ML with sklearn

Alekhya Pinnamaneni

2022-10-31

1. Read the Auto data

In [2]:
```
# a. use pandas to read the data
import pandas as pd
df = pd.read_csv('Auto.csv')
```

In [3]:
```
# b. output the first few rows
df.head()
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	8	302.0	140	3449	NaN	70.0	1	ford torino

In [4]:
```
# c. output the dimensions of the data
df.shape
```

Out[4]:
```
(392, 9)
```

2. Data exploration with code

In [5]:
```
# a. use describe() on the mpg, weight, and year columns
df.describe()[['mpg', 'weight', 'year']]
# b. write comments indicating the range and average of each column
#    mpg:
#       range = 46.6 - 9 = 37.6
#       average = 23.445918
#    weight:
#       range = 5140 - 1613 = 3,527
#       average = 2,977.584184
#    year:
#       range = 82 - 70 = 12
#       average = 76.010256
```

Out[5]:

	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

3. Explore data types

In [6]:
```
# a. check the data types of all columns
df.dtypes
```

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

In [7]:
```
# b. change the cylinders column to categorical using cat.codes
df.cylinders = df.cylinders.astype('category').cat.codes
```

In [8]:
```
# c. change the origin column to categorical without using cat.codes
df.origin = df.origin.astype('category')
```

In [9]:
```
# d. verify the changes with the dtypes attribute
df.dtypes
```

Out[9]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

4. Deal with NAs

In [10]:
```
# a. delete rows with NAs
df = df.dropna()
```

In [11]:
```
# b. output the new dimensions
df.shape
```

Out[11]:
```
(389, 9)
```

5. Modify Columns

In [12]:
```
# a. make a new column, mpg_high, and make it categorical
df['mpg_high'] = (df['mpg'] > df['mpg'].mean()).astype('category').cat.codes
```

In [13]:
```
# b. delete the mpg and name columns
df = df.drop(columns=['mpg', 'name'])
```

In [14]:
```
# c. output the first few rows of the modified data frame
df.head()
```

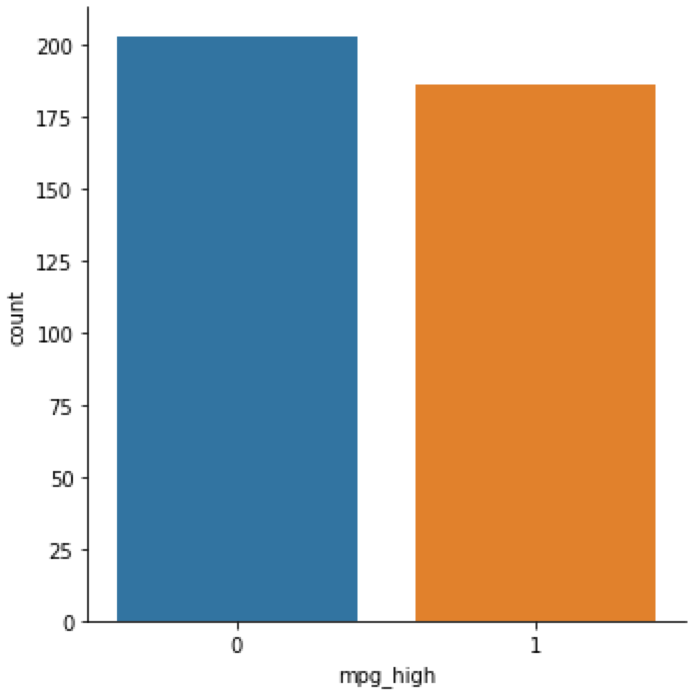
Out[14]:

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

6. Data exploration with graphs

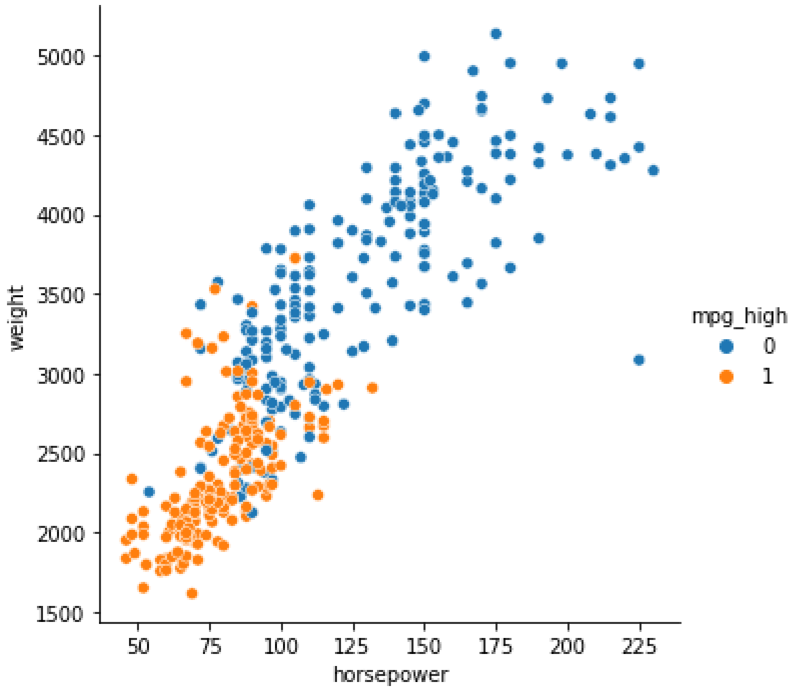
In [15]:
```
# a. seaborn catplot on the mpg_high column
import seaborn as sb
sb.catplot(x='mpg_high', kind='count', data=df)
# From this graph I learned that the majority of the cars in the data set have
a below average mpg.
```

Out[15]:
```
<seaborn.axisgrid.FacetGrid at 0x7f537bab0f90>
```



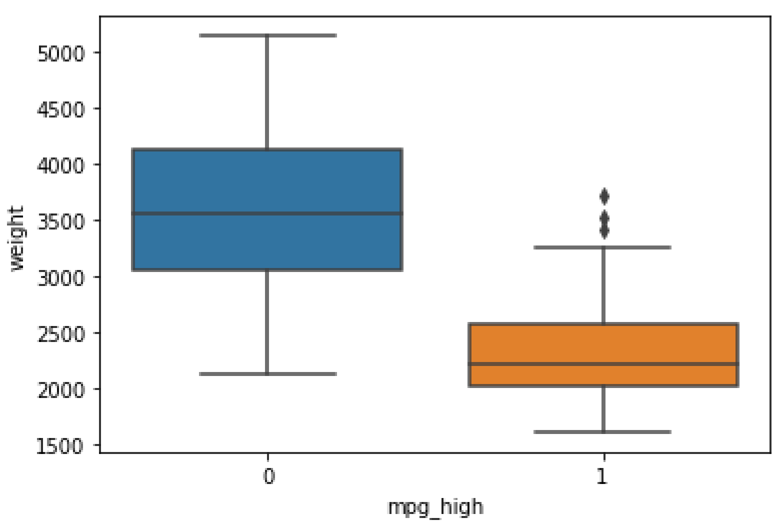
In [16]:
```
# b. seaborn relplot with horsepower on the x axis, weight on the y axis, sett
ing hue to mpg_high
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high)
# From this graph I learned that cars with higher horsepower tend to weigh mor
e and have lower mpg, and vice versa.
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x7f538756b250>



In [17]:
```
# c. seaborn boxplot with mpg_high on the x-axis and weight on the y-axis
sb.boxplot(x='mpg_high', y='weight', data=df)
# From this graph I learned that cars with lower mpg tend to weigh more than c
ars with higher mpg.
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f537841d250>



7. Train/test split

In [18]:
```
from sklearn.model_selection import train_test_split
import random
# set seed to 1234
random.seed(1234)

# create X and y dataframes
X = df.loc[:, df.columns != 'mpg_high']
y = df.mpg_high

# split into train (.80) and test (.20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=0)

# output the dimensions of train and test
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 7)
test size: (78, 7)
```

8. Logistic Regression

In [19]:
```
# a. train a logistic regression model using solve lbfgs
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='lbfgs')
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[19]:
```
0.9035369774919614
```

In [20]:
```
# b. test and evaluate
pred = clf.predict(X_test)  # make predictions using the model

# evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score
print('Accuracy = ', accuracy_score(y_test, pred))
print('Precision = ', precision_score(y_test, pred))
print('Recall = ', recall_score(y_test, pred))
print('F1 = ', f1_score(y_test, pred))
```

```
Accuracy =  0.8589743589743589
Precision =  0.7948717948717948
Recall =  0.9117647058823529
F1 =  0.8493150684931507
```

In [21]:
```
# c. print metrics using the classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.82      0.87        44
           1       0.79      0.91      0.85        34

    accuracy                           0.86        78
   macro avg       0.86      0.86      0.86        78
weighted avg       0.87      0.86      0.86        78
```

9. Decision Tree

In [22]:
```
# a. train a decision tree
from sklearn.tree import DecisionTreeClassifier
clf2 = DecisionTreeClassifier()
clf2.fit(X_train, y_train)
clf2.score(X_train, y_train)
```

Out[22]:
```
1.0
```

In [23]:
```
# b. test and evaluate
pred2 = clf2.predict(X_test)  # make predictions using the model

# evaluate
print('Accuracy = ', accuracy_score(y_test, pred2))
print('Precision = ', precision_score(y_test, pred2))
print('Recall = ', recall_score(y_test, pred2))
print('F1 = ', f1_score(y_test, pred2))
```

```
Accuracy =  0.8974358974358975
Precision =  0.8611111111111112
Recall =  0.9117647058823529
F1 =  0.8857142857142858
```

In [24]:
```
# c. print the classification report metrics
print(classification_report(y_test, pred2))
```

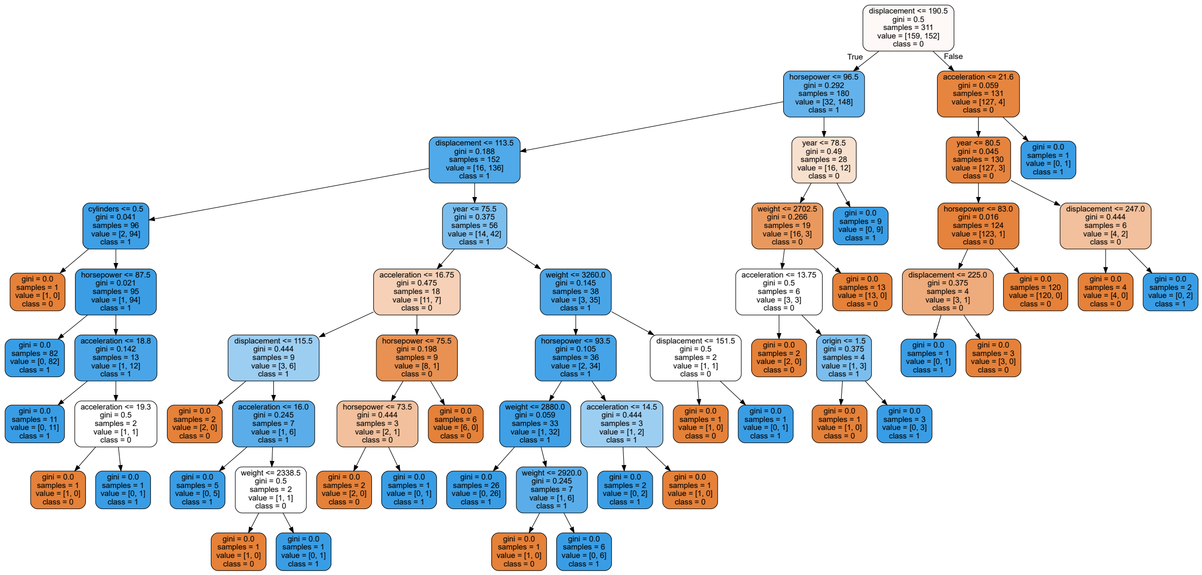
```
              precision    recall  f1-score   support

           0       0.93      0.89      0.91        44
           1       0.86      0.91      0.89        34

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

In [25]:
```
# d. plot the tree
from sklearn import tree
import graphviz
data = tree.export_graphviz(clf2, out_file=None, feature_names=X.columns, clas
s_names=['0', '1'], filled=True, rounded=True)
graphviz.Source(data)
```

Out[25]:



10. Neural Network

In [26]:
```
# normalize the data
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [27]:
```
# a. train a neural network, choosing network topology of your choice
from sklearn.neural_network import MLPClassifier
clf3 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500,
random_state=1234)
clf3.fit(X_train_scaled, y_train)
clf3.score(X_train_scaled, y_train)
```

Out[27]: 0.9421221864951769

In [28]:
```
# b. test and evaluate
pred3 = clf3.predict(X_test_scaled)  # make predictions using the model

# evaluate
print('Accuracy = ', accuracy_score(y_test, pred3))
print('Precision = ', precision_score(y_test, pred3))
print('Recall = ', recall_score(y_test, pred3))
print('F1 = ', f1_score(y_test, pred3))
```

```
Accuracy =  0.8846153846153846
Precision =  0.8787878787878788
Recall =  0.8529411764705882
F1 =  0.8656716417910447
```

In [32]:
```
# print the classification report metrics
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.89      0.91      0.90        44
           1       0.88      0.85      0.87        34

    accuracy                           0.88        78
   macro avg       0.88      0.88      0.88        78
weighted avg       0.88      0.88      0.88        78
```

In [29]:
```
# c. train a second network with a different topology and different settings
clf4 = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500, ran
dom_state=1234)
clf4.fit(X_train_scaled, y_train)
clf4.score(X_train_scaled, y_train)
```

Out[29]: 0.8778135048231511

In [30]:
```
# d. test and evaluate
pred4 = clf4.predict(X_test_scaled)  # make predictions using the model

# evaluate
print('Accuracy = ', accuracy_score(y_test, pred4))
print('Precision = ', precision_score(y_test, pred4))
print('Recall = ', recall_score(y_test, pred4))
print('F1 = ', f1_score(y_test, pred4))
```

```
Accuracy =  0.9102564102564102
Precision =  0.8648648648648649
Recall =  0.9411764705882353
F1 =  0.9014084507042254
```

In [31]:
```
# print the classification report metrics
print(classification_report(y_test, pred4))
```

```
              precision    recall  f1-score   support

           0       0.95      0.89      0.92        44
           1       0.86      0.94      0.90        34

    accuracy                           0.91        78
   macro avg       0.91      0.91      0.91        78
weighted avg       0.91      0.91      0.91        78
```

The second neural network performed better (with 91% accuracy) than the first one (88% accuracy). The main reason I think this happened is because the number of max iterations was higher for the second network than for the first one resulting in higher accuracy. Another explanation could be that the second network has only one hidden layer, while the first one has two. Since the dataset is fairly small, simpler networks like the second one generally work better.

11. Analysis

a. Which algorithm performed better?

The neural network algorithm performed the best out of the three algorithms in this notebook (Logistic Regression, Decision Tree, Neural Network).

b. Comparing accuracy, recall, and precision metrics by class

The accuracy (F1 score) for class 0 is higher than that of class 1. Class 0 had an accuracy (F1 score) of 92% and class 1 had an accuracy of 90%. Class 0 also has a higher precision (95%) than class 1 (86%). However, class 0 has a lower recall (89%) than class 1 (94%).

c. Why did the neural network algorithm outperform the others?

Neural networks have a better ability to predict non-linear and more complex data. Neural networks can also make inferences about unseen data to make predictions, unlike classic ML models that only learn using the given training data set. These attributes on neural networks allowed the model to outperform the logistic regression and decision tree models in this notebook.

d. Comparing my experiences using R versus sklearn

My experiences with coding using R and using sklearn were pretty similar. It is fairly easy and uncomplicated to perform machine learning using both of these methods. However, the runtime of some algorithms were much longer in R than in sklearn. I also feel that the Google Colab environment is much more user-friendly than RStudio. Therefore, I prefer performing machine learning using Python with sklearn over using R.