Bhuvi Ghosh
60009210191

Experiment No 8

Aim: To perform Text summarization using available libraries like Hugging Face's Transformers, TensorFlow or PyTorch

Introduction:

Text Summarization using BERT

Extractive Summarization: BERT can be utilized for extractive summarization, where key sentences or phrases are selected from the original text to form a summary.

Sentence Embedding's: Use BERT to generate embedding's for each sentence in the text.

Similarity Measures: Calculate similarity scores between sentences (e.g., using cosine similarity).

Select Top Sentences: Choose sentences with the highest similarity scores to form the summary.

Abstractive Summarization: This involves generating a summary that might not directly include sentences from the original text. BERT can aid in this by fine-tuning a model specifically for abstractive summarization.

Fine-tuning: Fine-tune the pre-trained BERT model on a summarization dataset (e.g., CNN/Daily Mail dataset).

Sequence-to-sequence Model: Employ techniques like seq2seq models or transformers to generate summaries.

Tools and Libraries:

Hugging Face's Transformers: This library provides easy access to pre-trained models like BERT and various other NLP-related functionalities for tasks like tokenization, model loading, and fine-tuning.

TensorFlow or PyTorch: Use these deep learning frameworks to implement BERT-based models and fine-tuning for specific tasks.

Lab Experiment to be performed in this session:

1. Perform Text Summarization using BERT (BERT summarizer library can be directly installed in python using the following commands python pip install bert-extractive-summarizer for the easies of the implementation.)

```
[ ] import torch
    from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
    from datasets import load_dataset
```

```
dataset = load_dataset("cnn_dailymail", "3.0.0", split='train')
train_texts = dataset['article'][:1000]
train_summaries = dataset['highlights'][:1000]
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 100% 15.6k/15.6k [00:00<00:00, 780kB/s]
train-00000-of-00003.parquet: 100% 257M/257M [00:02<00:00, 128MB/s]
train-00001-of-00003.parquet: 100% 257M/257M [00:01<00:00, 155MB/s]
train-00002-of-00003.parquet: 100% 259M/259M [00:02<00:00, 127MB/s]
validation-00000-of-00001.parquet: 100% 34.7M/34.7M [00:00<00:00, 107MB/s]
test-00000-of-00001.parquet: 100% 30.0M/30.0M [00:00<00:00, 122MB/s]
Generating train split: 100% 287113/287113 [00:13<00:00, 12738.39 examples/s]
Generating validation split: 100% 13368/13368 [00:00<00:00, 13782.84 examples/s]
Generating test split: 100% 11490/11490 [00:00<00:00, 11019.85 examples/s]
```

```
[ ] tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
    train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=512)
    summary_encodings = tokenizer(train_summaries, truncation=True, padding=True, max_length=150)
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 801B/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 4.37MB/s]
```

```
[ ] tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
    train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=512)
    summary_encodings = tokenizer(train_summaries, truncation=True, padding=True, max_length=150)
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 801B/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 4.37MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 6.72MB/s]
config.json: 100% 570/570 [00:00<00:00, 15.1kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_toke
  warnings.warn(
```

```
[ ] class SummarizationDataset(torch.utils.data.Dataset):
        def __init__(self, encodings, summaries):
            self.encodings = encodings
            self.summaries = summaries

        def __getitem__(self, idx):
            item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
            item["labels"] = torch.tensor(self.summaries["input_ids"][idx])
            return item

        def __len__(self):
            return len(self.encodings["input_ids"])

    train_dataset = SummarizationDataset(train_encodings, summary_encodings)
```

```
[ ] model = BertForSequenceClassification.from_pretrained("bert-base-uncased")
    training_args = TrainingArguments(
        output_dir="./results",
        evaluation_strategy="steps",
        num_train_epochs=2,
        per_device_train_batch_size=4,
        per_device_eval_batch_size=4,
        warmup_steps=500,
        weight_decay=0.01,
        logging_dir="./logs",
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
    )
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1525: FutureWarning: `evaluation_strategy` i
  warnings.warn(
```

```
from transformers import BartTokenizer, BartForConditionalGeneration, Trainer, TrainingArguments
import torch

tokenizer = BartTokenizer.from_pretrained("facebook/bart-large-cnn")
model = BartForConditionalGeneration.from_pretrained("facebook/bart-large-cnn")
```

```
vocab.json: 100% 899k/899k [00:00<00:00, 3.57MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 2.62MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 19.5MB/s]
config.json: 100% 1.58k/1.58k [00:00<00:00, 31.1kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tok
  warnings.warn(
model.safetensors: 100% 1.63G/1.63G [00:18<00:00, 54.6MB/s]
```

```
[ ] train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=512)
    target_encodings = tokenizer(train_summaries, truncation=True, padding=True, max_length=150)
```

```
[ ] class SummarizationDataset(torch.utils.data.Dataset):
        def __init__(self, encodings, summaries):
            self.encodings = encodings
            self.summaries = summaries

        def __getitem__(self, idx):
            item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
            item["labels"] = torch.tensor(self.summaries["input_ids"][idx])
            return item

        def __len__(self):
            return len(self.encodings["input_ids"])

    train_dataset = SummarizationDataset(train_encodings, target_encodings)
```

```
    training_args = TrainingArguments(
        output_dir="./results",
        per_device_train_batch_size=2,
        per_device_eval_batch_size=2,
        num_train_epochs=2,
        warmup_steps=500,
        weight_decay=0.01,
        logging_dir='./logs',
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
    )

    trainer.train()
```

Step	Training Loss
500	1.550800
1000	0.596300

```
Some non-default generation parameters are set in the model config. These should go into a GenerationConfig file (https://huggingfac
Non-default generation parameters: {'max_length': 142, 'min_length': 56, 'early_stopping': True, 'num_beams': 4, 'length_penalty': 2
Some non-default generation parameters are set in the model config. These should go into a GenerationConfig file (https://huggingfac
Non-default generation parameters: {'max_length': 142, 'min_length': 56, 'early_stopping': True, 'num_beams': 4, 'length_penalty': 2
TrainOutput(global_step=1000, training_loss=1.0735016479492188, metrics={'train_runtime': 716.1716, 'train_samples_per_second': 2.79
'total_flos': 2167104602112000.0, 'train_loss': 1.0735016479492188, 'epoch': 2.0})
```

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

```
[ ] def generate_summary(text):
        inputs = tokenizer(text, return_tensors="pt", max_length=512, truncation=True).to(device)  # Move inputs to device
        outputs = model.generate(inputs.input_ids, max_length=150, min_length=50, length_penalty=2.0, num_beams=4, early_stopping=True)
        summary = tokenizer.decode(outputs[0], skip_special_tokens=True)
        return summary
```

```
    text = "This is a news article about a college having a culture fest in the next two days. You have been invited and have to RSVP. M
    print("Generated Summary:", generate_summary(text))
```

```
Generated Summary: This is a news article about a college having a culture fest.
You have to RSVP. There is no theme for the party.
If you get extra guests you have to inform in advance.
Timings will be told later.
```