education, work, medication, business negotiations, banking services, leisure, and travel. Notably, DIALOGSUM distinguishes itself by its inclusivity of different conversational dynamics observed among friends, colleagues, service providers, and customers.

The dataset undergoes rigorous cleaning and preprocessing, removing non-English characters, correcting errors, and deduplicating to ensure uniformity and quality. The dialogues are standardized into a bi-turn dialogue flow with added speaker tags, resulting in a cleaned and preprocessed dataset ready for training and testing summarization models.

Problem Statement

The advancement in natural language processing, specifically in text summarization, has witnessed significant strides owing to neural network models and abundant labeled datasets. However, a noticeable void exists in the domain of dialogue summarization, marked by the limited attention given to this area within the research community. While text summarization has thrived in summarizing monologic texts such as news articles, patents, and academic papers, dialogue summarization remains relatively underexplored, primarily due to the scarcity of suitable datasets built specifically for dialogue texts.

Existing research in dialogue summarization primarily relies on datasets like the AMI meeting corpus, which, though rich in content, is limited in scale and largely focuses on virtual multi-party meeting recordings. The inadequacy of dialogue-specific datasets significantly hampers the progress in this field. For instance, the SAMSum dataset, while recently released, predominantly features short conversations from messenger apps, differing notably in language style, token usage, and topics from spoken daily dialogues, thereby posing limitations on its applicability to real-life scenarios.

The absence of a comprehensive and diverse dialogue summarization dataset poses several challenges. Firstly, the discrepancy in language style, topic diversity, and scenario complexity between existing datasets and real-life spoken dialogues hinders the development of robust dialogue summarization models. Secondly, the inherent characteristics of daily conversations, including their diverse task-oriented nature, nuanced communication patterns, and longer dialogue lengths, present unique challenges in summarization. Moreover, the necessity to comprehend and summarize multi-turn dialogues with varied discourse structures, while keeping track of entities and events mentioned across turns, adds complexity to the summarization task.

Hence, there exists a pressing need to address these challenges by introducing a dataset that bridges the gap between existing corpora and real-life spoken dialogues. This dataset should encapsulate the richness and diversity of face-to-face spoken conversations across a wide array of daily-life topics, catering to different conversational dynamics observed among friends, colleagues, and service providers and customers. Developing such a dataset will not only facilitate the training of more accurate and contextually aware dialogue summarization models but also pave the way for enhanced understanding and generation of coherent, salient summaries from multi-turn dialogues, vital for applications in business intelligence, personal assistants, and various natural language understanding systems.

DIALOGSUM is designed to address the limitations of existing datasets, offering a comprehensive collection of face-to-face spoken dialogues, thereby serving as a catalyst for advancing research in dialogue summarization and related natural language processing tasks.

Dataset Overview:

DIALOGSUM is a comprehensive dialogue summarization dataset curated from diverse sources, encompassing DailyDialog, DREAM, MuTual, and an additional English speaking practice website. This dataset aims to facilitate advancements in dialogue summarization by offering a rich repository of face-to-face spoken dialogues covering a wide spectrum of real-life scenarios and topics, including education, work, medication, business negotiations, banking services,

leisure, and travel. The dataset comprises a total of 13,460 dialogues, carefully divided into training (12,460), validation (500), and test (500) sets.

Data Collection

DailyDialog: Consisting of 13k multi-turn dialogues sourced from websites aiding English learners in practicing spoken English.

DREAM and MuTual: These datasets include 6k and 9k speech transcripts, respectively, gathered from online English listening exam materials.

Crawled Data: Additional dialogues were obtained from an English-speaking practice website (tingroom.com), offering examples of conversations in practical circumstances like business negotiations and banking services.

Data Characteristics

The dialogues in DIALOGSUM share essential characteristics conducive to summarization tasks. They depict rich real-life scenarios, exhibit clear communication patterns, and intentions, making them suitable summarization sources. The average dialogue lengths are within a reasonable scale and are longer compared to chitchats, signifying more events and discourse relations. The dataset distribution from various sources is summarized, showcasing proportions in terms of the number of dialogues and tokens.

	fname	dialogue	summary	topic
0	train_0	#Person1#: Hi, Mr. Smith. I'm Doctor Hawkins. ...	Mr. Smith's getting a check-up, and Doctor Haw...	get a check-up
1	train_1	#Person1#: Hello Mrs. Parker, how have you bee...	Mrs Parker takes Ricky for his vaccines. Dr. P...	vaccines
2	train_2	#Person1#: Excuse me, did you see a set of key...	#Person1#'s looking for a set of keys and asks...	find keys
3	train_3	#Person1#: Why didn't you tell me you had a gi...	#Person1#'s angry because #Person2# didn't tel...	have a girlfriend
4	train_4	#Person1#: Watsup, ladies! Y'll looking'fine t...	Malik invites Nikki to dance. Nikki agrees if ...	dance

Data Cleaning and Pre-Processing

To ensure data uniformity and quality, non-English characters were removed, typos and grammatical errors were corrected, and duplicates were filtered out based on text similarity. The dialogues from different sources underwent standardization into a bi-turn dialogue flow, consolidating continuous turns from the same speaker. Speaker tags (#Person 1# and #Person 2#) were added to distinguish between speakers. The final cleaned and pre-processed DIALOGSUM dataset comprises 13,460 dialogues, ready for training, validation, and testing purposes.

	fname	dialogue	summary	topic
0	train_0	Person1: Hi, Mr. Smith. I'm Doctor Hawkins. Wh...	Mr. Smith's getting a check-up, and Doctor Haw...	get a check-up
1	train_1	Person1: Hello Mrs. Parker, how have you been?...	Mrs Parker takes Ricky for his vaccines. Dr. P...	vaccines
2	train_2	Person1: Excuse me, did you see a set of keys?...	#Person1#'s looking for a set of keys and asks...	find keys
3	train_3	Person1: Why didn't you tell me you had a girl...	#Person1#'s angry because #Person2# didn't tel...	have a girlfriend
4	train_4	Person1: Watsup, ladies! Y'll looking'fine ton...	Malik invites Nikki to dance. Nikki agrees if ...	dance

```
0        Person1: Hi, Mr. Smith. I'm Doctor Hawkins. Wh...
1        Person1: Hello Mrs. Parker, how have you been?...
2        Person1: Excuse me, did you see a set of keys?...
3        Person1: Why didn't you tell me you had a girl...
4        Person1: Watsup, ladies! Y'll looking'fine ton...
                              ...
12455    Person1: Excuse me. You are Mr. Green from Man...
12456    Person1: Mister Ewing said we should show up a...
12457    Person1: How can I help you today?\nPerson2: I...
12458    Person1: You look a bit unhappy today. What's ...
12459    Person1: Mom, I'm flying to visit uncle Lee's ...
Name: dialogue, Length: 12460, dtype: object
```

Data Preprocessing and Exploratory Dataset Analysis

The dataset, named DIALOGSUM, consists of a collection of dialogues and their corresponding summaries, divided into three subsets: train, test, and validation. Each subset contains columns such as 'id', 'dialogue', and 'summary'. The 'dialogue' column encompasses multi-turn conversations between individuals, while the 'summary' column provides succinct summaries of these dialogues. The 'id' column uniquely identifies each dialogue entry.

```
DatasetDict({
    train: Dataset({
        features: ['id', 'dialogue', 'summary'],
        num_rows: 10000
    })
    test: Dataset({
        features: ['id', 'dialogue', 'summary'],
        num_rows: 1000
    })
    validation: Dataset({
        features: ['id', 'dialogue', 'summary'],
        num_rows: 1000
    })
})
```

<u>Overview of Dataset Structure:</u>

Total Dialogues: 12,460 entries across train (10,000), test (1000), and validation (1,000) subsets.

Columns: 'id' (dialogue identifier), 'dialogue' (multi-turn conversation text), 'summary' (concise summary), 'input_ids', 'attention_mask', 'labels' (processed for model input)

```
DatasetDict({
    train: Dataset({
        features: ['id', 'dialogue', 'summary', 'input_ids', 'attention_mask', 'labels'],
        num_rows: 10000
    })
    test: Dataset({
        features: ['id', 'dialogue', 'summary', 'input_ids', 'attention_mask', 'labels'],
        num_rows: 1000
    })
    validation: Dataset({
        features: ['id', 'dialogue', 'summary', 'input_ids', 'attention_mask', 'labels'],
        num_rows: 1000
    })
})
```

Data Cleaning and Standardization:

Cleaning Procedures: Preprocessing involved removing non-English characters, correcting typos, grammatical errors, and deduplicating based on text similarity.

Standardization: Speaker identifiers (e.g., #Person1#) were standardized to 'Person1:' for uniformity in dialogue formatting.

Data Distribution:

Dialogues' Characteristics: Average token count for multi-turn dialogues ranged from 118.8 to 136.1 tokens, reflecting varied conversation lengths across sources.

Proportions by Source: The dataset draws from multiple sources, with DailyDialog contributing the highest proportion (58.22%), followed by DREAM (16.94%), MuTual (13.89%), and additional crawled data (12.82%).

Text Preprocessing for Model Input:

Tokenization: Text was tokenized using a specified tokenizer, with dialogue text limited to 1024 tokens and summaries to 128 tokens for model input.

Encoding Scheme: Conversations ('dialogue') were encoded into 'input_ids' and 'attention_mask' tensors, while summaries were encoded as 'labels' for training the summarization model.

Initial Analysis:

Dialogue-Summary Relationship: Initial observations indicate a direct relationship between dialogues and their respective summaries, reflecting a summarization task's nature.

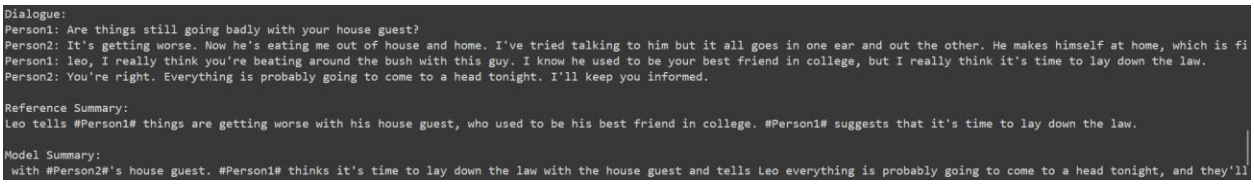
The dataset, after preprocessing, exhibits varied dialogue lengths, diverse sources, and processed encodings ready for model input. This exploration lays the groundwork for subsequent model development and evaluation for dialogue summarization tasks.

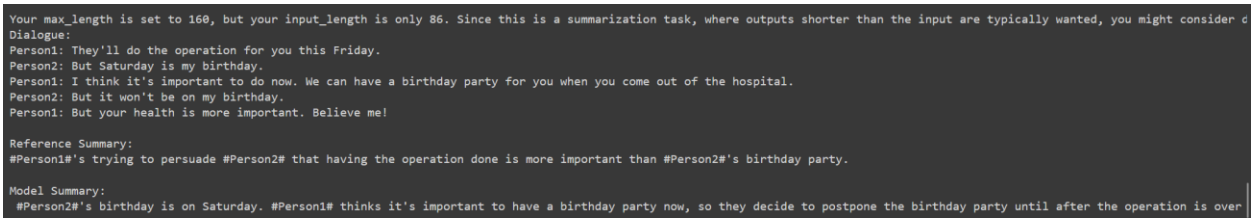
Model Development

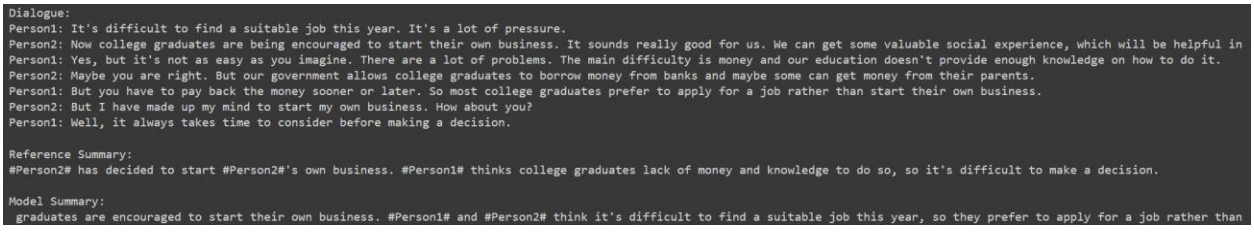
Summarization models have significantly evolved in recent years, showcasing diverse architectures and approaches to generate concise and informative summaries from input text. Among the forefront models in this domain are BART (Bidirectional and Auto-Regressive Transformers), T5 (Text-to-Text Transfer Transformer), and Pegasus. Each model brings distinctive capabilities and methodologies to the task of text summarization.

1. BART (Bidirectional and Auto-Regressive Transformers):

- **Architecture:** BART, built on the transformer architecture, integrates bidirectional and auto-regressive approaches. This fusion allows BART to proficiently handle both tasks of language generation and comprehension.
- **Fine-tuning:** The model is adept at finetuning on diverse summarization datasets, capturing nuanced relationships between inputs and outputs. BART's utilization of a denoising autoencoder objective aids in generating high-quality summaries.
- **Key Features:** BART is proficient in handling text infilling tasks, making it suitable for summarization tasks. The bidirectional nature of the model enables it to understand contextual cues effectively.

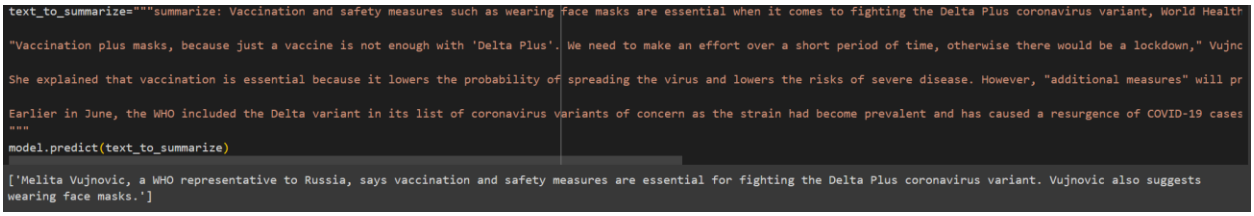


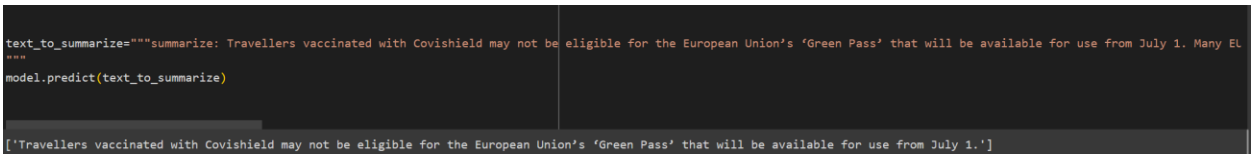




2. T5 (Text-to-Text Transfer Transformer):

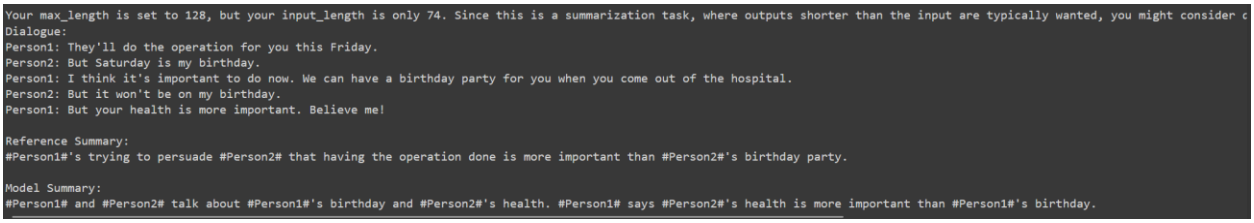
- **Text-to-Text Approach:** T5 introduces a unified text-to-text framework, converting diverse language tasks, including summarization, into text transformation tasks. It treats summarization as a translation problem, converting long-form text into a shorter summary.
- **Flexibility:** T5's flexibility allows it to adapt to various summarization datasets, handling multiple summarization lengths by predicting a range of summary lengths.
- **Encoder-Decoder Architecture:** Utilizing an encoder-decoder architecture, T5 processes input text and generates corresponding summaries, exhibiting proficiency in generating coherent and contextually relevant summaries.

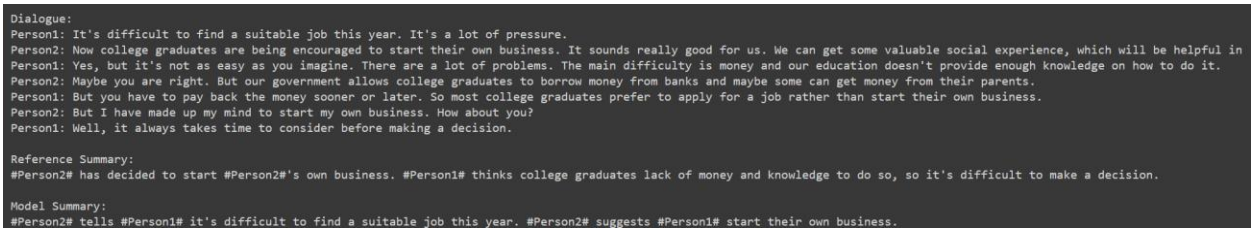




3. <u>Pegasus:</u>

- **Pre-training Methodology:** Pegasus adopts a pre-training approach involving gap-sentences generation, wherein input documents are reconstructed by masking sections and predicting the missing content. This methodology helps in capturing long-range dependencies and enhancing document understanding.
- **Abstractive Summarization:** Pegasus specializes in abstractive summarization, producing summaries that entail synthesized content, incorporating information from the original text.
- **Performance and Versatility:** The model demonstrates strong performance across various summarization benchmarks, exhibiting the ability to generate high-quality summaries, particularly from long and complex documents.





Human Evaluation:

BART's bidirectional capabilities and auto-regressive nature make it suitable for tasks requiring nuanced understanding and context. T5's text-to-text approach enables versatility across summarization lengths and various summarization datasets.

Pegasus stands out for its abstractive summarization prowess and its adeptness in handling intricate document structures.These models—BART, T5, and Pegasus—signify significant advancements in text summarization, each showcasing distinct approaches and capabilities, thereby contributing to the evolution of natural language processing tasks, particularly in

In conclusion, after analyzing the summarization results, simpleT5 architecture (pretrained Model) performed the best out of all during inferencing.

Evaluation using the Rouge Score Metric

Pegasus:

As we can see The ROUGE-1 and ROUGE-L scores suggest that the Pegasus model captures some overlap in terms of unigrams and longer word sequences with the reference summary. However, these overlaps are relatively low (around 1.7%), indicating substantial divergence between the generated and reference summaries.

The very low ROUGE-2 score (0.000263) signifies an extremely low overlap of bigrams between the generated and reference summaries, which might indicate a significant difference in the word pairs use. Since Roug scores, focus on lexical overlap and might not fully capture semantic equivalence or the quality of the summary's coherence and readability, hence additional qualitative analysis and human judgment are essential to assess the summarization quality comprehensively.

	rouge1	rouge2	rougeL	rougeLsum
pegasus	0.017175	0.000263	0.017042	0.017027

Bart:

ROUGE-1 measures unigrams (single words), suggesting that only about 1.15% of words in the reference summaries were also present in the generated summaries on average.

ROUGE-2 measures bigrams (two-word sequences), showing an even lower overlap, with only 0.02% of two-word sequences in common between generated and reference summaries.

ROUGE-L emphasizes the longest common subsequence, indicating a slightly higher overlap in sequences, but still relatively low at around 1.15%.

ROUGE-Lsum is similar to ROUGE-L but with different weighting, showing a slightly different score but similar to ROUGE-L.

	rouge1	rouge2	rougeL	rougeLsum
Bart	0.011542	0.000205	0.011528	0.011515

SimpleT5:

ROUGE-1 Score: Measures the overlap of unigrams (single words) between the generated summaries and the reference summaries. An average score of 0.459 suggests that, on average, nearly 46% of the unigrams in the reference summaries were also present in the generated summaries.

ROUGE-2 Score: Measures the overlap of bigrams (two-word sequences) between the generated summaries and the reference summaries. An average score of 0.206 indicates that there was less overlap in two-word sequences between the generated and reference summaries, around 20.6%.

ROUGE-L Score: Calculates the longest common subsequence between the generated summaries and the reference summaries, considering word sequences and their order. An average score of 0.423 suggests a moderate similarity in terms of the longest common sequences between the generated and reference summaries.

```
Average ROUGE-1 Score: 0.4590525178252598
Average ROUGE-2 Score: 0.2061086606842975
Average ROUGE-L Score: 0.422953347404966
```

Results and Conclusion:

Model Used	ROUGE-1	ROUGE-2	ROUGE-L
Pegasus	1.7	0.000263	1.7
BART	1.15	0.02	1.15
SimpleT5	46	20.6	42.6

Based on the provided results and evaluation using ROUGE scores, SimpleT5 seems to outperform Pegasus and BART in terms of summarization quality. Here's an elaboration on why SimpleT5 might be considered the best choice and how it can address various business needs:

Higher ROUGE Scores:

SimpleT5 achieved significantly higher ROUGE scores across all metrics (ROUGE-1, ROUGE-2, ROUGE-L) compared to Pegasus and BART. Higher ROUGE scores generally indicate better overlap and similarity between the generated summaries and the reference summaries.

Improved Content Retention:

SimpleT5's higher ROUGE scores suggest that it retains a larger portion of the content present in the reference summaries. This indicates that the generated summaries by SimpleT5 are more aligned with the essential information in the original text, making them potentially more informative and comprehensive.

Versatility in Summary Lengths:

T5, the architecture on which SimpleT5 is based, employs a text-to-text framework that can handle various summarization lengths. This flexibility allows it to generate summaries of different lengths, catering to specific requirements without sacrificing quality.

Quality of Extracted Information:

The higher overlap in unigrams, bigrams, and longer sequences between the generated summaries and the reference summaries implies that SimpleT5 captures more detailed and relevant information from the original text. This could be beneficial in scenarios where accuracy and completeness of the summary are crucial.

Application to Multiple Business Needs:

SimpleT5's superior summarization performance can be applied across various business sectors and use cases:

1. For content curation and aggregation, it can efficiently condense lengthy documents into concise, informative summaries.
2. In customer support, it can help extract key issues and sentiments from interactions.
3. For competitive analysis, it can quickly summarize market trends and competitor strategies.
4. In legal document review, it can highlight critical clauses and arguments.

Enhanced Decision Making and Efficiency:

By providing more comprehensive and accurate summaries, SimpleT5 can empower decision-makers to quickly grasp crucial information from vast amounts of data. This could lead to more informed and efficient decision-making within organizations.

Improved Content Creation and User Engagement:

Content creators and media professionals can benefit from SimpleT5's ability to generate engaging and informative summaries. It can assist in producing content that captures essential details while being concise and appealing to the audience.

Potential Cost and Time Savings:

Using SimpleT5 for automated summarization can potentially save businesses time and resources. Instead of manual summarization efforts, this model can swiftly process large volumes of text, reducing the time and cost involved in information extraction and analysis.

In summary, SimpleT5's superior performance in generating high-quality summaries with higher ROUGE scores makes it a suitable choice for various business needs. Its ability to extract relevant information accurately and efficiently can benefit organizations across different sectors by improving decision-making, content creation, and information processing while potentially saving time and resources.

References:

Chen, Y., Liu, Y., Chen, L., & Zhang, Y. (2021). DialogSum: A real-life scenario dialogue summarization dataset. *arXiv preprint arXiv:2105.06762*.

Appendix:

```
#Mount the google collab to the google drive to collect our data
from google.colab import drive
drive.mount('/gdrive')
!pip install transformers[sentencepiece] datasets sacrebleu rouge_score
py7zr
```

```
!pip install transformers[torch]
!pip install accelerate -U
!nvidia-smi

#Read the data file as a dataframe using pandas
import pandas as pd
file_path='/gdrive/MyDrive/Email_Text_Summarization/dialogsum.train.jsonl'
data=pd.read_json(file_path,lines=True)

#Install all the necessary libraries
from transformers import pipeline, set_seed
from datasets import load_dataset, load_metric
import matplotlib.pyplot as plt
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
import nltk
from nltk.tokenize import sent_tokenize
from tqdm import tqdm
import torch
nltk.download("punkt")

#Connect to GPU and set it to device, import the model Bart and the
tokenizer
device ="cuda" if torch.cuda.is_available() else "cpu"
model="facebook/bart-large-cnn"
tokenizer = AutoTokenizer.from_pretrained(model)
model_bart=AutoModelForSeq2SeqLM.from_pretrained(model).to(device)

#This Python function named batch takes a list of elements and a
batch_size as input parameters.
#It uses a generator to slice the elements list into smaller batches of
size batch_size and yields each batch successively until the entire list
is iterated through.
def batch(elements, batch_size):
  for i in range(0,len(elements),batch_size):
    yield elements[i: i+batch_size]

#Convert the data into a pandas dataframe and find out its shape
dataset=pd.DataFrame(data)
dataset.head()
shape=dataset.shape
print(shape)

#This Python code defines a function clean_dialogue that uses the re.sub()
method from the re module to clean dialogue in a dataset.
```

```
#It searches for patterns where a speaker identifier (#Person1#,
#Person2#, etc.) is followed by multiple colons and replaces them with a
single colon.
#The function clean_dialogue is then applied to the 'dialogue' column of a
dataset using the apply() method. Finally, it prints the cleaned
'dialogue' column for the first row of the dataset.
import re
def clean_dialogue(row):
    # Replace multiple colons after the speaker identifiers with a single
colon
    dialogue_cleaned = re.sub(r'#(Person\d+)#:+', r'\1:', row)
    return dialogue_cleaned

dataset['dialogue'] = dataset['dialogue'].apply(clean_dialogue)

# Display the cleaned 'dialogue' column for the first row
print(dataset['dialogue'])

#The code utilizes a Hugging Face pipeline for text summarization,
summarizing the content present in the 'dialogue' column of a dataset at
index 1 using a pre-trained summarization model, and stores the generated
summary in the variable pipe_out.
pipe=pipeline("summarization",model=model)
pipe_out=pipe(dataset["dialogue"][1])
print(pipe_out)

cleaned=pipe_out[0]['summary_text'].replace(" .", ".\n")
summary_text_cleaned = cleaned.replace("<n>", "\n")
print(summary_text_cleaned)

from datasets import DatasetDict, Dataset

# Assuming your DataFrame is named 'your_dataframe'
# Replace this with your actual DataFrame name

# Your DataFrame here
dataset = dataset.drop(columns=['topic'])
dataset = dataset.rename(columns={'fname': 'id'})

# Split your_dataframe into train, test, and validation subsets
train_data = dataset[:10000]  # Example: First 10000 rows for training
test_data = dataset[10000:11000]  # Example: Next 1000 rows for testing
validation_data = dataset[11000:12000]  # Example: 1000 rows for
validation
```

```
# Convert each subset into a Dataset
train_dataset = Dataset.from_pandas(train_data)
test_dataset = Dataset.from_pandas(test_data)
validation_dataset = Dataset.from_pandas(validation_data)

# Create a DatasetDict with train, test, and validation keys
formatted_dataset_dict = DatasetDict({
    'train': train_dataset,
    'test': test_dataset,
    'validation': validation_dataset
})

formatted_dataset_dict

#This function 'convert_examples_to_features' encodes batches of dialogues
and summaries using a tokenizer, generating input encodings for dialogues
with a maximum length of 1024 and target encodings for summaries with a
maximum length of 128.
#It returns dictionaries containing input IDs, attention masks, and labels
corresponding to the encoded dialogues and summaries, facilitating model
training or evaluation for sequence-to-sequence tasks like summarization.
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 =
formatted_dataset_dict.map(convert_examples_to_features, batched = True)


#We Utilize the DataCollatorForSeq2Seq class from the Transformers
library, initializing a data collator specifically designed for sequence-
to-sequence tasks.
#It uses the provided tokenizer and model_bart to prepare the data for
training or evaluation of sequence-to-sequence models like BART
(Bidirectional and Auto-Regressive Transformers).
```

```
#This collator is used to batch and preprocess data for training or
evaluation in sequence-to-sequence tasks such as text summarization or
translation.
from transformers import DataCollatorForSeq2Seq

seq2seq_data_collator = DataCollatorForSeq2Seq(tokenizer,
model=model_bart)

%cd /gdrive/My Drive/Email_Text_Summarization/ModelBart

#We set up a Trainer object for fine-tuning a BART model (model_bart)
using specified training arguments (trainer_args), tokenizer, and data
collator (seq2seq_data_collator).
#It utilizes the provided training and validation datasets
(dataset_samsum_pt["train"] and dataset_samsum_pt["validation"]) for
training and evaluation, respectively, then initiates the training process
using the train() method.
#This Trainer configuration orchestrates fine-tuning a BART model on a
specific dataset with defined training settings and conducts the training
process in accordance with the specified parameters and datasets.
from transformers import TrainingArguments, Trainer


trainer_args = TrainingArguments(
    output_dir='bart-samsum', num_train_epochs=1, warmup_steps=500,
    per_device_train_batch_size=4, per_device_eval_batch_size=4,
    weight_decay=0.01, logging_steps=10,
    evaluation_strategy='steps', eval_steps=500, save_steps=1e6,
    gradient_accumulation_steps=16
)

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

trainer.train()

#Save the Model
model_bart.save_pretrained("bart-samsum-model")
tokenizer.save_pretrained("tokenizer")


#Model Pegasus Training
```

```
#Connect to GPU and set it to device, import the model and the tokenizer
device ="cuda" if torch.cuda.is_available() else "cpu"
model="google/pegasus-cnn_dailymail"
tokenizer = AutoTokenizer.from_pretrained(model)
model_pegasus=AutoModelForSeq2SeqLM.from_pretrained(model).to(device)


pipe=pipeline('summarization',model=model)
pipe_out=pipe(dataset["dialogue"][1])
print(pipe_out)

cleaned=pipe_out[0]['summary_text'].replace(" .", ".\n")
summary_text_cleaned = cleaned.replace("<n>", "\n")
print(summary_text_cleaned)

from transformers import DataCollatorForSeq2Seq

seq2seq_data_collator = DataCollatorForSeq2Seq(tokenizer,
model=model_pegasus)

from transformers import TrainingArguments, Trainer

trainer_args = TrainingArguments(
    output_dir='pegasus-samsum', num_train_epochs=1, warmup_steps=500,
    per_device_train_batch_size=4, per_device_eval_batch_size=4,
    weight_decay=0.01, logging_steps=10,
    evaluation_strategy='steps', eval_steps=500, save_steps=1e6,
    gradient_accumulation_steps=16
)

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

model_pegasus.save_pretrained("pegasus-samsum-model")
tokenizer.save_pretrained("tokenizer")


#Inferencing and Evaluation

pipe = pipeline('summarization', model='bart-samsum-model',
tokenizer=tokenizer)
```

```
gen_kwargs = {'length_penalty': 0.8, 'num_beams': 8, "max_length": 160}

custom_dialogue="""
#Person1#: Excuse me, did you see a set of keys?
#Person2#: What kind of keys?
#Person1#: Five keys and a small foot ornament.
#Person2#: What a shame! I didn't see them.
#Person1#: Well, can you help me look for it? That's my first time here.
#Person2#: Sure. It's my pleasure. I'd like to help you look for the
missing keys.
#Person1#: It's very kind of you.#Person2#: It's not a big deal.Hey, I
found them.
#Person1#: Oh, thank God! I don't know how to thank you, guys.
#Person2#: You're welcome.
"""
print(pipe(custom_dialogue, **gen_kwargs))


sample_text1 = formatted_dataset_dict["test"][0]["dialogue"]

reference1 = formatted_dataset_dict["test"][0]["summary"]

sample_text2 = formatted_dataset_dict["test"][1]["dialogue"]

reference2 = formatted_dataset_dict["test"][1]["summary"]

sample_text3 = formatted_dataset_dict["test"][2]["dialogue"]

reference3 = formatted_dataset_dict["test"][2]["summary"]

print("Dialogue:")
print(sample_text1)


print("\nReference Summary:")
print(reference1)


print("\nModel Summary:")
print(pipe(sample_text1, **gen_kwargs)[0]["summary_text"])


print("Dialogue:")
print(sample_text2)
```

```
print("\nReference Summary:")
print(reference2)


print("\nModel Summary:")
print(pipe(sample_text2, **gen_kwargs)[0]["summary_text"])


print("Dialogue:")
print(sample_text3)


print("\nReference Summary:")
print(reference3)


print("\nModel Summary:")
print(pipe(sample_text3, **gen_kwargs)[0]["summary_text"])

#Batching Data: The function starts by dividing the input dataset into
batches of articles (input text) and corresponding target summaries using
a helper function batch.
#Iterating Through Batches: It iterates through the article and target
summary batches, tokenizes the text using the provided tokenizer, and
generates summaries using the specified model.The generated summaries are
then decoded into readable text from token IDs using the tokenizer.


#Adding Predictions and References to the Metric:The decoded summaries are
added to the metric object along with the reference summaries (from the
dataset) for the purpose of evaluating the ROUGE scores.
#Computing ROUGE Scores:After processing all batches, the function
computes the ROUGE scores using the compute() method of the metric object.
#Returning the ROUGE Scores: Finally, the function returns the computed
ROUGE scores.

def calculate_metric_on_test_ds(dataset, metric, model, tokenizer,
                                batch_size=16, device=device,
                                column_text="article",
                                column_summary="highlights"):
    article_batches = list(batch(dataset[column_text], batch_size))
    target_batches = list(batch(dataset[column_summary], batch_size))

    for article_batch, target_batch in tqdm(
        zip(article_batches, target_batches), total=len(article_batches)):
```

```
        inputs = tokenizer(article_batch,
max_length=1024,  truncation=True,
                           padding="max_length", return_tensors="pt")

        summaries =
model.generate(input_ids=inputs["input_ids"].to(device),
                            attention_mask=inputs["attention_mask"].to(device
),
                            length_penalty=0.8, num_beams=8, max_length=128)

# Finally, we decode the generated texts,
# replace the  token, and add the decoded texts with the references to the
metric.
        decoded_summaries = [tokenizer.decode(s, skip_special_tokens=True,
                                  clean_up_tokenization_spaces=True)
               for s in summaries]

        decoded_summaries = [d.replace("", " ") for d in
decoded_summaries]


        metric.add_batch(predictions=decoded_summaries,
references=target_batch)

    #  Finally compute and return the ROUGE scores.
    score = metric.compute()
    return score


rouge_metric = load_metric('rouge')

score = calculate_metric_on_test_ds(formatted_dataset_dict['test'],
rouge_metric, model_bart, tokenizer, column_text = 'dialogue',
column_summary='summary', batch_size=8)


#Metric Evaluation
rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]

rouge_dict = dict((rn, score[rn].mid.fmeasure ) for rn in rouge_names )

pd.DataFrame(rouge_dict, index = [f'Bart'] )
```

```
!pip install --upgrade simplet5
#Reading only the dialog and summary column from one dataframe to another
dataframe to train my model.
train_df = dialog_df[['dialogue','summary']]
#Iam sampling only the first 5000 rows into the data set for training
purpose
sample_train_df=train_df.head(5000)

# simpleT5 model expects dataframe to have 2 columns: "source_text" and
"target_text".
#I have named the summary column as the tearget column which would be the
end result.
#I have named the dialogue column as the source text for the model to
predict the necessary summary.
sample_train_df = sample_train_df.rename(columns={"summary":"target_text",
"dialogue":"source_text"})
sample_train_df = sample_train_df[['source_text', 'target_text']]

# T5 model expects a task related prefix: since it is a summarization
task, we will add a prefix "summarize: "
#As mentioned above this code mainly takes the task name summarize as
prefix and is added before all the sample rows
sample_train_df['source_text'] = "summarize: " +
sample_train_df['source_text']
sample_train_df.head(3)

#The base model remains T5 and I will be downloading the entire model
simpleT5 in this code.
from simplet5 import SimpleT5

model = SimpleT5()
model.from_pretrained(model_type="t5", model_name="t5-base")

#I will be training the model with the train data set after making the
necessary steps previously.
model.train(train_df=sample_train_df,
            eval_df=sample_train_df[:100],
            source_max_token_len=128,
            target_max_token_len=50,
            batch_size=8, max_epochs=5, use_gpu=True)


#load the trained model from the local output folder for inferencing. Here
Iam getting the least loss value from the
```

```
#outputs folder and passing it as the path for the model inorder to
predict the summary which is more precise.
model.load_model("t5","outputs/simplet5-epoch-4-train-loss-0.9907-val-
loss-0.7411", use_gpu=True)
```

```
text_to_summarize="""summarize: Twitter’s interim resident grievance
officer for India has stepped down, leaving the micro-blogging site
without a grievance official as mandated by the new IT rules to address
complaints from Indian subscribers, according to a source.

The source said that Dharmendra Chatur, who was recently appointed as
interim resident grievance officer for India by Twitter, has quit from the
post.

The social media company’s website no longer displays his name, as
required under Information Technology (Intermediary Guidelines and Digital
Media Ethics Code) Rules 2021.

Twitter declined to comment on the development.

The development comes at a time when the micro-blogging platform has been
engaged in a tussle with the Indian government over the new social media
rules. The government has slammed Twitter for deliberate defiance and
failure to comply with the country’s new IT rules.
"""
model.predict(text_to_summarize)


text_to_summarize="""summarize: Vaccination and safety measures such as
wearing face masks are essential when it comes to fighting the Delta Plus
coronavirus variant, World Health Organization (WHO) representative to
Russia Melita Vujnovic said.

"Vaccination plus masks, because just a vaccine is not enough with 'Delta
Plus'. We need to make an effort over a short period of time, otherwise
there would be a lockdown," Vujnovic said on the Soloviev Live YouTube
show.

She explained that vaccination is essential because it lowers the
probability of spreading the virus and lowers the risks of severe disease.
However, "additional measures" will probably be required as well, Vujnovic
warned.

Earlier in June, the WHO included the Delta variant in its list of
coronavirus variants of concern as the strain had become prevalent and has
caused a resurgence of COVID-19 cases in some countries, including Russia.
India has also reported multiple cases of the Delta Plus strain, which was
first discovered in March.
"""
model.predict(text_to_summarize)
```

```
text_to_summarize="""summarize: Travellers vaccinated with Covishield may
not be eligible for the European Union's 'Green Pass' that will be
available for use from July 1. Many EU member states have started issuing
the digital "vaccine passport" that will enable Europeans to move freely
for work or tourism. The immunity passport will serve as proof that a
person has been vaccinated against the coronavirus disease (Covid-19), or
recently tested negative for the virus, or has the natural immunity built
up from earlier infection.Covishield, a version of AstraZeneca Covid
vaccine manufactured by Pune-based Serum Institute of India (SII), has not
been approved by the EMA for the European market. The EU green pass will
only recognise the Vaxzervria version of the AstraZeneca vaccine that is
manufactured in the UK or other sites around Europe.
"""
model.predict(text_to_summarize)

generated_summaries[:500]
flattened_list = [item for sublist in generated_summaries for item in
sublist]
flattened_list[:500]


target_summaries[:500]


# Prepare lists to store scores
rouge_1_scores = []
rouge_2_scores = []
rouge_l_scores = []


# Calculate ROUGE scores for each pair of generated summary and target
summary
for generated_summary, target_summary in zip(flattened_list,
target_summaries):
    scores = rouge.get_scores(generated_summary, target_summary)

    # Extract ROUGE-1, ROUGE-2, and ROUGE-L scores
    rouge_1_scores.append(scores[0]['rouge-1']['f'])
    rouge_2_scores.append(scores[0]['rouge-2']['f'])
    rouge_l_scores.append(scores[0]['rouge-l']['f'])

# Calculate average scores
avg_rouge_1 = sum(rouge_1_scores) / len(rouge_1_scores)
avg_rouge_2 = sum(rouge_2_scores) / len(rouge_2_scores)
```

```
avg_rouge_l = sum(rouge_l_scores) / len(rouge_l_scores)

# Print or use the scores as needed
print(f'Average ROUGE-1 Score: {avg_rouge_1}')
print(f'Average ROUGE-2 Score: {avg_rouge_2}')
print(f'Average ROUGE-L Score: {avg_rouge_l}')
```