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:

1. If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
2. Check for null and unique values for test and train sets.
3. Apply label encoder.
4. Perform dimensionality reduction.
5. Predict your test_df values using XGBoost

In [1]:

```
# 1.Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
# 2.Load Data

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)
```

In [4]:

```
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).

In [5]:

```
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
```

In [6]:

```
train_data = train_data.drop(columns={'X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X330','X347'})
train_data.shape
```

Out[6]:

```
(4209, 366)
```

Check for null and unique values for test and train sets

In [7]:

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

Out[7]:

```
False
```

Apply label encoder

In [8]:

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

In [9]:

```
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,)
```

In [10]:

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

Out[10]:

	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

In [11]:

```
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.

In [12]:

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

```
(4209, 364)
(4209,)
```

In [13]:

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

In [14]:

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

Out[14]:

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

In [15]:

```
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.

In [16]:

```
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
```

In [17]:

```
train_x,test_x,train_y,test_y = train_test_split(train_data_feature_trans,train_data_targe
t,test_size=.3,random_state=7)
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

In [18]:

```
xgb_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate
= 0.4, max_depth = 10, alpha = 6,
                           n_estimators = 20)
model = xgb_reg.fit(train_x,train_y)
print('RMSE = ',sqrt(mean_squared_error(model.predict(test_x),test_y)))
```

```
[15:05:25] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linea
r is now deprecated in favor of reg:squarederror.
RMSE =  12.237860466379919
```

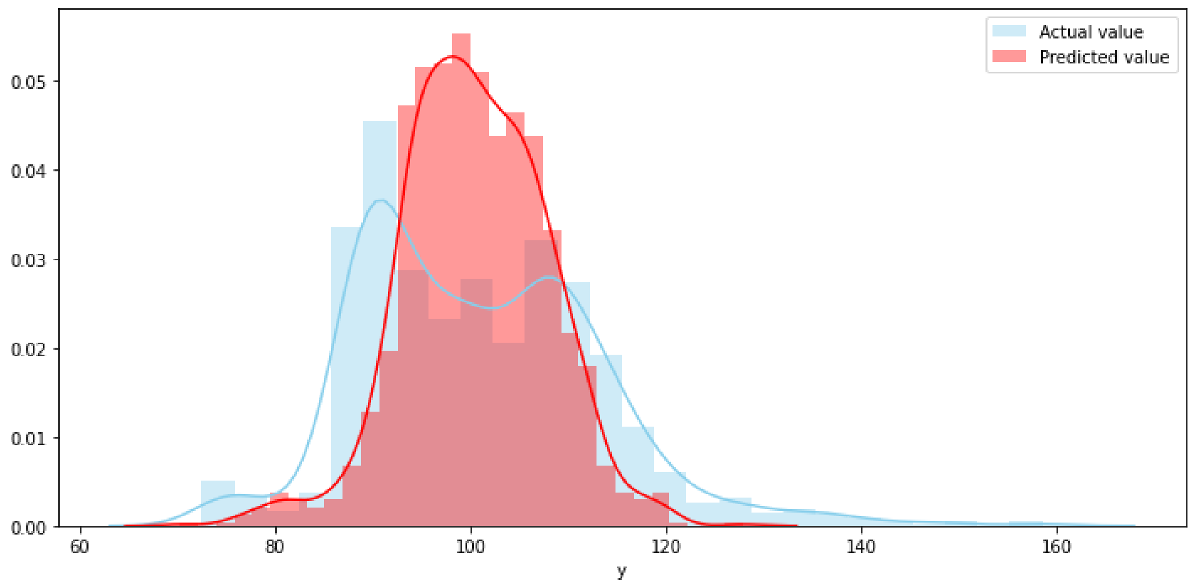
In [19]:

```
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

In [20]:

```
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_de
pth': 5, 'alpha': 10}

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)
```

```
[15:05:59] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linea
r is now deprecated in favor of reg:squarederror.
[15:05:59] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linea
r is now deprecated in favor of reg:squarederror.
[15:05:59] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linea
r is now deprecated in favor of reg:squarederror.
```

Out[20]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
26	9.218258	0.160322	11.033188	0.773833
27	9.169047	0.166877	11.030471	0.766387
28	9.085136	0.150843	11.034508	0.751673
29	9.045591	0.134536	11.028953	0.758330

Prediction on test data set using XGBoost

Preparing test data set.

In [21]:

```
test_data = test_data.drop(columns={'X11','X93','X107','X233','X235','X268','X289','X290',
'X293','X297','X330','X347'})
test_data.shape
```

Out[21]:

```
(4209, 365)
```

In [22]:

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

Out[22]:

```
False
```

In [23]:

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

```
(4209, 364)
```

In [24]:

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

Out[24]:

	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

In [25]:

```
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)
```

In [26]:

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

Out[26]:

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

In [27]:

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

```
(4209, 6)
```

In [28]:

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

Out[28]:

```
array([ 80.20003 ,  92.969574, 101.64637 , ...,  95.79453 , 110.24724 ,
        98.970345], dtype=float32)
```

In [29]:

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

train_plot = sns.distplot(train_data_target[train_data_target<200], bins=100, kde=True, ax
=ax[0])
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()
```