

Project - 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

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

```
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
print(train_data.shape)
print(test_data.shape)
```

```
(4209, 378)
```

```
(4209, 377)
```

```

for i in train_data.columns:
    data_type = train_data[i].dtype
    if data_type == 'object':
        print(i)

```

X0
X1
X2
X3
X4
X5
X6
X8

If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

```

variance = pow(train_data.drop(columns={'ID', 'y'}).std(),2).to_dict()

null_cnt = 0
for key, value in variance.items():
    if(value==0):
        print('Name = ',key)
        null_cnt = null_cnt+1
print('No of columns which has zero variance = ',null_cnt)

```

Name = X11
Name = X93
Name = X107
Name = X233
Name = X235
Name = X268
Name = X289
Name = X290
Name = X293
Name = X297
Name = X330
Name = X347
No of columns which has zero variance = 12

```

train_data = train_data.drop(columns={'X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X
train_data.shape

```

(4209, 366)

Check for null and unique values for test and train sets

```
train_data.isnull().sum().any()
```

False

Apply label encoder

```
from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()
```

```
train_data_feature = train_data.drop(columns={'y', 'ID'})  
train_data_target = train_data.y  
print(train_data_feature.shape)  
print(train_data_target.shape)
```

(4209, 364)

(4209,)

```
train_data_feature.describe(include='object')
```

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	47	27	44	7	4	29	12	25
top	z	aa	as	c	d	w	g	j
freq	360	833	1659	1942	4205	231	1042	277

```
train_data_feature['X0'] = le.fit_transform(train_data_feature.X0)  
train_data_feature['X1'] = le.fit_transform(train_data_feature.X1)  
train_data_feature['X2'] = le.fit_transform(train_data_feature.X2)  
train_data_feature['X3'] = le.fit_transform(train_data_feature.X3)  
train_data_feature['X4'] = le.fit_transform(train_data_feature.X4)  
train_data_feature['X5'] = le.fit_transform(train_data_feature.X5)  
train_data_feature['X6'] = le.fit_transform(train_data_feature.X6)  
train_data_feature['X8'] = le.fit_transform(train_data_feature.X8)
```

Perform dimensionality reduction.

```
print(train_data_feature.shape)  
print(train_data_target.shape)
```

(4209, 364)

(4209,)

```
from sklearn.decomposition import PCA  
pca = PCA(n_components=.95)
```

```
pca.fit(train_data_feature, train_data_target)
```

```
PCA(n_components=0.95)
```

```
train_data_feature_trans = pca.fit_transform(train_data_feature)
print(train_data_feature_trans.shape)
```

```
(4209, 6)
```

Predict your test_df values using XGBoost

Building model using the train data set.

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
```

```
train_x, test_x, train_y, test_y = train_test_split(train_data_feature_trans, train_data_target,
                                                    print(train_x.shape)
                                                    print(train_y.shape)
                                                    print(test_x.shape)
                                                    print(test_y.shape)
```

```
(2946, 6)
```

```
(2946,)
```

```
(1263, 6)
```

```
(1263,)
```

XGBoost's hyperparameters tuning manually

```
xgb_reg = xgb.XGBRegressor(objective='reg:linear', colsample_bytree = 0.3, learning_rate=0.1,
                           n_estimators = 20)
model = xgb_reg.fit(train_x, train_y)
print('RMSE = ', sqrt(mean_squared_error(model.predict(test_x), test_y)))
```

```
[22:43:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/objective/regression_obj.cu:170: reg:linear is now deprecated in favor of reg:squarederror.
```

```
RMSE = 12.288794806074309
```

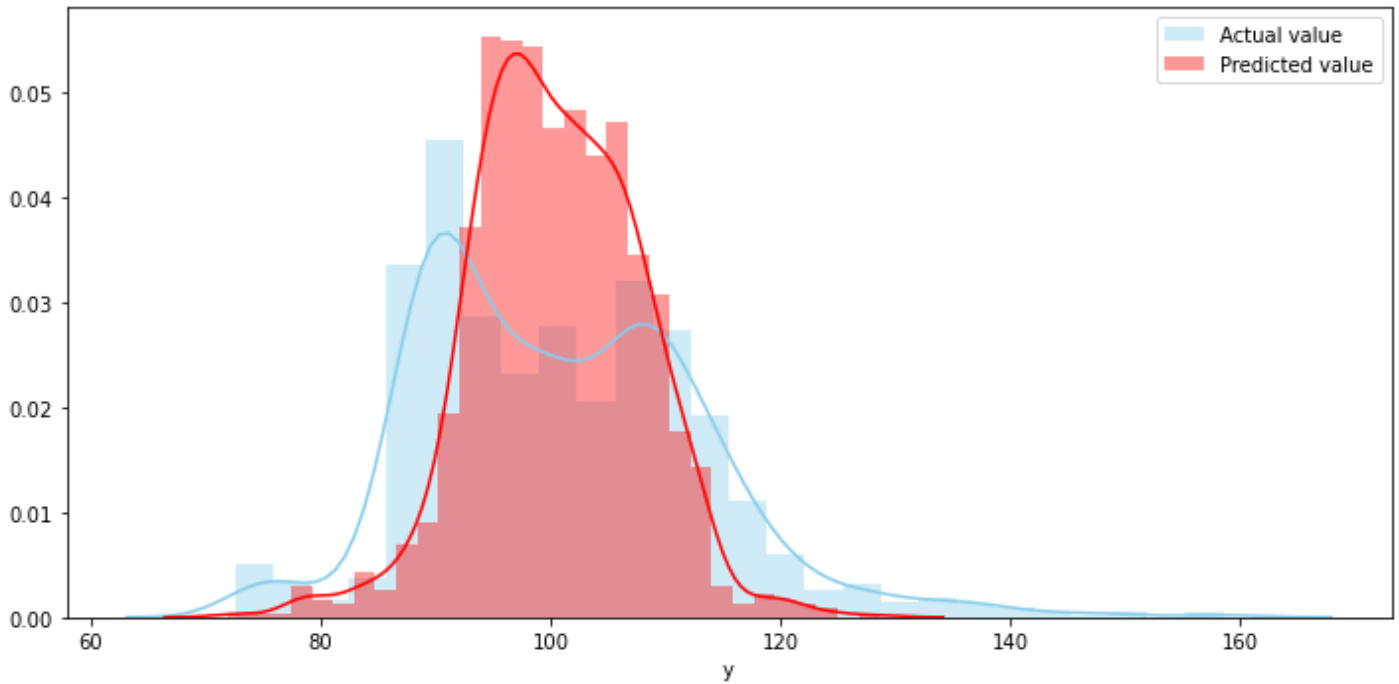
- After tuning the hyperparameters to meet minimum RMSE, RMSE turned out as 12.29

```
pred_test_y = model.predict(test_x)
```

```
plt.figure(figsize=(10,5))
```

```
sns.distplot(test_y[test_y<160], color="skyblue", label="Actual value")
sns.distplot(pred_test_y[pred_test_y<160], color="red", label="Predicted value")
plt.legend()
```

```
plt.tight_layout()
```



k-fold Cross Validation using XGBoost

```
dmatrix_train = xgb.DMatrix(data=train_data_feature_trans,label=train_data_target)

params = {'objective':'reg:linear', 'colsample_bytree': 0.3, 'learning_rate': 0.3, 'max_depth': 6}

model_cv = xgb.cv(dtrain=dmatrix_train, params=params, nfold=3, num_boost_round=50, early_stopping_rounds=10,
                  metrics="rmse", as_pandas=True, seed=7)

model_cv.tail(4)
```

[22:43:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/objective/regression_obj.cu:170: reg:linear is now deprecated in favor of reg:squarederror.

[22:43:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/objective/regression_obj.cu:170: reg:linear is now deprecated in favor of reg:squarederror.

[22:43:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/objective/regression_obj.cu:170: reg:linear is now deprecated in favor of reg:squarederror.

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
31	8.935206	0.183408	11.060047	0.736219
32	8.880285	0.174860	11.044372	0.740168
33	8.849044	0.185327	11.049080	0.738351
34	8.792401	0.202136	11.043289	0.728256

- However, using k-fold cross validation, RMSE comes as 11.04. So the RMSE reduced by ~ 10%

Prediction on test data set using XGBoost

Preparing test data set.

```
test_data = test_data.drop(columns={'X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X29'})
test_data.shape
```

(4209, 365)

```
test_data.isnull().sum().any()
```

False

```
test_data_feature = test_data.drop(columns={'ID'})
print(test_data_feature.shape)
```

(4209, 364)

```
test_data_feature.describe(include='object')
```

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	49	27	45	7	4	32	12	25
top	ak	aa	as	c	d	v	g	e
freq	432	826	1658	1900	4203	246	1073	274

```
test_data_feature['X0'] = le.fit_transform(test_data_feature.X0)
test_data_feature['X1'] = le.fit_transform(test_data_feature.X1)
test_data_feature['X2'] = le.fit_transform(test_data_feature.X2)
test_data_feature['X3'] = le.fit_transform(test_data_feature.X3)
test_data_feature['X4'] = le.fit_transform(test_data_feature.X4)
test_data_feature['X5'] = le.fit_transform(test_data_feature.X5)
test_data_feature['X6'] = le.fit_transform(test_data_feature.X6)
test_data_feature['X8'] = le.fit_transform(test_data_feature.X8)
```

```
pca.fit(test_data_feature)
```

PCA(n_components=0.95)

```
test_data_feature_trans = pca.fit_transform(test_data_feature)
print(test_data_feature_trans.shape)
```

(4209, 6)

```
test_pred = model.predict(test_data_feature_trans)
test_pred
```

array([86.12015 , 92.929794, 98.74635 , ..., 92.836525, 118.76457 ,

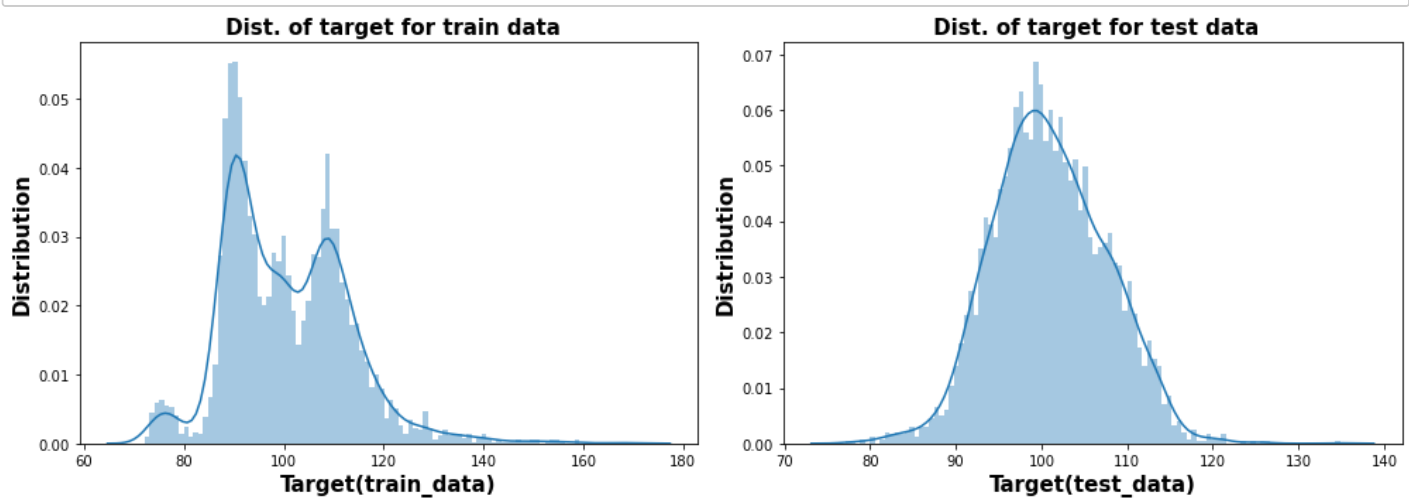
```
98.46741 ], dtype=float32)
```

```
fig, ax = plt.subplots(1,2, figsize=(14,5))

train_plot = sns.distplot(train_data_target[train_data_target<200], bins=100, kde=True,
train_plot.set_xlabel('Target(train_data)', weight='bold', size=15)
train_plot.set_ylabel('Distribution', weight='bold', size=15)
train_plot.set_title(' Dist. of target for train data', weight='bold', size=15)

test_plot = sns.distplot(test_pred[test_pred<200], bins=100, kde=True, ax=ax[1])
test_plot.set_xlabel('Target(test_data)', weight='bold', size=15)
test_plot.set_ylabel('Distribution', weight='bold', size=15)
test_plot.set_title(' Dist. of target for test data', weight='bold', size=15)

plt.tight_layout()
```



This is a pictorial view for comparison between the target for training data-set and predicted target for testing data-set.