Loading Dataset

```
# Load the dataset:
import pandas as pd
df = pd.read_csv('./Reviews.csv')
```

[4] Python

Pandas library has been used to load the dataset

Cleaning And Preprocessing methods used

```
def preprocess_text(text):
    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Tokenization
    tokens = word_tokenize(text)
    # Lowercasing
    tokens = [word.lower() for word in tokens]
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]



    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    # Join tokens back into a string
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text
```

Python

```
# Combining the two columns review and summary:
df['combined'] = df['Text']  + 'TL;DR' + df['Summary']
```

Py

```
/tmp/ipykernel_207442/2285386154.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
  df['combined'] = df['Text']  + 'TL;DR' + df['Summary']
```

For fine tune the model Text and summary combined with delimeter TL;DR.

The **TL;DR** symbol, stands for “too lengthy; didn’t read,” is an ideal job designator for summarizing. The **TL;DR** symbol is used as a padding element to shorten the review text and it indicate to the model that the important content ends there.

Model Training

```
from transformers import AutoTokenizer, AutoModelWithLMHead

tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelWithLMHead.from_pretrained("gpt2")
```

gpt2 model from hugging face has been used for finetuning

Dividing the dataset into training and testing (75:25)

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df,test_size=0.25)
```

It tokenizes reviews, adds end-of-sequence token, pads/truncates to max length, and converts to PyTorch tensors. The **pad_truncate** method adjusts sequence length, ensuring consistency for model input.

Fine-tuning the GPT-2 model on the review dataset to generate summaries.

```
def train(model, optimizer, dl, epochs):
    for epoch in range(epochs):
        print("Epoch :",(epoch+1))
        for idx, batch in enumerate(dl):
            with torch.set_grad_enabled(True):
                optimizer.zero_grad()
                batch = batch.to(device)
                output = model(batch, labels=batch)
                loss = output[0]
                loss.backward()
                optimizer.step()
                if idx % 5000 == 0:
                    # print("loss: %f"%(loss))
                    print(f"loss: {loss}, {idx}")
    torch.save(model.state_dict(), 'model_completeDataset.pth')  # Save the trained model
```

Python

The `train` function iterates through epochs, batches data, computes gradients, updates weights using the optimizer, and prints loss every 5000 batches. It trains a model on a DataLoader (`dl`) using supervised learning, then saves the trained model's state dictionary.

```
train(model=model, optZimizer=optimizer, dl=dataloader, epochs=5)
```

```
Epoch : 1
loss: 6.765002727508545, 0
loss: 2.2353031635284424, 5000
loss: 2.0522336959838867, 10000
loss: 1.942301630973816, 15000
Epoch : 2
loss: 1.8105591535568237, 0
loss: 1.8702399730682373, 5000
loss: 1.4940941333770752, 10000
loss: 1.5958985090255737, 15000
Epoch : 3
loss: 1.5737464427947998, 0
loss: 1.3289415836334229, 5000
loss: 1.408329963684082, 10000
loss: 1.3203850984573364, 15000
Epoch : 4
loss: 1.3660566806793213, 0
loss: 1.6720166206359863, 5000
```

]

Model Inference

```python
def topk(probs, n=9):
    # The scores are initially softmaxed to convert to probabilities
    probs = torch.softmax(probs, dim= -1)

    # PyTorch has its own topk method, which we use here
    tokensProb, topIx = torch.topk(probs, k=n)

    # The new selection pool (9 choices) is normalized
    tokensProb = tokensProb / torch.sum(tokensProb)

    # Send to CPU for numpy handling
    tokensProb = tokensProb.cpu().detach().numpy()

    # Make a random choice from the pool based on the new prob distribution
    choice = np.random.choice(n, 1, p = tokensProb)
    tokenId = topIx[choice][0]

    return int(tokenId)
```

The `topk` function selects the top `n` probable tokens from softmax probabilities, normalizes them, converts to numpy, then randomly chooses one token based on the adjusted probability distribution. It returns the index of the selected token.

```python
def model_infer(model, tokenizer, review, max_length=15):
    # Preprocess the init token (task designator)
    review_encoded = tokenizer.encode(review)
    result = review_encoded
    initial_input = torch.tensor(review_encoded).unsqueeze(0).to(device)

    with torch.set_grad_enabled(False):
        # Feed the init token to the model
        output = model(initial_input)

        # Flatten the logits at the final time step
        logits = output.logits[0,-1]

        # Make a top-k choice and append to the result
        result.append(topk(logits))

        # For max_length times:
        for _ in range(max_length):
            # Feed the current sequence to the model and make a choice
            input = torch.tensor(result).unsqueeze(0).to(device)
            output = model(input)
            logits = output.logits[0,-1]
            res_id = topk(logits)

            # If the chosen token is EOS, return the result
            if res_id == tokenizer.eos_token_id:
                return tokenizer.decode(result)
            else: # Append to the sequence
                result.append(res_id)
    # IF no EOS is generated, return after the max_len
    return tokenizer.decode(result)
```

This function generates a review continuation given an initial text using a trained model. It iteratively predicts tokens, stopping at end-of-sequence or after a maximum length. The function utilizes the `topk` method for token selection and decodes the final sequence.

Evaluation

```python
Given_Review_Text =  "The Fender CD-60S Dreadnought Acoustic Guitar is a great instrument for beginners. It has a solid construction, produces a rich sound, and feels comfortable to
# Given_Review_Text=input()
```

Python

```python
Given_Summary = "Good for beginners but has tuning stability issues."
# Given_Summary=input()
```

Python

```python
generated_summary = model_infer(model, tokenizer, Given_Review_Text + " TL;DR ").split(" TL;DR ")[1].strip()
print("Generated Summary :",generated_summary)
```

Python

Computing Rouge Score:

```python
from rouge import Rouge

# Initialize Rouge
rouge = Rouge()

# Assuming you have generated summaries and actual summaries
generated_summaries = [generated_summary,]  # Replace [...] with your generated summaries
actual_summaries = [Given_Summary]     # Replace [...] with your actual summaries

# Compute ROUGE scores
scores = rouge.get_scores(generated_summaries, actual_summaries, avg=False)[0]

# Print ROUGE scores
print("Rouge Scores")
# print(scores)
print(f"ROUGE-1 : Precision:{scores['rouge-1']['p']}, Recall :{scores['rouge-1']['r']}, F1-Score :{scores['rouge-1']['f']}")
print(f"ROUGE-2 : Precision:{scores['rouge-2']['p']}, Recall :{scores['rouge-2']['r']}, F1-Score:{scores['rouge-2']['f']}")
print(f"ROUGE-l : Precision:{scores['rouge-l']['p']}, Recall :{scores['rouge-l']['r']}, F1-Score:{scores['rouge-l']['f']}")
```

✓ 0.0s Python

```
Rouge Scores
ROUGE-1 : Precision:0.3333333333333333, Recall :0.125, F1-Score :0.18181817785123974
ROUGE-2 : Precision:0.0, Recall :0.0, F1-Score:0.0
ROUGE-l : Precision:0.3333333333333333, Recall :0.125, F1-Score:0.18181817785123974
```

The code computes ROUGE scores for generated summaries against actual summaries using the Rouge library. It prints precision, recall, and F1-score for ROUGE-1, ROUGE-2, and ROUGE-L metrics, evaluating the quality of generated summaries compared to the ground truth.
ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics for evaluating automatic summarization and machine translation systems.

- ROUGE-1 measures unigram overlap between the generated summary and the reference summary.

- ROUGE-2 measures bigram overlap.

- ROUGE-L computes the longest common subsequence between the generated and reference summaries, considering word sequences.

These metrics provide insight into the quality of generated text in terms of content overlap with the reference summaries.

References used :

- https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews
- https://huggingface.co/openai-community/gpt2
- https://www.kaggle.com/code/changyeop/how-to-fine-tune-gpt-2-for-beginners
- https://www.youtube.com/watch?v=nsdCRVuprDY
- https://youtu.be/CDmPBsZ09wg?t=2558
- https://towardsdatascience.com/text-summarization-with-gpt2-and-layer-ai-599625085d8e