In the previous three posts We used multiple linear regression, decision trees, gradient boosting, and support vector machine to predict miles per gallon for 2019 vehicles.  It was determined that svm produced the best model.  In this post, We are going to run [TensorFlow](https://www.tensorflow.org/) through R and fit a multiple linear regression model using the same data to predict MPG.

Part 1: [Multiple Linear Regression using R](https://blog.alpha-analysis.com/2019/06/predicting-mpg-for-2019-vehicles-using-r.html)

There are 1253 vehicles in the cars\_19 dataset.  I am simply running mlr using Tensorflow for demonstrative purposes as using lm() in R is more efficient and more precise for such a small dataset.

TensorFlow uses an [algorithm](https://www.tensorflow.org/api_docs/python/tf/estimator/LinearRegressor) that is dependent upon convergence whereas R computes the closed form estimates of beta.  I will be using 11 features and an intercept so R will be inverting a 12 x 12 matrix which is not computationally expensive with today’s technology.

The dataset below of 11 features contains 7 factor variables and 4 numeric variables.

str(cars\_19)  
'data.frame': 1253 obs. of 12 variables:  
 $ fuel\_economy\_combined: int 21 28 21 26 28 11 15 18 17 15 ...  
 $ eng\_disp : num 3.5 1.8 4 2 2 8 6.2 6.2 6.2 6.2 ...  
 $ num\_cyl : int 6 4 8 4 4 16 8 8 8 8 ...  
 $ transmission : Factor w/ 7 levels "A","AM","AMS",..: 3 2 6 3 6 3 6 6 6 5 ...  
 $ num\_gears : int 9 6 8 7 8 7 8 8 8 7 ...  
 $ air\_aspired\_method : Factor w/ 5 levels "Naturally Aspirated",..: 4 4 4 4 4 4 3 1 3 3 ...  
 $ regen\_brake : Factor w/ 3 levels "","Electrical Regen Brake",..: 2 1 1 1 1 1 1 1 1 1 ...  
 $ batt\_capacity\_ah : num 4.25 0 0 0 0 0 0 0 0 0 ...  
 $ drive : Factor w/ 5 levels "2-Wheel Drive, Front",..: 4 2 2 4 2 4 2 2 2 2 ...  
 $ fuel\_type : Factor w/ 5 levels "Diesel, ultra low sulfur (15 ppm, maximum)",..: 4 3 3 5 3 4 4 4 4 4 ...  
 $ cyl\_deactivate : Factor w/ 2 levels "N","Y": 1 1 1 1 1 2 1 2 2 1 ...  
 $ variable\_valve : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 2 2 ...

The factors need to be transformed into a format TensorFlow can understand.

cols <- feature\_columns(  
 column\_numeric(colnames(cars\_19[c(2, 3, 5, 8)])),  
 column\_categorical\_with\_identity("transmission", num\_buckets = 7),  
 column\_categorical\_with\_identity("air\_aspired\_method", num\_buckets = 5),  
 column\_categorical\_with\_identity("regen\_brake", num\_buckets = 3),  
 column\_categorical\_with\_identity("drive", num\_buckets = 5),  
 column\_categorical\_with\_identity("fuel\_type", num\_buckets = 5),  
 column\_categorical\_with\_identity("cyl\_deactivate", num\_buckets = 2),  
 column\_categorical\_with\_identity("variable\_valve", num\_buckets = 2)  
 )

Create an input function:

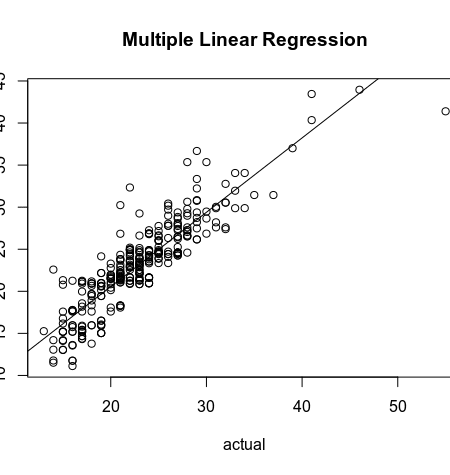
#input\_fn for a given subset of data  
cars\_19\_input\_fn <- function(data, num\_epochs = 1) {  
 input\_fn(  
 data,  
 features = colnames(cars\_19[c(2:12)]),  
 response = "fuel\_economy\_combined",  
 batch\_size = 64,  
 num\_epochs = num\_epochs  
 )  
}

Train, evaluate, predict:

model <- linear\_regressor(feature\_columns = cols)  
  
set.seed(123)  
indices <- sample(1:nrow(cars\_19), size = 0.75 \* nrow(cars\_19))  
train <- cars\_19[indices, ]  
test <- cars\_19[-indices, ]  
  
#train model  
model %>% train(cars\_19\_input\_fn(train, num\_epochs = 1000))  
  
#evaluate model  
model %>% evaluate(cars\_19\_input\_fn(test))  
  
#predict  
yhat <- model %>% predict(cars\_19\_input\_fn(test))

Results are very close to the R closed form estimates:

postResample(yhat, y)  
 RMSE Rsquared MAE   
2.5583185 0.7891934 1.9381757

[](https://i0.wp.com/s3-us-west-1.amazonaws.com/alpha-analysis.com/Pictures/TensorFlow/r_tensorflow.png?ssl=1)