

머신러닝 알고리즘을 위한 데이터 준비

Chapter 2-5

머신러닝을 위한 데이터 준비 조건

- 결측치가 있으면 안된다.
- 텍스트나 범주형 데이터를 처리해야 한다.(수치형 변환)

데이터 정제 – 결측치 제거

- 결측치 값을 제거 한다.
 - `Df.dropna(subset=['col'])`
- 전체 특성을 삭제한다.(컬럼 삭제)
 - `Df.drop('col',axis=1)`
- 특정 값으로 결측치를 대체한다.(평균, 중앙값 etc..)
 - `Df['col'].fillna(median,inplace=True)`

데이터 정제 – 결측치 제거

- Sklearn의 Imputer 모듈
- 핸드온에 나와있는 `sklearn.preprocessing.Imputer`는 버전업이 되면서 존재하지 않음
- 따라서 `SimpleImputer` 모듈을 사용

Sklearn.preprocessing.SimpleImputer

- 결측치를 처리해주는 모듈이다.
- 사용 예시

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp_mean.fit([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])
SimpleImputer()
>>> X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]
>>> print(imp_mean.transform(X))
[[ 7.  2.  3.]
 [ 4.  3.5  6.]
 [10.  3.5  9.]]
```

Strategy의 옵션을 바꿔가며 대체할 값을 정할 수 있다. (default='mean')

Sklearn.preprocessing.SimpleImputer

- 사용하는 이유

- 새로운 데이터에 어떤 값이 누락될지 모르므로 imputer로 각 특성의 대푯값을 저장한 후 transform으로 적용해줘야 하기 때문

```
1 imputer.statistics_ # 각 특성의 중간값을 계산해서 statistics_ 속성에 저장
array([-118.51 ,  34.26 ,  29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,  3.5409])
```

```
1 housing_num.median().values
array([-118.51 ,  34.26 ,  29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,  3.5409])
```

sklearn 설계 철학

- 일관성 : 모든 객체가 일관되고 단순한 인터페이스 공유
 - Estimator : 모델의 파라미터들을 추정하는 객체(지도 학습에서 학습을 담당. Fitting model 생각하면 됨, 모델을 설계한다.)
 - Transformer : 데이터셋 변환기
 - Predictor : 예측값 반환
- 검사 가능 : 하이퍼파라미터 검사 가능. `Imputer.statistics_` 처럼 값 확인도 가능
- 클래스 남용 방지 : numpy array or sparse matrix로 반환
- 조합성 : 기존의 구성요소를 최대한 재사용. 여러 개를 합칠 수 있음
- 합리적인 기본값 : default 값이 웬만하면 있다.

텍스트와 범주형 특성 다루기

- LabelEncoding, OneHotEncoding etc..
- Df['col'].factorize() -> 라벨 인코딩 형식으로 나옴
- LabelEncoding 이란?
 - 범주형 데이터를 숫자로 바꿔주는 것. Ex) 남:0 여:1
 - From sklearn.preprocessing import LabelEncoding
- OneHotEncoding 이란?
 - 범주형 데이터를 숫자로 바꿔 줌. 하지만! **한 특성이 1이면 나머지 특성은 0**
 - From sklearn.preprocessing import OneHotEncoding

OneHotEncoding vs LabelEncoding

성별(gender)	
M (0)	F (1)
1	0
1	0
1	0
0	1
0	1

문자형 분리 자체에 의미를 둘 때
Ex) male, female

→ Onehot 할 때 문제점 :
다중공선성 발생 가능

State (Nominal Scale)	State (Label Encoding)
Maharashtra	3
Tamil Nadu	4
Delhi	0
Karnataka	2
Gujarat	1
Uttar Pradesh	5

숫자의 크기가 의미가 있을 때 사용
예) AGE 카테고리형 변수를 바꿀 때

변환기 만들기

- `Sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin` 을 상속하며 변환기를 만들 수 있다.
- `BaseEstimator` : class의 매개변수를 가져오고 설정할 수 있음
- `TransformerMixin` : `transform` 함수를 구현하기 위한 모듈
- `Get_params()` 와 `set_params()`는 파이프라인과 그리드 탐색에 꼭 필요한 method임. 따라서 꼭 `BaseEstimator`를 상속시켜줘야 됨.

변환기 만들기

```
from sklearn.base import BaseEstimator, TransformerMixin

# 컬럼 인덱스
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kwargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                          bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

두 개의 1차원 배열을 칼럼으로
세로로 붙여서 2차원 배열 만들기

Sacling

Why do we have to Scaling?

데이터의 값이 너무 크거나 혹은 작은 경우에 모델 알고리즘 학습과정에서 0으로 수렴하거나 무한으로 발산해버릴 수 있기 때문

종류

1. From Sklearn.preprocessing import **StandardScaler**

: Z-정규화 -> 각 변수들의 평균을 0, 분산을 1로 변경

-> 이상치가 있는 경우 균형 잡힌 척도 보장 X

$$z = \frac{X - \mu}{\sigma}$$

2. From sklearn.preprocessing import **MinMaxScaler**

: 모든 변수들이 0~1 사이의 값을 갖게 만들

-> 이상치 값에 매우 민감

$$X_{\text{new}} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

3. From sklearn.preprocessing import **MaxAbsScaler**

: 모든 변수들이 절대값 1 사이 즉, -1~1 사이의 값을 갖게 됨

-> 큰 이상치에 민감. 상대적으로 standardscaler와 minmaxscaler에 비해 덜 민감

4. From sklearn.preprocessing import **RobustScaler**

: 평균대신 중앙값을 사용. 중위수를 뺀 다음 사분위간 범위(IQR)로 나눔

-> 이상치 영향 최소화

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

변환 파이프라인

- Pipeline이란? 변환을 순서대로 처리할 수 있도록 하는 모듈
- 즉, imputer(결측치 처리) -> onehotencoding(카테고리 처리) -> StandardScaling(정규화)를 한꺼번에 진행할 수 있게 도와줌

- 사용방법:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([('imputer', SimpleImputer(np.nan, strategy='median')),
                          ('attrs_adder', CombinedAttributesAdder()),
                          ('std_scaler', StandardScaler())])
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

변환 파이프라인

- 각 변수마다 다르게 파이프라인을 만들 수 있음
- 각 컬럼마다 다르게 파이프라인 하는 법
- Pipeline에 'selector'이름을 주고(이름 노상관) 값 선택(type: np.array)
- Pipeline 결합하기: FeatureUnion(transformer_list=['pipe1_name':pipe1, 'pipe2_name' : pipe2])
- 마지막에 fit_transform