Project Plan (v2): Predicting Accident Severity Using the US Accidents Dataset

GitHub - [MukulRay1603/US_ACCIDENT_STATS: Data analytics for the accident data in the US (github.com)]

Team

1. Mukul Rayana
2. Jayasurya
3. Srutileka S
4. Sai Vikram Karna

Current process – Data Cleaning.

- Given the size of the data is 7.7M, we brought it down to 6M and yet it seems overkill, so we decided on data sampling for now.
- Additionally, we are working on incremental learning which seemed feasible using XGBoost Model that might come handy for memory optimisation.
    - We can train 1M data values in one run followed by the next set of 1M (Drawback time consuming and requires trial and error)

1. Introduction

The objective of this project is to develop a predictive model that determines the severity of road accidents using the US Accidents dataset. The dataset provides extensive information on accident circumstances, including time, location, weather conditions, and traffic signals.

Updated Goals: Following the new suggestions:

- Implement a **time integrity check** by subtracting Start_Time from End_Time.
- Use a **Folium heatmap** to identify accident-prone areas and geographical anomalies.
- Incorporate **additional feature engineering** based on our EDA findings.
- Handle **imbalanced data** appropriately to ensure our model's accuracy and reliability.

Plan for Handling Imbalanced Data

Given that our target variable (Severity) is a multi-class classification problem, we need to handle potential class imbalance to ensure the model predicts all severity levels accurately.

1. Analyzing Class Distribution

- First, we will examine the distribution of severity levels using a count plot.
- If there is a **significant imbalance** (e.g., one class >70% of the data), we will need to apply techniques to handle it. If the imbalance is **moderate** (worst case around 7:3), class weighting might suffice.

2. Approaches to Handle Imbalance

2.1. Oversampling Minority Classes

- **Techniques**: We will use **Random Oversampling** or **SMOTE (Synthetic Minority Oversampling Technique)** to balance the dataset by creating synthetic samples for the underrepresented classes.

- **Implementation**: Apply SMOTE to the training set to generate synthetic data for minority classes, reducing model bias.
    - **Pros**: Improves sensitivity to minority classes, enhancing prediction of less common but critical outcomes.
    - **Cons**: Risk of overfitting, especially if using simple oversampling.

2.2. Class Weighting

- If the imbalance is moderate, we will use **class weights** in our models (e.g., Random Forest, XGBoost) to penalize misclassifications of minority classes.
- **Implementation**: Compute class weights and assign them in the model's training process.
    - **Pros**: Adjusts for imbalance without altering the dataset size; reduces overfitting risk.
    - **Cons**: Less effective for extreme imbalance cases.

2.3. No Imbalance Handling if EDA was successfully balanced

- If the class distribution shows **mild imbalance** (worst-case around 7:3), modern models like Random Forest can handle this using **class weighting** alone, without additional oversampling.

2. Target Outcome

The target variable for this project is:

- **Severity**: A categorical variable with values ranging from 1 to 4, where 1 indicates a minor accident and 4 indicates a severe accident.

3. Features to Use for Prediction

Based on our initial EDA and cleaning process, the following features will be used for the final project:

- **Start_Time**: To extract **Hour**, **Day of Week**, **Month**, **Duration**, and **Weekend** flags.
- **Temperature(F)**: To assess weather impact on accidents.
- **Visibility(mi)**: Visibility conditions at the time of the accident.
- **Wind_Speed(mph)**: Speed of the wind, which may affect driving conditions.
- **Weather_Condition**: Aggregated into broader categories (Clear, Precipitation, Foggy, etc.).
- **Start_Lat** and **Start_Lng**: To create a **Folium heatmap** and potentially cluster accident hotspots using KMeans.
- **Traffic_Signal**: To flag accidents occurring near traffic signals.
- **Duration**: Calculated from Start_Time and End_Time to understand the length of each accident.
- **Bad_Weather**: A Boolean flag indicating poor weather conditions (rain, snow, fog).
- **Rush_Hour**: To identify accidents that occurred during peak traffic times.

The exact feature set may evolve during the project as new insights are gained from EDA and feature engineering.

4. Feature Engineering

Here is our initial plan for feature engineering to enhance our model's predictive power:

4.1. Time Integrity Check

- **Why**: To ensure data quality, we'll subtract Start_Time from End_Time to identify cases where **an event ended before it started**. If found, these rows will be removed or cleaned.
- **Action**: Calculate Duration for each accident in hours. Check for negative durations and handle any anomalies.

4.2. Time-Based Features

- **Hour**, **Day of Week**, and **Month**: Extracted from the Start_Time column to identify time-based patterns.
- **Rush Hour**: A Boolean flag to indicate if the accident occurred during peak hours (7-9 AM, 4-6 PM).
- **Weekend**: A Boolean flag to identify if the accident occurred on a weekend (Saturday or Sunday).

4.3. Weather-Based Features

- **Bad Weather**: A Boolean flag indicating poor weather conditions. This will be derived from the Weather_Condition column and will include rainy, snowy, foggy, and stormy conditions.
- **Weather Category**: Aggregate Weather_Condition into broader categories like **Clear**, **Precipitation**, **Foggy**, etc., to simplify the weather feature.

4.4. Geospatial Features

- **Folium Heatmap**: We'll create a heatmap using Start_Lat and Start_Lng to visualize accident density and identify geographical anomalies.
- **KMeans Clustering**: Using the latitude and longitude data, we'll create a new Location_Cluster feature to identify high-risk areas.

4.5. Traffic Signal Proximity

- **Near Traffic Signal**: Create a Boolean feature from the Traffic_Signal column to flag accidents that occurred near traffic lights.

5. Handling Imbalanced Data

Since **Severity** is our target variable and is a multi-class classification problem, we need to handle potential class imbalance to avoid biased predictions.

5.1. Analyzing Class Distribution

- We will first analyze the distribution of severity levels in the dataset. If we find that certain severity levels are underrepresented (e.g., Severity 4), we will consider applying techniques to handle the imbalance.

5.2. Approaches to Handle Imbalance

1. **Oversampling**: If the dataset is significantly imbalanced, we will oversample the minority classes (e.g., Severity 3 and 4) using **Random Oversampling** or **SMOTE (Synthetic Minority Over-sampling Technique)** to create a balanced dataset.
2. **Class Weighting**: Alternatively, we can assign **class weights** in our machine learning models (e.g., Random Forest, XGBoost) to penalize the model for misclassifying underrepresented classes.
3. **Decision**: We will decide on the final approach based on the class distribution observed during the initial analysis. If the imbalance is mild (e.g., worst-case scenario being 7:3), we may choose to skip oversampling and rely on class weighting.

6. Exploratory Data Analysis (EDA)

- **Accident Severity Distribution**: Visualize the distribution of severity levels to assess the degree of imbalance.
- **Time-Based Analysis**: Use Hour, Day of Week, Month, and Rush Hour features to explore accident patterns.
- **Weather Analysis**: Examine the impact of weather conditions and visibility on accident severity.
- **Geospatial Analysis**:
    - **Folium Heatmap** to identify accident hotspots and flag any anomalies in geographical coordinates.
    - **KMeans Clustering** to create Location_Cluster and explore the severity distribution within these clusters.

7. Model Building and Evaluation

- **Model Choice**: We will start with a **Random Forest Classifier** due to its robustness and ability to handle categorical and numerical data. We may explore **XGBoost** and **LightGBM** if time permits.
- **Evaluation Metrics**: Use **Accuracy**, **Precision**, **Recall**, and **F1-score** to evaluate the model. We'll also use a **confusion matrix** to analyze class-wise performance, particularly for underrepresented classes.

Resources

XGBoost (For Advanced Modeling) - XGBoost Documentation

Folium Documentation - Folium — Folium 0.16.1.dev76+g2921126e documentation (python-visualization.github.io)

Geographical Heatmap Guide: Plotting Geographical Heatmaps Using Python Folium

Incremental Learning with partial fit - Scikit-Learn - Incremental Learning for Large Datasets (coderzcolumn.com)