Assignment 1:Deployment Instructions

By: Pradyumna Seethamraju

Prerequisites

1. **Install Python** (if not already installed): Ensure you have Python 3.7 or above.
2. **Install Required Libraries**: Use `pip` to install all necessary libraries. Based on the code, here are the dependencies:
   pip install torch transformers flask flask-caching flask-limiter pandas sklearn mlflow apscheduler matplotlib
3. **Download Model Assets**: The model uses the `bert-base-uncased` tokenizer and `BertForSequenceClassification` from Hugging Face. These assets will be downloaded automatically the first time the code runs.

Step-by-Step Deployment

Step 1: Prepare the Dataset (Optional)

If you want to retrain or test the model, ensure you have the dataset available as specified in the code (`stanfordnlp/sst2` from Hugging Face). You can use the existing code to load and split the dataset.

Step 2: Train the Model (Optional)

If the model hasn't been trained or you want to retrain it, execute the training code:

- The training function `train_model` in the provided code trains the BERT model on the sentiment analysis dataset.
- Run the training code, which will save the trained model in memory for immediate use.

Step 3: Test Model Evaluation (Optional)

The `evaluate_model` function allows you to test the model's performance. It calculates accuracy and F1 score on the validation data, which can be useful for validating the model before deploying it.

Step 4: Run the Flask API

1. **Set up the Flask Application**: The code already includes the Flask app and routes for single and batch prediction.
2. **Start the API Server**: Run the following command to start the server: Assignment1_V2.py
3. **Access the API**: Once started, the server will listen on `http://0.0.0.0:5000` (local IP and port 5000).
    a. **Single Prediction**: Send a POST request to `http://127.0.0.1:5000/predict` with JSON data containing a single `text` field.
    b. **Batch Prediction**: Send a POST request to `http://127.0.0.1:5000/batch_predict` with JSON data containing a list of texts under the `texts` key.

Step 5: Test the Endpoints

Using a tool like `curl` or Postman, you can test the endpoints:

- **Single Prediction Test**:

```
curl -X POST "http://127.0.0.1:5000/predict" -H "Content-Type: application/json" -d
'{"text": "This is a great product!"}'
```

- **Batch Prediction Test**:

```
curl -X POST "http://127.0.0.1:5000/batch_predict" -H "Content-Type: application/json" -d
'{"texts": ["This is great!", "I didn\'t like it."]}'
```

Step 6: Monitor and Maintain

- **Model Performance Monitoring**: The code allows for periodic evaluation with `apscheduler`. You can schedule the `evaluate_model` function to run periodically if you wish to monitor the model's staleness in real-time.
- **Caching and Rate Limiting**: The `Flask-Caching` and `Flask-Limiter` libraries are used to cache responses and limit the request rate, helping with resource management on local deployment.

Optional: Logging with MLflow

If MLflow is configured, it will automatically log model training parameters and metrics for tracking purposes. For local runs, ensure the MLflow server is configured to store experiment logs in a specified directory. You can view logs with:

```
mlflow ui
```

This starts a UI accessible at `http://127.0.0.1:5000` to review model performance over time.