NLP Tasks Project Overview

Memory updated

Here's the revised plan for the **Text Summarization Project** using **Streamlit** and the **BART** model, presented in a more descriptive way with detailed stages.

End-to-End Stages for Medical Text Summarization Project (with Streamlit)

Stage 1: Dataset Preparation (Medical Book Processing)

1.1 Extract Text from the Medical Book PDF

- **Objective**: Extract the content from the provided PDF `Medical_book.pdf`.
- **Approach**: Use a Python library such as **PyPDF2** or **pdfminer** to read the PDF and extract the text.
- **Steps**:
  - Load the PDF and extract the text.
  - Store the extracted text in a text file or directly use it for preprocessing.
  - Example code for extraction:

```
python
```

```python
import PyPDF2

def extract_text_from_pdf(pdf_path):
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = ''
        for page in reader.pages:
            text += page.extract_text()
    return text

pdf_text = extract_text_from_pdf('Data/Medical_book.pdf')
```

1.2 Preprocess the Extracted Text

- **Objective**: Clean and prepare the text for training.
- **Approach**: Remove unwanted characters (special symbols, page numbers, headers, etc.), and split the text into segments (e.g., based on disease descriptions, diagnosis, etc.).
- **Steps**:
  - Write a preprocessing function to remove unnecessary characters.
  - Optionally, segment the text into meaningful sections.
  - Example preprocessing code:

```
python


def preprocess_text(text):
    text = text.replace('\n', ' ').replace('\r', '').strip()
    # Additional cleaning steps can be added here
    return text

cleaned_text = preprocess_text(pdf_text)
```

1.3 Create Summarization Pairs

- **Objective**: Create training data by pairing longer text with their corresponding summaries.
- **Approach**: Either manually create summaries or extract summary-like sections from the book (e.g., diagnosis or key points).
- **Steps**:
  - Break the book into chunks and match each chunk with a summary. This will be your training data.

Stage 2: Model Selection and Fine-Tuning (BART)

2.1 Select BART for Summarization

- **Objective**: Use **BART** (Bidirectional and Auto-Regressive Transformer) as the model for abstractive summarization.
- **Model**: Pre-trained model `**facebook/bart-large-cnn**`.
- **Why BART?**: BART is highly effective for generating fluent and concise summaries, making it ideal for summarizing complex medical text.

2.2 Set Up the Environment

- **Steps**:
  - Create a virtual environment:

```
bash


python -m venv nlp_project
source nlp_project/bin/activate  # Linux/Mac
nlp_project\Scripts\activate  # Windows
```

  - Install the required libraries:

```
bash


pip install transformers datasets torch streamlit
```

2.3 Fine-Tuning the BART Model

- **Objective**: Fine-tune BART on your dataset to generate high-quality summaries for medical text.
- **Approach**: Write a Jupyter notebook to fine-tune BART using **Hugging Face Transformers**.

- **Steps**:
  - Tokenize the input text and summaries.
  - Fine-tune the BART model on your dataset.
  - Save the fine-tuned model for later inference.
  - Example fine-tuning code:

```
python
```

```python
from transformers import BartTokenizer, BartForConditionalGeneration, Trainer, TrainingArguments
from datasets import load_dataset

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

def preprocess_function(examples):
    inputs = tokenizer(examples['text'], max_length=1024, truncation=True)
    labels = tokenizer(examples['summary'], max_length=150, truncation=True)
    return {'input_ids': inputs['input_ids'], 'labels': labels['input_ids']}

# Assume your medical dataset is in Hugging Face's dataset format
dataset = load_dataset('your_dataset')
tokenized_dataset = dataset.map(preprocess_function, batched=True)

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy='epoch',
    learning_rate=3e-5,
    per_device_train_batch_size=2,
    num_train_epochs=3
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset['train'],
    eval_dataset=tokenized_dataset['validation']
)

trainer.train()
```

2.4 Saving the Fine-Tuned Model

- **Objective**: Save the model after fine-tuning so that it can be used in the Streamlit app.
- **Steps**:
  - After fine-tuning, save the model in the `**models/**` folder:

```
python
```

```python
model.save_pretrained('models/fine_tuned_bart/')
```

Stage 3: Model Evaluation

3.1 Evaluate the Model

- **Objective**: Evaluate the fine-tuned model's performance using summarization metrics.
- **Metrics**:

- **ROUGE**: Measures the overlap between generated summaries and reference summaries.
- **BLEU**: Measures how similar the generated summary is to the reference.

- **Steps**:
  - Load the fine-tuned model and evaluate it on a test set.
  - Example evaluation using **ROUGE**:

```
python
```

```python
from datasets import load_metric

rouge = load_metric("rouge")
predictions = trainer.predict(tokenized_dataset['test']).predictions
decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
decoded_labels = tokenizer.batch_decode(tokenized_dataset['test']['labels'],
skip_special_tokens=True)

rouge_score = rouge.compute(predictions=decoded_preds,
references=decoded_labels)
print(rouge_score)
```

3.2 Test on Real Medical Text

- **Objective**: Test the model on various sections of the medical textbook and verify the quality of summaries.
- **Approach**: Perform qualitative testing by summarizing different disease descriptions and checking the output.

Stage 4: App Development with Streamlit

4.1 Set Up the Streamlit App

- **Objective**: Develop a Streamlit app where users can input text and get a summarized version.
- **Approach**: Build a simple app that uses the fine-tuned BART model to generate summaries.

4.2 Streamlit App Code

- **Steps**:
  - Create a simple web interface with Streamlit.
  - Load the fine-tuned BART model and allow users to input text.
  - Example code:

```
python
```

```python
import streamlit as st
from transformers import BartTokenizer, BartForConditionalGeneration

# Load the fine-tuned BART model
model = BartForConditionalGeneration.from_pretrained('models/fine_tuned_bart/')
tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')

st.title("Medical Text Summarizer")

# Text input from user
user_input = st.text_area("Enter medical text to summarize:")

if st.button("Summarize"):
```

```
    inputs = tokenizer(user_input, return_tensors='pt', max_length=1024,
truncation=True)
    summary_ids = model.generate(inputs['input_ids'], max_length=150,
min_length=40, length_penalty=2.0)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    st.write("Summary:")
    st.write(summary)
```

4.3 Testing the App

- **Objective**: Test the app with different inputs to ensure the summarization output is accurate and meaningful.
- **Steps**:
  - Test with small paragraphs, larger sections, and technical descriptions to validate output quality.

Stage 5: Version Control with GitHub

5.1 GitHub Repository Setup

- **Objective**: Push all project files to the GitHub repository you've already created (`**NLP-Text-Summarization**`).
- **Steps**:
  - Add all necessary files to Git:

    bash

    ```
    git add .
    git commit -m "Add text summarization code and data"
    ```

  - Push to GitHub:

    bash

    ```
    git push origin main
    ```

5.2 Organize Files in the Repository

- **Folder Structure**:
  - Store **data**, **code**, and **models** in respective folders to keep the repository organized.
  - **README.md**: Update the README with project details, including how to run the Streamlit app.

Stage 6: Testing and Deployment

6.1 Test the App Locally

- **Objective**: Ensure the app runs smoothly on your local machine.

- **Steps**:
  - Run the Streamlit app:

```
bash

streamlit run app/app.py
```

6.2 Deploy the App

- **Objective**: Deploy

continue from the above

6.2 Deploy the App

- **Objective**: Deploy the Streamlit app so that it can be accessed online.
- **Options**:
  - **Streamlit Cloud**: Streamlit's own platform for deploying apps. This is the easiest way to deploy Streamlit apps.
  - **Heroku**: You can also deploy the app on Heroku or other cloud platforms.

Steps to Deploy on Streamlit Cloud:

- Push all your files to your GitHub repository.
- Go to **Streamlit Cloud** and log in using your GitHub account.
- Select your repository (`**NLP-Text-Summarization**`).
- Configure the app (select the appropriate `**app.py**` file).
- Deploy and test the app in your browser.

Steps to Deploy on Heroku:

- Create a `**Procfile**` in the project root directory:

```
arduino

web: streamlit run app/app.py
```

- Create a `**requirements.txt**`:

```
bash

pip freeze > requirements.txt
```

- Commit and push to GitHub:

```bash
git add .
git commit -m "Prepare for Heroku deployment"
git push origin main
```

- Install the Heroku CLI and run the following commands:

```bash
heroku create
git push heroku main
heroku open
```

Final Project Structure:

```bash
NLP-Text-Summarization/
│
├── Data/
│   └── Medical_book.pdf     # The medical textbook
│
├── app/
│   ├── app.py               # Streamlit app
│   └── __init__.py          # For app initialization (optional)
│
├── models/
│   └── fine_tuned_bart/     # Fine-tuned BART model
│
├── notebooks/
│   └── data_preprocessing.ipynb   # Jupyter notebook for text extraction and preprocessing
│   └── fine_tuning_bart.ipynb     # Jupyter notebook for model fine-tuning and evaluation
│
├── requirements.txt         # Required Python libraries
├── README.md                # Project description
└── .gitignore               # Files to ignore (e.g., models, virtual environment)
```

Summary of All Stages:

1. **Stage 1: Dataset Preparation**:
   - Extract and preprocess text from the medical book PDF.
   - Create input-text-summary pairs for training.

2. **Stage 2: Model Selection and Fine-Tuning**:
   - Use **BART** for abstractive summarization.
   - Fine-tune the model on your medical dataset using **Hugging Face**.

3. **Stage 3: Model Evaluation**:

- Evaluate the model's performance using **ROUGE** and **BLEU** metrics.
- Test on real medical text from the book.

4. **Stage 4: App Development with Streamlit**:
   - Build a simple web app where users can input medical text and get a summary.
   - Use Streamlit for the frontend.

5. **Stage 5: Version Control with GitHub**:
   - Organize your files and push everything to your GitHub repository.

6. **Stage 6: Testing and Deployment**:
   - Test the app locally.
   - Deploy the app using **Streamlit Cloud** or **Heroku** for online access.

Let me know if this detailed plan works for you, and I'll be ready to assist you with any further steps or coding!