PS16

April 23, 2023

1 Car test-time prediction

1.1 Loading MB dataset

[1]:
```
import pandas as pd
data = pd.read_csv('mercedes_test.csv')
```

[2]:
```
# Figure out "m" and "features"
data
# m=4209, n=376
```

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

[4209 rows x 378 columns]
```

1.2 Data pre-processing

[5]:
```
# Choose categorical data
data_temp = data.select_dtypes(include=['object'])
data_temp
```

[5]:
```
      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]
```

[3]:
```
# Choose categorical data columns
cf = data.select_dtypes(include=['object']).columns
print(cf)
```

```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object')
```

[6]:
```
# To change it into "categorical" data type
data[cf]=data[cf].astype('category')
```

[7]:
```
# One hot encoding
data = pd.get_dummies(data)
print(data)
```

```
        ID       y  X10  X11  X12  X13  X14  X15  X16  X17  …  X8_p  X8_q  \
0        0  130.81    0    0    0    1    0    0    0    0  …     0     0
1        6   88.53    0    0    0    0    0    0    0    0  …     0     0
2        7   76.26    0    0    0    0    0    0    0    1  …     0     0
3        9   80.62    0    0    0    0    0    0    0    0  …     0     0
4       13   78.02    0    0    0    0    0    0    0    0  …     0     0
…     …      …  …  …  …  …  …  …  …  …  …  …  …  …  …
4204  8405  107.39    0    0    0    0    1    0    0    0  …     0     1
4205  8406  108.77    0    0    0    0    0    0    0    0  …     0     0
4206  8412  109.22    0    0    1    1    0    0    0    0  …     0     0
4207  8415   87.48    0    0    0    0    1    0    0    0  …     0     0
4208  8417  110.85    0    0    0    0    0    0    0    0  …     0     0

      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
1          0     0     0     0     0     0     0     0
2          0     0     0     0     0     0     1     0
3          0     0     0     0     0     0     0     0
4          0     0     0     0     0     0     0     0
…    …     …     …     …     …     …     …     …
4204       0     0     0     0     0     0     0     0
4205       0     0     0     0     0     0     0     0
4206       0     0     0     0     0     0     0     0
4207       0     0     0     1     0     0     0     0
4208       0     0     0     0     0     1     0     0

[4209 rows x 565 columns]
```

[8]:
```
# Obtain X from data (excluding 'ID' and 'y')
X_df = data.drop(['ID','y'],axis=1)
# Obtain y from data
y_df = data['y']
print(X_df)
print(y_df)
```

```
      X10  X11  X12  X13  X14  X15  X16  X17  X18  X19  …  X8_p  X8_q  X8_r  \
0       0    0    0    1    0    0    0    0    1    0  …     0     0     0
1       0    0    0    0    0    0    0    0    1    0  …     0     0     0
2       0    0    0    0    0    0    0    1    0    0  …     0     0     0
3       0    0    0    0    0    0    0    0    0    0  …     0     0     0
4       0    0    0    0    0    0    0    0    0    0  …     0     0     0
…   …   …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  …
4204    0    0    0    0    1    0    0    0    0    0  …     0     1     0
4205    0    0    0    0    0    0    0    0    0    0  …     0     0     0
4206    0    0    1    1    0    0    0    0    0    0  …     0     0     0
4207    0    0    0    0    1    0    0    0    0    0  …     0     0     0
4208    0    0    0    0    0    0    0    0    0    0  …     0     0     0

      X8_s  X8_t  X8_u  X8_v  X8_w  X8_x  X8_y
0        0     0     0     0     0     0     0
1        0     0     0     0     0     0     0
2        0     0     0     0     0     1     0
3        0     0     0     0     0     0     0
4        0     0     0     0     0     0     0
…    …   …  …  …  …  …  …  …  …  …  …  …  …
4204     0     0     0     0     0     0     0
4205     0     0     0     0     0     0     0
4206     0     0     0     0     0     0     0
4207     0     0     1     0     0     0     0
4208     0     0     0     0     1     0     0

[4209 rows x 563 columns]
```

```
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
```

[9]:
```
# Convert y_df into binary labels
import numpy as np
TF_vector= (y_df<np.median(y_df))
y_df=TF_vector.astype(float)
print(TF_vector)
print(y_df)
```

```
0       False
1        True
2        True
3        True
4        True
        …
4204    False
4205    False
4206    False
4207     True
4208    False
Name: y, Length: 4209, dtype: bool
0       0.0
1       1.0
2       1.0
3       1.0
4       1.0
       …
4204    0.0
4205    0.0
4206    0.0
4207    1.0
4208    0.0
Name: y, Length: 4209, dtype: float64
```

[10]:
```
# Conver data frame into numpy array
X,y = X_df.values, y_df.values
```

```
print(X)
print(X.shape)
print(y)
print(y.shape)
```

```
[[0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 1 0]
 …
 [0 0 1 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 1 0 0]]
(4209, 563)
[0. 1. 1. … 0. 1. 0.]
(4209,)
```

1.3 Split into train and test datasets

[11]:
```
from sklearn.model_selection import train_test_split
#X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,stratify=y)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,shuffle=False)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3788, 563)
(421, 563)
(3788,)
(421,)
```

1.4 Least Squares: Hyperparameter search via cross validation

[18]:
```
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import RandomizedSearchCV

model_LS = RidgeClassifier()
#grid_LS = dict(alpha = [10,1,1e-1,1e-2,1e-3])
grid_LS = {'alpha':[10,1,1e-1,1e-2,1e-3]}
cv_LS = RandomizedSearchCV(model_LS,grid_LS,n_iter=5,cv=5)
#cv_LS = RandomizedSearchCV(model_LS,grid_LS,n_iter=5,cv=5,random_state=42)
cv_LS.fit(X_train,y_train)
```

[18]:
```
RandomizedSearchCV(cv=5, estimator=RidgeClassifier(), n_iter=5,
                   param_distributions={'alpha': [10, 1, 0.1, 0.01, 0.001]})
```

[19]:
```
cv_LS.cv_results_ #logs results
```

[19]:
```
{'mean_fit_time': array([0.0309052 , 0.02832351, 0.03132544, 0.02870741,
0.04051003]),
 'std_fit_time': array([0.00123246, 0.00148989, 0.00173667, 0.00077108,
0.01193252]),
 'mean_score_time': array([0.00160837, 0.00159402, 0.00140052, 0.00120325,
0.00179863]),
 'std_score_time': array([0.00049733, 0.00049032, 0.00048599, 0.00039563,
0.0003744 ]),
 'param_alpha': masked_array(data=[10, 1, 0.1, 0.01, 0.001],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'alpha': 10},
  {'alpha': 1},
  {'alpha': 0.1},
  {'alpha': 0.01},
  {'alpha': 0.001}],
 'split0_test_score': array([0.90633245, 0.90105541, 0.90237467, 0.90369393,
0.90369393]),
 'split1_test_score': array([0.89050132, 0.88258575, 0.87335092, 0.87335092,
0.87335092]),
 'split2_test_score': array([0.88654354, 0.88390501, 0.87994723, 0.87994723,
0.87994723]),
 'split3_test_score': array([0.84676354, 0.84544254, 0.84412153, 0.84544254,
0.84544254]),
 'split4_test_score': array([0.87714663, 0.86922061, 0.8665786 , 0.8665786 ,
0.8665786 ]),
 'mean_test_score': array([0.8814575 , 0.87644186, 0.87327459, 0.87380264,
0.87380264]),
 'std_test_score': array([0.01974171, 0.01850609, 0.01890017, 0.01890995,
0.01890995]),
 'rank_test_score': array([1, 2, 5, 3, 3])}
```

[20]:
```
cv_LS.best_params_
```

[20]:
```
{'alpha': 10}
```

1.5 Store logs into csv file

[21]:
```
# Store logs into csv file
import pandas as pd
df_LS = pd.DataFrame.from_dict(cv_LS.cv_results_,orient='columns')
# Select columns to be stored
columns = ['params','mean_test_score','std_test_score','rank_test_score']
df_LS = df_LS[columns]
```

```
df_LS.to_csv("logs_LS.csv")
```

1.6 Save the best model

[22]:
```
best_model_LS=cv_LS.best_estimator_
from joblib import dump
dump(best_model_LS, 'best_model_LS.joblib')
```

[22]:
```
['best_model_LS.joblib']
```

1.7 Load “best_model_LS.joblib’

[23]:
```
from joblib import load
loaded_model_LS = load('best_model_LS.joblib')
loaded_model_LS.score(X_test,y_test)
```

[23]:
```
0.8954869358669834
```

[ ]:
```

```