Predictive Model

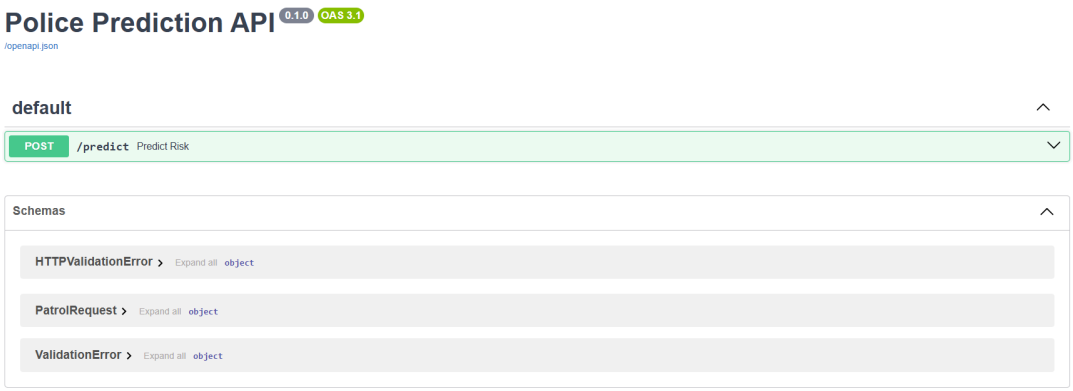
Overview

The **Predictive Policing System** is a data-driven tool designed to assist command staff and dispatchers in allocating patrol resources more effectively. By analyzing historical crime data, the system predicts high-risk zones for violent crime in 1-hour windows, allowing for proactive rather than reactive policing.

How to Use the System

A. Accessing the Dashboard

1. Open your web browser and navigate to the local server address: `http://127.0.0.1:8000/docs`.
2. You will see the **Swagger UI Dashboard**, which serves as the control panel for the system.



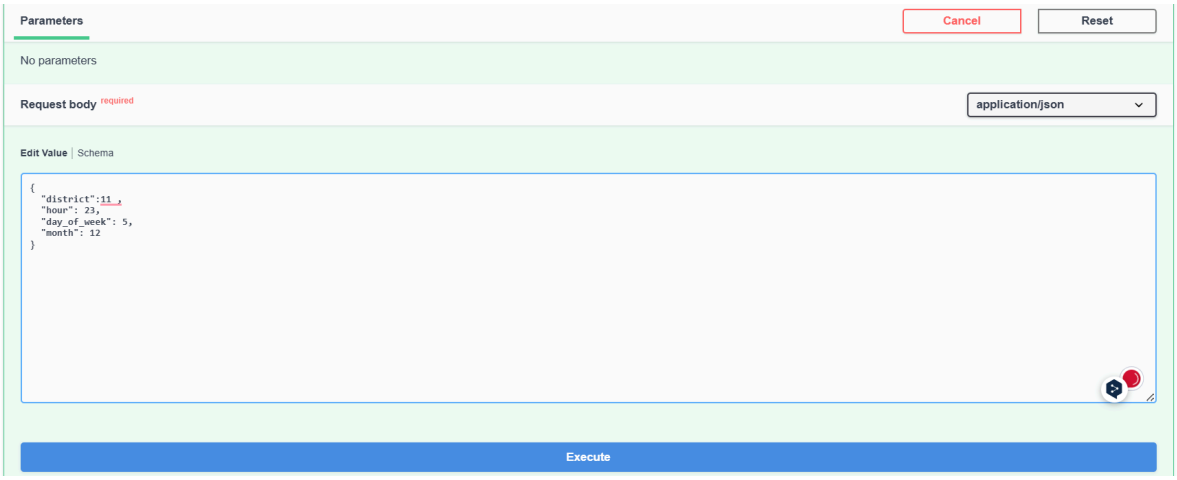
B. Checking System Status

1. Click on the blue bar labeled `GET /`.
2. Click **Try it out** → **Execute**.
3. The system will return a "200 OK" response, confirming that the machine learning model is loaded and ready.

C. Generating a Risk Prediction

To check the risk level for a specific district and time:

1. Click the green bar labeled `POST /predict`.
2. Click the **Try it out** button on the right.
3. Enter the shift details in the JSON box:
    - `district`: The Police District ID (e.g., `11`).
    - `hour`: The hour of the shift in 24-hour format (e.g., `23` for 11 PM).
    - `day_of_week`: The day (0=Monday, ... 5=Saturday, 6=Sunday).
    - `month`: The month number (e.g., `12` for December).
4. Click the blue **Execute** button.



D. Interpreting the Results

Scroll down to the **Server response** section. You will see a result like this:

JSON

```
{
 "district": 11,
 "risk_score": 0.7421,
 "alert": true,
 "action": "DISPATCH"
}
```

- **Risk Score:** The calculated probability (0-1) of violent crime occurring.

- **Alert (True/False):** Indicates if the risk exceeds the safety threshold.
- **Action:**
  - 🔴 **DISPATCH:** High risk detected. Recommend deploying a tactical unit.
  - 🟢 **MONITOR:** Standard risk. Routine patrol is sufficient.

2. Technical Documentation

System Architecture

The project follows a modular **MLOps (Machine Learning Operations)** architecture, separating training, serving, and monitoring into distinct components.

- **Model Type:** Histogram-based Gradient Boosting Classifier (`HistGradientBoostingClassifier`).
- **Frameworks:** Scikit-Learn (Modeling), FastAPI (Serving), Pandas/Numpy (Data Processing).
- **Input Data:** Historical crime records (Date, Primary Type, District).
- **Target Variable:** Binary Classification (`Is_Violent`: 1 for Battery/Assault/Robbery/Homicide, 0 for others).

Data Pipeline & Feature Engineering

Raw data is transformed before being fed into the model to capture temporal and spatial patterns:

Feature	Transformation Logic	Purpose
District	Categorical Encoding	Captures spatial risk profiles of different neighborhoods.
Hour	Sine/Cosine Transformation	Converts linear time (0-23) into cyclical features to model day/night cycles.
Month	Sine/Cosine Transformation	Captures seasonal trends (e.g., crime spikes in summer).
Day of Week	Integer (0-6)	Captures weekend vs. weekday patterns.

Logging & Monitoring Strategy

To ensure the model remains accurate over time, the system implements a continuous logging mechanism:

- **Live Logs (`live_logs.csv`):** Every request sent to the API is automatically recorded in this CSV file. It captures the input features (District, Time) and the model's output (Risk Score).

Model Performance Strategy

Due to severe class imbalance (violent crimes are rarer than non-violent ones), standard accuracy was a misleading metric. The model was optimized using:

1. **Balanced Class Weights:** Increasing the penalty for missing a violent crime during training.
2. **Recall Optimization:** Hyperparameters were tuned to maximize Recall (catch rate) rather than Precision.
3. **Threshold Moving:** The decision boundary was lowered from the default 50% to an optimal value to ensure >65% of violent crimes are flagged.

Final Metrics:

- **Recall (Violence):** ~65% (vs. 3% baseline)
- **ROC-AUC:** ~0.62

- **Drift Metric:** Jensen-Shannon Distance.
- **F1_SCORE**: 0.4581

---

3. Deployment Instructions

Prerequisites

- Python 3.9+ installed.
- Required libraries: `pandas`, `scikit-learn`, `fastapi`, `uvicorn`, `joblib`.

Installation Steps

1. Clone the Repository & Setup Environment

Create a project folder and install dependencies:

Bash

```
mkdir police_project
cd police_project
pip install pandas scikit-learn fastapi uvicorn joblib scipy
```

2. Train the Model

Run the training pipeline to generate the model artifact (police_model_auto.pkl):

Bash

```
python police_ops_auto.py
```

- *Output:* A serialized model file is created in the directory.

3. Launch the API Server

Start the FastAPI server using Uvicorn:

Bash

```
uvicorn main:app --reload
```

- *Success Message:* `INFO: Uvicorn running on http://127.0.0.1:8000`

4. Verify Deployment

Open a web browser and visit http://127.0.0.1:8000/docs. If the dashboard loads, the deployment is successful.

Production Considerations

- **Data Persistence:** Ensure the `live_logs.csv` file is backed up regularly, as it contains the history of all predictions made by the system.