Machine Learning Project – Mercedes-Benz Manufacturing

Problem Statement Scenario

1. 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.

2. 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.

3. In this competition, challenge is to reduce the time that cars spend on the test bench, which consequently will decrease carbon dioxide emissions associated with the testing procedure. All required data for the Greener Manufacturing competition was provided by Mercedes-Benz. The dataset was collected from thousands of safety and reliability tests run on a variety of Mercedes vehicles.

4. The evaluation metric for the competition is the R^2 measure, known as the coefficient of determination.

Data Description

1. This dataset contains set of variables, each representing a custom feature in a Mercedes car

2. The ground truth is labeled 'y' and represents the time (in seconds) that the car took to pass testing for each variable.

3. File descriptions: Variables with letters are categorical. Variables with 0/1 are binary values.

4. train.csv — the training set

5. test.csv — the test set, you must predict the 'y' variable for the 'ID's in this file

Features of Data

Training data include 4209 rows, 378 features.

[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	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	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	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	0	0

1. ID: ID column of data which will be removed during data cleaning.

2. y: Target Variable. Min: 72.11 Max: 265.32 Std: 12.679 Mean: 100.67.

```
df_train['y'].describe()
```

```
count    4209.000000
mean      100.669318
std        12.679381
min        72.110000
25%        90.820000
50%        99.150000
75%       109.010000
max       265.320000
Name: y, dtype: float64
```

3. X0-X8: Categorical Data

```
[108]: cateCols
```

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

4. X9-X385: There are 12 features include one unique value which will be removed during the data cleaning.

```
[96]: oneUniqueCols
```

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

Data Cleaning

Before we start with XGBoost model, we need to do some clean on the original data and use PCA to select features.

Below steps will show data cleaning process how it works.

Step1: Remove ID & y from training dataset / Remove ID from Test dataset

3.1. Remove Non-Relevent Features like Y & ID

```
X_train= df_train.drop(['y','ID'], axis=1)
X_test = df_test.drop(['ID'], axis = 1)
y_label = df_train['y']
print('Training Set: {}'.format(X_train.shape))
print('Training Label: {}'.format(y_label.shape))
print('Test Set: {}'.format(X_test.shape))
```

Step2: Checking Missing Value in Train & Test Dataset

3.2. Checking Missing Value Or Not

```
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
```

Step3: Remove features with unique values in train & test dataset

3.3. Features With Unique Values Will Be Removed from Training Set & Test Set

```
cateCols = []
oneUniqueCols = []
for c in X_train.columns:
    cardinality = len(np.unique(X_train[c]))
    if cardinality == 1:
        oneUniqueCols.append(c)
    if X_train[c].dtype == 'object':
        cateCols.append(c)
```

```
oneUniqueCols
```

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

```
X_train_clean = X_train.drop(oneUniqueCols, axis = 1)
X_train_clean.head()
```

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	k	v	at	a	d	u	j	o	0	0	...	0	0	1	0	0	0	0	0	0	0
1	k	t	av	e	d	y	l	o	0	0	...	1	0	0	0	0	0	0	0	0	0
2	az	w	n	c	d	x	j	x	0	0	...	0	0	0	0	0	0	1	0	0	0

Step 4: Encode categorical data before implementing PCA

4.2. Encode Categorical Data

In order to implement PCA, Categorical data need to be encode

```
X_train_clean_encode = X_train_clean
X_test_clean_encode = X_test_clean
```

```
from sklearn.preprocessing import LabelEncoder
trainLabEncoder = LabelEncoder()
testLabEncoder = LabelEncoder()
```

```
for c in cateCols:
    X_train_clean_encode[c] = trainLabEncoder.fit_transform(X_train_clean_encode[c])
    X_test_clean_encode[c] = testLabEncoder.fit_transform(X_test_clean_encode[c])
```

```
X_train_clean_encode.dtypes.value_counts()
```

```
int64    364
dtype: int64
```

```
X_train_clean_encode.head()
```

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	32	23	17	0	3	24	9	14	0	0	...	0	0	1	0	0	0	0	0	0	0
1	32	21	19	4	3	28	11	14	0	0	...	1	0	0	0	0	0	0	0	0	0
2	20	24	34	2	3	27	9	23	0	0	...	0	0	0	0	0	0	1	0	0	0

Step 5: Implement PCA on Train & Test Dataset

4.3. PCA on Training & Test Data Set

```
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
```

```
trainPCA = pca.fit_transform(X_train_clean_encode)
trainPCA.shape
```

```
(4209, 12)
```

```
testPCA = pca.transform(X_test_clean_encode)
testPCA.shape
```

```
(4209, 12)
```

XGBoost Model Training

Since we have done the data cleaning and feature selection, now we are ready to train XGBoost model with train dataset and predict "y" with test dataset.

In order to train XGBoost model, we need to specify model parameters. After 1000 times round training, we can see

Train-RSME: 7.23772

Test-RSME: 8.00635

5.4. Specify XGB Parameters

```
params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.005
params['max_depth'] = 6
params['n_trees'] = 500
params['subsample'] = 0.5
params['eval_metric'] = 'rmse'
params['base_score'] = np.mean(y_label)
params['silent'] = 1
```

```
evallist = [(xgb_X_train_split, 'train'), (xgb_X_valid_split, 'valid')]
```

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

5.5. Train XGB Model

```
model = xgb.train(params,xgb_X_train_split,1000,evallist,feval=xgb_r2_score,maximize=True,verbose_eval=10)
```

```
[0]	train-rmse:12.845	valid-rmse:11.8462	train-r2:0.003816	valid-r2:0.002443
[10]	train-rmse:12.5915	valid-rmse:11.5659	train-r2:0.042754	valid-r2:0.049089
[20]	train-rmse:12.3529	valid-rmse:11.3078	train-r2:0.078681	valid-r2:0.091054
[30]	train-rmse:12.1355	valid-rmse:11.0665	train-r2:0.110832	valid-r2:0.129436
[40]	train-rmse:11.9269	valid-rmse:10.8405	train-r2:0.141132	valid-r2:0.164621
[50]	train-rmse:11.7325	valid-rmse:10.637	train-r2:0.168903	valid-r2:0.195699
[60]	train-rmse:11.553	valid-rmse:10.4452	train-r2:0.194138	valid-r2:0.224435
[70]	train-rmse:11.3856	valid-rmse:10.2716	train-r2:0.217324	valid-r2:0.250012
[80]	train-rmse:11.2277	valid-rmse:10.0996	train-r2:0.238875	valid-r2:0.274919
[90]	train-rmse:11.0785	valid-rmse:9.94822	train-r2:0.258975	valid-r2:0.296486
[100]	train-rmse:10.938	valid-rmse:9.80937	train-r2:0.277647	valid-r2:0.315987
```

From the training result, we can say model is performing nice and not over-fitting. We will use this trained model to predict “y” value with test dataset.

```
r2_score(y_valid_split,model.predict(xgb.DMatrix(X_valid_split)))
```

```
0.5443294860369894
```

```
[990]	train-rmse:7.25269	valid-rmse:8.00748	train-r2:0.682408	valid-r2:0.544201
[999]	train-rmse:7.23772	valid-rmse:8.00635	train-r2:0.683718	valid-r2:0.544329
```

5.6. Predict with Test set

```
xgb_X_test = xgb.DMatrix(testPCA)
```

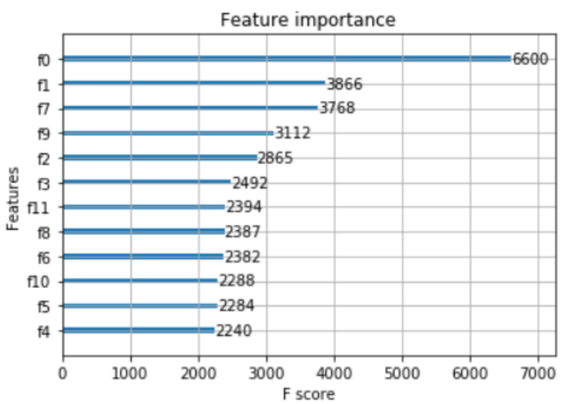
```
prediectY = model.predict(xgb_X_test)
prediectY
```

```
array([ 79.10548,  94.82104,  81.91955, ..., 100.73042, 109.27812,
        95.47911], dtype=float32)
```

5.7. Plot Importance Features

```
xgb.plot_importance(model)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb97578fdd8>
```



Observations

1. Categorical features X0 & X1 are highly important in the prediction of our XGBoost model.
2. With PCA feature selections, it is contributing effectively in the prediction.
3. In order to increase model production effectivity, we can drop features are less important