MercedesBenz-GM

March 26, 2022

0.1 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 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.

0.1.1 Importing required libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
from math import sqrt
warnings.filterwarnings('ignore')
```

0.1.2 Importing training and testing data

[2]:
```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
print("Train dataset:", train_df.shape)
print("Test dataset:", test_df.shape)
```

```
Train dataset: (4209, 378)
Test dataset: (4209, 377)
```

[3]:
```
train_df.head()
```

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

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 378 columns]
```

[4]:
```
cols = [c for c in train_df.columns if 'X' in c]
print("Number of features:", len(cols))
```

```
Number of features: 376
```

[5]:
```
train_df.dtypes.unique()
```

[5]:
```
array([dtype('int64'), dtype('float64'), dtype('O')], dtype=object)
```

[6]:
```
object_cols = []
int_cols = []
float_cols = []
other_cols = []

for i in train_df.columns:
    dtype = train_df[i].dtype
    if dtype == 'object':
        object_cols.append(i)
    elif dtype == 'int64':
```

```
        int_cols.append(i)
    elif dtype == 'float64':
        float_cols.append(i)

print('Object columns:\n', object_cols)
print('Integer coloums:\n', int_cols)
print('Float columns:\n', float_cols)
```

```
Object columns:
 ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
Integer coloums:
 ['ID', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
'X20', 'X21', 'X22', 'X23', 'X24', 'X26', 'X27', 'X28', 'X29', 'X30', 'X31',
'X32', 'X33', 'X34', 'X35', 'X36', 'X37', 'X38', 'X39', 'X40', 'X41', 'X42',
'X43', 'X44', 'X45', 'X46', 'X47', 'X48', 'X49', 'X50', 'X51', 'X52', 'X53',
'X54', 'X55', 'X56', 'X57', 'X58', 'X59', 'X60', 'X61', 'X62', 'X63', 'X64',
'X65', 'X66', 'X67', 'X68', 'X69', 'X70', 'X71', 'X73', 'X74', 'X75', 'X76',
'X77', 'X78', 'X79', 'X80', 'X81', 'X82', 'X83', 'X84', 'X85', 'X86', 'X87',
'X88', 'X89', 'X90', 'X91', 'X92', 'X93', 'X94', 'X95', 'X96', 'X97', 'X98',
'X99', 'X100', 'X101', 'X102', 'X103', 'X104', 'X105', 'X106', 'X107', 'X108',
'X109', 'X110', 'X111', 'X112', 'X113', 'X114', 'X115', 'X116', 'X117', 'X118',
'X119', 'X120', 'X122', 'X123', 'X124', 'X125', 'X126', 'X127', 'X128', 'X129',
'X130', 'X131', 'X132', 'X133', 'X134', 'X135', 'X136', 'X137', 'X138', 'X139',
'X140', 'X141', 'X142', 'X143', 'X144', 'X145', 'X146', 'X147', 'X148', 'X150',
'X151', 'X152', 'X153', 'X154', 'X155', 'X156', 'X157', 'X158', 'X159', 'X160',
'X161', 'X162', 'X163', 'X164', 'X165', 'X166', 'X167', 'X168', 'X169', 'X170',
'X171', 'X172', 'X173', 'X174', 'X175', 'X176', 'X177', 'X178', 'X179', 'X180',
'X181', 'X182', 'X183', 'X184', 'X185', 'X186', 'X187', 'X189', 'X190', 'X191',
'X192', 'X194', 'X195', 'X196', 'X197', 'X198', 'X199', 'X200', 'X201', 'X202',
'X203', 'X204', 'X205', 'X206', 'X207', 'X208', 'X209', 'X210', 'X211', 'X212',
'X213', 'X214', 'X215', 'X216', 'X217', 'X218', 'X219', 'X220', 'X221', 'X222',
'X223', 'X224', 'X225', 'X226', 'X227', 'X228', 'X229', 'X230', 'X231', 'X232',
'X233', 'X234', 'X235', 'X236', 'X237', 'X238', 'X239', 'X240', 'X241', 'X242',
'X243', 'X244', 'X245', 'X246', 'X247', 'X248', 'X249', 'X250', 'X251', 'X252',
'X253', 'X254', 'X255', 'X256', 'X257', 'X258', 'X259', 'X260', 'X261', 'X262',
'X263', 'X264', 'X265', 'X266', 'X267', 'X268', 'X269', 'X270', 'X271', 'X272',
'X273', 'X274', 'X275', 'X276', 'X277', 'X278', 'X279', 'X280', 'X281', 'X282',
'X283', 'X284', 'X285', 'X286', 'X287', 'X288', 'X289', 'X290', 'X291', 'X292',
'X293', 'X294', 'X295', 'X296', 'X297', 'X298', 'X299', 'X300', 'X301', 'X302',
'X304', 'X305', 'X306', 'X307', 'X308', 'X309', 'X310', 'X311', 'X312', 'X313',
'X314', 'X315', 'X316', 'X317', 'X318', 'X319', 'X320', 'X321', 'X322', 'X323',
'X324', 'X325', 'X326', 'X327', 'X328', 'X329', 'X330', 'X331', 'X332', 'X333',
'X334', 'X335', 'X336', 'X337', 'X338', 'X339', 'X340', 'X341', 'X342', 'X343',
'X344', 'X345', 'X346', 'X347', 'X348', 'X349', 'X350', 'X351', 'X352', 'X353',
'X354', 'X355', 'X356', 'X357', 'X358', 'X359', 'X360', 'X361', 'X362', 'X363',
'X364', 'X365', 'X366', 'X367', 'X368', 'X369', 'X370', 'X371', 'X372', 'X373',
'X374', 'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
```

'X385']
Float columns:
 ['y']

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

[7]:
```
variance = pow(train_df.drop(columns=['ID','y']).std(),2).to_dict()
zero_var_cols = []
for col, var in variance.items():
    if var == 0:
        zero_var_cols.append(col)

print("Columns with zero variance are:\n",zero_var_cols)
```

```
Columns with zero variance are:
 ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297',
'X330', 'X347']
```

0.1.4 Dropping zero variance columns from training data

[8]:
```
train_df_new = train_df.drop(columns=zero_var_cols)
print(train_df.shape)
print(train_df_new.shape)
```

```
(4209, 378)
(4209, 366)
```

0.1.5 Check for null and unique values

[9]:
```
train_df_new.isna().sum().any()
```

[9]: False

0.1.6 Applying label encoder for object cols

[10]:
```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
```

[11]:
```
train_df_new.describe(include='object')
```

[11]:

	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
```

[12]:
```python
train_features  = train_df_new.drop(columns=['y','ID'])
train_target = train_df.y

print(train_features.shape)
print(train_target.shape)
```

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

[13]:
```python
for col in object_cols:
    train_features[col] = enc.fit_transform(train_df_new[col])
```

[14]:
```python
train_features.head()
```

[14]:
```
   X0  X1  X2  X3  X4  X5  X6  X8  X10  X12  …  X375  X376  X377  X378  \
0  32  23  17   0   3  24   9  14    0    0  …     0     0     1     0
1  32  21  19   4   3  28  11  14    0    0  …     1     0     0     0
2  20  24  34   2   3  27   9  23    0    0  …     0     0     0     0
3  20  21  34   5   3  27  11   4    0    0  …     0     0     0     0
4  20  23  34   5   3  12   3  13    0    0  …     0     0     0     0

   X379  X380  X382  X383  X384  X385
0     0     0     0     0     0     0
1     0     0     0     0     0     0
2     0     0     1     0     0     0
3     0     0     0     0     0     0
4     0     0     0     0     0     0

[5 rows x 364 columns]
```

[15]:
```python
print(train_features.shape)
print(train_target.shape)
```

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

0.1.7 Performing dimensionality reduction

[16]:
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
```

[17]:
```python
pca.fit(train_features, train_target)
```

[17]: PCA(n_components=0.95)

[18]:
```
trans_train_features = pca.fit_transform(train_features ,train_target)
print(trans_train_features.shape)
```

```
(4209, 6)
```

0.1.8 Importing XGBoost and initializing train and test values

[19]:
```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

[20]:
```
X_train, X_test, y_train, y_test = train_test_split(trans_train_features,␣
 ↪train_target, test_size=0.3, random_state=42)
print('''
X train shape: {} \n
X test shape: {} \n
y train shape: {} \n
y test shape: {} \n
'''.format(X_train.shape, X_test.shape, y_train.shape, y_test.shape))
```

```
X train shape: (2946, 6)

X test shape: (1263, 6)

y train shape: (2946,)

y test shape: (1263,)
```

0.1.9 Tuning hyper parameters

[21]:
```
xgb = XGBRegressor(random_state=42, n_jobs=-1)
parameters = {'nthread': [4],
              'objective':['reg:linear'],
              'learning_rate':[0.01, 0.03, 0.05, 0.07, 0.1],
              'colsample_bytree':[0.1, 0.5, 0.7],
              'gamma': [0, 0.1, 0.01, 0.5, 1],
              'max_depth': [2, 3, 5, 10],
              'n_estimators':[30,50,100,200,500]
             }
```

```
xgb_gridsearch = GridSearchCV(xgb, parameters, cv=2, n_jobs=5)

xgb_gridsearch.fit(X_train, y_train)

print(xgb_gridsearch.best_score_)
print(xgb_gridsearch.best_params_)
print(xgb_gridsearch.best_estimator_)
```

```
[09:53:46] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.0/src/objective/regression_obj.cu:188: reg:linear is now
deprecated in favor of reg:squarederror.
0.3492438984265359
{'colsample_bytree': 0.7, 'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 10,
'n_estimators': 50, 'nthread': 4, 'objective': 'reg:linear'}
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False,
             gamma=0.1, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=10, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=50, n_jobs=-1, nthread=4,
             num_parallel_tree=1, objective='reg:linear', predictor='auto',
             random_state=42, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

[22]:
```
best_model = xgb_gridsearch.best_estimator_
y_pred = best_model.predict(X_test)
print('RMSE = ',sqrt(mean_squared_error(y_pred,y_test)))
```

```
RMSE =  11.219143649071343
```

[24]:
```
test_df = test_df.drop(columns=zero_var_cols)
test_df.shape
```

[24]: (4209, 365)

[25]:
```
test_df.isnull().sum().any()
```

[25]: False

[26]:
```
test_df_features = test_df.drop(columns={'ID'})
print(test_df_features.shape)
```

```
(4209, 364)
```

[27]:
```
for col in object_cols:
    test_df_features[col] = enc.fit_transform(test_df_features[col])
```

[28]:
```
test_df_features.head()
```

[28]:
```
   X0  X1  X2  X3  X4  X5  X6  X8  X10  X12  …  X375  X376  X377  X378  \
0  21  23  34   5   3  26   0  22    0    0  …     0     0     0     1
1  42   3   8   0   3   9   6  24    0    0  …     0     0     1     0
2  21  23  17   5   3   0   9   9    0    0  …     0     0     0     1
3  21  13  34   5   3  31  11  13    0    0  …     0     0     0     1
4  45  20  17   2   3  30   8  12    0    0  …     1     0     0     0

   X379  X380  X382  X383  X384  X385
0     0     0     0     0     0     0
1     0     0     0     0     0     0
2     0     0     0     0     0     0
3     0     0     0     0     0     0
4     0     0     0     0     0     0

[5 rows x 364 columns]
```

[29]:
```
pca.fit(test_df_features)
trans_test_features = pca.fit_transform(test_df_features)
print(test_df_features.shape)
```

```
(4209, 364)
```

0.1.10 Making predictions on test data

[30]:
```
test_preds = best_model.predict(trans_test_features)
test_preds
```

[30]:
```
array([ 73.61008 ,  96.798004,  89.871086, …, 103.09486 , 104.47703 ,
        95.45212 ], dtype=float32)
```