Predicting the price of cars

1. Introduction

Car industry is one of the biggest industries worldwide with a revenue of 2,86 trillion USD [1]. There are approximately 77 million cars sold yearly [2]. In car selling, pricing is an issue that needs to be addressed. No one buys an overpriced car. On the other hand, no one sells an underpriced car. That is the reason why pricing is a real question that needs to be answered. Back in the day, the question was asked from a professional car salesperson. Nowadays there is a lot of pricing data available which helps with the pricing issue. One might look manually at the prices of similar cars and use it as a measuring stick for the pricing of his own car.

The more data is searched the more accuracy in pricing is obtained. In the evolved world we live in, the search can be automated by using computers. It is possible to make a price prediction model based on a large dataset. The machine learning model finds a correlation between the price and variables.
In our car price prediction model, we are going to use a large dataset which is scraped from postings on a popular online car marketplace [3].

The structure of this paper is as follows:
The second part is problem formulation. Detailed definition of the problem as well as the data points, labels and features will be introduced.
The third part will introduce the two different methods that are used in the price prediction.
The fourth part will present the results. For example, the accuracy of the models will be revealed.
The fifth part is going to be dedicated to the conclusions and references.

2. Problem formulation

Problem: given the data in the dataset, how accurately can we predict the price of a car?

The used data was obtained on Kaggle [3]. The dataset is made up of discrete 9,378 data points and 32 columns, with no missing features or labels. Each datapoint represents a different car, and each column represents a different feature in that car. For example there is a column for mileage, price, maker etc. These variables are defined by **strings** or **integers**, depending on its type.

We chose the data points which cost between 10 000 - 60 000 USD. The reason for that is to minimize fluctuation that is caused by the few overly cheap or extremely expensive cars. After that decision there are 8381 data points left.

Our model, a supervised learning, predicts prices, so our **label** is going to be **Price**. Out of the 32 columns we chose 2 most important columns as features, which are **Mileage** and **Year.** They are the most important because their correlation with the price is quite big, compared to others, (Table 1) and they are the biggest factor in pricing. We did not choose the maker, even though it affects the price majorly as well, because we want to know the average price of similar cars with no brand effect.

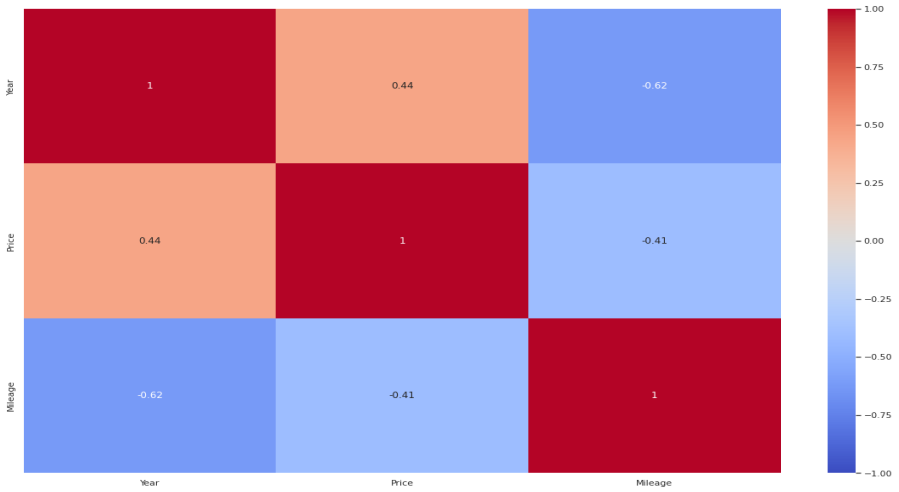
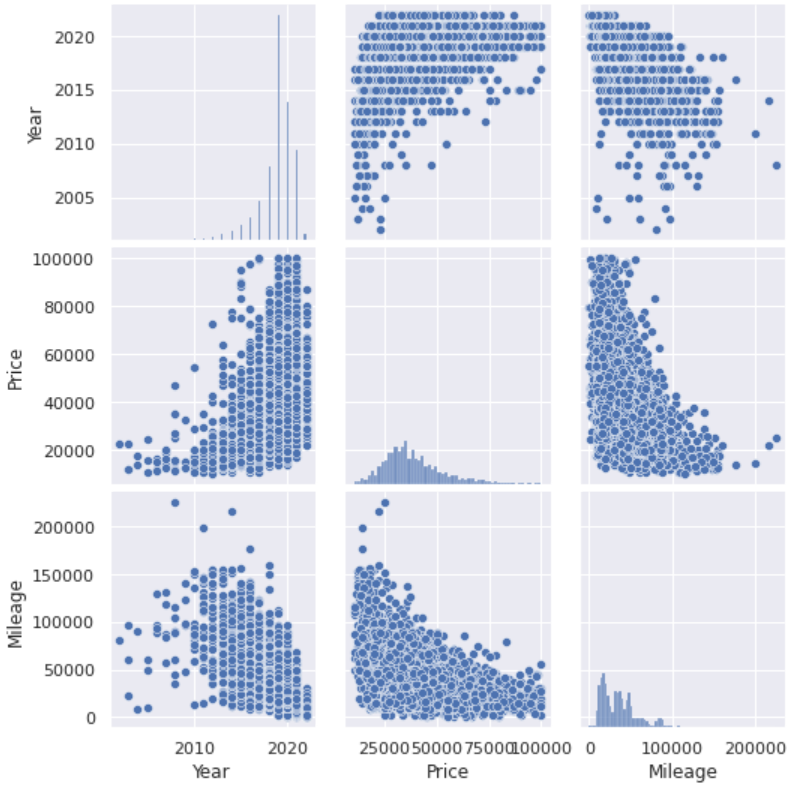


Table 1. Correlation matrix.

3. Methods

The first chosen method for this task is **polynomial regression**. It was chosen because when the label and features were put on a **sns.pairplot** function the output looked polynomial (figure 1). Since each different feature's relations with the label is polynomial, the sum of the two features relation with the price is polynomial as well.

Figure 1. As one can see, the features and labels relationship looks like a polynomial one.



The second chosen method is a **Multilayer perceptron**. MLP is used in representing a hypothesis space that includes nonlinear functions. It is the simplest type of neural work. We chose it because it showed its superiority in assignment 3. Its performance was better than polynomial regression. Based on that we assumed it might be reasonable to try it in our work as well.

3.1 Loss function

We used the **mean squared error** since it is the most commonly used option in these kinds of regression models and it allowed the use of a ready-made library. The formula of the mean squared error is $MSE = m \sum i=1(yi - bi)$. Where bi is the predicted label and yi is the real label. [4]

The **same** loss function is used in **MLP** for the same reasons and it is also recommended by the course material.

3.2 Train, validation and test sets

We split the dataset into training, validation and test datasets using the sklearn library's train_test_split() method twice. We put 60% of the data into the training

dataset, 20% into the test dataset and 20% into the validation dataset as recommended by course staff and almost everyone in the field of machine learning.

The test dataset consist of data points that have not been used to train the model or to choose between different methods.

4. Results

Figure 2

training errors, validation errors and test errors of PolynomialRegression

	poly degree	linear_train_errors	linear_val_errors	test_error
0	1	83854570.451810	83029551.825030	82582638.642289
1	2	83644241.891770	83206797.484560	82582638.642289
2	3	83666542.776601	82851983.986872	82582638.642289
3	4	83599441.937996	84057184.080924	82582638.642289
4	5	83506969.975948	99860735.366072	82582638.642289

Figure 3

training errors and validation errors of MLP

	num_hidden_layers	mlp_train_errors	mlp_val_errors
0	1	88089693.470046	91402780.652578
1	2	87915015.584408	91061290.059367
2	4	87566314.656659	90643088.625465
3	6	88389690.018802	91514648.912668
4	8	87456441.379322	90613461.913968
5	10	89010052.586718	91994824.158928

As the figure 2 and 3 shows, the results of both methods are pretty similar. Although, it is clear that the val_errors and train_errors are much lower in polynomial regression. The smallest value is marked in yellow. The most accurate method is achieved by using a 3rd degree polynomial regression. For that reason the final chosen method is a 3rd degree **polynomial regression.**

The **test error** of the final chosen method is visible in figure 2.

5. Conclusions

This work tries to find the best way to predict the price of a car. We used two different models to achieve the best results. The models tested are both supervised machine learning models. Tests were run and it came out that the best of the two models is the 3rd degree polynomial regression. However, the accuracy of it is also pretty low, 20%, predicting only about ⅕ of time. Therefore, there is a lot of room for improvement. The best way is using more features. A car is a complex mechanical device which has an enormous amount of variables that hugely affect its price and almost none of these are taken into consideration in our work, for the sake of simplicity. Logically, the next step in our project would be trying the chosen model with more features.

References:

[1] https://www.statista.com/statistics/574151/global-automotive-industry-revenue/
[2] https://www.statista.com/statistics/200002/international-car-sales-since-1990/
[3] https://www.kaggle.com/datasets/chancev/carsforsale
[4] Mean Squared Error: Definition and Example - Statistics How To

Appendix:

Untitled

October 8, 2022

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.metrics import accuracy_score, confusion_matrix  # evaluation␣
 ↪metrics

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.preprocessing import PolynomialFeatures
```

[ ]:
```
df = pd.read_csv('cars_raw.csv')
#df.head()
#df.info()

df = df[df['Price'] != 'Not Priced']
df['Price'] = df['Price'].str.replace('$','').str.replace(',','')
df['Price'] = df['Price'].astype(float)
df = df[df['Price'] < 60000]
df = df[df['Price'] > 10000]
```

[ ]:
```
df = df.drop(['Used/
 ↪New','ConsumerRating','ConsumerReviews','SellerType','SellerRating',
                 ␣
 ↪'SellerReviews','ComfortRating','InteriorDesignRating','ExteriorStylingRating','Reliability
 ↪'MinMPG','MaxMPG',
```

```
            'Transmission', 'PerformanceRating', 'Make', 'FuelType' ,␣
 ↪'Drivetrain'], axis = 1)

df = df.drop(['SellerName', 'StreetName',
              'State', 'Zipcode', 'DealType','ExteriorColor','InteriorColor',
              'Engine','VIN', 'Stock#','ValueForMoneyRating', 'Model'], axis =␣
 ↪1)
```

[ ]:
```
sn.set(rc = {'figure.figsize':(20,12)})
sn.heatmap(df.corr(), annot = True, fmt='.2g',cmap= 'coolwarm', vmin=-1, vmax=1)
plt.show()
```

[ ]:
```
pair = sn.pairplot(df)
plt.show()
```

[ ]:
```
y = df['Price']
X = df.drop('Price',axis=1)
```

[ ]:
```
# Now we choosing our predition
X_train, X_ram, y_train, y_ram = train_test_split(X, y, test_size=0.6)
X_val, X_test, y_val,y_test = train_test_split(X_ram, y_ram, test_size=0.2)
```

[ ]:
```
from sklearn.neural_network import MLPRegressor
num_layers = [1,2,4,6,8,10]    # number of hidden layers
num_neurons = 15  # number of neurons in each layer
mlp_tr_errors = []
mlp_val_errors = []
linear_val_errors = []
for i, num in enumerate(num_layers):
    hidden_layer_sizes = tuple([num_neurons]*num) # size (num of neurons) of␣
 ↪each layer stacked in a tuple

    mlp_regr = MLPRegressor(hidden_layer_sizes,random_state =42,max_iter=1000)␣
 ↪# Initialise an MLPRegressor

    mlp_regr.fit(X_train, y_train)   # Train MLP on the training set

    # YOUR CODE HERE
    #raise NotImplementedError()
    X_train_train_poly = poly.fit_transform(X_train)
    X_test_poly= poly.fit_transform(X_test)
    X_val_poly = poly.fit_transform(X_val)
    lin_regr.fit(X_train_train_poly,y_train)

    ## evaluate the trained MLP on both training set and validation set
    y_pred_train = mlp_regr.predict(X_train)    # predict on the training set
```

```
    tr_error = mean_squared_error(y_train, y_pred_train)    # calculate the␣
 ↪training error
    y_pred_val = mlp_regr.predict(X_val) # predict values for the validation␣
 ↪data
    val_error = mean_squared_error(y_val, y_pred_val) # calculate the␣
 ↪validation error
    y_pred_train_poly = lin_regr.predict(X_train_train_poly) # calculate the␣␣
 ↪ training error
    y_pred_test_poly = lin_regr.predict(X_test_poly) # calculate the test error␣
 ↪y_pred_val_poly = lin_regr.predict(X_val_poly) #calculate the validation␣
 ↪error
    y_pred_val_poly = lin_regr.predict(X_val_poly) #calculate the validation␣
 ↪error

    mlp_tr_errors.append(tr_error)
    mlp_val_errors.append(val_error)
    linear_val_errors.append(val_error)
    accuracy_test_poly = round(r2_score(y_test, y_pred_test_poly), 5)    #␣
 ↪accuracy for the test
    accuracy_train_poly = round(r2_score(y_train, y_pred_train_poly), 5) #␣
 ↪accuracy for the training
    accuracy_val_poly = round(r2_score(y_val, y_pred_val_poly), 5)       #␣
 ↪accuracy for the validation
print("accuracy of test  MLPRegressor: ", accuracy_test_poly)           ␣
 ↪#printing the accuracy of test  MLPRegressor
print("accuracy of training  MLPRegressor: ", accuracy_train_poly)      ␣
 ↪#printing the accuracy of training  MLPRegressor
print("accuracy of validatio  MLPRegressor: ", accuracy_val_poly)       ␣
 ↪#printing the accuracy of validation  MLPRegressor
```

```
fig, ax = plt.subplots(1,2,figsize = (10,4))
ax[0].set_title('Residual Plot of Train samples')
sn.distplot((y_train-y_pred_train_poly),hist = False,ax = ax[0])
ax[0].set_xlabel('y_train - y_pred_train')

# Y_test vs Y_train scatter plot

ax[1].set_title('y_test vs y_pred_test')
ax[1].scatter(x = y_test, y = y_pred_test_poly)
```

```
ax[1].set_xlabel('y_test')
ax[1].set_ylabel('y_pred_test')
plt.show()
```

[ ]:

[ ]:
```
degrees = [1,2,3,4,5]

# we will use this variables to store the resulting training and validation␣
 ↪errors for each polynomial degree
linear_tr_errors = []
linear_val_errors = []
linear_test_errors = []
for degree in degrees:
    lin_regr = LinearRegression(fit_intercept=False)
    poly = PolynomialFeatures(degree=degree)    # generate polynomial features
    X_train_poly = poly.fit_transform(X_train)    # fit the raw features
    lin_regr.fit(X_train_poly, y_train)    # apply linear regression to these␣
 ↪new features and labels

    y_pred_train = lin_regr.predict(X_train_poly)    # predict using the linear␣
 ↪model
    tr_error = mean_squared_error(y_train, y_pred_train)    # calculate the␣
 ↪training error
    X_val_poly = poly.transform(X_val) # transform the raw features for the␣
 ↪validation data
    y_pred_val = lin_regr.predict(X_val_poly) # predict values for the␣
 ↪validation data using the linear model
    val_error = mean_squared_error(y_val, y_pred_val) # calculate the␣
 ↪validation error
    test_error = mean_squared_error(y_test, y_pred_test_poly) #test error

    linear_tr_errors.append(tr_error)
    linear_val_errors.append(val_error)
    linear_test_errors.append(test_error)


    accuracy_test_poly = round(r2_score(y_test, y_pred_test_poly), 5)    #␣
 ↪accuracy for the test
```

```
    accuracy_train_poly = round(r2_score(y_train, y_pred_train_poly), 5) #␣
 ↪accuracy for the training
    accuracy_val_poly = round(r2_score(y_val, y_pred_val_poly), 5)       #␣
 ↪accuracy for the validation
    test_error = mean_squared_error(y_test, y_pred_test_poly)




print("accuracy of test PolyReg : ", accuracy_test_poly)      #printing the␣
 ↪accuracy of test PolyReg
print("accuracy of training PolyReg : ", accuracy_train_poly) #printing the␣
 ↪accuracy of training PolyReg
print("accuracy of validatio PolyReg : ", accuracy_val_poly)  #printing the␣
 ↪accuracy of validation PolyReg


fig, ax = plt.subplots(1,2,figsize = (10,4))
ax[0].set_title('Residual Plot of Train samples')
sn.distplot((y_train-y_pred_train_poly),hist = False,ax = ax[0])
ax[0].set_xlabel('y_train - y_pred_train')

# Y_test vs Y_train scatter plot


ax[1].set_title('y_test vs y_pred_test')
ax[1].scatter(x = y_test, y = y_pred_test_poly)
ax[1].set_xlabel('y_test')
ax[1].set_ylabel('y_pred_test')
plt.show()
```

[ ]:
```
errors = {"num_hidden_layers":num_layers,
          "mlp_train_errors":mlp_tr_errors,
          "mlp_val_errors":mlp_val_errors
         }
pd.DataFrame(errors)
```

[ ]:
```
l_errors = {"poly degree":degrees,"linear_train_errors":linear_tr_errors,␣
 ↪"linear_val_errors":linear_val_errors,}
print("training errors and validation errors of PolynomialRegression")
pd.DataFrame(l_errors).style.applymap(lambda x: "background-color: yellow" if␣
 ↪x==np.min(linear_val_errors) else "background-color: white")
```

[ ]:
```
errors = {"num_hidden_layers":num_layers,
          "mlp_train_errors":mlp_tr_errors,
          "mlp_val_errors":mlp_val_errors
```

```
        }
pd.DataFrame(errors)
```

[ ]:
```
# create a table to compare training and validation errors
print("training errors, validation errors and test errors of␣
 ↪PolynomialRegression")
errors = {"poly degree":degrees,
          "linear_train_errors":linear_tr_errors,
          "linear_val_errors":linear_val_errors,
          "test_error":linear_test_errors
         }
pd.DataFrame({ key:pd.Series(value) for key, value in errors.items()})
```

[ ]:
```
l_errors = {"poly degree":degrees,"linear_train_errors":linear_tr_errors,␣
 ↪"linear_val_errors":linear_val_errors,"test_error":linear_test_errors}
print("training errors, validation errors and test errors of␣
 ↪PolynomialRegression")
pd.DataFrame(l_errors).style.applymap(lambda x: "background-color: yellow" if␣
 ↪x==np.min(linear_val_errors) else "background-color: white")
```

[ ]:
```
tr_error = mean_squared_error(y_pred_train_poly , y_train) # calculate the␣
 ↪training error
X_val_poly = poly.fit_transform(X_val)
y_pred_val = lin_regr.predict(X_val_poly)
val_error = mean_squared_error(y_pred_val, y_val)
print("Polynomial degree: ", degree, "Training error: ", tr_error)
print("Polynomial degree: ", degree, "Validation error: ", val_error)
```

[ ]:
```

```

[ ]:
```

```