Applying Decision Trees To Detect Dangerous Driving Locations

Abstract

In the United States, car accidents are a leading cause of death, with tens of thousands of people dying every year. Recently, new technology has emerged that uses AI to help drivers drive safely. A notable example is Tesla's cars, which promise self-driving, and driver aids like automatic braking. In this project, I investigated if Decision Trees could accurately predict the severity of car accidents given data about the situation, which gives drivers a frame of reference for how dangerous the current situation is. I took a dataset of car accidents in the US from 2016 to 2023, and trained multiple Decision Tree models of varying configurations and sizes on it. Decision Tree models are generally faster than neural networks (previously used in many studies on predicting accident severity) and also outperform them when the data is in a tabular format. Furthermore, I also created an app that utilizes the decision tree model that I trained and tested its performance in a real-world driving situation. While my model was able to achieve accuracy rates of 78%-85% depending on the specific size, the models noticeably overfit to the data at high sizes and were heavily influenced by uneven training data at low sizes. Furthermore, my app was able to quickly run a decision tree model to predict the possible accident severity. Future avenues for research include using my app as a platform for other models and investigating optimization methods and other types of models that could possibly be applied.

Introduction

In the US, car accidents kill tens of thousands of people every year [1]. Certain locations are more prone to dangerous incidents, such as a San Jose man's house where cars have repeatedly crashed into the house [2]. If a system to detect dangerous driving locations can be detected, it will allow drivers to know which areas are the most dangerous, leading them to be more vigilant, potentially saving many lives.

Decision trees are a machine learning model that use trees with decisions at nodes to make predictions. Decision trees are lightweight compared to more complex neural network based models, and tend to perform better as well [3]. Furthermore, decision trees are easily interpretable, which could provide information to help policy-makers design safer roads.

In this paper, the effectiveness of decision trees in predicting accident severity is analyzed, and a real-world application is shown by demonstrating a phone app that predicts the possible severity of an accident in real-time. My initial hypothesis was that decision trees will be able to achieve at least 70% accuracy in predicting the possible severity, with <2 seconds spent for each prediction, and decision trees with a depth close to half the depth of the largest decision tree model will perform the best.

Background

Decision trees are a machine-learning model that uses trees to arrive at an outcome given inputs [4]. At each node, a comparison is done to determine which branch to follow, eventually leading to an outcome. Decision trees can be trained to fit a specific dataset for problems such as labeling.

Overfitting [5] is one of the main problems with decision trees, as too big of a tree can make the model become too specific to the training data and be inapplicable to data that wasn't present in the training data. However, shrinking the decision tree model too much is also problematic, as it results in underfitting, or the model not having enough information to predict correctly.

One advantage of decision trees over neural network models is that they often outperform them on tabular data while being more lightweight, requiring less resources to fine-tune and infer [3]. Furthermore, compared to neural networks or ensemble learning methods, decision trees are also much more easily interpretable [6] (having been applied to interpreting more complex machine learning models in the past), and for this specific problem it could provide insight on what specifically causes driving risk, which can be used to inform future driving safety systems or public policy.

To encode categorical data, a basic approach is a one-to-one mapping with integers i.e. the first category corresponds to 0, the second category corresponds to 1, and so on. However, this introduces

order into the categorical variable, so one-hot encoding is another method that is oftentimes used, and it involves converting each categorical value into a vector where one number is 1 i.e. the first category could be encoded as $\begin{pmatrix}1 & 0\end{pmatrix}$ and the second as $\begin{pmatrix}0 & 1\end{pmatrix}$. Both methods are tested in this project.

Methodology

Model Design

To train the model, the dataset *US Accidents (2016 - 2023)* [7] [8] was used. Scikit-learn's DecisionTreeClassifier [9] was used to implement the Decision Tree. Non-relevant variables like latitude/longitude, data source, etc. were eliminated during the preprocessing phase, leaving only relevant data. Different encodings of the categorical variable "Weather Condition" were also tried, namely one-hot encoding and assigning a scalar number to each category.

After creating a basic decision tree model, further optimization was attempted. The first thing that was tried was cost-complexity pruning, but it was infeasible on the large decision tree. Instead, decreasing the model depth by increments of 10 was tried, and depths of 5, 3, and 1 on the lower end of the spectrum were eventually used.

App Design And Implementation

To build an app to test my model in the real world, React Native was used as it allowed for quick prototyping and had a simple interface to ONNX Runtime. ONNX Runtime is a platform that allows AI models to be used across platforms through the ONNX model format.

The app integrates multiple APIs to gather data that is then fed into the AI model. Firstly, the device's native geolocation services are used to get the latitude and longitude of the device. Afterwards, this position is used to collect weather information from the OpenMeteo API. Lastly, the Overpass API from OpenStreetMap, an initiative to create maps using open data, is used to find nearby points-of-interest. After all of this data was collected, it was fed into the model.

Model size was also a concern, as loading the largest model with unlimited depth ultimately failed as it required too much time. The medium-sized model which still had a depth of 20 but was only 5-6 MB was chosen as the model to be used.

The app also has a logging function that records performance statistics, like inference time and time spent on API calls, to facilitate data collection. Logs are written to a text file on the device, and a script was written to import the data into Google Sheets.

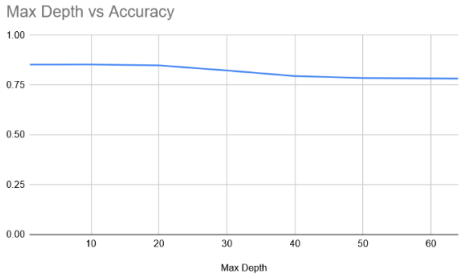
Testing

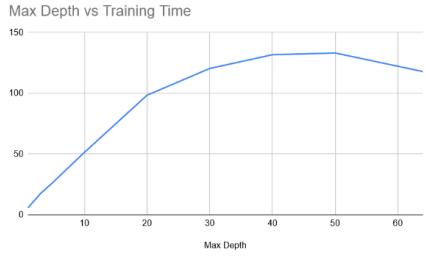
To test the model, Scikit-learn was used to split the data into a 75% training set/25% testing set with a fixed random state for every test to ensure that the data didn't change over different runs. The accuracy of the model was measured using Scikit-learn's accuracy_score function.

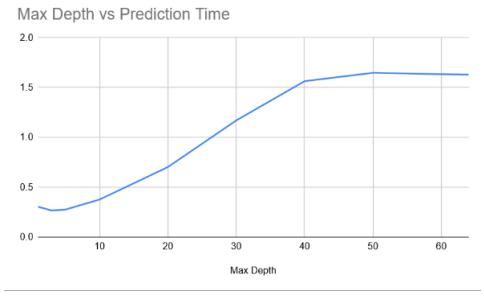
To test the app, the app's logging features were used to record performance data. A car was driven around town, and the app recorded performance data while the car was being driven.

Results

Model Results



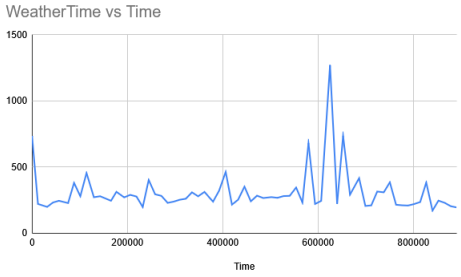


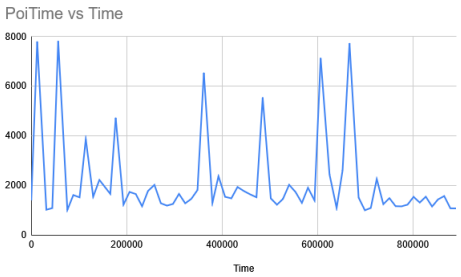


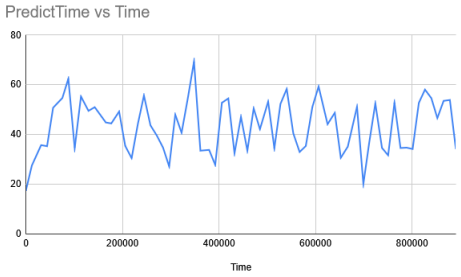
In addition to these graphs, the one-hot encoding model took 197.49 seconds to train, 2.39 seconds to infer, while having an accuracy of 78.295%, and the simple categorical encoding model took 125.19 seconds to train, 1.87 seconds to infer, and had 78.308% accuracy. Because of the minimal gains in accuracy and the large increase in training time, the one-hot model was not used in the app. The simple categorical model was not used due to complications with

the APIs.

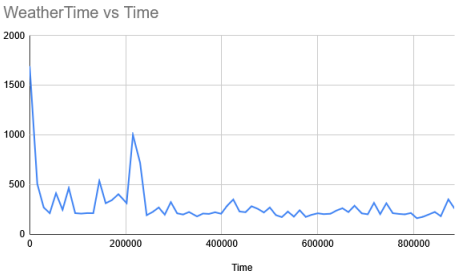
App Run 1 Results

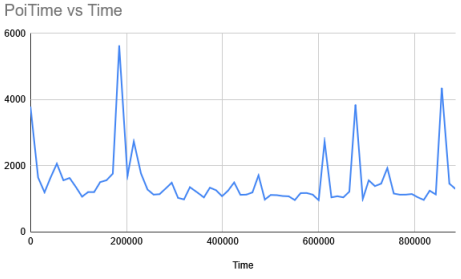


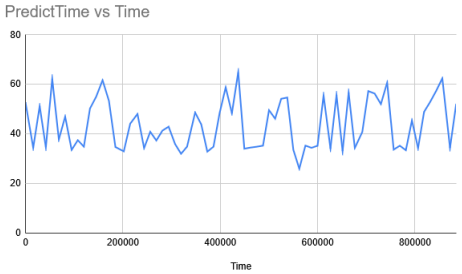




App Run 2 Results







Discussion

As the maximum depth of the decision tree was decreased, the accuracy of the decision tree increased from around 78% to around 85%, showing that there was overfitting present in the decision tree. However, analysis of the smaller decision trees shows that the model is also highly sensitive to imbalanced datasets (datasets where there is significantly more of one class than another class), as the smallest models essentially just output 2 no matter what, with more possible outputs added as the model gets larger. This indicates that there are much more accidents of severity 2 than other accidents, which can be confirmed through analyzing the dataset.

The results of the app testing show that APIs are the main limiting factor as the model is much faster than querying for weather or points of interest, even when run on a mobile device.

The curve of depth vs training time and inference time appears to be a roughly logarithmic shape. Since depth is roughly the base-2 logarithm of tree size for a perfect tree, it would suggest that the growth rate of the tree is slowing down, which is confirmed by the file sizes of the models.

Conclusion

The hypothesis on the performance of decision trees was validated in this project, as decision trees were able to achieve at least 70% accuracy in predicting the possible severity, with <2 seconds spent for each prediction. However, the second part of my hypothesis was invalidated, as small decision trees ended up performing better due to the unbalanced dataset. These results remained consistent in the phone app, with similarly if not faster inference times.

These results suggest that while decision trees are a viable candidate for a real-world, more optimization has to be done to optimize real-world performance due to overfitting and biased datasets. The most optimal model size theoretically would be the medium-sized models, as it balances both problems.

One of the main possible pathways for future research is improving the phone app, as it achieved

decent performance with the decision tree model and can easily serve as a platform for other models through ONNX. Another possible path for future exploration is investigating variations on the classic Decision Tree model, like Random Forests or XGBoost. Finally, a deep analysis of the larger decision tree models could reveal more insights about the data that could help with driving safety.

References

[1] *NHTSA launches Put the Phone Away or Pay campaign; Releases 2023 fatality early estimates*. Apr. 2024. URL: https://www.nhtsa.gov/press-releases/2022-traffic-deaths-2023-early-estimates.

[2] Shawn Chitnis. *San Jose home at dangerous intersection has been rammed by cars '23 times'*. Aug. 2022. URL: https://www.cbsnews.com/sanfrancisco/news/san-jose-home-at-dangerous-intersection-has-been-rammed-by-cars-23-times/.

[3] Ravid Shwartz-Ziv and Amitai Armon. "Tabular data: Deep learning is not all you need". In: *Information Fusion* 81 (Nov. 2021), pp. 84–90. DOI: 10.1016/j.inffus.2021.11.011. URL: https://doi.org/10.1016/j.inffus.2021.11.011.

[4] S. B. Kotsiantis. "Decision trees: a recent overview". In: *Artificial Intelligence Review* 39.4 (June 2011), pp. 261–283. DOI: 10.1007/s10462-011-9272-4. URL: https://doi.org/10.1007/s10462-011-9272-4.

[5] Tom Dietterich. "Overfitting and undercomputing in machine learning". In: *ACM Computing Surveys* 27.3 (Sept. 1995), pp. 326–327. DOI: 10.1145/212094.212114. URL: https://doi.org/10.1145/212094.212114.

[6] Mengnan Du, Ninghao Liu, and Xia Hu. "Techniques for interpretable machine learning". In: *Communications of the ACM* 63.1 (Dec. 2019), pp. 68–77. DOI: 10.1145/3359786. URL: https://doi.org/10.1145/3359786.

[7] Sobhan Moosavi et al. "A countrywide traffic accident dataset". In: *arXiv (Cornell University)* (Jan. 2019). DOI: 10.48550/arxiv.1906.05409. URL: https://arxiv.org/abs/1906.05409.

[8] Sobhan Moosavi et al. "Accident Risk Prediction based on Heterogeneous Sparse Data". In: *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Nov. 2019), pp. 33–42. DOI: 10.1145/3347146.3359078. URL: https://doi.org/10.1145/3347146.3359078.

[9] *DecisionTreeClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.