Mercedes-Benz Greener Manufacturing

May 24, 2020

0.1 Importing libraries

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
```

0.2 Loading the datasets

[2]:
```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

0.3 Exploring the data

[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]:
```
train_df.columns
```

[4]: Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)

```
[5]: train_df.shape
```

[5]: (4209, 378)

As you can see from above training data is having 4209 rows and 378 columns

0.3.1 Dropping the column ID from training data

```
[6]: train_df.drop('ID',inplace= True, axis=1)
```

We are dropping ID column because it is of no significance to us

```
[7]: train_df.shape
```

[7]: (4209, 377)

After dropping ID column we have one less column, as we can observe from above

```
[8]: test_df.shape
```

[8]: (4209, 377)

As you can see from above testing data is having 4209 rows and 377 columns

```
[9]: test_df.columns
```

[9]: Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=377)

0.3.2 Dropping the column ID from testing data

```
[10]: test_df.drop('ID',inplace=True,axis=1)
```

We are dropping ID column because it is of no significance to us

```
[11]: test_df.shape
```

[11]: (4209, 376)

After dropping ID column we have one less column, as we can observe from above

[12]:
```
test_df.describe()
```

[12]:
```
              X10          X11          X12          X13          X14  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.019007     0.000238     0.074364     0.061060     0.427893
std       0.136565     0.015414     0.262394     0.239468     0.494832
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     1.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

              X15          X16          X17          X18          X19  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.000713     0.002613     0.008791     0.010216     0.111665  …
std       0.026691     0.051061     0.093357     0.100570     0.314992  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       0.000000     0.000000     0.000000     0.000000     0.000000  …
max       1.000000     1.000000     1.000000     1.000000     1.000000  …

             X375         X376         X377         X378         X379  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.325968     0.049656     0.311951     0.019244     0.011879
std       0.468791     0.217258     0.463345     0.137399     0.108356
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       1.000000     0.000000     1.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

             X380         X382         X383         X384         X385
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.008078     0.008791     0.000475     0.000713     0.001663
std       0.089524     0.093357     0.021796     0.026691     0.040752
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

[8 rows x 368 columns]
```

[13]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 376 entries, X0 to X385
dtypes: int64(368), object(8)
memory usage: 12.1+ MB
```

0.4 Checking the columns having zero variance

[14]:
```
zero_var_col = train_df.var()[train_df.var() == 0].index.values
print(zero_var_col)
```

```
['X11' 'X93' 'X107' 'X233' 'X235' 'X268' 'X289' 'X290' 'X293' 'X297'
 'X330' 'X347']
```

We can observe there are 12 columns having zero variance

Dropping the columns having zero variance

[15]:
```
train_df.drop(zero_var_col,inplace = True, axis = 1)
```

[16]:
```
train_df.shape
```

[16]:
```
(4209, 365)
```

[17]:
```
train_df.describe()
```

[17]:
```
                 y         X10          X12          X13          X14  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean    100.669318     0.013305     0.075077     0.057971     0.428130
std      12.679381     0.114590     0.263547     0.233716     0.494867
min      72.110000     0.000000     0.000000     0.000000     0.000000
25%      90.820000     0.000000     0.000000     0.000000     0.000000
50%      99.150000     0.000000     0.000000     0.000000     0.000000
75%     109.010000     0.000000     0.000000     0.000000     1.000000
max     265.320000     1.000000     1.000000     1.000000     1.000000

               X15          X16          X17          X18          X19  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.000475     0.002613     0.007603     0.007840     0.099549  …
std       0.021796     0.051061     0.086872     0.088208     0.299433  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       0.000000     0.000000     0.000000     0.000000     0.000000  …
max       1.000000     1.000000     1.000000     1.000000     1.000000  …

              X375         X376         X377         X378         X379  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
```

```
mean       0.318841     0.057258     0.314802     0.020670     0.009503
std        0.466082     0.232363     0.464492     0.142294     0.097033
min        0.000000     0.000000     0.000000     0.000000     0.000000
25%        0.000000     0.000000     0.000000     0.000000     0.000000
50%        0.000000     0.000000     0.000000     0.000000     0.000000
75%        1.000000     0.000000     1.000000     0.000000     0.000000
max        1.000000     1.000000     1.000000     1.000000     1.000000

              X380         X382         X383         X384         X385
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.008078     0.007603     0.001663     0.000475     0.001426
std       0.089524     0.086872     0.040752     0.021796     0.037734
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

[8 rows x 357 columns]
```

[18]:
```python
train_df.columns
```

[18]:
```
Index(['y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=365)
```

[19]:
```python
test_df.drop(zero_var_col,inplace=True,axis=1)
```

[20]:
```python
test_df.shape
```

[20]:
```
(4209, 364)
```

[21]:
```python
test_df.columns
```

[21]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10', 'X12',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=364)
```

0.5 Checking for null values

[22]:
```python
np.sum(train_df.isnull().sum())
```

[22]:
```
0
```

There is no null values in training data, as we can observe from above

[23]:
```
np.sum(test_df.isnull().sum())
```

[23]: 0

There is no null values in testing data, as we can observe from above

0.6 Checking for columns not containing numeric data

[24]:
```
train_df.describe(include=['object'])
```

[24]:

	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

[25]:
```
label_columns = train_df.describe(include=['object']).columns.values
label_columns
```

[25]: array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype=object)

Above are the columns which do not have numeric data

0.7 Applying Label Encoder to the columns not having numeric data

[26]:
```
le = LabelEncoder()
for col in label_columns:
    le.fit(train_df[col].append(test_df[col]).values)
    train_df[col] = le.transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
```

0.8 Splitting target column from the testing data

[27]:
```
y= train_df['y']
```

[28]:
```
y
```

[28]:
```
0       130.81
1        88.53
2        76.26
3        80.62
4        78.02
         …
4204    107.39
4205    108.77
```

```
4206    109.22
4207     87.48
4208    110.85
Name: y, Length: 4209, dtype: float64
```

[29]:
```
X = train_df.drop('y',axis=1)
```

[30]:
```
X.columns
```

[30]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10', 'X12',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=364)
```

0.9 Splitting training data into training and validation data

[31]:
```
x_train,x_val,y_train,y_val = train_test_split(X,y,test_size = 0.25)
```

0.10 Applying PCA to perform dimensionality reduction

[32]:
```
pca = PCA(0.98, svd_solver="full")
```

[33]:
```
pca.fit(X)
```

[33]:
```
PCA(copy=True, iterated_power='auto', n_components=0.98, random_state=None,
    svd_solver='full', tol=0.0, whiten=False)
```

[34]:
```
pca.n_components_
```

[34]:
```
12
```

After applying PCA, we can observe only 12 columns are of significance to us

[35]:
```
pca.explained_variance_ratio_
```

[35]:
```
array([0.40868988, 0.21758508, 0.13120081, 0.10783522, 0.08165248,
       0.0140934 , 0.00660951, 0.00384659, 0.00260289, 0.00214378,
       0.00209857, 0.00180388])
```

If we add all the values above in the array, we get value = 0.98 i.e. it has retains features which contain overall 98% variance

[36]:
```
pca_X_train = pd.DataFrame(pca.transform(x_train))
```

[37]:
```
pca_X_val = pd.DataFrame(pca.transform(x_val))
```

[38]:
```
pca_test = pd.DataFrame(pca.transform(test_df))
```

0.11 Creating the model

Building a Xgboost Regressor

[39]:
```
model = xgb.XGBRegressor(objective='reg:linear',learning_rate = 0.1)
model.fit(pca_X_train,y_train)
```

```
[00:08:46] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:168: reg:linear is now
deprecated in favor of reg:squarederror.
[00:08:47] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:168: reg:linear is now
deprecated in favor of reg:squarederror.
```

[39]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:linear', random_state=0, reg_alpha=0, reg_lambda=1,
             scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

[40]:
```
pred_y_val = model.predict(pca_X_val)
```

[41]:
```
mean_squared_error(y_val,pred_y_val)
```

[41]:
```
75.3001974539391
```

Xgboost Regressor is giving a mean squared error of 87.99..

Building a Random Forest Regressor

[42]:
```
regr = RandomForestRegressor()
regr.fit(pca_X_train,y_train)
```

[42]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

[43]:
```
y_pred = regr.predict(pca_X_val)
```

[44]:
```
mean_squared_error(y_val,y_pred)
```

[44]:
```
80.8425039714641
```

Random Forest Regressor is giving a mean squared error of 91.17...

As we can see that mean squared error for Xgboost Regressor is lowest, so its the correct choice for prediction

0.12 Applying Xgboost on test_df

[45]:
```
pred_y_test = model.predict(pca_test)
pred_y_test
```

[45]:
```
array([ 78.172485,  95.30022 ,  79.78195 , …,  99.84983 , 106.8038  ,
        90.81117 ], dtype=float32)
```

pca_test is the data after applying PCA to test data