

머신러닝 파이프라인

학습 목표

- 머신러닝 파이프라인(pipeline)과 학습한 모델을 저장과 활용하는 방법을 알아본다.

목차

[01 데이터 불러오기](#)

[02 파이프라인 생성 및 학습](#)

[03 파이프라인 함수로 만들기](#)

[04 학습내용 저장 및 불러오기](#)

01 데이터 불러오기

[목차로 이동하기](#)

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

train = pd.read_csv("../dataset/Space_Titanic/train.csv")
test = pd.read_csv("../dataset/Space_Titanic/test.csv")
sub = pd.read_csv("../dataset/Space_Titanic/sample_submission.csv")

train.shape, test.shape, sub.shape
```

Out[1]:

```
((8693, 14), (4277, 13), (4277, 2))
```

In [2]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
#   Column          Non-Null Count  Dtype
---  -
0   PassengerId     8693 non-null   object
1   HomePlanet     8492 non-null   object
2   CryoSleep      8476 non-null   object
3   Cabin          8494 non-null   object
4   Destination     8511 non-null   object
5   Age            8514 non-null   float64
6   VIP            8490 non-null   object
7   RoomService    8512 non-null   float64
8   FoodCourt      8510 non-null   float64
9   ShoppingMall   8485 non-null   float64
10  Spa            8510 non-null   float64
11  VRDeck         8505 non-null   float64
12  Name           8493 non-null   object
13  Transported    8693 non-null   bool
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

In [3]:

```
train.head()
```

Out[3]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCo
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	357
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	128
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	7

In [4]:

```
train.columns
```

Out[4]:

```
Index(['PassengerId', 'HomePlanet', 'CryoSleep', 'Cabin', 'Destination', 'Age',
      'VIP', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck',
      'Name', 'Transported'],
      dtype='object')
```

In [5]:

```
sel = [ 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck' ]  
X = train[sel]  
y = train['Transported']  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

파이프라인에 사용될 임퓨터, 스케일러, 분류 모델 정의

In [6]:

```
from sklearn.impute import SimpleImputer  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.tree import DecisionTreeClassifier as DTC
```

In [7]:

```
imputer = SimpleImputer(strategy = "mean")  
scaler = MinMaxScaler()  
model = DTC()
```

02 파이프라인 생성 및 학습

[목차로 이동하기](#)

- 각 요소의 이름은 imputer, scaler, model 로 지정.

In [8]:

```
from sklearn.pipeline import Pipeline  
  
pipe_line = Pipeline([ ("imputer", imputer), ("scaler", scaler), ("model", model) ])  
pipe_line.fit(X_train, y_train)
```

Out[8]:

```
Pipeline(steps=[('imputer', SimpleImputer()), ('scaler', MinMaxScaler()),  
                ('model', DecisionTreeClassifier())])
```

파이프라인을 이용한 예측 수행

In [9]:

```
pipe_line.predict(X_test)
```

Out[9]:

```
array([False,  True, False, ...,  True,  True,  True])
```

In [10]:

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

Out[10]:

0.7378104875804968

정리

- 일부 변수를 선택 후, 결측치는 평균으로 대체된 후, 스케일링된 X_test가 DecisionTree 모델에 입력되어 나온 결과이다.
- predict 메서드는 파이프라인의 마지막 요소인 의사결정트리 모델의 메서드를 사용하였다.
- score()메서드를 활용하여 확인 결과, 74%의 정확도를 갖는다.

실습

- 파이프라인의 모델을 다른 모델(KNN)으로 변경 후, 수행해 보기.
- 모델이 앞의 모델과 비교하여 어떠한지 비교해 보기

모델의 스케일링 함수 변경해 보기

In [11]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier as KNN_C

imputer = SimpleImputer(strategy = "mean")
scaler = StandardScaler()
model = KNN_C()
```

In [12]:

```
pipe_line = Pipeline([ ("imputer", imputer),
                        ("scaler", scaler),
                        ("model", model) ])

pipe_line.fit(X_train, y_train)
pipe_line.score(X_test, y_test)
```

Out[12]:

0.764949402023919

Pipeline 클래스의 단점

- 손쉽게 파이프라인을 구축할 수 있는 장점이 있다.
- 단, 다음과 같은 문제도 존재한다.
 - 1. 사이킷런 인스턴스가 아닌 다른 클래스의 인스턴스가 파이프라인에 포함되면 정상적으로 작동하지 않을 수 있음.
 - 2. 하이퍼파라미터 튜닝 등을 할 때 전체 파이프라인을 계속해서 수정해야 한다.
 - 해결 제안 : 함수나 클래스를 사용하여 파이프라인을 함수화 해보기

03 파이프라인 함수로 만들기

[목차로 이동하기](#)

pipe_line_fnc 함수 만들기

In [13]:

```
def pipe_line_fnc(X, imputer, scaler, model):  
    X = imputer.transform(X)  
    X = scaler.transform(X)  
    pred = model.predict(X)  
    return pred  
  
pred_Y = pipe_line_fnc(X_test, imputer, scaler, model)  
  
pred_Y[:5]
```

Out[13]:

```
array([False,  True, False, False,  True])
```

In [14]:

```
pred_Y.shape
```

Out[14]:

```
(2174,)
```

In [15]:

```
from sklearn.metrics import accuracy_score  
accuracy_score(pred_Y, y_test)
```

Out[15]:

```
0.764949402023919
```

최종 결과 73.59%

In [16]:

```
from sklearn.metrics import classification_report  
print( classification_report(pred_Y, y_test) )
```

	precision	recall	f1-score	support
False	0.70	0.80	0.75	934
True	0.83	0.74	0.78	1240
accuracy			0.76	2174
macro avg	0.76	0.77	0.76	2174
weighted avg	0.77	0.76	0.77	2174

- 정밀도 : 0.85, 재현율 : 0.70, f1-score : 0.76

04 학습내용 저장 및 불러오기

[목차로 이동하기](#)

In [17]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
# from sklearn.externals import joblib
import sklearn.externals
import joblib
```

In [18]:

```
sel = [ 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck' ]
X = train[sel]
y = train['Transported']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [19]:

```
imputer = SimpleImputer(strategy = "mean")
scaler = MinMaxScaler()
model = LogisticRegression()
```

In [20]:

```
pipe_line_Log = Pipeline([ ("imputer", imputer), ("scaler", scaler), ("model", model) ])
pipe_line_Log.fit(X_train, y_train)
pred = pipe_line_Log.predict(X_test)
pred[0:5]
```

Out[20]:

```
array([ True,  True, False, False,  True])
```

In [21]:

```
# 정확도 확인
print( accuracy_score(pred, y_test) )
```

```
0.7382704691812327
```

In [22]:

```
model = KNeighborsClassifier()
```

In [23]:

```
pipe_line_knn = Pipeline([ ("imputer", imputer), ("scaler", scaler), ("model", model) ])
pipe_line_knn.fit(X_train, y_train)
pred = pipe_line_knn.predict(X_test)

# 정확도 확인
print( accuracy_score(pred, y_test) )
```

0.7723091076356946

In [24]:

```
joblib.dump(pipe_line_knn, "../dataset/Space_Titanic/model_pipe_knn.joblib" )
```

Out[24]:

```
['../dataset/Space_Titanic/model_pipe_knn.joblib']
```

In [25]:

```
import os
os.listdir("../dataset/Space_Titanic/")
```

Out[25]:

```
['first_sub.csv',
 'model_pipe_knn.joblib',
 'sample_submission.csv',
 'test.csv',
 'train.csv']
```

불러오기 후, 확인

In [26]:

```
import sklearn.externals
import joblib

pipe_knn = joblib.load("../dataset/Space_Titanic/model_pipe_knn.joblib")

pred = pipe_knn.predict(X_test)

# 정확도 확인
print( accuracy_score(pred, y_test) )
```

0.7723091076356946