Machine Learning Technologies in the Safe Route Model

Prepared for Academic Presentation

May 4, 2025

Abstract

This document provides a detailed description of the machine learning (ML) technologies used in the `safe_route_ml.py` model, designed to rank 6–7 routes in Delhi based on safety, prioritizing crime severity and spatial crime patterns over distance. The model integrates supervised learning (Random Forest Regression), clustering (DBSCAN), and spatial analysis (KDTree, geodesic distance) to process a large crime dataset and evaluate routes. Key formulas for safety score calculation, feature engineering, and clustering are presented, along with explanations of their roles and justifications for their selection. This document is intended to support an academic explanation of the model's methodology.

1 Introduction

The `safe_route_ml.py` model addresses the problem of ranking 6–7 driving routes between a source and destination in Delhi, prioritizing safety based on crime severity and spatial patterns (hotspots) over distance. The model processes a crime dataset (`2021-2024_DELHI_DATA.csv`, ∼509,425 records) with attributes `Latitude`, `Longitude`, `Severity`, `CrimeID`, and `CrimeCategory`. It fetches routes using the Open Source Routing Machine (OSRM) API and ranks them by a safety score (0.1–1.0), considering factors like crime severity, hotspot proximity, and time of day (Morning, Afternoon, Evening, Night). The model combines rule-based scoring with machine learning to ensure robust, interpretable results.

This document details the ML technologies used, including their mathematical formulations, roles, and justifications, to support an academic presentation. The technologies include Pandas, KDTree, DBSCAN, Random Forest Regression, geodesic distance calculations, and a Flask-based API.

2 Problem Definition

The objective is to rank 6–7 routes between a source and destination based on safety, where safety is determined by:

- **Crime Severity**: Routes with lower severity crimes are preferred.
- **Crime Patterns**: Routes avoiding high-severity crime hotspots (dense crime clusters) are prioritized.
- **Safety Over Distance**: Safer routes are chosen even if longer (e.g., Route B: 12 km, 350 low-severity crimes vs. Route C: 6 km, high-severity crimes).

Input:

- Crime dataset with ∼509,425 records.
- Source and destination coordinates (latitude, longitude).

- Time category (Morning, Afternoon, Evening, Night).

Output: A JSON array of 6–7 routes, each with:

- `route_name`, `total_crimes`, `safety_score` (0.1–1.0), `total_distance_km`, `nearby_crimes`, `route_coords`, `time_category`.

3 Machine Learning Technologies

3.1 Data Preprocessing and Spatial Analysis

3.1.1 Pandas for Data Loading

Technology: Pandas is used to load and preprocess the crime dataset.

Process: The dataset is read into a DataFrame, ensuring required columns are present. `Latitude` and `Longitude` are converted to floats for spatial calculations.

Purpose: Provides a structured representation of the large dataset for efficient querying and analysis.

3.1.2 KDTree for Spatial Indexing

Technology: `scipy.spatial.KDTree`.

Process: A KDTree is constructed from crime coordinates (`latitude`, `longitude`) to enable fast nearest-neighbor queries within a radius $r = 0.1$ degrees ($\sim$11.1 km).

Formula: For a query point $p = (lat, lon)$ and crime point $q = (lat_q, lon_q)$, the Euclidean distance is:

$$\text{Distance}(p, q) = \sqrt{(lat_p - lat_q)^2 + (lon_p - lon_q)^2}$$

Points with distance $\leq r/111$ ($\sim$100m) are returned.

Purpose: Efficiently identifies crimes near route points, critical for feature extraction in a dataset with $\sim$509,425 records.

3.1.3 Geodesic Distance Calculation

Technology: `geopy.distance.geodesic`.

Process: Computes the total route distance by summing geodesic distances between consecutive coordinate pairs using the WGS84 ellipsoid.

Formula: For coordinates $c_i = (lat_i, lon_i)$ and $c_{i+1} = (lat_{i+1}, lon_{i+1})$:

$$\text{Distance}_{\text{total}} = \sum_{i=0}^{n-2} \text{geodesic}(c_i, c_{i+1}).\text{km}$$

Purpose: Provides route length as a feature, though safety is prioritized over distance.

3.2 Clustering for Crime Pattern Detection

Technology: `sklearn.cluster.DBSCAN`.

Why Chosen: DBSCAN identifies crime hotspots without requiring a predefined number of clusters, ideal for spatial data with varying density.

Process:

- Clusters crime coordinates with `eps = 0.001` (∼100m) and `min_samples = 5`, producing ∼921 clusters.
- Assigns labels (-1 for noise, $0, 1, \ldots$ for clusters).
- For each non-noise cluster k:
  - **Centroid**:
$$\text{Centroid}_k = \left(\frac{1}{n_k} \sum_{i \in \text{cluster}_k} lat_i, \frac{1}{n_k} \sum_{i \in \text{cluster}_k} lon_i \right)$$
  - **Average Severity**:
$$\text{Severity}_{\text{avg},k} = \frac{1}{n_k} \sum_{i \in \text{cluster}_k} \text{Severity}_i$$

Purpose: Detects high-severity hotspots (severity $\geq 0.6 \times$ max_severity) to penalize routes passing through dangerous areas.

3.3 Feature Engineering

Technology: NumPy, `sklearn.preprocessing.LabelEncoder`.

Process: Extracts 10 features per route to capture crime severity, patterns, and context:

1. **total_crimes**: Number of unique crimes within 100m (via KDTree).
2. **avg_severity**:
$$\text{avg\_severity} = \frac{\sum_{c \in \text{nearby\_crimes}} \text{Severity}_c}{\text{total\_crimes}} \quad (0 \text{ if no crimes})$$
3. **max_severity**:
$$\text{max\_severity} = \max_{c \in \text{nearby\_crimes}} \text{Severity}_c \quad (0 \text{ if no crimes})$$
4. **high_severity_crimes**: Count of crimes with severity $\geq 0.6 \times$ max_severity.
5. **distance**: Total route distance (km).
6. **crime_types**: Number of unique crime categories.
7. **time_encoded**: Encoded time category (Morning=0, Afternoon=1, Evening=2, Night=3).
8. **num_hotspots**: Route points within 100m of any cluster:
$$\text{num\_hotspots} = \sum_{p \in \text{route\_coords}} \mathbb{1}\left(\min_{c \in \text{cluster\_centroids}} \|p - c\| < 0.001 \right)$$
9. **high_severity_hotspots**: Route points near high-severity clusters.
10. **min_distance_to_hotspot**: Minimum distance to any cluster (km):
$$\text{min\_distance\_to\_hotspot} = \min_{p,c} \|p - c\| \times 111$$

Purpose: Quantifies factors affecting safety, enabling differentiation of routes (e.g., Route B: low severity, no hotspots).

3.4 Safety Score Calculation (Rule-Based)

Process: Computes a safety score to prioritize routes with lower severity and fewer hotspots.

Formula:

$$\text{safety\_score}_{\text{raw}} = 100 \times (1 - \text{severity\_penalty} - \text{high\_severity\_penalty} - \text{hotspot\_penalty}) \times \text{time\_multiplier}$$

- **severity_penalty**:

$$\text{severity\_penalty} = \frac{\text{total\_severity}}{\text{max\_crimes\_per\_route} \times \text{max\_severity}}$$

  where $\text{total\_severity} = \sum \text{Severity}_c$, and max_crimes_per_route is the maximum crime count across routes.

- **high_severity_penalty**:

$$\text{high\_severity\_penalty} = \frac{\text{high\_severity\_crimes}}{\text{max\_crimes\_per\_route}} \times 0.5$$

- **hotspot_penalty**:

$$\text{hotspot\_penalty} = \text{high\_severity\_hotspots} \times 0.05$$

- **time_multiplier**:

$$\text{time\_multiplier} = \begin{cases} 1.0 & \text{(Morning, Afternoon)} \\ 0.9 & \text{(Evening)} \\ 0.7 & \text{(Night)} \end{cases}$$

Normalization:

$$\text{safety\_score}_{\text{normalized}} = 100 \times \frac{\text{safety\_score}_{\text{raw}}}{\max(100, \text{max\_total\_severity}/\text{max\_severity})}$$

if $\text{safety\_score}_{\text{raw}} > 10$.

Purpose: Penalizes high-severity crimes and hotspots, ensuring safer routes (e.g., Route B: $\sim$85–95%) score higher than unsafe ones (e.g., Route C: $\sim$20–40%).

3.5 Random Forest Regression

Technology: `sklearn.ensemble.RandomForestRegressor`.

Why Chosen: Handles non-linear feature interactions and is robust to overfitting.

Process:

- **Training Data**: 2000 synthetic routes with realistic crime distributions (via KDTree).
- **Model**: Random Forest with 100 trees (`n_estimators=100`).
- **Training**: Fits model to features X and safety scores y.
- **Prediction**:

$$\text{predicted\_score} = \text{model.predict}(\text{features})$$

- **Final Score**:

$$\text{final\_score} = 0.5 \times \text{safety\_score}_{\text{normalized}} + 0.5 \times \text{predicted\_score}$$

$$\text{safety\_score}_{\text{output}} = \text{round}(\max(10, \min(100, \text{final\_score}))/100, 2)$$

Purpose: Complements rule-based scores by learning complex patterns, contributing 50% to the final score.

3.6 Route Generation and Evaluation

Technology: OSRM API.
Process:

- Fetches 4 routes (1 primary + 3 alternatives) using OSRM.
- Generates additional routes via 3 waypoints if needed.
- Deduplicates and filters routes (distance $\leq 1.5 \times$ shortest).
- Evaluates 6–7 routes, ranking by safety score.

Purpose: Provides diverse route options for safety ranking.

3.7 Flask API

Technology: Flask with `flask_cors`.
Endpoints:

- `/evaluate_routes`: Returns ranked routes.
- `/load_crime_data`: Reloads dataset.

Purpose: Exposes the model for frontend integration.

4 Justification and Addressing Issues

Why These Technologies?

- **KDTree, DBSCAN**: Efficient for large-scale spatial queries and hotspot detection.
- **Random Forest**: Balances interpretability and predictive power.
- **Dynamic Normalization**: Prevents score clamping (e.g., 10%) by scaling penalties.
- **Rule-Based Scoring**: Explicitly prioritizes low-severity routes.

Addressing 10% Score Issue: Previous constant 10% scores were due to high penalties. Dynamic `max_crimes_per_route` and normalization ensure scores range from 20–95%.

5 Conclusion

The `safe_route_ml.py` model effectively combines spatial analysis, clustering, and supervised learning to rank routes by safety. Key formulas ensure low-severity, low-pattern routes are prioritized, addressing the requirement to select safer routes like Route B. The model is robust, interpretable, and scalable, with potential for further enhancements (e.g., temporal patterns, local OSRM).

References

[1] Scikit-learn: Machine Learning in Python, `https://scikit-learn.org`.

[2] Open Source Routing Machine, `http://project-osrm.org`.

[3] Geopy: Geocoding and Distance Calculations, `https://geopy.readthedocs.io`.