ML with sklearn

CS4375 Linus Fackler

This is our first assignment using Python and the sklearn library.

Reading the data

In [ ]:
```
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

In [ ]:
```
print('\nDescribe  mpg, weight, and year:\n', df.loc[:, ['mpg', 'weight', 'year']].des

print('\nRange of mpg:\t\t', df['mpg'].max() - df['mpg'].min())
print('Range of weight:\t', df['weight'].max() - df['weight'].min())
print('Range of year:\t\t', df['year'].max() - df['year'].min())

print('\nMean of mpg:\t', df['mpg'].mean())
print('Mean of weight:\t', df['weight'].mean())
print('Mean of year:\t', df['year'].mean())
```

```
Describe  mpg, weight, and year:
                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 of mpg:            37.6
Range of weight:         3527
Range of year:           12.0

Mean of mpg:     23.445918367346938
Mean of weight:  2977.5841836734694
Mean of year:    76.01025641025642
```

The mpg has a decent range and average that gives us enough information about where most of the data lies. Similar with the weight. The year, has a pretty low range compared to the rest.

Explore data types

In [ ]:

```
df.dtypes
```

Out[ ]:

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

The data is mostly represented in integers and floats.

Changing Cylinder column to categorical:

In [ ]:

```
df.cylinders = df.cylinders.astype('category').cat.codes
df = df.astype({"origin":'category'})
df.dtypes
```

Out[ ]:

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

The cylinders column, using "cat.codes", will be represented as int8 data type, while the name column will be represented as "category" type.

Deal with NAs

In [ ]:
```
df.isna().sum()
```

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

We see that there are only in total 3 entries with NAs, so we will drop all of them.

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

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

Modify columns

In [ ]:
```
import numpy as np
avg = df.mpg.mean()
df['mpg_high'] = np.where(df.mpg > avg, 1, 0)
```

In [ ]:
```
df = df.drop(columns=['name','mpg'])
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
```

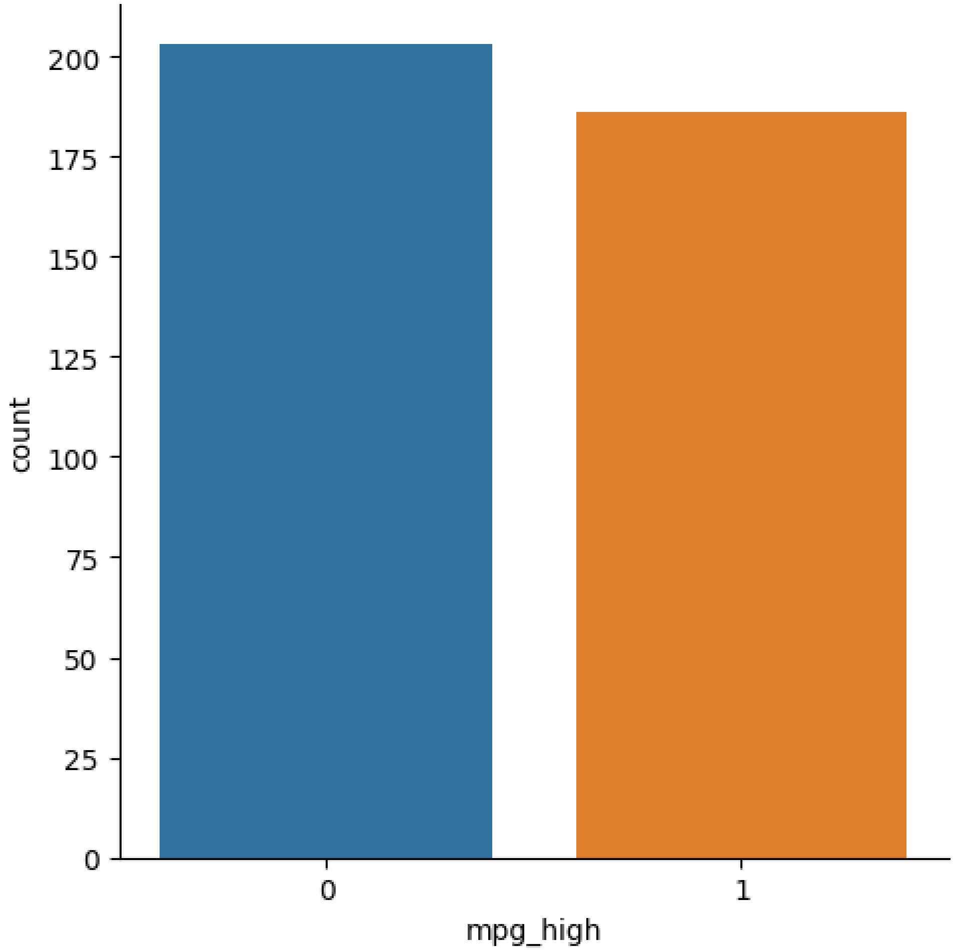
Data exlporation with graphs

Catplot on new mph_high column

In [ ]:
```
import seaborn as sb

sb.catplot(x = 'mpg_high', kind = 'count', data = df)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x1fc1e1676d0>

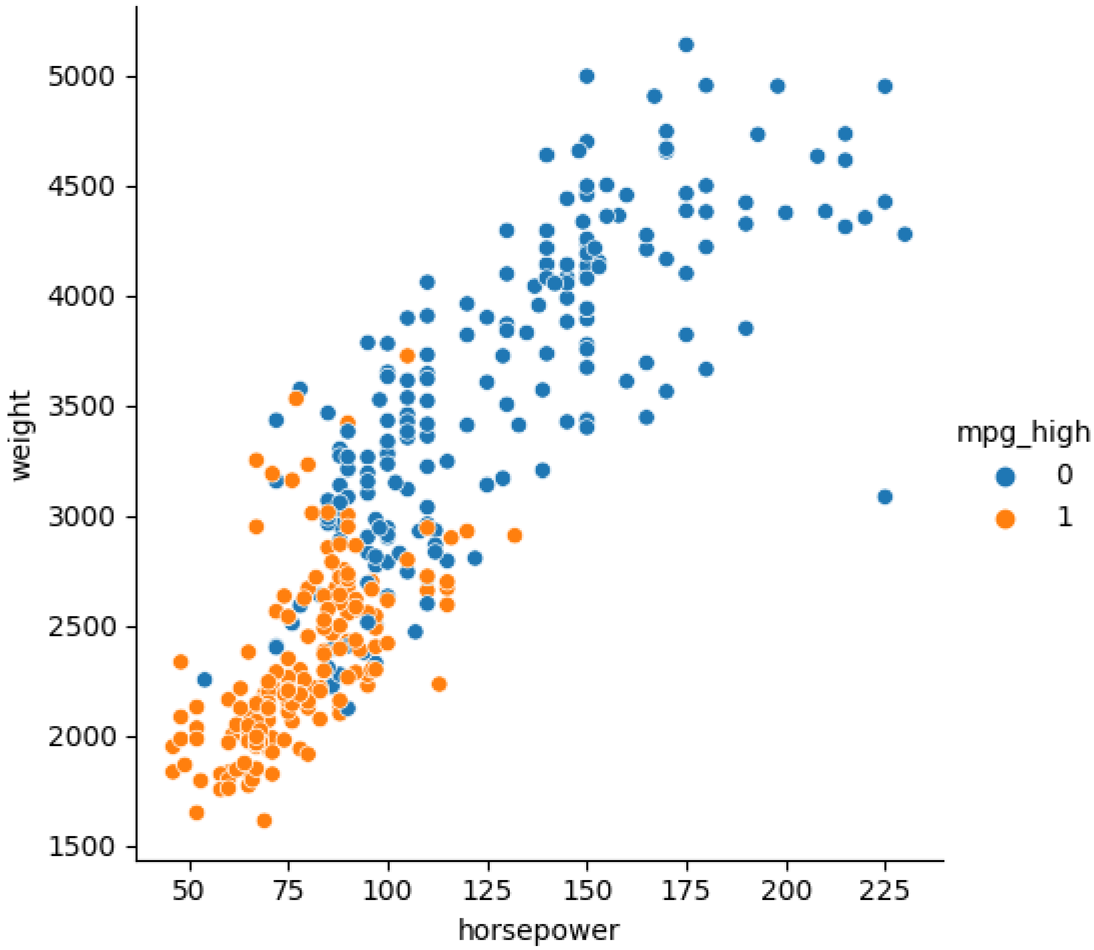


There is almost an equal amount of cars with a high and low amount of miles per gallon. This tells us that there is an equal distribution, meaning, this can't be the reason for a drift in a certain direction.

Relplot - horsepower vs weight

In [ ]:
```
sb.relplot(x = 'horsepower', y = 'weight', data = df, hue = df.mpg_high)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x1fc1e166620>

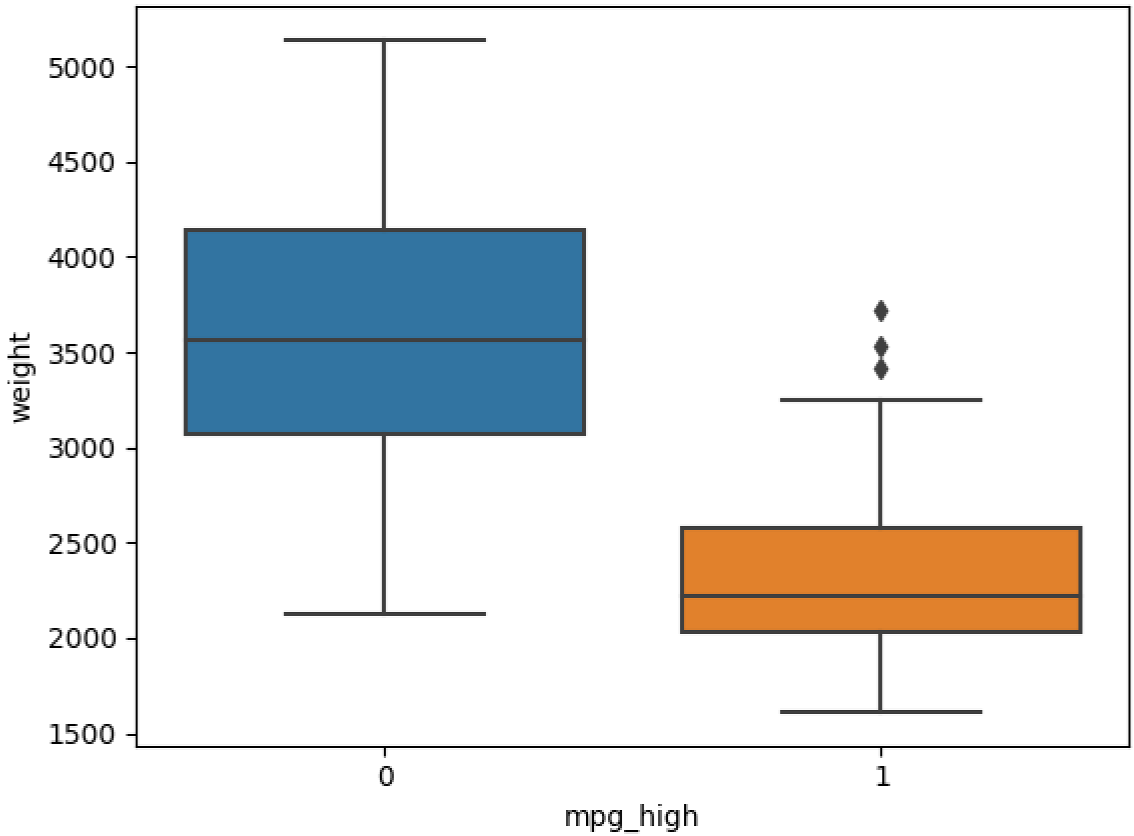


This graph clearly shows us that there is almost a linear correlation of the weight and horsepower and their effect on the mpg. Cars with a lower number of horepowers and lower weight tend to have a better mpg. The cars with the lower mpg are most likely big cars, like trucks or SUV's.

Boxplot - mpg_high vs. weight

In [ ]:
```
sb.boxplot(x = 'mpg_high', y = 'weight', data = df)
```

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



This graph, again clearly, shows us that most cars with a lower mpg tend to be in the havier range of cars. Cars with an above average mpg are weighing less.

Train/test split

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

X = df.iloc[:, 0:6]
y = df.iloc[:, 7]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat

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

```
train size: (311, 6)
test size: (78, 6)
```

Logistic Regression

Train logistic regression model using solver lbfgs

In [ ]:
```
from sklearn.linear_model import LogisticRegression

logReg = LogisticRegression(solver = 'lbfgs')
```

```
logReg.fit(X_train, y_train)
logReg.score(X_train, y_train)
```

Out[ ]: 0.9035369774919614

Predict

In [ ]:
```
predLR = logReg.predict(X_test)
```

Evaluate

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracyLR = accuracy_score(y_test, predLR)
precisionLR = precision_score(y_test, predLR)
recallLR = recall_score(y_test, predLR)
f1LR = f1_score(y_test, predLR)

print('accuracy score: ', accuracyLR)
print('precision score: ', precisionLR)
print('recall score: ', recallLR)
print('f1 score: ', f1LR)
```

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

Decision tree

Train a decision tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier

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

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Make predictions

In [ ]:
```
predDT = dTree.predict(X_test)
```

Evaluate

In [ ]:
```
accuracyDT = accuracy_score(y_test, predDT)
precisionDT = precision_score(y_test, predDT)
recallDT = recall_score(y_test, predDT)
f1DT = f1_score(y_test, predDT)
```

```
print('accuracy score: ', accuracyDT)
print('precision score: ', precisionDT)
print('recall score: ', recallDT)
print('f1 score: ', f1DT)
```

```
accuracy score:  0.9102564102564102
precision score:  0.8181818181818182
recall score:  0.9642857142857143
f1 score:  0.8852459016393442
```

Neural Network

Normalize the data

In [ ]:
```
from sklearn import preprocessing

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

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

Using hidden layer size of 5,2

Train neural network

In [ ]:
```
from sklearn.neural_network import MLPClassifier

nn1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_st
nn1.fit(X_train_scaled, y_train)
```

Out[ ]:
```
MLPClassifier

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

Make Predictions

In [ ]:
```
predNN1 = nn1.predict(X_test_scaled)
```

Output Result

In [ ]:
```
from sklearn.metrics import confusion_matrix

accuracyNN1 = accuracy_score(y_test, predNN1)
precisionNN1 = precision_score(y_test, predNN1)
recallNN1 = recall_score(y_test, predNN1)
f1NN1 = f1_score(y_test, predNN1)

print('accuracy score: ', accuracyNN1)
print('precision score: ', precisionNN1)
```

```
print('recall score: ', recallNN1)
print('f1 score: ', f1NN1)

confusion_matrix(y_test, predNN1)
```

```
accuracy score:  0.8461538461538461
precision score:  0.75
recall score:  0.8571428571428571
f1 score:  0.7999999999999999
```

Out[ ]:
```
array([[42,  8],
       [ 4, 24]], dtype=int64)
```

Using hidden layer size of 4,2

Train neural network

In [ ]:
```
nn2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(4, 2), max_iter=500, random_st
nn2.fit(X_train_scaled, y_train)
```

Out[ ]:
```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(4, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

Make predictions

In [ ]:
```
predNN2 = nn2.predict(X_test_scaled)
```

Output results

In [ ]:
```
accuracyNN2 = accuracy_score(y_test, predNN2)
precisionNN2 = precision_score(y_test, predNN2)
recallNN2 = recall_score(y_test, predNN2)
f1NN2 = f1_score(y_test, predNN2)

print('accuracy score: ', accuracyNN2)
print('precision score: ', precisionNN2)
print('recall score: ', recallNN2)
print('f1 score: ', f1NN2)

confusion_matrix(y_test, predNN2)
```

```
accuracy score:  0.9230769230769231
precision score:  0.8666666666666667
recall score:  0.9285714285714286
f1 score:  0.896551724137931
```

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

Comparing both models

Making the number of hidden layers smaller is better in this case, as we see in the results. This is mainly due to the size of our dataset. Neural Networks are intended for more complex dataset. Looking at the graphs above, we can almost make a prediction ourselves and guess most likely correct. A higher number of hidden layers (as in the first NN model) means it can learn more complex relationships. This is not suitable for a dataset with ~300 rows, such as this one.

Analysis

Which algorithm performed better?

Accuracy-wise, the decision tree and the second Neural Network model both had the same, and highest, accuracy.

Comparing accuracy, recall and precision metric

In [ ]:
```
print('accuracy score for Logistic Regression:\t\t', accuracyLR)
print('accuracy score for Decision Tree:\t\t', accuracyDT)
print('accuracy score for Neural Networks model 1:\t', accuracyNN1)
print('accuracy score for Neural Networks model 2:\t', accuracyNN2)
print('\n')
print('precision score for Logistic Regression:\t', precisionLR)
print('precision score for Decision Tree:\t\t', precisionDT)
print('precision score for Neural Networks model 1:\t', precisionNN1)
print('precision score for Neural Networks model 2:\t', precisionNN2)
print('\n')
print('recall score for Logistic Regression:\t\t', recallLR)
print('recall score for Decision Tree:\t\t\t', recallDT)
print('recall score for Neural Networks model 1:\t', recallNN1)
print('recall score for Neural Networks model 2:\t', recallNN2)
```

```
accuracy score for Logistic Regression:         0.8589743589743589
accuracy score for Decision Tree:               0.9102564102564102
accuracy score for Neural Networks model 1:     0.8461538461538461
accuracy score for Neural Networks model 2:     0.9230769230769231


precision score for Logistic Regression:        0.7297297297297297
precision score for Decision Tree:              0.8181818181818182
precision score for Neural Networks model 1:    0.75
precision score for Neural Networks model 2:    0.8666666666666667


recall score for Logistic Regression:           0.9642857142857143
recall score for Decision Tree:                 0.9642857142857143
recall score for Neural Networks model 1:       0.8571428571428571
recall score for Neural Networks model 2:       0.9285714285714286
```

The decision tree has the highest accuracy, precision and recall score. In the next paragraph, I will be explaining why.

Why some models were better

Logistic Regression vs Decision Tree: Looking at the data and the relplot above, the decision tree was clearly more suitable for this data. The data was not linearly separable. Logistic regression separates the space into 2 regions, which is not preferred for this dataset. Creating a line to split up the data leaves too much unwanted data on each side. The decision tree outperforms linear regression in accuracy and precision, and ties in recall.

As for the neural networks: As mentioned above, neural networks are for more complex datasets, rather than small ones, such as this, with clear relationships and correlations. To get better results with a NN, we would need more parameters per training observations. In the model above, though, we see a strong increase of all 3 score parameters just by decreasing the number of hidden layers.

The decision tree outperforms even the neural network. A decision tree is essentially a simplified version of a neural network. Since this data is obvious and easily predictible to a certain extend, decision trees work better.

R vs sklearn

I personally like sklearn way more, mainly because I prefer Python over R. I like Python's syntax more and of course have more experience in Python, since this was my first time using R. There are many other reasons. First of all, I am just personally not a big fan of RStudio. I like being able to do all my programming in VS Code, since I have a bunch of extensions and are most comfortable with it. But that could of course just be fixed if I could use R in VSCode. Another reason is the amount of resources on the internet specifically. Python is generally way more often used in the world, meaning, there is more help on the internet if you're stuck. Other than that, using R was completely fine. It's not like we're forced to write everything in Prolog or Racket, which we are in other courses.