Nguyen Tran

Analysis

a. Which algorithm performed better?

Overall, the Decision Tree classifier performed the best out of all the models. However, the sequential neural net in its later epochs seemed to perform above or on par with it as well.

The accuracy score of 94% was generally the highest for the Decision Tree classifier. Although, with more epochs, the sequential NN started approaching that number, and the MLP classifier was slightly below. The precision for the not high mpg (0) was identical for the same with the logistic regression model. For the positive class (1) high mpg, it was higher than that of the logistic regression. The recall for both classes was for the Decision Tree was the highest among all of the models.

Logistic Regression

	precision	recall	f1-score	support
0	0.98	0.80	0.88	50
1	0.73	0.96	0.83	28
accuracy			0.86	78
macro avg	0.85	0.88	0.85	78
weighted avg	0.89	0.86	0.86	78

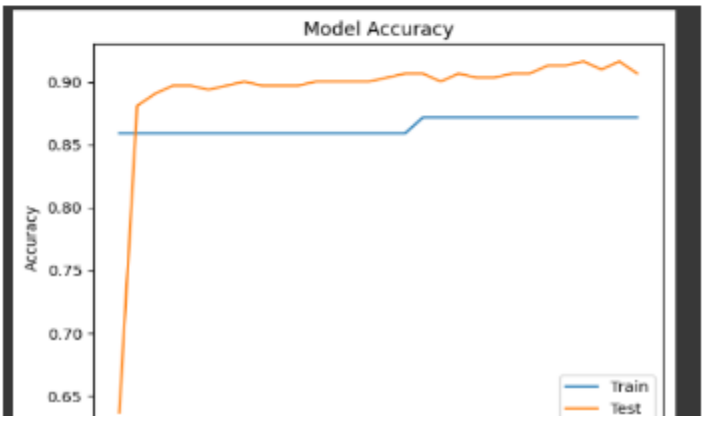
Decision Tree

	precision	recall	f1-score	support
0	0.98	0.92	0.95	50
1	0.87	0.96	0.92	28
accuracy			0.94	78
macro avg	0.92	0.94	0.93	78
weighted avg	0.94	0.94	0.94	78

MLP Classifier

	precision	recall	f1-score	support
0	0.94	0.92	0.93	50
1	0.86	0.89	0.88	28
accuracy			0.91	78
macro avg	0.90	0.91	0.90	78
weighted avg	0.91	0.91	0.91	78

Sequential NN



In general, the better performing algorithm was the decision tree. I think this algorithm performed better than the others because there was no limit on the growth of the tree and the tree wasn't pruned. So, it could be that there was some overfitting to the data going on.

Between writing code in R and writing code in python, I prefer python. It has a similar object structure to other languages that I'm familiar with such as Java, even in the way in which objects are referenced through qualifiers such as the **.** operator, it feels similar to how attributes/fields or methods of objects would be accessed from what I'm more used to. In addition, the way in which we get these objects seem to be bundled into libraries where import statements bring those libraries into our code so we can use them. This process is more familiar to me than what ought to be done in R via the library command and messing with R's package manager. That said, in the brief moments here I've had with python, it does seem like all of the libraries: pandas, numpy, matplotlib, seaborn, etc. are emulating what is already available in R. In that sense, I'm thankful that before starting to get into python's world of objects in addition to those offered by the extended libraries, that I got a foundation with stuff like dataframes and vectors; seeing those being mimicked over in the python world is something that looks familiar now to me. In terms of R versus sklearn, while I don't completely dislike R, I think the structure of sklearn makes it a bit more readable. Everything is broken up by modules in different levels of namespaces. It looks like it's organized, much like Java is, so reading the documentation for sklearn versus that of R in my opinion is a little more understandable. One thing I do think I will miss with R is the ability to specify equations for regressions explicitly using **~** syntax and having some short-hand enabled in there such as **.** and **-** to be able to include or exclude features from the equation. That's not a feature I've seen in python's ecosystem yet and from what I saw of sklearn, it's not a thing there either, so that will be missed. But other than that, in general, I find writing python more comfortable than R.