Predicting Accident Severity with Neural Networks

Scott Houston

3/17/2021

Keywords: Automobile, Neural Network, Machine Learning

Abstract

There have been a large amount of car accidents in the United States over the last few years. The focus of this paper is to reliably predict the severity of an accident given certain parameters. Severity in this paper refers to the length of time between the occurrence of an accident and traffic returning to normal. I will demonstrate the effectiveness of using a neural network to find the desired outcome, as well as comparing the results with other Machine Learning models. The accuracy for the simple Neural Network model comes to 0.75. Though this is an increase in accuracy based on randomly guessing, it isn't a significant increase in accuracy when considering the response vector.

Introduction

There are hundreds of thousands of car accidents across the United States every year. These result in fatalities, property damage, and traffic jams. While civil engineers and city planners can't eliminate the possibility of a car accident, they are tasked with alleviating as many of these situations as possible through the design of our roads. They help create clear guides on roadways, plan common detours in the case of an accident, and monitor traffic data to improve overall traffic flow. This helps to alleviate confusion as citizens commute to work each day or travel to see family. In an effort to minimize these accidents, data analysis is critical.

I've looked at data collected from 2016-2019 of numerous car accidents across the United States. With the reliance on GPS in smart phones and car displays, data has been easier to collect on the many different accidents that occur. This leads to many different factors to consider when evaluating the causes of an accident. The best target value in the data set is the severity. This represents the amount of time it takes for traffic to clear following an accident. Data on the injuries sustained in specific accidents wasn't collected in this data set. As well as information on the drivers themselves. Therefore, any variables or predictions will be solely based on the environment where the accident takes place.

Methods

I collected traffic data from kaggle.com; the data originally came from Mapquest and Bing. Both companies offer access to their live maps for drivers. The software reports back information like estimated time to arrival, driver's speed, density of traffic in the area, and modern ones list the speed limit in the area. This information allows the company to record a large number of traffic irregularities. The speed of the car very accurately signals what type of traffic situation that the driver is in. This data set condensed the known accidents into different groups based

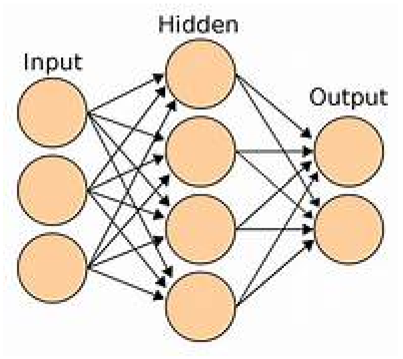

Figure 1: Neural Network Layers and Nodes

on the situation. The variables were almost all binary, dummy variables, related to the situation. For example, if the accident happened at an intersection, or if there was a traffic light. Other variables were recorded from the date/time of the accident and the weather conditions. The personal information about the drivers, such as age, sex, etc., wasn't included in the set. This limited the number of ways I could evaluate the data.

There were many different questions I considered when looking through my data. The variables were easy to group into things like weather conditions, which are very important when driving safely. The data set is also very large, 3.5 million observations with 47 variables each. Trying to reduce the number of missing values is a different challenge. The precipitation in particular was missing a large portion of the data.

Limiting the Covariates

After understanding the workings of Neural Networks, I started evaluating the data set. I began by evaluating the different variables, noting the many unique types : Boolean, integer, and character. I began cleaning the data by converting all variables into either binary dummy variables or as float (decimal) numbers. The next objective I tackled was the large number of missing values. There were so many missing values in a few categories, more than 50% of the data in some cases, so I decided to create two different data sets. I imputed the columns with their means or medians in one data set, and I removed any observations with missing values in the other data set. I also looked at the distribution of the data before and after in both cases. This gave me an idea of how exactly the values were changing. Finally I scaled all variables besides the dummy variables to remove their influence.

Then I considered which covariates to include in the model. The first step is to consider the covariance matrix for all of the input variables. There weren't any values beyond 0.4, which didn't indicate any problems to me. I then put the data into a linear regression model to see if significance values would indicate anything. There were a few covariates that were considered insignificant, so I decided to drop them. However, I noticed there was a high multi-collinearity problem. Temperature, Humidity, Precipitation, and Visibility had high VIF values, so I dropped all covariates besides Visibility. These two steps gave me a slightly better understanding of which values were important to consider.

Application of the Neural Network

When beginning to create Neural Network models, I chose to use a 80/20 split between the training and testing data. When considering the number of hidden layers, I decided to begin with a grid search of possible hyper-parameters. Hyper-parameters are the skeleton of the neural network. Things like the number of hidden layers, number of nodes per layer, dropout rate, and momentum. These key parameters determine the way the neural network learns. The different hyper-parameters are easy to estimate using a complex loop to compare the different combinations of parameters. By creating nested loops. I was able to run every different combination of parameters; the main problem was trying to run enough epochs to determine the models with the highest accuracy. By limiting the epochs in size, I'm able to get a brief view the model's effectiveness under unique conditions. The measurements for our model's effectiveness are loss and accuracy. However, the accuracy, based on the training data, is different to the validation accuracy, based on the testing data being used in the current iteration of the model.

I started with a model design that was a fully connected, feed-forward neural network. This model assumes that all nodes are completely connected from the previous layer and that all values are only moving in one direction. So the output layer of one run doesn't affect the weights of the same run. The model should then determine adjustments to the weights and biases based only on the previous model epoch. It had a structure of $[32, 20, 10, 4]$, where the numbers represent the number of nodes in each layer. This model had the highest accuracy of the grid search loop with

In Figure 2 below, I have the best results of a batch of trial runs. The model had a lower number of epochs due to to the time constraints. I expanded the neural network to accommodate the full number of variables. So the input layer expanded to 32. There are two pairs of values displayed in the graph. The two pairs of lines should converge given enough time. But the stable accuracy score still wasn't as high as I wanted.

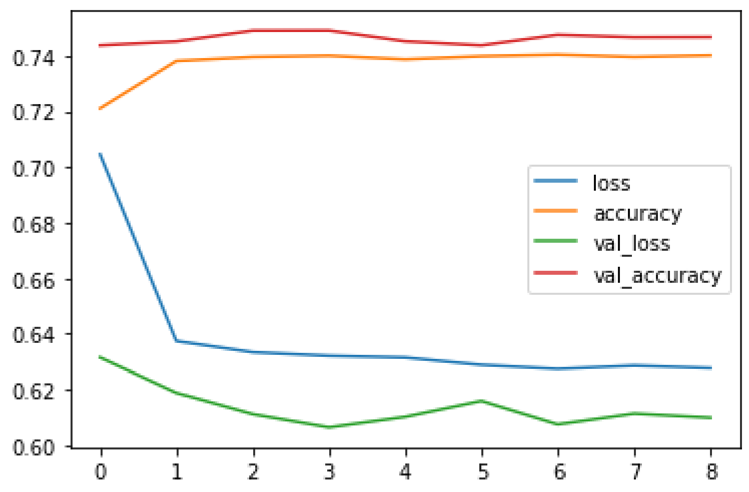

Figure 2: Full Model [32, 20, 10, 4]

The best possible model outcome I achieved was when I reapplied similar model hyper-parameters to the neural network with more epochs. This model was able to achieve an accuracy score of 0.7675. While this is better, I was originally hoping to boost the accuracy above 80%. Figure 3 below is the resulting accuracy and loss display. This gave me the idea that simply more nodes and wider model may increase accuracy, but I found it doesn't work in this situation. I attempted both 4 and 5 hidden layers in the model, but they both don't produce higher accuracy than this model. I originally set the number of epochs to 100, but as the model ran, the loss for the training data began to deviate from the validation loss. Therefore, I set a function to stop the model early when this began happening.

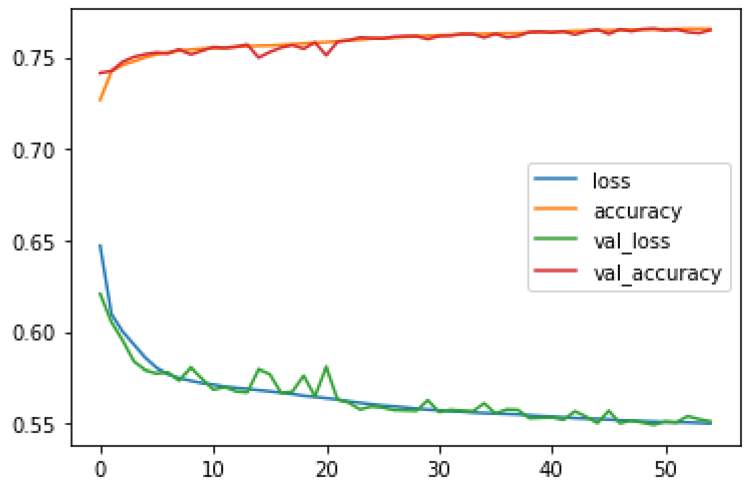

Figure 3: Full Model: [32, 20, 15, 10, 4]

When applying similar hyper-parameters to the model with the observations dropped, I got the results in Figure 4 below. I noticed that there wasn't a natural curve to the decrease in loss and increase in accuracy, but I believe this happened simply by random chance. The weights matrix and bias vector simply matched the optimal matrix and optimal vector by accident. The model still didn't increase the accuracy or decrease the loss more than the other models. So I decided to stay with the data set consisting of all observations.

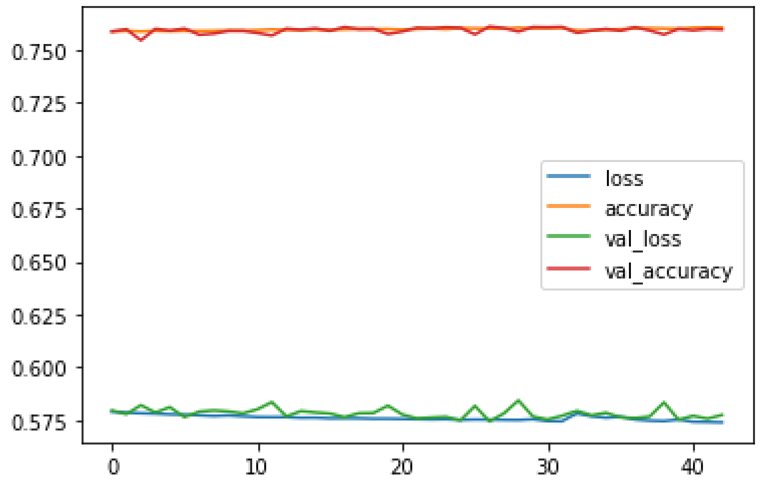

Figure 4: Reduced Observation Model: [24, 20, 20, 4]

The final point I want to bring up is the balance of the response variable. This was something I found that needed my consideration, because the severity values are very skewed. There are very few level 4 accidents, while the vast majority are level 2, about 70%. This can potentially cause problems in how the model runs. However when running this model with similar hyper-parameters, I found that there isn't a reasonable comparison in the accuracy and loss results. Figure 5 shows the way that loss and validation loss continue to deviate from each other as they move. What this indicates is that the model isn't able to predict the testing response, even if it is able to predict the training response. However the level of accuracy, roughly 67%, is less than the unbalanced model results.

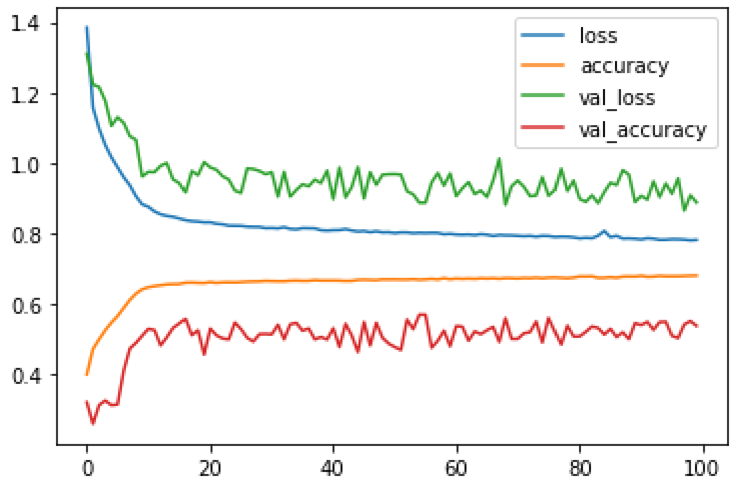

Figure 5: Balanced Input Model: [32, 20, 20, 20, 4]

Conclusion

The models that I created were able to accurately predict the severity of an accident given environmental conditions at roughly 76% accuracy. However the fact that severity 2 was almost 70% of the response, indicates to me that the neural network doesn't increase the accuracy significantly. If a model simply chose the severity of 2 in every situation, it would achieve roughly the same level of accuracy as the models I created. This indicates that there are missing pieces of information that need consideration. My thinking is that human error or individual characteristics are the missing variables that are necessary. The data sets I used only relayed environmental factors so there wasn't any consideration of the drivers. As cities become larger and larger, there is a higher need for considering accidents that may happen under certain conditions. The results of this model give insight into what combinations of conditions result in the highest accidents and what severity results should be expected, but more insight into the drivers in a given area may be necessary to increase the accuracy above 76%.

References

[1] Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. “A Countrywide Traffic Accident Dataset.”, 2019. Retrieved from https://www.kaggle.com/sobhanmoosavi/us-accidents

[2] Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, and 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. Retrieved from https://www.kaggle.com/sobhanmoosavi/us-accidents