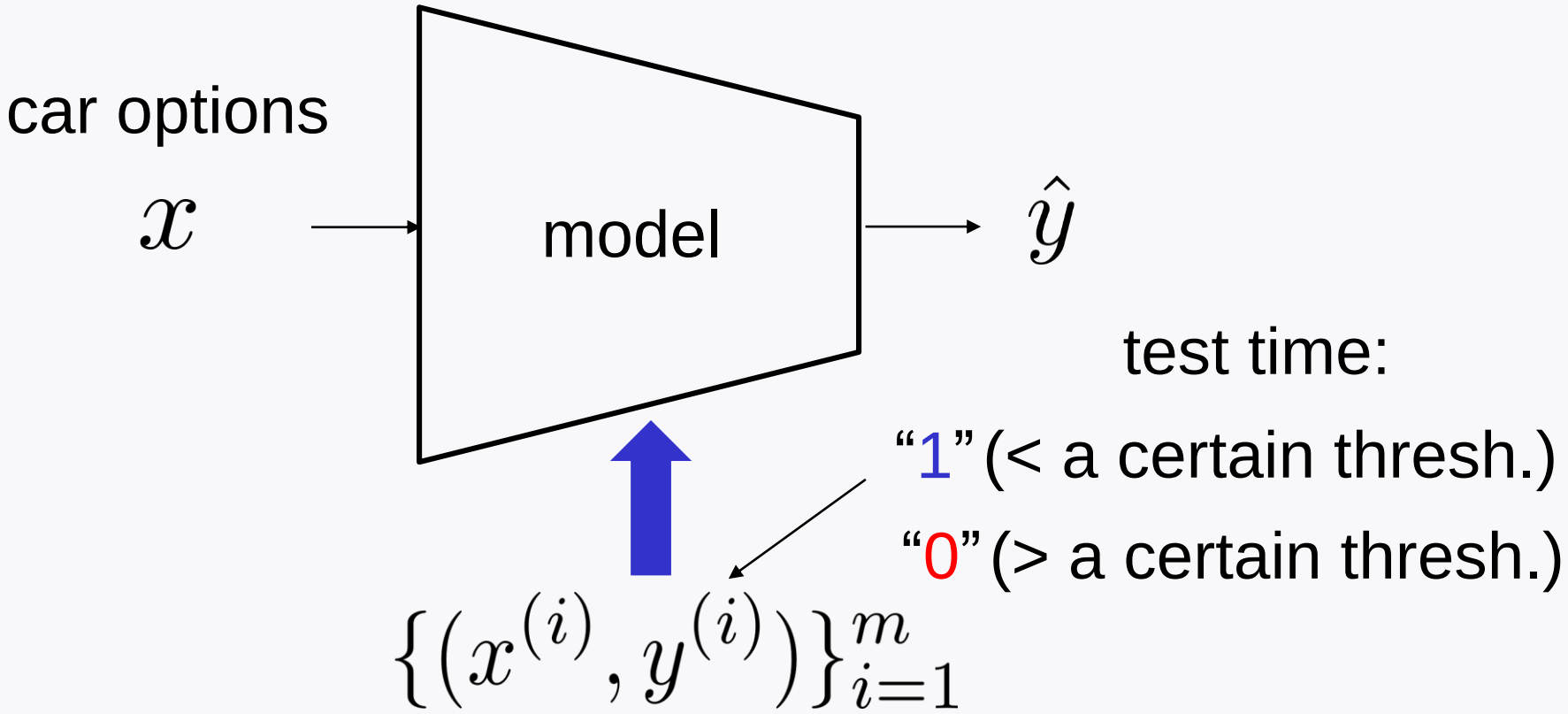
Mini-project #1

Practice Session 16

Changho Suh

January 29, 2024

Task: Test-time prediction



Dataset

Source: Mercedes-Benz provides **4209** examples

Data: anonymized **376** car options

Label: test time

Raw data is in **csv** file.

mercedes_test.csv

x_1	…	x_{376}	y
K	…	at	130.81
K	…	av	88.53
az	…	n	76.26

Loading MB dataset

```
import pandas as pd
data = pd.read_csv('mercedes_test.csv')
data
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	8405	107.39	ak	s	as	c	d	aa	d	q	...	1	0	0	0	0	0	0	0	0	0
4205	8406	108.77	j	o	t	d	d	aa	h	h	...	0	1	0	0	0	0	0	0	0	0
4206	8412	109.22	ak	v	r	a	d	aa	g	e	...	0	0	1	0	0	0	0	0	0	0
4207	8415	87.48	al	r	e	f	d	aa	l	u	...	0	0	0	0	0	0	0	0	0	0
4208	8417	110.85	z	r	ae	c	d	aa	g	w	...	1	0	0	0	0	0	0	0	0	0

4209 rows × 378 columns

strings (categorical data)

$$m = 4209 \quad n = 376 (= 378 - 2)$$

How to handle categorical data?

“strings columns” refer to the “object” dtype.

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

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

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

One hot encoding of categorical data

```
# 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]
```

$$n = 563$$

Obtain X and y from data

```
# 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]
```

X_df

```
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
```

y_df

Convert y into binary labels

```
# 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)
```

TF_vector

```
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
```

y_df

```
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
```

Convert data frame into numpy array

```
# Conver data frame into numpy array
X,y = X_df.values, y_df.values
print(X)
print(X.shape)
print(y)
print(y.shape)
```

X

```
[[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)
```

y

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

Split into train and test datasets

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

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

Model: Least Squares

```
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import RandomizedSearchCV

model_LS = RidgeClassifier()
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.fit(X_train,y_train)
```

Logs results

```
cv_LS.cv_results_ #logs results
```

```
{'mean_fit_time': array([0.03031492, 0.03012033, 0.02931895, 0.03749962, 0.02872453]),
 'std_fit_time': array([0.00101485, 0.00131544, 0.00103146, 0.0139438 , 0.00074677]),
 'mean_score_time': array([0.00100913, 0.00119596, 0.00139723, 0.00139709, 0.00139556]),
 'std_score_time': array([1.34647196e-05, 4.10985457e-04, 4.90121327e-04, 7.97296788e-04,
        4.88930546e-04]),
 '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([0.87730871, 0.87730871, 0.87862797, 0.87730871, 0.87730871]),
 'split1_test_score': array([0.88654354, 0.88390501, 0.88522427, 0.88258575, 0.88258575]),
 'split2_test_score': array([0.8707124 , 0.86939314, 0.86411609, 0.86411609, 0.86411609]),
 'split3_test_score': array([0.91413474, 0.91281374, 0.91149273, 0.91017173, 0.91017173]),
 'split4_test_score': array([0.87450462, 0.87054161, 0.8665786 , 0.8652576 , 0.8652576 ]),
 'mean_test_score': array([0.8846408 , 0.88279244, 0.88120793, 0.87988798, 0.87988798]),
 'std_test_score': array([0.01564618, 0.01588843, 0.0170065 , 0.01669633, 0.01669633]),
 'rank_test_score': array([1, 2, 3, 4, 4])}
```

Store logs results into csv file

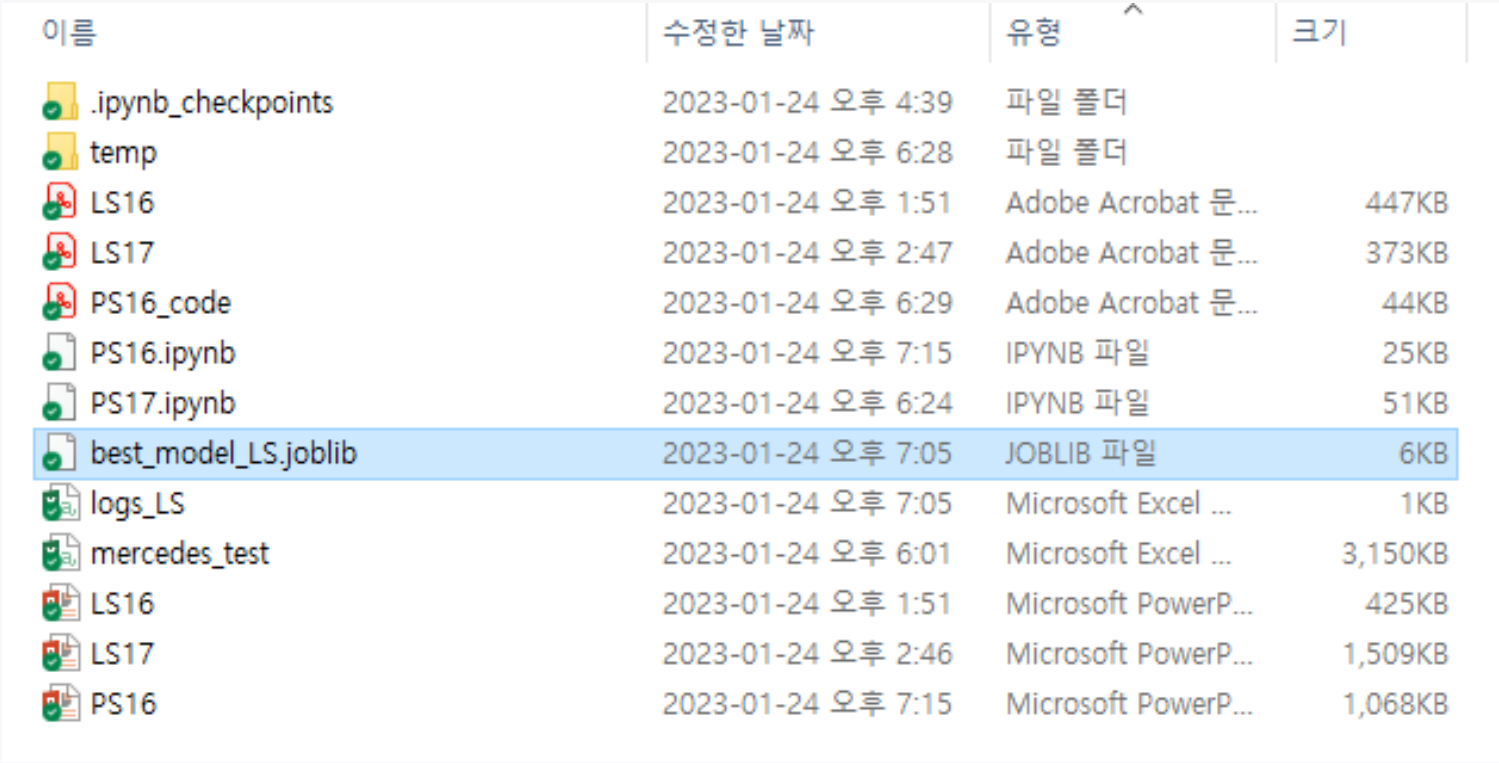
```
import pandas as pd
df_LS = pd.DataFrame.from_dict(cv_LS.cv_results_,orient='columns')

# 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")
```

logs_LS.csv

	A	B	C	D	E
1		params	mean_test_score	std_test_score	rank_test_score
2	0	{'alpha': 10}	0.884640802	0.015646182	1
3	1	{'alpha': 1}	0.882792442	0.015888433	2
4	2	{'alpha': 0.1}	0.881207934	0.017006497	3
5	3	{'alpha': 0.01}	0.879887976	0.016696327	4
6	4	{'alpha': 0.001}	0.879887976	0.016696327	4

Save the best model

```
best_model_LS=cv_LS.best_estimator_
from joblib import dump
dump(best_model_LS, 'best_model_LS.joblib')
```



이름	수정한 날짜	유형	크기
.ipynb_checkpoints	2023-01-24 오후 4:39	파일 폴더	
temp	2023-01-24 오후 6:28	파일 폴더	
LS16	2023-01-24 오후 1:51	Adobe Acrobat 문...	447KB
LS17	2023-01-24 오후 2:47	Adobe Acrobat 문...	373KB
PS16_code	2023-01-24 오후 6:29	Adobe Acrobat 문...	44KB
PS16.ipynb	2023-01-24 오후 7:15	IPYNB 파일	25KB
PS17.ipynb	2023-01-24 오후 6:24	IPYNB 파일	51KB
best_model_LS.joblib	2023-01-24 오후 7:05	JOBLIB 파일	6KB
logs_LS	2023-01-24 오후 7:05	Microsoft Excel ...	1KB
mercedes_test	2023-01-24 오후 6:01	Microsoft Excel ...	3,150KB
LS16	2023-01-24 오후 1:51	Microsoft PowerP...	425KB
LS17	2023-01-24 오후 2:46	Microsoft PowerP...	1,509KB
PS16	2023-01-24 오후 7:15	Microsoft PowerP...	1,068KB

Load “best_model_LS.joblib”

```
from joblib import load
loaded_model_LS = load('best_model_LS.joblib')
```

```
loaded_model_LS.score(X_test,y_test)
```

```
0.8836104513064132
```

Look ahead

Will implement logistic regression.