Real-Time Traffic Accident Severity Prediction Using Machine Learning

Project Overview

This project aims to develop a real-time traffic accident severity prediction system using ensemble machine learning techniques. Students will work on the entire project pipeline, from data collection to model deployment, completing the project within a one-month timeframe.

Learning Objectives

- Implement the full data science workflow from data collection to deployment
- Apply ensemble machine learning techniques to solve complex classification problems
- Develop skills in data preprocessing, feature engineering, and model evaluation
- Gain experience with visualization tools like Plotly for data exploration and results presentation
- Learn experiment tracking using both Comet ML and MLflow for model monitoring and comparison
- Create a web application using Flask for real-time predictions
- Integrate external APIs for real-time weather and traffic data

Project Timeline (4 Weeks)

Week 1: Data Collection, Preprocessing, and Exploration

- Source and collect traffic accident data from public repositories
- Integrate free APIs for weather and traffic data
- Perform exploratory data analysis (EDA) using Plotly
- Clean and preprocess the dataset

Week 2: Feature Engineering and Model Development

- Create domain-specific features incorporating API data
- Implement and compare multiple classification algorithms
- Set up experiment tracking with both Comet ML and MLflow
- Begin model training and initial evaluation

Week 3: Model Optimization and System Design

- Perform hyperparameter tuning and ensemble modeling
- Create HLD and LLD documentation
- Evaluate models using appropriate metrics
- Develop API integration components

Week 4: Web Application Development and Deployment

- Build the Flask web application with Plotly visualizations
- Implement real-time prediction capabilities using API data
- Deploy the application
- Prepare and deliver final presentations

Project Description

Problem Statement

Traffic accidents pose significant risks to public safety and create substantial economic burdens. Predicting the severity of accidents in real-time can help emergency services allocate resources effectively and potentially save lives. This project challenges students to develop a machine learning system that can accurately predict the severity of traffic accidents based on various factors, including real-time weather and traffic conditions obtained from free APIs.

Data Sources

1. **Historical Accident Data**:
    - NHTSA Crash Report Sampling System
    - UK Department for Transport STATS19 dataset
    - Kaggle traffic accident datasets
2. **API Integrations**:
    - **Weather APIs**: OpenWeatherMap, Weather.gov, Weatherbit.io
    - **Traffic APIs**: HERE Traffic API, TomTom Traffic API, OpenStreetMap
    - **Geolocation APIs**: Google Maps Geocoding API (free tier), OpenStreetMap Nominatim

Technical Requirements

Technologies to Use

1. **Programming Language**: Python
2. **Data Manipulation**: Pandas, NumPy
3. **Machine Learning**: Scikit-learn, XGBoost, LightGBM
4. **Visualization**: Plotly, Matplotlib, Seaborn

5. **Experiment Tracking**: Comet ML AND MLflow (students should compare both)
6. **Web Development**: Flask, HTML, CSS, JavaScript
7. **Version Control**: Git
8. **API Integration**: Requests, Flask-RESTful

API Implementation Requirements

1. **Weather Data Integration**:
    - Implement modules to fetch real-time weather data
    - Create a caching mechanism to reduce API calls
    - Develop fallback strategies for API failures
2. **Traffic Data Integration**:
    - Create components to fetch real-time traffic conditions
    - Implement geospatial processing for traffic data
    - Develop a system to correlate traffic conditions with accident risk
3. **Combined Data Pipeline**:
    - Design a system to merge historical data with real-time API data
    - Implement feature engineering based on combined data sources
    - Create a robust data validation system for API responses

Detailed Component Specifications

1. Data Collection and API Integration

- Create a data collection pipeline that combines historical accident data with real-time API data
- Implement API wrappers for weather and traffic APIs
- Develop error handling and rate limiting for API calls
- Create a data validation system for API responses

2. Feature Engineering with API Data

- Generate weather-related features from API data (precipitation intensity, visibility, temperature, etc.)
- Create traffic-related features (congestion level, road closures, construction zones)
- Develop temporal features that account for time-of-day patterns in traffic and weather
- Calculate derived features that combine multiple data sources

3. Experiment Tracking Comparison

- Set up both Comet ML and MLflow for parallel experiment tracking

- Compare the features, usability, and performance of both platforms
- Document the pros and cons of each platform for this specific use case
- Provide recommendations for which platform is better suited for different aspects of the project

4. Model Development

- Implement baseline models (Random Forest, Logistic Regression)
- Develop advanced models (Gradient Boosting, Neural Networks)
- Create ensemble methods that combine multiple models
- Implement model interpretation techniques (SHAP values, feature importance)

5. Real-time Prediction System

- Develop a system that can make predictions based on current conditions
- Implement a caching mechanism for API data to reduce latency
- Create a prediction pipeline that preprocesses real-time data
- Develop confidence scoring for predictions

6. Flask Web Application

- Create a responsive user interface with Bootstrap
- Implement interactive Plotly visualizations for data exploration
- Develop a real-time prediction interface
- Create a dashboard showing current conditions and predictions

High-Level Design (HLD) Framework

The HLD document should include:

1. **System Architecture Overview**:
    - Component diagram showing major system components
    - Data flow diagram illustrating how data moves through the system
    - Deployment diagram showing how components are deployed
2. **Component Descriptions**:
    - Data Collection Component
    - API Integration Component
    - Feature Engineering Pipeline
    - Model Training and Evaluation System
    - Prediction Service
    - Web Application

3. **Technology Stack**:
    - Detailed description of technologies used
    - Justification for technology choices
    - Integration points between different technologies
4. **Data Flow**:
    - Description of how data flows from sources to models to users
    - Data transformation processes
    - Data storage and retrieval mechanisms

Low-Level Design (LLD) Framework

The LLD document should include:

1. **Detailed Component Design**:
    - Class diagrams for major components
    - Sequence diagrams for key operations
    - Activity diagrams for complex processes
2. **Algorithm Specifications**:
    - Detailed descriptions of algorithms used
    - Pseudocode for complex algorithms
    - Optimization techniques
3. **Data Structures**:
    - Detailed specification of data structures
    - Database schema designs
    - API request/response formats
4. **Interface Definitions**:
    - API endpoint specifications
    - User interface wireframes
    - Component interface definitions

Evaluation Criteria

1. **Technical Implementation (40%)**
    - Quality and efficiency of code
    - Appropriate use of machine learning techniques
    - Model performance and accuracy

    - API integration and error handling
2. **Documentation (20%)**
    - Quality of HLD and LLD documents
    - Code documentation and comments
    - API documentation
    - User documentation
3. **Web Application (20%)**
    - Functionality and user experience
    - Real-time prediction capabilities
    - Visualization quality
    - Error handling and robustness
4. **Presentation and Demo (20%)**
    - Clarity of explanation
    - Quality of visual aids
    - Demonstration of system capabilities
    - Handling of questions

Deliverables

1. **Code Repository**:
    - Well-organized GitHub repository
    - Clear README with setup instructions
    - Properly documented code
2. **Documentation**:
    - HLD and LLD documents
    - API documentation
    - User manual for web application
3. **Trained Models**:
    - Serialized model files
    - Training scripts
    - Evaluation reports
4. **Web Application**:
    - Deployed Flask application
    - Source code for web application
    - Setup instructions
5. **Presentation**:

- Slides for final presentation
- Demo script
- Project report

Additional Resources

Free APIs for Weather and Traffic Data

1. **Weather APIs**:
    - OpenWeatherMap (Free tier: 60 calls/minute, 1,000,000 calls/month)
    - Weather.gov (Completely free, US only)
    - Weatherbit.io (Free tier: 50 calls/day)
2. **Traffic APIs**:
    - HERE Traffic API (Free tier: 250,000 transactions/month)
    - TomTom Traffic API (Free tier: 2,500 calls/day)
    - OpenStreetMap (Free with attribution)
3. **Geolocation APIs**:
    - Google Maps Geocoding API (Free tier available)
    - OpenStreetMap Nominatim (Free with usage limitations)

Sample Data Sources

1. US NHTSA Crash Report Sampling System: https://www.nhtsa.gov/crash-data-systems/crash-report-sampling-system
2. UK STATS19 collision data: https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data
3. Kaggle: https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents