Prediction Model on Traffic Accident Severity Using MLP

COMM7370 Group Project

He Qing 19450141

Shi Yi 19451512

Liu Zhaoxing 19456182

Wen Facheng 19456360

Hong Kong Baptist University

**Abstract**

Traffic accident has strong contingency of in modern society, so in this project we focus on accident prediction. Based on the countrywide traffic accident dataset of the USA in 2019, we build a three-layer perceptron to train the prediction model, of which the output layer is for predicting the level of severity (from 2 to 4). The features we used for prediction are mainly weather situation. After multiple training model and adjustment, we finally got 63% of test accuracy with 61% of training accuracy by using ‘adam’ learning function with 4-fold cross validation. The learning rate was 0.0001, the batch size 512 and number of epochs 100. The drop-out rate of the first layer was 15% and for the second lay it was 45%. Hence, given enough weather features, our model can predict the severity of traffic accidents with over 60% accuracy.

**Background**

Traffic accident is one of the events with high death toll and strong contingency in modern society. According to the WHO's most recent Global health estimates and the Global status report on road safety, there were 1.3 million people die on the world's roads and 20 - 50 million are injured every year. Road traffic crashes are a major cause of death among all age groups and the leading cause of death for children and young adults aged 5–29 years. The global epidemic of road traffic deaths and disabilities is increasingly being recognized as a major public health problem. The first step to understanding global road safety and developing effective road safety interventions is to know the facts. What exactly affects the occurrence of traffic accidents?

This project would focus on accident prediction. We would use the Mxnet to build our neural network to classify what factors would affect the traffic accidents. The neural network can be used in a vehicle's GPS system to predict extreme weather or traffic conditions and help drivers avoid accidents.

**Introduction**

This project would use Multilayer Perceptron (MLP) to build the neural network. Multilayer Perceptron (MLP) is an artificial neural network with forward structure, which maps a group of input vectors to a group of output vectors. MLP can be regarded as a directed graph, which is composed of multiple-node layers, each layer is connected to the next layer. Each node is a neuron with a nonlinear activation function except for the input node. Supervised learning methods using back propagation algorithms are used to train MLPS. MLP is an extension of perceptron, which overcomes the weakness of perceptron that cannot recognize linear non-scoreable data.

The multilayer perceptron can realize nonlinear discriminant if used for regression can approximate the input nonlinear function. And any function with continuous input and output can be approximated by MLP. MLPS with hidden layers can learn any nonlinear function from the input (the number of hidden nodes is unlimited).

Multilayer perceptron is also called artificial neural network. In addition to the input and output layer, it can have multiple hidden layers in the middle. The simplest MLP only contains one hidden layer, and the structure of three-layer perceptron is shown as follow in Fig1.

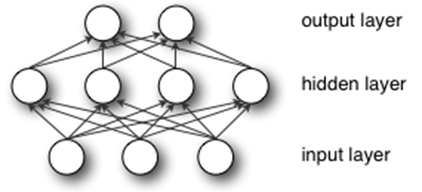


Fig1. Multilayer Perceptron (MLP)

Neurons in each layer of the perceptron are fully connected to neurons in different layer, but neurons in the same layer have no connection with each other, with the input layer at the bottom, the hidden layer in the middle, and the output layer at the top. The input layer is used to input data, and we can consider it as the first abstraction of information from input data. The hidden layer is fully connected with the input layer. Suppose the input layer is represented by vector X, then the output of the hidden layer is f(W1X+b1), where W1 is the weight vector (also known as the connection coefficient), and b1 is the bias. Function f is the activation function, usually using Rectified Linear Unit (ReLU), Tanh or Sigmoid. For ‘ReLU*’*, it uses to reach the non-linear effect. The output of the whole network needs an extra process which is called multinomial logistic regression to represent the prediction. The output of the whole network is just sum of multiple linear function’s result. We need to transform the output into probability of each category that we need to predict. Soft-max is used here to do that.

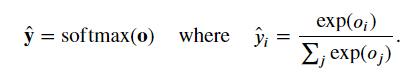


Fig2. Soft-max function

As shown in Fig2, can be used to predict class as it is considered as the probability, and the sum of elements in is 1. The class with highest probability is the prediction we make.

**System Model**

We have built a three-lay perceptron to make prediction, with 64 outputs in first layer, 16 outputs in second layer and 3 outputs (for the three classes of severity) in the output layer. Nodes in each layer is fully connected to each other. Each hidden layer uses the ‘ReLU’ to be the activation function. Fifteen percent of outputs from the first layer will be dropped, for the second layer, the drop-out rate is 45%. The output of the neural network will be transformed into probability by soft-max function, and then the loss of network is calculated by cross entropy function. With auto grad from mxnet package, the gradient of loss function can be easily computed.

In terms of how we build network on program, mxnet.gluon.nn.Sequential() is used to build the basic structure of a network block, and the structure is saved in variable named net. Then we use net.add(gluon.nn.Dense()) to add fully connected layers to this basic structure. We have added four layers at the first time when we built the model. Loss function is given by gluon.loss.SoftmaxCrossEntropyLoss(). We use stochastic gradient descent (sgd) function to optimize network’s parameters at the beginning.

**Training**

For the first time, we separated the data in 7:3 as training set and test set, and for each epoch, we sent 512 (the batch size) data instances a time to train the model. The learning rate was set to 0.1. We evaluated training accuracy of every batch and test accuracy every epoch. Number of epochs was set to 10 at first. The training accuracy and test accuracy reached 77% after the first epoch, but the loss is extremely high, reaching over one thousand, and loss reduced to about 1.5 after the first epoch which was extremely unusual. Then we found that the dataset is biased because it has more than 75% data instances with the same label, so we sampled the dataset with equal proportion for each label, and we dropped a label as the number of it was too small compared with others.

After data cleaning, the training accuracy and test accuracy became normal, both reaching about 53%. However, with training times increasing, the test accuracy and training accuracy didn’t change too much. The highest accuracy reached 58%.

To get a higher training accuracy and test accuracy, we tried another training method which is k fold. K-fold method is to separate dataset into k parts, use one part to be the test set and others to be the training set, and repeat this process for k times.

We set k equal to 4, so the 75% of data was used to train and 25% of data was to test. Also, we dropped 15% of outputs from the first layer and 45% of outputs from the second layer. The number of epochs is set to 40 and the learning rate is set to 0.0001, and the optimizer is changed to adam. We found that test accuracy is continuously raising but it stops when changing to another round of k-fold cross validation. Then we increased the number of epochs to 100. The test accuracy raised to 61% and it is stable for the last part of training epochs. Finally, we reduced the number of layers to 3 to reach a better test accuracy.

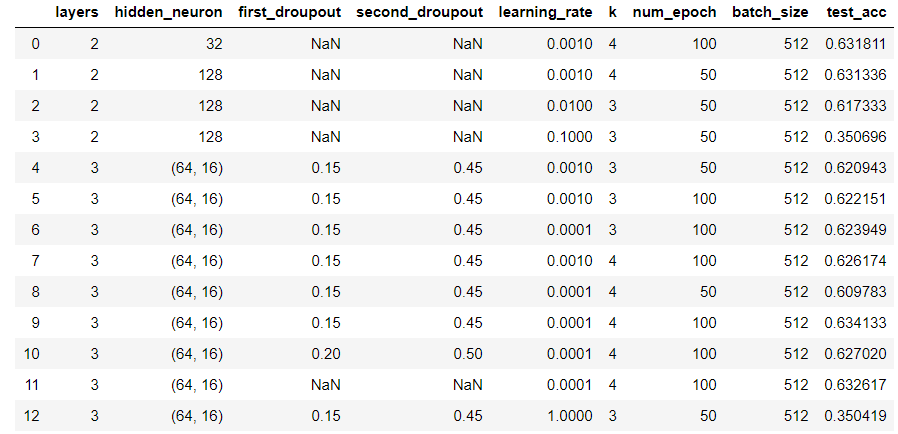
The final model reaches test accuracy at 63%, however, we have only three labels in total, and probability over 60% is a very special number as it is nearly 2/3, which probably means that our model is able to recognize one label and for another two labels, all it can do is to guess. Then we changed some parameters to see if we can have a higher test accuracy. We tried to set k equal to 3, learning rate equal to 1 or 2 or extremely small, training epoch equal to 150. We also tried to reduce the number of layers to 2, remove the drop out process. They all have limited influence or even worse results, and the results of those changes are shown as below:

Fig1

Our data is from a big dataset containing information about accidents happened in US in 2019. We only extract some weather conditions to be training features and severity of accidents to be labels. We tried to add more features to train our model, including traffic signs. However, the test accuracy gets lower after adding these features. We found that the number of data instances having these features is a little bit small (nearly 60,000) compared with the total number of data instances (nearly one million), and the sampling of data makes the features even more sparse, which makes our model even more confused to recognize different labels.

**Conclusion and limitation**

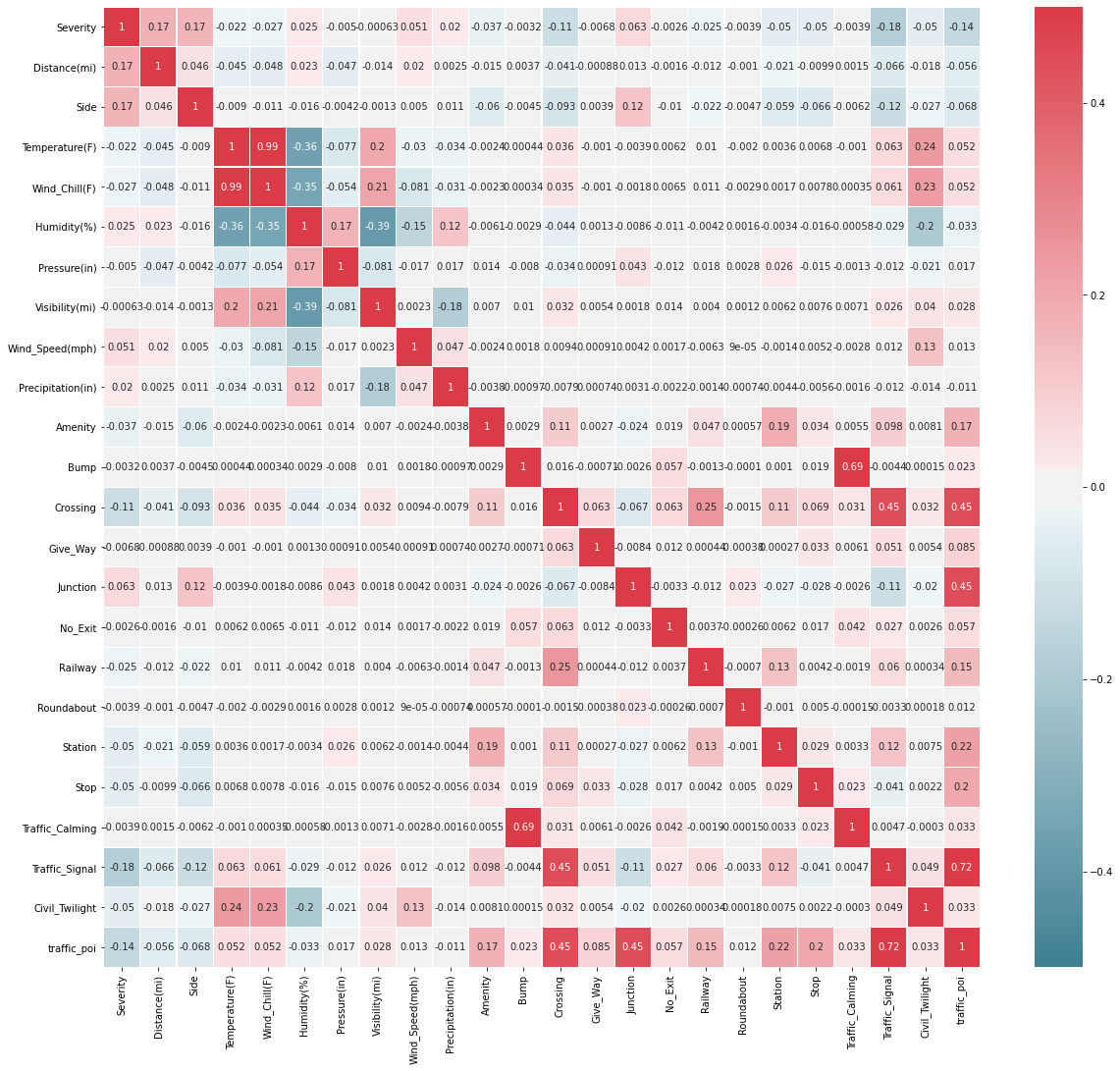


Fig2

To find out why we have such an unusual result, we tried to analyze the dataset in statistical way. We found that in different severity (labels), the average values of quite a lot of features are equal, which means that these features are not so expressive to support our model to recognize different labels. As shown in Fig2, the correlation between labels and other features is extremely low, with highest correlation reaching about 0.18, which means these features have almost no relation with our labels. We speculate that there is no obvious correlation paradigm between features and labels.

For this simple model we built, it has done its best to learn from the features to predict the labels, but it is difficult to make precise prediction based on such features that have only a very weak relationship with each other. Learning rate really has a significant impact on final test accuracy. Sometimes having a small learning rate and big number of training epochs can have some better results, but the increasement of test accuracy is limited. Therefore having 63% of test accuracy with 61% of training accuracy is not bad for this model and such dataset, at least it learned some patterns instead of nothing and it is better than totally guessing.

The limitation of our model is that the model itself is too simple to learn complicated patterns, and the dataset is not informative for our model to find the reflection from features to labels. The future optimization can be changing to another informative dataset and build a real deep neural network to make precise prediction.