ML with sklearn

by: Dinesh Angadipeta (DXA190032)

Reading in the Auto Data:

```
### load the data
import pandas as pd
df = pd.read_csv('Auto.csv')
print(df.head())
print('\nDimensions of data frame:', df.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 of data frame: (392, 9)
```

Data Exploration with Code:

Using describe() on mpg

```
print(df["mpg"].describe())
```

```
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
```

Here we can see that the average mpg for all of the cars in the dataset is about 23.4 miles per gallon. The lowest mpg in the dataset is 9 miles per gallon and the highest mpg is 46.6 miles per gallon. The min ans max values are pretty far off the mean.

Using describe() on weight

```
print(df["weight"].describe())
```

```
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
```

Here we can see that the average weight of the cars in the dataset is about 2977.6 pounds. The lightest car in the dataset weighs 1613 pounds and the heaviest car in the dataset weighs 5140 pounds.

Using describe() on year

```
print(df["year"].describe())
```

```
print(df['year'].describe())
```

```
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
```

Here we can see that the average year that the cars were manufactured in is around 1976. The oldest car was manufactured in 1970 and the newest car in the dataset was manufactured in 1982.

Explore Data Types

Data Types of all columns:

```
df.dtypes
```

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

Changing the cylinders column to catagorical using cat.codes:

```
df.cylinders = df.cylinders.astype('category').cat.codes
```

Changing the origin column to catagorical without using cat.codes:

```
df.origin = df.origin.astype('category')
```

Verifying the changes:

```
df.dtypes
```

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

Dealing with NAs:

```
df = df.dropna()
print('\nDimensions of data frame:', df.shape)
```

```

Dimensions of data frame: (389, 9)
```

Modifying Columns

Here we are making a new column named "mpg_high" and making it categorical. The conditions of this column is that the column value will be 1 if mpg > average mpg, or else it will be 0. We will also be dropping the mpg and name columns.

```
import numpy as np
df['mpg_high'] = np.where(df.mpg > np.mean(df.mpg), 1, 0)
df.mpg_high = df.mpg_high.astype('category')
df = df.drop(columns=['mpg', 'name'])
print(df.head())
```

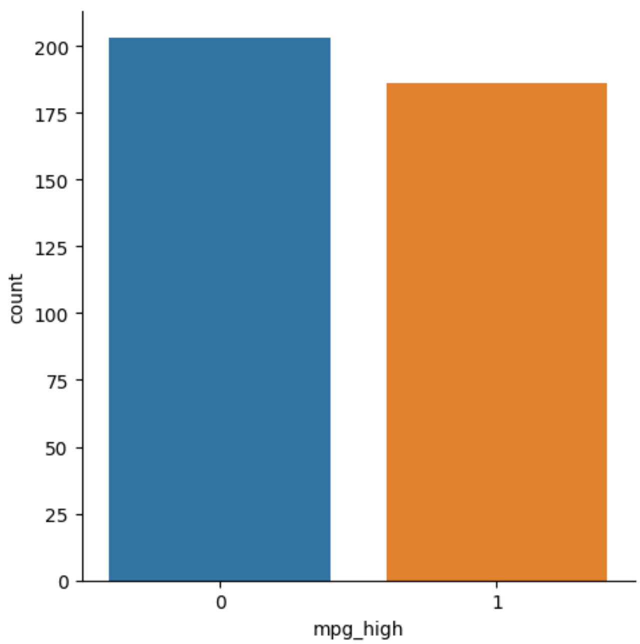
```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      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
<ipython-input-72-c6abe1a0dbdf>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  df['mpg_high'] = np.where(df.mpg > np.mean(df.mpg), 1, 0)
<ipython-input-72-c6abe1a0dbdf>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  df.mpg_high = df.mpg_high.astype('category')
```

Data Exploration with graphs

seaborn catplot on the mpg_high column

```
import pandas as pd
import seaborn as sb
from sklearn import datasets
sb.catplot(x="mpg_high", kind='count', data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f59695cfd90>
```

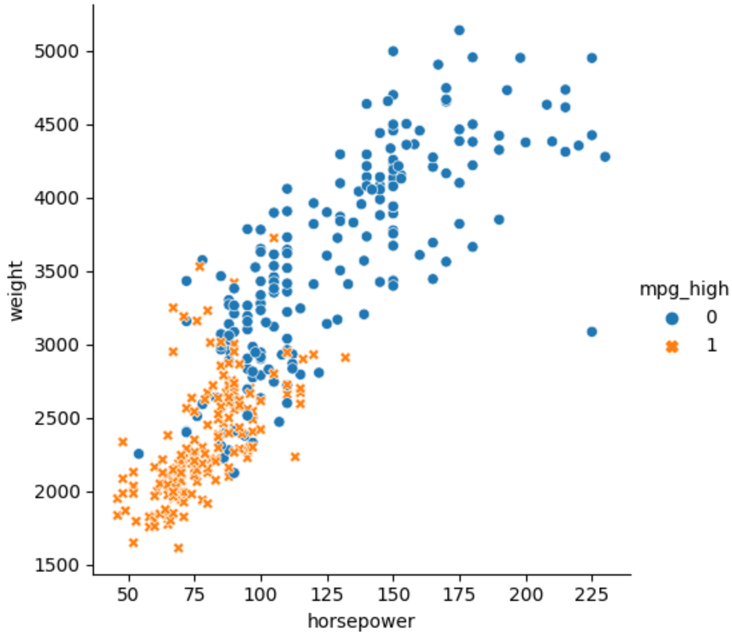


From this graph we can see that there is about a 50/50 split between vehicles with lower mpg and vehicles with higher mpg, with there being more vehicles with a lower mpg.

seaborn relplot with horsepower on the x axis and weight on the y axis

```
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high, style=df.mpg_high)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f59694d7ac0>
```

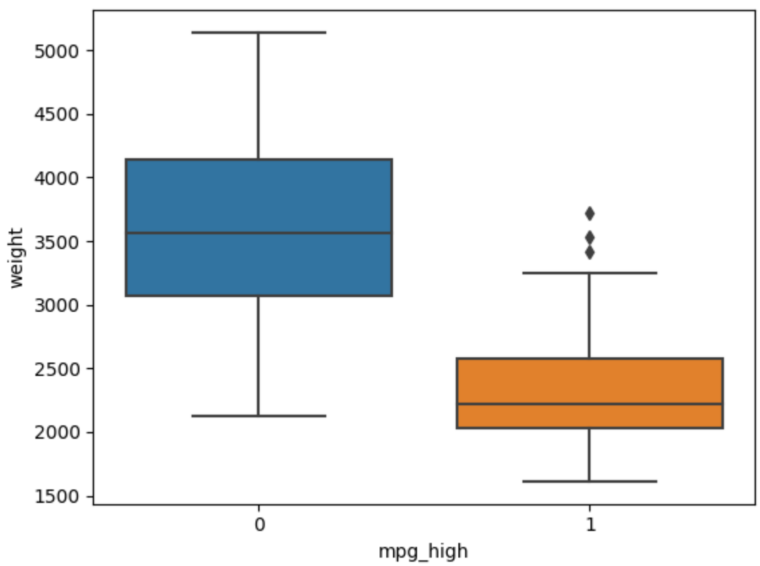


From this graph we can see that the higher the weight and power of a vehicle, the lower the mpg of the vehicle. An almost linear relationship can also be seen from this data.

seaborn boxplot with mpg_high on the x axis and weight on the y axis

```
sb.boxplot(x='mpg_high', y='weight', data=df)
```

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



Here he can deduce that the higher mpg vehicle's weighed less than the lower mpg vehicles.

Train/test split

```
from sklearn.model_selection import train_test_split

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

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

Logistic Regression

Training/Testing:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.9067524115755627
```

```
pred = clf.predict(X_test)
```

Evaluating:

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

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

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

Confusion Matrix:

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred)
```

```
array([[40, 10],
       [ 1, 27]])
```

Classification Report:

```
clf2 = LogisticRegression(class_weight='balanced')
clf2.fit(X_train, y_train)
pred2 = clf2.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred2))
print('precision score: ', precision_score(y_test, pred2))
```

```
print('recall score: ', recall_score(y_test, pred2))
print('f1 score: ', f1_score(y_test, pred2))
```

```
accuracy score:  0.8717948717948718
precision score:  0.75
recall score:  0.9642857142857143
f1 score:  0.8437499999999999
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Confusion Matrix:

```
confusion_matrix(y_test, pred2)
```

```
array([[41,  9],
       [ 1, 27]])
```

Decision Tree

Training/Testing:

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
pred = clf.predict(X_test)
```

Evaluating:

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

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

```
accuracy score:  0.8717948717948718
precision score:  0.8
recall score:  0.8571428571428571
f1 score:  0.8275862068965518
```

Confusion Matrix:

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred)
```

```
array([[44,  6],
       [ 4, 24]])
```

Classification Report:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.92      0.88      0.90        50
              1       0.80      0.86      0.83        28

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

Random Forest Classification Report:

```
from sklearn.ensemble import RandomForestClassifier

clf2 = RandomForestClassifier(max_depth=4, random_state=1234)
clf2.fit(X_train, y_train)

pred2 = clf2.predict(X_test)

print(classification_report(y_test, pred2))
```

```
                 precision    recall  f1-score   support

              0       0.95      0.84      0.89        50
              1       0.76      0.93      0.84        28

       accuracy                           0.87        78
      macro avg       0.86      0.88      0.87        78
   weighted avg       0.89      0.87      0.87        78
```

Neural Network

Normalizing 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)
```

Training/Testing:

```
from sklearn.neural_network import MLPClassifier

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

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptr
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
                              MLPClassifier
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
pred = clf.predict(X_test_scaled)
```

Evaluating:

```
print('accuracy = ', accuracy_score(y_test, pred))
```

```
accuracy =  0.8589743589743589
```

Confusion Matrix:

```
confusion_matrix(y_test, pred)
```

```
array([[42,  8],
       [ 3, 25]])
```

Classification Report:

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

```
              precision    recall  f1-score   support

           0       0.93      0.84      0.88        50
           1       0.76      0.89      0.82        28

    accuracy                           0.86        78
   macro avg       0.85      0.87      0.85        78
weighted avg       0.87      0.86      0.86        78
```

Second Network with different settings:

```
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500, random_state=1234)
clf.fit(X_train_scaled, y_train)
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(3,), max_iter=1500, random_state=1234,
              solver='sgd')
```

```
pred = clf.predict(X_test_scaled)
```

Evaluating:

```
print('accuracy = ', accuracy_score(y_test, pred))
```

```
accuracy =  0.8333333333333334
```

Confusion Matrix:

```
confusion_matrix(y_test, pred)
```

```
array([[40, 10],
       [ 3, 25]])
```

Classification Report:

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

```
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        50
           1       0.71      0.89      0.79        28

    accuracy                           0.83        78
   macro avg       0.82      0.85      0.83        78
weighted avg       0.85      0.83      0.84        78
```

Analysis:

- Which algorithm performed better:

Out of all the algorithms used, the one that performed the best and yielded the highest accuracy was the decision tree method.

- Comparing accuracy, recall, and precision:

Logistic Regression yielded: accuracy: 0.859 precision: 0.730 recall: 0.831

Decision Tree yielded: accuracy: 0.910 precision: 0.839 recall: 0.929

Neural Network: accuracy: 0.833 precision: 0.850 recall: 0.830

Here we can see that the decision tree yielded the highest result. Neural Network yielded around the lowest results.

- Analysis on why DT performed best:

My take on why the decision tree performed the best is that it is a flexible algorithm that was more suited for this dataset due to the way the data was set up and the way the data was clustered.

- My experience on R vs sklearn:

While both languages are similar in their notebook formats, I did find sklearn easier to use due to its easy conversion to pdf and connestion to stackoverflow. It provides many helpful resources and is easier to add new libraries. The only downside for me is that the data importing is pretty difficult to understand at the beginning.

✓ 0s completed at 1:44 AM