Brian Kilburn

Jamie Weiss

Dominic Holt

**Group Project Progress Report**

Our team started out by meeting one day where Jamie, Dominic, and Brian Kilburn searched through the available datasets that had been collected and posted on Kaggle.com. Jamie decided that it would be a good idea to create a machine learning project that could contribute to the automotive industry. We decided upon a vehicle dataset that contained various features of an automobile.

Cars.csv

A screenshot of a computer

Description automatically generated

In total there are 428 entities, it is not the ideal size for a machine learning dataset. However, it does provide enough information to find distinct correlations between each feature to be able to closely predict the MSRP of a vehicle based on known select features using a logistic regression model. Our Instructor Dr. Wooky informed our group that a Recurrent Neural Network would be the best implementation to accomplish our goals for the project.

Considering the information presented from the instructor our group ventured upon research of RNN implementation. Starting with a defining question: What is a Recurrent Neural Network? Over time we individually built our knowledge about how to build an RNN network, it was during this time that the group was instructed to work together and create a power point presentation to share our findings and project idea implementation with the class. Brian Kilburn created the fifteen-slide power point presentation and ran some tests on the group dataset to find the various correlation features to be able to implement our learning model. Brian Kilburn found a correlation between the MSRP and Horsepower data which displayed a strong R statistical value that provides evidence for this relation: 0.826945

A screenshot of a cell phone

Description automatically generated

Another relation that Brian Kilburn tested was an MSRP summation which visually showed that vehicles built in Europe where more expensive compared to vehicles from Asia and the United States, with this type of information it is possible to consider using this as a feature to help calculate estimated vehicle pricing:

A screenshot of a cell phone

Description automatically generated

A final test implemented by Brian Kilburn showed a price comparison for all the vehicle models available:

A screenshot of a cell phone

Description automatically generated

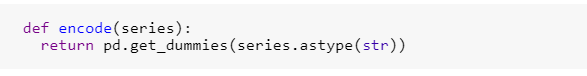
This data shows that Mercedes-Benz is the most expensive vehicle above all currently in the dataset. The price of the Mercedes-Benz doubles the price of another high-end vehicle called the BMW. This model nearly tops the 90% mark on the chart, the information here could also be used to factor price prediction for the model.

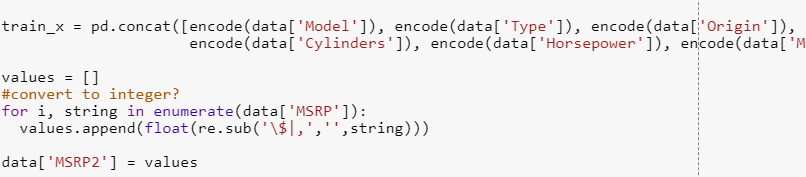
The next step was implementing the model, at first the difficulty level for creating an RNN at our experience level was too high. So, we chose an alternative route to implement the model and that was to build a simple neural network. It took some time to implement the SNN because the program kept crashing with a ‘Tensors must be broadcast able for logits and labels’ error message and getting the data into the correct type proved rather difficult to accomplish. After some research it turned out that the broadcast able error was a tensor shape mismatch:

A close up of a logo

Description automatically generated

The y placeholder at first expected the data to be of shape (?, 445) when in fact there were only 410 column classifications in the actual data feed. Most of the columned data was convertible to floats when implementing one hot encoding for the categorical data:





As you can see in the above image’s pandas get\_dummies function was utilized to implement encoding for the categorical data then the encoded data was concatenated to the train\_x variable which is our initial training set. In that second snippet of code the concatenation takes place, but another thing to point out here is the float conversion for the MSRP column. This column is our labels data it had to be converted from a string to a float value and special characters had to be removed to correctly format the data. Once all this was accomplished Brian Kilburn initialized the weights and biases, along with hypothesis function and cost function for the model:

A screenshot of a social media post

Description automatically generated

Finally, for the SNN the calculations after running the graph session:

A close up of text on a white background

Description automatically generated

The accuracy is 0% the implementation is found out to be incorrect for the situation. The idea behind our implementation for the model is for it to return a calculated price of a vehicle based on known features. However, this model was set up to perform classification hence:



The algorithm is trying to classify MSRP value in this situation there is only one classification per instance which means they are unique overall (i.e 410 classifications for 428 instances) so there were only 18 similar instances out of the entire dataset. The SNN implementation did not work.

Considering experience with the newfound technology about Simple Neural Networks, we used this knowledge to proceed with construction of the Recurrent Neural Network. Using a similar strategy, we processed the data pretty much the same; using the one hot encoding and converting the MSRP to float:

A screenshot of a social media post

Description automatically generated

The data needed to also be reshaped properly to feed into the RNN graph, the RNN cells can only receive data in a sequence (i.e [[ [1],[2],[3] ]] = (1,3,1) or [ [[1],[2],[3]],[[4],[5],[6]],[[7],[8],[9]] ] = (3,3,1) ) therefore our data was reshaped to (428,1072,1) Next the data was split 70% training 30% testing.

A screenshot of a cell phone

Description automatically generated

For the RNN to work there had to be LSTM cells created to store cell states and implement a prediction based on previous cell states. Brian Kilburn implemented the lstmCell() function:

A screenshot of a cell phone

Description automatically generated

Key here is to notice the activation function used, ‘selu’ which stands for Scaled Exponential Linear Units, this function receives input then will implement built in weights, biases and normalize the output. It also helps determine and maintain previous cell states. This LSTM cell factory is issued with 5 units per function call.

A screenshot of a cell phone

Description automatically generated