PS19

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')
```

1.2 Data pre-processing

[2]:
```
# 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

# 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,)
```

1.3 DNN: Hyparameter search via cross validation

[3]:
```
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
```

[4]:
```
from sklearn.model_selection import RandomizedSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
# Enables the use of Scikit-learn APIs for Keras models
```

[5]:
```
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
```

[7]:
```
# 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_}
#grid = dict(n_layer=n_layer,lambda_=lambda_)
cv = RandomizedSearchCV(model,grid,n_iter=15,cv=5)
```

```
C:\Users\chsuh\AppData\Local\Temp/ipykernel_33248/933115641.py:2:
DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras
(https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasClassifier(build_model)
```

[8]:
```
cv.fit(X_train,y_train,epochs=10,verbose=0)
```

```
24/24 [==============================] - 0s 618us/step - loss: 0.3773 - acc:
0.8536
24/24 [==============================] - 0s 564us/step - loss: 0.3353 - acc:
0.8892
```

```
24/24 [==============================] - 0s 615us/step - loss: 0.3560 - acc:
0.8879
24/24 [==============================] - 0s 650us/step - loss: 0.3478 - acc:
0.8851
24/24 [==============================] - 0s 595us/step - loss: 0.3773 - acc:
0.8587
24/24 [==============================] - 0s 618us/step - loss: 0.3621 - acc:
0.8760
24/24 [==============================] - 0s 617us/step - loss: 0.3858 - acc:
0.8852
24/24 [==============================] - 0s 628us/step - loss: 0.3774 - acc:
0.8747
24/24 [==============================] - 0s 651us/step - loss: 0.3593 - acc:
0.8785
24/24 [==============================] - 0s 607us/step - loss: 0.3958 - acc:
0.8613
24/24 [==============================] - 0s 587us/step - loss: 0.3935 - acc:
0.8707
24/24 [==============================] - 0s 671us/step - loss: 0.3759 - acc:
0.8826
24/24 [==============================] - 0s 650us/step - loss: 0.3845 - acc:
0.8813
24/24 [==============================] - 0s 689us/step - loss: 0.3800 - acc:
0.8771
24/24 [==============================] - 0s 780us/step - loss: 0.3987 - acc:
0.8560
24/24 [==============================] - 0s 651us/step - loss: 0.3893 - acc:
0.8852
24/24 [==============================] - 0s 626us/step - loss: 0.3884 - acc:
0.8865
24/24 [==============================] - 0s 563us/step - loss: 0.3972 - acc:
0.8879
24/24 [==============================] - 0s 629us/step - loss: 0.3964 - acc:
0.8811
24/24 [==============================] - 0s 564us/step - loss: 0.4236 - acc:
0.8639
24/24 [==============================] - 0s 564us/step - loss: 0.4532 - acc:
0.8813
24/24 [==============================] - 0s 607us/step - loss: 0.4572 - acc:
0.8865
24/24 [==============================] - 0s 650us/step - loss: 0.4600 - acc:
0.8865
24/24 [==============================] - 0s 696us/step - loss: 0.4489 - acc:
0.8785
24/24 [==============================] - 0s 650us/step - loss: 0.4925 - acc:
0.8560
24/24 [==============================] - 0s 607us/step - loss: 0.5260 - acc:
0.8786
```

```
24/24 [==============================] - 0s 662us/step - loss: 0.5355 - acc:
0.8813
24/24 [==============================] - 0s 650us/step - loss: 0.4959 - acc:
0.8852
24/24 [==============================] - 0s 650us/step - loss: 0.5024 - acc:
0.8692
24/24 [==============================] - 0s 651us/step - loss: 0.5751 - acc:
0.8626
24/24 [==============================] - 0s 564us/step - loss: 0.6212 - acc:
0.8865
24/24 [==============================] - 0s 607us/step - loss: 0.6160 - acc:
0.8575
24/24 [==============================] - 0s 607us/step - loss: 0.6304 - acc:
0.8826
24/24 [==============================] - 0s 651us/step - loss: 0.6188 - acc:
0.8719
24/24 [==============================] - 0s 607us/step - loss: 0.6332 - acc:
0.8468
24/24 [==============================] - 0s 631us/step - loss: 0.6933 - acc:
0.4934
24/24 [==============================] - 0s 781us/step - loss: 0.6934 - acc:
0.4855
24/24 [==============================] - 0s 636us/step - loss: 0.6933 - acc:
0.4921
24/24 [==============================] - 0s 607us/step - loss: 0.6932 - acc:
0.4980
24/24 [==============================] - 0s 630us/step - loss: 0.6933 - acc:
0.4967
24/24 [==============================] - 0s 651us/step - loss: 0.6934 - acc:
0.4934
24/24 [==============================] - 0s 737us/step - loss: 0.6935 - acc:
0.4855
24/24 [==============================] - 0s 736us/step - loss: 0.6934 - acc:
0.4921
24/24 [==============================] - 0s 651us/step - loss: 0.6935 - acc:
0.4980
24/24 [==============================] - 0s 650us/step - loss: 0.6934 - acc:
0.4967
24/24 [==============================] - 0s 621us/step - loss: 0.6948 - acc:
0.4934
24/24 [==============================] - 0s 619us/step - loss: 0.6956 - acc:
0.6095
24/24 [==============================] - 0s 654us/step - loss: 0.6957 - acc:
0.4921
24/24 [==============================] - 0s 607us/step - loss: 0.6949 - acc:
0.5020
24/24 [==============================] - 0s 639us/step - loss: 0.6941 - acc:
0.5033
```

```
24/24 [==============================] - 0s 600us/step - loss: 0.6950 - acc:
0.5066
24/24 [==============================] - 0s 607us/step - loss: 0.6945 - acc:
0.4855
24/24 [==============================] - 0s 636us/step - loss: 0.6956 - acc:
0.4921
24/24 [==============================] - 0s 607us/step - loss: 0.6942 - acc:
0.5020
24/24 [==============================] - 0s 694us/step - loss: 0.6948 - acc:
0.5033
24/24 [==============================] - 0s 694us/step - loss: 0.6952 - acc:
0.4934
24/24 [==============================] - 0s 694us/step - loss: 0.6959 - acc:
0.4855
24/24 [==============================] - 0s 681us/step - loss: 0.6956 - acc:
0.4921
24/24 [==============================] - 0s 675us/step - loss: 0.6951 - acc:
0.4980
24/24 [==============================] - 0s 650us/step - loss: 0.6954 - acc:
0.4967
24/24 [==============================] - 0s 628us/step - loss: 0.7037 - acc:
0.4934
24/24 [==============================] - 0s 655us/step - loss: 0.7028 - acc:
0.5145
24/24 [==============================] - 0s 607us/step - loss: 0.7040 - acc:
0.5079
24/24 [==============================] - 0s 780us/step - loss: 0.6944 - acc:
0.4980
24/24 [==============================] - 0s 585us/step - loss: 0.6973 - acc:
0.4967
24/24 [==============================] - 0s 911us/step - loss: 0.7095 - acc:
0.4934
24/24 [==============================] - 0s 693us/step - loss: 0.7050 - acc:
0.5145
24/24 [==============================] - 0s 650us/step - loss: 0.7130 - acc:
0.4921
24/24 [==============================] - 0s 615us/step - loss: 0.7074 - acc:
0.4980
24/24 [==============================] - 0s 648us/step - loss: 0.7130 - acc:
0.4967
24/24 [==============================] - 0s 694us/step - loss: 0.7152 - acc:
0.4934
24/24 [==============================] - 0s 694us/step - loss: 0.7131 - acc:
0.5145
24/24 [==============================] - 0s 711us/step - loss: 0.7168 - acc:
0.5079
24/24 [==============================] - 0s 666us/step - loss: 0.7207 - acc:
0.5020
```

```
24/24 [==============================] - 0s 694us/step - loss: 0.7117 - acc:
0.4967
```

[8]:
```
RandomizedSearchCV(cv=5,
                   estimator=<keras.wrappers.scikit_learn.KerasClassifier object
at 0x000002942CC583A0>,
                   n_iter=15,
                   param_distributions={'lambda_': [0.001, 0.01, 0.1, 1, 10],
                                        'n_layer': [2, 5, 10]})
```

[9]:
```
cv.cv_results_ # logs results
```

[9]:
```
{'mean_fit_time': array([0.81816645, 0.97575717, 1.28760223, 0.7794939 ,
1.01375632,
        1.32319827, 0.80969381, 1.03111253, 1.39536386, 0.79164624,
        1.01914234, 1.36449885, 0.87919998, 1.05041747, 1.32769833]),
 'std_fit_time': array([0.12923766, 0.06886391, 0.05507833, 0.01605459,
0.05976467,
        0.05491712, 0.03779203, 0.08212507, 0.10241848, 0.00807776,
        0.07424235, 0.09963074, 0.07041725, 0.04158356, 0.07000915]),
 'mean_score_time': array([0.10110211, 0.11616187, 0.15650449, 0.1202786 ,
0.11927738,
        0.15944762, 0.09946227, 0.12463989, 0.15906901, 0.09789214,
        0.12129426, 0.15575066, 0.09763913, 0.12981024, 0.15466461]),
 'std_score_time': array([0.01369132, 0.00207464, 0.00956652, 0.05263982,
0.00375598,
        0.00327182, 0.00325379, 0.00839755, 0.00468749, 0.00385157,
        0.00699148, 0.00267503, 0.0029595 , 0.02103595, 0.00157633]),
 '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='?',
             dtype=object),
 '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, 1, 10, 10, 10],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 '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([0.853562  , 0.87598944, 0.8707124 , 0.88522428,
0.88126647,
        0.87862796, 0.88654351, 0.4934037 , 0.4934037 , 0.4934037 ,
        0.50659633, 0.4934037 , 0.4934037 , 0.4934037 , 0.4934037 ]),
 'split1_test_score': array([0.88918203, 0.88522428, 0.88258576, 0.88654351,
0.88654351,
        0.88126647, 0.85751981, 0.48548812, 0.48548812, 0.60949868,
        0.48548812, 0.48548812, 0.51451188, 0.51451188, 0.51451188]),
 'split2_test_score': array([0.8878628 , 0.87467021, 0.88126647, 0.8878628 ,
0.88654351,
        0.88522428, 0.88258576, 0.49208444, 0.49208444, 0.49208444,
        0.49208444, 0.49208444, 0.50791556, 0.49208444, 0.50791556]),
 'split3_test_score': array([0.88507265, 0.87846762, 0.8771466 , 0.88110965,
0.87846762,
        0.86922061, 0.87186259, 0.4980185 , 0.4980185 , 0.5019815 ,
        0.5019815 , 0.4980185 , 0.4980185 , 0.4980185 , 0.5019815 ]),
 'split4_test_score': array([0.85865259, 0.86129457, 0.85601056, 0.8639366 ,
0.85601056,
        0.86261559, 0.84676355, 0.49669749, 0.49669749, 0.50330251,
        0.50330251, 0.49669749, 0.49669749, 0.49669749, 0.49669749]),
 'mean_test_score': array([0.87486641, 0.87512922, 0.87354436, 0.88093537,
0.87776634,
        0.87539098, 0.86905504, 0.49313845, 0.49313845, 0.52005417,
        0.49789058, 0.49313845, 0.50210943, 0.4989432 , 0.50290202]),
 'std_test_score': array([0.01545818, 0.00781541, 0.00969338, 0.00879572,
0.0113149 ,
        0.00828224, 0.01501187, 0.00438613, 0.00438613, 0.04494466,
        0.00786571, 0.00438613, 0.0078657 , 0.00807478, 0.00760902]),
 'rank_test_score': array([ 5,  4,  6,  1,  2,  3,  7, 13, 13,  8, 12, 13, 10,
11,  9])}
```

1.4 Store logs into csv file

[10]:
```
# 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")
```

1.5 Save the best model

[11]:
```
best_model_DNN=cv.best_estimator_
best_model_DNN.model.save('best_model_DNN')
```

```
INFO:tensorflow:Assets written to: best_model_DNN\assets
```

1.6 Load the best model

[12]:
```
from tensorflow.keras.models import load_model
loaded_model = load_model('best_model_DNN')
loaded_model.evaluate(X_test, y_test)
```

```
14/14 [==============================] - 0s 729us/step - loss: 0.3798 - acc:
0.8931
```

[12]:
```
[0.3797912299633026, 0.8931116461753845]
```

[ ]:
```

```