Mercedes-Benz Greener Manufacturing Course-end Project 1

October 16, 2023

1 Mercedes-Benz Greener Manufacturing

Course-end Project 1

Name-Nishant Dubey

Objective-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

[1]:
```
# Importing the required libraries

import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Importing the data

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[3]:
```
train.head()
```

[3]:
```
   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]
```

[4]:
```
test.head()
```

[4]:
```
   ID  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X375  X376  X377  X378  X379  X380  \
0   1  az  v   n  f  d  t  a  w    0  …     0     0     0     1     0     0
1   2   t  b  ai  a  d  b  g  y    0  …     0     0     1     0     0     0
2   3  az  v  as  f  d  a  j  j    0  …     0     0     0     1     0     0
3   4  az  l   n  f  d  z  l  n    0  …     0     0     0     1     0     0
4   5   w  s  as  c  d  y  i  m    0  …     1     0     0     0     0     0

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

[5 rows x 377 columns]
```

[5]:
```
train.columns
```

[5]:
```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

[6]:
```
test.columns
```

[6]:
```
Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=377)
```

[7]:
```
print('Size of training set: {} rows and {} columns'.format(*train.shape))
print('Size of testing set: {} rows and {} columns'.format(*test.shape))
```

```
Size of training set: 4209 rows and 378 columns
Size of testing set: 4209 rows and 377 columns
```

[8]:
```
# Collect the Y values into an array
y_train = train['y'].values
```

[9]:
```
y_train
```

[9]:
```
array([130.81,  88.53,  76.26, …, 109.22,  87.48, 110.85])
```

[10]:
```
# Understanding the data types:
cols = [c for c in train.columns if 'X' in c]
print('Number of faetures: {}'.format(len(cols)))
print('Feature types:')
train[cols].dtypes.value_counts()
```

```
Number of faetures: 376
Feature types:
```

[10]:
```
int64     368
object      8
dtype: int64
```

[11]:
```
# Count the data in each of the columns

counts = [[], [], []]
for c in cols:
    typ = train[c].dtype
    uniq = len(np.unique(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 feature: {} Categorical features: {}\n'
 .format(*[len(c) for c in counts]))
print('Constant features:',counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary feature: 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']
```

[12]:
```
# Splitting the data

usable_columns = list(set(train.columns) - set(['ID','y']))
y_train = train['y'].values
id_test = test['ID'].values
x_train = train[usable_columns]
x_test = test[usable_columns]
```

2 Checking for null values and unique values for train and test data

[13]:
```
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')
```

[14]:
```
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
```

3 Label Encoding the categorical values

[15]:
```
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()
```

[15]:
```
   X24  X356  X291  X274  X292  X324  X178  X325  X142  X308  …  X196  X107  \
0    0     0     0     0     0     1     0     0     1     0  …     0     0
1    0     0     0     0     0     0     1     0     1     0  …     0     0
2    0     0     0     1     0     1     0     0     0     0  …     0     0
3    0     0     1     0     0     0     0     0     1     0  …     0     0
4    0     0     0     0     0     0     0     0     0     0  …     0     0

   X385  X187  X311  X212  X217  X164  X355  X314
0     0     1     0     0     0     0     0     0
1     0     1     1     0     0     0     0     0
2     0     0     0     0     0     0     0     0
3     0     0     0     0     0     0     0     0
4     0     0     0     0     0     0     0     0

[5 rows x 376 columns]
```

[16]:
```
# Make sure the data is changed into numerical values
```

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

```
featurectypes:
```

[16]:
```
int64    376
dtype: int64
```

4 Perform Dimensionality Reduction

[17]:
```
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)
```

5 Training Using XGBoost

[18]:
```
# Training Using XGBoost

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

[19]:
```
x_train,x_val,y_train,y_val = train_test_split(pca2_results_train, y_train,␣
 ↪test_size=0.2, random_state=4242)
```

[20]:
```
d_train = xgb.DMatrix(x_train,label = y_train)
d_val = xgb.DMatrix(x_val,label = y_val)

# dtest = xgb.DMatrix(x_test)

d_test = xgb.DMatrix(pca2_results_test)
```

[21]:
```
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_val,'valid')]
clf = xgb.train(params, d_train, 1000, watchlist, early_stopping_rounds=50,
               feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[13:09:23] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "Objective" } are not used.

[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:54.86913	train-r2:-17.17722	valid-rmse:53.89147
valid-r2:-19.64534
[40]	train-rmse:45.24710	train-r2:-11.36098	valid-rmse:44.22334
valid-r2:-12.90225
[50]	train-rmse:37.44856	train-r2:-7.46723	valid-rmse:36.37638
valid-r2:-8.40634
[60]	train-rmse:31.14585	train-r2:-4.85695	valid-rmse:30.02279
valid-r2:-5.40743
[70]	train-rmse:26.08417	train-r2:-3.10795	valid-rmse:24.91516
valid-r2:-3.41275
[80]	train-rmse:22.04312	train-r2:-1.93371	valid-rmse:20.83299
valid-r2:-2.08521
[90]	train-rmse:18.84671	train-r2:-1.14458	valid-rmse:17.59846
valid-r2:-1.20156
[100]	train-rmse:16.33186	train-r2:-0.61043	valid-rmse:15.08617
valid-r2:-0.61786
[110]	train-rmse:14.39874	train-r2:-0.25176	valid-rmse:13.15521
valid-r2:-0.23020
[120]	train-rmse:12.92910	train-r2:-0.00927	valid-rmse:11.70051
valid-r2:0.02682
[130]	train-rmse:11.81536	train-r2:0.15712	valid-rmse:10.62244
valid-r2:0.19790
[140]	train-rmse:10.99099	train-r2:0.27063	valid-rmse:9.86019
valid-r2:0.30888
[150]	train-rmse:10.38667	train-r2:0.34863	valid-rmse:9.33123
valid-r2:0.38104
[160]	train-rmse:9.93418	train-r2:0.40415	valid-rmse:8.96192
valid-r2:0.42907
[170]	train-rmse:9.59640	train-r2:0.44398	valid-rmse:8.71810
valid-r2:0.45971
[180]	train-rmse:9.35220	train-r2:0.47192	valid-rmse:8.55750
valid-r2:0.47943
[190]	train-rmse:9.16592	train-r2:0.49275	valid-rmse:8.45262
valid-r2:0.49212
[200]	train-rmse:9.02357	train-r2:0.50838	valid-rmse:8.38960
valid-r2:0.49966
[210]	train-rmse:8.92419	train-r2:0.51915	valid-rmse:8.35118
valid-r2:0.50423
```

```
[220]   train-rmse:8.84149      train-r2:0.52802        valid-rmse:8.32911
valid-r2:0.50685
[230]   train-rmse:8.77383      train-r2:0.53522        valid-rmse:8.31164
valid-r2:0.50892
[240]   train-rmse:8.72642      train-r2:0.54023        valid-rmse:8.30160
valid-r2:0.51010
[250]   train-rmse:8.68650      train-r2:0.54442        valid-rmse:8.29958
valid-r2:0.51034
[260]   train-rmse:8.64705      train-r2:0.54855        valid-rmse:8.29340
valid-r2:0.51107
[270]   train-rmse:8.61922      train-r2:0.55145        valid-rmse:8.29457
valid-r2:0.51093
[280]   train-rmse:8.58611      train-r2:0.55489        valid-rmse:8.29251
valid-r2:0.51118
[290]   train-rmse:8.55652      train-r2:0.55796        valid-rmse:8.29217
valid-r2:0.51121
[300]   train-rmse:8.53319      train-r2:0.56036        valid-rmse:8.29254
valid-r2:0.51117
[310]   train-rmse:8.50784      train-r2:0.56297        valid-rmse:8.29156
valid-r2:0.51129
[320]   train-rmse:8.48199      train-r2:0.56562        valid-rmse:8.29100
valid-r2:0.51135
[330]   train-rmse:8.45003      train-r2:0.56889        valid-rmse:8.28928
valid-r2:0.51155
[340]   train-rmse:8.42263      train-r2:0.57168        valid-rmse:8.28797
valid-r2:0.51171
[350]   train-rmse:8.39358      train-r2:0.57463        valid-rmse:8.28693
valid-r2:0.51183
[360]   train-rmse:8.37163      train-r2:0.57685        valid-rmse:8.28655
valid-r2:0.51188
[370]   train-rmse:8.34326      train-r2:0.57972        valid-rmse:8.28542
valid-r2:0.51201
[380]   train-rmse:8.31805      train-r2:0.58225        valid-rmse:8.28393
valid-r2:0.51219
[390]   train-rmse:8.28994      train-r2:0.58507        valid-rmse:8.28216
valid-r2:0.51239
[400]   train-rmse:8.26600      train-r2:0.58746        valid-rmse:8.28089
valid-r2:0.51254
[410]   train-rmse:8.24563      train-r2:0.58949        valid-rmse:8.28154
valid-r2:0.51247
[420]   train-rmse:8.21978      train-r2:0.59206        valid-rmse:8.28123
valid-r2:0.51250
[430]   train-rmse:8.19649      train-r2:0.59437        valid-rmse:8.27975
valid-r2:0.51268
[440]   train-rmse:8.17680      train-r2:0.59632        valid-rmse:8.28042
valid-r2:0.51260
[450]   train-rmse:8.15583      train-r2:0.59839        valid-rmse:8.27997
valid-r2:0.51265
```

```
[460]	train-rmse:8.13170	train-r2:0.60076	valid-rmse:8.27872
valid-r2:0.51280
[470]	train-rmse:8.10759	train-r2:0.60312	valid-rmse:8.28004
valid-r2:0.51264
[480]	train-rmse:8.08873	train-r2:0.60497	valid-rmse:8.27962
valid-r2:0.51269
[490]	train-rmse:8.06167	train-r2:0.60761	valid-rmse:8.28014
valid-r2:0.51263
[500]	train-rmse:8.03613	train-r2:0.61009	valid-rmse:8.27783
valid-r2:0.51290
[510]	train-rmse:8.01630	train-r2:0.61201	valid-rmse:8.27981
valid-r2:0.51267
[520]	train-rmse:7.98437	train-r2:0.61510	valid-rmse:8.28011
valid-r2:0.51263
[530]	train-rmse:7.96313	train-r2:0.61714	valid-rmse:8.28034
valid-r2:0.51261
[540]	train-rmse:7.93430	train-r2:0.61991	valid-rmse:8.28030
valid-r2:0.51261
[550]	train-rmse:7.91141	train-r2:0.62210	valid-rmse:8.28270
valid-r2:0.51233
[554]	train-rmse:7.90739	train-r2:0.62248	valid-rmse:8.28240
valid-r2:0.51237
```

6 Predicting test_df using XGBoost

[22]:
```
p_test = clf.predict(d_test)
```

[23]:
```
sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('test_df.csv', index = False)
sub.head()
```

[23]:

	ID	y
0	1	83.397812
1	2	97.286064
2	3	83.171097
3	4	76.930611
4	5	112.544647

[ ]:
```
```