

Review Summarization using GPT2 [100 Marks]

1. Use the Amazon Fine Food Reviews dataset

2. Clean and preprocess the 'Text' and 'Summary' column from the dataset.

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

import nltk

df = pd.read_csv("../content/drive/MyDrive/in-4/Reviews.csv")
df.head()
```

0	1	B001E4KFG0	A3SGXH7AUHU8GW	deimartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dli pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...	
2	3	B000LQOCH0	ABXLMWJDXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...	
3	4	B000UA0QIQ	A395BORC6FGVXV	Kari	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...	

```
[ ] df.shape

(568454, 10)

[ ] df.isnull().sum()

Id                0
ProductId         0
UserId           0
ProfileName      26
HelpfulnessNumerator  0
HelpfulnessDenominator  0
Score            0
Time             0
--            --
```

```
[ ] df.shape
```

```
(568454, 10)
```

```
[ ] df.isnull().sum()
```

```
Id                0
ProductId         0
UserId           0
ProfileName       26
HelpfulnessNumerator  0
HelpfulnessDenominator  0
Score            0
Time             0
Summary          27
Text             0
dtype: int64
```

```
[ ] df.dropna(subset=['Summary'], inplace=True)
```

```
▶ df.isnull().sum()
```

```
Id                0
ProductId         0
UserId           0
ProfileName       26
HelpfulnessNumerator  0
HelpfulnessDenominator  0
Score            0
Time             0
Summary          0
Text             0
dtype: int64
```

```
[ ] newdf = df.copy()
```

```
[ ] newdf = df.copy()
```

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[ ] # Lowercase conversion
newdf['text_pp'] = newdf['Text'].apply(lambda x: x.lower())
newdf['summary_pp'] = newdf['Summary'].apply(lambda x: x.lower())

# Removing special characters
newdf['text_pp'] = newdf['text_pp'].apply(lambda x: re.sub(r'^a-zA-Z0-9\s', '', x))
newdf['summary_pp'] = newdf['summary_pp'].apply(lambda x: re.sub(r'^a-zA-Z0-9\s', '', x))

# Tokenization
newdf['text_pp'] = newdf['text_pp'].apply(lambda x: word_tokenize(x))
newdf['summary_pp'] = newdf['summary_pp'].apply(lambda x: word_tokenize(x))

# Removing stop words
stop_words = set(stopwords.words('english'))
newdf['text_pp'] = newdf['text_pp'].apply(lambda x: [word for word in x if word not in stop_words])
newdf['summary_pp'] = newdf['summary_pp'].apply(lambda x: [word for word in x if word not in stop_words])

# Lemmatization
lemmatizer = WordNetLemmatizer()
newdf['text_pp'] = newdf['text_pp'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])
newdf['summary_pp'] = newdf['summary_pp'].apply(lambda x: [lemmatizer.lemmatize(word) for word in x])
```

Data loading:-

->loaded dataset from the csv using pandas

->printed its shape to understand its dimensions

->checked the missing values and dropped missing values

Text Preprocessing:-

->created copy of original dataframe to preserve the preprocessed data separately

->converted text data to lowercase to ensure consistency

->removed special characters

->applied tokenization using nltk's word_tokenize function

->removed stop words from the text using nltk's stopwords corpus

->lemmatized the text to convert words to their base form using nltk's WordNetLemmatizer

Model Training

1. Initialize a GPT-2 tokenizer and model from Hugging Face.

2. Divide the dataset into training and testing (75:25)
3. Implement a custom dataset class to prepare the data for training.
4. Fine-tune the GPT-2 model on the review dataset to generate summaries.
5. Experiment with different hyperparameters such as learning rate, batch size, and number of epochs to optimize the model's performance.

```
import pandas as pd
import torch
from torch.utils.data import Dataset
from transformers import GPT2Tokenizer
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import torch
```

```
[ ] # Initialize GPT-2 tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Initialize GPT-2 model
model = GPT2Model.from_pretrained("gpt2")
```

```
[ ] from transformers import GPT2Tokenizer, GPT2LMHeadModel
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import torch
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.add_special_tokens({'pad_token': tokenizer.eos_token})
model = GPT2LMHeadModel.from_pretrained("gpt2")
```

```
class CreatingDataset(Dataset):
    def __init__(self, data, tokenizer, max_length):
        self.text_ = list(data['Cleaned_Text'])
        self.summary_ = list(data['Cleaned_Summary'])
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.length = len(self.text_)
    def __len__(self):
        return self.length
    def __getitem__(self, idx):
        review_text = self.text_[idx]
        summary = self.summary_[idx]

        inputs = self.tokenizer(review_text, max_length=self.max_length, truncation=True, padding='max_length', return_tensors="pt")
        labels = self.tokenizer(summary, max_length=self.max_length, truncation=True, padding='max_length', return_tensors="pt")

        return inputs["input_ids"].squeeze(0), labels["input_ids"].squeeze(0)
```

```

# Divide dataset into training and testing sets
train_data, test_data = train_test_split(newdf, test_size=0.25, random_state=42)

# Instantiate datasets and dataloaders with a larger max_length
train_dataset = CreatingDataset(train_data, tokenizer, max_length=1024)
test_dataset = CreatingDataset(test_data, tokenizer, max_length=1024)

# Define batch size
batch_size = 32

# Define data loaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Print dataset sizes
print("Training data size:", len(train_dataset))
print("Testing data size:", len(test_dataset))
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.1)
num_epochs = 3

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

for epoch in range(num_epochs):
    for batch_idx, batch in enumerate(train_loader):
        inputs, labels = batch
        inputs = inputs.to(device)
        labels = labels.to(device)

        # Check for empty tensors
        if inputs.size(0) == 0 or labels.size(0) == 0:
            continue

        optimizer.zero_grad()

        try:
            outputs = model(inputs, labels=labels)
            loss = outputs.loss
            loss.backward()
            optimizer.step()
            scheduler.step()

            print(f"Epoch [{epoch+1}/{num_epochs}], Batch [{batch_idx+1}/{len(train_loader)}], Loss: ")

        except IndexError as e:
            print(f"IndexError occurred: {e}")
            print(f"Batch {batch_idx}, Input IDs: {inputs}")
            print(f"Batch {batch_idx}, Labels: {labels}")
            continue

] # Save the fine-tuned model
model.save_pretrained("fine_tuned_gpt2")

] model_config = GPT2Config.from_pretrained("gpt2")
model = GPT2LMHeadModel(model_config)
state_dict = torch.load("fine_tuned_gpt2.pth", map_location=torch.device('cpu'))
model.load_state_dict(state_dict)
model.eval()

```

In the above code shown, I have first initialised the gpt2 tokenizer and gpt2 model from hugging face

->then I created custom dataset class to format data fro training

->split the dataset into training and testing sets of 75:25 ratio

->defined batch size, data loaders , optimizer , scheduler and no of epochs for training our model

->trained the gpt2 model using the training dataset and fine tuned it for review summarization

->utilized the adamW optimizer and implemented training loops, also checked for empty tensors and handled exceptions also

->savd the fine tuned model for future use.

Evaluation

After training, compute ROUGE scores on the test set to assess the model's overall performance i.e. compute ROUGE score for every predicted summary vs the actual summary.

->evaluated the gpt2-model's performance for generating summaries for review text using rough metrics

->calculated rouge scores(rouge-1,rouge-2,rouge-L) for evaluation

Results:-

->provided a sample text and given the actual summary or ground truth summary

->obtained generated summary from the gpt2 model

->calculated rough scores-precision,recall and f1 scores.