Enhancing the Accuracy of Abstractive and Extractive Summarization of Patient Discharge Reports Using Transfer Models

Gaurav Gupta
x22212311@student.ncirl.ie
National College of Ireland

Abstract

The primary goal of this research will be to improve the efficacy of text-summarizing patient discharge reports by applying extractive and abstractive approaches. Therefore, to compare the results of the modern transfer learning models like T5, DistilBART, PEGASUS, BERTSum, and XLNet, and evaluate by using ROUGE and BLEU scores.

1 System Requirements

- **Operating System:** Windows, Mac, or Linux.
- **Processor :** Intel Core i5 8th Gen
- **RAM:** At least 8GB (16GB preferred).
- **Disk Space:** Free space that is retrievable and easily accessible, and this should be, at least, 5GB.
- **Internet Connection:** During downloading of pre-trained models and datasets.

2 Software Requirements

- **Python Version:** Python 3.8 or higher.
- **Jupyter Notebook or Google collab**.

3 Installation Guide

3.1 Python Installation and Version

Install python and check its version and it should be higher than 3.8.

```
!python --version
```

3.2 Install Required Packages

To install necessary packages, use pip to download and install them. Example command:

```
pip install torch transformers pandas nltk scikit-learn
sentence-transformers rouge-score
```

3.3 Install Required Libraries

Other related libraries can be installed using:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import nltk
from collections import Counter
from nltk.tokenize import word_tokenize
import re
from nltk.corpus import stopwords
from transformers import PegasusForConditionalGeneration, PegasusTokenizer
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from transformers import LEDTokenizer, LEDForConditionalGeneration
import torch
from transformers import XLNetTokenizer, XLNetModel
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer
from sklearn.manifold import TSNE
from rouge_score import rouge_scorer
from nltk.translate import bleu_score
import plotly.express as px
```

Figure 1: Import Essential Libraries

4 Data Preparation

4.1 Dataset Loading

The dataset, named notevents.csv, which should then be brought into Google Colab or Jupyter Notebook. Figure 2 that will help you load the dataset and possibly address errors at the same time: Specify the map of the interesting columns for analysis, for instance 'TEXT' and 'ROW ID'. Figure 3 shows how the loaded dataset look like.

```
# Define the columns you want to read
columns_to_read = ['ROW_ID', 'SUBJECT_ID', 'HADM_ID', 'CHARTDATE',
                   'CHARTTIME', 'STORETIME', 'CATEGORY', 'DESCRIPTION',
                   'CGID', 'ISERROR','TEXT']

# Read the CSV file with specified columns and handle bad lines
try:
    data = pd.read_csv('NOTEEVENTS.csv.gz',
                       usecols=columns_to_read,
                       nrows=100,  # Read only the first 100 rows for quick inspection
                       compression='gzip',
                       on_bad_lines='skip')  # Skip lines with too many/few fields
    print(data.head())  # Display the first few rows of the DataFrame
except pd.errors.ParserError as e:
    print("A parsing error occurred:", e)
```

Figure 2: Data Loading

```
data
```

	ROW_ID	SUBJECT_ID	HADM_ID	CHARTDATE	CHARTTIME	STORETIME	CATEGORY	DESCRIPTION	CGID	ISERROR	TEXT
0	174	22532	167853	2151-08-04	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2151-7-16**] Dischar...
1	175	13702	107527	2118-06-14	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2118-6-2**] Discharg...
2	176	13702	167118	2119-05-25	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2119-5-4**] D...
3	177	13702	196489	2124-08-18	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2124-7-21**] ...
4	178	26880	135453	2162-03-25	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2162-3-3**] D...
...	...	...	...	...	...	...	...	...	...	...	...
95	20	15472	148372	2178-12-02	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2178-11-15**] ...
96	21	15472	188190	2178-12-09	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2178-12-5**] ...
97	22	15472	104146	2179-02-08	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2179-2-1**] D...
98	23	15472	143651	2179-03-26	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2179-3-21**] ...
99	24	15472	184486	2179-04-15	NaN	NaN	Discharge summary	Report	NaN	NaN	Admission Date: [**2179-4-12**] ...

100 rows × 11 columns

Figure 3: Dataset

4.2 Exploratory Data Analysis (EDA)

Show the head of the given dataset to get an idea of the data format. One should always look for any instances of NA values and how they should be dealt with when distributing. General probability characteristics of text lengths' distribution. Compose a word frequency diagram to show the most used words. Generate frequency distribution of the words and the most frequent words are identified.

```python
print(data.head())

# Summary statistics for the 'TEXT' column
print("\nSummary Statistics for 'TEXT' column:")
print(data['TEXT'].describe())

# Check for missing values in 'TEXT'
missing_values = data['TEXT'].isnull().sum()
print(f"\nMissing values in 'TEXT' column: {missing_values}")

# Distribution of text length
data['TEXT_LENGTH'] = data['TEXT'].apply(len)
plt.figure(figsize=(12, 6))
sns.histplot(data['TEXT_LENGTH'], bins=50, kde=True)
plt.title('Distribution of Text Length')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.show()

# Display a Word Cloud of the most frequent words
text_data = ' '.join(data['TEXT'].dropna())
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text_data)

plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of the TEXT Column')
plt.show()

# The most common words and their counts
tokens = word_tokenize(text_data.lower())
common_words = Counter(tokens).most_common(20)
print("\nMost Common Words:")
for word, count in common_words:
    print(f"{word}: {count}")
```

Figure 4: Exploratory Data Analysis

4.3 Data Preprocessing

Pre-process the textual data using the suitable tokenizers. Stop words must be eliminated to concentrate on the core language terms. Convert the given text to a matrix that the models will use as input as in Figure 5.

```python
# Handle missing values in 'TEXT'
data['TEXT'] = data['TEXT'].fillna('')  # Fill missing text with an empty string

# Define a function for text preprocessing
def preprocess_text(text):
    # Remove non-alphabetic characters
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    # Reconstruct text
    return ' '.join(tokens)

# Apply text preprocessing
data['PROCESSED_TEXT'] = data['TEXT'].apply(preprocess_text)

# Display the first few rows of the transformed data
print("\nTransformed Data:")
print(data.head())
```

Figure 5: Data Preprocessing

admission date discharge date service addendum radiologic studies radiologic studies also included chest ct con
admission date discharge date date birth sex f service history present illness patient yearold female complex mec
admission date discharge date service icu history present illness patient yearold female admitted mental status c
admission date discharge date service ccu addendum discharge medications enalapril po bid lasix po qd digoxin
admission date discharge date date birth sex f service addendum neurological patient mri eeg evaluate neurologi
admission date death date service medicinedoctor last name history present illness patient yearold male history e

Figure 6: Dataset after Pre-Processing

4.4 Data Preparation

In this step, we filter the dataset based on specific ROW IDs that were randomly selected for analysis as we have to create their summaries manually and is not possible to do for all the data points so we choose randomly 236 points. The following code snippet demonstrates how to filter the dataset using these ROW IDs. Splitting the dataset into 80:20 ratio.

```
# Filter the DataFrame based on ROW_IDs
row_ids_to_filter = [
    174, 245, 189, 209, 93, 105, 165, 528, 348, 712, 454, 524, 274, 278, 354, 355, 383, 421, 663, 928,
    780, 789, 858, 611, 1172, 1377, 1434, 1217, 1221, 1000, 1102, 1682, 1819, 1829, 1610, 1541, 1280,
    1294, 1923, 1942, 2203, 2334, 1751, 1773, 1794, 2149, 2155, 1986, 1999, 2454, 2461, 2480, 2232,
    2233, 2345, 3003, 3004, 2937, 2656, 2685, 2701, 2894, 2533, 2534, 2535, 2568, 3311, 2739, 3142,
    3711, 3717, 3722, 3630, 3633, 3349, 3416, 4129, 3759, 3959, 4236, 4387, 4562, 4451, 4480, 3923,
    3934, 4853, 4873, 4610, 4566, 4717, 4882, 4808, 4939, 5303, 5108, 4695, 5007, 5613, 5333, 5319,
    5322, 5476, 5645, 5610, 9900, 9933, 7868, 7860, 8733, 9170, 9177, 8398, 8687, 13259, 7124, 10536,
    11548, 10357, 10364, 10911, 5846, 12278, 12510, 10942, 6357, 7284, 6725, 12534, 13267, 13274, 7292,
    7400, 7409, 13275, 11553, 11563, 10372, 10566, 5898, 10648, 10672, 11570, 6602, 11904, 10269, 5727,
    6405, 7313, 12166, 10882, 10886, 10893, 6451, 7169, 10578, 13147, 6257, 6277, 10403, 5920, 5930,
    11520, 6656, 12190, 12191, 10304, 10943, 6461, 12452, 10949, 10951, 10952, 6289, 6296, 7228, 10579,
    10581, 13208, 13216, 13225, 10452, 6002, 11527, 11529, 12204, 10322, 5818, 6690, 12223, 10955, 10956,
    6513, 12506, 6336, 6351, 7253, 9625, 9724, 9831, 6158, 8037, 7741, 8350, 8609, 7501, 9007, 9352,
    8312, 7503, 10855, 8598, 9764, 9863, 10260, 9780, 7822, 8358, 6930, 6950, 8346, 8383, 8825, 9430,
    7559, 7567, 8662, 9209, 9234, 8217, 10673, 10015, 10022, 8131, 9081, 8476, 6066
]

data = df[df['ROW_ID'].isin(row_ids_to_filter)]
```

Figure 7: Randomly selected Data ponits

5 Modelling

Most advanced transfer learning approaches were used for the summarization of patient discharge reports in this research. Specifications of the model such as the PEGASUS model for producing the summaries of the text data were integrated with error handling mechanisms for optimality. In the same manner, T5 was used for summarization the merits of which were always preserved systematically for future use. The BERTSUM model, which is acknowledged for its vibrant performance while solving extractive summarization tasks, was also useful to implement proficient summary generation on the text data. Moreover, DistilBART for generation of summary in a smaller size and quicker was also applied. Finally, the same text data was used with the XLNet model to know about its potential and its approach towards handling and summarizing the given data and comparison was also made between different models. The outputs of every model were then saved and preprocessed for further analysis in terms of the standard performance measures such as ROUGE and BLEU. Figure 8 shows the PEGASUS model for the purpose of text summarization. It splits the input text into tokens, applies a summary using the model, and then translate the summary back into natural language. It manages exceptions and provides "No text available" if text preprocessing has failed.

```
model_name = "google/pegasus-xsum"
tokenizer = PegasusTokenizer.from_pretrained(model_name)
model = PegasusForConditionalGeneration.from_pretrained(model_name)

# Function to summarize text using Pegasus model
def summarize_text(text):
    try:
        if pd.notna(text):
            # Tokenize the input text
            inputs = tokenizer(text, max_length=1024, return_tensors='pt',
truncation=True)
            # Generate summary ids
            summary_ids = model.generate(
                inputs['input_ids'],
                max_length=150,
                min_length=40,
                length_penalty=2.0,
                num_beams=4,
                early_stopping=True
            )
            # Decode the summary ids to text
            summary_text = tokenizer.decode(summary_ids[0],
skip_special_tokens=True)
            return summary_text
        else:
            return "No text available"
    except Exception as e:
        return f"Error during summarization: {e}"
```

Figure 8: Pegasus Function and Model

ABDOMINAL CT: Head CT showed no intracranial hemorrhage or mass effect. a chest CT confirmed cavitary
the patient is a 70-year-old female with a complex medical history. she was admitted after a cardiac arrest on
the patient is an 84 year-old woman admitted with inflammatory bowel disease. she was admitted with a histor
the patient should have potassium followed in a couple of days and monitored closely and her potassium dose
the patient had an MRI and EEG to evaluate neurologic status. the MRI showed diffuse encephalopathy and t
the patient is a 78-year-old male with a history of encephalitis, oral cancer. the patient had shortness of breath

Figure 9: Text After Applying Pegasus Model

6 Evaluation

6.1 ROUGE Score

The quality of the generated summaries is evaluated by the help of ROUGE indices, comparing the summaries with the reference texts. ROUGE-N assesses matching n-grams, for example, unigrams or bigrams of the generated and reference summaries but at the n-th level, whereas ROUGE-L compares the longest continuable match that will tell the coherence and fluency of the summaries. These metrics involve coming up with precision, recall and F1 score, which gives a quantitative measure as to how the generated summaries are able to capture important information from the reference summaries. These scores are useful in the assessment of the various models for summarization and for comparisons to be made.

```
# Function to calculate ROUGE scores
def calculate_rouge(reference, hypothesis):
    scores = rouge_scorer.score(reference, hypothesis)
    return {
        'ROUGE-1': scores['rouge1'].fmeasure,
        'ROUGE-2': scores['rouge2'].fmeasure,
        'ROUGE-L': scores['rougeL'].fmeasure
    }

# Function to apply ROUGE score calculation
def apply_rouge_scores(row):
    return calculate_rouge(row['Manual_Summary'], row['Pegasus_Summary'])

# Calculate ROUGE scores row-wise
combined_df['ROUGE'] = combined_df.apply(apply_rouge_scores, axis=1)
# Calculate average ROUGE scores
avg_rouge = combined_df['ROUGE'].apply(pd.Series).mean()*a

print("Average ROUGE Scores:")
print(avg_rouge)
```

Figure 10: ROUGE Score

Model	ROUGE-1	ROUGE-2	ROUGE-L
XLNet	0.614	0.519	0.570
DistilBART	0.671	0.418	0.608
BERTSUM	0.599	0.267	0.497
T5	0.618	0.356	0.540
PEGASUS	0.608	0.169	0.495

Table 1: Average ROUGE Scores for Different Models

6.2 BLEU Score

The degree of the generality of the summaries is then assessed using the BLEU (Bilingual Evaluation Understudy) scores whereby it calculates the resemblance between the generated summaries and the reference summaries. BLEU measures the number of matching n-grams (for example, unigrams, bigrams) in the generated text in relation to reference summaries; it also takes the problem of the precision and makes use of brevity penalty in the cases of the short summaries. This metric comes up with a score that is numerical in nature, hence showing the level of proximity of the generated summaries with the reference summaries with regard to both content and word choice.

```
# Function to apply BLEU score calculation
def apply_bleu_scores(row):
    return calculate_bleu(row['Manual_Summary'], row['Pegasus_Summary'])

# Calculate BLEU scores row-wise
combined_df['BLEU'] = combined_df.apply(apply_bleu_scores, axis=1)

# Calculate average BLEU score
avg_bleu = combined_df['BLEU'].mean()*b

print("Average BLEU Score:")
print(avg_bleu)
```

Figure 11: BLEU Score

Model	BLEU
XLNet	0.634
DistilBART	0.600
BERTSUM	0.606
T5	0.628
PEGASUS	0.621

Table 2: BLEU Scores for Different Models

6.3 Summary Length Distribution

It helps to check the distribution of the summary's length and thus be certain these correspond to minimum and maximum values prescribed.

```python
# 1. Summary Length Distribution
combined_df_t5['Length'] = combined_df_t5['T5_Summary'].apply(len)
plt.figure(figsize=(8, 6))
plt.hist(combined_df_t5['Length'], bins=10, color='lightblue')
plt.title('Summary Length Distribution for T5')
plt.xlabel('Summary Length')
plt.ylabel('Frequency')
plt.show()
```

Figure 12: Summary Length Distribution

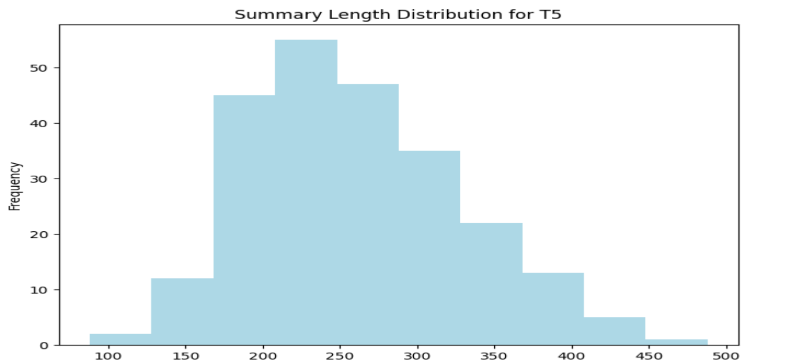

Figure 13: Histogram of T5 Model

6.4 Word Frequency Chart

Visualize word frequencies in generated summaries to understand the content focus.

```python
text_t5 = " ".join(combined_df_t5['T5_Summary'].tolist())
wordcloud_t5 = WordCloud(width=800, height=400,
background_color='white').generate(text_t5)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud_t5, interpolation='bilinear')
plt.axis('off')
plt.title('Word Frequency Visualization for T5')
plt.show()
```

Figure 14: Word Frequency Chart

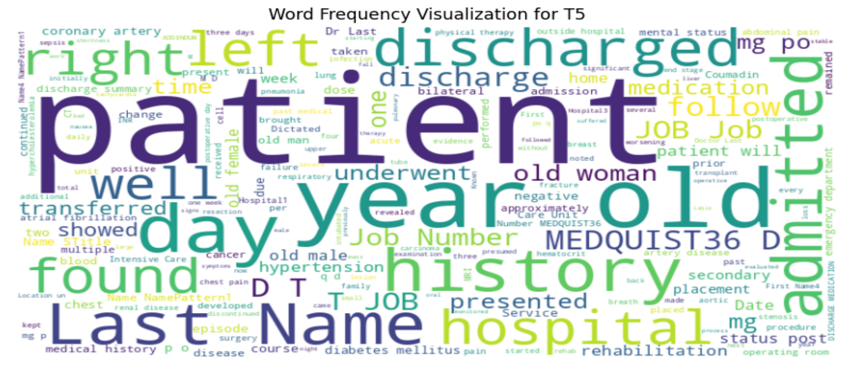

Figure 15: Word Frequency Chart of T5

7 Execution of the Code

1. Download the Dataset
2. Unzip the Files into a Folder
3. Open the Python File in Google Colab or Jupyter Notebook
4. Change the Path to Refer to the Dataset Location
5. Run the Code

8 Conclusion

This project demonstrates that with the help of more enhanced NLP models, it is possible to enhance the quality and speed of the summery of the reports on patients' discharge. In this case, I want to present valuable insights and methodologies for applying the text summing technologies in targeted healthcare environments, thus, by developing advanced transfer models and evaluating their performance in regard to the comprehensive metrics.

9 References

References

Hugging Face, 2024. Transformer Models Documentation. Available at: https://huggingface.co/docs/transformers/.

Google Research, 2024. ROUGE Metrics. Available at: https://github.com/google-research/google-research/tree/master/rouge.

NLTK, 2024. BLEU Score Explanation. Available at: https://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleu.