National College of Ireland

Abstractive Summarization of Multi-Documents using Fairseq

MSc Research Project
Data Analytics

Akash Senthil Kumar
Student ID: x21175641

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Akash Senthil Kumar		
Student ID:	X21175641		
Programme:	MSc Data Analytics	**Year:**	2022-23
Module:	MSc Research Project		
Lecturer:	Mr. Hicham Rifai		
Submission Due Date:	14/08/2023		
Project Title:	Abstractive Summarization of Muti Documents using Fairseq		
Word Count:	632		
Page Count:	7		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Akash Senthil Kumar
Date:	14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Abstractive Summarization of Multi Documents using Fairseq

Akash Senthil Kumar
x21175641

1 Introduction

This configuration manual is about how to perform ‘Multi Document summarization using fairseq’ and this document will provide step by step method to implement from data combining to evaluation of the model.

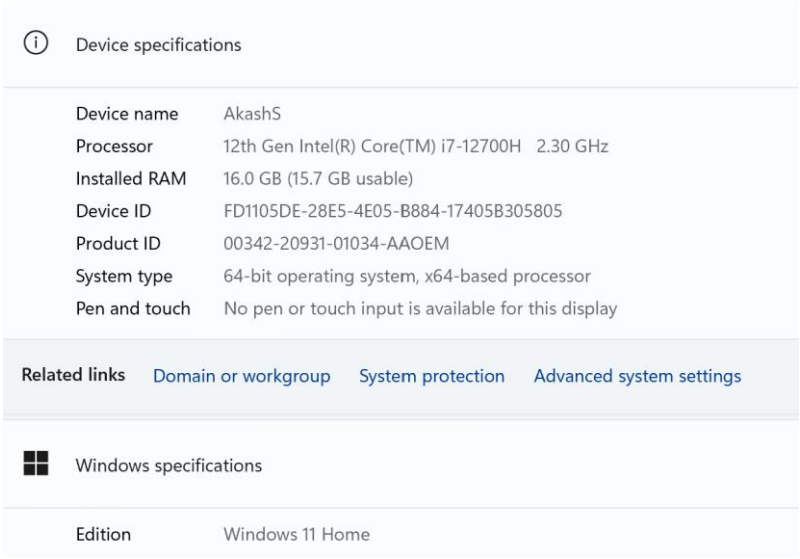

Figure 1 System Specification

The above are system specifications in which this project has been built from scratch.

Software Used:

- Python
- Jupyter Notebook
- Microsoft Excel

These are software that has been used in this research. Python Language has been used to create the deep learning model and to generate the summary. To execute the code Jupyter Notebook has been used from the Anaconda package. Any python version above 7 will be able to execute the code without any issues.

```
import os
import matplotlib.pyplot as plt
import pandas as pd
from fairseq.data import Dictionary
from fairseq import options, tasks, utils, checkpoint_utils
from fairseq.trainer import Trainer
import torch
import torch.nn as nn
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import torch
import torch.nn as nn
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
from transformers import BartTokenizer, BartForConditionalGeneration
```

Figure 2 Necessary Packages

```
import pandas as pd
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
!pip install annoy
from annoy import AnnoyIndex
from datasketch import MinHash, MinHashLSHEnsemble
from sklearn.cluster import KMeans
```

Figure 3 Necessary Packages 2

The above is the necessary packages that has to be imported before starting the project.

2 Data Combining

The first step of this project is merge the data of src content which is document and tgt content which is target i.e summaries. The data has been combined for training data where two dataframe ‘src_content’ and ‘tgt_content’ are combined into one dataframe by creating a dummy column in both dataframes and further joining them based on the dummy column. There are about four jupyter notebook files they are train data claning, test data cleaning, model building and also one failed experiment for which a separate jupyter notebook has been attached.

```
!pip install jsonlines
import pandas as pd
```

```
equirement already satisfied: jsonlines in c:\users\senth\anaconda3\lib\site-packages (3.1.0)
equirement already satisfied: attrs>=19.2.0 in c:\users\senth\anaconda3\lib\site-packages (from jsonlines) (21.4.0)
```

```
with open('train.src', 'r', encoding='utf') as src_file:
    data = src_file.readlines()
```

```

df = pd.DataFrame(data, columns=['src_content'])
```

```
df.head()
```

Figure 4 Data Reading

```
with open('train.tgt', 'r', encoding='utf') as tgt_file:
    datatgt = tgt_file.readlines()
```

```
df1 = pd.DataFrame(datatgt, columns=['tgt_content'])
```

```
df1
```

Figure 5 Target Data Reading

```
# Combine summaries and sources into a single DataFrame
combined_df = pd.concat([df, df1], axis=1)
```

Figure 6 Combining

```
combined_df.to_csv('train.csv', index=False)
```

Figure 7 Saving into CSV

In the same way test dataset is also formed and the whole data has been downloaded from (Fabbri *et al.*, 2019) dataset paper.

3 Text Cleaning

In this section all codes executed related to text cleaning will be pasted step by step.

```
#stop words removal
```

```
stop_words = stopwords.words('english')
df['src_content'] = df['src_content'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
```

Figure 8 Stop words removal

The above code is to remove the stop words from the dataset.

```
#Stemming
```

```
# Initialize the stemmer
stemmer = PorterStemmer()

# Apply stemming to the 'text' column
df['src_content'] = df['src_content'].apply(lambda x: stemmer.stem(x))
```

Figure 9 Stemming

The above code is to pass the stemming function and this removes different tenses of a word.

```
#Removal of unwanted email ids and websites
```

```
import re

def remove_emails_websites(text):
    # Remove email addresses
    text = re.sub(r'\S+@\S+', '', text)

    # Remove website links starting with http:// or https://
    text = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '', text)

    return text
```

```
df['src_content'] = df['src_content'].apply(remove_emails_websites)
```

Figure 10 Removing email id and websites

The above code is to remove unwanted email id's and websites present in the dataset.

```
#Removing Paranthesis and Hyphens
```

```
df['src_content'] = df['src_content'].str.replace(r'[\(\)\-]', '', regex=True)
```

Figure 11 Removing Paranthesis

Above is the regex function to remove the paranthesis and hyphens and hyphens which is applied on the data frame.

```
#function to remove newline_char
```

```
def remove_word(df, src_content, word):
    df[src_content] = df[src_content].str.replace(word, '')
    return df
word_to_remove = 'newline_char'
column_name = 'src_content'
df = remove_word(df,column_name,word_to_remove)
```

Figure 12 Removing Repeated word

A word has been found repetitive in the dataset that convey no meaning so a regex function is created to remove that particular word from the dataset.

4 Handling Redundancy

Below code is to handle the redundancy present in the data where cosine similarity matrix is is constructed and text are removed based on the similarity scores.

```
#Removing the redundancy present in the text
```

```
def remove_redundancy(Truncated_data, src_content):
    # Create TF-IDF vectorizer
    tfidf_vectorizer = TfidfVectorizer()

    # Fit and transform the text data
    tfidf_matrix = tfidf_vectorizer.fit_transform(Truncated_data[src_content])

    # Compute pairwise cosine similarity
    similarity_matrix = cosine_similarity(tfidf_matrix)
    print(similarity_matrix)

    # Create a mask to track redundant sentences
    mask = []

    for i in range(len(similarity_matrix)):
        # Check if the sentence is similar to any previous sentences
        if not any(similarity_matrix[i, j] > 0.9 for j in range(i)):
            mask.append(True)
        else:
            mask.append(False)

    # Filter out redundant sentences
    Truncated_data_filtered = Truncated_data[mask]

    return Truncated_data_filtered
```

Figure 13 Handling Redundancy

5 Model Building

```
# Custom Dataset and DataLoader
class CustomDataset(Dataset):
    def __init__(self, df, tokenizer, max_length):
        self.df = df
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        src_text = self.df.loc[idx, 'src_content'].strip()
        tgt_text = self.df.loc[idx, 'tgt_content'].strip()

        # Encode the inputs as tensors
        encoding = self.tokenizer.encode_plus(src_text, tgt_text, max_length=self.max_length, padding='max_length', return_t
        input_ids = encoding['input_ids'].squeeze()  # Remove extra batch dimension
        attention_mask = encoding['attention_mask'].squeeze()

        return input_ids, attention_mask
```

Figure 14 Data Loader

A custom function is created where the input are tokenized and they are encoded further converted into tensors.

```
# Train the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
model.train()

num_epochs = 1  # Set the number of training epochs
learning_rate = 1e-4  # Set the learning rate
optimizer = Adam(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()
```

Figure 15 Model Training

With number of epochs is equal to 1 the model has been trained with the cross entropy loss as the measure of training loss. Fairseq based Bart has been used to build the model.

```
]:
for epoch in range(num_epochs):
    total_loss = 0.0
    for input_ids, attention_mask in dataloader:
        input_ids, attention_mask = input_ids.to(device), attention_mask.to(device)
        labels = input_ids.clone()  # Clone input_ids to use as labels for summarization

        # Forward pass
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        total_loss += loss.item()

    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {total_loss / len(dataloader)}')

# Save the trained model
model.save_pretrained('bartModel')

# Optionally, you can also save the tokenizer
tokenizer.save_pretrained('Barttokenizer')
```

Figure 16 Model Training-2

Actual training code where the model is trained.

6 Evaluation

```
from rouge_score import rouge_scorer

# Function to generate summaries using the trained model
def generate_summary(model, tokenizer, src_text):
    input_ids = tokenizer.encode(src_text, return_tensors='pt', max_length=512, truncation=True)
    input_ids = input_ids.to(device)
    summary_ids = model.generate(input_ids, num_beams=4, max_length=150, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return summary

# Example function to calculate ROUGE scores
def calculate_rouge_scores(model, tokenizer, data_df):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

    rouge1_scores = []
    rouge2_scores = []
    rougeL_scores = []

    for idx, row in data_df.iterrows():
        src_text = row['src_content']
        tgt_text = row['tgt_content']

        generated_summary = generate_summary(model, tokenizer, src_text)
        print(generated_summary)

        scores = scorer.score(generated_summary, tgt_text)

        rouge1_scores.append(scores['rouge1'].fmeasure)
        rouge2_scores.append(scores['rouge2'].fmeasure)
        rougeL_scores.append(scores['rougeL'].fmeasure)

    avg_rouge1 = sum(rouge1_scores) / len(rouge1_scores)
    avg_rouge2 = sum(rouge2_scores) / len(rouge2_scores)
    avg_rougeL = sum(rougeL_scores) / len(rougeL_scores)

    return avg_rouge1, avg_rouge2, avg_rougeL

# Assuming you have a test DataFrame named 'test_df' with columns 'src_content' and 'tgt_content'
# Call the function to calculate the ROUGE scores
avg_rouge1, avg_rouge2, avg_rougeL = calculate_rouge_scores(model, tokenizer, test_df)

# Print the average ROUGE scores
print(f"Average ROUGE-1: {avg_rouge1}")
print(f"Average ROUGE-2: {avg_rouge2}")
print(f"Average ROUGE-L: {avg_rougeL}")
```

Figure 17 Evaluation code

The above evaluation code is to evaluate the model performance based on Rouge Metrics.

```
custom_row_idx = 1  # Change this to the index of the desired row in the test_df
custom_src_text = test_df.loc[custom_row_idx, 'src_content']

# Generate summary for the custom input
summary = generate_summary_customInput(model, tokenizer, custom_src_text)
print("Custom Input:")
print(custom_src_text)
print("\nGenerated Summary:")
print(summary)
```

Figure 18 Custom Input

The above code is to take custom input from the test dataset from which the model has never seen or trained on them.

The model has been built in the PC so a cpu based bart has been trained and then used to predict the sequence. It does not require any cuda graphics core or won't ask any dedicated graphics installed already.

References

Fabbri, R.A. *et al.* (2019) 'Multi-News: a Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model', *ARXIV* [Preprint]. Available at: https://doi.org/arXiv:1906.01749v3.