Predicting Used Car Prices with a Decision Tree and Polynomial Regression

Introduction

When wanting to buy a car second hand, how do you know the price is fair - especially if you know nothing about cars? This is the problem we aimed to solve by predicting used car prices based on the car's characteristics. In section "problem formulation" we will first discuss the aims of the project and describe the dataset and data points used. In the "methods" section we will explain all the preprocessing done and discuss the different machine learning methods used. In the "results" section we will discuss and interpret the results by comparing the outcomes of the two machine learning models. Finally, we give our final thoughts and conclusion in the "conclusion" section. In summary, after training two models based on decision trees and polynomial regression we concluded that the decision tree was the better choice.

Problem formulation

The aim of the project was to accurately predict the selling price of a used car based on information about the car and its condition. In order to simplify and scope down the problem, we decided to only consider Ford trucks instead of a wide range of different types of cars.

Our dataset is a collection of used car sales listings from Craigslist.org, collected by a scraper built by Austin Reese. The dataset can be found on kaggle.com under the name "Used Cars Dataset" (Used Cars Dataset, 2021). The data points represent a sales listing of a used car on Craigslist. That is, a data point includes information about the technical details, condition and age of a car in the form of a csv file. One row of data represents one sales listing.

The features include the year the car was manufactured (Year), how many miles the car has been driven for (Distance), the type of fuel (Fuel: diesel, gas, hybrid, electric, other) and the type of transmission (Transmission: automatic, manual, other). As we are trying to predict price, we are using supervised learning. The label is the selling price listed on Craigslist. Year, distance and price are discrete data while fuel and transmission are categorical data.

Methods

Before training our model, we had to do some preprocessing. As mentioned above our data was in the form of a csv file: First we dropped any possible rows where the price was set to 0 as well as rows with null values (or empty strings) in any of the columns. We also removed all columns that did not correspond to our features or label.

As we had decided to scope down the data to Ford trucks, we chose only those rows that had the manufacturer-column set as ‘ford’ and type-column as ‘truck’. We also decided to convert the transmission and fuel types from strings to integers in order to simplify working with them. You can find our conversion tables below (electric transmission is excluded as those were no longer left in our data after focusing on Ford trucks):

Transmission	Number
Manual	0
Automatic	1
Other	2

Fuel	Number
gas	0
diesel	1
hybrid	2
other	3

Lastly, all columns (i.e. features and label) were transformed into numpy arrays and then saved in their respective variables. We also made sure to transform string-valued numbers to integers. Finally, we ended up with 9564 rows and 5 columns of data, i.e. 9564 data points. All of the data was stored either as integers or decimal numbers.

Our chosen features, which were explained under problem formulation, were directly available in the data as columns. They were chosen based on the expert opinion of a family member by interviewing them about which aspects affect car price the most, as well as our own intuition and knowledge: The age and kilometres driven are good indicators of a car’s condition, which naturally affects the price. The type of fuel and transmission also can affect the price, especially between otherwise two similar cars: e.g. typically hybrid-automatic cars are more expensive than gas-manual ones.

For our first method we came to the conclusion that using DecisionTreeRegressor would yield us the best results. As we want to estimate a price given certain conditions it would be more beneficial for us to use regression models than classification models as we do not want discrete but rather continuous values for our predicted labels (Regression vs. Classification in Machine Learning: What’s the Difference?, 2021). Other reasons why we picked decision trees were because they are very easy to use, read, understand, and more resistant to outliers than other models. (Decision Tree Regression in Machine Learning, 2020)

The model works by constructing decision leaves and nodes and then asking true or false questions starting from the root node. By answering said questions it navigates through the tree until it eventually reaches a leaf node. The model leverages recursive partitioning to construct the decision tree. As decision trees employ supervised learning they are able to map data points to outputs. (Decision Tree Regression in Machine Learning, 2020)

For the second method, we wanted to try to use another regression method with the same data split. We decided to use polynomial regression as it was clear to us that with such a complex combination of features there likely was not going to be any linear correlation. Another reason for choosing polynomial regression over other regression methods was that we were most familiar with it and the logic behind it was easy to understand.

The polynomial regression model works by drawing a line of best fit (Agrawal, 2022). Unlike linear regression it leverages the non-linear relationship between the features and the labels to draw more of a curve of best bit instead of a line. The reason we decided to pick polynomial regression instead of Linear was exactly because of that (Agrawal, 2022). For example we may have features that increase or decrease the price of the car exponentially instead of linearly.

We decided to use mean squared error with both methods, as it is clear to see that the training data we have does not exactly fit all the points from our validation data. Hence, we have noise in our prediction. Due to the Central Limit Theorem our total noise is distributed normally. Because of this, the best way to reduce the noise in our prediction and make our prediction more accurate is by reducing our variance in the distribution. The variance is given by the standard deviation (σ). In this case it also represents the noise in our prediction. Since variance is just the standard deviation squared (σ^2) it is obvious to see the fastest way to get the noise (σ) to get as close as possible to 0 is by using the mean squared error loss function. (Why do we use the square loss, 2015)

The mean squared error loss functions works by trying to get the value of r in equation 1:

$$r(x, y) = \frac{1}{m} \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) - y \quad (1)$$

where m is the number of datapoints, h is the value of the predicted label when given the input of x^i. The ith feature in the feature vector is x^i and the true value of the label is y. (Why do we use the square loss, 2015).

Choosing the training, validation and testing sets was simply done by splitting the data 70/15/15: 6694 data points for testing, 1434 for both validation and testing sets. For the sake of convenience, we simply used the train test split function. For our dataset, there was really no difference which rows belong to which set so this approach made sense. We chose the percentages 70/15/15 since we wanted to keep most of our data for training and 70/15/15 intuitively felt like a good choice.

Results

To our surprise, the polynomial regression does not produce great results at all. In fact, it gives us an accuracy of 0% along with a training error of 383867867, and a validation error of 383867867. This clearly shows us that using polynomial regression is the wrong choice for our data set. Our hypothesis for why this is the case is that our data includes a lot of outliers and a wide variety of different price ranges for cars: when a polynomial regression model is given a dataset that had that wide of a range would struggle in being accurate (Agrawal, 2022). For example, one would assume that the older the car is and the more miles it has on it the cheaper the car would be. In our dataset we have cars that at their time of release would be considered luxury cars and some cars that would be considered budget cars. So even though the luxury car is older and has more miles on it, it is significantly more expensive than a newer budget car. On top of that, since the prices we have are from Craigslist there may be many people listing their cars for way more than what the car is worth or way cheaper than it is worth. Hence, our polynomial regression model struggles.

For decision tree regression however bears more fruitful results. We have an accuracy of 92.76% along with a training error of 7.70, validation error of 7.24. Our hypothesis for why the decision tree performed better was because it was able to create leaves and nodes to traverse the data (Decision Tree Regression in Machine Learning, 2020), effectively considering different categories of cars separately: following a different path in the decision tree if the car was a e.g. manual or automatic. This way the decision tree might be able to narrow down a price range more effectively based on all the different category tags.

In the end, it is obvious that our method of choice would be a decision tree. Not only does it have better accuracy, but being able to leverage separating the dataset is a huge advantage for the decision tree. The final testing error is 32.2, with the test set constructed as mentioned in the methods section of the report.

Conclusion

In conclusion, we find that the price of a car is affected by many factors - some of which are not the first that come to mind. We see that trying to draw a line or curve of best fit with this data does not seem to be the best way to go. Instead, using decision trees and their ability to create nodes and leaves produces the most accurate predictions (Decision Tree Regression in Machine Learning, 2020). All in all, we believe that with an accuracy of almost 93% the problem that we set out to solve is in fact satisfactorily solved but there is still room for improvement: our testing error is considerably higher than our training and validation errors. Another area of improvement could be using a more varied selection of cars and not only one specific brand and one specific type of car from that brand. It is also possible that with a dataset consisting of actual car sales instead of listings we would be able to obtain more accurate results, as the dataset would contain fewer outliers caused by inaccurate prices.

References

Used Cars Dataset. (2021, May 6). Kaggle. Retrieved September 23, 2022, from https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data

Regression vs. Classification in Machine Learning: What's the Difference?. (2021, October 6). Springboard. Retrieved September 23, 2022, from https://www.springboard.com/blog/data-science/regression-vs-classification/#:~:text=The%20most%20significant%20difference%20between,types%20of%20machine%20learning%20algorithms.

Decision Tree Regression in Machine Learning. (2020, December 4) Upgrad. Retrieved September 23, 2022, from https://www.upgrad.com/blog/pros-and-cons-of-decision-tree-regression-in-machine-learning/

Why do we use the square loss? (2016, February 23). DataScience Stackexchange. Retrieved September 23, 2022, from https://datascience.stackexchange.com/questions/10188/why-do-cost-functions-use-the-square-error

Agrawal, R. (2022, July 26). All you need to know about Polynomial Regression. Analytics Vidhya. Retrieved October 9, 2022, from https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/

Cars

October 9, 2022

Import all the needed libraries

```
[18]: import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures    # function to generate␣
 ↪polynomial and interaction features
from sklearn.linear_model import LinearRegression    # classes providing Linear␣
 ↪Regression with ordinary squared error loss and Huber loss, respectively
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
```

Read the data

```
[19]: df = pd.read_csv('vehicles.csv')
```

Check if the correct data as imported

```
[20]: nan_value = float("NaN")
df.replace("", nan_value, inplace=True)
df.drop((df['price'] == 0).index)
df.dropna(subset =␣
 ↪['price','year','odometer','type','fuel','transmission','cylinders','drive'],␣
 ↪inplace=True)
df = df[['price','year','type','manufacturer','odometer', 'fuel',␣
 ↪'transmission']]
```

```
[21]: df = df[df['manufacturer'] == 'ford']
df = df[df['type'] == 'truck']
df = df.replace('gas',0)
df = df.replace('diesel',1)
df = df.replace('hybrid',2)
df = df.replace('other',3)
df = df.replace('electric',4)
df = df.replace('manual',0)
df = df.replace('automatic',1)
df = df.replace('other',2)
```

[22]:
```
price = df['price']
year = df['year']
distance = df['odometer']
fuel = df['fuel']
transmission = df['transmission']
```

[23]:
```
price = price.to_numpy()
year = year.to_numpy()
distance = distance.to_numpy()
fuel = fuel.to_numpy()
transmission = transmission.to_numpy()
```

[24]:
```
df = df[['price','year','odometer', 'fuel', 'transmission']]
```

[25]:
```
display(df)
```

```
        price    year  odometer  fuel  transmission
31      15000  2013.0  128000.0     0             1
65      22500  2001.0  144700.0     1             0
175     12500  2008.0  141345.0     0             1
177     22950  2014.0  166380.0     0             1
337      8950  2011.0  164000.0     0             1
...       ...     ...       ...   ...           ...
426692  56988  2019.0   18671.0     0             1
426716  65950  2020.0   21300.0     1             1
426766  55995  2019.0   23670.0     0             1
426776  52995  2019.0   31558.0     0             1
426785  23495  2015.0  146795.0     0             1

[9564 rows x 5 columns]
```

[26]:
```
features = []
labels = []
m = 0
for row in df.iterrows() :
    datapoint_one = distance[m].astype('int')
    datapoint_two = year[m].astype('int')
    datapoint_three = fuel[m]
    datapoint_four = transmission[m]
    label_one = price[m]
    features.append([datapoint_one,␣
 ↪datapoint_two,datapoint_three,datapoint_four])
    labels.append(label_one)
    m = m + 1
```

```
print(m)
```

```
9564
```

[27]:
```
X = np.array(features)
y = np.array(labels)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,␣
 ↪random_state = 1)
X_val, X_test, y_val, y_test = train_test_split(X, y, test_size=0.5,␣
 ↪random_state = 1)
```

[29]:
```
regressor = DecisionTreeRegressor(criterion = 'squared_error',random_state=1)
regressor.fit(X_train, y_train)
y_pred_val= regressor.predict(X_val)
y_pred_train = regressor.predict(X_train)
accuracy_val = accuracy_score(y_val,y_pred_val.astype('int'))
accuracy_train = accuracy_score(y_train,y_pred_train.astype('int'))
```

[34]:
```
y_reg= regressor.predict(X_test)
```

[35]:
```
acc_reg = accuracy_score(y_test,y_reg.astype('int'))
```

[36]:
```
lin = LinearRegression()
lin.fit(X_train,y_train)
y_lin = lin.predict(X_val)
acc = accuracy_score(y_val,y_lin.astype('int'))
```

[37]:
```
poly = PolynomialFeatures(degree= 10)     # initialize a polynomial feature␣
 ↪transformer
X_poly = poly.fit_transform(X_train)     # fit and transform the raw features

lin_regr = LinearRegression(fit_intercept=True) # NOTE: "fit_intercept=False"␣
 ↪as we already have a constant iterm in the new feature X_poly
lin_regr.fit(X_poly,y_train)# fit linear regression to these new features and␣
 ↪labels (labels remain same)
X_val = poly.fit_transform(X_val)

y_lin = lin_regr.predict(X_val)     # predict using the learnt linear model
tr_error = mean_squared_error(y_val,y_lin)     # calculate the training error
acc = accuracy_score(y_val,y_lin.astype('int'))
```

[ ]:
```

```