

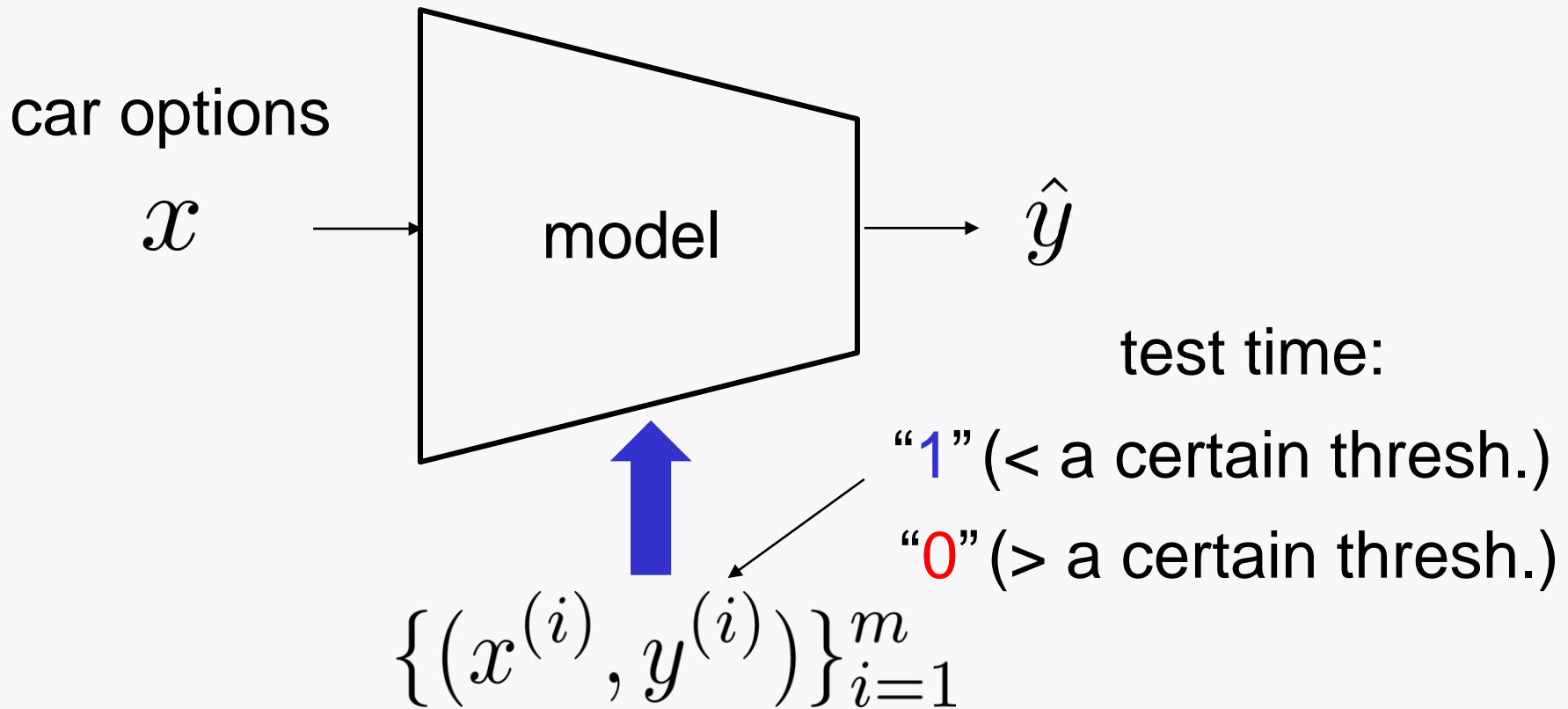
Mini-project #1

Practice Session 18

Changho Suh

January 29, 2024

Recap: Test-time prediction



Recap: 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)

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

Recap: Preprocessing

```
# Choose categorical data columns
cf = data.select_dtypes(include=['object']).columns
# To change it into "categorical" data type
data[cf]=data[cf].astype('category')
# One hot encoding
data = pd.get_dummies(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']

# Convert y_df into binary labels
import numpy as np
TF_vector= (y_df<np.median(y_df))
y_df=TF_vector.astype(float)

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

Recap: 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: DNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import RandomizedSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
# Enables the use of Scikit-Learn APIs for Keras models

def build_model(n_layer=2, lambda_=0, lr=1e-3):
    model = Sequential()
    for i in range(n_layer-1):
        model.add(Dense(20, activation='relu',
                        kernel_regularizer=l2(lambda_), bias_regularizer=l2(lambda_)))

    model.add(Dense(1, activation='sigmoid',
                    kernel_regularizer=l2(lambda_), bias_regularizer=l2(lambda_)))
    optimizer = Adam(learning_rate=lr)
    model.compile(optimizer=optimizer,
                  loss='binary_crossentropy',
                  metrics=['acc'])

    return model
```

Model: DNN

```
from sklearn.model_selection import RandomizedSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
# Enables the use of Scikit-Learn APIs for Keras models

# return a scikit-learn-like Keras model
model = KerasClassifier(build_model)

n_layer = [2,5,10]
lambda_ = [1e-3,1e-2,1e-1,1,10]
grid = {'n_layer':n_layer, 'lambda_':lambda_}

cv = RandomizedSearchCV(model,grid,n_iter=15,cv=5)

cv.fit(X_train,y_train,epochs=10,verbose=0)
```

Logs results

cv.cv_results_ # logs results

```
{ 'mean_fit_time': array([0.80088468, 0.99679437, 1.30053182, 0.80052128, 1.0428587,
    1.31682577, 0.7942235, 1.02531514, 1.42052927, 0.86476107,
    1.1448904, 1.43833871, 0.91565795, 1.1521472, 1.5095562 ]),
  'std_fit_time': array([0.07948719, 0.08256747, 0.08644163, 0.0252914, 0.07358794,
    0.08842912, 0.02503514, 0.06401977, 0.10777069, 0.04044909,
    0.07504672, 0.06786973, 0.07615218, 0.04095048, 0.12140173]),
  'mean_score_time': array([0.10044403, 0.1201704, 0.15436425, 0.13417854, 0.12991586,
    0.15232453, 0.09623604, 0.12205276, 0.16975675, 0.10253868,
    0.12651033, 0.16565557, 0.1109097, 0.12981415, 0.17698035]),
  'std_score_time': array([0.01255555, 0.00358549, 0.00364132, 0.07562545, 0.00795389,
    0.00355397, 0.00266729, 0.003046, 0.01128532, 0.00450796,
    0.00454143, 0.0088088, 0.0070474, 0.00567858, 0.00874225]),
  'param_n_layer': masked_array(data=[2, 5, 10, 2, 5, 10, 2, 5, 10, 2, 5, 10, 2, 5, 10],
    mask=[False, False, False, False, False, False, False, False, False,
    False, False, False, False, False, False],
    fill_value='?'),
  'param_lambda': masked_array(data=[0.001, 0.001, 0.001, 0.01, 0.01, 0.01, 0.1, 0.1, 0.
    1, 1, 1, 10, 10, 10],
    mask=[False, False, False, False, False, False, False, False, False,
    False, False, False, False, False, False],
    fill_value='?'),
  'params': [{'n_layer': 2, 'lambda': 0.001},
    {'n_layer': 5, 'lambda': 0.001},
    {'n_layer': 10, 'lambda': 0.001},
    {'n_layer': 2, 'lambda': 0.01},
    {'n_layer': 5, 'lambda': 0.01},
    {'n_layer': 10, 'lambda': 0.01},
    {'n_layer': 2, 'lambda': 0.1},
    {'n_layer': 5, 'lambda': 0.1},
    {'n_layer': 10, 'lambda': 0.1},
    {'n_layer': 2, 'lambda': 1},
    {'n_layer': 5, 'lambda': 1},
    {'n_layer': 10, 'lambda': 1},
    {'n_layer': 2, 'lambda': 10},
    {'n_layer': 5, 'lambda': 10},
    {'n_layer': 10, 'lambda': 10}],
  'split0_test_score': array([0.88522428, 0.8854351, 0.86147755, 0.88522428, 0.88522428,
    0.88258576, 0.86939311, 0.4828496, 0.4828496, 0.4828496,
    0.4828496, 0.4828496, 0.4828496, 0.5171504, 0.5171504 ]),
  'split1_test_score': array([0.88918203, 0.88654351, 0.87730873, 0.89050132, 0.88126647,
    0.86939311, 0.88390499, 0.49472296, 0.49472296, 0.49472296,
    0.49472296, 0.50527704, 0.49472296, 0.49472296, 0.49472296]),
  'split2_test_score': array([0.85883904, 0.86147755, 0.86147755, 0.8751981, 0.86015832,
    0.86015832, 0.83905011, 0.47757256, 0.47757256, 0.47757256,
    0.47757256, 0.47757256, 0.52242744, 0.52242744, 0.47757256]),
  'split3_test_score': array([0.87978864, 0.8678996, 0.87582564, 0.88110965, 0.87978864,
    0.87054163, 0.84940553, 0.49669749, 0.49669749, 0.49669749,
    0.50330251, 0.50330251, 0.55350065, 0.49669749, 0.49669749]),
  'split4_test_score': array([0.89431965, 0.87318361, 0.88903564, 0.89431965, 0.87978864,
    0.4953765, 0.87582564, 0.4953765, 0.4953765, 0.50462353,
    0.4953765, 0.4953765, 0.4953765, 0.4953765, 0.4953765 ]),
  'mean_test_score': array([0.88147073, 0.87512956, 0.87302502, 0.88173494, 0.87724527,
    0.79561107, 0.86351588, 0.48944382, 0.48944382, 0.5171849,
    0.49076483, 0.48287564, 0.50977543, 0.50527496, 0.49630398]),
  'std_test_score': array([0.01227739, 0.01002989, 0.01048066, 0.01291637, 0.00877195,
    0.15028647, 0.01673, 0.00774716, 0.00774716, 0.05529023,
    0.0092824, 0.01098612, 0.02541997, 0.01198441, 0.01256148]),
  'rank_test_score': array([ 2,  4,  5,  1,  3,  7,  6, 14, 14,  8, 13, 12,  9, 10, 11])}
```


Store logs results into csv file

```
# Store logs into csv file
import pandas as pd
df_DNN=pd.DataFrame.from_dict(cv.cv_results_,orient='columns')
# Select columns to be stored
columns = ['params','mean_test_score','std_test_score','rank_test_score']
df_DNN = df_DNN[columns]
df_DNN.to_csv("logs_DNN.csv")
```




















logs_DNN.csv

	A	B	C	D	E
1		params	mean_test_score	std_test_score	rank_test_score
2	0	{'n_layer': 2, 'lambda_': 0.001}	0.880150437	0.012390875	3
3	1	{'n_layer': 5, 'lambda_': 0.001}	0.871702981	0.009922373	5
4	2	{'n_layer': 10, 'lambda_': 0.001}	0.873552036	0.016688979	4
5	3	{'n_layer': 2, 'lambda_': 0.01}	0.883054197	0.012948202	1
6	4	{'n_layer': 5, 'lambda_': 0.01}	0.880679882	0.011110238	2
7	5	{'n_layer': 10, 'lambda_': 0.01}	0.79782958	0.1538105	7
8	6	{'n_layer': 2, 'lambda_': 0.1}	0.865622532	0.025063451	6
9	7	{'n_layer': 5, 'lambda_': 0.1}	0.491025543	0.006963835	13
10	8	{'n_layer': 10, 'lambda_': 0.1}	0.491025543	0.006963835	13
11	9	{'n_layer': 2, 'lambda_': 1}	0.491025543	0.006963835	13
12	10	{'n_layer': 5, 'lambda_': 1}	0.492080951	0.008143987	12
13	11	{'n_layer': 10, 'lambda_': 1}	0.494984365	0.010192118	11
14	12	{'n_layer': 2, 'lambda_': 10}	0.533545125	0.050735938	8
15	13	{'n_layer': 5, 'lambda_': 10}	0.508181858	0.007879908	9
16	14	{'n_layer': 10, 'lambda_': 10}	0.498683184	0.011282819	10

Save the best model

```
best_model_DNN=cv.best_estimator_  
best_model_DNN.model.save('best_model_DNN')
```

이름	수정한 날짜	유형	크기
 .ipynb_checkpoints	2023-01-25 오후 10:11	파일 폴더	
 best_model_DNN	2023-01-25 오후 10:28	파일 폴더	
 logs	2023-01-25 오후 2:42	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	2023-01-24 오후 7:22	Adobe Acrobat 문...	717KB
 PS16_code	2023-01-24 오후 7:21	Adobe Acrobat 문...	41KB
 PS17	2023-01-24 오후 8:12	Adobe Acrobat 문...	551KB
 PS17_code	2023-01-24 오후 7:55	Adobe Acrobat 문...	35KB
 PS18	2023-01-25 오후 10:11	Adobe Acrobat 문...	1,495KB
 PS18_code	2023-01-25 오후 2:45	Adobe Acrobat 문...	49KB
 PS_tot.ipynb	2023-01-25 오후 1:49	IPYNB 파일	20KB
 PS16.ipynb	2023-01-24 오후 7:21	IPYNB 파일	16KB
 PS17.ipynb	2023-01-24 오후 8:10	IPYNB 파일	8KB
 PS18.ipynb	2023-01-25 오후 9:24	IPYNB 파일	19KB
 PS19.ipynb	2023-01-25 오후 10:41	IPYNB 파일	20KB

Load the best model

```
from tensorflow.keras.models import load_model  
loaded_model = load_model('best_model_DNN')  
loaded_model.evaluate(X_test, y_test)
```

```
14/14 [=====] - 0s 890us/step - loss: 0.3867 - acc: 0.  
8789
```

```
[0.38672053813934326, 0.8788598775863647]
```