Machine Learning Project Documentation

Project Title: Mental health detection from social media posts

Team Members

1. Aung Thu Phyo (testing data)
2. Su Mon Hlaing
3. Moe Thet Hmue (testing data)
4. Nang Mon Kham (testing data)
5. Phyo Ko Ko (deployment)

1. Overview

The deployment phase focuses on transitioning the fine-tuned DistilBERT model from a training environment to a production-ready **Azure Managed Online Endpoint**. The goal is to provide real-time inference capabilities for the web application.

The process involves registering the model in Azure Machine Learning, defining a scoring script (score.py) to handle data preprocessing and prediction, and deploying the model to a Standard_DS2_v2 compute instance. This setup ensures that when a user submits a profile URL, the system can instantly scrape, process, and analyze the text to return a mental health assessment without the latency of batch processing.

2. Model Serialization

Process:

We utilized the Hugging Face transformers library to serialize the model. After fine-tuning the pre-trained distilbert-base-uncased model, the model artifacts were saved using the save_pretrained() method.

Format and Storage:

- **Format:** The model is serialized into standard PyTorch weights (pytorch_model.bin), along with the configuration file (config.json) and tokenizer assets (vocab.txt, tokenizer.json).

- **Storage:** These artifacts are uploaded and versioned in the **Azure Machine Learning Model Registry**. This ensures reproducibility and allows easy rollback to previous model versions if necessary.

3. Model Serving

Serving Mechanism:

The serialized model is served using an Azure Managed Online Endpoint (Real-time). This manages the underlying infrastructure, OS patching, and scaling automatically.

- **Scoring Script (score.py):** A Python script was developed to load the model into memory upon container startup (init() function) and handle incoming requests (run() function). The script includes logic to clean and tokenize raw text input before passing it to the model.
- **Platform:** We chose **Azure Cloud** over on-premises solutions for its scalability and integration with other cloud services (like Logic Apps).
- **Compute:** The deployment runs on a **Standard_DS2_v2** instance (2 vCPUs, 7 GB RAM), which provides sufficient memory to load the BERT model and low latency for inference.

4. API Integration

The machine learning model is exposed via a **RESTful API** provided by the Azure Online Endpoint.

- **Integration Flow:** A Logic App acts as the orchestrator. It receives the user's request, scrapes the data (via Apify), and sends the post content to the ML API.
- **Endpoint URL:** https://<your-endpoint-name>.eastasia.inference.ml.azure.com/score
- **Input Format (JSON):**

```
JSON
{
  "inputs": ["I feel very anxious today", "This is a normal post"]
}
```

- Response Format (JSON):

The API returns a JSON array containing the original text and the predicted class index (e.g., 0 for Anxiety, 3 for Normal).

```
JSON
[
 { "text": "I feel very anxious today", "prediction": 0 },
 { "text": "This is a normal post", "prediction": 3 }
]
```

5. Security Considerations

Several security measures were implemented to protect the deployed model:

- **Authentication:** The endpoint is secured using **Key-based Authentication**. The consuming Logic App must include a valid API Key in the Authorization header (Bearer <API_KEY>) to access the model.
- **Encryption:** All traffic between the client (Logic App) and the Azure Endpoint is encrypted using **HTTPS (TLS 1.2)**, ensuring data integrity and privacy during transit.
- **Network Isolation:** (Optional note if relevant) The endpoint is hosted within the Azure secure infrastructure, limiting access only to authorized traffic.

6. Monitoring and Logging

Tools:

We utilize Azure Monitor and Azure Application Insights which are natively integrated with Managed Online Endpoints.

Metrics Tracked:

- **Request Latency:** To ensure the model responds within acceptable time limits (target < 500ms).
- **HTTP Response Codes:** Monitoring for 200 OK vs 5xx errors to detect service crashes.
- **Resource Utilization:** CPU and Memory usage of the container to prevent crashes due to Out-Of-Memory (OOM) errors.

Logging:

The score.py script utilizes the Python logging module to capture standard output and error traces. These logs are accessible via the Azure Portal console, allowing for rapid debugging of failed requests or model loading issues.