Machine Learning with SKLearn

Date: April 4, 2023 By: David Teran

Overview

This notebook will cover machine learning using python scripts and the sciki-learn library for machine learning in python. This will be done using Google Colab to read in the data and run machine learning algorithms. The dataset that will be used for this notebook is named "Auto.csv" and is provided beforehand.

Data Exploration

First, the csv file "Auto.csv" will be read in using the pandas library and the dimensions and first few rows will be printed out to verify that the data is read in correctly.

```
#Read in the data and print out the first few rows and dimensions of data
import pandas as pd

autoData = pd.read_csv('Auto.csv')
print(autoData.head())
print('\nDimensions of the Data Frame Auto:', autoData.shape)
```



```
                                    t  horsepower  weight  acceleration  year  \
                                    0         130    3504          12.0  70.0
                                    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 of the Data Frame Auto: (392, 9)
```

Once the data has been read in and the dimensions and the first few rows of the dataset have been printed, some data exploration can be done. The describe() function will be used for some of the columns to obtain details on the dataset.

The describe function will be used to get details on the mpg, weight, and year data columns.

```
#Using the normal describe funcion will only show the average and only the
#minumum and maximum values. A function has been created to obtain the range
def describe_new(df):
    df1 = df.describe()
    df1.loc["range"] = df1.loc['max'] - df1.loc['min']
    return df1
print(describe_new(autoData[["mpg", "weight", "year"]]))

#The mean values for the three columns:
#MPG Mean = 23.445918
#weight Mean = 2977.584184
#year Mean = 76.010256

#The range values for the three columns
#mpg range = 37.6
#weight range = 3527
#year range = 12
```

```
             mpg       weight        year
count  392.000000   392.000000  390.000000
mean    23.445918  2977.584184   76.010256
std      7.805007   849.402560    3.668093
min      9.000000  1613.000000   70.000000
25%     17.000000  2225.250000   73.000000
50%     22.750000  2803.500000   76.000000
75%     29.000000  3614.750000   79.000000
max     46.600000  5140.000000   82.000000
range   37.600000  3527.000000   12.000000
```


data present. First, the types of data present must be checked first.

```
print(autoData.dtypes)
```

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

There are different methods when it comes to changing data types of columns, using both cat.codes and without cat.codes. Any changes done to the data types can be checked using dtypes.

```
#using cat.codes
autoData.cylinders = autoData.cylinders.astype('category').cat.codes

#not using cat.codes
autoData.origin = autoData.origin.astype('category')

#check the data types again
print(autoData.dtypes)
```

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

The dtype call shows that the cat.codes data type for cylinders is int8, meaning that the column data type is categorical using 1's and 0's. The origin column also has been changed to a categorical data type, but without the use of cat.codes, changes it to use 'yes' and 'no' instead of 1's and 0's.

Next is dealing with any NA's present in the dataset. Using isnull, the dataset is check for NA's in the data before dropping them from the dataset.

Saved successfully!

```
#Drop the NA's and reprint dimensions
autoData = autoData.dropna()
print('\nDimensions of the Data Frame Auto:', autoData.shape)
```

```
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
mpg_high        0
dtype: int64

Dimensions of the Data Frame Auto: (389, 8)
```

With the NA's removed from the dataset, the dimensions from the dataset have slightly changed. The next step is to see about adding columns. A new column mpg_high will be added and the mpg and name columns will be dropped from the dataset.

```
#Creating new data column based on mpg column
autoData['mpg_high'] = [1 if x > autoData['mpg'].mean() else 0 for x in autoData['mpg']]
autoData.mpg_high = autoData.mpg_high.astype('category').cat.codes
#Delete name and mpg data columns to have algorithm predict from mpg_high
autoData = autoData.drop(columns=['mpg', 'name'])

#print out first few rows of updated dataset
print(autoData.head())
```

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

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

With that done, the mpg_high can be used for predictions. The next thing for data exploration is the use of graphs. The seaborn package in Python will be used for converting the data in the data frame onto a graph, using several different columns as the x and y axis.First, the mpg_high column will be

Saved successfully!

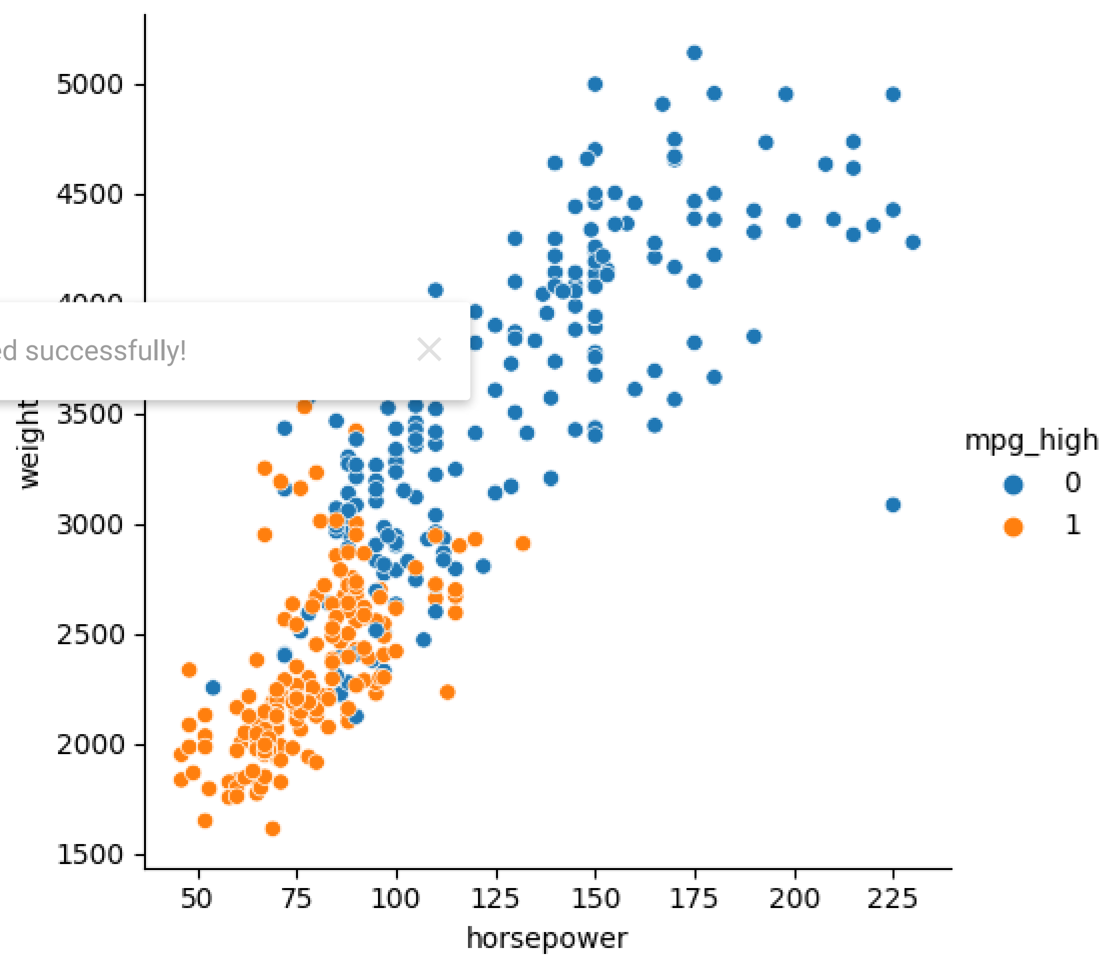
```
import seaborn as sns

#plotting out mpg_high
sns.catplot(x='mpg_high', kind="count", data=autoData)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9e68978a90>
```

The graph above is a categorical plot done using the mpg_high column. The next kind of plot is a relation plot using horsepower as the x-axis and weight on the y-axis.

```
sns.relplot(x='horsepower', y='weight', data=autoData, hue=autoData.mpg_high)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9e61002880>
```

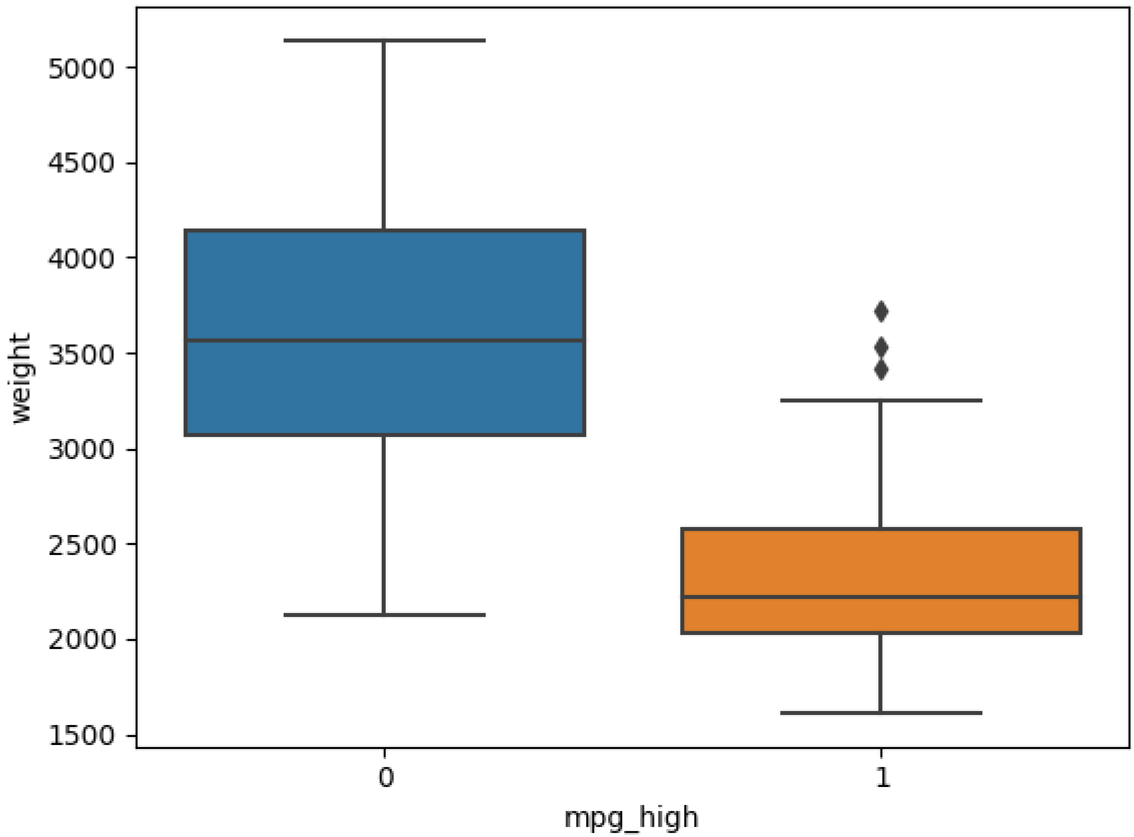


The relational plot data does show that lighter vehicles with lower horsepower have a higher mpg compared to heavier vehicles with higher horsepower. The data also splits into 2 different clusters.

The last plot used on this data is a boxplot with mpg_high on the x-axis and weight on the y-axis.

```
sns.boxplot(x='mpg_high', y='weight', data=autoData)
```

```
<Axes: xlabel='mpg_high', ylabel='weight'>
```



Saved successfully!

itself...

Next, for using some ML algorithm functions on the data set, the data will be split into train/test sets using an 80/20 split.

```
#import from sklearn
from sklearn.model_selection import train_test_split

#Split the data to train/test
X = autoData.iloc[:, 0:7]
y = autoData.loc[:, 'mpg_high']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)

print('train size:', X_train.shape)
```

```
print('test size:', X_test.shape)
print(X_train.head())
```

```
train size: (311, 7)
test size: (78, 7)
     cylinders  displacement  horsepower  weight  acceleration  year origin
184          1         101.0          83    2202          15.3  76.0      2
355          3         145.0          76    3160          19.6  81.0      2
57           1          97.5          80    2126          17.0  72.0      1
170          1          90.0          71    2223          16.5  75.0      2
210          4         350.0         180    4380          12.1  76.0      1
```

With a train test set created, we can train a logistic regression model using solver lbfgs.

```
#Train the logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

autoLR = LogisticRegression(solver='lbfgs', max_iter=150)
autoLR.fit(X_train, y_train)
autoLR.score(X_train, y_train)

#Test and evaluate the model
predLR = autoLR.predict(X_test)
print("Prediction: ", predLR)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, predLR))
print('precision score: ', precision_score(y_test, predLR))
print('recall score: ', recall_score(y_test, predLR))
print('f1 score: ', f1_score(y_test, predLR))
```



```
print(classification_report(y_test, predLR))
```

```
Prediction:  [0 0 0 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0
 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0
 1 1 0 0]
accuracy score:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307

              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
```

Next is a decision tree, using the same classification report to print the metrics.

```
#Create decision tree model
from sklearn.tree import DecisionTreeClassifier

autoTree = DecisionTreeClassifier()
autoTree.fit(X_train, y_train)

#Predict and evaluate
predTree = autoTree.predict(X_test)
print("Prediction: ", predTree)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, predTree))
print('precision score: ', precision_score(y_test, predTree))
print('recall score: ', recall_score(y_test, predTree))
print('f1 score: ', f1_score(y_test, predTree))
print('\n')

print(classification_report(y_test, predTree))
```

```
Prediction:  [0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 1 0
 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 1 1 1 1 0
 0 1 0 0]
accuracy score:  0.8974358974358975
                           33334
                        29
f1 score:  0.8620689655172413
```

Saved successfully!

```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92        50
           1       0.83      0.89      0.86        28

    accuracy                           0.90        78
   macro avg       0.89      0.90      0.89        78
weighted avg       0.90      0.90      0.90        78
```

Next is the neural networks to see if there is a better performance.

```
#Testing 2 different neural network topologies
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor

#Neural Network Classification
autoNNClass = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(4, 2), max_iter=500, random_s
autoNNClass.fit(X_train, y_train)

#Neural Network Regression
autoNNReg = MLPRegressor(hidden_layer_sizes=(6, 4), solver='lbfgs', max_iter=1500, random_sta
autoNNReg.fit(X_train, y_train)

#Predict and evaluate
predClass = autoNNClass.predict(X_test)
from sklearn.metrics import mean_squared_error, r2_score
print('NN Classification mse=', mean_squared_error(y_test, predClass))
print('NN Classification correlation=', r2_score(y_test, predClass))

predNNReg = autoNNReg.predict(X_test)
print('NN Regression mse=', mean_squared_error(y_test, predNNReg))
print('NN Regression correlation=', r2_score(y_test, predNNReg))
```

```
NN Classification mse= 0.11538461538461539
NN Classification correlation= 0.49857142857142844
NN Regression mse= 0.08135940793345033
NN Regression correlation= 0.6464352586663487
```

When comparing both models, the neural network Regression model performed better than the neural classification ... er

Saved successfully!

Overall, the Decision Tree performed better than the other algorithms. In terms of accuracy and precision, the Decision Tree algorithm performed better, but falls short when it comes to recall. Logistic Regression has a higher recall value compared to the Decision Tree. As to why the better-performing algorithm could have outperformed the other, it is possible due to how the algorithm works, with Decision Tree spliting the data to smaller groups, although Logistic Regression still does a good job giving a general analysis of the data.

When comparing both models, the neural network Regression model performed better than the neural classification model, having a lower mse and a higher correlation.

Overall, the Decision Tree performed better than the other algorithms. In terms of accuracy and precision, the Decision Tree algorithm performed better, but falls short when it comes to recall. Logistic Regression has a higher recall value compared to the Decision Tree. As to why the better-performing algorithm could have outperformed the other, it is possible due to how the algorithm works, with Decision Tree spliting the data to smaller groups, although Logistic

```
Sklearn isn't so much different from
using R, although it can be a tad bit
easier running certain algorithms than
in R. On the other hand, R does make
accessing and reading data a bit easier
and modifying datasets a little easier.
Overall, sklearn does provide a better
experience in running machine learning
algorithms.
```

Regression still does a good job giving a general analysis of the data.

Sklearn isn't so much different from using R, although it can be a tad bit easier running certain algorithms than in R. On the other hand, R does make accessing and reading data a bit easier and modifying datasets a little easier. Overall, sklearn does provide a better experience in running machine learning algorithms.