Traffic Accident Severity Analysis and Prediction

Tikki Cui (ttcui@vt.edu), Kshitiz Dhakal (kshitiz@vt.edu)

1. Introduction

Each year, an average of 6 million traffic accidents happen in the United States, resulting in tens of thousands of deaths [1]. In a world where cars are one of the most popular modes of travel, reducing these accidents is a top priority for safety. Traffic accidents significantly impact lives and infrastructure, and understanding their causes can help form preventative measures. Our goal is to analyze traffic accidents on different severity levels and identify key factors that influence the severity of accidents.

2. Project Problem Statement

In this project, the main objective is to correctly classify a traffic accident's severity level given the details surrounding the accident. If we are able to successfully classify severity levels, we can gain crucial insights into what causes these accidents. Our plan to achieve this was two-fold. First, we ran some numerical data analytics on the data itself. We analyzed the different distributions of features and examined their details. We also created some different ways of visualizing the data in order to spot trends. The other part involved training several supervised learning classification models on the dataset. We compared the results and attempted to find the best model design that can accurately predict these severity levels.

3. Dataset Overview

The main dataset we are using is sourced from Kaggle and includes 500k detailed records of traffic accidents. This data spans from February 2016 to March 2023 and covers 49 states (excluding Hawaii). It contains 45 initial features related to accident details, such as time, location, weather, and road conditions. The raw data is sourced from several credible sources, such as the U.S. Department of Transportation, traffic cameras, and traffic sensors. Overall, the dataset provides a thorough and detailed overview of each accident and their surrounding circumstances.

One key feature is *Severity*. This feature categorizes how severe an accident is on a discrete scale ranging from 1 to 4. This feature is key because it serves as our target variable. We used this feature for predictive models that take in accident details and attempt to classify how severe it was. This also gave us the chance to analyze any correlations between features and the severity.

We also have 2 helper datasets to assist in our data analysis. The first is from GitHub user *cphalpert*, which provides information about which regions a state belongs to. For example, the state of Virginia is considered to be from the *South* region. This user took information from the U.S. Census and presented in CSV format. The other dataset contains U.S. population information in 2023. This dataset comes from Kaggle and we used it to help normalize some statistics.

Main Dataset: https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents
Regions: https://github.com/cphalpert/census-regions
Population:
https://www.kaggle.com/datasets/dataanalyst001/us-states-ranked-by-population-2024

4. Data Preprocessing

4.1 Pruning

The first thing we did was to get rid of unnecessary features. There were 2 features, `ID` and `Source` that were purely metadata. Since these features make no meaningful contributions to our analysis, we dropped these features from the dataset. We also dropped the `Country` feature. The only value was “US” since all of these accident records are from the USA. Therefore, it provided no meaningful information.

4.2 Feature Engineering

Next, we used One-Hot Encoding to convert some categorical features into binary features so our machine learning models could work with the data. The initial features we one-hot encoded were `Timezone`, `Weather_Condition`, `Wind_Direction`, and `State`. We also noticed that every timezone had a “US/” prefix for every value. As mentioned above, all records are from the USA. Therefore, we stripped off the prefix to make the values more concise. Although there are plenty of other categorical features, such as `Airport_Code`, we didn’t immediately one-hot encode them. We didn’t see any value in these features, so we decided to one-hot encode it on a case-by-case basis if future analysis required it.

Next, we fixed inconsistencies in the timestamp data. Some timestamps contained milliseconds while others did not. For the ones that did, they were typicall all 0’s. Since this level of precision is so small, we decided it was unnecessary to keep it. By removing the milliseconds from the timestamps, we ensured a consistent datetime format for all timestamp data. This will help ensure consistency and accuracy down the road when we starting training our models.

Finally, we added several features that helped to encapsulate additional detail. These features were `Is_Weekend`, `Elapsed_Time`, `Region`, and `Division`. All these features were derived from existing values in the original dataset. The purpose of these features was to capture extra details to make it easier to analyze and train our models.

4.3 Missing Values

Many features contained missing values of varying degrees. We imputed the missing values with the best strategy we came up with for each feature. Our strategy was to use common sense to smartly select other features to help estimate what the missing value would be. For example, we noticed that `Temperature(F)` and `Wind_Chill(F)` often shared very similar values, so we imputed them with each other.

We decided not to drop any data records containing missing values for now. Instead, we would drop them if and when a particular section of our project required to do so. This allowed us to preserve as much information as possible, only dropping when absolutely necessary. This concluded our process to handle missing values.

4.4 Final Results

After all this preprocessing, we finished with 243 features to work with.

5. Methods & Models

5.1 Data Analysis & Visualizations

The data was loaded using Pandas, and necessary libraries for numerical analysis and visualizations were imported. The dataset was preprocessed by normalizing numerical features and encoding categorical variables. The distribution of the target feature `Severity` was analyzed to understand the imbalance in the dataset. Correlation matrices were created to identify relationships between numerical features and the target variable `Severity`. We plotted several histograms to show the distribution of accidents by the hour of the day. Also a count plot was created to show the distribution of accidents by the day of the week.

5.2 Machine Learning

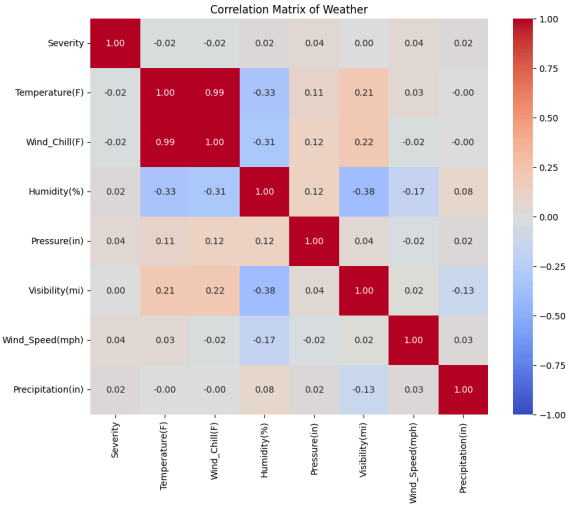
We initially wanted to use 5-fold cross validation to test our models. However, due to the sheer size of our dataset, it proved to be very time and resource intensive. Instead, we split the dataset into training, validation, and testing sets. The split we used for train/validation/test was 70/20/10 respectively. Before we trained, we also went through the dataset and dropped features we believed were unnecessary. This helped to reduce the dataset dimensionality, which sped up training. We also normalized our numerical values to ensure consistency in the data.

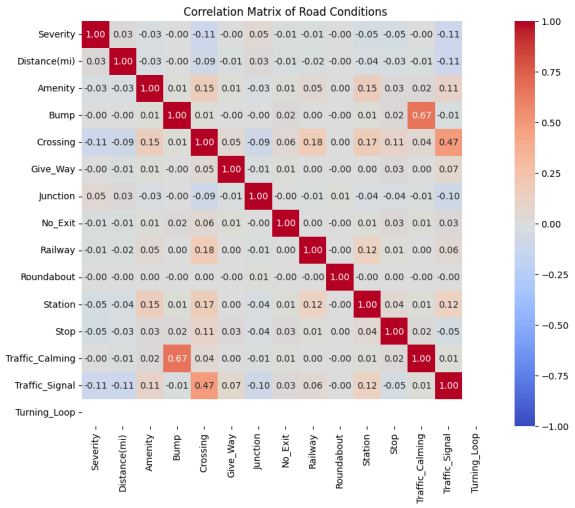
We looked at 4 different supervised learning models: logistic regression, decision trees, neural networks, and k-nearest neighbors. For each model, we tested out different model architectures and compared the performances based on model accuracy. Once we chose the best architecture for each model, we compared the precision, recall, and F1 scores of each model. Once we selected the very best model for the group, we ran a final performance check on our test set.

6. Results

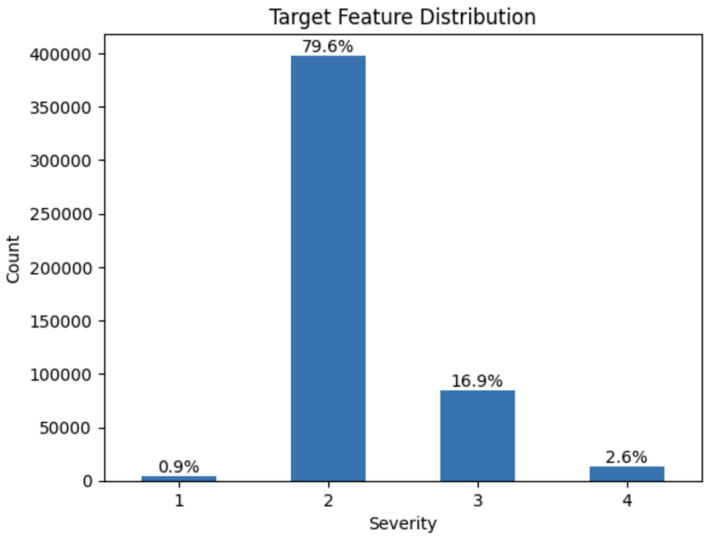
6.1 Numerical Analysis

First, we drew up correlation matrices for different sets of features. Ultimately, they didn't show too much information. There weren't any particularly strong correlations that we found. There does appear to be a weak negative correlation with `Crossing` and `Traffic_Signal`. This would indicate that the presence of those infrastructure may slightly help reduce accident severity. We also analyzed correlations with different weather conditions, but we didn't notice anything worth to report.

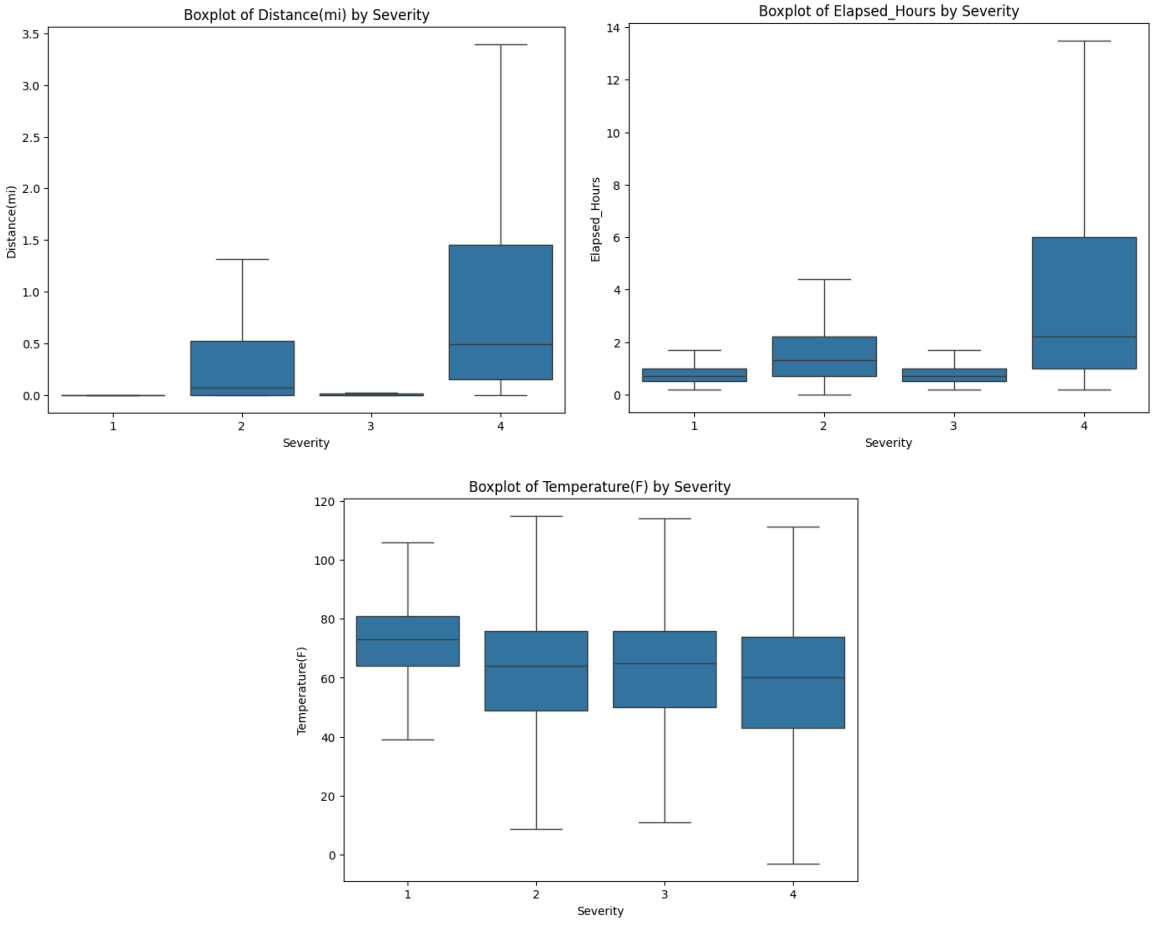




Next, we examined the distribution of our target feature `Severity`. The bar plot shows a significant imbalance in the severity levels of accidents, with the majority being of severity level 2. This imbalance needs to be considered when training machine learning models to ensure they are not biased towards the majority class.



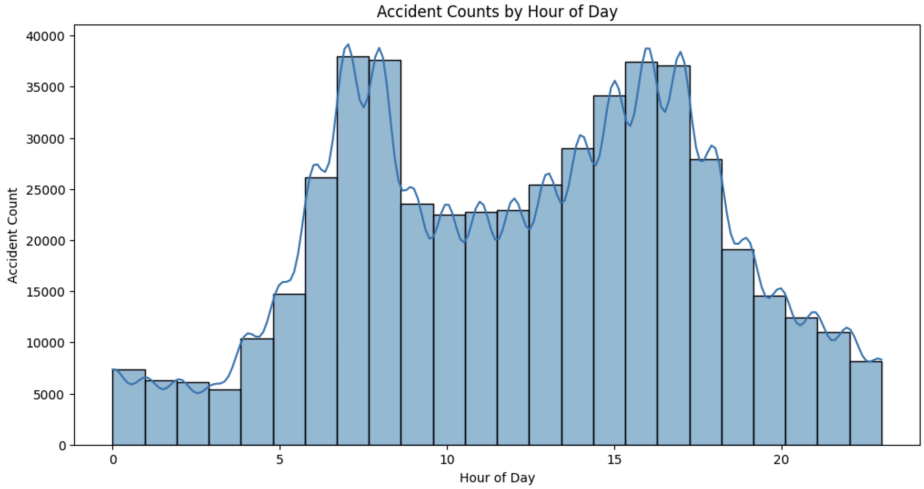
Finally, we drew up box plots for different features and observed their distribution changes across different `Severity` levels. Some of the notable ones we found were:



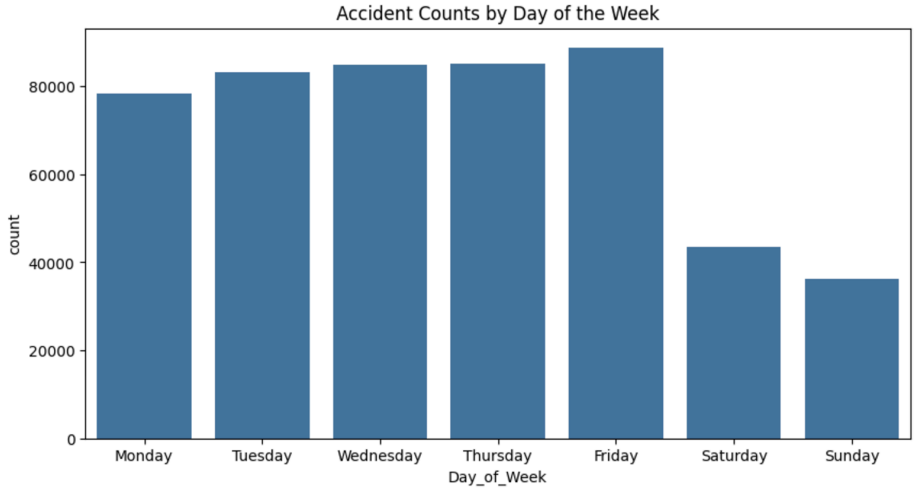
We can see that generally speaking, both the distance affected and elapsed time of an accident increases as severity increases. This could be useful information for our ML models to classify accidents. We also see that the distribution of temperature slightly drops as severity increases. This could correspond to colder conditions resulting in more hazardous driving conditions, which in turn causes accidents to be more severe.

6.2 Visualizations

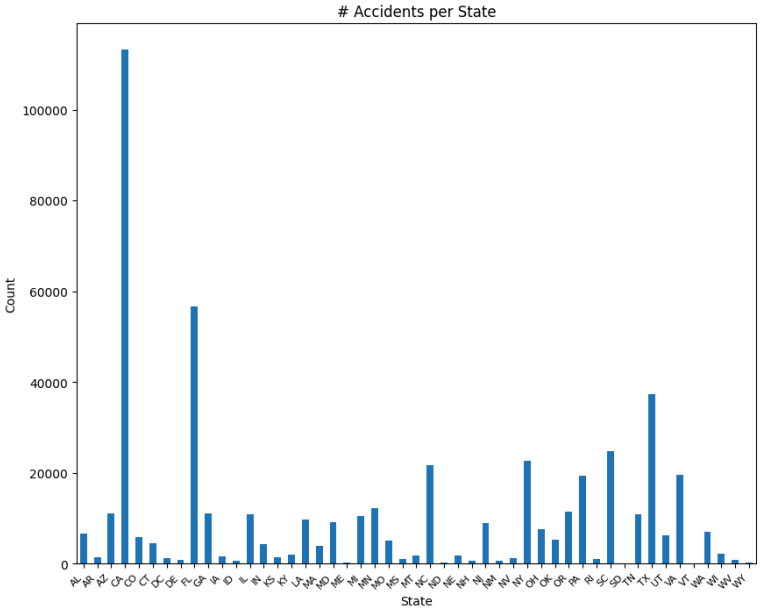
The histogram reveals that accidents are more frequent during the morning and evening rush hours, specifically around 8 AM and 5 PM. This pattern suggests that traffic congestion during commuting times contributes to a higher number of accidents.



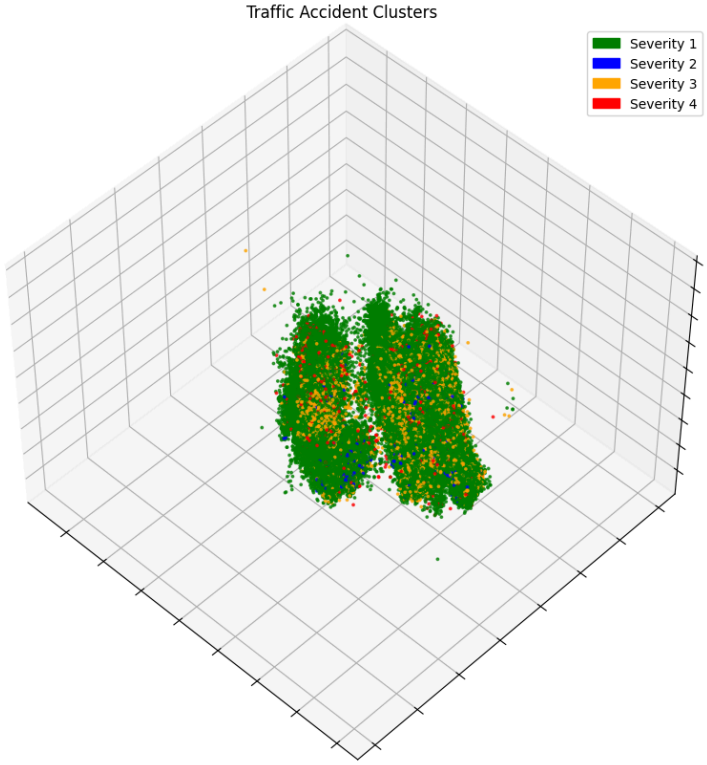
The count plot indicates that accidents are more common on weekdays, with a noticeable peak on Fridays. This trend may be due to increased traffic as people commute to work and engage in social activities at the end of the work week.



We also compared the number of accidents for each state. Before we did, we normalized the number of accidents by the state's population to account for population biases. The accidents per state reveal that California has the highest frequency of accidents relative to their population size followed by Alabama and South Carolina.



Finally, we ran principal component analysis (PCA) on our dataset to reduce the number of features to just 3. This allowed us to plot the data on a 3D scatter plot to see if we can spot anything unusual. The resulting plot didn't show anything significant.



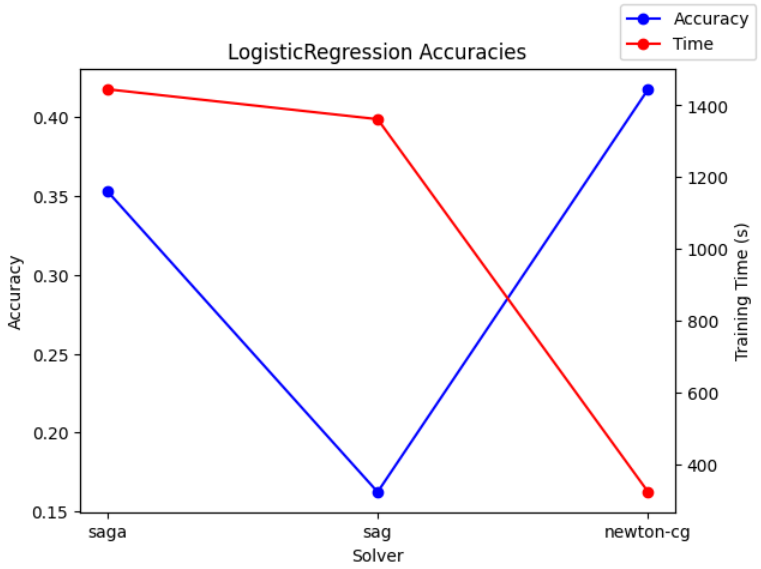
6.3 ML Model Results

Logistic Regression

For our logistic regressor, we experimented with 3 different solvers: `saga`, `sag`, and `newton-cg`. According to sklearn's documentation, these solvers were best for large datasets. Every model had the following hyperparameter settings:

- `random_state=1234` ensures consistency between notebook runs
- `multi_class='multinomial'` this is a multiclass problem since we have 4 possible target values
- `class_weight='balanced'` addresses the imbalance of records per class
- `max_iter=1000` limits train time
- `n_jobs=-2` uses all but 1 of the computer's cores to train

After training, our results were:



Solver	**saga**	sag	newton-cg
Accuracy	0.353	0.162	**0.418**
Time (s)	1445.2	1361.9	**324.6**

It's clear that `newton-cg` performed the best. It had the highest accuracy 0.443 as well as having significantly faster train time compared to the other 2 solvers. However, 0.443 is still very low. It is worth noting that both `saga` and `sag` failed to converge. If more iterations were allocated and they were allowed to converge, it's possible they would've performed better.

Decision Tree Classifier

Our decision tree had a lot more combinations of parameters that we experimented with. The parameters we experimented with were: `criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf`, and `max_features`. We had a total of 72 unique combinations, which is too much to completely list out (see notebook for accuracies for each combination). Our best decision tree model had the following parameters:

- `criterion='gini'`

- `max_depth=None`
- `min_samples_split=2`
- `min_samples_leaf=1`
- `max_features=None`

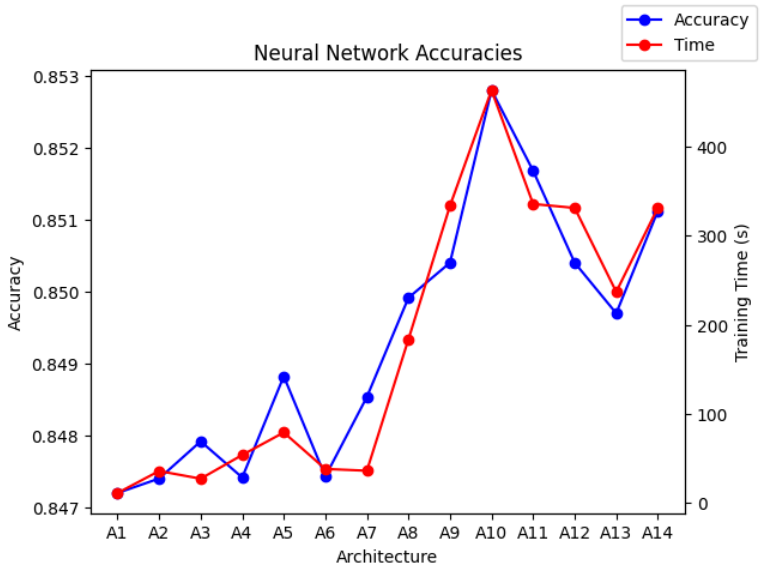
for an accuracy of 0.800 and training time of 10.6s. Although this looks like a significant improvement over logistic regression, we'll later demonstrate how it's misleading.

Neural Network

For our neural network, we experimented only with the network architecture, specifically the hidden layers. Each tuple represents the number of hidden layers, with each value being the number of nodes for each layer. The designs we experimented with were:

- A1: (20)
- A2: (50)
- A3: (100)
- A4: (200)
- A5: (50,50)
- A6: (100,50)
- A7: (10,100)
- A8: (200,100)
- A9: (100,100,100)
- A10: (50,100,50)
- A11: (100,100,50)
- A12: (100,100,100)
- A13: (100,150,100)
- A14: (50,100,100,50)

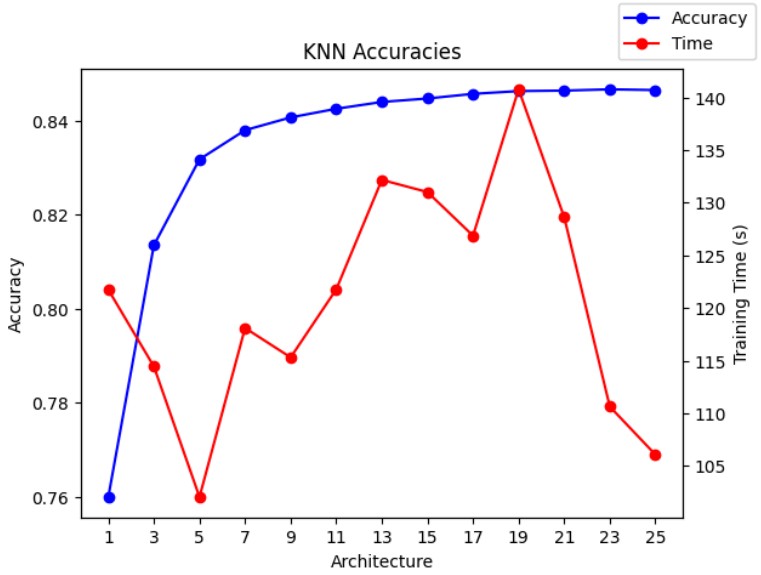
After training each architecture, we get the following results:



As we can see, the architecture with the highest accuracy was A10, which is (50,100,50), scoring 0.853. One thing we observed is that A10 also had the highest training time, taking almost 9 minutes. By comparison, the second longest architecture was A11 with around 6 minutes.

K-Nearest Neighbor

For our KNN model, we simply experimented with all odd *k*-values in the range of 1-25 inclusive. We used odd *k*-values so there's no chance of any ties. The results we got were pretty interesting. Unlike all previous cases, as *k* increased, so did accuracy. However, if we plot the accuracies, we get the following graph:



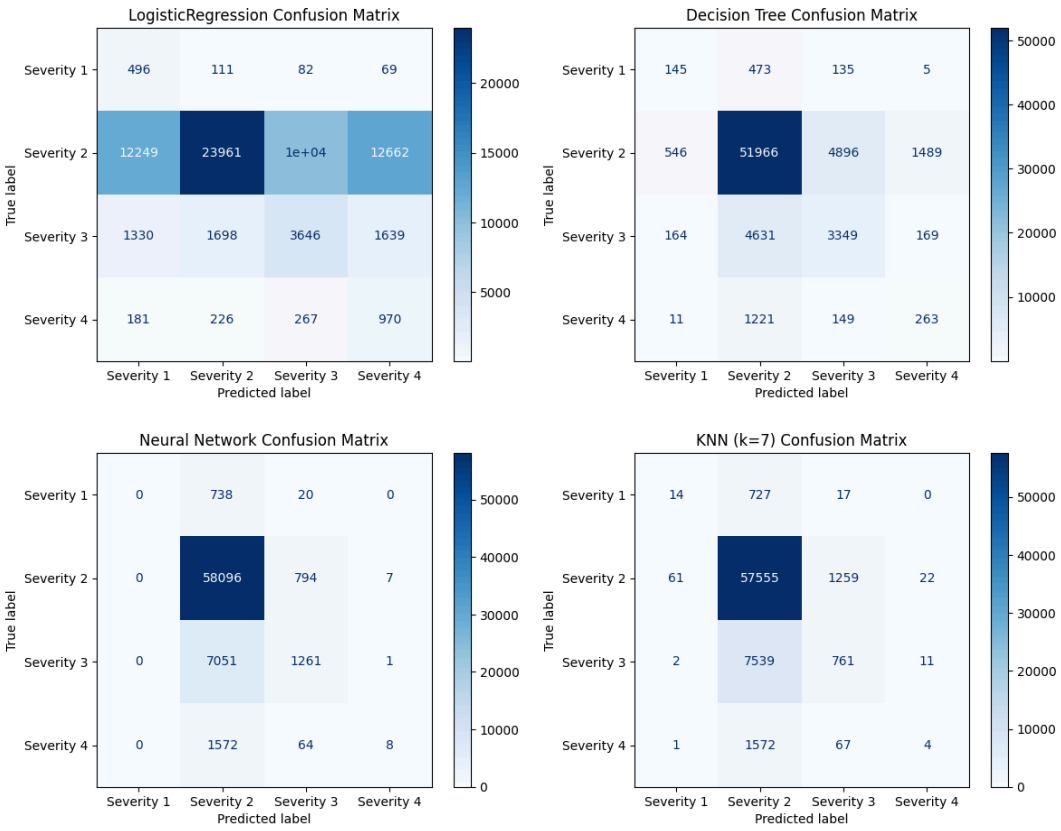
This graph is essentially a scree plot. Although accuracy continues to increase as *k* increase, we can see that we start to get diminishing returns. The tiny increase in accuracy is not worth the increased complexity and train time. We identified the elbow as being around *k*=7, which had an accuracy of 0.838.

Precision, Recall, F1

After selecting the best architecture for each model type, we decided to take a look at the confusion matrices for each and compare precision, recall, and F1 scores. The results we got were very surprising.

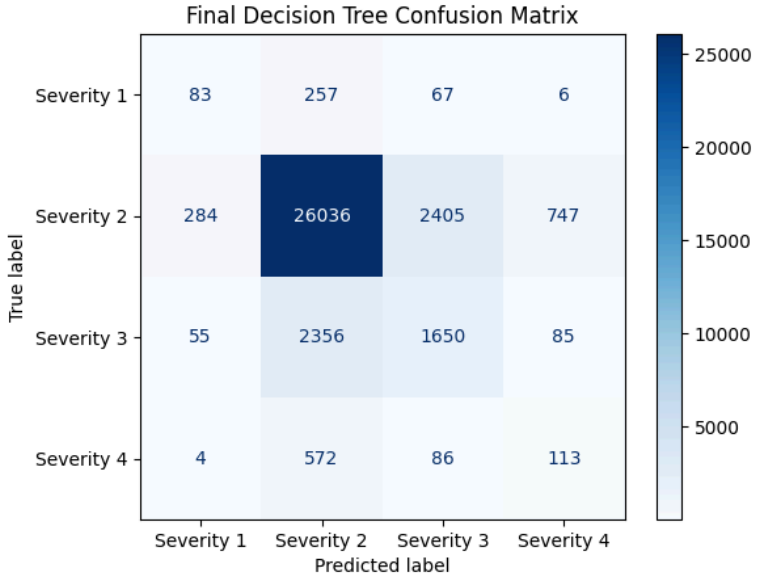
	Logistic Regression	Decision Tree	Neural Network	KNN (k=7)
Acc	*0.443*	*0.800*	***0.853***	*0.838*
Precision	0.320	0.397	**0.487**	0.376
Recall	**0.522**	0.409	0.286	0.272
F1	0.268	**0.403**	0.293	0.274

These scores are pretty bad, which is surprising since most of our models had accuracies of >80%. Looking at the confusion matrices immediately points out the problem.



As we can see, the cause lies in the Severity distribution. The models are frequently misclassifying records as `Severity` = 2 (referred to as *S2*). This explains why accuracy was high, but precision, recall, and F1 were low. The accuracy was only high because even though our models were only successful with predicting S2, since S2 was the majority of the classes, the overall accuracy was higher than it should be.

Even though all our models weren't great, we selected the decision tree as the best of the bunch and tried it out on our test set. Unsurprisingly, it did pretty poorly as it was clearly biased towards S2.



Accuracy	0.801
Precision	0.399
Recall	0.407
F1	0.403

7. Conclusion

Unfortunately, the results of our project did not align with our original goals for the project. Although our analysis revealed some interesting things, we failed to find anything convincing. Our machine learning models also performed a lot more poorly than we expected, with none of them reaching acceptable accuracy standards. However, we learned a lot from this project throughout the whole process.

The main takeaway we got is that the accuracy metric doesn't always show the whole picture. Although it is an important metric, it can easily be influenced by class distributions. Instead, it is better to use multiple metrics, such as precision and recall, which help provide a more complete representation of how a model performs.

Another takeaway is that the distribution of the dataset can also affect model performance. It's possible that there's simply little to no correlation between the features, and therefore not possible to predict. However, it is quite likely that the skewed target feature distribution is causing the models to not train evenly on all features. We also could've filtered out some of the features. Instead of trying to train our models on all the features, it may have been better if we smartly picked only the ones that contained useful information.

In the end, this project was a great experience. Although we didn't get the results we were looking for, we did learn a lot of valuable lessons regarding data analytics and machine learning.

8. Individual Contributions

Tikki	• Dataset preprocessing • Feature engineering • ML models
Kshitiz	• Feature engineering • Data visualizations • Numerical analysis

References

[1] Darrigo & Diaz Law. 2021. Car Accident Statistics You Need to Know in 2021. Retrieved December 14, 2024, from https://www.ddlawtampa.com/resources/car-accident-statistics-you-need-to-know-in-2021.