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.
```

Loading Datasets

In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
```

In [ ]:
```
train_df.info()
```

In [ ]:
```
train_df.head()
```

In [ ]:
```
test_df.head()
```

In [ ]:
```
print(train_df.shape)
print(test_df.shape)
```

In [ ]:
```
train_y = train_df['y']
train_df = train_df.drop(['ID', 'y'], axis = 1)
test_df = test_df.drop('ID', axis = 1)
```

In [ ]:
```
train_df.head()
```

In [ ]:
```
test_df.head()
```

Removing variable having variance 0

In [ ]:
```
for i in train_df.columns:
    if train_df[i].dtype != object:
        if train_df[i].var() == 0:
            train_df = train_df.drop(i, axis = 1)
            test_df = test_df.drop(i, axis = 1)
total_df = pd.concat([train_df, test_df])
```

In [ ]:
```
total_df
```

In [ ]:
```
train_df.head()
```

In [ ]:
```
test_df
```

Check for null and unique values for test and train sets.

In [ ]:
```
print(train_df.columns[train_df.isnull().any()])
train_df[train_df.isnull().any(axis = 1)]
```

In [ ]:
```
print(test_df.columns[test_df.isnull().any()])
test_df[test_df.isnull().any(axis = 1)]
```

Apply label encoder.

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [ ]:
```
total_df
```

In [ ]:
```
for i in total_df.columns:
    if total_df[i].dtype == object:
        total_df[i] = le.fit_transform(total_df[i])
```

In [ ]:
```
total_df
```

In [ ]:
```
train_df = total_df[:len(train_df)]
test_df = total_df[len(train_df):]
```

In [ ]:
```
train_df
```

In [ ]:
```
test_df
```

Dimensonality reduction

Using PCA for dimensionality reduction

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

In [ ]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(total_df)
```

In [ ]:
```
X_scaled.shape
```

In [ ]:
```
pca = PCA()
pca.fit(X_scaled)
```

In [ ]:
```
pca.explained_variance_ratio_
```

In [ ]:
```
plt.plot(['PC'+ str(i) for i in range(364)],pca.explained_variance_ratio_)
```

In [ ]:
```
pca.explained_variance_ratio_.cumsum()
```

In [ ]:
```
plt.bar(['PC' + str(i) for i in range(364)],pca.explained_variance_ratio_.cumsum())
```

In [ ]:
```
pca = PCA(n_components = 0.90)
```

In [ ]:
```
PCA_X = pca.fit_transform(X_scaled)
```

In [ ]:
```
print(PCA_X.shape)
```

In [ ]:
```
PCA_df = pd.DataFrame(PCA_X, columns = ['PC'+ str(i) for i in range(123)])
```

In [ ]:
```
PCA_df
```

In [ ]:
```
train_df = PCA_X[:len(train_df)]
test_df = PCA_X[len(train_df):]
```

In [ ]:
```
train_y
```

Applying XGBoost algorithm

In [ ]:
```
from xgboost import XGBRegressor
xgb = XGBRegressor()
xgb.fit(train_df, train_y)
print('Train acc : ', xgb.score(train_df, train_y))
predected_y = xgb.predict(test_df)
print('Predicted target variable using test_df : ',predected_y)
```