MLprojectGoranDelic

May 30, 2024

Problem statement

A student wants to become a businessman and try earning some real money. He decides to open a car dealership. Even though his father taught him a lot about cars, he did not teach him how certain car properties affect the price of a car. Therefore, the student decides to delve into the problem himself and use a machine learning model on a dataset about cars. He wants to achieve: Predicting the price of cars based on a wide range of attributes and features. Using a dataset containing car details such as year of manufacture, car power, mileage and more, he aims to develop a machine learning model that accurately estimates the price of different car models.

Data Information

ID: Identification Number for Each Car

Name: Name of the Car Model

Location: Location of the car

Year: Year of manufacture

Km_Driven Number of kilometers driven with the car

Fuel_Type: Type of Fuel Used (CNG, Diesel, Petrol, LPG, Electric)

Transmission: Type of transmission (Manual, Automatic)

Owner_Type: Number of previous owners

Mileage: Kilometers driven per liter of fuel spent (Kilometers per liter)

Engine: Engine power (Cubic centimeters)

Power: Power of the car (Break horsepower)

Seats: Number of seats in the car

Price: Price of the car

Importing modules

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.interpolate import interp1d
```

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
from IPython.display import display
from catboost import CatBoostRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import make_pipeline

import warnings
warnings.filterwarnings('ignore')
```

Importing the dataset and inspecting it

A small remark: I converted the ruppies to EUR and removed a couple of columns from the dataset before importing it.

[ ]:
```python
df = pd.read_csv('/kaggle/input/carspriceprediction-goran-delic/CarInfo.csv')
print(df.head())
```

Performing datachecks

[ ]:
```python
#checking for missing values
df.isnull().sum()
```

[ ]:
```python
#checking for duplicates
df.duplicated().sum()
```

[ ]:
```python
#checking datatypes
df.dtypes
```

Descriptive data analysis

[ ]:
```python
# Price Analysis
plt.figure(figsize=(8, 6))
sns.histplot(data=df['Price'], bins=20, kde=True)
plt.title('Distribution of Price')
plt.show()
```

[ ]:
```python
# Calculate average price for each car model
avg_prices_by_car = df.groupby('Name')['Price'].mean().
  ↪sort_values(ascending=False)

# Plot top N car models by average price
n = 20  # Number of top car models to plot
top_car_models = avg_prices_by_car.head(n)

plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=top_car_models.values, y=top_car_models.index)
plt.title(f'Top {n} Car Models by Average Price')
plt.xlabel('Average Price')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```

[ ]:
```
n = 20  # Number of top car models to plot
top_car_models = df['Name'].value_counts().head(n)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_car_models.values, y=top_car_models.index)
plt.title(f'Top {n} Car Models by Frequency')
plt.xlabel('Frequency')
plt.ylabel('Car Model')
plt.tight_layout()
plt.show()
```

#

Insights from data analysis:

Oldest car: 1998

Newest car: 2019

Average car age: 2013

Most expensive car: Range Rover

Most popular car: Mahindra

Average car price is around €10,400

Data Pre-Processing

[ ]:
```
# Removing missing values
df.dropna(inplace=True)
```

[ ]:
```
df.isnull().sum()
```

[ ]:
```
#Converting 'Mileage', 'Engine' and 'Power' to numeric (removing units from␣
↪rows)
df['Mileage'] = df['Mileage'].str.extract('(\d+\.\d+)', expand=False).
↪astype(float)
df['Engine'] = df['Engine'].str.extract('(\d+)', expand=False).astype(int)
df['Power'] = df['Power'].str.extract('(\d+\.\d+)', expand=False).astype(float)
```

[ ]:
```
print(df.head())
```

[ ]:
```
 # Extract brand and model from CarName
df['Brand'] = df['Name'].apply(lambda x: x.split(' ')[0])
df['Model'] = df['Name'].apply(lambda x: ' '.join(x.split(' ')[1:]))

# Defining categorical and numerical columns
categorical_columns = ['Fuel_Type', 'Transmission', 'Location', 'Owner_Type',␣
 ↪'Brand','Model']
numerical_columns = ['Year', 'Km_Driven', 'Mileage', 'Engine', 'Power',
                     'Seats']

# Encoding categorical variables
label_encoder = LabelEncoder()
for column in categorical_columns:
    df[column] = label_encoder.fit_transform(df[column])
```

Training the model(LinearRegression)

[ ]:
```
# Splitting the dataset
X = df.drop(['Price', 'Name'], axis=1)
y = df['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# Removing missing values
X.dropna(inplace=True)
X_train.dropna(inplace=True)
X_test.dropna(inplace=True)

#Ensuring consistency of rows
y_train = y_train[X_train.index]
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
```

[ ]:
```
X_test.isnull().sum()
```

[ ]:
```
# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

[ ]:
```
#Checking for inconsistencies
print("Shape of y_test:", y_test.shape)
print("Shape of y_pred:", y_pred.shape)
```

[ ]:
```
# Find indices where y_test has missing values
missing_indices = np.isnan(y_test)
```

```
# Create an array of indices for y_test and y_pred
index_array = np.arange(len(y_test))

# Remove indices where y_test has missing values
index_array = index_array[~missing_indices]

# Truncate y_pred to match the common length of y_test and y_pred
y_pred_truncated = y_pred[:len(index_array)]

# Ensure that index_array and y_pred_truncated have the same length
index_array = index_array[:len(y_pred_truncated)]

# Create a linear interpolation function for truncated y_pred
interp_func = interp1d(index_array, y_pred_truncated, kind='linear',␣
 ↪fill_value='extrapolate')

# Interpolate missing values in y_test
y_test_interp = y_test.copy()
y_test_interp[missing_indices] = interp_func(np.
 ↪arange(len(y_test_interp)))[missing_indices]
```

```
[ ]: # Truncate y_test_interp to match the length of y_pred_truncated
y_test_interp = y_test_interp[:len(y_pred_truncated)]

# Evaluating the model
mse = mean_squared_error(y_test_interp, y_pred_truncated)
r2_square = r2_score(y_test_interp, y_pred_truncated)
print(f" R-squared: {r2_square}")
print(f'Mean Squared Error: {mse}')
```

Training the model (CatBoost)

```
[ ]: # Splitting the dataset for CatBoost
X = df.drop(['Price', 'Name'], axis=1)
y = df['Price']
X_trainCat, X_testCat, y_trainCat, y_testCat = train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)

#Ensuring consistency of rows
y_trainCat = y_trainCat[X_trainCat.index]
print("Shape of X_train:", X_trainCat.shape)
print("Shape of y_train:", y_trainCat.shape)
```

```
[ ]: # Model training
catboost_model = CatBoostRegressor()
catboost_model.fit(X_trainCat, y_trainCat)
```

```
# Predictions
y_pred_catboost = catboost_model.predict(X_testCat)
```

[ ]:
```
# Evaluate the model
mse_catboost = mean_squared_error(y_testCat, y_pred_catboost)
r2_square_catboost = r2_score(y_testCat, y_pred_catboost)
print(f" R-squared (CatBoost): {r2_square_catboost}")
print(f'Mean Squared Error (CatBoost): {mse_catboost}')
```

Hyperparameter experimentation

I decided to explore different hyperparameters on both models and see how the results are affected.

[ ]:
```
# Hyperparameters for LinearRegression
linear_params = {
    'fit_intercept': True,
    'n_jobs': -1,
    'positive': True,
}
# Train Linear Regression with best hyperparameters
best_linear_model = LinearRegression(**linear_params)
best_linear_model.fit(X_train, y_train)
# Predictions
y_predNew = model.predict(X_test)

# Fixing inconsistencies before evaluation
# Find indices where y_test has missing values
missing_indices = np.isnan(y_test)

# Create an array of indices for y_test and y_pred
index_array = np.arange(len(y_test))

# Remove indices where y_test has missing values
index_array = index_array[~missing_indices]

# Truncate y_pred to match the common length of y_test and y_pred
y_pred_truncatedNew = y_pred[:len(index_array)]

# Ensure that index_array and y_pred_truncated have the same length
index_array = index_array[:len(y_pred_truncatedNew)]

# Create a linear interpolation function for truncated y_pred
interp_func = interp1d(index_array, y_pred_truncatedNew, kind='linear',␣
 ↪fill_value='extrapolate')

# Interpolate missing values in y_test
y_test_interpNew = y_test.copy()
```

```
y_test_interpNew[missing_indices] = interp_func(np.
 ↪arange(len(y_test_interpNew)))[missing_indices]

# Truncate y_test_interp to match the length of y_pred_truncated
y_test_interpNew = y_test_interpNew[:len(y_pred_truncatedNew)]

# Evaluating the model
mseCustom = mean_squared_error(y_test_interpNew, y_pred_truncatedNew)
r2_squareCustom = r2_score(y_test_interpNew, y_pred_truncatedNew)
print(f" R-squared: {r2_squareCustom}")
print(f'Mean Squared Error: {mseCustom}')
```

```
[ ]: # Define hyperparameters for CatBoost
custom_catboost_params = {
    'learning_rate': 0.05,      # Learning rate for gradient boosting
    'depth': 6,                 # Depth of the trees
    'iterations': 500,          # Number of boosting iterations (trees)
    'l2_leaf_reg': 3,           # L2 regularization coefficient
    'loss_function': 'RMSE'     # Loss function to optimize
}
# Train CatBoost with best hyperparameters
custom_catboost_model = CatBoostRegressor(**custom_catboost_params)
custom_catboost_model.fit(X_train, y_train)

# Evaluate model
catboost_mseCustom = mean_squared_error(y_test_interpNew, custom_catboost_model.
 ↪predict(X_test))

# Print results
print("CatBoost MSE:", catboost_mseCustom)
```

After exploring a few different combinations of hyperparameters, the results were not affected by a significant amount.

RandomSearch

```
[ ]: # Define the parameter grid for RandomizedSearchCV
param_dist = {
    'iterations': stats.randint(100, 300),
    'learning_rate': stats.uniform(0.01, 0.1),
    'depth': stats.randint(4, 8),
    'l2_leaf_reg': stats.randint(1, 7),
    'bagging_temperature': stats.uniform(0, 1),
    'random_strength': stats.randint(1, 3)
}
# Initialize the CatBoost Regressor
catboost_model = CatBoostRegressor(silent=True)
```

```
# Initialize RandomizedSearchCV with CatBoost Regressor
random_search = RandomizedSearchCV(estimator=catboost_model,␣
 ↪param_distributions=param_dist, n_iter=50, cv=3, n_jobs=-1, verbose=2,␣
 ↪scoring='r2', random_state=42)

# Fit RandomizedSearchCV
random_search.fit(X_trainCat, y_trainCat)

# Get the best parameters and model
best_params = random_search.best_params_
best_model = random_search.best_estimator_

# Print the best parameters
print("Best parameters found by RandomizedSearchCV: ", best_params)
```

```
[ ]: # Predict using the best model
y_pred_catboost_best = best_model.predict(X_testCat)

# Evaluate the best model
mse_catboost_best = mean_squared_error(y_testCat, y_pred_catboost_best)
r2_square_catboost_best = r2_score(y_testCat, y_pred_catboost_best)

print("Best CatBoost Model from RandomizedSearchCV:")
print(f" R-squared: {r2_square_catboost_best}")
print(f"Mean Squared Error: {mse_catboost_best}")
```

RESULTS

```
[ ]: #Linear Regression results
pred_df=pd.DataFrame({'Actual Value':y_test_interp,'Predicted Value':
 ↪y_pred_truncated,'Difference':y_test_interp-y_pred_truncated})
display(pred_df)
```

```
[ ]: #CatBoost results
pred_df_catboost = pd.DataFrame({'Actual Value': y_testCat, 'Predicted Value␣
 ↪(CatBoost)': y_pred_catboost, 'Difference': y_test - y_pred_catboost})
display(pred_df_catboost)
```

```
[ ]: # Print evaluation metrics
print("Evaluation Metrics:")
print(f"Linear Regression - R-squared: {r2_square}, Mean Squared Error: {mse}")
print(f"CatBoost - R-squared: {r2_square_catboost}, Mean Squared Error:␣
 ↪{mse_catboost}")
```

```
[ ]: #CatBoost after RandomizedSearchCV for finding best parameters:

improvement_r2 = r2_square_catboost_best - r2_square_catboost
```

```
improvement_mse = mse_catboost - mse_catboost_best

print("Comparison:")
print(f"Improvement in R-squared: {improvement_r2}")
print(f"Improvement in Mean Squared Error: {improvement_mse}")
```

Conclusion

CatBoost significantly outperforms Linear Regression in this scenario based on the evaluation metrics. Here's why:

- **R-squared**:
  - R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variables. A higher R-squared value indicates that the model explains more of the variance in the data. In this case, CatBoost achieved an R-squared value of 0.8641, which indicates that it explains around 86.41% of the variance in the data. On the other hand, Linear Regression resulted in a negative R-squared value (-0.4021), which suggests that the model performs worse than simply using the mean of the dependent variable for prediction.
- **Mean Squared Error (MSE)**:
  - MSE measures the average squared difference between the actual and predicted values. A lower MSE indicates better performance, as it means the model's predictions are closer to the actual values. CatBoost achieved a much lower MSE of 23,943,609.24 compared to Linear Regression's MSE of 303,281,332.84. This suggests that CatBoost's predictions are more accurate and closer to the actual values compared to Linear Regression.
- I would assume that a big part of why I got this result is because I had to deal with a lot of missing data for Linear Regression, unlike with CatBoost which accepts the data with missing values and has the ability to handle it within the model itself. Hence, the more precise prediction.

After the attempt of using GridSearch, and the realization its taking too much computational power for the current task, I shifted into using RandomizedSearch and found the "best" hyperparameters:

Fitting 3 folds for each of 50 candidates, totalling 150 fits Best parameters found by RandomizedSearchCV: {'bagging_temperature': 0.8422847745949985, 'depth': 7, 'iterations': 261, 'l2_leaf_reg': 2, 'learning_rate': 0.09948273504276488, 'random_strength': 2}

Despite the extensive search for better hyperparameters, the original model's simple setup appears to perform slightly better in terms of both R-squared and MSE, as the comparison shows:

Improvement in R-squared: -0.004421008547851768 - marginal decrease in R-squared compared to the original model, implying the new model fits the data slightly worse in terms of explained variance.

Improvement in Mean Squared Error: -778751.0006942637 - less accurate than those from the original model.

EXTERNAL LINKS

ChatGpt

KAGGLE