MLWithSKLearn

November 6, 2022

1 Library Imports

[1]:
```
# Imports
import pandas as pd
import numpy as np
import seaborn as sb

from sklearn import tree
from sklearn import preprocessing as pp
from sklearn.model_selection import train_test_split as tts
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
 ↪recall_score, f1_score, classification_report
from sklearn.neural_network import MLPClassifier

# Seed
np.random.seed(1234)
```

2 Loading the Data Set

The below code loads the data set given to us, which is simply an automobile data set that can be used to predict miles per galon (mpg).

[2]:
```
# Load data set
Data = pd.read_csv('Auto.csv');
print( Data.head(), '\n' )
print("Dimensions:", Data.shape)
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0  70.0
1  15.0          8         350.0         165    3693          11.5  70.0
2  18.0          8         318.0         150    3436          11.0  70.0
3  16.0          8         304.0         150    3433          12.0  70.0
4  17.0          8         302.0         140    3449           NaN  70.0

   origin                       name
0       1  chevrolet chevelle malibu
```

```
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino

Dimensions: (392, 9)
```

3 Correcting Data Types

The code below prints a description of each column in the data set.

[3]:
```
# Describe MPG
Data.mpg.describe()
```

[3]:
```
count    392.000000
mean      23.445918
std        7.805007
min        9.000000
25%       17.000000
50%       22.750000
75%       29.000000
max       46.600000
Name: mpg, dtype: float64
```

Based off of the output above, We can observe that the MPG's range is between 9.0 - 46.6, and that its average is roughly. 23.45.

[4]:
```
# Describe Weight
Data.weight.describe()
```

[4]:
```
count     392.000000
mean     2977.584184
std       849.402560
min      1613.000000
25%      2225.250000
50%      2803.500000
75%      3614.750000
max      5140.000000
Name: weight, dtype: float64
```

Likewise for our Weight column, its range is between 1613.0 - 5140.0, and its average is roughly 2977.58.

[5]:
```
# Describe Year
Data.year.describe()
```

[5]:
```
count    390.000000
mean      76.010256
std        3.668093
```

```
min        70.000000
25%        73.000000
50%        76.000000
75%        79.000000
max        82.000000
Name: year, dtype: float64
```

And finally for our Year column, its range is between 70.0 - 82.0, and its average is roughly 76.01.

The output below shows each of the columns' data types. As you can see, some data best represented categorically is being represented as an integer.

[6]:
```python
# Check the data types of the columns
Data.dtypes
```

[6]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

We'll go ahead and convert those columns - cylinders and origin - to categorical data types. The output below shows the changes being made, with the categorical columns now being of type int8 instead of type int64.

[7]:
```python
# Change categorical data to categorical types
Data.cylinders = Data.cylinders.astype('category').cat.codes
Data.origin = Data.origin.astype('category').cat.codes

# Verify changes
Data.dtypes
```

[7]:
```
mpg             float64
cylinders          int8
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin             int8
name             object
dtype: object
```

4 Dealing with NAs

Next, we'll remove any NA rows from our data frame. It's relatively simple in Python, as shown below.

[8]:
```
Data = Data.dropna()
print('Dimensions:', Data.shape)
```

```
Dimensions: (389, 9)
```

5 Modifying Columns

Since we're using this data set for classification, we'll want to set up a column to be used that will classify the mpg by whether or not it is greater than the average. Those greater than the average will be considered 'high'.

[9]:
```
# Create & add mpg_high column
Data['mpg_high'] = Data.mpg.apply(lambda x: 1 if x > np.mean(Data.mpg) else 0)
Data.mpg_high = Data.mpg_high.astype('category').cat.codes

# Delete the mpg and name columns
Data = Data.drop(columns=['mpg', 'name'])

# Output first few rows of our newly modified data frame
Data.head()
```

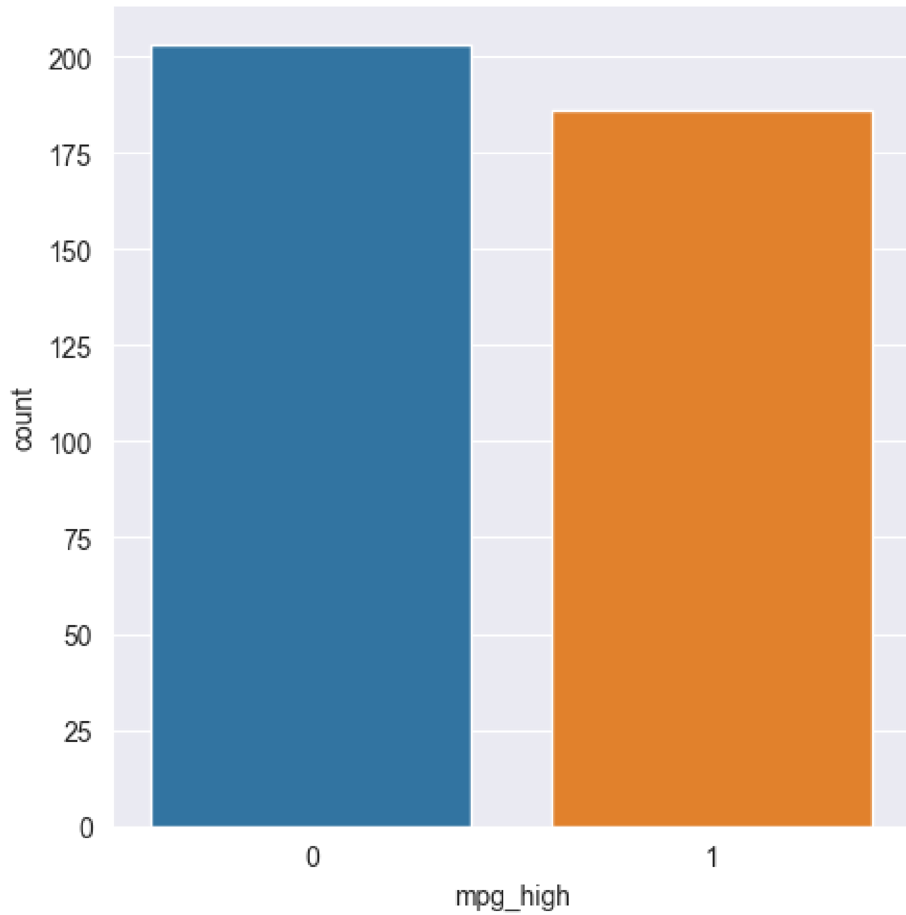
[9]:
```
   cylinders  displacement  horsepower  weight  acceleration  year  origin  \
0          4         307.0         130    3504          12.0  70.0       0
1          4         350.0         165    3693          11.5  70.0       0
2          4         318.0         150    3436          11.0  70.0       0
3          4         304.0         150    3433          12.0  70.0       0
6          4         454.0         220    4354           9.0  70.0       0

   mpg_high
0         0
1         0
2         0
3         0
6         0
```

6 Data Exploration (w/ Graphs)

Now we'll explore the data by creating graphs, and seeing what we can learn about the data from these graphs.

[10]:
```
# Seaborn catplot; mpg_high
sb.catplot(data=Data, x='mpg_high', kind='count')
```

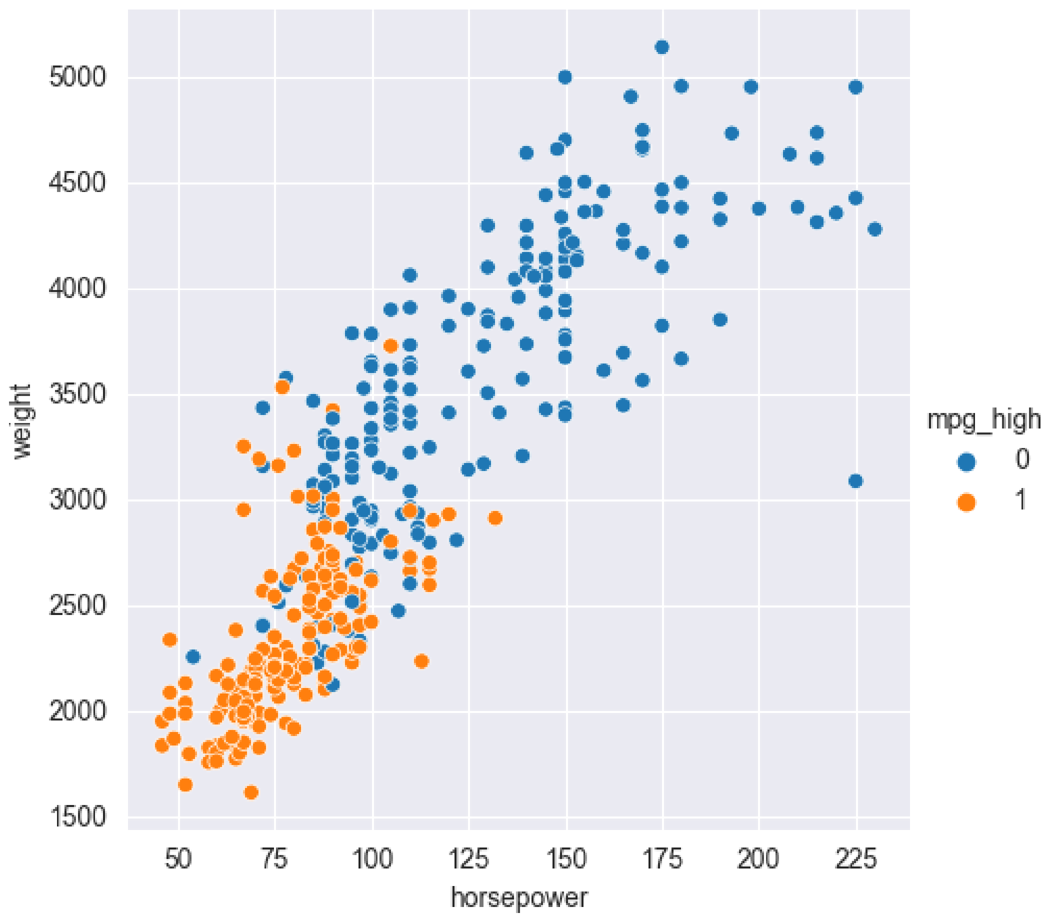
[10]: <seaborn.axisgrid.FacetGrid at 0x20c709ebe80>



The above graph simply tells us that the number of vehicles that fall above the average is somewhat notably lower than those that do not. For the most part, we can observe that the data is relatively balanced.

[11]:
```
# Seaborn relplot; x=horsepower, y=weight, hue/style=mpg_high
sb.relplot(data=Data, x='horsepower', y='weight', hue='mpg_high')
```

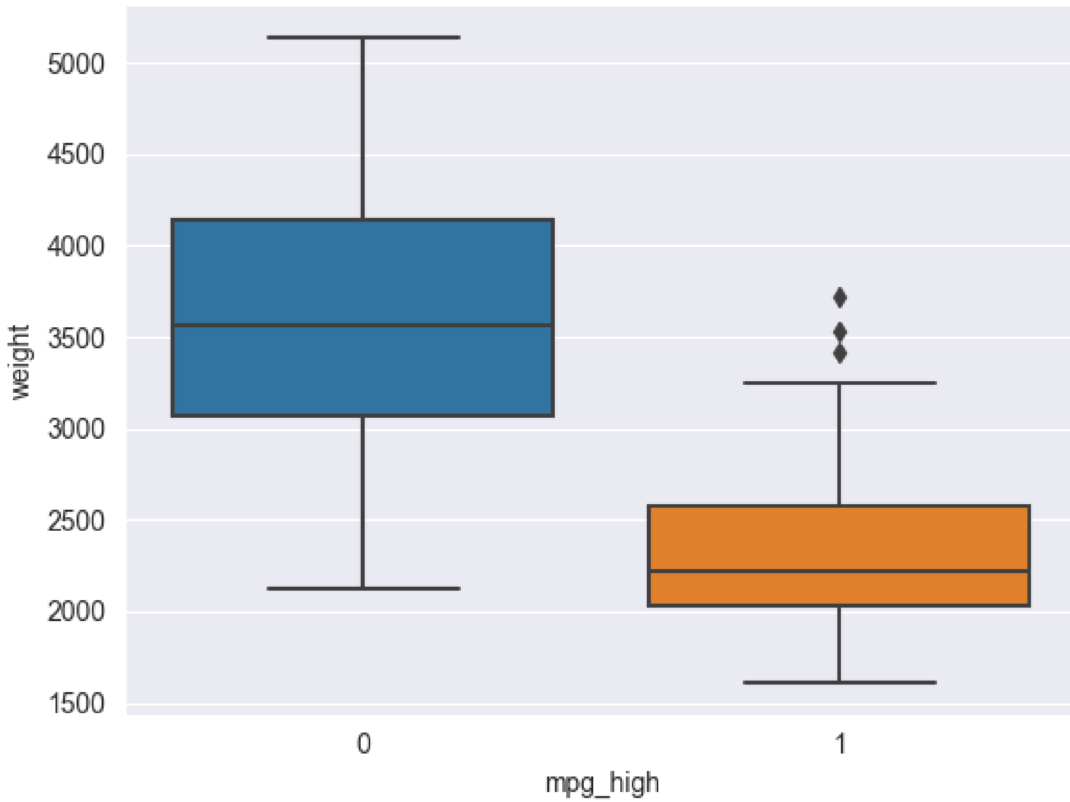
[11]: <seaborn.axisgrid.FacetGrid at 0x20c72c1bac0>



The above graph tells us that vehicles with low weight and horsepower tend to have high mpg rates, while those with high horsepower or high weight tend to have low mpg rates. This is clear by the fact that the data clusters up into the bottom-left most portion of the graph, in the lower ends of both values, with some outliers. This doesn't seem to be the case for all vehicles with low weight and horsepower, however, as there are a number of blue dots that fall below the average that lie within this cluster. It does seem though that weight and horsepower are rather linearly related.

[12]:
```
# Seaborn boxplot; x=mpg_high, y=weight
sb.boxplot(data=Data, x='mpg_high', y='weight')
```

[12]:
```
<AxesSubplot: xlabel='mpg_high', ylabel='weight'>
```



The above graph tells us that vehicles with lower weights tend to have high mpg rates, while vehicles with higher weights tend to have lower mpg rates. The relationship between mpg and weight seems to be inverse.

7 Train/Test Split

Now, we'll split the data to train/test in preparation for the machine learning algorithms. We'll give it seed 1234 so that we'll always get the same results.

[13]:
```
# Split train/test
x = Data.iloc[:, 0:7]
y = Data.mpg_high
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, random_state=1234)

# Output dimensions
print('Train shape:', x_train.shape)
print('Test shape:', x_test.shape)
```

```
Train shape: (311, 7)
Test shape: (78, 7)
```

8 Model Training & Prediction

Now we can finally start performing machine learning on the data using a variety of models! Specifically, we'll use the Logistic Regression, Decision Tree, and the newly covered Neural Network algorithms and experiment with their outputs.

8.1 Logistic Regression

First, we'll train a Logistic Regression Model and run predictions on the test data using it.

[14]:
```
# Train model
lrm = LogisticRegression(max_iter=1000, solver='lbfgs')
lrm.fit(x_train, y_train)

# Make predictions
pred_lrm = lrm.predict(x_test)

# Evaluate predictions
print('\nClassification Report:')
print(classification_report(y_test, pred_lrm))
print('Notable CR Metrics:')
print('Accuracy:', accuracy_score(y_test, pred_lrm))
print('Precision:', precision_score(y_test, pred_lrm))
print('Recall:', recall_score(y_test, pred_lrm))
print('F1:', f1_score(y_test, pred_lrm))
confusion_matrix(y_test, pred_lrm)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        50
           1       0.78      1.00      0.88        28

    accuracy                           0.90        78
   macro avg       0.89      0.92      0.89        78
weighted avg       0.92      0.90      0.90        78

Notable CR Metrics:
Accuracy: 0.8974358974358975
Precision: 0.7777777777777778
Recall: 1.0
F1: 0.8750000000000001
```

[14]:
```
array([[42,  8],
       [ 0, 28]], dtype=int64)
```

We can see from the results that we achieve an 89.74% accuracy on this data set with Logistic Regression. It's notable to mention that it would seem all of our incorrect predictions are false negatives.

8.2 Decision Tree

Second, we'll build a Decision Tree and run predictions on the test data using it.

[15]:
```
# Train model
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)

# Make predictions
pred_dt = dt.predict(x_test)

# Evaluate predictions
print('\nClassification Report:')
print(classification_report(y_test, pred_dt))
print('Notable CR Metrics:')
print('Accuracy:', accuracy_score(y_test, pred_dt))
print('Precision:', precision_score(y_test, pred_dt))
print('Recall:', recall_score(y_test, pred_dt))
print('F1:', f1_score(y_test, pred_dt))
confusion_matrix(y_test, pred_dt)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.87      0.93      0.90        28

    accuracy                           0.92        78
   macro avg       0.91      0.92      0.92        78
weighted avg       0.93      0.92      0.92        78

Notable CR Metrics:
Accuracy: 0.9230769230769231
Precision: 0.8666666666666667
Recall: 0.9285714285714286
F1: 0.896551724137931
```

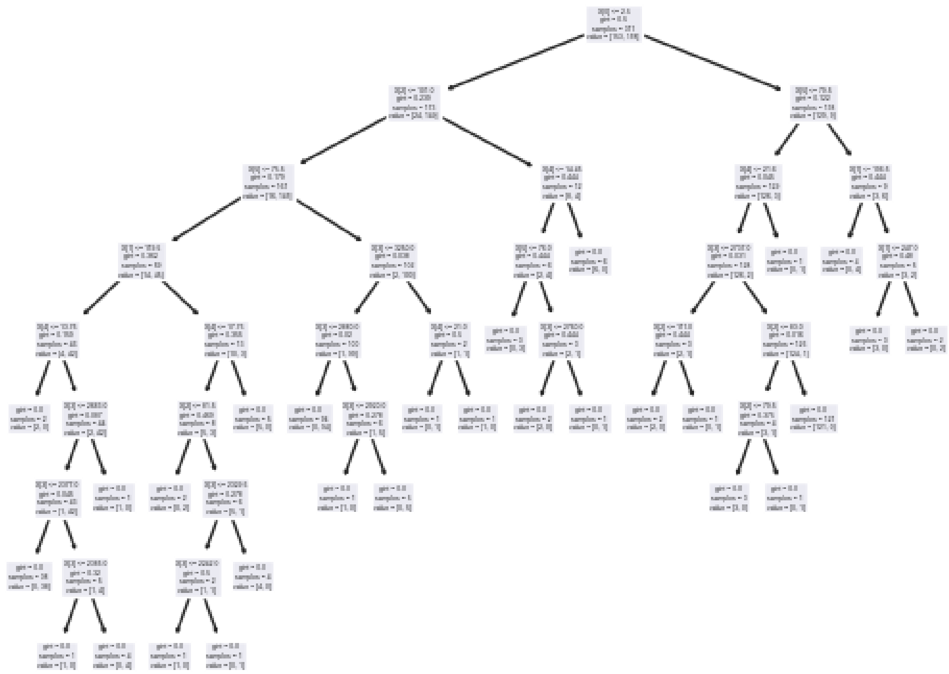
[15]:
```
array([[46,  4],
       [ 2, 26]], dtype=int64)
```

The Decision Tree performed better than the Logistic Regression model, which came as a surprise to me given one of the previous projects where we'd utilized decision trees. Though given the structure of the data, it makes sense why the decision tree model is performing better given what we learned about the correlation between the weight/horsepower and the mpg rate.

Plot of the tree:

```
[16]: tree.plot_tree(dt)
      ;
```

```
[16]: ''
```



8.3 Neural Network

Finally, we'll build a Neural Network and run predictions on the test data using it. We use 7 hidden nodes (the number of predictors) and arrange them into two separate layers.

```
[17]: # Normalize the data
      scaler = pp.StandardScaler().fit(x_train)
      x_train_scaled = scaler.transform(x_train)
      x_test_scaled = scaler.transform(x_test)

      # Train model
      nn1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5,2), max_iter=1000,␣
       ↪random_state=1234)
      nn1.fit(x_train_scaled, y_train)

      # Make predictions
      pred_nn1 = nn1.predict(x_test_scaled)
```

```python
# Evaluate predictions
print('\nClassification Report:')
print(classification_report(y_test, pred_nn1))
print('Notable CR Metrics:')
print('Accuracy:', accuracy_score(y_test, pred_nn1))
print('Precision:', precision_score(y_test, pred_nn1))
print('Recall:', recall_score(y_test, pred_nn1))
print('F1:', f1_score(y_test, pred_nn1))
confusion_matrix(y_test, pred_nn1)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.86      0.90        50
           1       0.78      0.89      0.83        28

    accuracy                           0.87        78
   macro avg       0.86      0.88      0.86        78
weighted avg       0.88      0.87      0.87        78

Notable CR Metrics:
Accuracy: 0.8717948717948718
Precision: 0.78125
Recall: 0.8928571428571429
F1: 0.8333333333333334
```

[17]:
```
array([[43,  7],
       [ 3, 25]], dtype=int64)
```

Looking at the results above, we can see that the model performs worse than the previous two models, in terms of accuracy. I believe this is largely due to the layering sizes that were chosen, which may require further experimentation.

Moreover, I built another Neural Network to attempt to get better results on the data, but this time using 6 hidden nodes (2/3 of input layer size [5] + output layer size [1]). We'll also use a different solver; sgd.

[18]:
```python
# Train model
nn2 = MLPClassifier(solver='sgd', hidden_layer_sizes=(6,), max_iter=1000,
 ↪random_state=1234)
nn2.fit(x_train_scaled, y_train)

# Make predictions
pred_nn2 = nn2.predict(x_test_scaled)

# Evaluate predictions
print('\nClassification Report:')
print(classification_report(y_test, pred_nn2))
```

```
print('Notable CR Metrics:')
print('Accuracy:', accuracy_score(y_test, pred_nn2))
print('Precision:', precision_score(y_test, pred_nn2))
print('Recall:', recall_score(y_test, pred_nn2))
print('F1:', f1_score(y_test, pred_nn2))
confusion_matrix(y_test, pred_nn2)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        50
           1       0.78      1.00      0.88        28

    accuracy                           0.90        78
   macro avg       0.89      0.92      0.89        78
weighted avg       0.92      0.90      0.90        78

Notable CR Metrics:
Accuracy: 0.8974358974358975
Precision: 0.7777777777777778
Recall: 1.0
F1: 0.8750000000000001
```

[18]:
```
array([[42,  8],
       [ 0, 28]], dtype=int64)
```

The results we get are definitely better, and now perform the same as the Logistic Regression model. I think this is doing better largely because of the trial-and-error of tuning the hidden layer size to something better, as the topology of the neural network itself makes a large difference on the performance.

9 Analysis

Out of all of the algorithms we'd run, the Decision Tree algorithm performed the best, achieving the highest accuracy score.

The Decision Tree model achieved the best accuracy and precision scores, while Logistic Regression and the Second Neural Network both achieved the best recall scores.

I think the Decision Tree model outperformed the rest due to the structure of the data, and the problem at hand. The data has a number of important features, as we explored in our exploration of the data before, which is likely why it performed the best. As decision trees are able to acquire better accuracy when it builds off of the most important features.

Between R and SKLearn, I have a stronger preference for R, mainly due to the helpful metrics it gives you, despite the syntax being a little less trivial than SKLearn. It's also likely because we've been using it for most of the class. However, SKLearn is in some ways more preferable for its ease of use. But overall, at this very moment I'd likely choose R over SKLearn if I were given a choice.