KPI's for Used Car Prices

An Exploration into Random Forest with Hyperparameter Tuning

What determines a used car price, age, mileage, or something else? In this analysis a Random Forest model with hyperparameter tuning and cross validation will attempt to illustrate the key factors for determining used car price. This model can be utilized to assist in setting the base price for a used car at a dealership without the guess work involved in setting a price. This model does not take into consideration local infrastructure in consumer behavior nor popularity of model choice to a region, however it serves as an indicator over what drives a used car price.

Methodologies

Used car price data was uploaded using Spark and Hadoop, the data was cleaned using Pandas and normalized. A pipeline was built for the Random Forest and the Random Forest was trained then tested. Visualization of the KPI's based on feature importance for was built using MatPlotLib.

Data Analysis

The Random Forest returned over 100 trees, the model was best fit using hyperparameters and cross validated for performance using the default Root Mean Square Error, Mean Square Error, Mean Absolute Error, and R square as an illustration of hyperparameter tuning.

Summary

Random Forest analysis of 100 trees returned Mean Square Error (the default) as the best model to utilize in determining factors for used car price. Going forward, it is feasible the model will return a prediction based upon key factors and may be utilized as an accurate base price for a used car, less local conditions and markets.

Conclusion

The data set used for this example of Random Forest with Hyperparameter Tuning was collected in 2015, further insight into any used car price setting or utlization of this model will need to include more up to date data. Further analysis, including mapping key performance indicators (KPI's) of the Random Forest may be further utilized to gain insight into the data to determine weights of each of the variables and how they interplay into the Random Forest estimation. This project is an illustration of building a Random Forest model using limited data, however it is with the basic usage of this model that a building of understanding on larger data sets can be aquired and easily understood in the future.

In [ ]:
```
#Installing latest version of Pyspark via Hadoop
!pip install pyspark
!conda install -c menpo wget
!wget -q https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop3.2.tgz
```

In [ ]:
```
#Installing Java
!pip install install-jdk
```

In [ ]:
```
#Importing the necessary libraries
import os
os.environ["JAVA_HOME"] = ("\\users\odont\appdata\local\pip\cache\wheels\89\a9\a3\03dc1
os.environ["SPARK_HOME"] = "\\users\odont\anaconda3\lib\site-packages"
import findspark
findspark.init()
from google.colab import files
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import isnan, when, count, col, lit
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder
```

In [ ]:
```
#Creating a Spark Session
sc = SparkSession.builder.master("local[*]").getOrCreate()
```

In [ ]:
```
#Uploading the data file from https://www.kaggle.com/CooperUnion/cardataset
files.upload()
```

In [ ]:
```
#looking at the data file to see it uploaded correctly
!ls
```

In [ ]:
```
#Reading the datafile to make sure Spark uploaded it correctly
data = sc.read.csv('data.csv', inferSchema=True, header=True)
```

In [ ]:
```
#Printing the Schema and transposing the Panda's dataframe into statistical values
data.printSchema()
data.describe().toPandas().transpose()
```

In [ ]:
```
#Create function to replace all string N/A to NONE so Schema can read it
def replace(column, value):
  return when(column !=value, column).otherwise(lit(None))
data = data.withColumn("Market Category", replace(col("Market Category"), "N/A"))
```

In [ ]:
```
data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in data.columns]
```

In [ ]:
```
#Dropping Market Category as it has the most null values and looking at the data
data = data.drop('Market Category')
data = data.na.drop()
print((data.count(), len(data.columns)))
```

In [ ]:
```
#Building the pipeline for RandomForest with the 2 variables needed then write, overigh
assembler = VectorAssembler(inputCols=['Year', 'Engine HP', 'Engine Cylinders',
                                       'Number of Doors', 'highway MPG', 'city mpg', 'P
```

```
                            outputCol='Attributes')
regressor = RandomForestRegressor(featuresCol='Attributes', labelCol='MSRP')
pipeline = Pipeline(stages=[assembler, regressor])
pipeline.write().overwrite().save("pipeline")
!ls
```

In [ ]:
```
#Load the pipeline in new object called pipelinemodel
pipelineModel = Pipeline.load("pipeline")
#Establishing the grid
paramGrid = ParamGridBuilder() \
  .addGrid(regressor.numTrees, [100, 500]) \
  .build()
#Create the cross validation for Hyperparameter tuning
crossval = CrossValidator(estimator=pipelineModel,
                          estimatorParamMaps = paramGrid,
                          evaluator=RegressionEvaluator(labelCol='MSRP'),
                          numFolds=3)
```

In [ ]:
```
#Split the training and test set using random split using 80/20
train_data, test_data = data.randomSplit([0.8, 0.2], seed=123)
cvModel = crossval.fit(train_data)
```

In [ ]:
```
#Extracting best model and viwing all stages of the pipeline the data went through
bestModel = cvModel.bestModel
for x in range(len(bestModel.stages)):
  print(bestModel.stages[x])
```

In [ ]:
```
#Using the best model using Transform function
pred = cvModel.transform(test_data) #using cvModel object so it will use the best model
pred.select('MSRP', 'prediction').show()
```

In [ ]:
```
#Evaluating the Model's Performance
#Mean Square Error (MSE)
#Root Mean Square Error (RMSE)
#Mean Absolute Error (MAE)
#R squared R2
#Using Regression Model function
eval = RegressionEvaluator(labelCol='MSRP') #default metric is RMSE
rmse = eval.evaluate(pred)
mse = eval.evaluate(pred, {eval.metricName: "mse"}) #changing default metric to MSE
mae = eval.evaluate(pred, {eval.metricName: "mae"} ) #changing default metric to MAE
r2 = eval.evaluate(pred, {eval.metricName: "r2"} ) #changing default metric to R square

print("RMSE: %.3f" %rmse)
print("MSE: %.3f" %mse)
print("MAE: %.3f" %mae)
print("R2: %.3f" %r2)
```

In [ ]:
```
#Visualizing Feature Importance of Random Forest
bestModel.feature_importances_
```

In [ ]:

In [ ]: