

## 2. 전처리와 최적화

---

with 사이킷런 (scikit learn)

# 데이터 전처리

---

데이터의 품질은 데이터 분석의 90%를 좌우한다

# 데이터 전처리 정의

- 데이터 전처리 정의
  - 데이터의 품질을 올리는 과정
- 데이터 전처리 과정
  - 데이터 실수화: 컴퓨터가 이해할 수 있는 값으로의 변환
  - 불완전한 데이터 제거: NULL, NA, NAN 값의 제거
  - 잡음 섞인 데이터 제거
    - 가격 데이터에 있는 (-) 값 제거
    - 연령 데이터 중 과도하게 큰 값 제거
      - 예) 나이 값으로 200, 300, 400 등의 값이 존재하는 경우
  - 모순된 데이터 제거: 남성 데이터 중 주민번호가 '2'로 시작하는 경우
  - 불균형 데이터 해결
    - 과소표집(undersampling), 과대표집(oversampling)

# 데이터 전처리 기법

- 데이터 전처리의 주요 기법

- ① **데이터 실수화** (Data Vectorization)

- 범주형 자료, 텍스트 자료, 이미지 자료 등을 실수로 구성된 형태로 전환하는 것

- ② **데이터 정제** (Data Cleaning)

- 없는 데이터는 채우고, 잡음 데이터는 제거하고, 모순 데이터를 올바른 데이터로 교정하는 것

- ③ **데이터 통합** (Data Integration)

- 여러 개의 데이터 파일을 하나로 합치는 과정

- ④ **데이터 축소** (Data Reduction)

- 데이터가 과도하게 큰 경우, 분석 및 학습에 시간이 오래 걸리고 비효율적이기 때문에 데이터의 수를 줄이거나(Sampling), 데이터 차원을 축소하는 작업

# 데이터 전처리 기법

- 데이터 전처리의 주요 기법

- ⑤ 데이터 변환 (Data Transformation)

- 데이터를 정규화 하거나, 로그를 씌우거나, 평균값을 계산하여 사용하거나, 사람 나이 등을 10대, 20대, 30대 등으로 구간화 하는 작업

- ⑥ 데이터 균형 (Data Balancing)

- 특정 클래스의 관측치가 다른 클래스에 비해 매우 낮을 경우 샘플링을 통해 클래스 비율을 맞추는 작업

# 데이터 실수화(Data Vectorization)

- 정의

- 범주형 자료, 텍스트 자료, 이미지 자료 등을 실수로 구성된 형태로 전환 하는 것
- 2차원 자료의 예시
  - $[n_{\text{sample}}, n_{\text{features}}]$ 
    - $n_{\text{sample}}$  : 샘플 수
    - $n_{\text{features}}$  : 특성의 수
  - 2차원 자료는 행렬 혹은 2차원 텐서라 불림

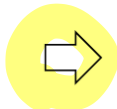
- 자료의 유형

- 연속형 자료 (Continuous data)
- 범주형 자료 (Categorical data)
- 텍스트 자료 (Text data)

# 범주형 자료의 실수화

- One-hot encoding 을 이용한 데이터 실수화

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
  - 범주형 자료의 실수화 하는 함수
  - Input argument : 디폴트 옵션 Sparse=True

```
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. ]])
```

# 범주형 자료의 실수화

## ▪ 희소행렬(Sparse Matrix)

- 행렬의 값이 대부분 0인 경우를 가리키는 표현
- 희소 행렬은 프로그램 시 불필요한 0 값으로 인해 메모리 낭비가 심함
- 행렬의 크기가 커서 연산시 시간도 많이 소모됨
- COO표현식과 CSR표현식을 통해 문제 해결 가능

## ▪ CSR 표현식 (Compressed Sparse Row)

- COO형식에 비해 메모리가 적게 들고 빠른 연산이 가능함

```
[2] 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. ]])
```

```
▶ 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 자료의 실수화

- 단어의 출현 횟수를 이용한 데이터 실수화

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

- 출현 횟수가 정보의 양과 비례하는 것은 아님. 때문에 TF-IDF 기법을 이용해야 함
  - TF-IDF (Term Frequency Inverse Document Frequency)
  - 자주 등장하여 분석에 의미를 갖지 못하는 단어의 중요도를 낮추는 기법
    - 예) The, a 등의 관사

# 텍스트 자료의 실수화

## 단어의 출현 횟수를 이용한 데이터 실수화

```
[ ] 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
```

→ 해줘야 하는 애매한 위치를 보아지 않음

	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

# 텍스트 자료의 실수화

- 단어의 출현 횟수를 이용한 데이터 실수화
  - TF-IDF
    - 가중치 재계산
    - 높은 빈도에 낮은 가중치를, 낮은 빈도에 높은 가중치를

```
[ ] 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	뒀다	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
- 데이터가 가진 특성 간 스케일 차이가 심하면 패턴을 찾는데 문제가 발생함

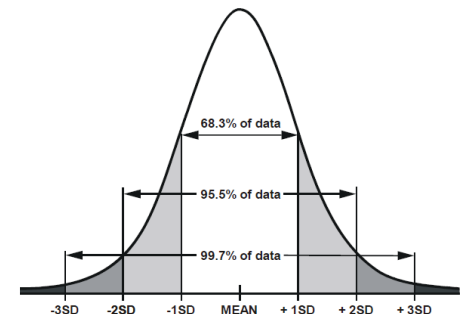
## ■ 데이터 변환 법

- 표준화 (Standardization)

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

- 정규화 (Normalization)

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



Bell shape 분포

- <sup>일반적</sup> 정규화가 표준화보다 유용함. 단, 데이터 특성이 bell-shape 이거나 이상치가 있을 경우에는 표준화가 유용함

# 데이터 정제 (Data Cleaning)

www.csv

## 결측 데이터 채우기 (Empty Values)

10/2/0/3/0

- 결측 데이터: np.nan, npNaN, None
- 평균(mean), 중위수(median), 최빈수(most frequent value)로 대체하는 기법 사용
- 사용가능함수
  - sklearn의 Imputer(): 입력인자로 평균, 중위수, 최빈수 선택
  - sklearn의 dropna(), fillna(): 결측 데이터 0으로 채우기

```
[ ] 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)



## 데이터 통합이란?

- 여러 개의 데이터 파일을 하나로 합치는 과정
- 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)
```

```
(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 #1 로드

~> # include 다 같은 역할

~> 땀지 않아봐도  
뭔가.

# 데이터 통합 (Data Integration)

- 데이터 통합이란?
  - 여러 개의 데이터 파일을 하나로 합치는 과정
  - Pandas의 merge() 함수 사용

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	Promo2Since
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)

# 데이터 통합 (Data Integration)

- 데이터 통합이란?

- 여러 개의 데이터 파일을 하나로 합치는 과정
- Pandas의 `df.dtypes` → 변수의 자료 타입 확인

```
[18] 1 df.dtypes
```

```
Store                int64
DayOfWeek             int64
Date                 object
Sales               int64
Customers           int64
Open                int64
Promo               int64
StateHoliday         object
SchoolHoliday        int64
StoreType            object
Assortment           object
CompetitionDistance  float64
CompetitionOpenSinceMonth float64
CompetitionOpenSinceYear float64
Promo2              int64
Promo2SinceWeek     float64
Promo2SinceYear     float64
PromoInterval        object
dtype: object
```

```
[19] 1 print(len(df['Store'].unique()))
     2 print(len(df['Date'].unique()))
     3 print(df['DayOfWeek'].value_counts())
```

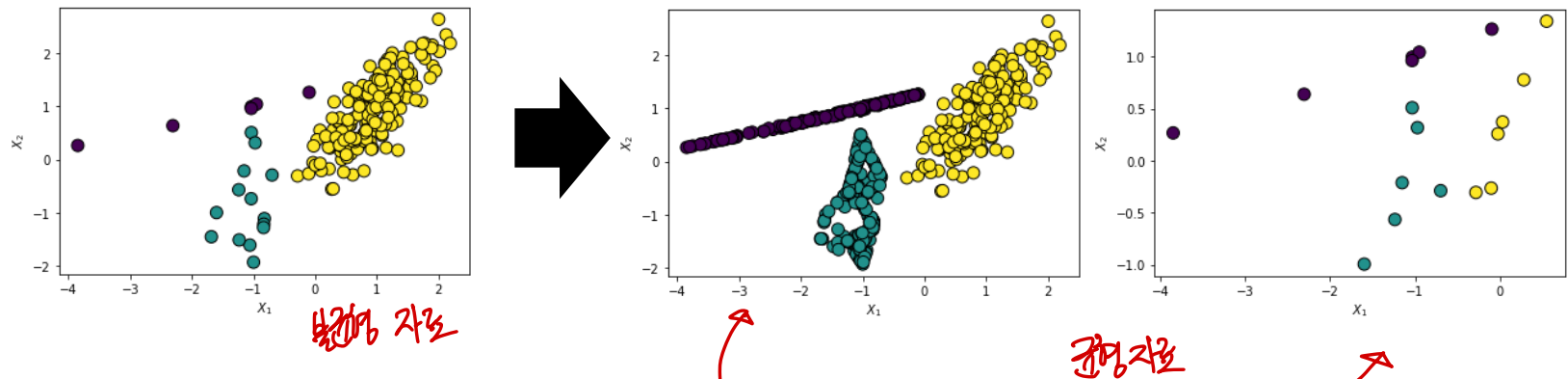
```
1115
942
5    145845
4    145845
3    145665
2    145664
7    144730
6    144730
1    144730
Name: DayOfWeek, dtype: int64
```



# 데이터 불균형 (Data Imbalance)

## ■ 데이터 불균형이란?

- 머신러닝의 목적이 분류 일 때, 특정 클래스의 관측치가 다른 클래스에 비해 매우 낮게 나타나면 이러한 자료를 불균형자료라고 함

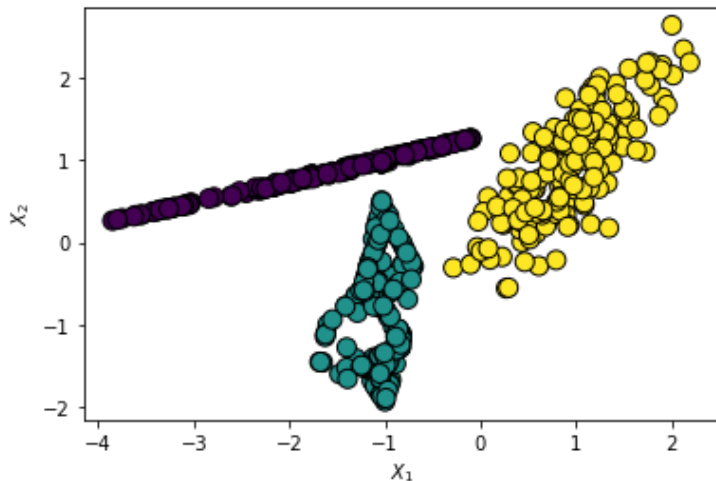


## ■ 데이터 불균형 해소 기법

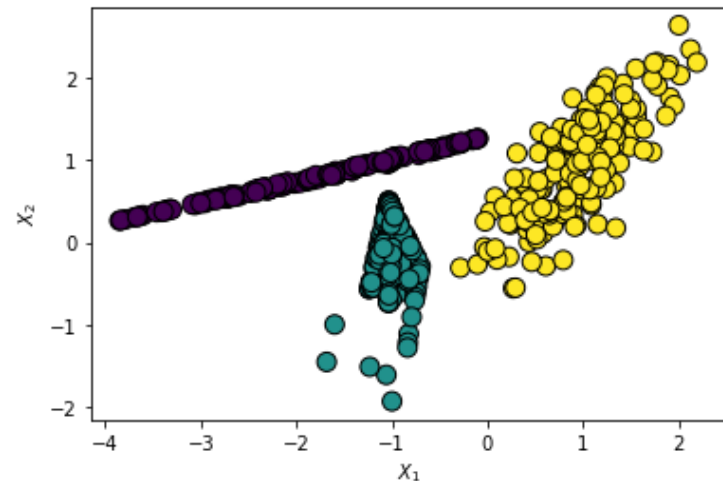
- 과소표집 (undersampling), 과대표집 (oversampling)
- 일반적으로 과소표집보다 과대표집이 통계적으로 유용함
- 의사결정나무 (decision tree)와 앙상블 (ensemble)은 상대적으로 불균형자료에 강인한 특성을 보임

# 데이터 불균형 (Data Imbalance)

- **과소표집** (undersampling)
  - 다수클래스의 표본을 임의로 학습데이터로부터 제거하는 것
- **과대표집** (oversampling)
  - 소수클래스의 표본을 복제하여 이를 학습데이터에 추가하는 것
  - 대표적인 방법론
    - SMOTE (Synthetic minority oversampling technique)
    - ADASYN (adaptive synthetic sampling method)

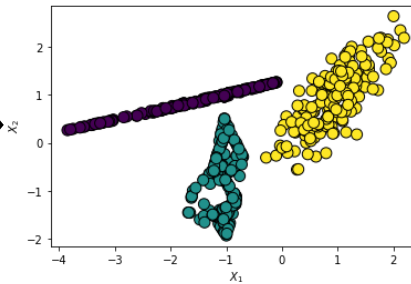


SMOTE



ADASYN

A scatter plot showing data points in a 2D space defined by axes  $X_1$  (horizontal) and  $X_2$  (vertical). The data points are categorized into three clusters, labeled #1, #2, and #3 in red. Cluster #1 (yellow circles) is located in the upper right quadrant, centered around  $X_1 = 1.5$  and  $X_2 = 1.0$ . Cluster #2 (teal circles) is located in the lower left quadrant, centered around  $X_1 = -1.0$  and  $X_2 = -1.0$ . Cluster #3 (purple circles) is located in the upper left quadrant, centered around  $X_1 = -2.5$  and  $X_2 = 0.5$ .



A scatter plot showing the relationship between two variables,  $X_1$  (horizontal axis) and  $X_2$  (vertical axis). The data points are categorized into three distinct groups, represented by different colors: purple, teal, and yellow. The purple points are clustered in the upper-left region, the teal points are in the center, and the yellow points are in the lower-right region. The axes range from approximately -4 to 1 for  $X_1$  and -1.0 to 1.0 for  $X_2$ .

## NearMiss

$$n\_classes = 3$$

```
↳ 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()
```

★ y에 클래스 정보가 들어있다.

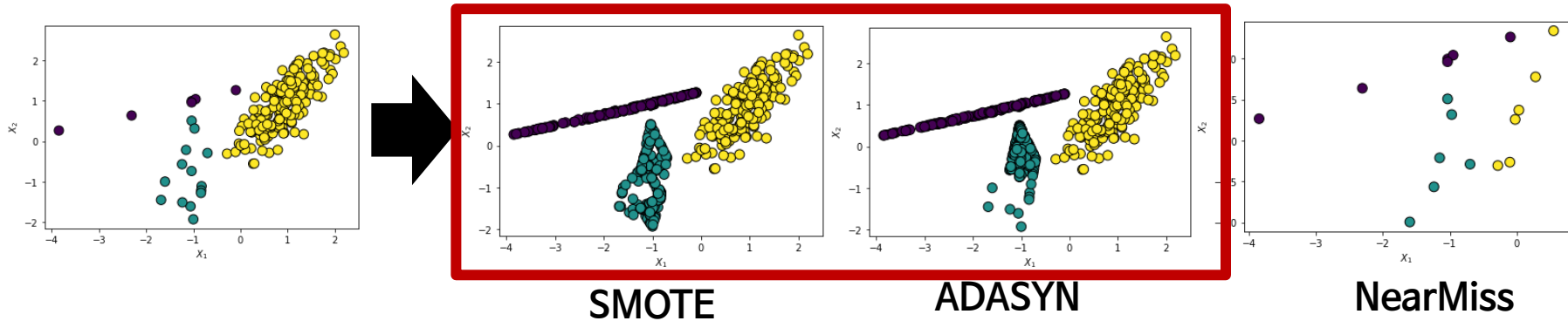
유용한  
라이프hack

∴ 자주 생각하고 이를 행하자!

$$X = \begin{matrix} n_{\text{sample}} \\ 5200 \end{matrix}$$
$$\therefore 20 \times 2$$

## ଅନୁପ୍ରାଣିତା

# 데이터 불균형 (Data Imbalance)



```
[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))
```

*sm.fit\_resample*

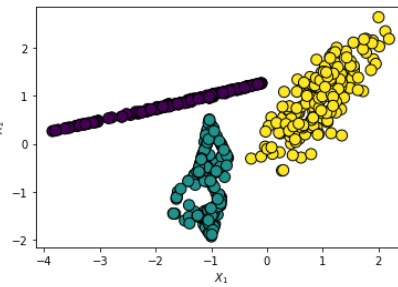
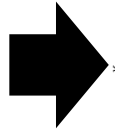
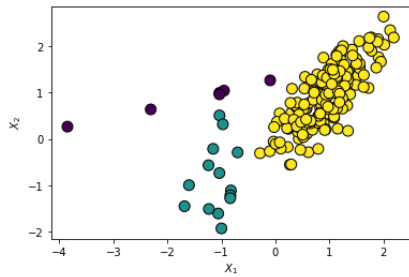
```
[52] 1 import matplotlib.pyplot as plt
      2
      3 plt.scatter(X_res[:, 0], X_res[:, 1], marker='o', c=y_res,
      4           s=100, edgecolor="k", linewidth=1)
      5
      6 plt.xlabel("$X_1$")
      7 plt.ylabel("$X_2$")
      8 plt.show()
```

*n-sample 수 늘어난 것만가?*

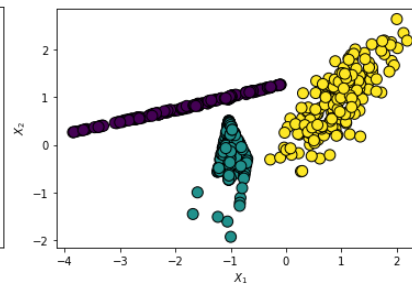
```
[63] 1 ada=ADASYN(random_state=0)
      2 X_syn, y_syn=ada.fit_resample(X, y)
      3 print('Resampled dataset shape from ADASYN %s' % Counter(y_syn))
```

```
[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()
```

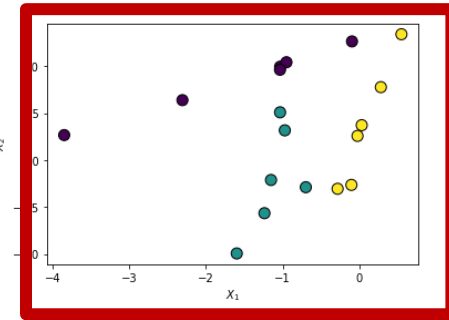
# 데이터 불균형 (Data Imbalance)



SMOTE



ADASYN



NearMiss

```
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()
```