COE 379L - Project 1

James Grant Robinett (jgr2722)

Introduction

Located in this folder are two files, this PDF “Project 1 Writeup JGR2722” and the code for parts 1 and 2, “part1_part2.ipynb”. The dataset used in this project was downloaded from the class github repository using this link: https://raw.githubusercontent.com/joestubbs/coe379L-sp24/master/datasets/unit01/project1.data, and was saved as “cars.csv”, which I will leave included just for convenience when running my code.

Data Pre-Processing

The first step I took after reading the ‘cars.csv’ data into a pandas dataframe was to look at the structure of the dataframe. This was done using the `cars.info()` function, which displays not only the size of the dataframe (in this case, 397 rows and 9 columns), but the types of each column of data within the pandas dataframe as well. Next, I checked if there were any null values within the loaded cars data by using `cars.isnull().sum()`, and luckily there were no null values detected. With no necessary additions needed, I then moved on to converting the `cars['horsepower']` column from an object to numerical data. The type of the horsepower object is a string, so first I tried converting the horsepower using the `astype(float)` function. However, when trying to use this function, many errors were thrown saying there were non-numeric string values within the column. This means I had to look closer at the data.

The error did provide some interesting information, indicating that a “?” character was seen in the column. To find where these rows were, I used `cars[cars['horsepower'] == '?']` to return only the rows that included the question mark. Success! From this, I saw that six rows contained a “?” value in the horsepower column, which means that this is technically a null value, just hidden a bit. The tricky part was deciding how to replace these values with numerical data. I decided to find the mean horsepower for the cars grouped by the number of cylinders the car has. To achieve this, a few steps needed to be taken.

First, I created a small separate data frame “cars2” (great movie by the way) which contained only the rows that did not include the missing horsepower values. This was done to convert the column to numeric values without any errors and find the means. I saved these means to a small list, cylinder_means, and then used the replacement loop discussed in class to replace the missing horsepower values in the original “cars” dataframe. Now, I was able to convert the entire horsepower column to float values using the astype() function.

As a quick final check that my data had no missing values, I also checked to see if any 0 values were included in the other numerical columns, and found that everything was in order, no more changes needed. I also used the function `cars.describe()` to check if there were any outliers or strange looking values for each of the columns, but found nothing of the sort. The second to last step I took processing this data was to drop the ‘origin’ column from the dataframe. This is because there were only three numeric values possible (1, 2 and 3), and did not seem to correlate with any major changes in the other numerical values. Additionally, I was not sure why car origin would matter in this dataframe, so I made the decision that it would not be needed. Lastly, I encoded the cars_name column so that it could be processed by the linear

regression model using OneHotEncoder. Although this may cause some bloat, I think for this project it was appropriate for analysis.

Machine Learning Linear Regression

With the data processed and prepared for analysis, we can now start to look at how to fit a model to predict fuel efficiency. The first step in finding a linear regression model is to define what our explanatory and response variables are. In this case, `cars['mpg']` was used as our response variable (y) and everything else was used for the explanatory (X) variable (This includes cylinders, displacement, horsepower, weight, acceleration, model_year and encoded car_names). This was done by dropping the 'mpg' column from the dataset and using the remaining columns.

With X and Y defined, we could now move to training the linear regression model using Scikit-Learn. The module used to train and test this model comes from Scikit-Learn's native library, `from sklearn.model_selection import train_test_split`. Using train test split, the module automatically splits the data into training and test sets for model processing and testing. This was done using the line `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)`, where test_size=0.2 means that 20 percent of the data was used to test, and random_state=1 was used for reproducibility. It is important to note, however, that the stratify component was not used in this train test split. This was a deliberate choice, due to stratification only being possible when classification can be done on a categorical variable. 'Mpg' is a numeric, continuous variable, stratification is not needed.

Finally, with the training and testing data saved to their respective variables, we can now train the linear regression model. Using the LinearRegression module imported from scikit-learn, fitting a model to the processed data is as easy as calling one function. We use `lr=LinearRegression()` followed by `lr.fit(X_train, y_train)` to train a linear regression model on our X_train and y_train data. The computer does the rest of the heavy lifting, and we have now successfully trained a linear regression model on our cars dataset!

Now, with the model trained, we need to test the accuracy of the module to make sure it is an accurate predictor. This is the part where I got a bit stuck, since normally we would use the accuracy_score module provided by scikit-learn. However, this did not seem to work whenever I tried to run this on the trained model I had produced. To solve this problem, I asked chatGPT4 what I could do to test the accuracy of my linear regression model, or if I had messed up the application of accuracy_score. The AI responded with this: "...using the accuracy_score function from sklearn.metrics is not appropriate because accuracy_score is intended for classification tasks… In regression, your target variable (y) is continuous, and the predictions are also continuous values". This cleared up my confusion on how to best represent the model accuracy and guided me to use a different approach of analysis.

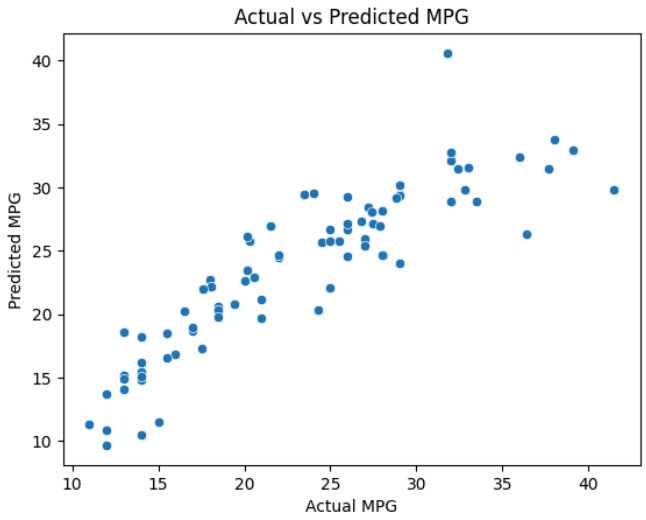
With the accuracy analysis cleared up, the best way I decided to show model accuracy was with Mean Absolute Error (MAE), Mean Squared Error (MSE) and R2 Score (R2). The MAE describes the average of the absolute differences between predictions and actual values, the MSE describes the average of the squared differences between predictions and actual values, and the R2 score describes how well the data points fit a line or curve. All of these metrics are included in the `sklearn.metrics` module, and were used comparing both the training and testing data to the trained linear regression model. An example of how these values were found

looks like this `mae_test = mean_absolute_error(y_test, y_pred_test)`. These were the following accuracy results:

```
Test MAE: 2.6409273036381777
Train MAE: 0.3929869901223548
Test MSE: 12.090325407820675
Train MSE: 0.7469805502142375
Test R2: 0.7854085138853747
Train R2: 0.9879651531036833
```

These accuracy results show that the model is promising and can decently predict a correct ‘mpg’ value when seeing new data, but still can be improved. The test MAE is within 2.64 units of predicting the actual values, while the train MAE is within 0.39 units. The test MSE is relatively small, meaning that larger errors are not seen in the prediction. Finally, the test R2 score is 0.785, which means the model can explain around 78.5% of the variance seen. The train R2 is 0.987, which is to be expected since the data was used to train the model.

In conclusion, I am confident that the model has good/moderate predictive ability. The test MAE being reasonably low, 2.64, means that the model, on average, has good predictions on actual fuel efficiency values. Additionally, the R-squared score of 0.785 for the test set is quite promising, showing that the model can explain approximately 78.5% of the variance in mpg fuel efficiency. However, the difference between the test and train R-squared scores show that the model may be overfitting the training data. This means that there is room for improvement, and further analysis needs to be done to see if the accuracy of the model can be tuned and refined.



(Fig. 1) This graph shows the comparison between actual mpg values and predicted mpg values. As we can see, the model generally predicts the correct value for MPG.