Auto-MPG Report

**Dataset**

The dataset examined in this report was obtained from [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/auto+mpg). It contains 398 instances and 8 attributes.

* Input Variables:
  + 2 - Cylinders
  + 3 - Displacement
  + 4 - Horsepower
  + 5 - Weight
  + 6 - Acceleration
  + 7 - Model Year
  + 8 - Origin
  + 9 - Car Name (unique for each instance)
* Output Variable:
  + 1 - MPG

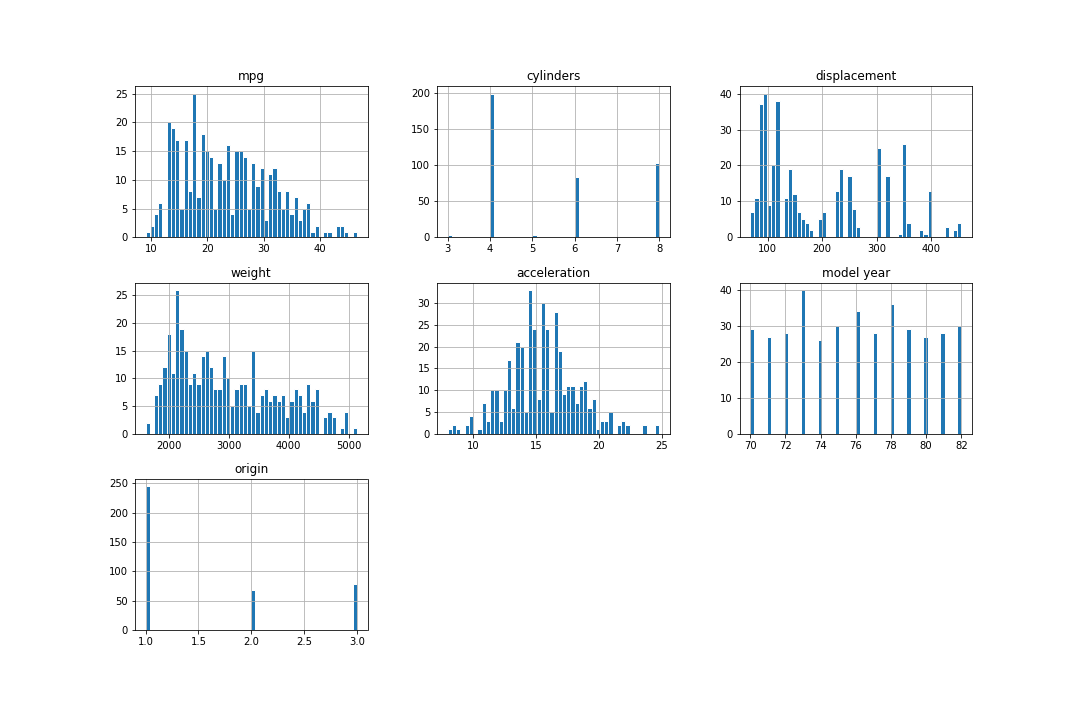
The input variable of “Car Name” was eliminated due to its object type being a unique string to each instance.

**Exploratory Data Analysis**

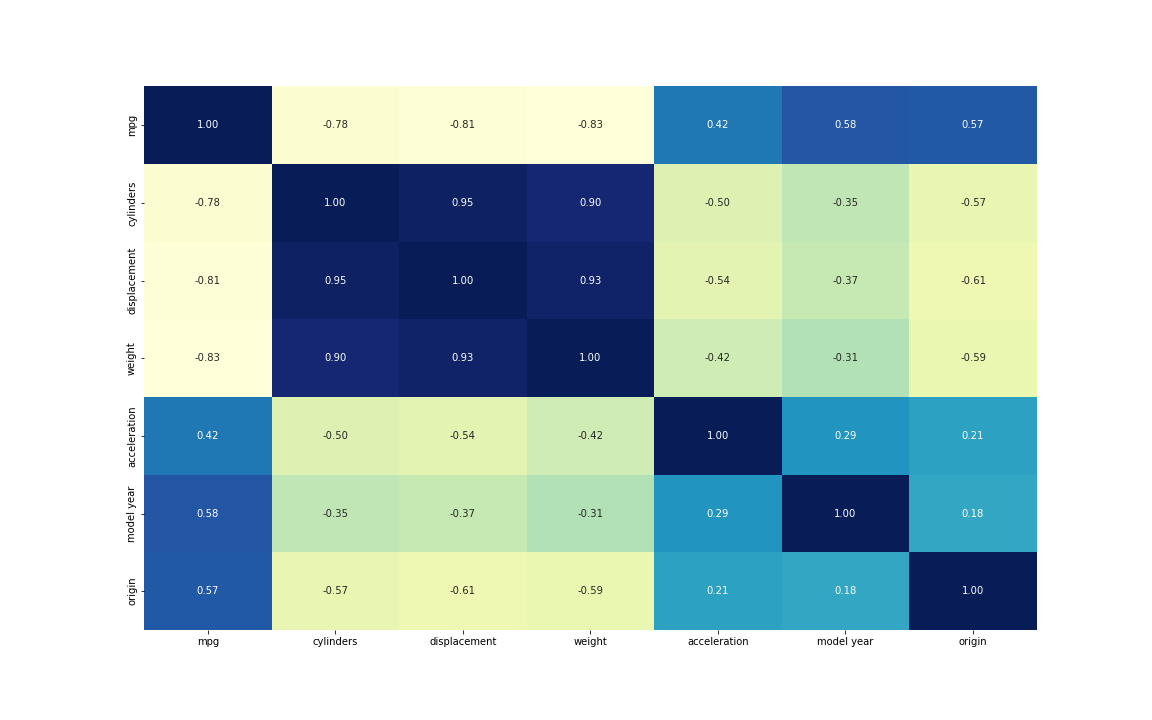
Table 1. Summary Statistics of auto-mpg.csv.

|  | **mpg** | **cylinders** | **displacement** | **weight** | **acceleration** | **model year** | **origin** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 392.0000 | 392.0000 | 392.000000 | 392.00000 | 392.00000 | 392.0000 | 392.0000 |
| **mean** | 23.44591 | 5.471939 | 194.411990 | 2977.5841 | 15.541327 | 75.97959 | 1.576531 |
| **std** | 7.805007 | 1.705783 | 104.644004 | 849.40256 | 2.758864 | 3.683737 | 0.805518 |
| **min** | 9.000000 | 3.000000 | 68.000000 | 1613.0000 | 8.000000 | 70.00000 | 1.000000 |
| **25%** | 17.00000 | 4.000000 | 105.000000 | 2225.2500 | 13.775000 | 73.00000 | 1.000000 |
| **50%** | 22.75000 | 4.000000 | 151.000000 | 2803.5000 | 15.500000 | 76.00000 | 1.000000 |
| **75%** | 29.00000 | 8.000000 | 275.750000 | 3614.7500 | 17.025000 | 79.00000 | 2.000000 |
| **max** | 46.60000 | 8.000000 | 455.000000 | 5140.0000 | 24.800000 | 82.00000 | 3.000000 |

Table 1 displays basic statistics over the dataset. A note of point is the reduction of the files original 398 instances to 392 as 6 of the instances contained one or more missing attributes. As the amount of instances with missing data points were low, a complete removal of them was found to be appropriate.



**Figure 1**: Histogram plots of the distribution of each parameter against the response variable.



**Figure 2**: Correlation ‘heatmap’ of the auto-mpg.csv data file.

The histograms in Figure 1 allowed for preliminary insight on what possible variables would impact the response variable the most. Both “acceleration” and “model year” show a uniform distribution while the rest of the variables have different degrees of being skewed right. Moreover, the correlation ‘heatmap’ displayed in Figure 2 gave insights into collinearity in the data set. For instance, there is a weak positive correlation between acceleration and model year which is intuitively sound as machining of parts and improvements in engines have taken place over time. Multicollinearity is also present in the dataset, with both displacement and cylinders having strong positive correlation with weight as expected. These observations, as well as the moderately strong positive correlation of origin, model year and acceleration with the response variable, give insight on how feature selection algorithms used on the dataset will perform.

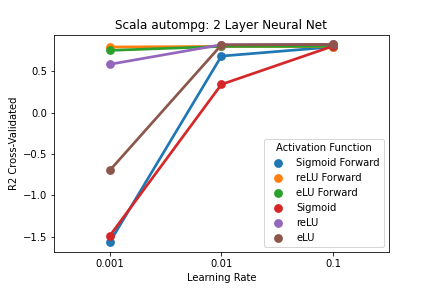
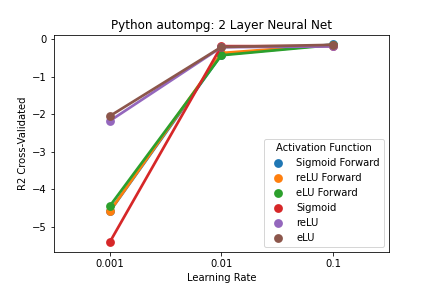
**Proposed Differences in Feature Selection**

The crux of the analysis on the neural nets used with this dataset do not only lie in comparing different activation functions and learning rates but as in comparing if the usage of a feature selection prior to the neural net would have any impact on the results obtained as the neural itself also utilizes some feature selection method. For the analysis done, forward feature selection was chosen and the hypothesis that these forward feature selected models would perform better was made due to the better understanding of how the feature selection algorithm works compared to the “black box” of what occurs inside a neural net. When forward feature selection was used, the features given to the neural net were that of “weight”, “model year” and “origin” being chosen to predict “mpg”.

**Results**

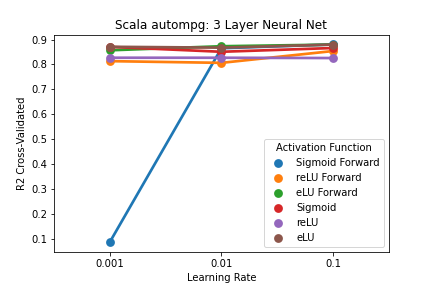
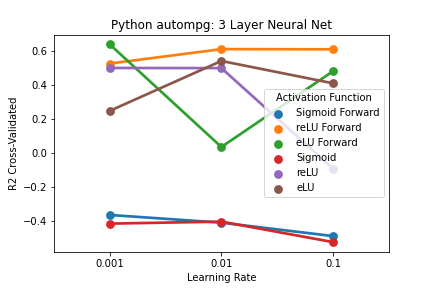
The full results (R2 adjusted, R2 cross validated, feature selection and final MSE) can be found in the accompanied summary file.

Throughout the testing on the autompg dataset, the Python version of Neural Nets, especially the 2 layer level, substantially underperformed in comparison to the Scala versions. Two attempts of normalizing (mean centering and min/max) the data were given with no significant difference between each one found. The Scalation package offered a rescale method on the dataset but attempts to recreate this effect in Python did not succeed. In both languages, negative R2 adjusted and R2 cross validated were seen. These two issues may be an inherent consequence of the size of the autompg data set with only 398 instances in it. This may have led to an automatic poor neural net setting as neural nets thrive on using mass amounts of data to their advantage. Small datasets and neural net usage has been investigated in literature as of late, known as shallow neural nets, and results from these reports are as well not as receptive to deep learning with larger datasets, especially as the number of layers in the model decreases.



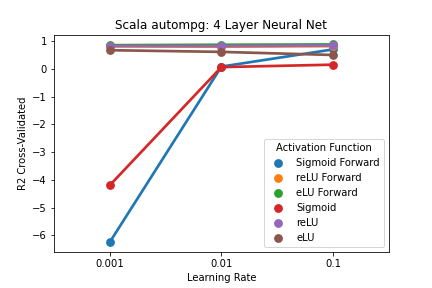
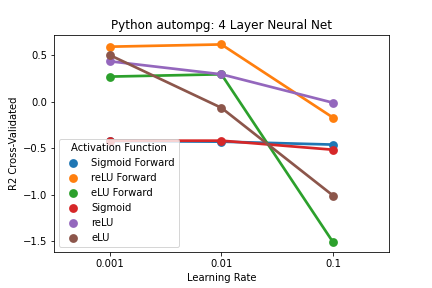
**Figure 3**: R2 cross validated values for the 2 Layer models in both Python and Scala.

Out of the three main neural net models tested, the 2 layer models in both languages performed the worst. Notably the lower learning rates performed significantly worse than at the higher learning rates. Even more interesting, is despite the activation function or if forward feature selection was used, all the variation in the two layer model ended at approximately the same R2 cross validated model. Looking at the Scala models, it could be said the forward selected reLU and eLU models as well as the base reLU models are the best due to their consistent R2 cross validated values across different learning rates.



**Figure 4**: R2 cross validated values for the 3 Layer models in both Python and Scala.

Like in the two layer models, the Scala models performed significantly better than in Python. Despite the aforementioned problem with the size of the data set, the Scala code was able to reliably predict the response variable at 3 layers, regardless of the learning rate, with both elu and eLU forward select performing the best. Likewise, the reLU forward select performed the best in the Python version but lower than its respective Scala model. Both the sigmoid variant models in Python performed the worst.



**Figure 5**: R2 cross validated values for the 4 Layer models in both Python and Scala.

The Scala 4 layer models performed slightly better than the 3 layer models, indicating once a hidden layer is introduced into the model, it does not have any problem with reliably predicting the response variable in the auto-mpg dataset. Still the dataset itself doesn't seem to be suited towards the sigmoid activation function with it performing in the negative region of the R2 cross validated. Interestingly, against the trend seen with the 2 layer and 3 three layer models, the Python models seem to perform better at a lower learning rate, again with reLU performing the best. As well, it's interesting to see that the model actually performed worse than the Python three layer model, which intuitively seems backwards as the more layers introduced, the more parameters can be efficiently updated to better predict the response variable.

**Discussion**

A question that should be answered in terms of the results in the Python code is exactly how the Keras neural net models work in order to address the issue of the small data set. As well, fully understanding how the Scalation rescale method works could help elucidate any changes needed to be done in Python.

In terms of comparing the forward feature select usage and non-usage, there does not seem to really be a significant difference between the two. The way the neural nets themselves pick features to use or drop and how forward feature selection operates, may always arrive at the same result due to the low number of instances in the data set as well as their only being six predictor variables.

Overall, across all the models and learning rates tested, it appears using reLU as the activation function at a 0.1 learning rate results in the best predicting capabilities, with the four layer model with these parameters performing the best.