Mercedes-Benz Greener Manufacturing

Course-end Project 1

DESCRIPTION

Reduce the time a Mercedes-Benz spends on the test bench.

Problem Statement Scenario: Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, the company's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.

Following actions should be performed:

If for any column(s), the variance is equal to zero, then you need to remove those variable(s). Check for null and unique values for test and train sets. Apply label encoder. Perform dimensionality reduction. Predict your test_df values using XGBoost. Find the datasets here.

```
# Step1: Import the required libraries

# linear algebra
import numpy as np
# data processing, CSV file I/O (e.g. pd.read_csv)
import pandas as pd
# for dimensionality reduction
from sklearn.decomposition import PCA

from google.colab import files
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>

Saving train.csv to train.csv
```

```
# Step2: Read the data from train.csv

df_train = pd.read_csv('train.csv')
# let us understand the data
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))
# print few rows and see how the data looks like
df_train.head()
```

```
Size of training set: 4209 rows and 378 columns

   ID       y  X0 X1  X2 X3 X4 X5 X6 X8  ...  X375  X376  X377  X378
X379  \
0   0  130.81   k  v  at  a  d  u  j  o  ...     0     0     1     0
0
1   6   88.53   k  t  av  e  d  y  l  o  ...     1     0     0     0
0
2   7   76.26  az  w   n  c  d  x  j  x  ...     0     0     0     0
0
3   9   80.62  az  t   n  f  d  x  l  e  ...     0     0     0     0
0
4  13   78.02  az  v   n  f  d  h  d  n  ...     0     0     0     0
0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 378 columns]
```

```
# Step3: Collect the Y values into an array

# seperate the y from the data as we will use this to learn as
# the prediction output
y_train = df_train['y'].values
```

```
# Step4: Understand the data types we have

# iterate through all the columns which has X in the name of the
column
cols = [c for c in df_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))
```

```
print('Feature types:')
df_train[cols].dtypes.value_counts()
```

```
Number of features: 376
Feature types:

int64     368
object      8
dtype: int64
```

```
# Step5: Count the data in each of the columns

counts = [[], [], []]
for c in cols:
    typ = df_train[c].dtype
    uniq = len(np.unique(df_train[c]))
    if uniq == 1:
        counts[0].append(c)
    elif uniq == 2 and typ == np.int64:
        counts[1].append(c)
    else:
        counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features:
{}\n'
      .format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268',
'X289', 'X290', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

```
from google.colab import files
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>

Saving test.csv to test (1).csv
```

```
# Step6: Read the test.csv data

df_test = pd.read_csv('test.csv')

# remove columns ID and Y from the data as they are not used for
learning
usable_columns = list(set(df_train.columns) - set(['ID', 'y']))
y_train = df_train['y'].values
id_test = df_test['ID'].values
```

```python
x_train = df_train[usable_columns]
x_test = df_test[usable_columns]
```

```python
# Step7: Check for null and unique values for test and train sets

def check_missing_values(df):
    if df.isnull().any().any():
        print("There are missing values in the dataframe")
    else:
        print("There are no missing values in the dataframe")
check_missing_values(x_train)
check_missing_values(x_test)
```

```
There are no missing values in the dataframe
There are no missing values in the dataframe
```

```python
# Step8: If for any column(s), the variance is equal to zero,
# then you need to remove those variable(s).
# Apply label encoder

for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality == 1:
        x_train.drop(column, axis=1) # Column with only one
        # value is useless so we drop it
        x_test.drop(column, axis=1)
    if cardinality > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train[column] = x_train[column].apply(mapper)
        x_test[column] = x_test[column].apply(mapper)
x_train.head()
```

```
<ipython-input-14-9fdc2c8730c7>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  x_train[column] = x_train[column].apply(mapper)
<ipython-input-14-9fdc2c8730c7>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  x_test[column] = x_test[column].apply(mapper)
```

```
   X327  X352  X60  X334   X2  X341  X42  X197  X95  X84  ...  X251  
X180  \
0     1     0    0     1  213     0    0     0    0    0  ...     0     
0   
1     0     0    0     0  215     0    0     0    0    0  ...     0     
0   
2     0     0    0     1  110     0    0     0    0    1  ...     0     
0   
3     0     0    0     0  110     0    0     0    0    1  ...     0     
0   
4     0     0    0     1  110     0    0     0    0    0  ...     0     
0   

   X156  X375   X4  X90  X181  X111  X270  X307  
0     1     0  100    0     0     1     0     0  
1     1     1  100    0     0     1     0     0  
2     0     0  100    0     0     1     0     0  
3     0     0  100    0     0     1     0     0  
4     0     0  100    0     0     1     0     0  

[5 rows x 376 columns]
```

```python
# Step9: Make sure the data is now changed into numericals

print('Feature types:')
x_train[cols].dtypes.value_counts()
```

```
Feature types:

int64    376
dtype: int64
```

```python
# Step10: Perform dimensionality reduction
# Linear dimensionality reduction using Singular Value Decomposition
of
# the data to project it to a lower dimensional space.
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)

# Step11: Training using xgboost

import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(
        pca2_results_train,
        y_train, test_size=0.2,
        random_state=4242)
```

```python
d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
#d_test = xgb.DMatrix(x_test)
d_test = xgb.DMatrix(pca2_results_test)

params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4

def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

clf = xgb.train(params, d_train,
                1000, watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[20:09:50] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
[0]	train-rmse:99.1484	valid-rmse:98.263	train-r2:-58.353
	valid-r2:-67.6375
Multiple eval metrics have been passed: 'valid-r2' will be used for
early stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]	train-rmse:81.2766	valid-rmse:80.3643	train-r2:-38.8843
	valid-r2:-44.9101
[20]	train-rmse:66.7161	valid-rmse:65.7733	train-r2:-25.874
	valid-r2:-29.7526
[30]	train-rmse:54.8692	valid-rmse:53.8912	train-r2:-17.1772
	valid-r2:-19.6451
[40]	train-rmse:45.2456	valid-rmse:44.2223	train-r2:-11.3602
	valid-r2:-12.9016
[50]	train-rmse:37.4474	valid-rmse:36.3776	train-r2:-7.46672
	valid-r2:-8.40697
[60]	train-rmse:31.1511	valid-rmse:30.0177	train-r2:-4.85891
	valid-r2:-5.40527
[70]	train-rmse:26.0869	valid-rmse:24.9072	train-r2:-3.10881
	valid-r2:-3.40994
[80]	train-rmse:22.0489	valid-rmse:20.8234	train-r2:-1.93526
	valid-r2:-2.08237
[90]	train-rmse:18.8543	valid-rmse:17.5971	train-r2:-1.14631
	valid-r2:-1.20123
[100]	train-rmse:16.342	valid-rmse:15.0822	train-r2:-0.612441
	valid-r2:-0.617009
[110]	train-rmse:14.4101	valid-rmse:13.1544	train-r2:-0.253738
```

```
       valid-r2:-0.230046
[120]  train-rmse:12.9391    valid-rmse:11.6979    train-r2:-0.010831
       valid-r2:0.027265
[130]  train-rmse:11.8303    valid-rmse:10.6328    train-r2:0.154986
       valid-r2:0.196323
[140]  train-rmse:10.9985    valid-rmse:9.86564    train-r2:0.269639
       valid-r2:0.308118
[150]  train-rmse:10.3966    valid-rmse:9.33551    train-r2:0.347392
       valid-r2:0.380476
[160]  train-rmse:9.94684    valid-rmse:8.97542    train-r2:0.402633
       valid-r2:0.427347
[170]  train-rmse:9.60982    valid-rmse:8.73192    train-r2:0.442427
       valid-r2:0.457997
[180]  train-rmse:9.36416    valid-rmse:8.57288    train-r2:0.470569
       valid-r2:0.477561
[190]  train-rmse:9.18013    valid-rmse:8.46885    train-r2:0.491175
       valid-r2:0.490164
[200]  train-rmse:9.03562    valid-rmse:8.40846    train-r2:0.507067
       valid-r2:0.497408
[210]  train-rmse:8.92586    valid-rmse:8.37003    train-r2:0.51897
       valid-r2:0.501992
[220]  train-rmse:8.84709    valid-rmse:8.34919    train-r2:0.527423
       valid-r2:0.504469
[230]  train-rmse:8.78618    valid-rmse:8.33786    train-r2:0.533909
       valid-r2:0.505813
[240]  train-rmse:8.73171    valid-rmse:8.3294     train-r2:0.539669
       valid-r2:0.506815
[250]  train-rmse:8.69122    valid-rmse:8.32544    train-r2:0.543928
       valid-r2:0.507284
[260]  train-rmse:8.64159    valid-rmse:8.32031    train-r2:0.549122
       valid-r2:0.507891
[270]  train-rmse:8.60658    valid-rmse:8.31549    train-r2:0.552768
       valid-r2:0.508461
[280]  train-rmse:8.57887    valid-rmse:8.31482    train-r2:0.555644
       valid-r2:0.508541
[290]  train-rmse:8.55738    valid-rmse:8.31495    train-r2:0.557867
       valid-r2:0.508525
[300]  train-rmse:8.53046    valid-rmse:8.31356    train-r2:0.560644
       valid-r2:0.508689
[310]  train-rmse:8.4958     valid-rmse:8.3082     train-r2:0.564207
       valid-r2:0.509323
[320]  train-rmse:8.47229    valid-rmse:8.3055     train-r2:0.566616
       valid-r2:0.509642
[330]  train-rmse:8.45016    valid-rmse:8.30588    train-r2:0.568877
       valid-r2:0.509596
[340]  train-rmse:8.4205     valid-rmse:8.30266    train-r2:0.571898
       valid-r2:0.509977
[350]  train-rmse:8.39591    valid-rmse:8.30062    train-r2:0.574395
       valid-r2:0.510218
[360]  train-rmse:8.37206    valid-rmse:8.29875    train-r2:0.57681
```

```
      valid-r2:0.510439
[370] train-rmse:8.34821    valid-rmse:8.297 train-r2:0.579216
      valid-r2:0.510645
[380] train-rmse:8.32289    valid-rmse:8.29276    train-r2:0.581765
      valid-r2:0.511145
[390] train-rmse:8.29903    valid-rmse:8.29151    train-r2:0.58416
      valid-r2:0.511293
[400] train-rmse:8.28126    valid-rmse:8.29096    train-r2:0.585938
      valid-r2:0.511357
[410] train-rmse:8.25654    valid-rmse:8.28965    train-r2:0.588408
      valid-r2:0.511511
[420] train-rmse:8.23701    valid-rmse:8.28729    train-r2:0.590352
      valid-r2:0.51179
[430] train-rmse:8.21037    valid-rmse:8.2863     train-r2:0.592998
      valid-r2:0.511907
[440] train-rmse:8.18911    valid-rmse:8.28825    train-r2:0.595103
      valid-r2:0.511677
[450] train-rmse:8.16221    valid-rmse:8.28829    train-r2:0.597759
      valid-r2:0.511672
[460] train-rmse:8.13467    valid-rmse:8.28829    train-r2:0.600468
      valid-r2:0.511672
[470] train-rmse:8.10329    valid-rmse:8.28776    train-r2:0.603544
      valid-r2:0.511734
Stopping. Best iteration:
[428] train-rmse:8.21419    valid-rmse:8.28557    train-r2:0.592619
      valid-r2:0.511992
```

```python
# Step12: Predict your test_df values using xgboost

p_test = clf.predict(d_test)

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('xgb.csv', index=False)

sub.head()
```

	ID	y
0	1	82.841324
1	2	97.597923
2	3	83.489418
3	4	77.218788
4	5	113.017982

Project Completed By : Santhosh TN.