Machine Learning

Project: Mercedes-Benz Greener Manufacturing

Alhamouieh Dima

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.

[2]:
```
# Project: Mercedes-Benz Greener Manufacturing
```

[3]:
```
# Step1 : Import the required Librairies
```

53]:
```
# Linear algebra
import numpy as np
```

54]:
```
# data processing
import pandas as pd
```

55]:
```
# for dimensionality reduction
from sklearn.decomposition import PCA
```

56]:
```
# Step2: Read the data from train.csv
df_train = pd.read_csv("C:\\Users\\Dima\\OneDrive\\Desktop\\train.csv")
```

57]:
```
# Let us understand the data
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))
```

```
Size of training set: 4209 rows and 378 columns
```

58]:
```
# Let's print few rows and see how the data Looks Like
df_train.head()
```

58]:

	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

5 rows × 378 columns

[59]:
```
# Step3: Collect the Y values into an array
# Seperate the y from the data as we will use this to Learn as the prediction output

y_train = df_train['y'].values
```

[60]:
```
# Step4 : Understand the data types we have iterate through all the columns which has X in the name column
cols = [c for c in df_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))
```

```
Number of features: 376
```

[61]:
```
print('Feature types:')
df_train[cols].dtypes.value_counts()
```

```
Feature types:
```

[61]:
```
int64     368
object      8
dtype: int64
```

[62]:
```
# Step5: Count the data in each of the columns

counts = [[], [], []]
for c in cols:
    typ = df_train[c].dtype
    uniq = len(np.unique(df_train[c]))
    if uniq == 1:
        counts[0].append(c)
    elif uniq == 2 and typ == np.int64:
        counts[1].append(c)
    else:
        counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'
      .format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[64]:
```
# Step6: Read the test.csv data

df_test = pd.read_csv("C:\\Users\\Dima\\OneDrive\\Desktop\\test.csv")

# remove columns ID and Y from the data as they are not used for Learning

usable_columns = list(set(df_train.columns) - set(['ID', 'y']))
y_train = df_train['y'].values
id_test = df_test['ID'].values

x_train = df_train[usable_columns]
x_test = df_test[usable_columns]
```

[65]:

```
# Step7: Check for null and unique values for test and train sets

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
```

[66]:

```
# Step8: If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
# Apply label encoder

for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality == 1:
        x_train.drop(column, axis=1) # Column with only one
        # value is useless so we drop it
        x_test.drop(column, axis=1)
    if cardinality > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train[column] = x_train[column].apply(mapper)
        x_test[column] = x_test[column].apply(mapper)
x_train.head()
```

```
<ipython-input-66-1f63d7e649bc>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x_train[column] = x_train[column].apply(mapper)
<ipython-input-66-1f63d7e649bc>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x_test[column] = x_test[column].apply(mapper)
```

[66]:

	X93	X179	X160	X277	X91	X352	X383	X122	X58	X42	...	X10	X118	X227	X92	X341	X273	X190	X351	X89	X181
0	0	1	0	0	0	0	0	0	1	0	...	0	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	1	0	0	0	0
2	0	1	0	0	0	0	0	0	1	0	...	0	0	0	0	0	1	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0

5 rows × 376 columns

[67]:

```
# Step9: Make sure the data is now changed into numericals

print('Feature types:')
x_train[cols].dtypes.value_counts()
```

```
Feature types:
```

[67]:

```
int64    376
dtype: int64
```

[68]:

```
# Step10: Perform dimensionality reduction
# Linear dimensionality reduction using Singular Value Decomposition of
# the data to project it to a lower dimensional space.
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)
```

[69]:

```
# Step11: Training using xgboost

import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(
        pca2_results_train,
        y_train, test_size=0.2,
        random_state=4242)

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
#d_test = xgb.DMatrix(x_test)
d_test = xgb.DMatrix(pca2_results_test)

params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4

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

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

clf = xgb.train(params, d_train,
                1000, watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[10:56:43] 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.
[0]	train-rmse:99.14835	train-r2:-58.35295	valid-rmse:98.26297	valid-r2:-67.63754
[10]	train-rmse:81.27653	train-r2:-38.88428	valid-rmse:80.36433	valid-r2:-44.91014
[20]	train-rmse:66.71610	train-r2:-25.87403	valid-rmse:65.77334	valid-r2:-29.75260
[30]	train-rmse:54.86956	train-r2:-17.17751	valid-rmse:53.88963	valid-r2:-19.64393
[40]	train-rmse:45.24492	train-r2:-11.35979	valid-rmse:44.21995	valid-r2:-12.90012
[50]	train-rmse:37.44736	train-r2:-7.46669	valid-rmse:36.37245	valid-r2:-8.40431
[60]	train-rmse:31.14762	train-r2:-4.85762	valid-rmse:30.01881	valid-r2:-5.40573
[70]	train-rmse:26.08681	train-r2:-3.10878	valid-rmse:24.90893	valid-r2:-3.41054
[80]	train-rmse:22.04670	train-r2:-1.93467	valid-rmse:20.83095	valid-r2:-2.08461
[90]	train-rmse:18.84431	train-r2:-1.14403	valid-rmse:17.60402	valid-r2:-1.20296
[100]	train-rmse:16.34085	train-r2:-0.61221	valid-rmse:15.08339	valid-r2:-0.61726
[110]	train-rmse:14.40636	train-r2:-0.25308	valid-rmse:13.15278	valid-r2:-0.22975
[120]	train-rmse:12.92814	train-r2:-0.00912	valid-rmse:11.69885	valid-r2:0.02710
[130]	train-rmse:11.81469	train-r2:0.15722	valid-rmse:10.62024	valid-r2:0.19823
[140]	train-rmse:10.98928	train-r2:0.27086	valid-rmse:9.85276	valid-r2:0.30992
[150]	train-rmse:10.38152	train-r2:0.34928	valid-rmse:9.31839	valid-r2:0.38275
[160]	train-rmse:9.92876	train-r2:0.40480	valid-rmse:8.95283	valid-r2:0.43023
[170]	train-rmse:9.59912	train-r2:0.44367	valid-rmse:8.70703	valid-r2:0.46108
[180]	train-rmse:9.35429	train-r2:0.47169	valid-rmse:8.54489	valid-r2:0.48097
[190]	train-rmse:9.16741	train-r2:0.49258	valid-rmse:8.44206	valid-r2:0.49338
[200]	train-rmse:9.01908	train-r2:0.50887	valid-rmse:8.37807	valid-r2:0.50104
[210]	train-rmse:8.91005	train-r2:0.52067	valid-rmse:8.33802	valid-r2:0.50579
[220]	train-rmse:8.83257	train-r2:0.52897	valid-rmse:8.31548	valid-r2:0.50846
[230]	train-rmse:8.76183	train-r2:0.53649	valid-rmse:8.29701	valid-r2:0.51064
[240]	train-rmse:8.71715	train-r2:0.54120	valid-rmse:8.29061	valid-r2:0.51140
[250]	train-rmse:8.67319	train-r2:0.54582	valid-rmse:8.28605	valid-r2:0.51194
[260]	train-rmse:8.64135	train-r2:0.54915	valid-rmse:8.27979	valid-r2:0.51267
[270]	train-rmse:8.60642	train-r2:0.55278	valid-rmse:8.27898	valid-r2:0.51277
[280]	train-rmse:8.57676	train-r2:0.55586	valid-rmse:8.27733	valid-r2:0.51296
[290]	train-rmse:8.54857	train-r2:0.55878	valid-rmse:8.27936	valid-r2:0.51272
[300]	train-rmse:8.52241	train-r2:0.56147	valid-rmse:8.27807	valid-r2:0.51287
[310]	train-rmse:8.49832	train-r2:0.56395	valid-rmse:8.27651	valid-r2:0.51306
[320]	train-rmse:8.47707	train-r2:0.56613	valid-rmse:8.27503	valid-r2:0.51323
[330]	train-rmse:8.44755	train-r2:0.56914	valid-rmse:8.27171	valid-r2:0.51362
[340]	train-rmse:8.42575	train-r2:0.57137	valid-rmse:8.26946	valid-r2:0.51389
[350]	train-rmse:8.40558	train-r2:0.57341	valid-rmse:8.27014	valid-r2:0.51381
[360]	train-rmse:8.37834	train-r2:0.57617	valid-rmse:8.26694	valid-r2:0.51418
[370]	train-rmse:8.34955	train-r2:0.57908	valid-rmse:8.26552	valid-r2:0.51435
[380]	train-rmse:8.32922	train-r2:0.58113	valid-rmse:8.26521	valid-r2:0.51439
[390]	train-rmse:8.30088	train-r2:0.58398	valid-rmse:8.26410	valid-r2:0.51452
[400]	train-rmse:8.27501	train-r2:0.58656	valid-rmse:8.26422	valid-r2:0.51450
[410]	train-rmse:8.24785	train-r2:0.58927	valid-rmse:8.26112	valid-r2:0.51487
[420]	train-rmse:8.22787	train-r2:0.59126	valid-rmse:8.26387	valid-r2:0.51455
[430]	train-rmse:8.20595	train-r2:0.59344	valid-rmse:8.26271	valid-r2:0.51468
[440]	train-rmse:8.18438	train-r2:0.59557	valid-rmse:8.26104	valid-r2:0.51488
[450]	train-rmse:8.15946	train-r2:0.59803	valid-rmse:8.26066	valid-r2:0.51492
[460]	train-rmse:8.13683	train-r2:0.60026	valid-rmse:8.25829	valid-r2:0.51520
[470]	train-rmse:8.11442	train-r2:0.60245	valid-rmse:8.25823	valid-r2:0.51521
[480]	train-rmse:8.08928	train-r2:0.60491	valid-rmse:8.26072	valid-r2:0.51492
[490]	train-rmse:8.06864	train-r2:0.60693	valid-rmse:8.26237	valid-r2:0.51472
[500]	train-rmse:8.04565	train-r2:0.60916	valid-rmse:8.25921	valid-r2:0.51509
[510]	train-rmse:8.02235	train-r2:0.61142	valid-rmse:8.25921	valid-r2:0.51509
[513]	train-rmse:8.01403	train-r2:0.61223	valid-rmse:8.26093	valid-r2:0.51489
```

]:

```python
# Step12: Predict your test_df values using xgboost

p_test = clf.predict(d_test)

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('xgb.csv', index=False)

sub.head()
```

]:

	ID	y
0	1	83.005119
1	2	97.758469
2	3	82.942375
3	4	76.957230
4	5	112.741913