Machine Learning Project Documentation

Deployment

1. Overview

The deployment phase involves setting up a Flask-based web service to serve a machine learning model for prediction. This includes setting up necessary environments, installing dependencies, serializing the model, and making it accessible through API endpoints. Docker is used for containerization to ensure consistent deployment across different environments.

2. Model Serialization

The machine learning model is serialized using a Python library like pickle, which is common for models built using scikit-learn, xgboost, and other machine learning frameworks included in the requirements.txt file. This process allows the model to be stored in a binary format, making it easier to load and use during inference. Efficient storage can be achieved by compressing the model if it's large, especially when deploying in resource-constrained environments.

3. Model Serving

The serialized model is served via a Flask API. Flask, included in the requirements.txt file, is a lightweight web framework suitable for hosting the model and exposing endpoints for inference. The app likely listens for incoming POST requests, containing input data, runs the model, and returns predictions.

The Dockerfile indicates that the service is containerized, ensuring that the Flask app runs consistently across different environments. The Docker setup allows the application to be deployed in various cloud platforms, such as AWS, Google Cloud, or Azure, or even on-premise.

4. API Integration

The Flask application is integrated with an API that exposes at least one endpoint for making predictions. The input format would be a JSON payload, where the input features are provided by the user. The app then processes the input, uses the trained model for inference, and responds with a prediction in JSON format. Details of the API such as endpoint URL, methods (e.g., POST), and input-output formats are defined in the app.py file.

6. Monitoring and Logging

The model’s performance and usage can be monitored by logging requests and responses, tracking model prediction times, and monitoring resource usage. Flask's built-in logging capabilities can be used to capture these events, while additional tools or platforms like Prometheus or CloudWatch could be used for tracking custom metrics (like model accuracy drift, latency, or error rates) and setting up alerting mechanisms.