Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 3 – 14, 2022

Python for Data Analytics

Data Preprocessing II

Data Scaling

Why Data Scaling?

- Features in a dataset can have very different scales

- Unscaled data can degrade the predictive performance of many machine learning algorithms
  - Many estimators assume that each feature takes values close to zero and all features vary on comparable scales
  - Metric-based and gradient-based estimators often assume approximately standardized data (normal distribution)
  - (cf.) Decision tree-based estimators are robust to arbitrary scaling of the data

- Unscaled data can slow down or even prevent the convergence of many gradient-based estimators

Data Scaling

- Standard scaling: → $\tilde{x}_i$ ~ Normal distribution (μ = 0, σ = 1) $$\tilde{x}_i = \frac{x_i - mean(x)}{std(x)}$$
- Min-Max Scaling: → $\tilde{x}_i$ in [0, 1] $$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$
- Max-Abs Scaling: → $|\tilde{x}_i| \leq 1$ $$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$
- Robust Scaling: → Based on median and IQR $$\tilde{x}_i = \frac{x_i - median(x)}{Q3(x) - Q1(x)}$$

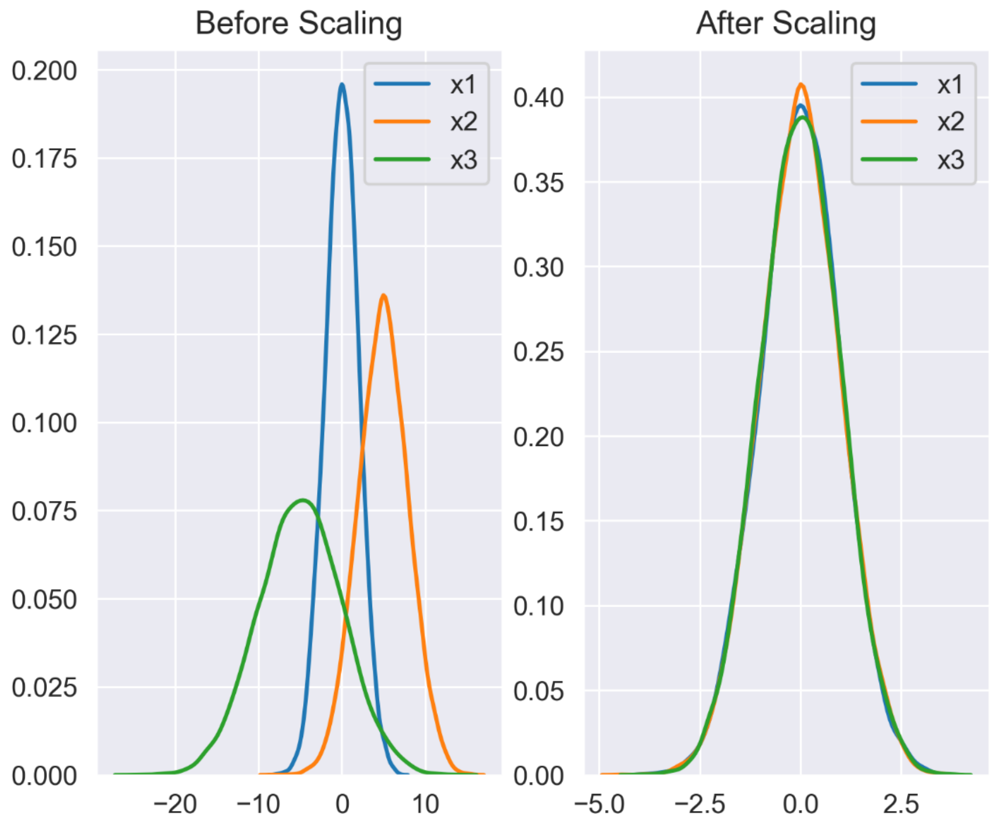
Data Scaling supported by SK-Learn

Scaling	Function	Class
Standard	`scale(x)`	`StandardScaler`
Min-Max	`minmax_scale()`	`MinMaxScaler`
Max-Abs	`maxabs_scale()`	`MaxAbsScaler`
Robust	`robust_scale()`	`RobustScaler`

Standard Scaling

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing

df = pd.DataFrame({
        'x1': np.random.normal(0, 2, 10000),
        'x2': np.random.normal(5, 3, 10000),
        'x3': np.random.normal(-5, 5, 10000)
})
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

sns.set_style('darkgrid')
_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

$$\tilde{x}_i = \frac{x_i - mean(x)}{std(x)}$$

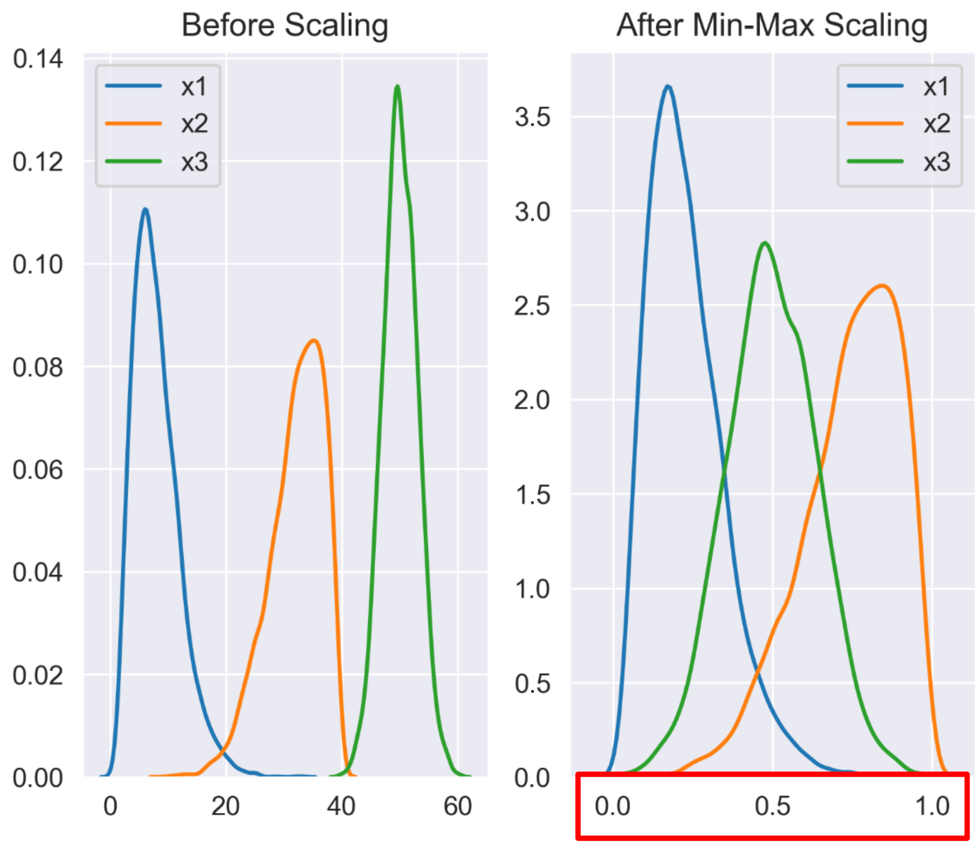


Min-Max Scaling

- All values are mapped in the range [0, 1]
- Very sensitive to the presence of outliers

$$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

```
df = pd.DataFrame({
        'x1': np.random.chisquare(8, 10000),    # positive skew
        'x2': np.random.beta(8, 2, 10000)*40,   # negative skew
        'x3': np.random.normal(50, 3, 10000)    # no skew
})

scaler = preprocessing.MinMaxScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Min-Max Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

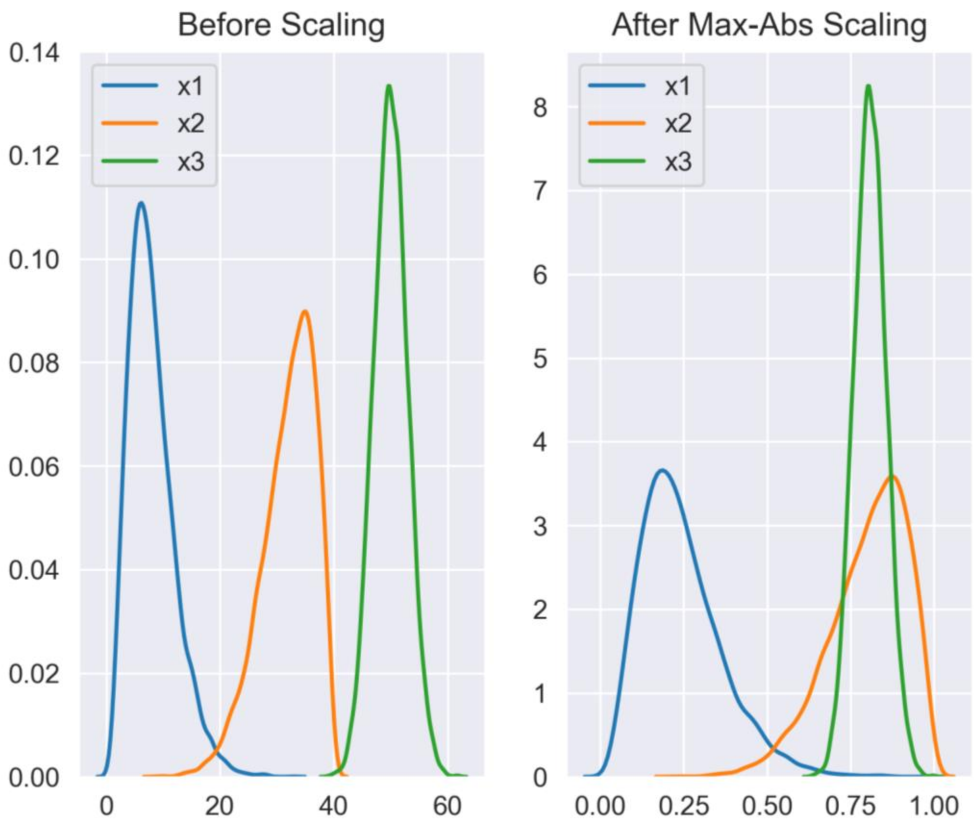


Max-Abs Scaling

- Doesn't change the shape of the distribution
- Also suffers from the presence of large outliers

$$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$

```python
df = pd.DataFrame({
        'x1': np.random.chisquare(8, 10000),     # positive skew
        'x2': np.random.beta(8, 2, 10000)*40,    # negative skew
        'x3': np.random.normal(50, 3, 10000)     # no skew
})

scaler = preprocessing.MaxAbsScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Max-Abs Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```



Robust Scaling (1)

- Based on percentiles
- Not influenced by a few number of very large marginal outliers

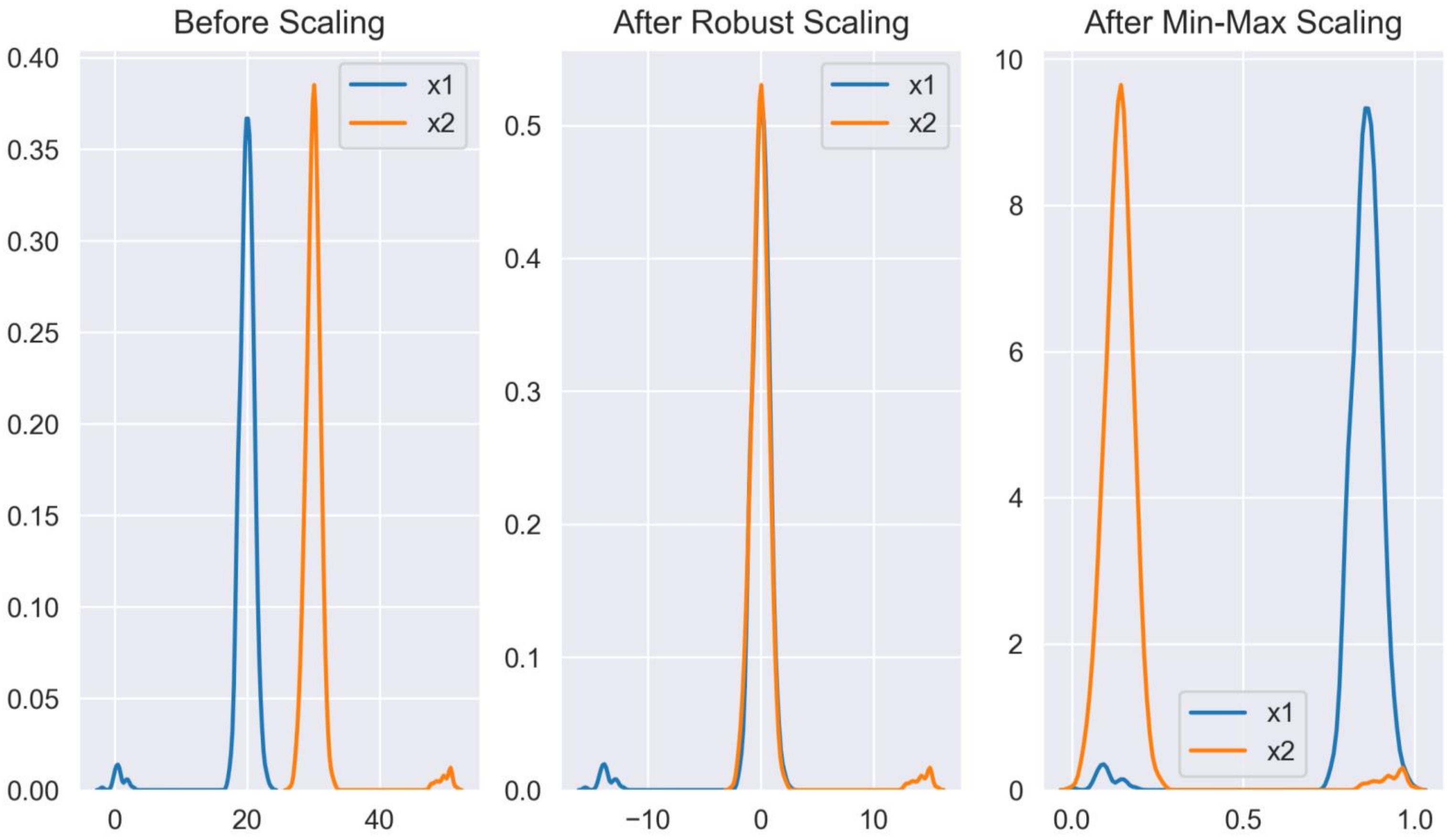
$$\widetilde{x_i} = \frac{x_i - median(x)}{Q3(x) \ - Q1(x)}$$

```
df = pd.DataFrame({
        # distribution with lower outliers
        'x1': np.hstack((np.random.normal(20,1,1000),
                         np.random.normal(1,1,25))),
        # distribution with upper outliers
        'x2': np.hstack((np.random.normal(30,1,1000),
                         np.random.normal(50,1,25)))
})

robust_scaler = preprocessing.RobustScaler()
robust_df = robust_scaler.fit_transform(df)
robust_df = pd.DataFrame(robust_df, columns=['x1', 'x2'])

minmax_scaler = preprocessing.MinMaxScaler()
minmax_df = minmax_scaler.fit_transform(df)
minmax_df = pd.DataFrame(minmax_df, columns=['x1', 'x2'])
```

```
_, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(9,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df['x1'], ax=ax2)
sns.kdeplot(robust_df['x2'], ax=ax2)

ax3.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df['x1'], ax=ax3)
sns.kdeplot(minmax_df['x2'], ax=ax3)
```

Robust Scaling (2)

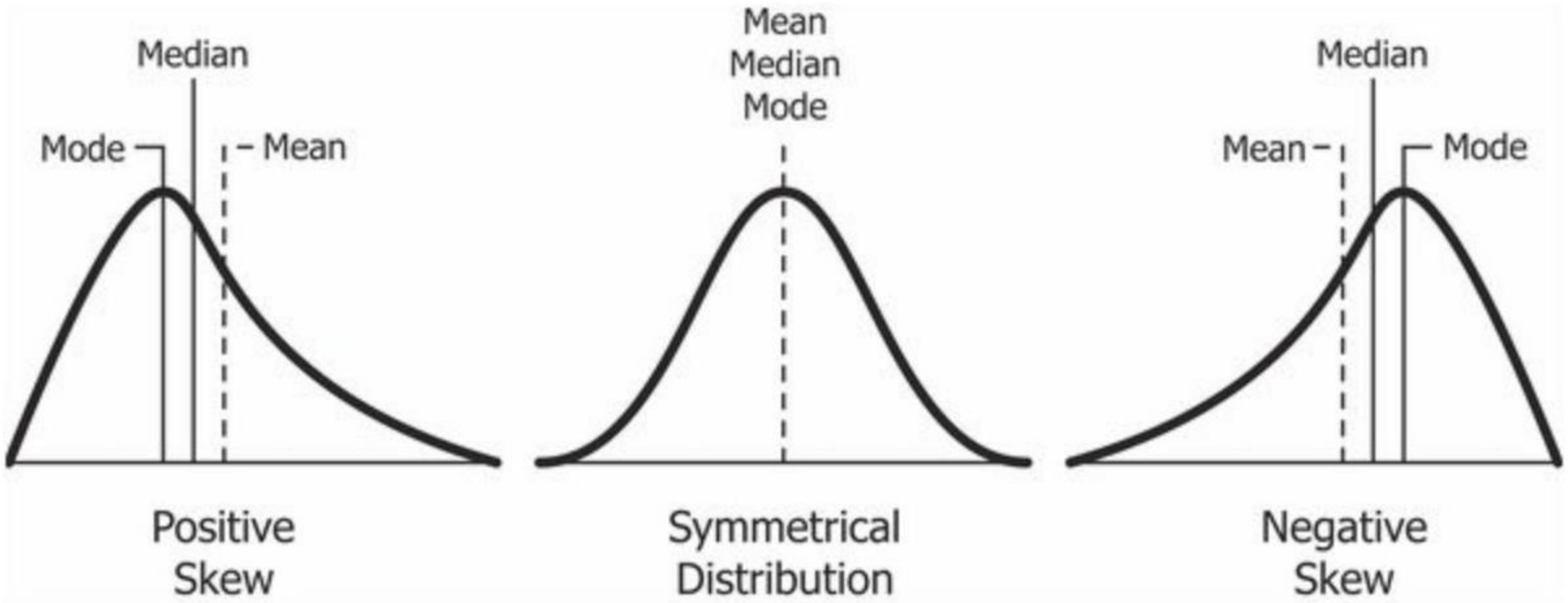
- **Min-Max Scaling**
  - Significantly affected by outliers
- **Robust Scaling**
  - Inliers are in [-2, 2]
  - Outliers still exist at the end of each distribution



Data Standardization

Data Skewness

- A measure of asymmetry of a distribution
  - Symmetrical (skewness = 0, e.g., normal distribution): mean == median == mode
  - Positive skew (skewness > 0): tail at right, mode < median < mean
  - Negative skew (skewness < 0): tail at left, mean < median < mode



Measuring Data Skewness

- *df*.skew([*axis*], [*skipna*], ...)
  - Return unbiased skew over requested axis
  - *axis*: axis for the function to be applied on
  - *skipna*: if True, exclude null values when computing the result (default True)

- Meaning of skewness value
  - -0.5 <= skewness <= 0.5: fairly symmetrical
  - -1 < skewness < -0.5 or 0.5 < skewness < 1: moderately skewed
  - skewness < -1 or skewness > 1: highly skewed

Handling Data Skewness

- Linear model performs better when the dataset follows normal distribution

- Dealing with positive skewness
  - Square root transformation (x to $x^{1/2}$)
  - Cube root transformation (x to $x^{1/3}$)
  - Log transformation (x to $\log_2 x, \log_e x, \ln x, \ldots$)

- Dealing with negative skewness
  - Square transformation (x^2)
  - Cube transformation (x^3)
  - Reflect the values and apply the methods used to reduce the positive skewness

The Boston Housing Dataset

- Dataset for housing values in areas of Boston in 70's
- 506 rows, 14 columns (13 attributes + housing value)
- Available in the SK-Learn datasets

CRIM: 범죄율
ZN: 25,000ft^2 초과 거주지역 비율
INDUS: 비소매상업지역 면접 비율
CHAS: 찰스강 경계에 위치한 경우 1
NOX: 일산화질소 농도
AGE: 1940년 이전 건축된 주택 비율
RM: 주택당 방 수
RAD: 방사형 고속도로까지의 거리
LSTAT: 인구 중 하위 계층 비율
DIS: 직업 센터의 거리
B: 인구 중 흑인 비율
TAX: 재산세율
PTRATIO: 학생/교사 비율
MEDV: 주택 가격의 median (단위: $1,000)

```
from sklearn import datasets
import pandas as pd
boston = datasets.load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df.head()
```

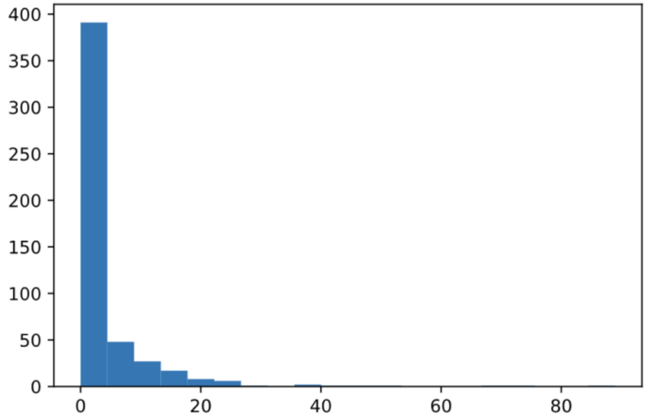
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Skewness in Boston Housing Dataset

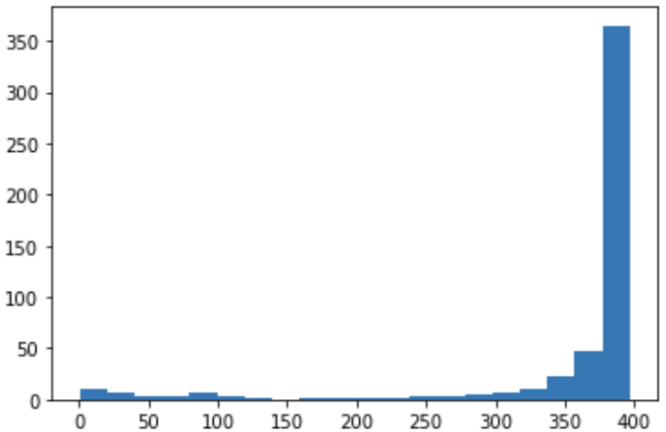
```
df.skew()
```

```
CRIM        5.223149
ZN          2.225666
INDUS       0.295022
CHAS        3.405904
NOX         0.729308
RM          0.403612
AGE        -0.598963
DIS         1.011781
RAD         1.004815
TAX         0.669956
PTRATIO    -0.802325
B          -2.890374
LSTAT       0.906460
dtype: float64
```

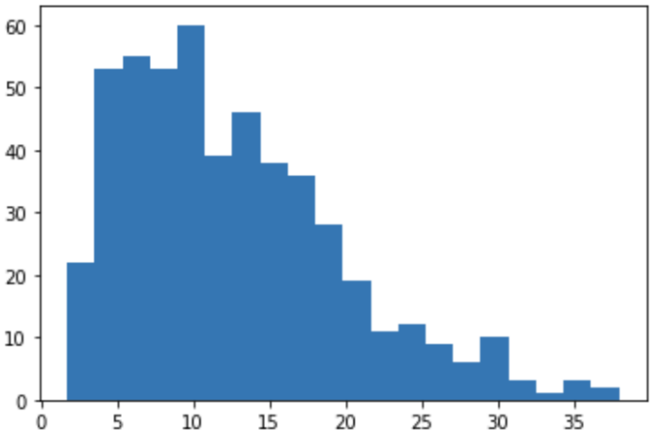
```
plt.hist(df.CRIM, bins=20)
```



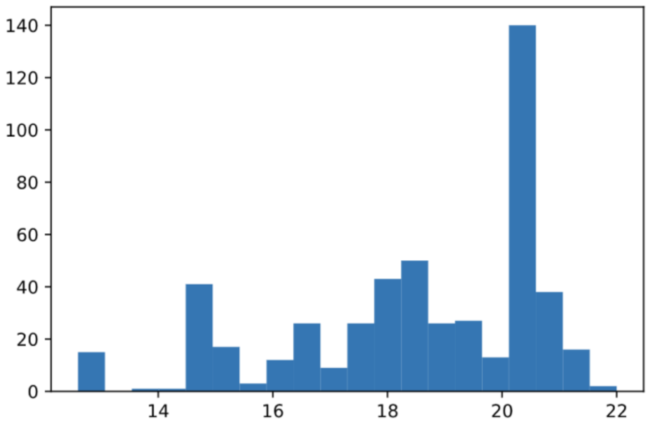

```
plt.hist(df.B, bins=20)
```




```
plt.hist(df.LSTAT, bins=20)
```




```
plt.hist(df.PTRATIO, bins=20)
```



Transforming Data (1)

- *sklearn.preprocessing.*`scale(`*X*, ...`)`
  - Standardize a dataset along any axis (standard scaler)
  - Center to the zero mean and component wise scale to unit variance
  - *X*: the data to center and scale

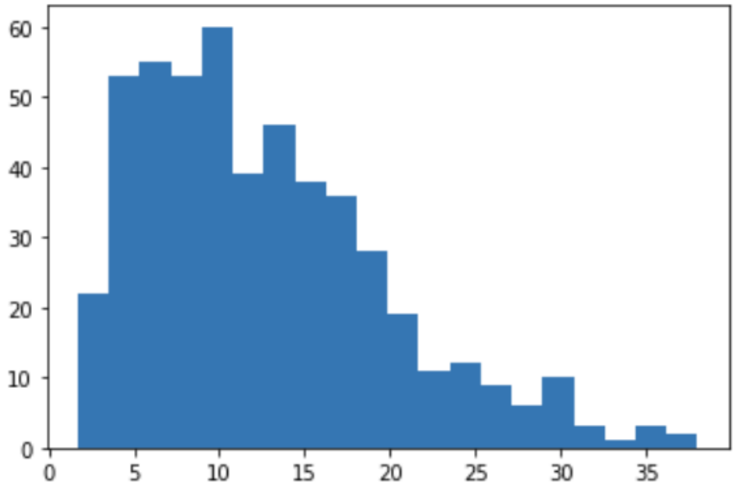
```
from sklearn import preprocessing

df['LSTAT_log'] = preprocessing.scale(np.log(df['LSTAT']+1))
df['LSTAT_sqrt'] = preprocessing.scale(np.sqrt(df['LSTAT']+1))
df[['LSTAT', 'LSTAT_log', 'LSTAT_sqrt']].skew()
```

```
LSTAT          0.906460
LSTAT_log     -0.187195
LSTAT_sqrt     0.359606
dtype: float64
```

Transforming Data (2)

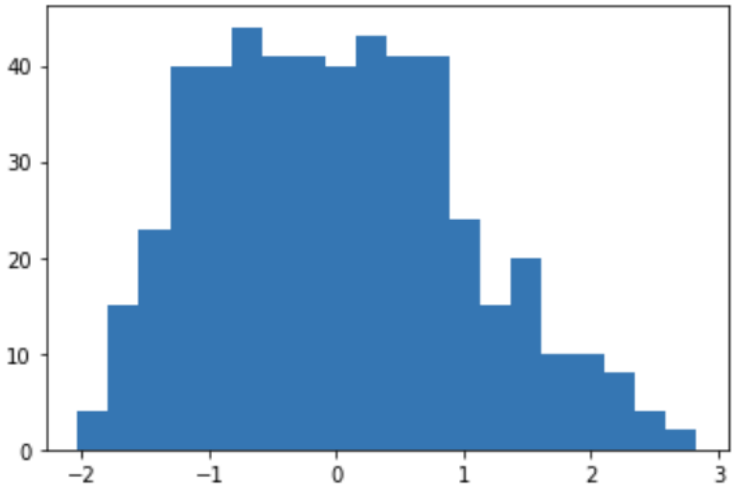
Original data

```python
import matplotlib.pyplot as plt

plt.hist(df['LSTAT'], bins=20)
plt.show()
```



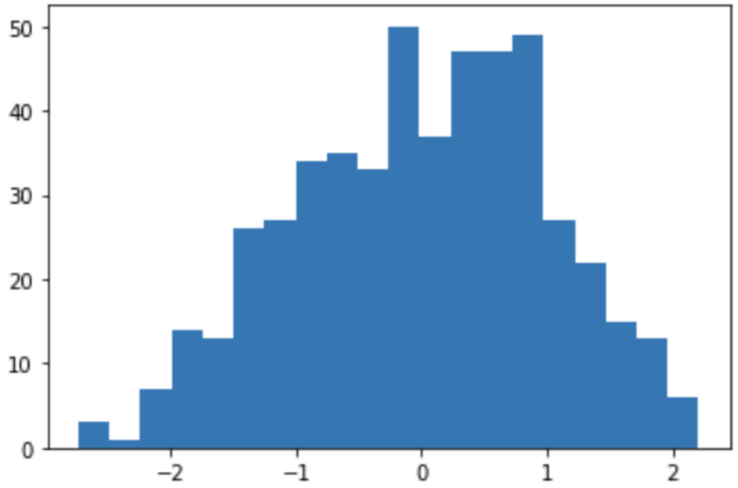
Square root transformation

```python
import matplotlib.pyplot as plt

plt.hist(df['LSTAT_sqrt'], bins=20)
plt.show()
```



Log transformation

```python
import matplotlib.pyplot as plt

plt.hist(df['LSTAT_log'], bins=20)
plt.show()
```



Sampling for Imbalanced Data

Imbalanced Data

- A problem with classification where the classes are not represented equally
  - The model will be mostly tuned for the majority class
- Example:
  - A dataset with Class A : Class B = 9 : 1
  - The percentage of correct answers in the test dataset will also be 9 : 1
  - Even if a model classifies everything to Class A, it will have a 90% of accuracy
- Solutions: Balance data using sampling
  - Oversampling: increase the amount of minority class
  - Undersampling: use only part of majority class

imbalanced-learn module

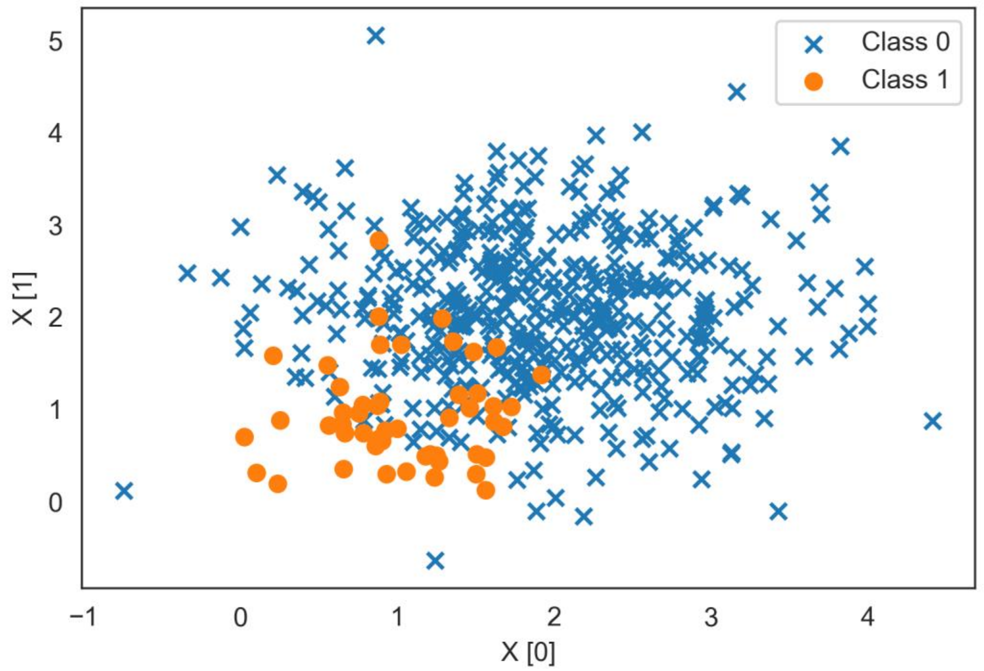
- A python package offering a number of re-sampling techniques
- Commonly used for datasets showing strong between-class imbalance
- Part of scikit-learn-contrib projects
- https://github.com/scikit-learn-contrib/imbalanced-learn

- Installation
  - pip install -U imbalanced-learn
  - conda install -c conda-forge imbalanced-learn

```
>>> import imblearn.under_sampling
>>> import imblearn.over_sampling
```

Creating Imbalanced Data

```python
def plot(X, y):
    plt.scatter(X[y==0, 0], X[y==0, 1], marker='x', label='Class 0')
    plt.scatter(X[y==1, 0], X[y==1, 1], marker='o', label='Class 1')
    plt.xlabel('X [0]')
    plt.ylabel('X [1]')
    plt.legend()

n0 = 450
n1 = 50

a = np.random.randn(n0, 2)*0.8 + 2  # N(2, 0.8)
b = np.random.randn(n1, 2)*0.5 + 1  # N(1, 0.5)

X = np.vstack([a, b])
y = np.hstack([np.zeros(n0), np.ones(n1)])

plot(X, y)
```

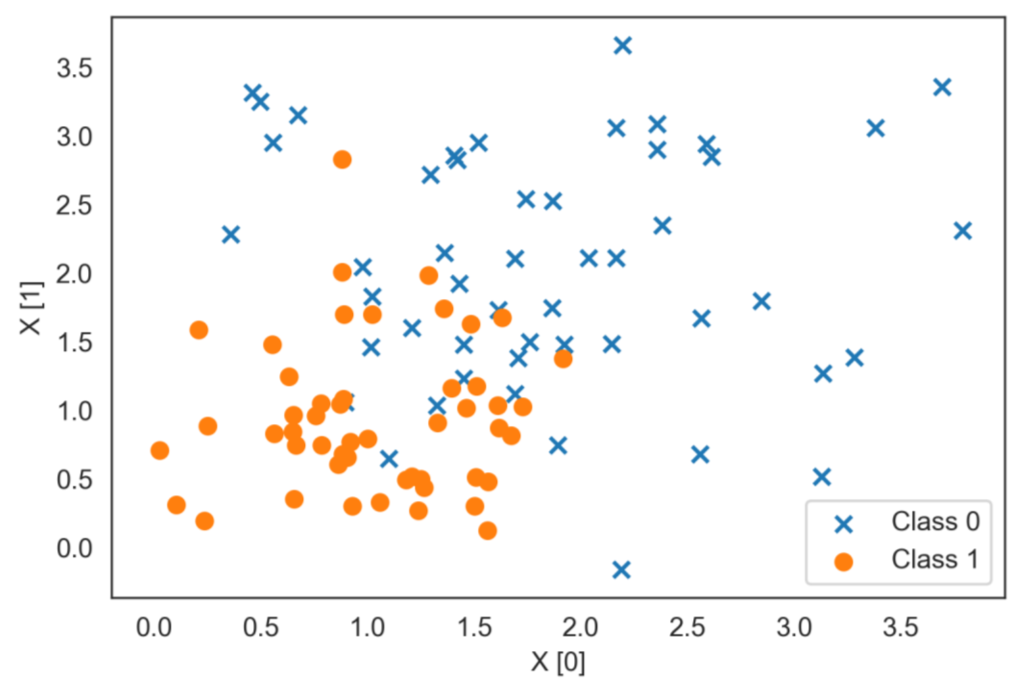


Undersampling: RandomUnderSampler()

- Under-sample the majority class by randomly picking samples

```
from imblearn.under_sampling import RandomUnderSampler

X_samp, y_samp = RandomUnderSampler(random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(100, 2) (100,)
```

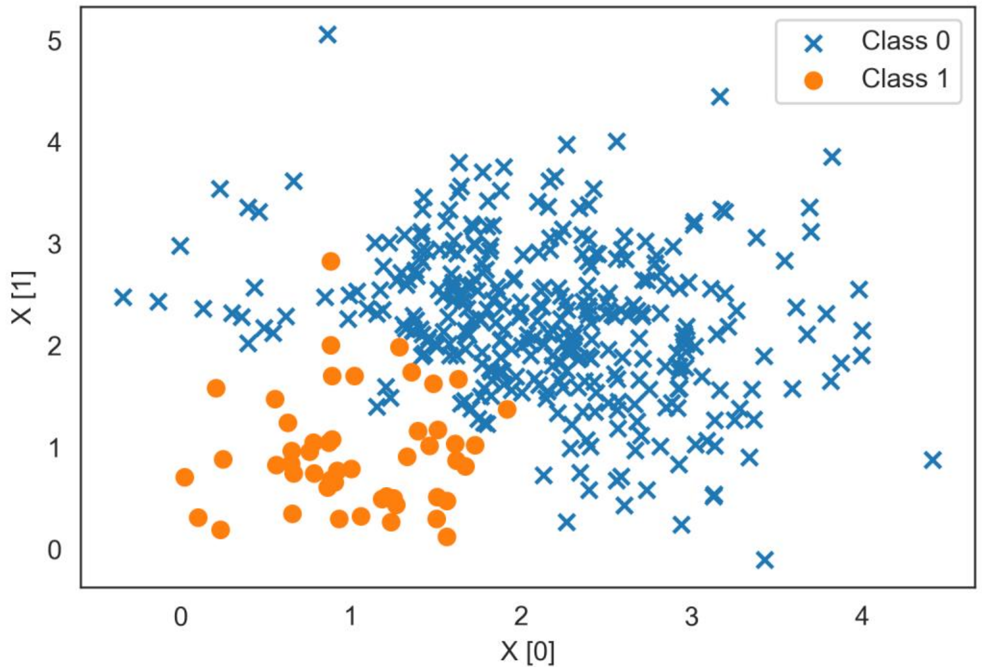


Undersampling: EditedNearestNeighbours()

- Keep a sample if all or majority of the NN's belong to the same class

```
from imblearn.under_sampling import EditedNearestNeighbours

X_samp, y_samp = EditedNearestNeighbours(kind_sel='all', n_neighbors=10,
                                         random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(388, 2) (388,)
```



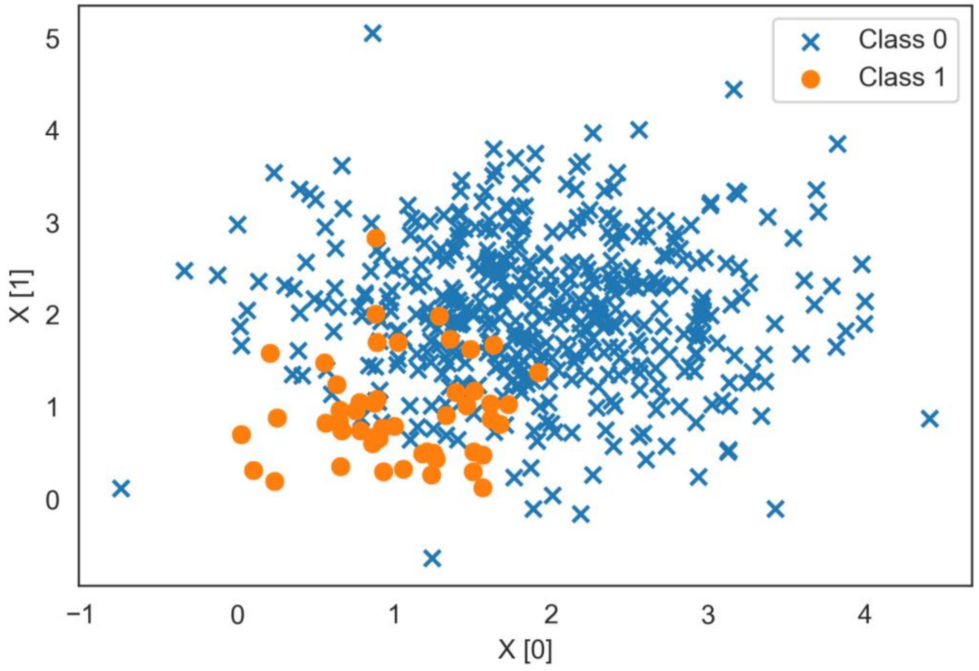
- *n_neighbors*: size of the neighbourhood to consider to compute the nearest neighbors
- *kind_sel*: 'all' (all have to agree to keep),
  'mode' (majority vote to keep)

Oversampling: RandomOverSampler()

- Over-sample the minority class by picking samples at random

```
from imblearn.over_sampling import RandomOverSampler

X_samp, y_samp = RandomOverSampler(random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(900, 2) (900,)
```



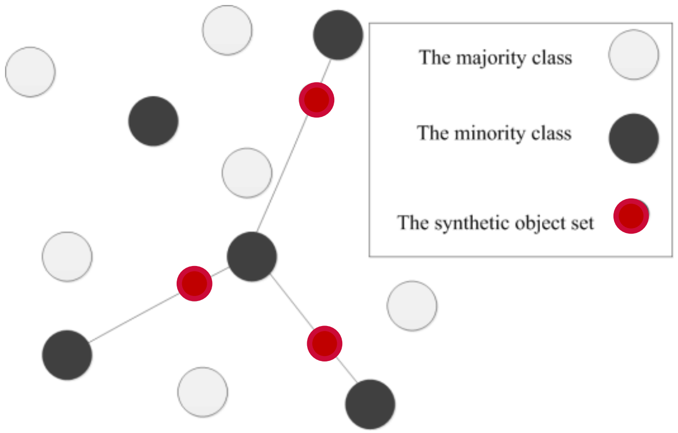
- Graph looks same, but the count has increased to 450

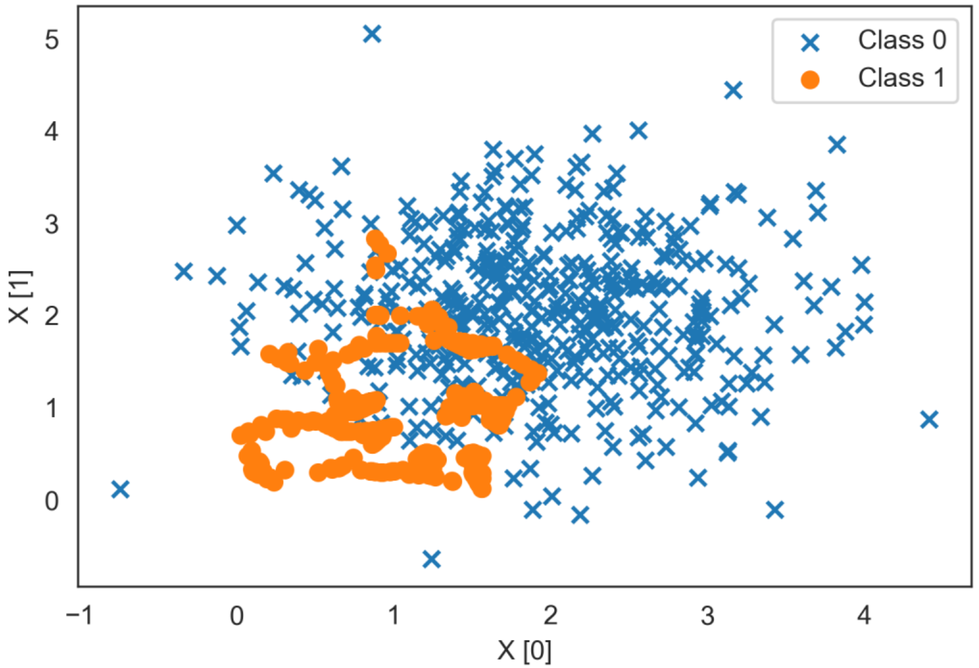
Oversampling: SMOTE()

- A sample is created at a randomly selected point between a minority sample and its neighbor which is randomly selected among *k* neighbors

```
from imblearn.over_sampling import SMOTE

X_samp, y_samp = SMOTE(k_neighbors=3).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(900, 2) (900,)
```



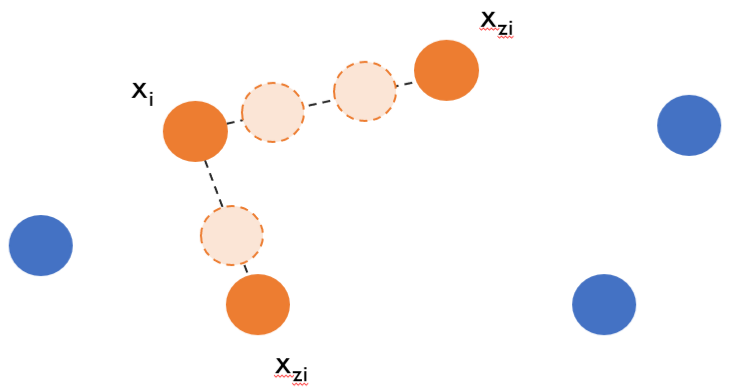


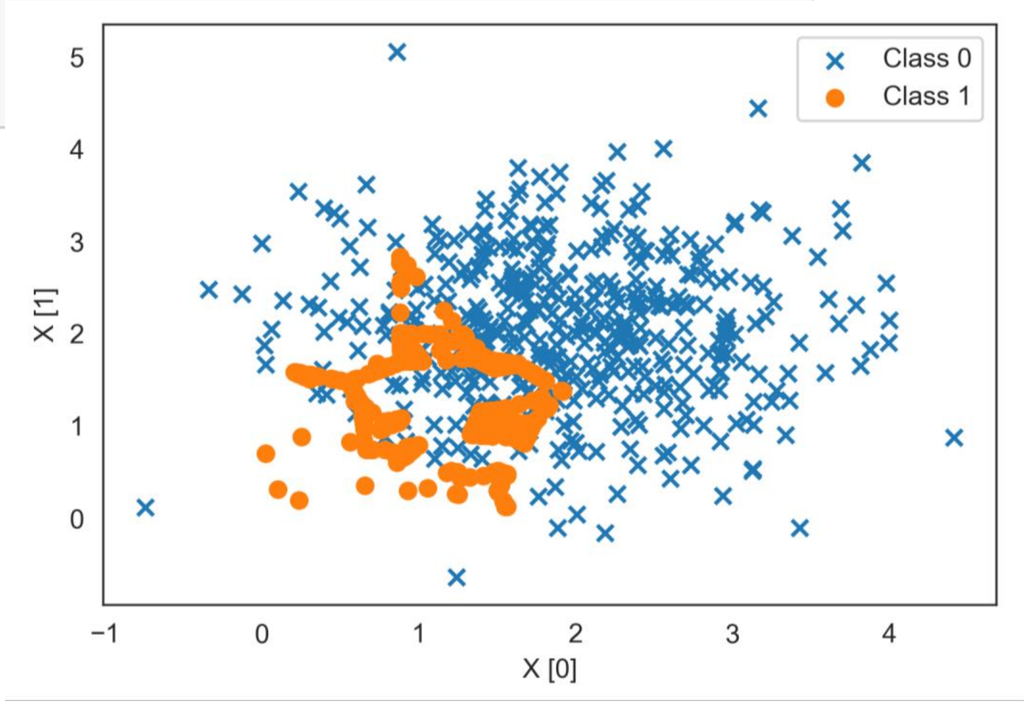
Oversampling: ADASYN()

- For a minority sample dominated by majority class samples, more synthetic minority class samples are generated

```
from imblearn.over_sampling import ADASYN

X_samp, y_samp = ADASYN(n_neighbors=3, random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(908, 2) (908,)
```





Thank You!