Mercedes-Benz Greener Manufacturing

December 9, 2023

Mercedes-Benz Greener Manufacturing 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.

[1]:
```
# Step1: Import the required libraries
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
```

[2]:
```
# 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
```

[2]:
```
   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]
```

[3]:
```
# 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
```

[4]:
```
# 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:
```

[4]:
```
int64     368
object      8
dtype: int64
```

[5]:
```
# 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']
```

[6]:
```
# 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

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

[7]:
```
# 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
```

[8]:
```
# Step8: If for any column(s), the variance is equal to zero,
# then you need to remove those variable(s).
```

```
# Apply label encoder
import warnings
warnings.filterwarnings('ignore')
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()
```

[8]:

```
   X373  X372  X147  X76  X271  X265  X310  X49  X234  X140  …  X267  X113  \
0     0     0     0    0     0     0     0    0     1     0  …     0     0
1     0     0     0    0     0     1     0    0     0     0  …     0     0
2     0     0     0    1     0     0     0    0     0     0  …     0     0
3     0     1     0    1     0     0     0    0     0     0  …     0     0
4     0     0     0    1     0     0     0    0     0     0  …     0     0

   X281  X362  X225  X65  X245  X277  X137  X80
0     0     0     0    0     0     0     1    0
1     0     0     0    0     0     0     0    1
2     0     0     0    0     0     0     1    1
3     0     0     0    0     0     0     0    1
4     0     0     0    0     0     0     0    1

[5 rows x 376 columns]
```

[9]:

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

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

```
Feature types:
```

[9]:

```
int64    376
dtype: int64
```

[10]:

```
# 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)

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)

# 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()
```

```
[13:54:22] WARNING: ../src/objective/regression_obj.cu:203: reg:linear is now
deprecated in favor of reg:squarederror.
[0]     train-rmse:99.14834     train-r2:-58.35295      valid-rmse:98.26297
```

```
valid-r2:-67.63754
[10]    train-rmse:81.27653     train-r2:-38.88428      valid-rmse:80.36433
valid-r2:-44.91014
[20]    train-rmse:66.71610     train-r2:-25.87403      valid-rmse:65.77334
valid-r2:-29.75260
[30]    train-rmse:54.86957     train-r2:-17.17752      valid-rmse:53.88974
valid-r2:-19.64401
[40]    train-rmse:45.24492     train-r2:-11.35979      valid-rmse:44.21970
valid-r2:-12.89996
[50]    train-rmse:37.44729     train-r2:-7.46666       valid-rmse:36.37238
valid-r2:-8.40428
[60]    train-rmse:31.14748     train-r2:-4.85757       valid-rmse:30.01873
valid-r2:-5.40570
[70]    train-rmse:26.08660     train-r2:-3.10872       valid-rmse:24.90890
valid-r2:-3.41053
[80]    train-rmse:22.04638     train-r2:-1.93458       valid-rmse:20.83274
valid-r2:-2.08514
[90]    train-rmse:18.84403     train-r2:-1.14397       valid-rmse:17.60316
valid-r2:-1.20274
[100]   train-rmse:16.33631     train-r2:-0.61131       valid-rmse:15.08444
valid-r2:-0.61749
[110]   train-rmse:14.40372     train-r2:-0.25262       valid-rmse:13.14818
valid-r2:-0.22889
[120]   train-rmse:12.92869     train-r2:-0.00921       valid-rmse:11.68936
valid-r2:0.02868
[130]   train-rmse:11.80809     train-r2:0.15816        valid-rmse:10.61528
valid-r2:0.19898
[140]   train-rmse:10.98599     train-r2:0.27130        valid-rmse:9.84963
valid-r2:0.31036
[150]   train-rmse:10.37389     train-r2:0.35024        valid-rmse:9.32208
valid-r2:0.38226
[160]   train-rmse:9.92853      train-r2:0.40483        valid-rmse:8.96099
valid-r2:0.42919
[170]   train-rmse:9.59537      train-r2:0.44410        valid-rmse:8.71967
valid-r2:0.45952
[180]   train-rmse:9.35225      train-r2:0.47192        valid-rmse:8.55413
valid-r2:0.47984
[190]   train-rmse:9.16428      train-r2:0.49293        valid-rmse:8.45174
valid-r2:0.49222
[200]   train-rmse:9.02204      train-r2:0.50855        valid-rmse:8.38659
valid-r2:0.50002
[210]   train-rmse:8.91662      train-r2:0.51997        valid-rmse:8.35371
valid-r2:0.50393
[220]   train-rmse:8.83872      train-r2:0.52832        valid-rmse:8.33102
valid-r2:0.50662
[230]   train-rmse:8.77391      train-r2:0.53521        valid-rmse:8.31548
valid-r2:0.50846
[240]   train-rmse:8.72920      train-r2:0.53993        valid-rmse:8.30832
```

```
valid-r2:0.50931
[250]	train-rmse:8.68828	train-r2:0.54424	valid-rmse:8.30543
valid-r2:0.50965
[260]	train-rmse:8.65591	train-r2:0.54763	valid-rmse:8.30428
valid-r2:0.50979
[270]	train-rmse:8.62669	train-r2:0.55068	valid-rmse:8.30624
valid-r2:0.50955
[280]	train-rmse:8.60017	train-r2:0.55343	valid-rmse:8.30456
valid-r2:0.50975
[290]	train-rmse:8.57294	train-r2:0.55626	valid-rmse:8.30370
valid-r2:0.50985
[300]	train-rmse:8.54996	train-r2:0.55863	valid-rmse:8.30237
valid-r2:0.51001
[310]	train-rmse:8.52763	train-r2:0.56093	valid-rmse:8.30148
valid-r2:0.51012
[320]	train-rmse:8.50268	train-r2:0.56350	valid-rmse:8.29956
valid-r2:0.51034
[330]	train-rmse:8.47538	train-r2:0.56630	valid-rmse:8.29828
valid-r2:0.51049
[340]	train-rmse:8.44939	train-r2:0.56895	valid-rmse:8.29625
valid-r2:0.51073
[350]	train-rmse:8.42733	train-r2:0.57120	valid-rmse:8.29222
valid-r2:0.51121
[360]	train-rmse:8.40640	train-r2:0.57333	valid-rmse:8.29305
valid-r2:0.51111
[370]	train-rmse:8.38456	train-r2:0.57554	valid-rmse:8.29154
valid-r2:0.51129
[380]	train-rmse:8.36129	train-r2:0.57790	valid-rmse:8.28932
valid-r2:0.51155
[390]	train-rmse:8.33024	train-r2:0.58103	valid-rmse:8.28363
valid-r2:0.51222
[400]	train-rmse:8.30203	train-r2:0.58386	valid-rmse:8.27584
valid-r2:0.51314
[410]	train-rmse:8.27943	train-r2:0.58612	valid-rmse:8.27272
valid-r2:0.51350
[420]	train-rmse:8.24873	train-r2:0.58919	valid-rmse:8.27160
valid-r2:0.51364
[430]	train-rmse:8.22790	train-r2:0.59126	valid-rmse:8.27188
valid-r2:0.51360
[440]	train-rmse:8.20397	train-r2:0.59363	valid-rmse:8.27117
valid-r2:0.51369
[450]	train-rmse:8.17496	train-r2:0.59650	valid-rmse:8.27284
valid-r2:0.51349
[460]	train-rmse:8.15489	train-r2:0.59848	valid-rmse:8.27202
valid-r2:0.51359
[470]	train-rmse:8.12334	train-r2:0.60158	valid-rmse:8.26981
valid-r2:0.51385
[480]	train-rmse:8.10440	train-r2:0.60344	valid-rmse:8.26924
```

```
valid-r2:0.51391
[490]   train-rmse:8.08424      train-r2:0.60541        valid-rmse:8.26888
valid-r2:0.51396
[500]   train-rmse:8.06390      train-r2:0.60739        valid-rmse:8.26706
valid-r2:0.51417
[510]   train-rmse:8.03087      train-r2:0.61060        valid-rmse:8.26732
valid-r2:0.51414
[520]   train-rmse:8.00590      train-r2:0.61302        valid-rmse:8.26711
valid-r2:0.51416
[530]   train-rmse:7.98370      train-r2:0.61516        valid-rmse:8.26574
valid-r2:0.51433
[540]   train-rmse:7.96486      train-r2:0.61698        valid-rmse:8.26585
valid-r2:0.51431
[550]   train-rmse:7.94249      train-r2:0.61912        valid-rmse:8.26507
valid-r2:0.51440
[560]   train-rmse:7.92011      train-r2:0.62127        valid-rmse:8.26554
valid-r2:0.51435
[570]   train-rmse:7.90228      train-r2:0.62297        valid-rmse:8.26451
valid-r2:0.51447
[580]   train-rmse:7.87991      train-r2:0.62510        valid-rmse:8.26470
valid-r2:0.51445
[590]   train-rmse:7.85943      train-r2:0.62705        valid-rmse:8.26602
valid-r2:0.51429
```

[ ]:

```
#Finished
```