Alper Duru

Date: 04/08/2023

Machine Learning with Sklearn

```
# some necessary packages
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder


import pickle
import numpy as np
import pandas as pd
import io


# set seed for reproducibility
np.random.seed(1234)


# set up import from local file
from google.colab import files
uploaded = files.upload()
```

Choose Files Auto.csv

- **Auto.csv**(text/csv) - 17859 bytes, last modified: 4/8/2023 - 100% done

```
Saving Auto.csv to Auto.csv
```

```
df = pd.read_csv(io.BytesIO(uploaded['Auto.csv']))
shape_beforeRemovingNAs = df.shape
print('rows and columns:', shape_beforeRemovingNAs)
df.head()
```

```
rows and columns: (392, 9)
```

	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

```
df['mpg'].describe()
df['weight'].describe()
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
```

```
dataRange = df.iloc[:, 0:8]
print(dataRange.max() - dataRange.min())
```

```
mpg              37.6
cylinders         5.0
displacement    387.0
horsepower      184.0
weight         3527.0
acceleration     16.8
year             12.0
origin            2.0
dtype: float64
```

```
# Calculate the mean of each column
column_means = df.mean(axis=0)

# Print the column means
print("Column means:\n", column_means)
```

```
Column means:
 mpg                23.445918
cylinders           5.471939
displacement      194.411990
horsepower        104.469388
weight           2977.584184
acceleration       15.554220
year               76.010256
origin              1.576531
dtype: float64
<ipython-input-20-30e7a4c5df18>:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=No
  column_means = df.mean(axis=0)
```

```
df.dtypes
```

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

```
# Convert 'cylinders' column to categorical variable
df['cylinders'] = df['cylinders'].astype('category').cat.codes
print(df)
```

```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          4         307.0         130    3504          12.0  70.0
1    15.0          4         350.0         165    3693          11.5  70.0
2    18.0          4         318.0         150    3436          11.0  70.0
3    16.0          4         304.0         150    3433          12.0  70.0
4    17.0          4         302.0         140    3449           NaN  70.0
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          1         140.0          86    2790          15.6  82.0
388  44.0          1          97.0          52    2130          24.6  82.0
389  32.0          1         135.0          84    2295          11.6  82.0
390  28.0          1         120.0          79    2625          18.6  82.0
391  31.0          1         119.0          82    2720          19.4  82.0

    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
..     ...                        ...
387      1            ford mustang gl
388      2                  vw pickup
389      1              dodge rampage
390      1                ford ranger
391      1                 chevy s-10

[392 rows x 9 columns]
```

```
# Convert 'origin' column to categorical variable
df['origin'] = df['origin'].astype('category')
print(df)
```

```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          4         307.0         130    3504          12.0  70.0
1    15.0          4         350.0         165    3693          11.5  70.0
2    18.0          4         318.0         150    3436          11.0  70.0
3    16.0          4         304.0         150    3433          12.0  70.0
4    17.0          4         302.0         140    3449           NaN  70.0
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          1         140.0          86    2790          15.6  82.0
388  44.0          1          97.0          52    2130          24.6  82.0
389  32.0          1         135.0          84    2295          11.6  82.0
390  28.0          1         120.0          79    2625          18.6  82.0
391  31.0          1         119.0          82    2720          19.4  82.0
```

```
    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
..     ...                        ...
387      1            ford mustang gl
388      2                  vw pickup
389      1              dodge rampage
390      1                ford ranger
391      1                 chevy s-10

[392 rows x 9 columns]
```

```
df.dtypes
```

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

```
df = df.dropna()
shape_afterRemovingNAs = df.shape
```

```
print('rows and columns before removing NAs:', shape_beforeRemovingNAs)
print('rows and columns after removing NAs:', shape_afterRemovingNAs)
```

```
rows and columns before removing NAs: (392, 9)
rows and columns after removing NAs: (389, 9)
```

```
mpg_average = df['mpg'].mean()
df['mpg_high'] = 0
df.loc[df['mpg'] > mpg_average, 'mpg_high'] = 1
print(df)
```

```
      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
6    14.0          8         454.0         220    4354           9.0  70.0
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          4         140.0          86    2790          15.6  82.0
388  44.0          4          97.0          52    2130          24.6  82.0
389  32.0          4         135.0          84    2295          11.6  82.0
390  28.0          4         120.0          79    2625          18.6  82.0
391  31.0          4         119.0          82    2720          19.4  82.0

     origin                        name  mpg_high
0         1  chevrolet chevelle malibu         0
1         1          buick skylark 320         0
2         1         plymouth satellite         0
3         1              amc rebel sst         0
6         1            chevrolet impala         0
..      ...                        ...       ...
387       1            ford mustang gl         1
388       2                  vw pickup         1
389       1              dodge rampage         1
390       1                ford ranger         1
391       1                 chevy s-10         1

[389 rows x 10 columns]
```

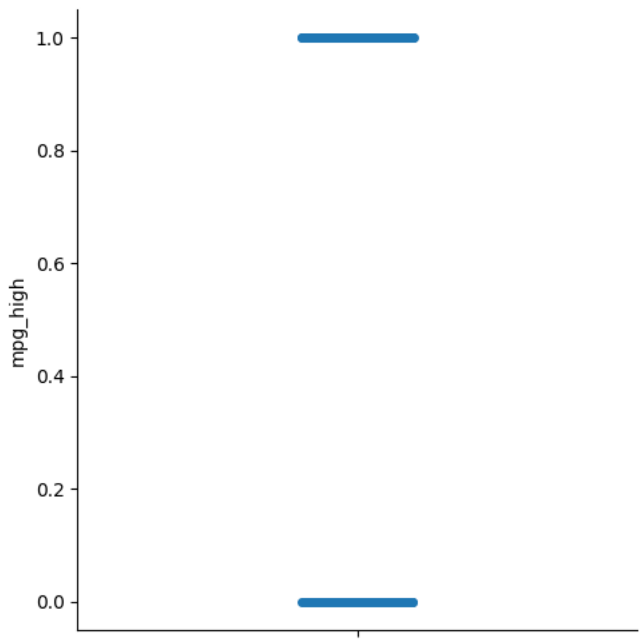
```
df = df.drop(columns=['mpg', 'name'])
```

```
df.head()
```

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

```
import seaborn as sb
sb.catplot(df['mpg_high'])
```

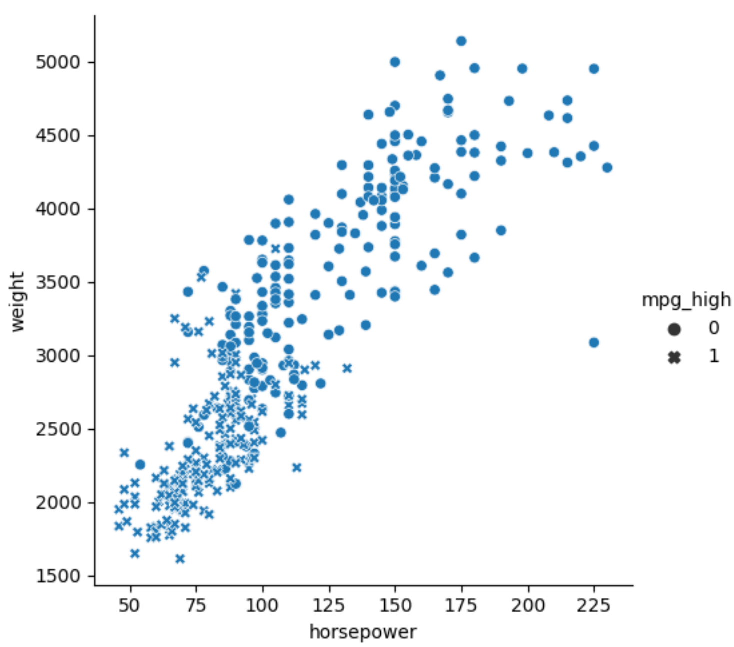
```
<seaborn.axisgrid.FacetGrid at 0x7fdf14ed5b80>
```



This graph shows that mpg_high category has the values only 0 and 1.

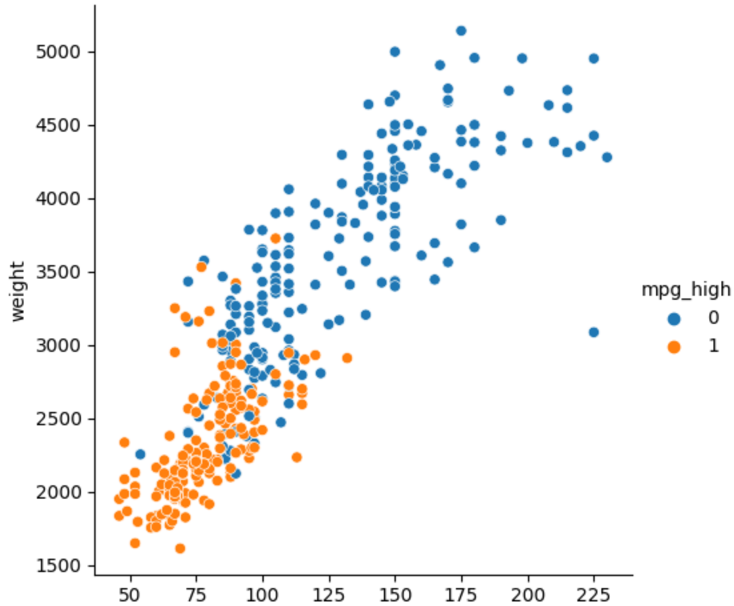
```
sb.relplot(x='horsepower', y='weight', style='mpg_high', data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fdf10879790>
```




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

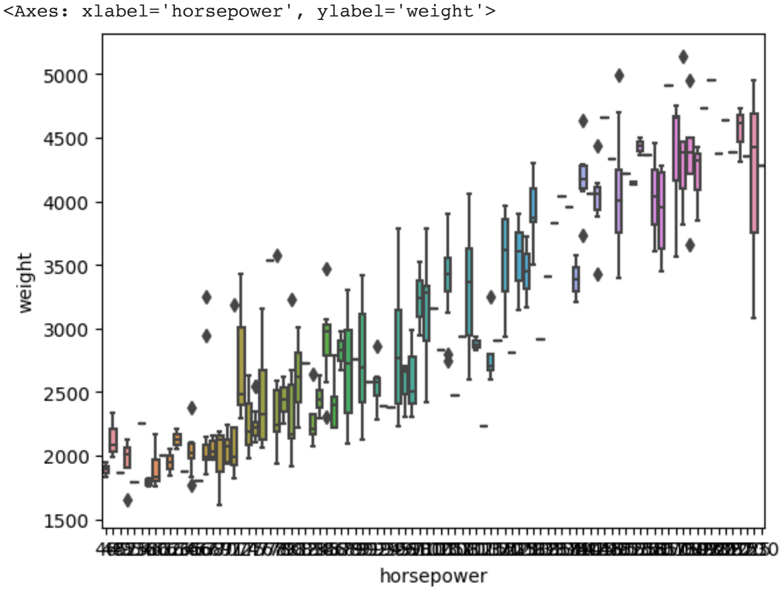
```
<seaborn.axisgrid.FacetGrid at 0x7fdf0f023df0>
```



These graphs show that observations where mpg_high has the value of 1 tend to have low horsepower and low weight compared to the observations where the mpg_high has the value of 0.

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

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



From this graph, we can say that the outliers tend to have high weight while they have low horsepower.

```
splitPercent = 0.2
seed = 1234
```

```
from sklearn.model_selection import train_test_split
allData = df
x = allData.iloc[:, 0:7]
y = allData['mpg_high']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = splitPercent, random_state = seed)
```

```
print('x_train shape: ', x_train.shape, '\tx_test shape: ', x_test.shape )
print('y_train shape: ', y_train.shape, '\t\ty_test shape: ', y_test.shape )
```

```
x_train shape:  (311, 7)        x_test shape:  (78, 7)
y_train shape:  (311,)          y_test shape:  (78,)
```

```
from sklearn.linear_model import LogisticRegression
# Train logistic regression model using LBFGS solver
model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(x_train, y_train)

# Evaluate model on test set
y_predict = model.predict(x_test)
```

```
from sklearn.metrics import classification_report
# print classification report
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.98      0.82      0.89        50
           1       0.75      0.96      0.84        28

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

```
from sklearn.tree import DecisionTreeClassifier
# Train a decision tree
decTree_model = DecisionTreeClassifier()
decTree_model.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# Evaluate model on test set
y_pred_dt = decTree_model.predict(x_test)
```

```
# Print classification report
print(classification_report(y_test, y_pred_dt.round()))
```

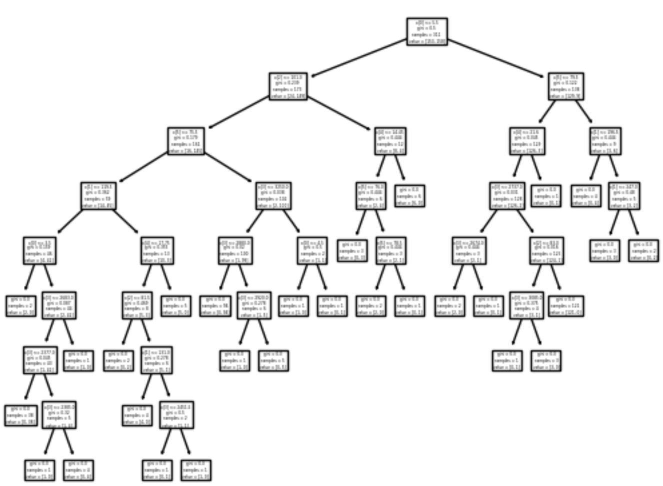
```
              precision    recall  f1-score   support

           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
```

```
from sklearn import tree
import matplotlib.pyplot as py_pl

# Plot the tree
tree.plot_tree(decTree_model)
py_pl.show()
```



Neural Network

```
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler

scaler1 = StandardScaler()
scaler2 = StandardScaler()

scaler1.fit(x_train)
scaler2.fit(x_test)

x_train_scaled = scaler1.transform(x_train)
x_test_scaled = scaler2.transform(x_test)


regr = MLPRegressor(hidden_layer_sizes=(6,3), max_iter=1000, random_state = seed)
regr.fit(x_train_scaled, y_train)
regr.fit(x_test_scaled, y_test)
```

```
▾ MLPRegressor
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=1000, random_state=1234)
```

```
from sklearn.metrics import mean_squared_error, r2_score

y_pred_reg = regr.predict(x_test_scaled)

print('mse: ', mean_squared_error(y_test, y_pred_reg))
print('correlation: ', r2_score(y_test, y_pred_reg))
```

```
mse:  0.05278456704375717
correlation:  0.770613352932701
```

```
scaler3 = StandardScaler()
scaler4 = StandardScaler()

scaler3.fit(x_train)
scaler4.fit(x_test)

x_train_scaled = scaler3.transform(x_train)
x_test_scaled = scaler4.transform(x_test)


regr2 = MLPRegressor(hidden_layer_sizes=(6,3), max_iter=1000, random_state = seed, solver='adam')
regr2.fit(x_train_scaled, y_train)
regr2.fit(x_test_scaled, y_test)
```

```
▾ MLPRegressor
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=1000, random_state=1234)
```

```
y_pred_reg2 = regr2.predict(x_test_scaled)

print('mse: ', mean_squared_error(y_test, y_pred_reg2))
print('correlation: ', r2_score(y_test, y_pred_reg2))
```

```
mse:  0.05278456704375717
correlation:  0.770613352932701
```

```
scaler5 = StandardScaler()
scaler6 = StandardScaler()

scaler5.fit(x_train)
scaler6.fit(x_test)

x_train_scaled = scaler5.transform(x_train)
x_test_scaled = scaler6.transform(x_test)


regr3 = MLPRegressor(hidden_layer_sizes=(6,3), max_iter=1000, random_state = seed, solver='lbfgs')
regr3.fit(x_train_scaled, y_train)
regr3.fit(x_test_scaled, y_test)
```

```
▾                               MLPRegressor
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=1000, random_state=1234,
             solver='lbfgs')
```

```
y_pred_reg3 = regr3.predict(x_test_scaled)

print('mse: ', mean_squared_error(y_test, y_pred_reg3))
print('correlation: ', r2_score(y_test, y_pred_reg3))
```

```
mse:  0.002110318060878252
correlation:  0.9908291606554405
```

The three models use the same number of hidden layers and neurons, but different optimization algorithms. The first two models use the 'adam' solver while the third model uses the 'lbfgs' solver. The 'adam' solver uses adaptive learning rates, which adjust during training, while 'lbfgs' is a quasi-Newton method that approximates the second derivative of the loss function to determine the direction to update the weights.

The first two models have the same MSE and correlation values, suggesting that the performance of the models is comparable. However, the third model has significantly better performance in terms of MSE and correlation. This could be because the 'lbfgs' solver is better at handling the optimization problem and converged to a better solution, resulting in lower MSE and higher correlation values. It is important to note that the performance of the models may depend on the specific dataset and the complexity of the problem.

Analysis

Comparing the two results, the decision tree algorithm had an accuracy of 0.91, higher than the logistic regression algorithm which had an accuracy of 0.87. In terms of precision and recall, the decision tree algorithm had a higher precision score for class 0, while logistic regression had a higher precision score for class 1. The recall scores were higher for both classes in the decision tree algorithm. Overall, it seems that the decision tree algorithm performed better than the logistic regression algorithm in this scenario.

In this case, there may be many reasons why the decision tree algorithm is better than the logistic regression algorithm.

First, decision trees can manage nonlinear relationships between inputs and outputs, whereas logistic regression assumes a linear relationship between inputs and outputs. This can enable decision tree algorithms to better capture complex patterns in data.

Second, decision trees are non-parametric, that is, they do not consider the distribution of the data. In contrast, logistic regression is a parametric method, that is, it assumes a certain function of the relationship between input and output. Assuming the underlying distribution of the data is a logistic regression model of the worksheet, its performance will suffer.

Finally, decision tree algorithms are more flexible than logistic regression when dealing with unbalanced data where one class has fewer samples than the other. Decision trees can change the decision at each node to better match the data, while logistic regression uses a fixed threshold.

In general, the effectiveness of the decision tree algorithm may be due to its ability to handle nonlinear relationships without assumptions about the distribution of the data and the ease of dealing with conflicting information.

Experience using R versus sklearn

Since I am new to both R and Python, it took me some time to figure out which methods to use. Overall, using R and sklearn offers flexibility for data manipulation and statistical analysis. Google Colab and RStudio are both user-friendly for building and evaluating machine learning models, although I think sklearn felt more straightforward.

0s completed at 5:25 PM