Machine Learning Project Documentation

Project Title: Offline AI-Powered Community Support and Vulnerability Prediction System for Conflict-Affected Areas in Myanmar

Group Members (Active): Thant Zin Moe, Aye Yu Yu San

Deployment

1. Overview

The deployment phase focuses on transitioning the trained machine learning model from a development environment to a production-ready state suitable for the target environment: offline, low-resource devices in Myanmar. Unlike traditional web-based deployments, this project utilizes an embedded deployment strategy. The model is serialized and packaged directly into the application or used to pre-generate static intelligence assets (risk maps) that are distributed physically. This approach ensures the system functions without internet connectivity, meeting the core requirement of the project.

2. Model Serialization

Process: To deploy the trained Random Forest model, we utilized the joblib library in Python. joblib is more efficient than Python's standard pickle module for objects carrying large numpy arrays, which is characteristic of our scikit-learn model.

Format: The model was serialized into a binary file format (.joblib or .pkl).

Considerations: We prioritized minimizing the file size to ensure it can be stored on devices with limited storage capacity. By pruning the decision trees (setting max_depth=10 during training), we significantly reduced the complexity and size of the serialized object.

3. Model Serving

Serving Strategy: Since the target environment lacks internet, we do not use a cloud-based model server (like AWS SageMaker or Flask API). Instead, we employ two serving methods:

- **Method A (Static Serving)**: The primary "serving" method is batch inference. We run the model on our development machine to generate “Vulnerability Heatmap” and “Risk Scores” for the next 12 months. These outputs are saved as static files (CSV/JSON) and images.
- **Method B (On-Device Serving)**: For the future mobile app integration, the serialized model file is embedded directly into the application's assets. The app loads this file into memory and runs predictions locally on the device's CPU using a lightweight Python runtime (e.g., Chaquopy for Android) or a converted format (e.g., TensorFlow Lite for the 1D-CNN component).

4. API Integration

Status: As this is an offline-first project designed for disconnected regions, there is no live REST API integration.

Alternative: The “API” in this context is the internal function call within the offline application code. The app's logic calls a local function predict_risk(region_id) which queries the pre-loaded model or static database, returning the risk level and recommended chatbot content immediately without network requests.

5. Security Considerations

Data Privacy: The most critical security measure is the offline architecture itself. Since no user data is ever transmitted to a cloud server, the risk of interception or data leakage is effectively zero. All processing happens locally on the user's device.

Model Security: To prevent tampering, the serialized model file and the application code are obfuscated and signed. This ensures that the risk logic cannot be altered by malicious actors to provide false safety information.

Encryption: The local knowledge base (JSON) is stored in an encrypted format within the app's private storage, accessible only by the application, to prevent unauthorized modification of the aid information.

6. Monitoring and Logging

Challenge: Traditional real-time monitoring (like Prometheus or Grafana) is impossible without internet connectivity.

Offline Logging Strategy: The application implements a local logging mechanism. It records performance metrics (e.g., inference time, app crashes, and successful information retrievals) to a local, encrypted log file on the device.

Feedback Loop: These logs are designed to be manually retrieved by field workers during periodic site visits or when a user temporarily gains connectivity. This "store-and-forward" approach allows the development team to analyze model performance and user behavior retrospectively to improve future versions of the model and app.