Mercedes-Benz Greener Manufacturing

July 18, 2020

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
import pandas as pd
from sklearn.decomposition import PCA
```

[2]:
```
# Step2: Read the data from train.csv
df_train = pd.read_csv('train.csv')
```

[3]:
```python
# let us understand the data
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))
```

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

[4]:
```python
df_train.head()
```

[4]:
```
   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]
```

[5]:
```python
# 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
```

[6]:
```python
# 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)))
```

```
Number of features: 376
```

[7]:
```python
print('Feature types:')
df_train[cols].dtypes.value_counts()
```

```
Feature types:
```

[7]:
```
int64     368
object      8
dtype: int64
```

[8]:
```
# 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)
```

[9]:
```
print('Constant features: {} Binary features: {} Categorical features: {}\n'
      .format(*[len(c) for c in counts]))
```

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

[10]:
```
print('Constant features:', counts[0])
```

```
Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289',
'X290', 'X293', 'X297', 'X330', 'X347']
```

[11]:
```
print('Categorical features:', counts[2])
```

```
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[12]:
```
# Step6: Read the test.csv data
df_test = pd.read_csv('test.csv')
```

[13]:
```
# 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
```

[14]:
```
x_train = df_train[usable_columns]
x_test = df_test[usable_columns]
```

[15]:
```
# 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
```

[16]:
```
# 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)
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py: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
  del sys.path[0]
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py: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
```

[17]:
```
x_train.head()
```

[17]:

	X194	X211	X54	X317	X89	X144	X122	X315	X127	X242	…	X280	X251	\
0	1	0	0	0	0	1	0	0	0	0	…	0	0	
1	1	0	0	0	0	1	0	0	1	0	…	0	0	
2	1	0	1	0	0	1	0	0	0	0	…	0	0	
3	1	0	1	0	0	1	0	0	0	0	…	0	0	
4	1	0	1	0	0	1	0	0	0	0	…	0	0	

X358 X333 X380 X319 X11 X115 X71 X230

```
0    0    0    0    0    0    0    0    0
1    0    0    0    0    0    0    0    0
2    1    0    0    0    0    0    1    0
3    1    0    0    0    0    0    1    0
4    1    0    0    0    0    0    0    0

[5 rows x 376 columns]
```

[18]:
```
# Step9: Make sure the data is now changed into numericals

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

```
Feature types:
```

[18]:
```
int64    376
dtype: int64
```

[19]:
```
# 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)
```

[20]:
```
# Step11: Training using xgboost

import xgboost as xgb
#from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

[21]:
```
x_train, x_valid, y_train, y_valid = train_test_split(
        pca2_results_train,
        y_train, test_size=0.2,
        random_state=4242)
```

[22]:
```
d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
```

[23]:
```
d_test = xgb.DMatrix(pca2_results_test)
```

[24]:
```
params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4
```

[25]:
```
def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

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

[26]:
```
clf = xgb.train(params, d_train,
                1000, watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[17:00:53] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linear
is now deprecated in favor of reg:squarederror.
[0]	train-rmse:99.14835	valid-rmse:98.26297	train-r2:-58.35295
valid-r2:-67.63754
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:81.27653	valid-rmse:80.36433	train-r2:-38.88428
valid-r2:-44.91014
[20]	train-rmse:66.71610	valid-rmse:65.77334	train-r2:-25.87403
valid-r2:-29.75260
[30]	train-rmse:54.86957	valid-rmse:53.88974	train-r2:-17.17752
valid-r2:-19.64401
[40]	train-rmse:45.24491	valid-rmse:44.21970	train-r2:-11.35979
valid-r2:-12.89996
[50]	train-rmse:37.44729	valid-rmse:36.37237	train-r2:-7.46666
valid-r2:-8.40428
[60]	train-rmse:31.14748	valid-rmse:30.01874	train-r2:-4.85757
valid-r2:-5.40570
[70]	train-rmse:26.08660	valid-rmse:24.90889	train-r2:-3.10872
valid-r2:-3.41053
[80]	train-rmse:22.04638	valid-rmse:20.83274	train-r2:-1.93458
valid-r2:-2.08514
[90]	train-rmse:18.84403	valid-rmse:17.60316	train-r2:-1.14397
valid-r2:-1.20274
[100]	train-rmse:16.33631	valid-rmse:15.08444	train-r2:-0.61131
valid-r2:-0.61749
[110]	train-rmse:14.40372	valid-rmse:13.14818	train-r2:-0.25262
valid-r2:-0.22889
[120]	train-rmse:12.92871	valid-rmse:11.68941	train-r2:-0.00921
valid-r2:0.02867
[130]	train-rmse:11.80812	valid-rmse:10.61535	train-r2:0.15815
valid-r2:0.19897
[140]	train-rmse:10.98603	valid-rmse:9.84998	train-r2:0.27129
valid-r2:0.31031
[150]	train-rmse:10.37399	valid-rmse:9.32204	train-r2:0.35023
```

```
valid-r2:0.38226
[160]	train-rmse:9.92031	valid-rmse:8.95919	train-r2:0.40581
valid-r2:0.42942
[170]	train-rmse:9.59074	valid-rmse:8.71396	train-r2:0.44464
valid-r2:0.46022
[180]	train-rmse:9.34336	valid-rmse:8.55559	train-r2:0.47292
valid-r2:0.47967
[190]	train-rmse:9.15816	valid-rmse:8.45149	train-r2:0.49361
valid-r2:0.49225
[200]	train-rmse:9.01375	valid-rmse:8.38981	train-r2:0.50945
valid-r2:0.49963
[210]	train-rmse:8.90230	valid-rmse:8.34348	train-r2:0.52151
valid-r2:0.50515
[220]	train-rmse:8.82531	valid-rmse:8.32075	train-r2:0.52975
valid-r2:0.50784
[230]	train-rmse:8.76746	valid-rmse:8.30670	train-r2:0.53589
valid-r2:0.50950
[240]	train-rmse:8.71689	valid-rmse:8.29998	train-r2:0.54123
valid-r2:0.51029
[250]	train-rmse:8.67718	valid-rmse:8.29160	train-r2:0.54540
valid-r2:0.51128
[260]	train-rmse:8.64381	valid-rmse:8.29092	train-r2:0.54889
valid-r2:0.51136
[270]	train-rmse:8.61463	valid-rmse:8.28517	train-r2:0.55193
valid-r2:0.51204
[280]	train-rmse:8.58311	valid-rmse:8.28490	train-r2:0.55520
valid-r2:0.51207
[290]	train-rmse:8.55391	valid-rmse:8.28413	train-r2:0.55823
valid-r2:0.51216
[300]	train-rmse:8.53239	valid-rmse:8.28459	train-r2:0.56044
valid-r2:0.51211
[310]	train-rmse:8.50149	valid-rmse:8.27928	train-r2:0.56362
valid-r2:0.51273
[320]	train-rmse:8.47670	valid-rmse:8.28208	train-r2:0.56617
valid-r2:0.51240
[330]	train-rmse:8.44919	valid-rmse:8.28049	train-r2:0.56898
valid-r2:0.51259
[340]	train-rmse:8.42588	valid-rmse:8.27964	train-r2:0.57135
valid-r2:0.51269
[350]	train-rmse:8.40057	valid-rmse:8.27450	train-r2:0.57392
valid-r2:0.51329
[360]	train-rmse:8.37867	valid-rmse:8.27492	train-r2:0.57614
valid-r2:0.51324
[370]	train-rmse:8.35279	valid-rmse:8.27231	train-r2:0.57876
valid-r2:0.51355
[380]	train-rmse:8.32472	valid-rmse:8.27000	train-r2:0.58158
valid-r2:0.51382
[390]	train-rmse:8.30042	valid-rmse:8.26915	train-r2:0.58402
```

```
valid-r2:0.51392
[400]   train-rmse:8.27459      valid-rmse:8.26357      train-r2:0.58661
valid-r2:0.51458
[410]   train-rmse:8.24758      valid-rmse:8.26041      train-r2:0.58930
valid-r2:0.51495
[420]   train-rmse:8.22015      valid-rmse:8.25880      train-r2:0.59203
valid-r2:0.51514
[430]   train-rmse:8.19520      valid-rmse:8.25946      train-r2:0.59450
valid-r2:0.51506
[440]   train-rmse:8.16508      valid-rmse:8.25651      train-r2:0.59748
valid-r2:0.51541
[450]   train-rmse:8.13492      valid-rmse:8.25754      train-r2:0.60044
valid-r2:0.51529
[460]   train-rmse:8.11542      valid-rmse:8.25609      train-r2:0.60236
valid-r2:0.51546
[470]   train-rmse:8.09273      valid-rmse:8.25339      train-r2:0.60458
valid-r2:0.51577
[480]   train-rmse:8.06888      valid-rmse:8.25512      train-r2:0.60690
valid-r2:0.51557
[490]   train-rmse:8.04720      valid-rmse:8.25369      train-r2:0.60901
valid-r2:0.51574
[500]   train-rmse:8.02014      valid-rmse:8.25349      train-r2:0.61164
valid-r2:0.51576
[510]   train-rmse:8.00181      valid-rmse:8.25385      train-r2:0.61341
valid-r2:0.51572
[520]   train-rmse:7.97879      valid-rmse:8.25200      train-r2:0.61563
valid-r2:0.51594
[530]   train-rmse:7.95944      valid-rmse:8.25450      train-r2:0.61750
valid-r2:0.51565
[540]   train-rmse:7.93501      valid-rmse:8.25315      train-r2:0.61984
valid-r2:0.51580
[550]   train-rmse:7.91364      valid-rmse:8.25487      train-r2:0.62189
valid-r2:0.51560
[560]   train-rmse:7.88721      valid-rmse:8.25441      train-r2:0.62441
valid-r2:0.51566
[570]   train-rmse:7.87277      valid-rmse:8.25539      train-r2:0.62578
valid-r2:0.51554
Stopping. Best iteration:
[521]   train-rmse:7.97743      valid-rmse:8.25188      train-r2:0.61576
valid-r2:0.51595
```

[27]:

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

p_test = clf.predict(d_test)

sub = pd.DataFrame()
```

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

sub.head()
```

[27]:

	ID	y
0	1	82.544060
1	2	97.454353
2	3	83.058586
3	4	76.981377
4	5	112.576813

[ ]:

```python

```