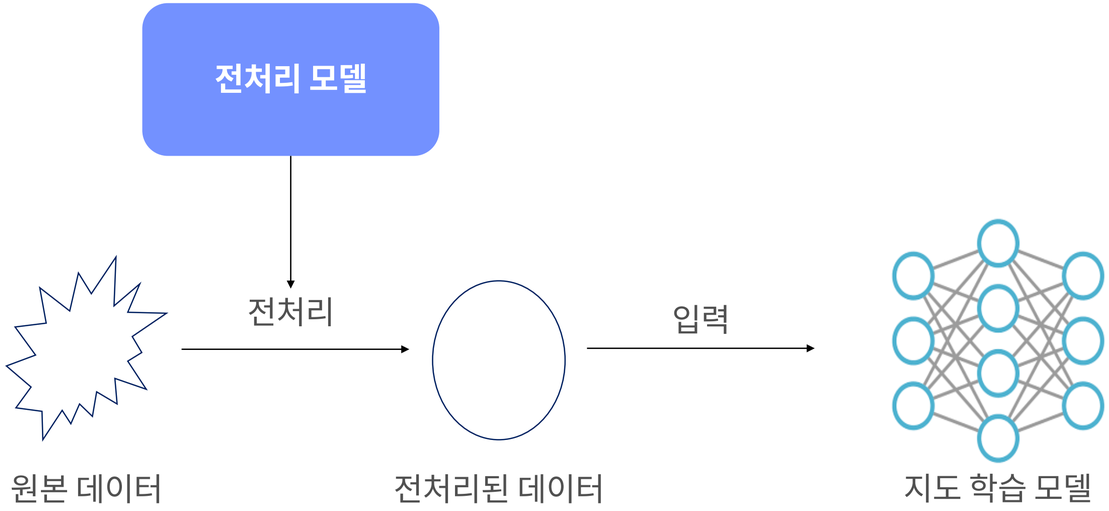
2. 데이터 탐색 및 전처리

1 사이킷런을 이용한 데이터 전처리

전처리 모델 인스턴스

새로 입력된 데이터에 모델 학습 시에 사용했던 전처리를 동일하게 적용해야 하기 때문에, 사이킷런을 이용한 데이터 전처리는 함수가 아니라 전처리 모델 인스턴스를 통해 이뤄집니다.



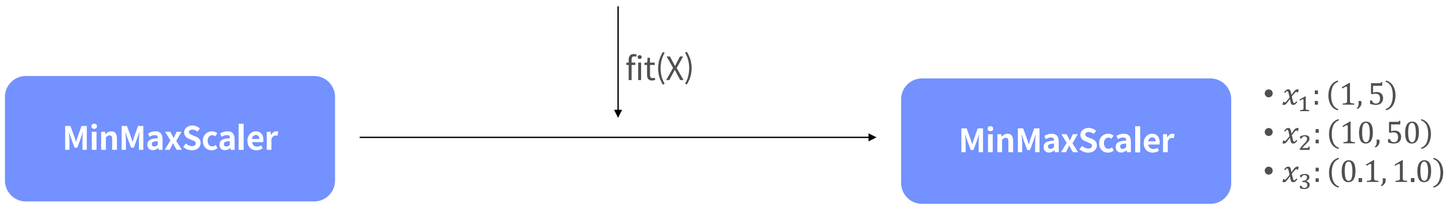
fit 메서드

fit 메서드는 전처리 인스턴스를 학습하는 데 사용하며, 학습 데이터의 특징 벡터만 입력받거나 라벨까지 입력받습니다.

(예시) 최소-최대 스케일링을 수행하는 MinMaxScaler 인스턴스

X

x_1	x_2	x_3
1	50	0.2
2	20	0.1
5	30	0.5
4	20	0.0
3	10	1.0

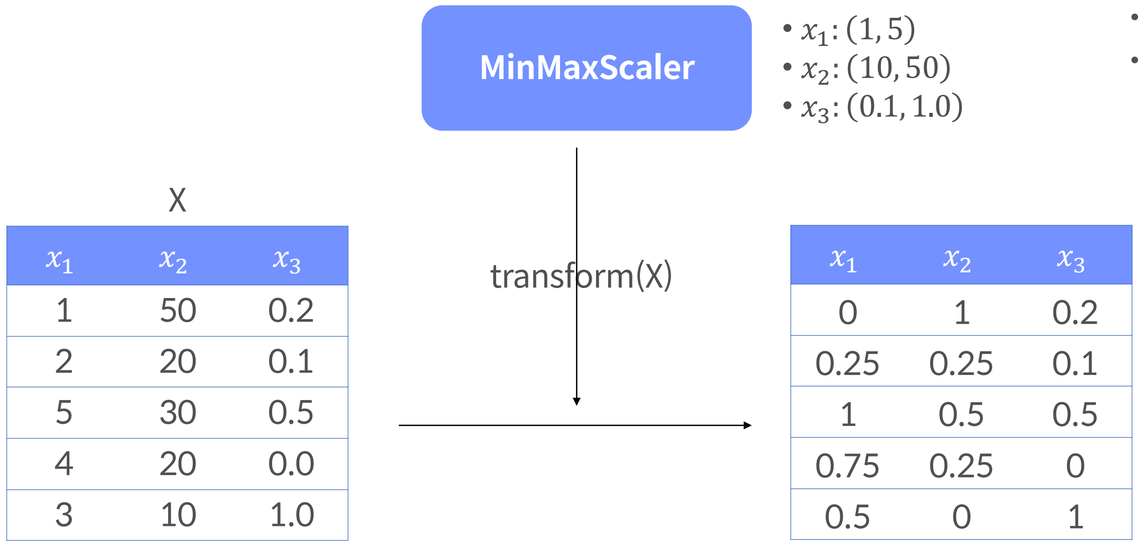


- x_1: $(1, 5)$
- x_2: $(10, 50)$
- x_3: $(0.1, 1.0)$

transform 메서드

transform 메서드는 학습한 내용을 바탕으로 데이터를 전처리합니다.

(예시) 최소-최대 스케일링을 수행하는 MinMaxScaler 인스턴스



x_1	x_2	x_3
1	50	0.2
2	20	0.1
5	30	0.5
4	20	0.0
3	10	1.0

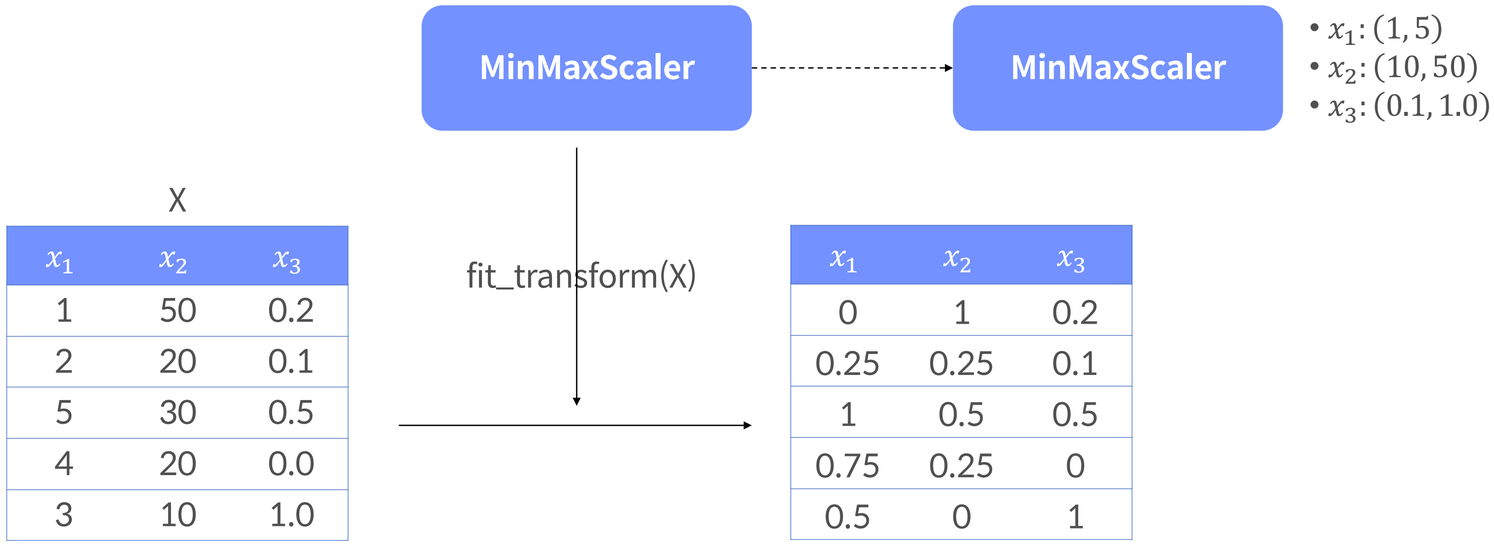
x_1	x_2	x_3
0	1	0.2
0.25	0.25	0.1
1	0.5	0.5
0.75	0.25	0
0.5	0	1

- 전처리 인스턴스를 비롯한 사이킷런 인스턴스 대부분은 ndarray 자료형을 출력함
- 이 그림에서 설명을 위해 transform(X)가 열 인덱스를 갖고 있지만 실제로는 그렇지 않음
- 그러므로 인덱스가 필요하다면 다시 데이터프레임으로 변환하는 과정이 필요함

fit_transform 메서드

fit_transform 메서드는 이름에서 알 수 있다시피, fit과 transform을 동시에 수행합니다.

(예시) 최소-최대 스케일링을 수행하는 MinMaxScaler 인스턴스



x_1	x_2	x_3
1	50	0.2
2	20	0.1
5	30	0.5
4	20	0.0
3	10	1.0

x_1	x_2	x_3
0	1	0.2
0.25	0.25	0.1
1	0.5	0.5
0.75	0.25	0
0.5	0	1

fit_transform 메서드는 전처리된 배열을 반환함에 주의해야 함

2. 데이터 탐색 및 전처리

2 결측치 처리

예제 데이터 불러오기

판다스와 사이킷런을 사용해 결측을 확인하고 처리하는 방법을 설명하는 데 사용할 예제 데이터를 불러옵니다.

데이터 불러오기 예제

```
1  import os
2  import pandas as pd
3  os.chdir("../../data")
4  df = pd.read_csv("classification/bands.csv")
```

결측 탐색
: isnull 메서드

isnull 메서드는 데이터프레임 혹은 시리즈의 요소가 결측이면 True를 그렇지 않으면 False를 반환합니다.

결측 확인 예제

```
1  display(df.isnull().head())
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	y
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False

결측 탐색
: isnull 메서드 (계속)

개별 요소가 결측인지 일일이 확인하는 것은 매우 비효율적이므로 보통 특징별 결측 개수를 계산합니다. 파이썬에서 부울 자료형 간 사칙 연산을 수행하면 True를 1로 False를 0으로 간주하므로, insull 메서드를 적용한 결과의 열별 합으로 열별 결측 개수를 구할 수 있습니다.

결측 확인 예제

```
1    display(df.isnull().sum(axis = 0))
```

```
x1    54
x2     5
x3    27
x4     2
x5     1
x6    30
x7    63
x8    55
x9    10
x10   55
x11   55
x12   56
x13   54
x14    6
x15    7
x16   54
x17    7
x18    7
x19    3
y      0
dtype: int64
```

결측 제거
: dropna 메서드

dropna 메서드는 결측을 제거합니다.

주요 인자

인자	설명	기본 값
axis	제거 방향을 설정하는 인자로, 0이면 결측이 있는 행을, 1이면 결측이 있는 열을 제거	0
how	제거 기준을 설정하는 인자로, "any"면 하나의 요소라도 결측인 행이나 열을, "all"이면 모든 요소가 결측인 행이나 열을 제거	“any”
inplace	False면 결측을 제거한 데이터를 반환하고, True면 데이터 자체에서 결측을 제거하고 어떠한 값도 반환하지 않음	False

결측 행 제거 예제

```
1  df.dropna(inplace = True)
2  print(df.isnull().sum().sum())
```

```
0
```

대푯값을 활용한 결측 대체

사이킷런의 SimpleImputer를 이용하면 결측을 특징별 대푯값으로 대체할 수 있습니다.

주요 인자

인자	설명	기본 값
strategy	결측을 대체할 대푯값을 결정 (“mean”: 평균, “median”: 중위수, “most_frequent”: 최빈값)	“mean”

결측치 평균으로 대체 예제

```
1  X = df.drop('y', axis = 1)
2  y = df['y']
3
4  from sklearn.model_selection import train_test_split
5  X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 2022)
6  from sklearn.impute import SimpleImputer
7  imputer = SimpleImputer(strategy = "mean")
8  imputer.fit(X_train)
9  Z_train = pd.DataFrame(imputer.transform(X_train), columns = X_train.columns)
10 Z_test = pd.DataFrame(imputer.transform(X_test), columns = X_test.columns)
```

- **라인 1 – 2:** 특징과 라벨로 분리합니다.
- **라인 4 – 5**: 학습 데이터와 평가 데이터로 분리합니다.
- **라인 7**: 결측을 평균으로 대체하는 임퓨터 인스턴스를 생성합니다.
- **라인 8**: imputer를 X_train으로 학습합니다. 이 과정에서 imputer는 특징별 평균을 계산해 저장합니다.
- **라인 9**: imputer로 X_train을 변환한 결과를 Z_train에 저장합니다. 이때, 칼럼 명을 보존하기 위해 DataFrame의 columns 인자에 원래 칼럼 명인 X_train.columns를 입력합니다.

이웃을 활용한 결측 대체

사이킷런의 KNNImputer를 사용하면 이웃을 활용하여 결측을 대체할 수 있습니다.

주요 인자

인자	설명	기본 값
n_neighbors	이웃 수	5
metric	거리 척도	'nan_euclidean'

이웃을 활용한 결측 대체 예제

```
1  from sklearn.impute import KNNImputer
2  imputer = KNNImputer()
3  imputer.fit(X_train)
4  Z_train = pd.DataFrame(imputer.transform(X_train), columns = X_train.columns)
5  Z_test = pd.DataFrame(imputer.transform(X_test), columns = X_test.columns)
```

- **라인 2**: 이웃으로 결측을 대체하는 임퓨터 인스턴스를 생성합니다.

2. 데이터 탐색 및 전처리

3 범주형 변수 처리

더미화 함수 비교

더미화는 사이킷런의 OneHotEncoder, 판다스의 get_dummies, 피쳐엔진의 OneHotEncoder 등을 이용해 구현할 수 있습니다.

사이킷런의 OneHotEncoder는 연속형 특징도 더미화하므로 범주형 특징과 연속형 특징을 구분해서 적용하고 다시 병합해야 해서 번거로움

get_dummies는 사용 방법이 매우 간편하나 학습되지 않는 함수이기에 새롭게 입력된 데이터에 동일하게 적용하기 어려움

이러한 이유로 강사는 더미화에 피쳐엔진 패키지를 주로 사용함

피쳐엔진 설치

```
$ pip install feature-engine
```

예제 데이터 불러오기

판다스와 피쳐엔진(feature_engine)을 사용해 범주 및 서열형 변수를 처리하는 예제에 사용할 데이터를 불러옵니다.

데이터 불러오기 예제

```
1  df = pd.read_csv("classification/german.csv")
2  X = df.drop('y', axis = 1)
3  y = df['y']
4  X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 2022)
```

더미화

OneHotEncoder 클래스는 사이킷런의 전처리 인스턴스와 유사하게 작동하는 더미화 인스턴스를 생성합니다.

주요 인자

인자	설명	기본 값
variables	더미화할 변수의 이름 목록으로, 기본값인 None을 입력하면 자료형이 object인 변수를 더미화함	None
drop_last	마지막 더미 변수를 제거할지 여부	False
top_categories	한 변수로부터 만드는 더미 변수 개수로, 빈도가 높은 값이 더미 변수가 됨	None

- top_categories 인자는 상태 공간이 큰 범주형 변수를 더미화했을 때 특징이 지나치게 많아지는 문제를 방지하는 데 적합함
- 해당 인자를 사용하면 빈도가 적은 더미 변수는 모두 무시되므로 이들을 하나의 그룹 으로 묶기 위해 drop_last가 자동으로 False로 설정됨

더미화예제

```
1  from feature_engine.encoding import OneHotEncoder as OHE
2  dummy_model = OHE(drop_last = True).fit(X_train)
3  Z_train = dummy_model.transform(X_train)
4  Z_test = dummy_model.transform(X_test)
```

- **라인 2**: 자료형이 object인 변수를 더미화하는 인스턴스 dummy_model을 만든 뒤, X_train로 학습합니다.
- **라인 3 – 4**: X_train과 X_test를 dummy_model을 이용해 각각 Z_train과 Z_test로 변환합니다. 사이킷런과 다르게 피쳐엔진의 출력은 ndarray가 아니라 데이터프레임이므로 데이터프레임으로 따로 변환하지 않았습니다.

라벨을 이용한 치환

라벨을 활용하여 범주형 변수의 값을 치환하려면 해당 변수에 따른 라벨의 대푯값을 계산해야 하는데, 이때 사용할 수 있는 메서드로 groupby가 있습니다.

groupby 사용 구조

```
DataFrame.groupby(조건 변수)[대상 변수].통계 관련 메서드
```

- 조건 변수와 대상 변수는 DataFrame에 포함돼 있어야 하며, 조건 변수와 대상 변수가 둘 이상이면 리스트로 입력함
- groupby 해석: 조건 변수에 따른 대상 변수의 통계량

groupby 사용 예

```
1  S = df.groupby('x1')['y'].mean()
2  display(S)
```

```
x1
A11   0.492701
A12   0.390335
A13   0.222222
A14   0.116751
Name: y, dtype: float64
```

라벨을 이용한 치환 (계속)

to_dict는 시리즈를 사전으로 변환하는 메서드로, 인덱스가 키로 데이터가 값으로 바뀝니다. replace는 시리즈의 요소 가운데 사전의 키와 같은 요소를 사전의 값으로 바꾸는 메서드입니다.

to_dict와 replace 예시

```
1  display(df['x1'].replace(S.to_dict()))
```

```
0    0.492701
1    0.390335
2    0.116751
3    0.492701
4    0.492701
     ...
995  0.492701
996  0.390335
997  0.390335
998  0.116751
999  0.390335
Name: x1, Length: 1000, dtype: float64
```

라벨을 이용한 치환 (계속)

자료형이 object인 변수에 대해 변수에 따른 라벨의 평균을 구하고 이를 바탕으로 변수의 값을 대체합니다.

라벨을 이용한 치환 예제

```
1    train = pd.concat([X_train, y_train], axis = 1)
2    for col, dtype in zip(X_train.columns, X_train.dtypes):
3        if dtype == object:
4            S = train.groupby(col)['y'].mean().to_dict()
5            X_train.loc[:, col] = X_train[col].replace(S)
6            X_test.loc[:, col] = X_test[col].replace(S)
7
8 display(X_train['x1'].head())
9 display(X_test['x1'].head())
```

- **라인 1:** 특징과 라벨, 학습과 평가 데이터가 분리된 상태이므로 X_train과 y_train을 먼저 합칩니다.
- **라인 2:** X_train의 칼럼 목록(columns)과 칼럼별 자료형 목록(dtypes)을 zip 함수를 이용해 각각 col과 dtype으로 순회합니다.
- **라인 3 - 6:** dtype이 object인 col에 대해서 col에 따른 'y'의 평균으로 X_train과 X_test의 칼럼 col을 대체합니다.

```
357   0.497512
964   0.405797
337   0.113402
980   0.235294
455   0.113402
Name: x1, dtype: float64
652   0.497512
579   0.113402
836   0.113402
586   0.113402
226   0.113402
Name: x1, dtype: float64
```

2. 데이터 탐색 및 전처리

4 스케일링

예제 데이터 불러오기

스케일링에 사용할 데이터를 불러옵니다.

데이터 불러오기 예제

```
1  df = pd.read_csv("classification/glass.csv")
2  X = df.drop('y', axis = 1)
3  y = df['y']
4   X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 2022)
```

특징별 스케일 확인

특징별 스케일을 describe 메서드를 활용하여 확인해보겠습니다.

특징별 스케일 확인 예제

```
1  display(X_train.describe())
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9
count	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000
mean	1.518363	13.440179	2.707666	1.451238	72.623665	0.504136	8.940115	0.180554	0.039174
std	0.002738	0.770260	1.417633	0.514204	0.776014	0.725813	1.275246	0.500161	0.090792
min	1.512995	10.730000	0.000000	0.290000	69.810000	0.000000	5.871160	0.000000	0.000000
25%	1.516560	12.987675	2.197855	1.186393	72.275400	0.111780	8.238360	0.000000	0.000000
50%	1.517688	13.330150	3.484240	1.363745	72.755600	0.555795	8.609580	0.000000	0.000000
75%	1.519174	13.873788	3.609960	1.634188	73.044000	0.603922	9.236350	0.000000	0.052275
max	1.531242	15.790650	4.490000	3.500000	75.180400	6.210000	14.963360	3.150000	0.510000

- 특징별 스케일 차이가 매우 크다는 것을 알 수 있음
- x1은 1.512995부터 1.531242 사이의 값을 갖는 작은 스케일의 특징임
- x5는 69.81부터 75.1804까지 x1에 비해 약 50배 가까이 큰 스케일을 가짐

최소-최대 스케일링

최소-최대 스케일링은 사이킷런의 MinMaxScaler를 이용해 구현할 수 있습니다.

최소-최대 정규화 예제

```
1  from sklearn.preprocessing import MinMaxScaler
2  scaler = MinMaxScaler().fit(X_train)
3  Z_train = pd.DataFrame(scaler.transform(X_train), columns = X_train.columns)
4  Z_test = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns)
5  display(Z_train.describe())
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9
count	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000
mean	0.294187	0.535540	0.603044	0.361756	0.523921	0.081181	0.337537	0.057319	0.076813
std	0.150068	0.152206	0.315731	0.160188	0.144498	0.116878	0.140257	0.158781	0.178023
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.195381	0.446124	0.489500	0.279250	0.459072	0.018000	0.260355	0.000000	0.000000
50%	0.257179	0.513798	0.776000	0.334500	0.548488	0.089500	0.301183	0.000000	0.000000
75%	0.338639	0.621222	0.804000	0.418750	0.602190	0.097250	0.370118	0.000000	0.102500
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

- 모든 특징의 최솟값이 0으로 최댓값이 1로 변했음

표준화

표준화는 사이킷런의 StandardScaler를 이용해 구현할 수 있습니다.

표준화 예제

```
1  from sklearn.preprocessing import StandardScaler
2  scaler = StandardScaler().fit(X_train)
3  Z_train = pd.DataFrame(scaler.transform(X_train), columns = X_train.columns)
4  Z_test = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns)
5  display(Z_train.describe())
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9
count	1.600000e+02	1.600000e+02	1.600000e+02	1.600000e+02	1.600000e+02	1.600000e+02	1.600000e+02	1.600000e+02	1.600000e+02
mean	-3.455847e-14	-1.509903e-15	7.285839e-17	-2.081668e-17	1.426012e-14	1.221245e-16	-4.149459e-16	-8.985868e-17	-1.040834e-17
std	1.003140e+00	1.003140e+00	1.003140e+00	1.003140e+00	1.003140e+00	1.003140e+00	1.003140e+00	1.003140e+00	1.003140e+00
min	-1.966508e+00	-3.529571e+00	-1.915988e+00	-2.265410e+00	-3.637174e+00	-6.967617e-01	-2.414116e+00	-3.621250e-01	-4.328292e-01
25%	-6.604750e-01	-5.893132e-01	-3.607508e-01	-5.166751e-01	-4.501959e-01	-5.422714e-01	-5.520178e-01	-3.621250e-01	-4.328292e-01
50%	-2.473847e-01	-1.432947e-01	5.495160e-01	-1.706856e-01	1.705500e-01	7.139809e-02	-2.600070e-01	-3.621250e-01	-4.328292e-01
75%	2.971433e-01	5.647055e-01	6.384775e-01	3.569093e-01	5.433594e-01	1.379147e-01	2.330258e-01	-3.621250e-01	1.447460e-01
max	4.718043e+00	3.061110e+00	1.261208e+00	3.996845e+00	3.305045e+00	7.886029e+00	4.738033e+00	5.955617e+00	5.202050e+00

- MinMaxScaler의 결과와는 다르게 스케일이 작게는 -1.91에서 1.26 사이인 특징도 있고, 크게는 -4.32부터 5.20인 특징도 있음
- 그러나 전반적으로 특징의 스케일이 유사해졌으며, 다른 값에 비해 얼마나 크고 작은지를 알 수 있음

2. 데이터 탐색 및 전처리

5 재샘플링

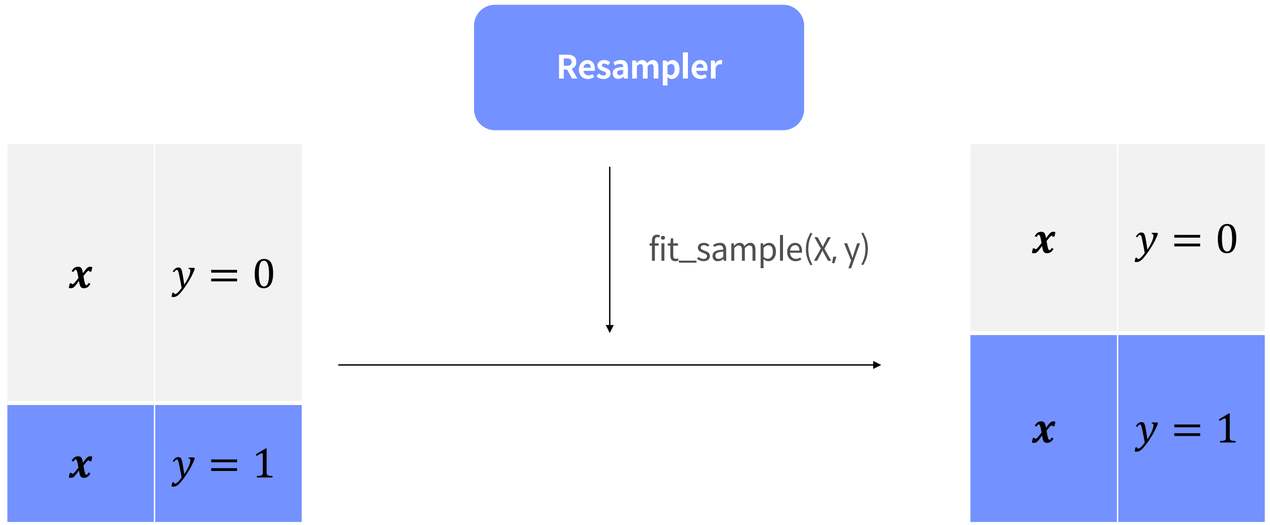
임밸런 패키지

임밸런은 다양한 재샘플링 클래스를 제공하는 패키지입니다.

임밸런 설치

```
$ pip install imblearn
```

사용 방법 개요



예제 데이터 불러오기

재샘플링에 사용할 데이터를 불러오고 value_counts 메서드를 사용해 라벨의 분포를 확인합니다.

데이터 불러오기 예제

```
df = pd.read_csv("classification/yeast-1_vs_7.csv")
X = df.drop('y', axis = 1)
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 2022)
```

라벨의 분포 확인 예제

```
display(y_train.value_counts())
```

```
0   320
1    24
Name: y, dtype: int64
```

- 다수 클래스의 샘플 수가 320, 소수 클래스의 샘플 수가 24로 약 13:1의 비율임을 알 수 있음

SMOTE

SMOTE는 임밸런의 SMOTE 클래스의 인스턴스로 구현할 수 있습니다.

SMOTE 예제

```
1  from imblearn.over_sampling import SMOTE
2  smote = SMOTE()
3   s_X_train, s_y_train = smote.fit_resample(X_train, y_train)
4  s_X_train = pd.DataFrame(s_X_train, columns = X_train.columns)
5  s_y_train = pd.Series(s_y_train)
6  display(s_y_train.value_counts())
```

- **라인 2**: SMOTE 클래스를 사용해 인스턴스 smote를 생성합니다.
- **라인 3**: fit_resample 메서드를 이용해 X_train과 y_train을 오버샘플링합니다. 그 결과는 각각 s_X_train과 s_y_train에 저장합니다.
- **라인 4 – 5**: 임밸런의 인스턴스도 사이킷런처럼 ndarray를 반환하므로, 특징 벡터와 라벨을 각각 데이터프레임과 시리즈로 다시 바꿔줍니다.

```
0   320
1   320
Name: y, dtype: int64
```

- 소수 클래스의 샘플이 24개에서 320개로 296개가 생성되면서 클래스의 균형이 맞음

NearMiss

NearMiss는 임밸런의 NearMiss 클래스로 만든 인스턴스를 사용해 다음과 같이 구현할 수 있습니다.

NearMiss 예제

```
1  from imblearn.under_sampling import NearMiss
2  nm = NearMiss()
3  s_X_train, s_y_train = nm.fit_resample(X_train, y_train)
4  s_X_train = pd.DataFrame(s_X_train, columns = X_train.columns)
5  s_y_train = pd.Series(s_y_train)
6  display(s_y_train.value_counts())
```

```
0   24
1   24
Name: y, dtype: int64
```

- 다수 클래스의 샘플이 320개에서 24개로 296개가 제거되면서 클래스의 균형이 맞음

2. 데이터 탐색 및 전처리

6 특징 선택

예제 데이터 불러오기

특징 선택에 사용할 데이터를 불러옵니다.

데이터 불러오기 예제

```
df = pd.read_csv("classification/wdbc.csv")
X = df.drop('y', axis = 1)
y = df['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 2022)
```

SelectKBest

SelectKbest 클래스는 이름에서 알 수 있듯이 클래스 관련성이 높은 K개의 특징을 선택하는 클래스입니다.

주요 인자

인자	설명
scoring_func	클래스 관련성 측정 함수 (sklearn.feature_selection라는 세부 패키지에 포함)
k	선택하는 특징 개수

특징 및 라벨 유형에 따른 적절한 클래스 관련성 측정 함수

통계량	특징 유형	라벨 유형	관련 함수
카이제곱 통계량	이진형	이진형 (분류)	chi2
상호 정보량	이진형	이진형 (분류)	mutual_info_classif
	연속형		
	이진형	연속형 (예측)	mutual_info_regression
	연속형		
F – 통계량	연속형	이진형 (분류)	f_classif
	연속형	연속형 (예측)	f_regression

주요 메서드

메서드	설명
fit(X, y)	X에 속한 특징의 y에 대한 클래스 관련성을 측정함
get_support()	선택된 특징과 같은 위치의 요소는 True를 그렇지 않은 요소는 False를 갖는 부울 배열을 반환

특징 선택

SelectKBest 클래스를 이용해 F-통계량(f_classif)이 큰 특징 10개를 선택해보겠습니다.

특징 선택 예제

```
1  from sklearn.feature_selection import *
2  selector = SelectKBest(f_classif, k = 10)
3  selector.fit(X_train, y_train)
4  selected_features = X_train.columns[selector.get_support()]
5  Z_train = X_train.loc[:, selected_features]
6  Z_test = X_test.loc[:, selected_features]
7  print(X_train.shape)
8  print(Z_train.shape)
```

- **라인 1:** SelectKBest 뿐만 아니라 f_classif까지 불러와야 하므로, sklearn.feature_selection 세부 클래스의 모든 함수를 불러옵니다.
- **라인 2:** f_classif를 기준으로 상위 10개의 특징을 선택하는 인스턴스 selector를 생성합니다.
- **라인 4:** selector.get_support()를 인덱스로 사용해 선택한 특징 목록을 selected_features에 저장합니다.
- **라인 5:** selected_features를 사용해 특징을 선택합니다.

```
(426, 30)
(426, 10)
```

- 특징 개수가 30개에서 10개로 줄었음