3.3 scikit-learn의 전처리 기능

scikit-learn 패키지도 preprocessing 서브패키지에서 다양한 전처리 기능을 제공한다. 여기에서는 회귀분석과 관련된 기능만을 살펴본다.

스케일링

스케일링은 자료 집합에 적용되는 전처리 과정으로 모든 자료에 선형 변환을 적용하여 전체 자료의 분포를 평균 0, 분산 1이 되도록 만드는 과정이다.스케일링은 자료의 오버플로우(overflow)나 언더플로우(underflow)를 방지하고 독립 변수의 공분산 행렬의 조건수(condition number)를 감소시켜 최적화 과정에서의 안정성 및 수렴 속도를 향상시킨다.

scikit-learn에서는 다음과 같은 스케일링 클래스를 제공한다.

- `StandardScaler(X)` : 평균이 0과 표준편차가 1이 되도록 변환.
- `RobustScaler(X)` : 중앙값(median)이 0, IQR(interquartile range)이 1이 되도록 변환.
- `MinMaxScaler(X)` : 최대값이 각각 1, 최소값이 0이 되도록 변환
- `MaxAbsScaler(X)` : 0을 기준으로 절대값이 가장 큰 수가 1또는 -1이 되도록 변환

사용방법은 다음과 같다.

(1) 학습용 데이터의 분포 추정: 학습용 데이터를 입력으로 하여 `fit` 메서드를 실행하면 분포 모수를 객체내에 저장 (2) 학습용 데이터 변환: 학습용 데이터를 입력으로 하여 `transform` 메서드를 실행하면 학습용 데이터를 변환 (3) 검증용 데이터 변환: 검증용 데이터를 입력으로 하여 `transform` 메서드를 실행하면 검증용 데이터를 변환

(1)번과 (2)번 과정을 합쳐서 `fit_transform` 메서드를 사용할 수도 있다.

In [1]:

```
X = (np.arange(9, dtype=np.float) - 3).reshape(-1, 1)  # -3부터 5까지의 분포
X = np.vstack([X, [100]])  # 아웃라이어(outlier) 값을 추가
pd.DataFrame(X).describe()
```

Out[1]:

	0
count	10.000000
mean	10.900000
std	31.412842
min	-3.000000
25%	-0.750000
50%	1.500000
75%	3.750000
max	100.000000

In [2]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
np.mean(X_scaled), np.std(X_scaled)
```

Out[2]:

(0.0, 1.0)

In [3]:

```
from sklearn.preprocessing import RobustScaler

robust_scaler = RobustScaler()
robust_scaler.fit(X)
X_robust_scaled = robust_scaler.transform(X)
np.mean(X_robust_scaled), np.std(X_robust_scaled)
```

Out[3]:

(2.088888888888889, 6.622408647636923)

전체 스케일링 결과는 비슷하지만 아웃라이어를 제거한 나머지 데이터의 분포는 로버스트 스케일링을 사용했을 때가 더 좋다.

In [4]:

```
pd.DataFrame(
    np.hstack([X_scaled[:-2], X_robust_scaled[:-2]]),
    columns=["Standard Scaler", "Robust Scaler"]
).describe()[3:]
```

Out[4]:

	Standard Scaler	Robust Scaler
min	-0.466430	-1.000000
25%	-0.407707	-0.611111
50%	-0.348983	-0.222222
75%	-0.290260	0.166667
max	-0.231537	0.555556

파이프라인

전처리용 객체는 scikit-learn의 파이프라인(pipeline) 기능을 이용하여 분류 모형과 합칠 수 있다. 예를 들어 스케일러와 선형회귀모형은 다음처럼 합친다.

In [5]:

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression

model = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression()),
])
```

파이프라인으로 결합된 모형은 원래의 모형이 가지는 fit , predict 메서드를 가지며 각 메서드가 호출되면 그에 따른 적절한 메서드를 파이프라인의 각 객체에 대해서 호출한다. 예를 들어 파이프라인에 대해 fit 메서드를 호출하면 전처리 객체에는 fit_transform 이 내부적으로 호출되고 분류 모형에서는 fit 메서드가 호출된다. 파이프라인에 대해 predict 메서드를 호출하면 전처리 객체에는 transform 이 내부적으로 호출되고 분류 모형에서는 predict 메서드가 호출된다.

다항 변환

PolynomialFeatures 입력값 x를 다항식으로 변환한다.

$$x \rightarrow [1, x, x^2, x^3, \cdots]$$

만약 열의 갯수가 두 개이고 2차 다항식으로 변환하는 경우에는 다음처럼 변환한다.

$$[x_1, x_2] \rightarrow [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$

다음과 같은 입력 인수를 가진다.

- degree : 차수
- interaction_only : True면 2차항에서 상호작용항만 출력
- include_bias : 상수항 생성 여부

In [6]:

```
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(6).reshape(3, 2)
X
```

Out[6]:

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

In [7]:

```
poly = PolynomialFeatures(2)
poly.fit_transform(X)
```

Out[7]:

```
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

In [8]:

```
poly = PolynomialFeatures(interaction_only=True)
poly.fit_transform(X)
```

Out[8]:

```
array([[ 1.,  0.,  1.,  0.],
       [ 1.,  2.,  3.,  6.],
       [ 1.,  4.,  5., 20.]])
```

일반 수학 변환

FunctionTransformer 입력값 x를 다항식이 아닌 사용자가 원하는 함수를 사용하여 변환한다.

$$x \quad \rightarrow \quad [f_1(x), f_2(x), f_3(x), \cdots]$$

In [9]:

```
from sklearn.preprocessing import FunctionTransformer


def kernel(X):
    x0 = X[:, :1]
    x1 = X[:, 1:2]
    x2 = X[:, 2:3]
    X_new = np.hstack([x0, 2 * x1, x2 ** 2, np.log(x1)])
    return X_new
```

In [10]:

```
X = np.arange(12).reshape(4, 3)
X
```

Out[10]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [11]:

```
kernel(X)
```

Out[11]:

```
array([[ 0.        ,  2.        ,  4.        ,  0.        ],
       [ 3.        ,  8.        , 25.        ,  1.38629436],
       [ 6.        , 14.        , 64.        ,  1.94591015],
       [ 9.        , 20.        , 121.       ,  2.30258509]])
```

In [12]:

```
FunctionTransformer(kernel).fit_transform(X)
```

Out[12]:

```
array([[  0.        ,   2.        ,   4.        ,   0.        ],
       [  3.        ,   8.        ,  25.        ,   1.38629436],
       [  6.        ,  14.        ,  64.        ,   1.94591015],
       [  9.        ,  20.        , 121.        ,   2.30258509]])
```