ML with sklearn

November 8, 2022

[2]:
```
# To view all outputs in jupyter
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import pandas as pd
import numpy as np

df = pd.read_csv('Downloads/Auto.csv')
df.head()
df.size
df.shape
df.ndim
```

[2]:

	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

[2]: 3528

[2]: (392, 9)

[2]: 2

[3]:
```
df['mpg'].describe()

# Range
print(f"Range: {df['mpg'].max() - df['mpg'].min()}")
```

```
# Average
print(f"Average::  {df['mpg'].mean()}")
```

[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

Range: 37.6
Average::  23.44591836734694
```

[4]:
```
df['weight'].describe()

# Range
print(f"Range: {df['weight'].max() - df['weight'].min()}")

# Average
print(f"Average::  {df['weight'].mean()}")
```

[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

Range: 3527
Average::  2977.5841836734694
```

[5]:
```
df['year'].describe()

# Range
print(f"Range: {df['year'].max() - df['year'].min()}")

# Average
print(f"Average::  {df['year'].mean()}")
```

[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

Range: 12.0
Average::  76.01025641025642
```

[6]:
```
for col in df:
    print(f"Column {col} is of type {df[col].dtypes}")
```

```
Column mpg is of type float64
Column cylinders is of type int64
Column displacement is of type float64
Column horsepower is of type int64
Column weight is of type int64
Column acceleration is of type float64
Column year is of type float64
Column origin is of type int64
Column name is of type object
```

[7]:
```
# change cylinders col to categorical with cat.codes
df_catcodes = df.copy()
df_catcodes.cylinders = df_catcodes.cylinders.astype('category').cat.codes
df_catcodes.dtypes
```

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

[8]:
```
# change origin col to categorical without cat.codes
df_cat = df.copy()
df_cat.origin = df_cat.origin.astype('category')
df_cat.dtypes
```

[8]:
```
mpg              float64
cylinders          int64
```

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

[9]:
```
# Delete rows with NAs in original df
df.isnull().sum()
df = df.dropna()
df.shape
df.isnull().sum()
```

[9]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

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

[9]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

[10]:
```
# New col mpg_high as cat
df['mpg_high'] = np.where((df['mpg'] > df['mpg'].mean()), 1, 0)
df['mpg_high'] = df['mpg_high'].astype('category').cat.codes
df.dtypes

# Drop 'mpg' and 'names' cols
df = df.drop(columns=['mpg', 'name'])
print(df.head())
```

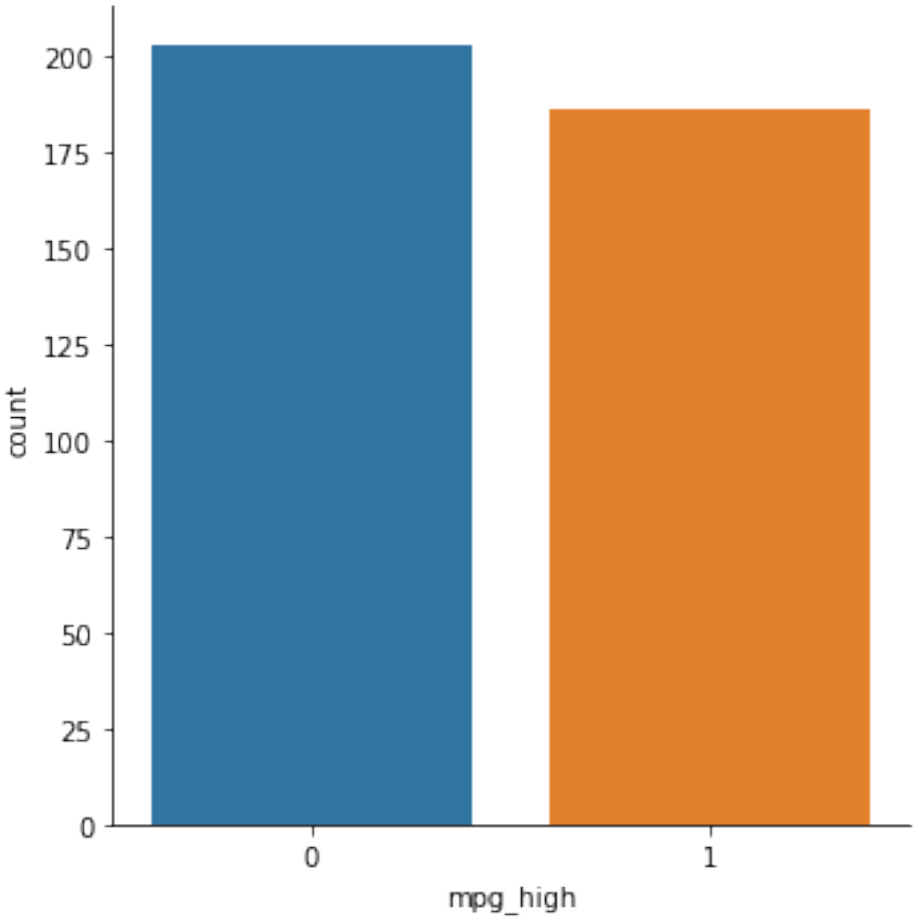
[10]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
mpg_high           int8
dtype: object
```

	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	8	307.0	130	3504	12.0	70.0	1
1	8	350.0	165	3693	11.5	70.0	1
2	8	318.0	150	3436	11.0	70.0	1
3	8	304.0	150	3433	12.0	70.0	1
6	8	454.0	220	4354	9.0	70.0	1

	mpg_high
0	0
1	0
2	0
3	0
6	0

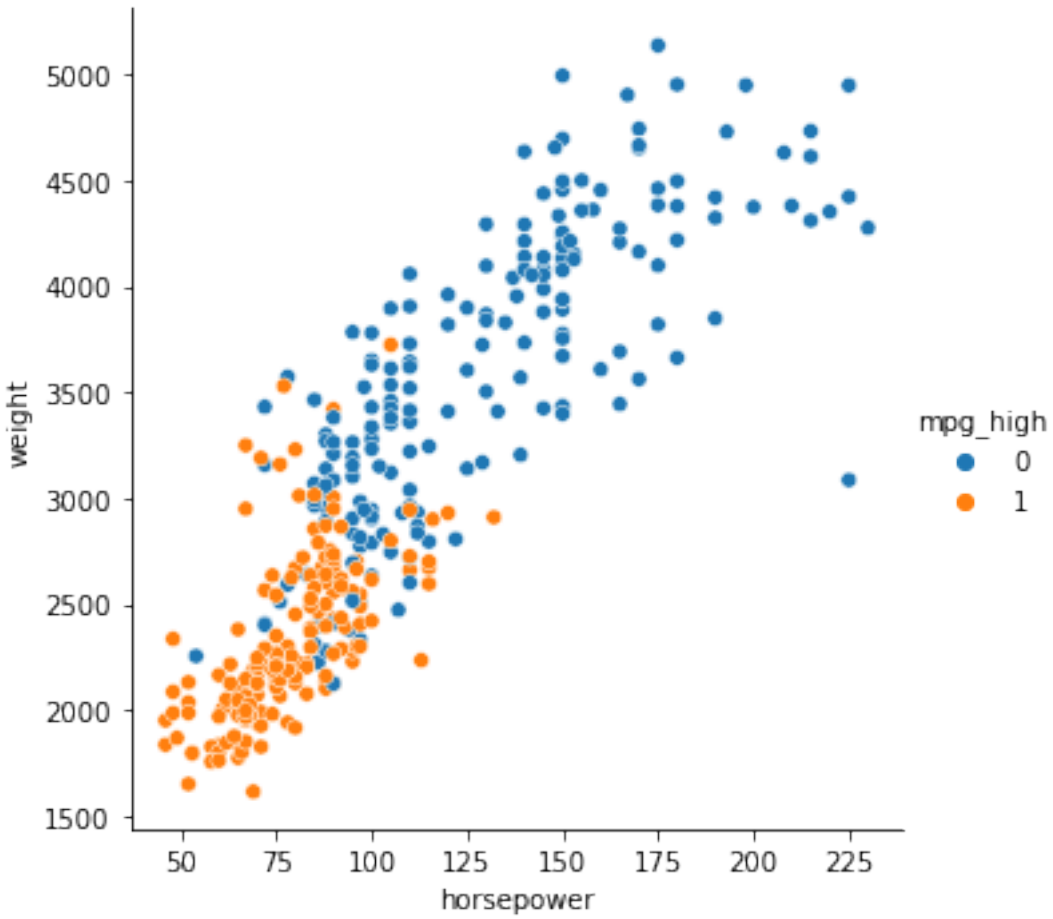
[11]:
```
# Data exploration with graphs
import seaborn as sb

sb.catplot(data=df, x='mpg_high', kind='count')
# One think I learned from this graph is that there are more data
# points that are below the average mpg
```

[11]:
```
<seaborn.axisgrid.FacetGrid at 0x7f77be602040>
```



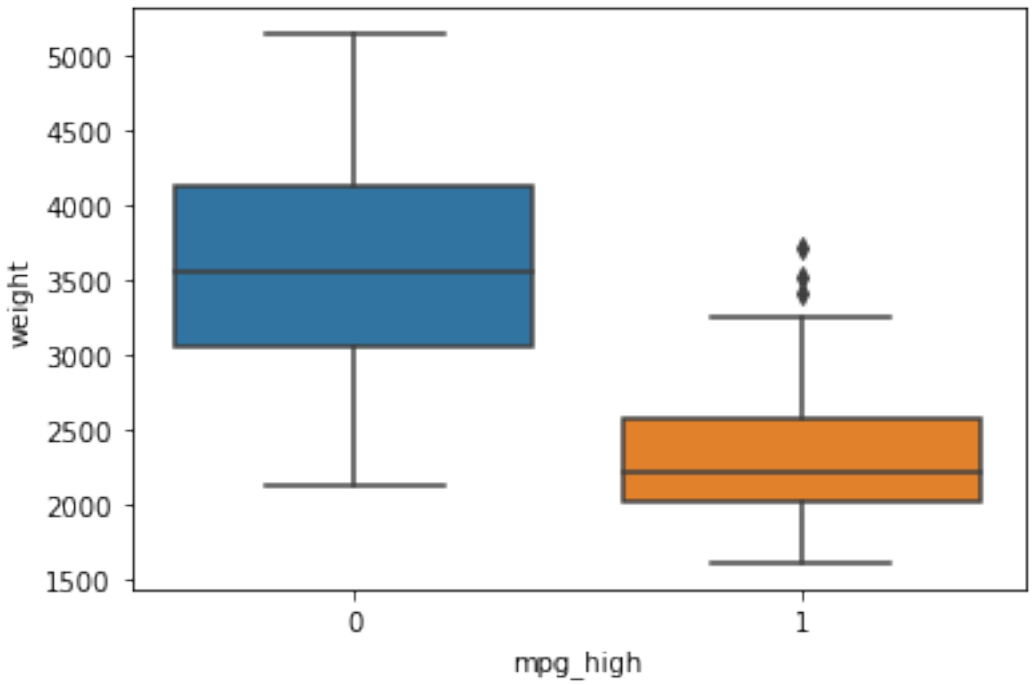
[12]:
```
sb.relplot(data=df, x='horsepower', y='weight', hue='mpg_high')
# One thing I learned from this graph is that typically the lighter
# and less horsepower a car has, it will have a high mpg.
```

[12]: <seaborn.axisgrid.FacetGrid at 0x7f77ba9d6c40>



[13]:
```
sb.boxplot(data=df, x='mpg_high', y='weight')
# One thing I learned from this graph is that although cars that are
# lighter tend to have a higher mpg, there is less data within the
# box portion, or the 75%, and so the data may not fully represent
# this notion.
```

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



[14]:
```
from sklearn.model_selection import train_test_split

# Split train/test
X = df.loc[:, ['cylinders', 'displacement', 'horsepower', 'weight',␣
 ↪'acceleration', 'year', 'origin']]
y = df['mpg_high']

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

print('Train dimensions: ', X_train.shape)
print('Test dimensions: ', X_test.shape)
```

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

[15]:
```
# Logistic regression
from sklearn.linear_model import LogisticRegression

# With the default value of 'max_iter=100', lbfgs failed to converge error
# so increase 'max_iter' by large enough value
clf = LogisticRegression(max_iter=500)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

[15]:
```
LogisticRegression(max_iter=500)
```

[15]:
```
0.9035369774919614
```

[16]:
```
# test and evalulate logistic regression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

y_pred = clf.predict(X_test)
print('accuracy socre: ', accuracy_score(y_test, y_pred))
print('precision socre: ', precision_score(y_test, y_pred))
print('recall socre: ', recall_score(y_test, y_pred))
print('f1 socre: ', f1_score(y_test, y_pred))
```

```
accuracy socre:  0.8974358974358975
precision socre:  0.7777777777777778
recall socre:  1.0
f1 socre:  0.8750000000000001
```

[17]:
```
# try a second classifier to see if scores can be improved
clf2 = LogisticRegression(class_weight='balanced')
clf2.fit(X_train, y_train)
y_pred2 = clf2.predict(X_test)
print('accuracy socre: ', accuracy_score(y_test, y_pred2))
print('precision socre: ', precision_score(y_test, y_pred2))
print('recall socre: ', recall_score(y_test, y_pred2))
print('f1 socre: ', f1_score(y_test, y_pred2))
```

[17]:
```
LogisticRegression(class_weight='balanced')
```

```
accuracy socre:  0.8589743589743589
precision socre:  0.7297297297297297
recall socre:  0.9642857142857143
f1 socre:  0.8307692307692307
```

[18]:
```
# classification report for logistic regression
from sklearn.metrics import classification_report
print('Classification report:\n', classification_report(y_test, y_pred))
print('\nClassification report with 2nd classifier: \n',␣
 ↪classification_report(y_test, y_pred2))
```

```
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


Classification report with 2nd classifier:
              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
```

[19]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# Test and eval
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('accuracy socre: ', accuracy_score(y_test, y_pred))
print('precision socre: ', precision_score(y_test, y_pred))
print('recall socre: ', recall_score(y_test, y_pred))
print('f1 socre: ', f1_score(y_test, y_pred))

# classification report
print(classification_report(y_test, y_pred))
```

[19]:
```
DecisionTreeClassifier()

accuracy socre:  0.8846153846153846
precision socre:  0.8518518518518519
recall socre:  0.8214285714285714
f1 socre:  0.8363636363636364
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        50
           1       0.85      0.82      0.84        28

    accuracy                           0.88        78
   macro avg       0.88      0.87      0.87        78
weighted avg       0.88      0.88      0.88        78
```

[20]:
```
# Neural Network
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier

# scale data for possible better results and train
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500,␣
 ↪random_state=1234)
clf.fit(X_train_scaled, y_train)

# predictions
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))

print(classification_report(y_test, pred))
```

[20]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')

accuracy =  0.8717948717948718
              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
```

[21]:
```
# Neural Network 2
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier

# scale data for possible better results and train
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(7, 7), max_iter=1000,␣
 ↪random_state=1234)
clf.fit(X_train_scaled, y_train)

# predictions
```

```
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))

print(classification_report(y_test, pred))
```

[21]:
```
MLPClassifier(hidden_layer_sizes=(7, 7), max_iter=1000, random_state=1234,
              solver='lbfgs')

accuracy =  0.9102564102564102
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        50
           1       0.86      0.89      0.88        28

    accuracy                           0.91        78
   macro avg       0.90      0.91      0.90        78
weighted avg       0.91      0.91      0.91        78
```

[22]:
```
# The accuracy of the second neural network is higher than the first
# neural network. This could be because the hidden layers in the
# second neural network is more complex. However, having a more complex
# topology for such a small dataset leads to overfitting.
```