Name: Marcela Pantoja

Class: CS 4375.004

Date: 4/8/2023

```
# 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 seaborn as sb
from sklearn import datasets
# set seed for reproducibility
np.random.seed(1234)
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

Read the Auto data

```
import pandas as pd

df = pd.read_csv('Auto.csv')
print("Here are the first few rows of the data\n")
print(df.head())
print('\nDimensions of data frame:', df.shape)
```

```
Here are the first few rows of the data

    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

```
print("Describe on mpg")
df.describe().mpg
#Range of the column is
#average of column is 23.490
```

```
Describe on mpg
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
```

```
SumofMpg = df.sum().mpg
CountofMpg = df.count().mpg
AverageMPG = SumofMpg/CountofMpg

AverageMPG
```

```
23.445918367346938
```

```
Maxmpg = df.max().mpg
Minmpg = df.min().mpg
Rangempg = Maxmpg - Minmpg

Rangempg
```

```
37.6
```

```
print("Describe on weight")
df.describe().weight
#Range of the column is
```

```
Describe on weight
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
```

```
Sumofweight = df.sum().weight
Countofweight = df.count().weight
Averageweight = Sumofweight/Countofweight

Averageweight
```

```
2977.5841836734694
```

```
Maxweight = df.max().weight
Minweight = df.min().weight
Rangeweight = Maxweight - Minweight

Rangeweight
```

```
3527
```

```
print("Describe on year")
df.describe().year
#Range of the column is
```

```
Describe on year
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
```

```
Maxyear = df.max().year
MinYear = df.min().year
RangeYear = Maxyear - MinYear

RangeYear
```

```
12.0
```

```
Sumofyear = df.sum().year
Countofyear = df.count().year
Averageyear = Sumofyear/Countofyear

Averageyear
```

```
76.01025641025642
```

Explore data types

```
#a. check the data types of all columns
#Original data rtpes
df.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
```

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

```
#change the cylinders column to categorical (use cat.codes)
df1 = df.copy()
df.cylinders = df1.cylinders.astype('category').cat.codes

#change the origin column to categorical (don't use cat.codes
df.origin = df1.origin.astype('category')

#Verify new types on attributes
df.dtypes
```

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

Deal with NAs

```
#a. delete rows with NAs
df.isnull().sum()
df = df.dropna()

#b. output the new dimensions
print('\nDimensions of data frame:', df.shape)
```

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

** Modify columns**

```
#a. make a new column, mpg_high, and make it categorical:
df['mpg_high'] = list(range(len(df.index)))
df.mpg_high = df.mpg_high.astype('category')



#i. the column == 1 if mpg > average mpg, else == 0
df['mpg_high'] = np.where(df['mpg'] >= AverageMPG, 1, 0)


#b. delete the mpg and name columns (delete mpg so the algorithm doesn't just learn to predict mpg_high from mpg)
df = df.drop(columns=['mpg', 'name'])
#c. output the first few rows of the modified data frame
print(df.head())
```

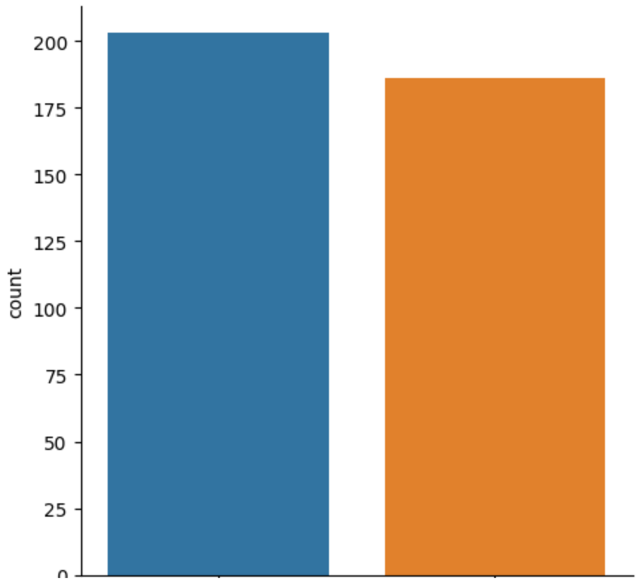
```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      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 exploration with graphs

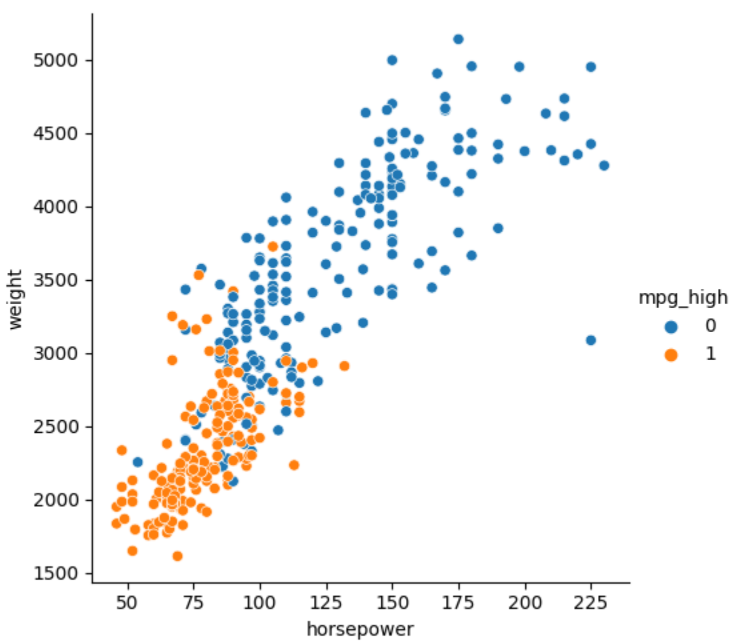
```
#a. seaborn catplot on the mpg_high column
sb.catplot(x="mpg_high", kind='count', data=df)
```

<seaborn.axisgrid.FacetGrid at 0x7f9d86f59b20>




```
#b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_high
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high)  # style=df.mpg_high)
```

<seaborn.axisgrid.FacetGrid at 0x7f9d86831c40>




```
 #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)
```

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

One thing learned about the data from each graph

Catplot: This graph shows that there are slightly more cars without a high mpg then with.

Replot: This plot shows that the cars without high mpg's on average also have a higher horsepower and weight, they are also more sparsely distributed

Boxplot: The cars with a high mpg have outliers in the data, but those without a high mpg have a higher distribution.

Train/test split

```
# 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=0)

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

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

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

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

```
0.9035369774919614
```

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

```
# evaluate
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:  0.7948717948717948
recall score:  0.9117647058823529
f1 score:  0.8493150684931507
```

```
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
```

Decision Tree

```
#train a decision tree
from sklearn.tree import DecisionTreeClassifier

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

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

```
#  test and evaluate
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:  0.8974358974358975
    precision score:  0.8823529411764706
    recall score:  0.8823529411764706
    f1 score:  0.8823529411764706
```

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

```
                  precision    recall  f1-score   support

               0       0.91      0.91      0.91        44
               1       0.88      0.88      0.88        34

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

10. Neural Network

```
# First Neural Network
from sklearn import preprocessing

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

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

```
# train
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)
```

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

```
# make predictions

pred = clf.predict(X_test_scaled)
```

```
# output results

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

```
    accuracy =  0.8846153846153846
```

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

```
                  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
```

```
# Second Neural Network
# train
#gonna change the hidden layers and solver to sgd instead of lbfgs
from sklearn.neural_network import MLPClassifier
```

```
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(6, 3), max_iter=500, random_state=1234)
clf.fit(X_train_scaled, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: Conv
  warnings.warn(
                    MLPClassifier
MLPClassifier(hidden_layer_sizes=(6, 3), max_iter=500, random_state=1234,
              solver='sgd')
```

```
# make predictions

pred = clf.predict(X_test_scaled)
```

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

```
accuracy =  0.9102564102564102
```

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

```
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        44
           1       0.83      1.00      0.91        34

    accuracy                           0.91        78
   macro avg       0.91      0.92      0.91        78
weighted avg       0.93      0.91      0.91        78
```

Compare the two models and why you think the performance was same/different

I think the performance was different mainly because of the change in solver as I only changed the solver and the hidden layers. And, when looking at the sgd algorithim, it seems to work well on data sets that have similer plots tot he Auto Data set.

Analysis

According to the classification reports and the other information, the Second Neural Network performed the best out of all three different algorithims. The Logistic regression algorithim performed the worst, with the Decision tree and 2nd Neural Network having a slightly higher accuracy score. However, the 2nd Neural Network had a much higher precision, recall and f1-score than the Decision tree which makes it the best performing algorithim.

When looking at accuracy, recall, and precision metrics by class, it can be seen that the weight and horsepower of the different cars were a good indicator of whether or not the mgp was going to be high or not. This could also be seen eaither in the data exploration through the information displayed in the graphs. But, there were also outlers with the weight class with the cars with a high mgp which influenced the three factors as well.

I believe that the Neural Network algorithm outperformed the other two because I was able to go and switch up the internal settings, such as the solver and hidden layer sizes. This is because the first neural network I tried did not outperform the decision trees algorithm and it was only after modifying it that it performed better. I especially think that the 2nd Neural Network outperformed the first two algorithms because of switch from the lbfgs solver to the sgd solver, as the stochastic gradient descent algorithm seems to fit the data based on the plots made.

I heavily prefer using R to using sklearn. This may be because I learned R before I learned Sklearn, but I find R to be much easier and clearer to use than sklearn. I prefer how data cleansing is done in R and I especially like that my data and values are on the right-hand side of the screen where I can look at them easily. I also find how R reads in data as well as how it performs the Machine Learning algorithms to be much easier to understand and cleaner to execute, especially since I have trouble getting multiple functions to print in one code block using Sklearn.

0s completed at 5:41 PM