ml_benz

August 1, 2023

[1]:
```
# importing libraries
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline
```

0.1 Data Exploration

[2]:
```
# uploading the dataset
d1= pd.read_csv("train.csv")
```

[3]:
```
d1.head(10)
```

[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
5  18   92.93   t  b   e  c  d  g  h  s  …     0     0     1     0     0
6  24  128.76  al  r   e  f  d  f  h  s  …     0     0     0     0     0
7  25   91.91   o  l  as  f  d  f  j  a  …     0     0     0     0     0
8  27  108.67   w  s  as  e  d  f  i  h  …     1     0     0     0     0
9  30  126.99   j  b  aq  c  d  f  a  e  …     0     0     1     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     0     0     0     0     0
6     0     0     0     0     0
7     0     0     0     0     0
```

```
8       0       0       0       0       0
9       0       0       0       0       0

[10 rows x 378 columns]
```

[4]:
```
d1.tail(10)
```

[4]:
```
        ID       y  X0  X1  X2 X3 X4  X5 X6 X8  …  X375  X376  X377  X378  \
4199  8395   88.24   t  aa  ay  c  d  aa  l  o  …     1     0     0     0
4200  8397  108.59   z  aa   e  c  d  aa  i  w  …     1     0     0     0
4201  8399  107.39   w   v   t  d  d  aa  h  g  …     0     1     0     0
4202  8402  123.34  ap   l   s  c  d  aa  d  r  …     0     0     0     0
4203  8403   85.71  aq   s  as  c  d  aa  a  g  …     1     0     0     0
4204  8405  107.39  ak   s  as  c  d  aa  d  q  …     1     0     0     0
4205  8406  108.77   j   o   t  d  d  aa  h  h  …     0     1     0     0
4206  8412  109.22  ak   v   r  a  d  aa  g  e  …     0     0     1     0
4207  8415   87.48  al   r   e  f  d  aa  l  u  …     0     0     0     0
4208  8417  110.85   z   r  ae  c  d  aa  g  w  …     1     0     0     0

      X379  X380  X382  X383  X384  X385
4199     0     0     0     0     0     0
4200     0     0     0     0     0     0
4201     0     0     0     0     0     0
4202     0     0     0     0     0     0
4203     0     0     0     0     0     0
4204     0     0     0     0     0     0
4205     0     0     0     0     0     0
4206     0     0     0     0     0     0
4207     0     0     0     0     0     0
4208     0     0     0     0     0     0

[10 rows x 378 columns]
```

[5]:
```
d1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

[6]:
```
pd.options.display.float_format = '{:,.4f}'.format
var = d1.var()
v1 = var.reset_index()
v1.columns = ["id","values"]
variance= v1.sort_values("values",ascending=1)
variance.head()
```

[6]:
```
         id  values
275  X289  0.0000
315  X330  0.0000
254  X268  0.0000
332  X347  0.0000
97   X107  0.0000
```

0.2 Data Processing

[7]:
```
# We will remove the varialbes with variance 0 and
# We will also remove id since it has a huge variance
var = variance.loc[variance["values"] < 0,"id"]
data1 = d1.drop(var,axis=1)
data1.drop("ID",axis=1,inplace=True)
data1.head()
```

[7]:
```
          y  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X375  X376  X377  X378  X379  \
0  130.8100   k  v  at  a  d  u  j  o    0  …     0     0     1     0     0
1   88.5300   k  t  av  e  d  y  l  o    0  …     1     0     0     0     0
2   76.2600  az  w   n  c  d  x  j  x    0  …     0     0     0     0     0
3   80.6200  az  t   n  f  d  x  l  e    0  …     0     0     0     0     0
4   78.0200  az  v   n  f  d  h  d  n    0  …     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 377 columns]
```

[8]:
```
data1.head()
```

[8]:
```
          y  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X375  X376  X377  X378  X379  \
0  130.8100   k  v  at  a  d  u  j  o    0  …     0     0     1     0     0
1   88.5300   k  t  av  e  d  y  l  o    0  …     1     0     0     0     0
2   76.2600  az  w   n  c  d  x  j  x    0  …     0     0     0     0     0
3   80.6200  az  t   n  f  d  x  l  e    0  …     0     0     0     0     0
4   78.0200  az  v   n  f  d  h  d  n    0  …     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 377 columns]
```

[9]:
```
data1.isnull().sum()
```

[9]:
```
y       0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 377, dtype: int64
```

[10]:
```
data1.isnull().any(axis=1)
```

[10]:
```
0       False
1       False
2       False
3       False
4       False
        …
4204    False
4205    False
4206    False
4207    False
4208    False
Length: 4209, dtype: bool
```

[11]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 377 entries, y to X385
dtypes: float64(1), int64(368), object(8)
memory usage: 12.1+ MB
```

[ ]:
```

```

[12]:
```
#we do not have any missing valuese
```

[13]:
```
data1.describe()
```

[13]:

```
                y       X10       X11       X12       X13       X14  \
count 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000
mean    100.6693    0.0133    0.0000    0.0751    0.0580    0.4281
std      12.6794    0.1146    0.0000    0.2635    0.2337    0.4949
min      72.1100    0.0000    0.0000    0.0000    0.0000    0.0000
25%      90.8200    0.0000    0.0000    0.0000    0.0000    0.0000
50%      99.1500    0.0000    0.0000    0.0000    0.0000    0.0000
75%     109.0100    0.0000    0.0000    0.0000    0.0000    1.0000
max     265.3200    1.0000    0.0000    1.0000    1.0000    1.0000

              X15       X16       X17       X18  …      X375      X376  \
count 4,209.0000 4,209.0000 4,209.0000 4,209.0000  … 4,209.0000 4,209.0000
mean      0.0005    0.0026    0.0076    0.0078  …    0.3188    0.0573
std       0.0218    0.0511    0.0869    0.0882  …    0.4661    0.2324
min       0.0000    0.0000    0.0000    0.0000  …    0.0000    0.0000
25%       0.0000    0.0000    0.0000    0.0000  …    0.0000    0.0000
50%       0.0000    0.0000    0.0000    0.0000  …    0.0000    0.0000
75%       0.0000    0.0000    0.0000    0.0000  …    1.0000    0.0000
max       1.0000    1.0000    1.0000    1.0000  …    1.0000    1.0000

             X377      X378      X379      X380      X382      X383  \
count 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000
mean      0.3148    0.0207    0.0095    0.0081    0.0076    0.0017
std       0.4645    0.1423    0.0970    0.0895    0.0869    0.0408
min       0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
25%       0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
50%       0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
75%       1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
max       1.0000    1.0000    1.0000    1.0000    1.0000    1.0000

             X384      X385
count 4,209.0000 4,209.0000
mean      0.0005    0.0014
std       0.0218    0.0377
min       0.0000    0.0000
25%       0.0000    0.0000
50%       0.0000    0.0000
75%       0.0000    0.0000
max       1.0000    1.0000

[8 rows x 369 columns]
```

[14]:

```
#giong to apply the dataframe for better flexiblity and intuitive way of␣
↪storing.
c = data1.corr().abs()
```

[15]:
```
# unstack for index labeling
b = c.unstack()
```

[16]:
```
s= pd.DataFrame(b)
s.reset_index(inplace = True)
s.head()
```

[16]:
```
  level_0 level_1      0
0       y       y 1.0000
1       y     X10 0.0270
2       y     X11    nan
3       y     X12 0.0898
4       y     X13 0.0483
```

[17]:
```
s["flag"] = np.where(s["level_0"] == s["level_1"],"same","not same")
s.columns.values[2] = "corr"
s.head()
```

[17]:
```
  level_0 level_1   corr      flag
0       y       y 1.0000      same
1       y     X10 0.0270  not same
2       y     X11    nan  not same
3       y     X12 0.0898  not same
4       y     X13 0.0483  not same
```

[18]:
```
# Remove the variables with correlation more than .9
#.loc is the function for slicing the data and here we are using label based␣
↪slicing.
#s.loc[s["flag"] != "same",]
name = s.loc[(s["corr"] > .9) & (s["flag"] != "same") ,"level_1"]
```

[19]:
```
# going to findout unique elements and elements are sorted in array format
final_name = name.unique()
final_name
```

[19]:
```
array(['X251', 'X382', 'X215', 'X54', 'X76', 'X136', 'X162', 'X232',
       'X263', 'X272', 'X276', 'X279', 'X328', 'X35', 'X37', 'X39', 'X31',
       'X33', 'X302', 'X66', 'X111', 'X113', 'X134', 'X147', 'X198',
       'X222', 'X129', 'X61', 'X120', 'X102', 'X214', 'X239', 'X370',
       'X29', 'X137', 'X324', 'X248', 'X253', 'X385', 'X52', 'X172',
       'X216', 'X379', 'X48', 'X213', 'X84', 'X244', 'X101', 'X179',
       'X348', 'X71', 'X90', 'X94', 'X99', 'X122', 'X217', 'X242', 'X243',
       'X249', 'X320', 'X245', 'X88', 'X150', 'X363', 'X80', 'X98', 'X53',
       'X371', 'X199', 'X119', 'X311', 'X118', 'X227', 'X264', 'X130',
       'X49', 'X128', 'X58', 'X140', 'X146', 'X138', 'X158', 'X96',
       'X226', 'X326', 'X219', 'X360', 'X157', 'X156', 'X142', 'X62',
       'X250', 'X262', 'X266', 'X378', 'X187', 'X194', 'X362', 'X186',
```

```
'X238', 'X265', 'X112', 'X247', 'X205', 'X204', 'X368', 'X67',
'X19', 'X155', 'X152', 'X125', 'X229', 'X228', 'X254', 'X189',
'X364', 'X365', 'X89', 'X358', 'X202', 'X60', 'X178', 'X14',
'X230', 'X314', 'X184', 'X126', 'X296', 'X295', 'X299', 'X298',
'X44', 'X261', 'X346', 'X352', 'X367', 'X337', 'X334', 'X331',
'X246', 'X240', 'X208', 'X108', 'X185', 'X63', 'X17'], dtype=object)
```

[20]:
```
# going to drop unique elements
data2 = data1.drop(final_name,axis=1)
data2.head()
```

[20]:
```
          y  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X369  X372  X373  X374  X375  \
0  130.8100   k  v  at  a  d  u  j  o    0  …     0     0     0     0     0
1   88.5300   k  t  av  e  d  y  l  o    0  …     0     0     0     0     1
2   76.2600  az  w   n  c  d  x  j  x    0  …     0     0     0     0     0
3   80.6200  az  t   n  f  d  x  l  e    0  …     0     1     0     0     0
4   78.0200  az  v   n  f  d  h  d  n    0  …     0     0     0     0     0

   X376  X377  X380  X383  X384
0     0     1     0     0     0
1     0     0     0     0     0
2     0     0     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 231 columns]
```

[21]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 231 entries, y to X384
dtypes: float64(1), int64(222), object(8)
memory usage: 7.4+ MB
```

[22]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 377 entries, y to X385
dtypes: float64(1), int64(368), object(8)
memory usage: 12.1+ MB
```

0.3 Apply Label Encoder

[23]:
```
char = data2.select_dtypes(exclude='number')
char
```

[23]:
```
       X0 X1  X2 X3 X4  X5 X6 X8
0       k  v  at  a  d   u  j  o
1       k  t  av  e  d   y  l  o
2      az  w   n  c  d   x  j  x
3      az  t   n  f  d   x  l  e
4      az  v   n  f  d   h  d  n
…     .. ..  .. .. ..  .. .. ..
4204   ak  s  as  c  d  aa  d  q
4205    j  o   t  d  d  aa  h  h
4206   ak  v   r  a  d  aa  g  e
4207   al  r   e  f  d  aa  l  u
4208    z  r  ae  c  d  aa  g  w

[4209 rows x 8 columns]
```

[24]:
```
num = data2.select_dtypes(include='number')
num.describe()
```

[24]:
```
                y        X10        X11        X12        X13        X15  \
count 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000
mean     100.6693     0.0133     0.0000     0.0751     0.0580     0.0005
std       12.6794     0.1146     0.0000     0.2635     0.2337     0.0218
min       72.1100     0.0000     0.0000     0.0000     0.0000     0.0000
25%       90.8200     0.0000     0.0000     0.0000     0.0000     0.0000
50%       99.1500     0.0000     0.0000     0.0000     0.0000     0.0000
75%      109.0100     0.0000     0.0000     0.0000     0.0000     0.0000
max      265.3200     1.0000     0.0000     1.0000     1.0000     1.0000

              X16        X18        X20        X21  …       X369       X372  \
count 4,209.0000 4,209.0000 4,209.0000 4,209.0000  … 4,209.0000 4,209.0000
mean       0.0026     0.0078     0.1428     0.0026  …     0.0005     0.0005
std        0.0511     0.0882     0.3499     0.0511  …     0.0218     0.0218
min        0.0000     0.0000     0.0000     0.0000  …     0.0000     0.0000
25%        0.0000     0.0000     0.0000     0.0000  …     0.0000     0.0000
50%        0.0000     0.0000     0.0000     0.0000  …     0.0000     0.0000
75%        0.0000     0.0000     0.0000     0.0000  …     0.0000     0.0000
max        1.0000     1.0000     1.0000     1.0000  …     1.0000     1.0000

             X373       X374       X375       X376       X377       X380  \
count 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000 4,209.0000
mean       0.0192     0.2274     0.3188     0.0573     0.3148     0.0081
std        0.1374     0.4192     0.4661     0.2324     0.4645     0.0895
```

```
min        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
25%        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
50%        0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
75%        0.0000     0.0000     1.0000     0.0000     1.0000     0.0000
max        1.0000     1.0000     1.0000     1.0000     1.0000     1.0000

            X383       X384
count 4,209.0000 4,209.0000
mean      0.0017     0.0005
std       0.0408     0.0218
min       0.0000     0.0000
25%       0.0000     0.0000
50%       0.0000     0.0000
75%       0.0000     0.0000
max       1.0000     1.0000

[8 rows x 223 columns]
```

[25]:
```
char1 = pd.get_dummies(char.astype(str),drop_first=True)
char1.head()
```

[25]:
```
   X0_aa  X0_ab  X0_ac  X0_ad  X0_af  X0_ai  X0_aj  X0_ak  X0_al  X0_am  …  \
0      0      0      0      0      0      0      0      0      0      0  …
1      0      0      0      0      0      0      0      0      0      0  …
2      0      0      0      0      0      0      0      0      0      0  …
3      0      0      0      0      0      0      0      0      0      0  …
4      0      0      0      0      0      0      0      0      0      0  …

   X8_p  X8_q  X8_r  X8_s  X8_t  X8_u  X8_v  X8_w  X8_x  X8_y
0     0     0     0     0     0     0     0     0     0     0
1     0     0     0     0     0     0     0     0     0     0
2     0     0     0     0     0     0     0     0     1     0
3     0     0     0     0     0     0     0     0     0     0
4     0     0     0     0     0     0     0     0     0     0

[5 rows x 187 columns]
```

[26]:
```
# going to concatinating both objects(char1,num) in row wise
final_data = pd.concat([char1,num],axis=1)
final_data.head()
```

[26]:
```
   X0_aa  X0_ab  X0_ac  X0_ad  X0_af  X0_ai  X0_aj  X0_ak  X0_al  X0_am  …  \
0      0      0      0      0      0      0      0      0      0      0  …
1      0      0      0      0      0      0      0      0      0      0  …
2      0      0      0      0      0      0      0      0      0      0  …
3      0      0      0      0      0      0      0      0      0      0  …
4      0      0      0      0      0      0      0      0      0      0  …
```

```
   X369  X372  X373  X374  X375  X376  X377  X380  X383  X384
0     0     0     0     0     0     0     1     0     0     0
1     0     0     0     0     1     0     0     0     0     0
2     0     0     0     0     0     0     0     0     0     0
3     0     1     0     0     0     0     0     0     0     0
4     0     0     0     0     0     0     0     0     0     0

[5 rows x 410 columns]
```

0.4 Model Development

[27]:
```
# spliting data into independent and dependent features
X = final_data.drop("y",axis=1) #axis=1,means we are referring to columns(to␣
↪drop)
y = final_data.loc[:,"y"]
```

[28]:
```
# going to do 30,70 split
from sklearn.model_selection import train_test_split
X_test,X_train,y_test,y_train = train_test_split( X,y, test_size = 0.
↪3,random_state=42)
```

[29]:
```
import xgboost as xg
```

[30]:
```
xgr = xg.XGBRegressor(objective ='reg:squarederror',n_estimators = 10, seed =␣
↪42)
```

[31]:
```
xgr.fit(X_train,y_train)
```

[31]:
```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=10, n_jobs=0, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
             subsample=1, tree_method=None, validate_parameters=False,
             verbosity=None)
```

[32]:
```
ypredict = xgr.predict(X_test)
```

[33]:
```
# Create file for the competition submission
```

[34]:
```
d = pd.DataFrame()
d["y_test"] = y_test
```

```
d["ypredict"] = ypredict
d["mp"] = abs((d["y_test"]- d["ypredict"])/d["y_test"])
```

[35]:
```
d.head()
```

[35]:

	y_test	ypredict	mp
370	95.1300	90.4262	0.0494
3392	117.3600	108.3275	0.0770
2208	109.0100	108.3275	0.0063
3942	93.7700	87.1329	0.0708
1105	103.4100	92.0061	0.1103

[36]:
```
#ROOT MEAN SQUARE
from sklearn.metrics import mean_squared_error
```

[ ]:
```

```

[37]:
```
rmse = np.sqrt(mean_squared_error(y_test, ypredict))
RSME=("RMSE: %f" % (rmse))
print(RSME)
```

```
RMSE: 8.559700
```

[42]:
```
#Accuracy
from sklearn.metrics import mean_squared_error, r2_score

# evaluate predictions
```

[43]:
```
predictions = [round(value) for value in ypredict]
```

[44]:
```
y_test = [95.1300, 117.3600, 109.0100, 93.7700, 103.4100]
ypredict = [90.4262, 108.3275, 108.3275, 87.1329, 92.0061]

# Calculate metrics
rmse = np.sqrt(mean_squared_error(y_test, ypredict))
r2 = r2_score(y_test, ypredict)

# Print the metrics
print("RMSE:", rmse)
print("R-squared:", r2)
```

```
RMSE: 7.460263112786301
R-squared: 0.280786050169866
```

0.5 Model Evaluation

[45]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

[46]:
```
kfold = KFold(n_splits=50)
results = cross_val_score(xgr, X_train, y_train, cv=kfold)
AC=("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
print(AC)
```

```
Accuracy: 47.66% (26.77%)
```

[47]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=xgr,X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
```

```
Accuracy: 43.42 %
```

[48]:
```
xgr.get_params
```

[48]:
```
<bound method XGBModel.get_params of XGBRegressor(base_score=0.5, booster=None,
colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=10, n_jobs=0, num_parallel_tree=1, random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
             subsample=1, tree_method=None, validate_parameters=False,
             verbosity=None)>
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
```

[ ]:
```
parameters= [{"learning_rate": (0.05, 0.10, 0.15),
                        "max_depth": [ 3, 4, 5, 6, 8],
                        "min_child_weight": [ 1, 3, 5, 7],
                        "gamma":[ 0.0, 0.1, 0.2],
                        "colsample_bytree":[ 0.3, 0.4],}]
```

[ ]:
```
grid_search = GridSearchCV(estimator = xgr,param_grid =parameters,scoring =␣
 ↪'accuracy',cv = 10, n_jobs = -1)
```

[ ]:
```
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_accuracy
```

[ ]:
```
best_parameters = grid_search.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:", best_parameters)
```

[ ]:
```
print(RSME)
```

[ ]:
```
print(AC)
```

1 the RSME score 8.5

2 the KFold accuracy 47.6%

end

[ ]:

[49]:

[ ]:

[52]:

[ ]:

[ ]: