

데이터 전처리 : 데이터의 품질을 올린다! (왜? 그래야 학습이 잘 되겠지?)

데이터 전처리 과정

- 데이터 실수화 : 문자열과 같은 데이터를 컴퓨터가 이해할 수 있게 실수화 하는 것.
- 불완전한 데이터 제거 : NULL, NAN, NA 같은 빈 데이터를 제거.
- 잡음이 섞인 데이터 제거 : 잡음이라고 판단되는 값들을 제거 (ex 음수인 가격, 과도하게 큰 연령데이터 등).
- 모순된 데이터 제거 : 남성 중 주민번호가 '2'로 시작하는 경우와 같이 모순적인 값 제거 또는 수정.
- 불균형 데이터 해결 : 특정 클래스만 많은 (ex 남성 데이터 : 여성 데이터 = 99 : 1)
 - 과소표집(undersampling), 과대표집(oversampling) 등의 기법으로 교정

데이터 전처리 주요 기법

- 데이터 실수화 (Data Vectorization) : 범주형, 텍스트, 이미지 자료 등을 실수 형태로 전환.
- 데이터 정제 (Data Cleaning) : 없는 데이터는 채우고, 잡음 데이터는 제거하고, 모순 데이터를 올바른 데이터로 교정하는 것.
- 데이터 통합 (Data Integration) : 여러 개의 데이터 파일을 하나로 합치는 과정
- 데이터 축소 (Data Reduction) : 데이터가 과도하게 크면 분석 및 학습이 오래 걸리고 비효율 적이기 때문에 데이터의 수를 줄이거나(Simpling) 데이터 차원을 축소하는 작업
- 데이터 변환 (Data transformation) : 데이터를 정규화(normalization, standardization)하거나, 로그를 씌우거나, 평균값을 계산하여 사용하거나, 사람 나이 등을 10대, 20대, 30대 등으로 구간 화 하는 작업
- 데이터 균형 (Data balancing) : 특정 클래스의 관측치가 다른 클래스에 비해 매우 낮을 경우 샘플링(과소표집, 과대표집)을 통해 클래스 비율을 맞추는 작업

데이터 실수화 (Data Vectorization) : 범주형, 텍스트, 이미지 자료 등을 실수 형태로 전환.


- 2차원 자료의 예시
 - [n_samples, n_features], 2차원 자료는 2차원 행렬 또는 2차원 텐서라고 불림
- 자료의 유형

- 연속형 자료 (Continuous data)의 실수화 : 사람들의 나이 같은 데이터를 구간을 나누어 mapping을 한다.

- 범주형 자료 (Categorical data)의 실수화 :

- One-hot encoding 을 이용한 데이터 실수화 : 0, 1을 행렬의 형태로 나타낸다.

| id | City |
|----|-------|
| 1 | Seoul |
| 2 | Dubai |
| 3 | LA |



| id | City1 | City2 | City3 |
|----|-------|-------|-------|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |

Scikit-learn의 DictVectorizer 함수 : 위와 같은 One-hot encoding을 해준다. Input argument의 디폴트는 sparse=True이다. 이걸 아래서 설명

```

1 # 범주형 자료의 수량화 -----
2 x=[{'city':'seoul','temp':10.0}, {'city':'Dubai','temp':33.5}, {'city':'LA','temp':20.0}]
3 x

[{'city': 'seoul', 'temp': 10.0},
 {'city': 'Dubai', 'temp': 33.5},
 {'city': 'LA', 'temp': 20.0}]

[ ] 1 from sklearn.feature_extraction import DictVectorizer
2 vec=DictVectorizer(sparse=False)
3 vec.fit_transform(x) # x를 범주형 수량화 자료로 변환

array([[ 0.,  0.,  1., 10. ],
       [ 1.,  0.,  0., 33.5],
       [ 0.,  1.,  0., 20. ]])

```

- 희소행렬(Sparse Matrix) : 위에서 봤듯이 One-hot encoding을 하면 행렬의 값이 대부분 0인데, 이는 메모리 낭비이기 때문에 이를 COO 표현식과 CSR 표현식으로 해결하는데 그러면 행렬처럼 우리 눈으로 볼 수는 없다.
- CSR 표현식 (Compressed Sparse Row) : COO 형식에 비해 메모리 및 속도에서 효율적임. COO는 데이터 값만 저장한 배열과 그 데이터 값의 행렬 좌표를 저장한 배열을 만드는 방식. CSR은 이 좌표 배열을 또 최적화 한 것임.

```

1 vec1=DictVectorizer(sparse=True) # 메모리를 줄이기 위해 sparse=True
2 x1=vec1.fit_transform(x)
3 x1

<3x4 sparse matrix of type '<class 'numpy.float64''>'
with 6 stored elements in Compressed Sparse Row format>

```

- 텍스트 자료 (Text data)의 실수화 :

- 단어의 출현 횟수를 이용한 데이터 실수화 (TF) : 아래처럼 단어의 빈도 수로 표현 하는 방법이다. 그러나 영어의 관사 a, the 처럼 의미가 없으나 빈도수가 높은 경우가 있다. 그래서 TF-IDF 기법을 이용한다.

| id | |
|----|-------------------|
| 1 | 떴다 떴다 비행기 날아라 날아라 |
| 2 | 높이 높이 날아라 우리 비행기 |
| 3 | 내가 만든 비행기 날아라 날아라 |
| 4 | 멀리 멀리 날아라 우리 비행기 |

| id | 날아라 | 내가 | 높이 | 떴다 | 만든 | 멀리 | 비행기 | 우리 |
|----|-----|----|----|----|----|----|-----|----|
| 1 | 2 | 0 | 0 | 2 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 2 | 0 | 0 | 0 | 1 | 1 |
| 3 | 2 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 2 | 1 | 1 |

CountVectorizer의 fit_transform()

```
[ ] 1 # 텍스트 자료의 수량화 =====
2 text=[ '떴다 떴다 비행기 날아라 날아라',
3        '높이 높이 날아라 우리 비행기',
4        '내가 만든 비행기 날아라 날아라',
5        '멀리 멀리 날아라 우리 비행기' ]
6 text
```

```
[ '떴다 떴다 비행기 날아라 날아라',
  '높이 높이 날아라 우리 비행기',
  '내가 만든 비행기 날아라 날아라',
  '멀리 멀리 날아라 우리 비행기' ]
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 vec2 = CountVectorizer()# default는 sparse=True
3 t=vec2.fit_transform(text).toarray() # sparse=True를 풀고 text를 수량화 배열 자료로 변환
4 import pandas as pd
5 t1=pd.DataFrame(t, columns=vec2.get_feature_names())
6 t1
```

```
날아라  내가  높이  떴다  만든  멀리  비행기  우리
0      2    0    0    2    0    0    1    0
1      1    0    2    0    0    0    1    1
2      2    1    0    0    1    0    1    0
3      1    0    0    0    0    2    1    1
```

- fit_transform()
- toarray() : CSR 표현의 압축을 풀기 위해 사용

- TF-IDF (Term Frequency Inverse Document Frequency) : 각 단어의 가중치를 구해서 빈도에 곱해준다.

tf(d,t) : 특정 문서 d에서 특정 단어 t의 등장 횟수.

df(t) : 특정 단어 t가 등장한 문서의 수.

idf(d, t) : df(t)에 반비례. 즉 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단.

$$idf(d, t) = \log\left(\frac{n}{1 + df(t)}\right)$$

log를 쓰는 이유는 전체 문서의 수(n)이 커질 경우 아래의 예시

처럼 희귀 단어들에 엄청난 가중치가 부여될 수 있기 때문이다.

$$df(d, t) = \log(n/df(t))$$

$$n = 1,000,000$$

| 단어 t | $df(t)$ | $idf(d, t)$ |
|--------|-----------|-------------|
| word1 | 1 | 6 |
| word2 | 100 | 4 |
| word3 | 1,000 | 3 |
| word4 | 10,000 | 2 |
| word5 | 100,000 | 1 |
| word6 | 1,000,000 | 0 |

1렇다면 log를 사용하지 않으면 idf의 값이 어떻게 커지는지 보겠습니다.

$$df(d, t) = n/df(t)$$

$$n = 1,000,000$$

| 단어 t | $df(t)$ | $idf(d, t)$ |
|--------|-----------|-------------|
| word1 | 1 | 1,000,000 |
| word2 | 100 | 10,000 |
| word3 | 1,000 | 1,000 |
| word4 | 10,000 | 100 |
| word5 | 100,000 | 10 |
| word6 | 1,000,000 | 1 |

TfidfVectorizer의 fit_transform() 함수

```
[ ] 1 from sklearn.feature_extraction.text import TfidfVectorizer
    2 tfidf=TfidfVectorizer()
    3 x2=tfidf.fit_transform(text).toarray() # 높은 빈도는 낮은 가중치, 낮은 빈도는 높은 가중치
    4 x3=pd.DataFrame(x2,columns=tfidf.get_feature_names())
    5 x3
```

| | 날아라 | 내가 | 높이 | 땀다 | 만든 | 멀리 | 비행기 | 우리 |
|---|----------|----------|----------|----------|----------|----------|----------|----------|
| 0 | 0.450735 | 0.000000 | 0.000000 | 0.86374 | 0.000000 | 0.000000 | 0.225368 | 0.000000 |
| 1 | 0.229589 | 0.000000 | 0.87992 | 0.000000 | 0.000000 | 0.000000 | 0.229589 | 0.346869 |
| 2 | 0.569241 | 0.545415 | 0.000000 | 0.000000 | 0.545415 | 0.000000 | 0.284620 | 0.000000 |
| 3 | 0.229589 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.87992 | 0.229589 | 0.346869 |

데이터 변환 (Data Transformation) :

- 데이터 변환의 필요성 : 머신러닝은 데이터가 가진 특성(Feature)들을 비교하여 데이터 패턴을 찾음. 이 때 데이터가 가진 특성 간 스케일 차이가 심하면 패턴을 찾는데 문제가 발생함.
- 데이터 변환 법 :

표준화 (Standardization) : 평균값을 기준으로 1~0의 분포

$$x_{std} = \frac{x - \text{mean}(x)}{sd(x)}$$

정규화 (Normalization) : 최대값과 최소값을 기준으로 1~0의 분포

$$x_{nor} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

정규화가 표준화보다 유용함. 단, 데이터 특성이 bell-shape 이거나 이상치가 있을 경우에는 표준화가 유용함.

데이터 정제 (Data Cleaning) :

- 결측 데이터 채우기 (Empty Values) : np.nan, np.NaN, None 과 같은 빈 데이터를 평균 (mean), 중위수(median), 최빈수(most frequent value)등 으로 대체한다. 사용가능 함수는

sklearn의 Imputer() : 입력 인자로 평균, 중위수, 최빈수를 선택한다. 버전에 따라 함수가 다르다. 최신 : Simpleimputer()

```
[ ] 1 # 결측자료 대체 =====
2 x_miss=np.array([[1,2,3,None],[5,np.NaN,7,8],[None,10,11,12],[13,np.nan,15,16]])
3 x_miss

array([[1, 2, 3, None],
       [5, nan, 7, 8],
       [None, 10, 11, 12],
       [13, nan, 15, 16]], dtype=object)
```

```
1 from sklearn.preprocessing import Imputer
2 im=Imputer(strategy='mean')
3 im.fit_transform(x_miss) # 열의 평균값으로 대체

C:\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:66: DeprecationWarning: Class Imputer is deprecated; Im
warnings.warn(msg, category=DeprecationWarning)
array([[ 1.         ,  2.         ,  3.         , 12.         ],
       [ 5.         ,  6.         ,  7.         ,  8.         ],
       [ 6.33333333, 10.         , 11.         , 12.         ],
       [13.         ,  6.         , 15.         , 16.         ]])
```

데이터 통합 (Data Integration) : 여러 개의 데이터 파일을 하나로 합친다.

Pandas의 merge() 함수 사용

```
[13] 4 import pandas as pd
5 df1=pd.read_csv("rossmann-store-sales-train.csv",engine='python')
6 print(df1.shape)
7 type(df1)
```

데이터 #1 로드

```
(1017209, 9)
pandas.core.frame.DataFrame
```

```
1 df1.head()
```

| | Store | DayOfWeek | Date | Sales | Customers | Open | Promo | StateHoliday | SchoolHoliday |
|---|-------|-----------|------------|-------|-----------|------|-------|--------------|---------------|
| 0 | 1 | 5 | 2015-07-31 | 5263 | 555 | 1 | 1 | 0 | 1 |
| 1 | 2 | 5 | 2015-07-31 | 6064 | 625 | 1 | 1 | 0 | 1 |
| 2 | 3 | 5 | 2015-07-31 | 8314 | 821 | 1 | 1 | 0 | 1 |
| 3 | 4 | 5 | 2015-07-31 | 13995 | 1498 | 1 | 1 | 0 | 1 |
| 4 | 5 | 5 | 2015-07-31 | 4822 | 559 | 1 | 1 | 0 | 1 |

데이터 #2 로드 & 통합

```
[15] 1 df2=pd.read_csv("rossmann-store-sales-store.csv",engine='python')
      2 df2.shape
```

```
(1115, 10)
```

```
1 df2.head()
```

| | Store | StoreType | Assortment | CompetitionDistance | CompetitionOpenSinceMonth | CompetitionOpenSinceYear | Promo2 | Promo2Since1 |
|---|-------|-----------|------------|---------------------|---------------------------|--------------------------|--------|--------------|
| 0 | 1 | c | a | 1270.0 | 9.0 | 2008.0 | 0 | |
| 1 | 2 | a | a | 570.0 | 11.0 | 2007.0 | 1 | |
| 2 | 3 | a | a | 14130.0 | 12.0 | 2006.0 | 1 | |
| 3 | 4 | c | c | 620.0 | 9.0 | 2009.0 | 0 | |
| 4 | 5 | a | a | 29910.0 | 4.0 | 2015.0 | 0 | |

```
[17] 1 df=pd.merge(df1,df2,on='Store')
      2 df.shape
```

```
(1017209, 18)
```

DataFrame이라는 pandas의 자료형 두 개를 merge 하여 shape로 크기를 확인.

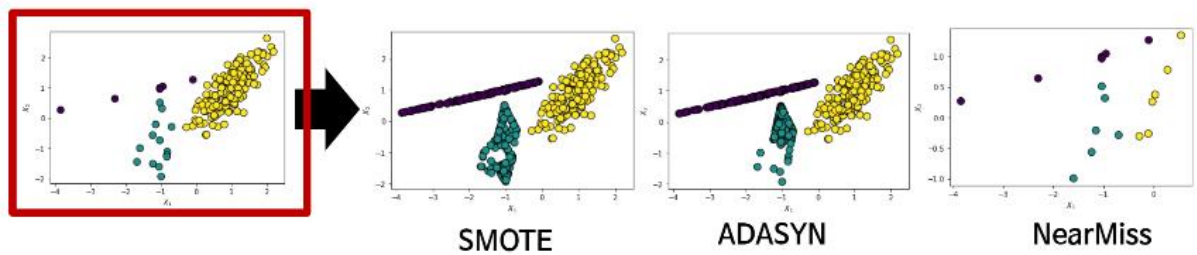
두 데이터프레임 모두 Store이라는 공통된 열을 가지고 있기 때문에 on='Store'을 해주면 Store 값에 맞는 애들에 붙는다. 공통된 값이 없으면 생략된다.

How='outer' 하면 없는 애에는 NaN이 붙는다.

df1.shape : (1017209, 9) + df2.shape : (1115, 9) = df.shape : (1017209, 18)

데이터 불균형 (Data Imbalance) : 머신러닝의 목적이 분류일 때 특정 클래스의 관측치가 다른 클래스에 비해 매우 낮게 나타나면 이를 과소표집이나 과대표집 기법으로 맞춰준다. 일반적으로 과소표집보다 과대표집이 유용하다. 의사결정나무와 앙상블은 상대적으로 불균형 자료에 강인한 특성을 보인다.

- 과소표집(undersampling) : 다수클래스의 표본을 임의로 학습데이터로부터 제거하는 것. (ex : NearMiss)
- 과대표집(oversampling) : 소수클래스의 표본을 복제하여 이를 학습데이터에 추가하는 것. (ex : SMOTE, ADASYN)

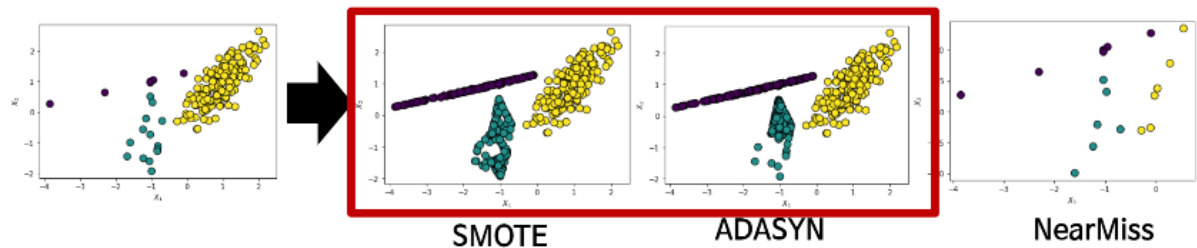


```
[62] 1 from collections import Counter
      2 from sklearn.datasets import make_classification
      3 from imblearn.over_sampling import SMOTE, ADASYN
      4
      5 # n_feature 변경, n_informative와 n_redundant 추가
      6 X, y = make_classification(n_classes=3, weights=[0.03, 0.07, 0.9], n_features=2, n_informative=2, n_redundant=0,
      7                           n_clusters_per_class=1, n_samples=200, random_state=10)
      8
      9
     10 print('Original dataset shape %s' % Counter(y))
```

Original dataset shape Counter({2: 180, 1: 14, 0: 6})

```
1 import matplotlib.pyplot as plt
2
3 plt.scatter(X[:, 0], X[:, 1], marker='o', c=y,
4             s=100, edgecolor='k', linewidth=1)
5
6 plt.xlabel("$X_1$")
7 plt.ylabel("$X_2$")
8 plt.show()
```

노란색이 불균형하게 많은 걸 볼 수 있다. Matplotlib의 scatter() 함수.



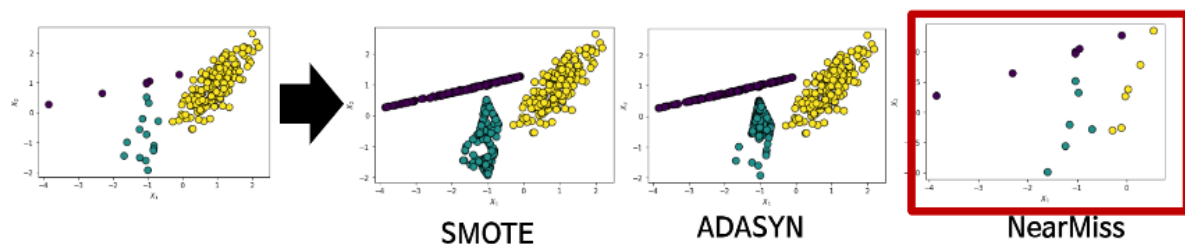
```
[51] 1 sm = SMOTE(random_state=42)
      2 X_res, y_res = sm.fit_resample(X, y)
      3 print('Resampled dataset shape %s' % Counter(y_res))

[52] 1 import matplotlib.pyplot as plt
      2
      3 plt.scatter(X_res[:, 0], X_res[:, 1], marker='o', c=y_res,
      4             s=100, edgecolor='k', linewidth=1)
      5
      6 plt.xlabel("$X_1$")
      7 plt.ylabel("$X_2$")
      8 plt.show()
```

```
[63] 1 ada=ADASYN(random_state=0)
      2 X_syn, y_syn=ada.fit_resample(X,y)
      3 print('Resampled dataset shape from ADASYN %s' % Counter(y_syn))
```

```
[64] 1 import matplotlib.pyplot as plt
      2
      3 plt.scatter(X_syn[:, 0], X_syn[:, 1], marker='o', c=y_syn,
      4             s=100, edgecolor='k', linewidth=1)
      5
      6 plt.xlabel("$X_1$")
      7 plt.ylabel("$X_2$")
      8 plt.show()
```

SMOTE와 ADASYN을 돌리면 보라색과 초록색 값이 많아진 걸 볼 수 있다. Fit_resample() 함수.



```

1 from imblearn.under_sampling import NearMiss
2
3 # define the undersampling method
4 undersample = NearMiss(version=3, n_neighbors_ver3=3)
5 # transform the dataset
6 X_Under, y_Under = undersample.fit_resample(X, y)

[60] 1 import matplotlib.pyplot as plt
2
3 plt.scatter(X_Under[:, 0], X_Under[:, 1], marker='o', c=y_Under,
4            s=100, edgecolor="k", linewidth=1)
5
6 plt.xlabel("$X_1$")
7 plt.ylabel("$X_2$")
8 plt.show()

```

NearMiss를 돌리면 다 같이 줄었다. Under_sampling() 함수.