Project_Solution

Mercedes-Benz Greener Manufacturing

[1]:
```
# Importing the required libraries
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
```

[2]:
```
# Importing the data

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[3]:
```
train.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]:
```
test.head()
```

[4]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	\
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	

```
4   5   w   s   as   c   d   y   i   m   0  ...    1    0    0    0    0    0

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

[5 rows x 377 columns]
```

[5]:
```
print("Size of training set: {} rows and {} columns".format(*train.shape))
print("Size of testing set: {} rows and {} columns".format(*test.shape))
```

```
Size of training set: 4209 rows and 378 columns
Size of testing set: 4209 rows and 377 columns
```

[6]:
```
# Collect the Y values into an array
y_train = train["y"].values
```

[7]:
```
y_train
```

[7] :
```
array([130.81,  88.53,  76.26, ..., 109.22,  87.48, 110.85])
```

[8]:
```
# Understand the data types
cols = [c for c in train.columns if "X" in c]
print("Number of features: {}".format(len(cols)))
print("Feature types:")
train[cols].dtypes.value_counts()
```

```
Number of features: 376
Feature types:
```

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

[9]:
```
# Count the data in each of the columns

counts = [[], [], []]
for c in cols:
    typ = train[c].dtype
    uniq = len(np.unique(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']

[10]:
```
# Splitting the data
usable_columns = list(set(train.columns) - set(["ID", "y"]))
y_train = train["y"].values
id_test = test["ID"].values
x_train = train[usable_columns]
x_test = test[usable_columns]
```

Check for null values and unique values for train & test data

[11]:
```
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")
```

[12]:
```
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

Label Encoding the categorical values

[13]:
```
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()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:9:
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
  if __name__ == '__main__':
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:10:
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
  # Remove the CWD from sys.path while we load stuff.
```

[13]:

	X39	X265	X246	X237	X373	X308	X62	X143	X296	X267	...	X15	X243	\
0	0	0	0	1	0	0	0	0	0	0	...	0	0	
1	0	1	0	0	0	0	0	0	0	0	...	0	0	
2	0	0	1	0	0	0	0	0	0	0	...	0	0	
3	0	0	1	0	0	0	0	0	0	0	...	0	0	
4	0	0	1	0	0	0	0	0	0	0	...	0	0	

	X311	X200	X37	X78	X97	X179	X174	X371
0	0	0	1	0	0	1	0	0
1	1	0	1	0	0	0	0	0
2	0	0	1	0	0	1	0	0
3	0	0	1	0	0	1	1	0
4	0	0	1	0	0	1	0	1

[5 rows x 376 columns]

[14]:
```
# Make sure the data is changed into numerical values

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

```
Feature types:
```

[14]:
```
int64    376
dtype: int64
```

1.0.1 Perform Dimensionality reduction

[15]:
```
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)
```

Training using XGBoost

[16]:
```
# Training using XGBoost

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

[17]:
```
x_train,x_val,y_train,y_val = train_test_split(pca2_results_train, y_train,␣
↪test_size=0.2, random_state=4242)
```

[18]:
```
d_train = xgb.DMatrix(x_train,label = y_train)
d_val = xgb.DMatrix(x_val,label = y_val)

# dtest = xgb.DMatrix(x_test)

d_test = xgb.DMatrix(pca2_results_test)
```

[19]:
```
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_val, 'valid')]
clf = xgb.train(params, d_train, 1000, watchlist, early_stopping_rounds=50,
 feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[15:03:21] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linear
is now deprecated in favor of reg:squarederror.
[0]     train-rmse:99.14835     valid-rmse:98.26297     train-r2:-58.35295
valid-r2:-67.63754
Multiple eval metrics have been passed: 'valid-r2' will be used for early
stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]    train-rmse:81.27653     valid-rmse:80.36433     train-r2:-38.88428
valid-r2:-44.91014
```

[20] train-rmse:66.71610 valid-rmse:65.77334 train-r2:-25.87403 valid-r2:-29.75260
[30] train-rmse:54.86915 valid-rmse:53.89120 train-r2:-17.17724 valid-r2:-19.64513
[40] train-rmse:45.24564 valid-rmse:44.22231 train-r2:-11.36018 valid-r2:-12.90160
[50] train-rmse:37.44742 valid-rmse:36.37758 train-r2:-7.46672 valid-r2:-8.40697
[60] train-rmse:31.15105 valid-rmse:30.01771 train-r2:-4.85891 valid-r2:-5.40526
[70] train-rmse:26.08769 valid-rmse:24.90855 train-r2:-3.10906 valid-r2:-3.41041
[80] train-rmse:22.04899 valid-rmse:20.82566 train-r2:-1.93528 valid-r2:-2.08304
[90] train-rmse:18.84732 valid-rmse:17.59580 train-r2:-1.14472 valid-r2:-1.20090
[100] train-rmse:16.33602 valid-rmse:15.07912 train-r2:-0.61125 valid-r2:-0.61635
[110] train-rmse:14.40459 valid-rmse:13.14868 train-r2:-0.25278 valid-r2:-0.22898
[120] train-rmse:12.93437 valid-rmse:11.68702 train-r2:-0.01009 valid-r2:0.02907
[130] train-rmse:11.81328 valid-rmse:10.60818 train-r2:0.15742 valid-r2:0.20005
[140] train-rmse:10.98910 valid-rmse:9.84164 train-r2:0.27089 valid-r2:0.31148
[150] train-rmse:10.38670 valid-rmse:9.31149 train-r2:0.34863 valid-r2:0.38366
[160] train-rmse:9.93406 valid-rmse:8.95125 train-r2:0.40417 valid-r2:0.43043
[170] train-rmse:9.60179 valid-rmse:8.70644 train-r2:0.44336 valid-r2:0.46116
[180] train-rmse:9.35700 valid-rmse:8.54541 train-r2:0.47138 valid-r2:0.48090
[190] train-rmse:9.17218 valid-rmse:8.44794 train-r2:0.49206 valid-r2:0.49268
[200] train-rmse:9.02681 valid-rmse:8.38293 train-r2:0.50803 valid-r2:0.50046
[210] train-rmse:8.92066 valid-rmse:8.34528 train-r2:0.51953 valid-r2:0.50493
[220] train-rmse:8.83656 valid-rmse:8.32250 train-r2:0.52855 valid-r2:0.50763
[230] train-rmse:8.77145 valid-rmse:8.30870 train-r2:0.53547 valid-r2:0.50926
[240] train-rmse:8.72003 valid-rmse:8.30321 train-r2:0.54090 valid-r2:0.50991
[250] train-rmse:8.67607 valid-rmse:8.29589 train-r2:0.54552 valid-r2:0.51078

[260] train-rmse:8.63885 valid-rmse:8.29111 train-r2:0.54941 valid-r2:0.51134
[270] train-rmse:8.60996 valid-rmse:8.28973 train-r2:0.55242 valid-r2:0.51150
[280] train-rmse:8.57784 valid-rmse:8.28530 train-r2:0.55575 valid-r2:0.51202
[290] train-rmse:8.54968 valid-rmse:8.28160 train-r2:0.55866 valid-r2:0.51246
[300] train-rmse:8.52080 valid-rmse:8.28095 train-r2:0.56164 valid-r2:0.51254
[310] train-rmse:8.49303 valid-rmse:8.28140 train-r2:0.56449 valid-r2:0.51248
[320] train-rmse:8.46923 valid-rmse:8.28174 train-r2:0.56693 valid-r2:0.51244
[330] train-rmse:8.45028 valid-rmse:8.27854 train-r2:0.56886 valid-r2:0.51282
[340] train-rmse:8.42114 valid-rmse:8.28089 train-r2:0.57183 valid-r2:0.51254
[350] train-rmse:8.40192 valid-rmse:8.27823 train-r2:0.57379 valid-r2:0.51286
[360] train-rmse:8.37694 valid-rmse:8.27552 train-r2:0.57632 valid-r2:0.51318
[370] train-rmse:8.35140 valid-rmse:8.27106 train-r2:0.57890 valid-r2:0.51370
[380] train-rmse:8.32580 valid-rmse:8.26504 train-r2:0.58147 valid-r2:0.51441
[390] train-rmse:8.30213 valid-rmse:8.26442 train-r2:0.58385 valid-r2:0.51448
[400] train-rmse:8.27511 valid-rmse:8.26158 train-r2:0.58655 valid-r2:0.51481
[410] train-rmse:8.24606 valid-rmse:8.26197 train-r2:0.58945 valid-r2:0.51477
[420] train-rmse:8.22087 valid-rmse:8.25998 train-r2:0.59196 valid-r2:0.51500
[430] train-rmse:8.19967 valid-rmse:8.25700 train-r2:0.59406 valid-r2:0.51535
[440] train-rmse:8.17517 valid-rmse:8.25513 train-r2:0.59648 valid-r2:0.51557
[450] train-rmse:8.15067 valid-rmse:8.25417 train-r2:0.59889 valid-r2:0.51568
[460] train-rmse:8.13282 valid-rmse:8.25400 train-r2:0.60065 valid-r2:0.51570
[470] train-rmse:8.09823 valid-rmse:8.25143 train-r2:0.60404 valid-r2:0.51601
[480] train-rmse:8.07605 valid-rmse:8.25276 train-r2:0.60621 valid-r2:0.51585
[490] train-rmse:8.05222 valid-rmse:8.24982 train-r2:0.60853 valid-r2:0.51619

```
[500]	train-rmse:8.03193	valid-rmse:8.24989	train-r2:0.61050
valid-r2:0.51619
[510]	train-rmse:7.99967	valid-rmse:8.24886	train-r2:0.61362
valid-r2:0.51631
[520]	train-rmse:7.98118	valid-rmse:8.24874	train-r2:0.61540
valid-r2:0.51632
[530]	train-rmse:7.96085	valid-rmse:8.24718	train-r2:0.61736
valid-r2:0.51650
[540]	train-rmse:7.94052	valid-rmse:8.25061	train-r2:0.61931
valid-r2:0.51610
[550]	train-rmse:7.91974	valid-rmse:8.24936	train-r2:0.62130
valid-r2:0.51625
[560]	train-rmse:7.89511	valid-rmse:8.25098	train-r2:0.62365
valid-r2:0.51606
[570]	train-rmse:7.86969	valid-rmse:8.24952	train-r2:0.62607
valid-r2:0.51623
[580]	train-rmse:7.84870	valid-rmse:8.25072	train-r2:0.62806
valid-r2:0.51609
Stopping. Best iteration:
[533]	train-rmse:7.95699	valid-rmse:8.24668	train-r2:0.61773
valid-r2:0.51656
```

1.1 Predict test_df using XGBoost

[20]:
```
p_test = clf.predict(d_test)
```

[21]:
```
sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('test_df.csv', index = False)
sub.head()
```

[21]:

	ID	y
0	1	82.844788
1	2	97.869873
2	3	82.781380
3	4	77.284958
4	5	113.026222

[ ]: