Predicting Impact of Road Accidents on Traffic Flow

Tsu-Hao Fu
Purdue University
West Lafayette, IN, USA
fu390@purdue.edu

Koushiki Basu
Purdue University
West Lafayette, IN, USA
kbasu@purdue.edu

Yohan Berg
Purdue University
West Lafayette, IN, USA
yberg@purdue.edu

Gopireddy Avinash Reddy
Purdue University
West Lafayette, IN, USA
gavinash@purdue.edu

Abstract

The project will address a pressing concern in urban transportation management, as predicting traffic accidents' impact on traffic flow is crucial for enhancing road safety and minimizing congestion. By employing advanced machine learning techniques and thorough data analysis, we aim to develop a predictive model capable of accurately forecasting real-time traffic disruptions. Such forecasts can empower authorities to implement proactive measures, improve emergency response times, and optimize traffic management strategies. Additionally, the study will contribute to the broader goal of enhancing overall road safety and efficiency, ultimately benefiting commuters, businesses, and communities alike.

ACM Reference Format:
Tsu-Hao Fu, Koushiki Basu, Yohan Berg, and Gopireddy Avinash Reddy. 2024. Predicting Impact of Road Accidents on Traffic Flow. In *Proceedings of CS 57300 Data Minig (Project Report).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In 2019, urban U.S. commuters were delayed an average 54 hours in traffic [1]. A large cause of these delays are traffic accidents, which can increase congestion and slow down traffic flow [2]. Accidents represent an unavoidable aspect of daily life, affecting people, families, and entire communities throughout the United States. It's critical to grasp the reasons, trends, and outcomes of accidents to enhance public safety, develop transportation strategies, and inform policymaking. This report explores the realm of accidents, with a particular emphasis on those that take place on U.S. roads and highways. Being able to predict traffic accidents and their impact on traffic flow in real time would allow authorities and navigation services to reroute traffic, reducing congestion, compounding traffic risk, and emergency response times.

Traffic flow analysis presents several challenges: large and complex data, noise, and real-time changes in conditions. To address these challenges, machine learning has been applied to multiple facets of traffic analysis including accident prediction [3] and Navigation [4]. Yet with many different data preparation and machine learning algorithms available, it can be difficult to design a machine learning problem and model that will perform well within specified requirements. Thus, we suggest a project to analyze several data mining techniques and algorithms in the context of traffic accidents' effect on traffic flow in an attempt to create performant models and identify distinct advantages and disadvantages between techniques and algorithms.

2 Literature Review

Accident analysis and prediction has been the topic of many research during the past few decades. This field of research delves into understanding how various environmental factors, such as weather conditions, traffic flow dynamics, and road network characteristics, influence the likelihood or severity of traffic accidents. Exploring the effects of weather elements, like precipitation, on road accidents [14, 15, 17, 18], employing data mining methodologies to uncover association rules for causality analysis [7, 11], and conducting statistical analyses to examine the impact of unobserved variables (e.g., missing data) on accident severity [16] are typical endeavors in this domain. While these studies often yield valuable insights, they may not always be directly applicable to real-time accident prediction and planning.

Over the past few decades, there has been a considerable focus on accident analysis and prediction in research circles. However, much of this research has been based on small-scale datasets with limited coverage, typically encompassing only a few road segments or a single city [7–13]. For instance,

Project Report, April 2024, West Lafayette, IN

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Kumar et al. [11] employed data mining techniques to extract association rules for causality analysis from a small-scale dataset. Similarly, Chang et al. [9] utilized information such as road geometry, annual average daily traffic, and weather data to predict accident frequency on a highway road segment using a neural network model. Likewise, Wenqi et al. [13] employed a convolutional neural network model for accident prediction on a specific road segment. Although these studies offer intriguing insights, their findings may be limited in applicability and generalizability due to the constrained scale of the datasets utilized.

3 Exploratory Data Analysis

3.1 Dataset

For our primary dataset, we have chosen the US Accidents dataset *US Accidents* [5] available on Kaggle. This extensive dataset provides comprehensive coverage of traffic accidents across 49 states of the USA, spanning a timeframe from February 2016 to March 2023. It aggregates data from various sources, including prominent APIs such as "MapQuest Traffic" and "Microsoft Bing Map Traffic." From this vast dataset, we have selected a representative sample comprising 500,000 entries out of approximately 7.7 million records. This subset encompasses 46 distinct features, capturing crucial aspects such as accident severity, duration of traffic flow disruption, prevailing weather conditions, geographical location, and surrounding traffic infrastructure near the accident sites. Detailed descriptions of all variables can be found in Table 1 and Table 2.

The US-Accidents dataset presents a wealth of opportunities for diverse applications, ranging from real-time accident prediction to in-depth analysis of accident hotspots and casualty patterns. Researchers can leverage this rich dataset to extract cause-and-effect rules that enable the prediction of accidents, contributing to enhanced safety measures and accident prevention strategies. Furthermore, the dataset offers a unique opportunity to investigate the influence of environmental factors such as precipitation on accident occurrence, providing valuable insights into the interplay between weather conditions and traffic safety. Given the extensive scale of the dataset, researchers have the potential to derive a myriad of insights that can inform various fields, including urban planning and the enhancement of transportation infrastructures. By harnessing the comprehensive data available in the US-Accidents dataset, stakeholders can make informed decisions and implement targeted interventions to mitigate accident risks and improve overall safety on the roadways.

3.2 Visualization

We performed some basic data analysis and visualization before the preprocessing of the data. Figure 1 shows the distribution of accidents in different cities in the US, and in

Table 1. Definition of Point-Of-Interest (POI) annotation tags

Amenity	Refers to particular places such as restaurants, libraries, colleges, bars, etc.
Bump	Refers to speed bump or hump to reduce the speed.
Crossing	Refers to any crossing across roads for pedestrians, cyclists, etc.
Give-way	A sign on road which shows priority of passing.
Junction	Refers to any highway ramp, exit, or entrance.
No-exit	Indicates there is no possibility to travel further by any transport mode along a formal path or route.
Railway	Indicates the presence of railways.
Roundabout	Refers to a circular road junction.
Station	Refers to public transportation station (bus, metro, etc.)
Stop	Refers to stop sign.
Traffic Calming	Refers to any means for slowing down traffic speed.
Traffic Signal	Refers to traffic signal on intersections.
Turning Loop	Indicates a widened area of a highway with a non-traversable island for turning around.

Table 2. Details of attributes

Traffic Attributes	ID, source, severity, start_time, end_time, start_lattitude, end_lattitude, start_longitude, end_longitude, distance, and description
Address Attributes	street, city, county, state, zipcode, country, timezone, and airport_code
Weather Attributes	time, temperature, wind_chill, humidity, pressure, visibility, wind_direction, wind_speed, precipitation, and condition (e.g., rain, snow, etc.)
POI Attributes	amenity, bump, crossing, give-way, junction, no-exit, railway, roundabout, station, stop, traffic calming, traffic signal, and turning loop
Period-of-Day Attributes	Sunrise/Sunset, Civil Twilight, Nautical Twilight, and Astronomical Twilight

this plot, we have listed 20 cities with the highest cases of accidents.

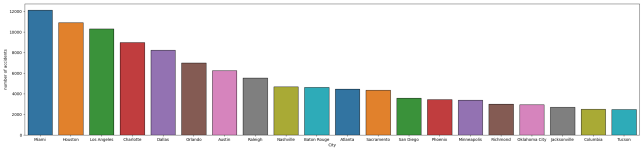

Figure 1. Distribution of Accidents across US cities

Figure 2 shows the distribution of accidents with respect to severity, and we observed that there is a class imbalance in the dataset as the majority of the data entries are labeled as severity level 1.

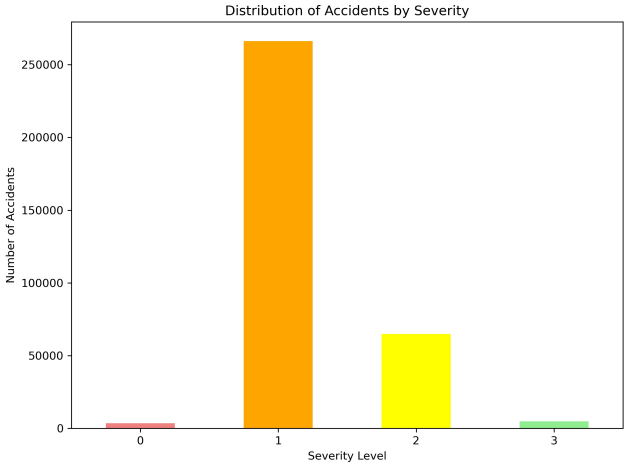

Figure 2. Distribution of Accident Severity

Figure 3 shows the distribution of accidents over the month, which shows more accidents occur during winter when the weather conditions are harsh. Figure 4 and Figure 5 show that significantly more accidents were observed during the weekdays than on weekends and mainly during office hours.

3.3 Correlation Map After Preprocessing

The correlation matrix 6 represents the relationships between various factors relating to traffic accidents. Below are key insights: Source label encoded: It is negatively correlated with both Accident Duration(min) standardized (-0.446832) and Distance(mi) standardized (-0.512031), which might suggest that different sources report shorter durations or cover shorter distances in their reporting.

Visibility(mi) standardized and Weather Condition label encoded: A strong negative correlation (-0.550688) suggests that weather conditions have a significant impact on visibility, which is intuitive.

Figure 3. Distribution of Accident over Months

Traffic Signal label encoded with Crossing label encoded: A positive correlation (0.458415) indicates these two are commonly associated, likely because traffic signals are often present at or near pedestrian crossings.

Time of Day: Variables like Sunrise Sunset label encoded, Civil Twilight label encoded, Nautical Twilight label encoded, and Astronomical Twilight label encoded have strong positive correlations with each other, as they all relate to the time of day.

Geographical factors: City-frequency-encoded, County-frequency-encoded, and Zipcode-frequency-encoded show strong positive correlations with each other (ranging from 0.58 to 0.65), reflecting the nested nature of these geographical units.

4 Data Preprocessing

Looking at the distribution of values of the “target” column, almost 80% of the labels are 2, leaving 19% as 3, 1.5% as 4, and 1% as 1. This clearly indicates that the dataset is imbalanced and thus we should keep this in mind while preprocessing and model selection. This also means that we should not only use accuracy but also apply precision, recall, and f1-score as our evaluation metrics.

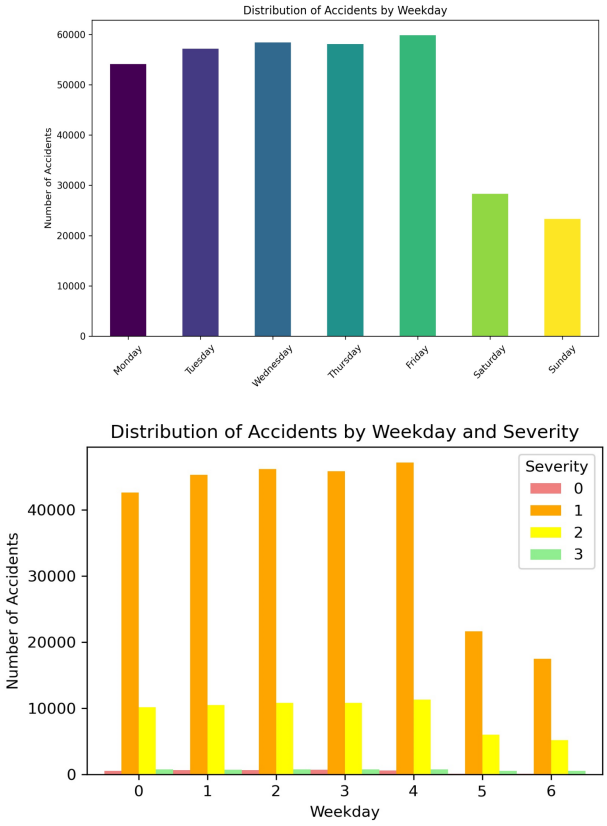

Figure 4. Distribution of Accident throughout Weekday

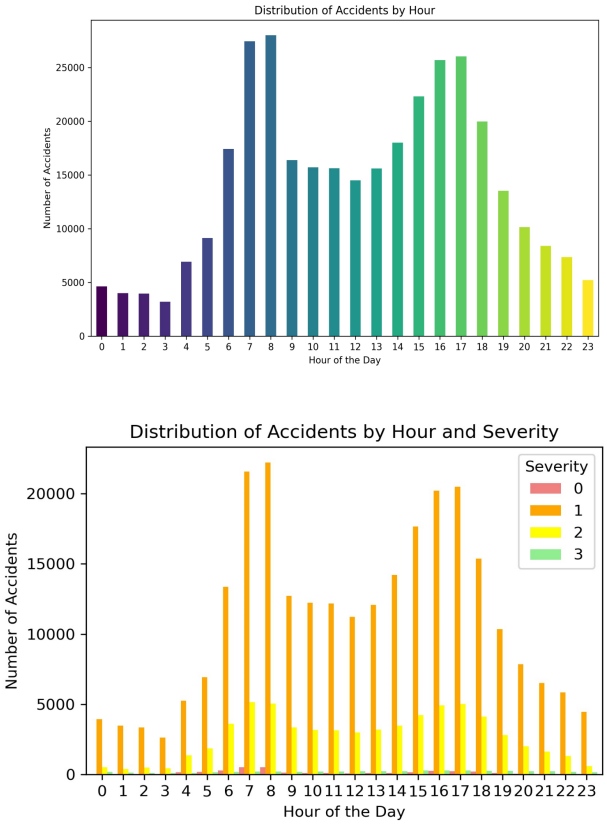

Figure 5. Distribution of Accident over Hours

4.1 Missing Value

Data loss may also be caused in the process of data recording and transmission, so data loss is inevitable. Similarly, there is missing data in this data set, which is shown in Table 2. The missing rate of End Lat, End Lng is almost 50%. If they are filled in, large errors will be introduced, so these two variables are discarded. Therefore, we only use Start Lat and Start Lng and change their names into "Latitude" and "Longitude". For all the location features such as City, Street, Zipcode, Timezone, and Airport Code, as well as the day/night indicators, we will straightforwardly drop all remaining missing values.

By the Correlation Map 6, we can see that Wind Chill and Temperature are highly correlated. Therefore, we replace missing temperature data with wind chill and drop Wind Chill. We apply median imputation to numerical weather columns (Temperature, Humidity, Pressure, Visibility, Wind Speed, and Precipitation grouping the data by Airport Code and Month to account for local and seasonal variations. Furthermore, we implement mode imputation for categorical variables like Wind Direction and Weather Condition, applying a similar grouping strategy to replace missing entries with the most common value within each group. After imputations, both sections include a step to drop any rows that still contain missing values in the imputed columns, thereby ensuring that the dataset is complete for subsequent analysis.

4.2 Outlier

Outliers refer to observations that clearly deviate from the rest of the observations. We use the classic interquartile range (IQR) method to detect outliers, some abnormal points cannot be properly processed. Some numerical columns (Distance, Temperature, Pressure, and WindSpeed were processed.

4.3 Standardization

Standardization is a crucial preprocessing technique used in machine learning to transform numerical data so that it has a mean of 0 and a standard deviation of 1. This technique is particularly important for models that rely on the magnitude of variables, such as support vector machines (SVMs) and k-nearest neighbors (KNN). We apply it to the numerical variables, such as Accident Duration, Distance, Temperature, Humidity, Pressure, Visibility, Wind Speed, and Precipitation.

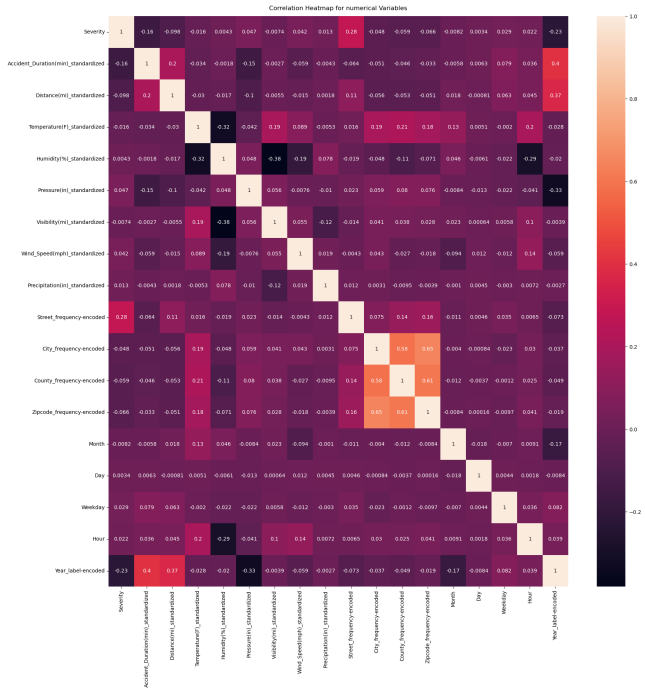

Figure 6. Correlation Heatmap

5 Feature Engineering

Feature engineering is a critical step in the process of data analysis and machine learning, where we transform raw data into meaningful and informative features that can enhance the performance of predictive models. In the context of the US Accidents dataset, effective feature engineering can significantly improve our understanding of accident patterns, help us build more accurate predictive models, and ultimately contribute to safer roads and transportation systems.

Dropping unnecessary columns during feature engineering is a strategy to improve the quality of your data and the efficiency of your modeling process. Description, ID, Country, Amenity, Bump, Railway, Give Way, No Exit, Roundabout, Traffic Calming, Turning Loop, and Airport Code will be dropped because these columns will not provide anything informative to the model

5.1 Binning

Since the factors related to Wind direction and Weather conditions are too complex, the prediction of these two variables by available variables cannot achieve ideal results. Similarly, for better filling, we divide the Weather conditions variables into sunny, cloudy, overcast, rain, snow, haze, fog, sand, smoke, ice pellets, hail, rain, and snow. Wind direction is divided into north, east, south, west, calm, and variable.

5.2 Time Extraction

While DateTime information is essential for many analyses, it often contains rich contextual details that can be challenging for machine learning models to interpret directly. To address this, feature extraction from the Start time of accident involves breaking down these timestamps into more understandable and informative components, including Year, Month, Day, Weekday, and Hour. Additionally, we derive a new feature, Accident Duration, by subtracting the Start Time from the End Time. Given that further insights from the End Time and Weather Timestamp are unnecessary for our analysis, these features will be excluded.

5.3 Label Encoding

Label encoding is a common technique in feature engineering that transforms categorical data into numerical values. It assigns a unique integer to each category or label within a categorical feature, thereby converting it into a format that machine learning algorithms can readily understand and process. We will apply it to most of the categorical variables, such as Source, State, Timezone, Wind Direction, Weather Condition, Crossing, Junction, Station, Stop,

5.4 Frequency Encoding

Frequency encoding is a technique used to handle categorical variables by replacing each category with the frequency (or count) of that category in the dataset. This method converts categories into a numerical format that can be used by machine learning algorithms, and it provides a way to retain the importance of a category based on how often it appears. We apply it to the categorical with large number of unique values, such as Street, City, County, Zipcode.

6 Methodology

6.1 Linear Models

One of the most basic types of models used in machine learning for regression and classification tasks is the linear model. The fundamental assumption of linear models is that the relationship between the input features and the target variable can be represented by a linear function. For classification tasks, linear models employ a decision boundary, which can be linear or hyperplane in higher dimensions, to separate different classes.

6.1.1 Logistic Regression. Logistic regression model estimates the probability that a given input belongs to a particular class. Although logistic regression is a binary classification algorithm, it may be extended to handle multiclass classification problems through various techniques, the most common being one-vs-rest (OvR) and multinomial (also known as softmax) approaches. The OvR approach involves training a different binary logistic regression classifier for every class. For each class, we treat it as the positive class

and group all other classes as the negative class. The multinomial approach involves training a single logistic regression model with multiple output classes. The softmax function takes the raw output scores from the model, converts them into probabilities and outputs a probability distribution over all classes. In both approaches, we choose the class with the highest probability during prediction.

Both approaches have their advantages and disadvantages. OvR is simpler and more interpretable, but it is susceptible to overlapping decision boundaries and imbalanced class distributions. Multinomial, on the other hand, directly models the joint probability of all classes and can learn more complex decision boundaries. However, it requires more computational resources and may be less interpretable.

6.1.2 Stochastic Gradient Descent (SGD). Stochastic Gradient Descent is a popular optimization algorithm used in machine learning for training models, particularly in large-scale. Gradient descent is an optimization algorithm used to minimize the loss function of a model by adjusting its parameters iteratively. It computes the gradient of the loss function with respect to the parameters and updates the parameters in the opposite direction of the gradient. In traditional gradient descent, the gradient is computed using the entire training dataset, which can be computationally expensive, especially for large datasets. In contrast, SGD computes the gradient using only a single or a small subset of training examples (mini-batch) at each iteration.

SGD is computationally efficient and can handle large datasets. While SGD converges to a minimum of the loss function, it may not reach the global minimum, especially in non-convex optimization problems. However, in practice, SGD often finds good solutions, and techniques like learning rate scheduling and momentum can help improve convergence. One of the key hyperparameters in SGD is the learning rate, which controls the size of the parameter updates. A high learning rate can cause the algorithm to overshoot the minimum, while a low learning rate can slow down convergence.

6.2 KNN

k nearest neighbors (k-NN) algorithm is a popular non-linear machine learning algorithm used for classification tasks. It belongs to the family of instance-based or memory-based learning algorithms. Classification is computed from a simple majority vote of the *k* closest training data points (neighbors) in the feature space and assigns the majority class label among them to the new data point. The most important hyperparameter in k-NN is the value of k, which determines the number of neighbors to consider.

KNeighborsClassifier is commonly used for classification tasks when the decision boundary is complex and not easily separable by a linear function. It's simple to understand and implement, however, it may not scale well to large datasets due to its computational cost during prediction, as it requires calculating distances to all training data points.

6.3 Decision Tree

In machine learning, decision trees are simple, flexible models applied to regression and classification problems. Decision trees are based on the recursive partitioning of data according to feature values, producing a predictive model visualized as a tree. At the root node, the complete dataset is first processed, and the algorithm chooses the feature that best divides the data into homogenous subsets. Iteratively, this splitting process is carried out, with each leaf node denoting a final prediction or decision and each internal node reflecting a decision based on a feature. The goal is to create a tree that maximally reduces uncertainty or impurity in the data at each step, resulting in subsets that are as homogeneous as possible with respect to the target variable.

The method assesses various splitting criteria at every decision tree node to choose the optimal feature for splitting. This entails computing impurity metrics like entropy or Gini impurity. Entropy indicates the average amount of information required to classify an instance, whereas Gini impurity measures the probability of erroneously classifying a randomly selected element if it were labeled according to the distribution of labels in the node. The split produces more homogeneous subgroups concerning the target variable by selecting the feature that optimizes information gain or minimizes impurity.

Decision trees offer interpretability and versatility, managing numerical and categorical data without preprocessing. They're easily understood and visualized, beneficial for tasks requiring clarity. They're resilient to outliers and missing values, capturing nonlinear relationships effectively, and rendering them applicable across various domains.

6.4 Ensemble

Ensemble learning synthesizes multiple-base learners through certain combination strategies to obtain the final result. According to the way of ensemble, ensemble learning can be divided into parallel integration (Bagging) and serial integration (Boosting).

6.4.1 Bagging. In machine learning, improving predictive performance while maintaining model robustness is a perpetual pursuit. Ensemble learning techniques, which combine multiple models to achieve better results than any individual model, have gained considerable attention. One such technique is Bagging, short for Bootstrap Aggregating, which was introduced by Leo Breiman in 1996.

Bagging operates through a multi-step process that harnesses the power of randomness and aggregation. It begins by generating multiple subsets of the original training data through bootstrap sampling, randomly selecting instances with replacement. This results in diverse subsets, each containing variations of the original data. Base learners like decision trees or neural networks are then independently trained on these subsets, capturing unique patterns and relationships due to exposure to different data variations. This diversity among base learners is pivotal for bagging's success, enabling the ensemble to capture a broader range of data features and nuances. Following training, base learners predict on new data, with final predictions typically determined through aggregation methods like majority voting. This leverages collective decision-making, enhancing accuracy and reliability.

An inherent advantage of bagging lies in its ability to mitigate variance, making it particularly beneficial for models prone to overfitting. By training on diverse data subsets and aggregating predictions, it reduces noise impact and provides an unbiased performance estimate through out-of-bag instances, enhancing generalization and robustness evaluation. Additionally, its scalability and efficiency enable parallelized training, expediting model training through the simultaneous processing of different data subsets.

- Random Forest: Random Forest is an ensemble learning method that constructs multiple decision trees through bootstrap sampling and random feature selection. Each tree is trained independently on a subset of data and features, reducing the correlation between trees. Once trained, the trees' predictions are combined through majority voting for classification, resulting in the final prediction. Random Forest excels in handling high-dimensional data, is robust against overfitting, and offers scalability. Its ability to produce accurate and stable predictions makes it a popular choice in various machine learning tasks.

6.4.2 Boosting. In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these weak learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance.

In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. Such small trees, which are not very deep, are highly interpretable. Parameters like the number of trees or iterations, the rate at which the gradient boosting learns, and the depth of the tree, could be optimally selected through validation techniques like k-fold cross validation. Having a large number of trees might lead to overfitting. So, it is necessary to carefully choose the stopping criteria for boosting.

- XGBoost: eXtreme Gradient Boosting, is a sophisticated machine learning algorithm that enhances traditional gradient boosting methods by incorporating regularization to prevent overfitting, improving its generalization capabilities.
- LightGBM: Light Gradient Boosting Machine, is a highly efficient gradient boosting framework that stands out due to its speed and minimal memory usage. LightGBM optimizes performance through innovative techniques like Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), which streamline data processing and reduce dimensionality.
- CatBoost: CatBoost employs ordered boosting, a permutation driven alternative to the classic gradient boosting method, which significantly reduces overfitting and improves the model's accuracy.

6.4.3 Imbalanced Learning.

- Self-paced Ensemble (SPE): This technique focuses on iteratively re-weighting harder-to-classify examples, giving them more importance over time. It is particularly useful when dealing with highly imbalanced datasets. It starts with simple examples and gradually takes on more complex ones as the training progresses, thus refining the decision boundary over time.
- Balanced Bagging: Balanced Bagging works similarly to regular bagging by creating multiple subsets of the original data through bootstrap sampling and training base learners independently. In this technique, each base estimator (usually a decision tree) is trained on a different balanced subset created by under-sampling the majority class. The diverse and balanced training sets help to create a more generalized model.

6.4.4 Voting. A Voting Classifier is a machine learning model that combines multiple models to make predictions. Instead of using separate models, it uses an ensemble of models. It predicts the output class by either taking the majority vote of the predicted classes (Hard Voting) or by averaging the probabilities assigned to each class by the individual models (Soft Voting). Hard Voting selects the class with the most votes, while Soft Voting selects the class with the highest average probability. We combine the predictions from the best models we got from Ensemble Learning (Random Forest + XGBoost + LightGBM) and Imbalanced Learning (SPE + Balanced Bagging).

6.5 Neural Network

Neural Networks are a machine learning algorithm that can learn complex non-linear patterns in data by effectively stacking linear transformations accompanied with a non-linear activation functions. As layers are added, the model becomes more expressive, which allows it to learn increasingly complicated patterns and relationships, but also risks overfitting, where specific patterns found only in the training set are learned, proving to be meaningless in prediction at best or damaging performance at worst. Because of Neural Networks' tendency to overfit, further regularization techniques such as Batch Normalization and Dropout were employed to facilitate meaningful learning.

- Batch Normalization (BatchNorm): Neural Networks generally learn best when their inputs are normalized. This not only applies to the input layer, but for each layer after as well. Thus, the main strategy behind BatchNorm is to normalize the outputs of each intermediate layer before passing it to the next. This generally improves training speed and often has a regularizing effect.
- Dropout: One reason for overfitting in Neural Networks is the formation of "conspiracies," groups of neurons that learn specific spurrous patterns in the training data that fail to generalize to broader data. By randomly disabling neuron during training, the model cannot rely on conspiracies to reliably predict labels, and the layer as a whole must learn broader patterns that are more likely to generalize into testing data.

Neural Networks are also known to be sensitive to highly imbalanced data, which is present in our dataset. We explore undersampling as a technique to attempt to help the model learn to predict rare labels. The model itself was trained with 4 layers total including the input and outputs layers with ReLU as the intermediate activation function and Softmax as the output activation function. BatchNorm and Dropout were both applied to the 2nd and 3rd layers and the hyperparameters were tuned using K-fold cross validation.

7 Experiment and Evaluation

7.1 Hyperparameter Tuning

We employ random search coupled with cross-validation to fine-tune all of our classification models, ensuring optimal performance. This method involves randomly selecting combinations of hyperparameters, which are then rigorously evaluated through cross-validation. This process not only helps in identifying the most effective hyperparameters but also validates the model's stability and accuracy across different subsets of data, leading to a robust and reliable classification model.

7.2 Evaluation Metrics

We used Accuracy, Precision, Recall, F1 score and Confusion Matrix as our metrics to evaluate the performance of classification models.

Accuracy: Accuracy measures the ratio of correctly classified instances out of the total instances.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Precision: Precision measures the accuracy of positive predictions. It is the ratio of true positive predictions to the total predicted positives.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall: Recall (Sensitivity) measures the ability of the model to capture positive instances. It is the ratio of true positive predictions to the total actual positives.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

F1 Score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall.

$$F1Score = 2\ x\ \frac{Precision\ x\ Recall}{Precision\ + Recall}$$

Confusion Matrix: A confusion matrix is a table showing the number of true positives, true negatives, false positives, and false negatives. It provides a detailed breakdown of the model's performance

7.3 Linear Models

7.3.1 Logistic Regression. We implemented the The logistic regression using 'LogisticRegression' in scikit-learn. We trained the model using both the approaches OvR and multinomial via 'multi_class' argument ('auto', 'ovr', 'multinomial') with different solvers ('lbfgs', 'newton-cg', 'sag', 'saga') and regularization terms via 'penalty' argument ('l1', 'l2', 'elasticnet', None).

Logistic Regression model achieved a decent level of accuracy, correctly classifying 84% of the instances for all sovers. However, the F1-score is relatively low at 0.38 for all solvers, indicating that the model's performance in terms of precision and recall is not satisfactory. This might be due to issues: class imbalance, misclassification of rare classes (severity 1 and 4) and a poor choice of threshold for decision-boundary.

Classification Report:

	precision	recall	f1-score	support
0	0.52	0.03	0.06	1081
1	0.86	0.95	0.90	79845
2	0.69	0.49	0.57	19392
3	1.00	0.00	0.00	1443
accuracy			0.84	101761
macro avg	0.77	0.37	0.38	101761
weighted avg	0.83	0.84	0.82	101761

Figure 7. Classification Report for Logistic Regression

7.3.2 Stochastic Gradient Descent (SGD).

SGDClassifier in scikit-learn is a versatile linear classifier that implements regularized linear models (SVM, logistic regression, etc.) with Stochastic Gradient Descent (SGD) optimization, and it supports various loss functions. In our project, we trained the model using 'hinge' (for SVM) and 'log_loss' (for logistic regression) for parameter 'loss'. We used learning_rate as 'optimal' which is given by eta = 1.0 / (alpha * (t + t0)) where t0 is chosen by a heuristic proposed by Leon Bottou.

SGDClassifier model achieved an accuracy of 0.83 for both loss functions 'hinge' (for SVM) and 'log_loss' (for logistic regression). F1 Score is relatively low at 0.39 for hinge loss and at 0.35 for log_loss which is lower than the F1-score achieved with the Hinge loss function. This may be due to the reason that the log loss, also known as cross-entropy loss, aims to minimize the difference between predicted probabilities and actual class labels whereas the Hinge loss in SVM aims to maximize the margin between classes which possibly resulted in better optimizing the decision boundary for the given dataset. The reason for low F1 score is the same as Logistic regression as both SGDClassifier and LogisticRegression fit a linear decision boundary and class imbalance which resulted in low precision and recall.

Classification Report:

	precision	recall	f1-score	support
0	0.35	0.10	0.15	1081
1	0.84	0.97	0.90	79845
2	0.73	0.37	0.49	19392
3	0.00	0.00	0.00	1443
accuracy			0.83	101761
macro avg	0.48	0.36	0.39	101761
weighted avg	0.80	0.83	0.80	101761

Figure 8. Classification report for 'hinge' loss

Classification Report:

	precision	recall	f1-score	support
0	0.31	0.00	0.01	1081
1	0.84	0.96	0.90	79845
2	0.70	0.40	0.51	19392
3	0.00	0.00	0.00	1443
accuracy			0.83	101761
macro avg	0.46	0.34	0.35	101761
weighted avg	0.80	0.83	0.80	101761

Figure 9. Classification report for log_loss

7.4 KNN

We implemented k- nearest neighbors algotithm using KNeighborsClassifier in scikit-learn. KNeighborsClassifier implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user. We used the metric "minkowski" for distance computation, which results in the standard Euclidean distance when the Power parameter for the Minkowski metric, p = 2 for training the model.

We trained the model with k = 1 to 20 to find the best k for our dataset. KNeighborsClassifier achieved the best accuracy (0.81) and F1 score (0.39) for k = 5. For k = 7 to 20, the accuracy is 0.82 but the F1 score decreases as the precision increases and recall decreases respectively as k increases. This suggests that as the number of neighbors considered (k) increased, the model became less precise in identifying true positives and more sensitive in capturing positive instances, leading to a decrease in the F1-score.

Classification Report for k=5:

	precision	recall	f1-score	support
0	0.37	0.11	0.17	1081
1	0.84	0.94	0.89	79845
2	0.62	0.39	0.48	19392
3	0.34	0.02	0.03	1443
accuracy			0.81	101761
macro avg	0.54	0.36	0.39	101761
weighted avg	0.79	0.81	0.79	101761

Figure 10. Classification Report for KNeighborsClassifier

7.5 Decision Tree

Initially, we implemented the default DecisionTreeClassifier in sklearn without hyperparameter tuning, which aimed to establish a baseline understanding of the classifier's performance using its default settings. We delved into the classification report to gain a detailed breakdown of the model's performance (metrics such as precision, recall, F1-score, and support) on both the training and testing datasets. We also

included the confusion matrix provided a visual representation of the model's performance by illustrating the number of correct and incorrect predictions for each class.

We also embarked on hyperparameter tuning to optimize the model's performance by systematically searching for the best combination of hyperparameters. Through Randomized Search cross-validation, we evaluated various hyperparameter values efficiently, aiming to identify settings that enhanced the model's performance without exhaustive search. For the hyperparameter-tuning step, we have fixed the following parameters:

- criterion = gini
- splitter = best
- max_depth = 16
- min_samples_split = 17
- min_samples_leaf = 19
- random_state = 42

Following hyperparameter tuning, we retrained the model using the best hyperparameters identified during the search, and the optimized model was anticipated to exhibit improved performance. We finally conducted a thorough evaluation of the optimized model, examining its performance on both the training and testing datasets.

	precision	recall	f1-score	support
0	0.66	0.60	0.63	1081
1	0.91	0.94	0.92	79845
2	0.76	0.66	0.71	19392
3	0.39	0.16	0.23	1443
accuracy			0.88	101761
macro avg	0.68	0.59	0.62	101761
weighted avg	0.87	0.88	0.87	101761

Figure 11. Classification Report for Decision Tree

We also identified the most influential features by visualizing feature importance. Understanding which features contributed most to the model's predictions was crucial for interpreting its behavior and gaining insights into the underlying data.

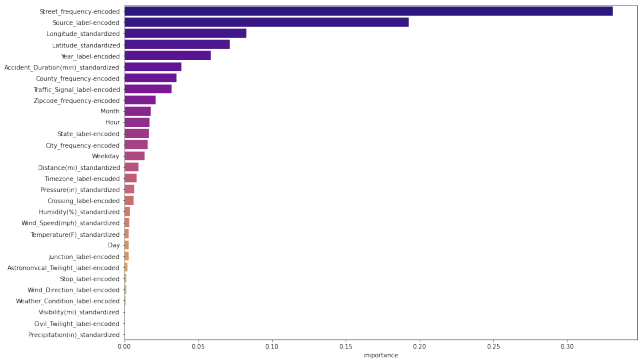

Figure 12. Feature Importance for Decision Tree

Finally, we also visualized the best decision tree model and provided an intuitive understanding of its structure and decision-making process.

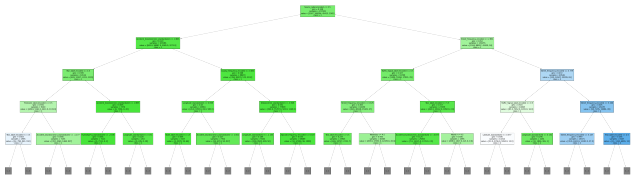

Figure 13. Best Decision Tree

7.6 Random Forest

Initially, we employed the default RandomForestClassifier in sklearn without hyperparameter tuning to gain a baseline understanding of the classifier's performance with its default parameters. We looked through the classification report to get a full breakdown of the model's performance (metrics like precision, recall, F1-score, and support) on both the training and testing datasets and also incorporated a confusion matrix, which visually represented the model's performance.

We also implemented hyperparameter tuning to improve the model's performance by methodically looking for the optimal combination of hyperparameters. We used Randomized Search cross-validation to effectively test alternative hyperparameter values, to identify settings that improved the model's performance without an exhaustive search.
For the hyperparameter-tuning step, we have fixed the following parameters:

- bootstrap = False
- ccp_alpha = 0.0
- class_weight = None
- criterion = gini
- max_depth = 22
- max_features = log2
- max_leaf_nodes = None
- max_samples = None
- min_impurity_decrease = 0.0

- min_samples_leaf = 5
- min_samples_split = 9
- min_weight_fraction_leaf = 0.0
- monotonic_cst = None
- n_estimators = 100
- n_jobs = None
- oob_score = False
- random_state = None
- verbose = 0
- warm_start = False

Following hyperparameter tuning, we retrained the model with the best hyperparameters identified during the search, and the optimized model was expected to perform better. Finally, we thoroughly evaluated the optimized model, analyzing its performance on both the training and testing datasets.

	precision	recall	f1-score	support
0	0.80	0.45	0.58	1081
1	0.91	0.96	0.93	79845
2	0.82	0.68	0.74	19392
3	0.75	0.06	0.12	1443
accuracy			0.89	101761
macro avg	0.82	0.54	0.59	101761
weighted avg	0.89	0.89	0.88	101761

Figure 14. Classification Report for Random Forest

We also identified the most influential features by visualizing feature importance for the Random Forest classifier model.

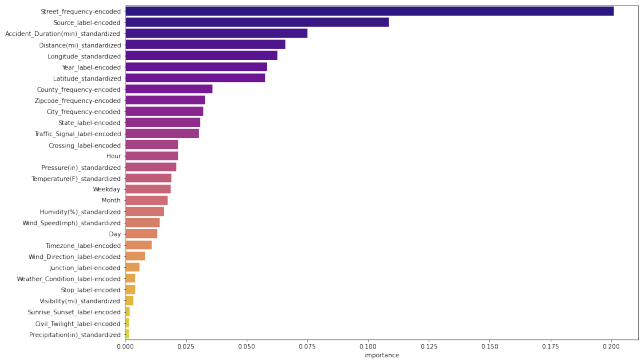

Figure 15. Feature Importance for Random Forest

7.7 Boosting

7.7.1 XGBoost. After an extensive hyperparameter search for the XGBoost model tailored for a multi-class classification task, we optimized several parameters to enhance performance and manage complexity. We settled on using 150 estimators (n_estimators=150) to balance between overfitting and underfitting while achieving robust model predictions.

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.64	0.68	1081
1	0.93	0.96	0.94	79845
2	0.82	0.76	0.79	19392
3	0.58	0.16	0.24	1443
accuracy			0.91	101761
macro avg	0.76	0.63	0.66	101761
weighted avg	0.90	0.91	0.90	101761

Figure 16. Classification Report for XGBoost

A learning rate (learning_rate=0.3) was chosen to ensure efficient convergence without sacrificing accuracy. The tree depth was set to 12 (max_depth=12), allowing the model to capture complex patterns in the data. The gamma value was adjusted to 0.2, providing a moderate regularization effect to avoid overfitting. We also utilized 80% subsampling of data (subsample=0.8) and features (colsample_bytree=0.8) to further enhance the model's generalization capabilities. These parameters were fine-tuned to strike an optimal balance between prediction accuracy and computational efficiency, resulting in a robust and performant model suitable for handling diverse multi-class scenarios.

The Classification Report for the XGBoost model shows high accuracy and strong performance in predicting class 1, with robust precision and recall metrics, but indicates significant room for improvement in class 3, which has notably lower scores. The overall accuracy of the model is high, yet the macro average suggests that this is largely due to the model's performance on the more prevalent classes, and not equally distributed across all classes. The weighted average f1-score reflects the model's effectiveness in handling the class imbalance by giving more weight to the performance on classes with more instances.

7.7.2 LightGBM. We selected 200 trees (n_estimators=200) to ensure sufficient model complexity without over-training, coupled with a learning rate of 0.05 (learning_rate=0.05) to achieve a deliberate pace of learning that enhances convergence stability. The model's depth was set to 5 (max_depth=5), allowing us to capture significant interactions without excessively fragmenting the data. To further prevent overfitting while ensuring that the model is robust and generalizable, we adjusted the number of leaves to 30 (num_leaves=30), and set the minimum child samples to 300 (min_child_samples=300). These choices provide a careful balance, ensuring that each tree in the model contributes meaningfully to the overall predictions, optimizing the model's ability to generalize well to new, unseen data.

The LightGBM model exhibits a performance close to the XGBoost model, excelling in class 1 but struggling with class

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.63	0.67	1081
1	0.92	0.96	0.94	79845
2	0.81	0.74	0.78	19392
3	0.56	0.14	0.23	1443
accuracy			0.90	101761
macro avg	0.75	0.62	0.65	101761
weighted avg	0.89	0.90	0.90	101761

Figure 17. Classification Report for LightGBM

Classification Report:

	precision	recall	f1-score	support
0	0.73	0.62	0.67	1081
1	0.92	0.96	0.94	79845
2	0.81	0.73	0.77	19392
3	0.56	0.15	0.23	1443
accuracy			0.90	101761
macro avg	0.76	0.61	0.65	101761
weighted avg	0.89	0.90	0.89	101761

Figure 18. Classification Report for CatBoost

3. It has a slightly lower overall accuracy and average scores, indicating that while both models perform similarly, XGBoost has a marginal advantage, particularly in addressing class imbalance. Both models, however, share a common area of weakness in class 3, pointing to a potential issue with the data representation for this class.

7.7.3 CatBoost. In optimizing the CatBoostClassifier for multiclass classification, we settled on a highly effective hyperparameter configuration. We chose a learning rate of 0.2 for swift yet stable convergence and a tree depth of 9 to capture complex patterns without excessive overfitting. The model runs for 1000 iterations, ensuring thorough learning, and applies an L2 regularization of 5 to prevent overfitting by penalizing large weights.

The classification report for the CatBoost model, much like the previous XGBoost and LightGBM models, shows strong results for class 1 and notably poorer results for class 3. While the accuracy is comparable to LightGBM at 0.90, the precision, recall, and f1-score for class 3 are marginally better in CatBoost than in LightGBM, suggesting a slight improvement for the least represented class. Nonetheless, class 3 remains a challenge across all three models, hinting at an inherent difficulty with this class that may require further data or model adjustments.

Classification Report:

	precision	recall	f1-score	support
0	0.31	0.94	0.47	1081
1	0.98	0.66	0.79	79845
2	0.51	0.89	0.65	19392
3	0.11	0.80	0.19	1443
accuracy			0.71	101761
macro avg	0.47	0.82	0.52	101761
weighted avg	0.87	0.71	0.75	101761

Figure 19. Classification Report for SPE

7.8 Imbalanced Learning

7.8.1 Self-Paced Ensemble. In tuning the SelfPacedEnsembleClassifier (SPE) for handling class imbalance, we identified an optimal set of parameters to maximize effectiveness. The final configuration uses 100 estimators, striking a balance between diversity and stability in predictions. We chose 10 bins for stratification to effectively focus on challenging minority class examples in manageable segments. The replacement=True setting allows for sampling with replacement, adding diversity, while soft_resample_flag=True enables the model to adaptively prioritize harder-to-classify instances, improving the focus and efficiency of learning. This configuration was carefully selected to enhance the SPE's ability to address imbalanced datasets efficiently.

The classification report for the SPE model shows a significant deviation in performance compared to the XGBoost, LightGBM, and CatBoost models. It has a notably lower precision for classes 0 and 2, and especially for class 3, alongside a high recall for classes 0, 2, and 3. The f1-scores are lower across all classes, indicating a lack of balance between precision and recall. The overall accuracy is substantially lower at 0.71, and the macro averages reflect this with the lowest f1-score of 0.52 among the models discussed. This suggests that while the SPE model may be identifying a high number of true positives, it is doing so at the cost of incorrectly labeling many other instances, which is not an ideal trade-off for most applications.

7.8.2 Balanced Bagging. Initially, we employed the default BalancedBaggingClassifier in imblearn with RandomForestClassifier as the base classifier without hyperparameter tuning to gain a baseline understanding of the classifier's performance with its default parameters. We looked through the classification report to obtain a comprehensive overview of the model's performance (metrics like precision, recall, F1-score, and support) on both the training and testing datasets. Additionally, we utilized a confusion matrix to visually depict the model's performance.
We have fixed the following parameters:

- sampling_strategy = auto

Classification Report:

	precision	recall	f1-score	support
0	0.61	0.81	0.70	1081
1	0.94	0.93	0.94	79845
2	0.78	0.81	0.80	19392
3	0.41	0.33	0.36	1443
accuracy			0.90	101761
macro avg	0.69	0.72	0.70	101761
weighted avg	0.90	0.90	0.90	101761

Figure 21. Classification Report for Voting

- n_estimators = 10
- random_state = 42

In our efforts to enhance the model's performance, we conducted hyperparameter tuning. This involved a systematic exploration of different hyperparameter combinations aimed at optimizing the model's effectiveness. Employing Randomized Search cross-validation, we methodically tested alternative hyperparameter values to pinpoint configurations that significantly improved the model's performance, all without the need for exhaustive searching. However, the accuracy and the macro average of F1-score did not improve.

	precision	recall	f1-score	support
0	0.25	0.95	0.39	1081
1	0.96	0.66	0.79	79845
2	0.54	0.83	0.65	19392
3	0.09	0.82	0.17	1443
accuracy			0.70	101761
macro avg	0.46	0.82	0.50	101761
weighted avg	0.86	0.70	0.75	101761

Figure 20. Classification Report for Balanced Bagging

7.9 Voting

The classification report for the Voting Classifier, which aggregates predictions from an ensemble of models including Random Forest, XGBoost, LightGBM, SPE, and Balanced Bagging, indicates a balanced performance with an overall accuracy of 0.90. This model achieves high precision and recall for class 1 and good results for class 2, but like the other models, it still shows room for improvement in class 3. The macro average f1-score of 0.70 reflects a more balanced performance across all classes when compared to the individual models, and the weighted average scores are consistent with the overall accuracy, demonstrating the efficacy of combining models to improve prediction stability and handle class imbalance.

7.10 Neural Network

The final evaluation results for the Neural Network are given in figures 22 and 23. In figure 22, it is apparent that the Neural Network ignored the rare labels, effectively reducing the problem to binary classification between the more common labels. While it achieved a decent accuracy of 85%, it's macro average of the f1-score suffered greatly because of this. In figure 23, we trained the Neural Network again on an undersampled dataset. This helped it greatly improve it's macro average, but it underperforms greatly on the original dataset. We hypothesize this is both because the resulting training set is small and that the transformed dataset no longer accurately represents the distribution of the original dataset.

Classification Report:

	precision	recall	f1-score	support
0	0.00000	0.00000	0.00000	1081
1	0.88318	0.93917	0.91031	79845
2	0.70636	0.61391	0.65690	19392
3	0.00000	0.00000	0.00000	1443
accuracy			0.85389	101761
macro avg	0.39738	0.38827	0.39180	101761
weighted avg	0.82758	0.85389	0.83944	101761

Figure 22. Classification Report for Neural Network

Classification Report:

	precision	recall	f1-score	support
0	0.86847	0.93140	0.89883	1035
1	0.69532	0.51985	0.59492	1058
2	0.73136	0.78952	0.75933	1031
3	0.75089	0.83235	0.78952	1014
accuracy			0.76655	4138
macro avg	0.76151	0.76828	0.76065	4138
weighted avg	0.76122	0.76655	0.75958	4138

Figure 23. Classification Report for Neural Network with Undersampled Dataset

8 Conclusion and Future Work

Model	Accuracy	Macro Avg. F1 score
Logistic Regression	0.84	0.38
SGD Classifier	0.83	0.39
KNN	0.81	0.39
Decision Tree	0.88	0.62
Bagging - Random Forest	0.89	0.59
Boosting - XGBoost	0.91	0.66
Boosting - LightGBM	0.90	0.65
Boosting - CatBoost	0.90	0.65
Deep Neural Network	0.85	0.39
Imblearn - Self-Paced Ensemble	0.71	0.52
Imblearn - Balanced Bagging	0.70	0.50
Voting	**0.90**	**0.70**

Initially, our hypothesis centered around the belief that weather attributes would serve as significant predictors of accident severity. However, upon conducting a thorough analysis, we discovered that location attributes held more explanatory power regarding accident severity. Interestingly, linear models, KNN, and Deep Learning algorithms exhibited subpar performance, particularly in distinguishing rarer severity labels. This limitation indicated the difficulties these models encountered in capturing nuanced patterns in the data. Conversely, ensemble methods emerged as robust alternatives, with boosting models showcasing superior performance over bagging models. Notably, our most successful model was the Voting ensemble, with an outstanding accuracy rate of 90% and a macro average F1-score of 0.70. This achievement underscored the efficacy of combining multiple models' predictions, harnessing their collective intelligence to achieve more accurate and reliable predictions.

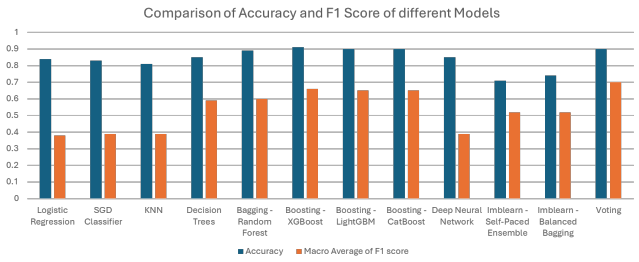

Figure 24. Comparison of accuracy and F1 score of different models

In the future, predictive models can be enhanced by incorporating geospatial or time-series data, recognizing their potential to provide valuable insights into accident severity. This integration will enable to capture spatial or temporal patterns that may influence accident outcomes, thereby improving the accuracy and robustness of the predictions. Furthermore, our models can be refined (oversampling, undersampling, or synthetic data generation) for managing class imbalance, as addressing this challenge is crucial for ensuring fair and accurate predictions across all severity levels. We can also refine our data preprocessing and feature engineering processes by utilizing one-hot encoding. To further advance our project, a real-time accident risk prediction framework can be integrated into the predictive model. This implementation will empower stakeholders, such as transportation authorities or emergency responders, to proactively identify and mitigate potential risks, thereby enhancing public safety and minimizing the impact of accidents. In addition, the relationships between key factors and accident severity can be thoroughly investigated and by analyzing these relationships comprehensively, deeper insights can be gained into the underlying dynamics driving accident outcomes, facilitating informed decision-making and policy formulation. finally, our findings can be translated into actionable insights that can inform policy interventions aimed at reducing accident severity, improving road safety, and enhancing the overall well-being of communities.

9 Task Distribution

Tsu-Hao Fu : Literature Review, Exploratory Data Analysis, Data Preprocessing, Feature Engineering, Models (XGBoost, LightGBM, CatBoost, Self-Paced Ensemble, Voting)
Koushiki Basu : Literature Review, Dataset visualization, Models (Decision Tree, Random Forest, Balanced Bagging), Conclusion and Future Work
Yohan Berg : Background study and Literature Review. Neural Network development, regularization, and fine-tuning.
Gopireddy Avinash Reddy : Models (Logistic Regression, Stochastic Gradient Descent (SGD), KNN), Evaluation Metrics

References

[1] David Schrank, Luke Albert, Bill Eisele, Tim Lomax. "2021 Urban Mobility Report and Appendices." The Texas A&M Transportation Institute, 2021.
[2] Zeng J, Qian Y, Wang B, Wang T, Wei X. "The Impact of Traffic Crashes on Urban Network Traffic Flow." Sustainability, 2019.
[3] Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, Rajiv Ramnath. "Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights." In proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2019.
[4] Johann Lau. 2020. "Google Maps 101: How AI helps predict traffic and determine routes" from https://blog.google/products/maps/google-maps-101-how-ai-helps-predict-traffic-and-determine-routes/
[5] Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. "A Countrywide Traffic Accident Dataset.", 2019.
[6] https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[7] Joaquín Abellán, Griselda López, and Juan De OñA. 2013. Analysis of traffic accident severity using decision rules via decision trees. Expert Systems with Applications 40, 15 (2013), 6047–6054.

[8] Ciro Caliendo, Maurizio Guida, and Alessandra Parisi. 2007. A crash-prediction model for multilane roads. Accident Analysis & Prevention 39, 4 (2007), 657–670.

[9] Li-Yen Chang. 2005. Analysis of freeway accident frequencies: negative binomial regression versus artificial neural network. Safety science 43, 8 (2005), 541–557.

[10] Li-Yen Chang and Wen-Chieh Chen. 2005. Data mining of tree-based models to analyze freeway accident frequency. Journal of safety research 36, 4 (2005), 365–375.

[11] Sachin Kumar and Durga Toshniwal. 2015. A data mining framework to analyze road accident data. Journal of Big Data 2, 1 (2015), 26.

[12] Lei Lin, Qian Wang, and Adel W Sadek. 2015. A novel variable selection method based on frequent pattern tree for real-time traffic accident risk prediction. Transportation Research Part C: Emerging Technologies 55 (2015), 444–459.

[13] Lu Wenqi, Luo Dongyu, and Yan Menghua. 2017. A model of traffic accident prediction based on convolutional neural network. In 2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE). IEEE, 198–202.

[14] Daniel Eisenberg. 2004. The mixed effects of precipitation on traffic crashes. Accident analysis & prevention 36, 4 (2004), 637–647.

[15] David Jaroszweski and Tom McNamara. 2014. The influence of rainfall on road accidents in urban areas: A weather radar approach. Travel behaviour and society 1, 1 (2014), 15–21.

[16] Fred L Mannering, Venky Shankar, and Chandra R Bhat. 2016. Unobserved heterogeneity and the statistical analysis of highway accident data. Analytic methods in accident research 11 (2016), 1–16.

[17] JD Tamerius, X Zhou, R Mantilla, and T Greenfield-Huitt. 2016. Precipitation effects on motor vehicle crashes vary by space, time, and environmental conditions. Weather, Climate, and Society 8, 4 (2016), 399–407.

[18] Athanasios Theofilatos. 2017. Incorporating real-time traffic and weather data to explore road accident likelihood and severity in urban arterials. Journal of safety research 61 (2017), 9–21.