Project 1 - 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 the 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 cars are leaders 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, Daimler'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 Daimler'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 Daimler'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

Start of Project

Step 1: Importing the required libraries

In [25]:
```
# for linear algebra
import numpy as np

# for data processing - CSV file I/O
import pandas as pd

# for dimensionality reduction
from sklearn.decomposition import PCA

# for ignoring warnings
import warnings
warnings.filterwarnings('ignore')
```

Step 2: Reading the data from train.csv

In [26]:
```
df_train = pd.read_csv('train.csv')

# for understanding the data
print('Size of training set: {} rows and {} columns'.format(*df_train.shape))

# for printing few rows and see how the data looks like
df_train.head()
```

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

Out[26]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0

5 rows × 378 columns

Step 3: Collecting the Y values into an array

In [27]:
```
# for seperating y from the data as we will use this to learn as the prediction outpu
t
y_train = df_train['y'].values
```

Step 4: Understanding the data types we have

In [28]:
```
# for iterating 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:
```

Out[28]:
```
int64     368
object      8
dtype: int64
```

Step 5: Counting the data in each of the columns

In [29]:
```
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', 'X
293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

Step 6: Reading the test.csv data

In [30]:
```
df_test = pd.read_csv('test.csv')

# for removing 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]
```

Step 7: Checking for null and unique values in the test and train datasets

In [31]:
```
def check_missing_values(df):
    if df.isnull().any().any():
        print("\nThere are missing values in the dataframe")
    else:
        print("\nThere 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
```

Step 8: Checking if for any column(s), the variance is equal to zero, then need to remove those variable(s).

In [32]:
```
for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality == 1:
        x_train.drop(column, axis=1) # As column is with only one value hence drop it
        x_test.drop(column, axis=1)
    if cardinality > 2: # As 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()
```

Out[32]:

	X2	X272	X317	X113	X289	X196	X158	X281	X263	X195	...	X74	X262	X214	X49	X139	X7!
0	213	0	0	0	0	0	0	0	1	0	...	1	1	0	0	0	(
1	215	0	0	0	0	0	0	0	1	0	...	1	0	0	0	0	(
2	110	1	0	0	0	0	1	0	0	0	...	1	0	0	0	0	(
3	110	1	0	0	0	0	0	0	0	0	...	1	0	0	0	0	(
4	110	1	0	0	0	0	1	0	0	0	...	1	0	0	0	0	(

5 rows × 376 columns

Step 9: Making sure that data is now changed into numerical

In [33]:
```
print('Feature types:')

x_train[cols].dtypes.value_counts()
```

```
Feature types:
```

Out[33]:
```
int64    376
dtype: int64
```

Step 10: Performing dimensionality reduction, Linear Dimensionality Reduction by using Singular Value Decomposition for the data to project it to a lower dimensional space.

In [34]:
```
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)
```

Step 11: Training using XGboost

In [35]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in e:\programs\anaconda3\lib\site-packages
(1.2.1)
Requirement already satisfied: numpy in e:\programs\anaconda3\lib\site-packages (fro
m xgboost) (1.18.5)
Requirement already satisfied: scipy in e:\programs\anaconda3\lib\site-packages (fro
m xgboost) (1.5.0)
```

In [36]:

```python
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=0.3,
        random_state=422)

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)

# for d_test is used as 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)
```

```
[16:45:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.2.0/sr
c/objective/regression_obj.cu:174: reg:linear is now deprecated in favor of reg:squa
rederror.
[0]     train-rmse:98.90280     valid-rmse:99.15183     train-r2:-59.81084      vali
d-r2:-60.29545
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:81.06773     valid-rmse:81.34905     train-r2:-39.85638      vali
d-r2:-40.26023
[20]    train-rmse:66.53464     valid-rmse:66.84795     train-r2:-26.52069      vali
d-r2:-26.86139
[30]    train-rmse:54.70628     valid-rmse:55.05984     train-r2:-17.60537      vali
d-r2:-17.90152
[40]    train-rmse:45.09587     valid-rmse:45.47532     train-r2:-11.64263      vali
d-r2:-11.89372
[50]    train-rmse:37.30692     valid-rmse:37.71103     train-r2:-7.65252       vali
d-r2:-7.86673
[60]    train-rmse:31.00880     valid-rmse:31.44166     train-r2:-4.97770       vali
d-r2:-5.16365
[70]    train-rmse:25.94241     valid-rmse:26.40733     train-r2:-3.18393       vali
d-r2:-3.34786
[80]    train-rmse:21.89302     valid-rmse:22.38741     train-r2:-1.97972       vali
d-r2:-2.12489
[90]    train-rmse:18.68544     valid-rmse:19.22096     train-r2:-1.17055       vali
d-r2:-1.30344
[100]   train-rmse:16.16211     valid-rmse:16.75008     train-r2:-0.62390       vali
d-r2:-0.74929
[110]   train-rmse:14.21262     valid-rmse:14.85639     train-r2:-0.25577       vali
d-r2:-0.37611
[120]   train-rmse:12.72305     valid-rmse:13.42720     train-r2:-0.00634       vali
d-r2:-0.12408
[130]   train-rmse:11.60097     valid-rmse:12.37028     train-r2:0.16333        vali
d-r2:0.04592
[140]   train-rmse:10.76386     valid-rmse:11.59724     train-r2:0.27972        vali
d-r2:0.16144
[150]   train-rmse:10.14333     valid-rmse:11.03807     train-r2:0.36038        vali
d-r2:0.24035
[160]   train-rmse:9.68290      valid-rmse:10.64420     train-r2:0.41713        vali
d-r2:0.29360
[170]   train-rmse:9.36013      valid-rmse:10.35524     train-r2:0.45534        vali
d-r2:0.33143
[180]   train-rmse:9.11822      valid-rmse:10.16152     train-r2:0.48313        vali
d-r2:0.35621
[190]   train-rmse:8.94065      valid-rmse:10.02018     train-r2:0.50306        vali
d-r2:0.37400
[200]   train-rmse:8.78864      valid-rmse:9.92615      train-r2:0.51982        vali
d-r2:0.38569
[210]   train-rmse:8.66969      valid-rmse:9.85424      train-r2:0.53273        vali
d-r2:0.39456
[220]   train-rmse:8.56730      valid-rmse:9.80281      train-r2:0.54370        vali
d-r2:0.40086
[230]   train-rmse:8.48600      valid-rmse:9.76267      train-r2:0.55232        vali
d-r2:0.40576
[240]   train-rmse:8.39960      valid-rmse:9.73654      train-r2:0.56139        vali
d-r2:0.40893
[250]   train-rmse:8.33889      valid-rmse:9.71448      train-r2:0.56771        vali
d-r2:0.41161
[260]   train-rmse:8.27284      valid-rmse:9.69895      train-r2:0.57453        vali
d-r2:0.41349
[270]   train-rmse:8.23122      valid-rmse:9.68557      train-r2:0.57880        vali
d-r2:0.41511
[280]   train-rmse:8.19004      valid-rmse:9.67438      train-r2:0.58300        vali
```

```
d-r2:0.41646
[290]	train-rmse:8.15480	valid-rmse:9.66742	train-r2:0.58658	vali
d-r2:0.41730
[300]	train-rmse:8.11006	valid-rmse:9.66298	train-r2:0.59110	vali
d-r2:0.41783
[310]	train-rmse:8.07228	valid-rmse:9.65484	train-r2:0.59491	vali
d-r2:0.41881
[320]	train-rmse:8.04500	valid-rmse:9.64865	train-r2:0.59764	vali
d-r2:0.41956
[330]	train-rmse:7.99633	valid-rmse:9.64532	train-r2:0.60249	vali
d-r2:0.41996
[340]	train-rmse:7.96786	valid-rmse:9.64302	train-r2:0.60532	vali
d-r2:0.42023
[350]	train-rmse:7.93808	valid-rmse:9.64156	train-r2:0.60826	vali
d-r2:0.42041
[360]	train-rmse:7.90765	valid-rmse:9.64072	train-r2:0.61126	vali
d-r2:0.42051
[370]	train-rmse:7.86297	valid-rmse:9.63646	train-r2:0.61564	vali
d-r2:0.42102
[380]	train-rmse:7.83090	valid-rmse:9.63157	train-r2:0.61877	vali
d-r2:0.42161
[390]	train-rmse:7.78861	valid-rmse:9.63284	train-r2:0.62288	vali
d-r2:0.42146
[400]	train-rmse:7.74889	valid-rmse:9.63509	train-r2:0.62671	vali
d-r2:0.42119
[410]	train-rmse:7.71412	valid-rmse:9.63245	train-r2:0.63006	vali
d-r2:0.42151
[420]	train-rmse:7.68438	valid-rmse:9.63487	train-r2:0.63290	vali
d-r2:0.42121
[430]	train-rmse:7.64837	valid-rmse:9.63755	train-r2:0.63633	vali
d-r2:0.42089
Stopping. Best iteration:
[382]	train-rmse:7.82060	valid-rmse:9.63136	train-r2:0.61977	vali
d-r2:0.42163

[16:45:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.2.0/sr
c/objective/regression_obj.cu:174: reg:linear is now deprecated in favor of reg:squa
rederror.
```

Step 12: Predicting the test values using XGboost

In [37]:
```python
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()
```

Out[37]:

	ID	y
0	1	80.308640
1	2	93.417969
2	3	81.051048
3	4	76.639000
4	5	109.382736

End of Project