기계학습 (Machine Learning)

L05
- Data Preprocessing and Machine Learning with Scikit-Learn
- Nearest Neighbor Methods (kNN)

한밭대학교

정보통신공학과

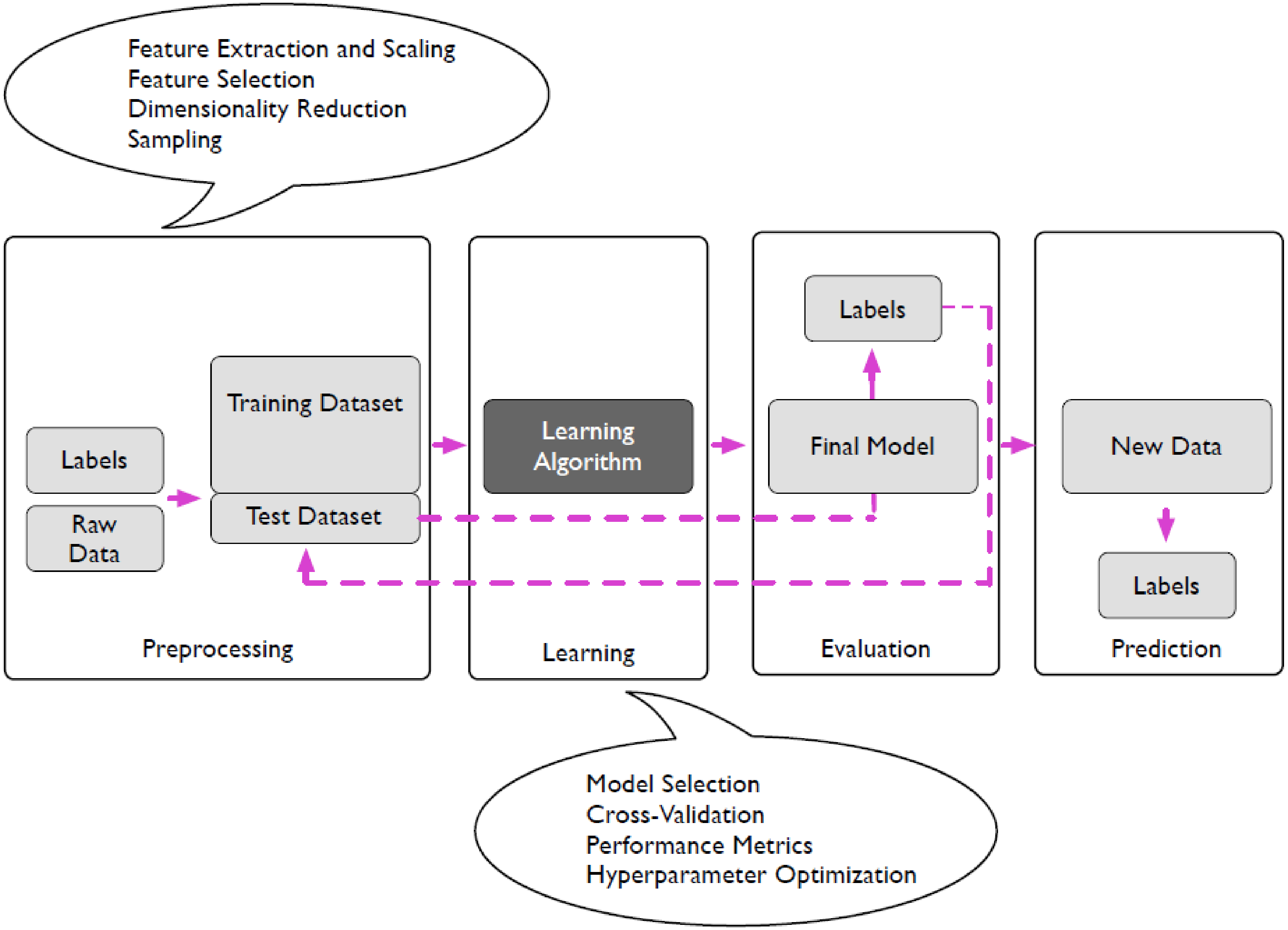
최 해 철

국립 한밭대학교

VISUAL MEDIA LAB.
HANBAT NATIONAL UNIVERSITY

ToC

Machine Learning Workflow



Reading a Dataset from a Tabular Text File

The Iris Dataset

Iris-Setosa

Iris-Versicolor

Iris-Virginica

Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).

.CSV 파일

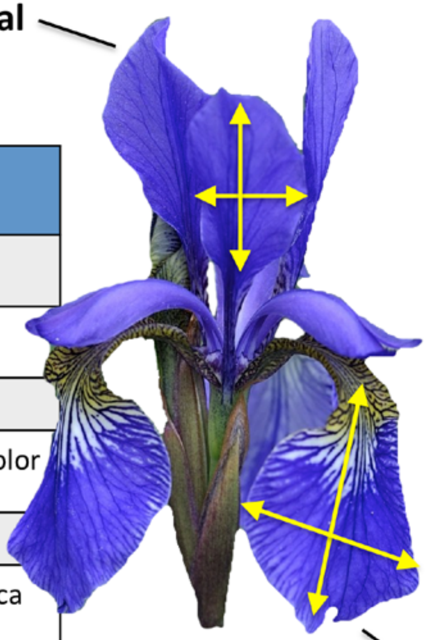
◆ 표 형태의 데이터를 저장하는 파일

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			…		
50	6.4	3.5	4.5	1.2	Versicolor
			…		
150	5.9	3.0	5.0	1.8	Virginica

Features
(attributes, measurements, dimensions)

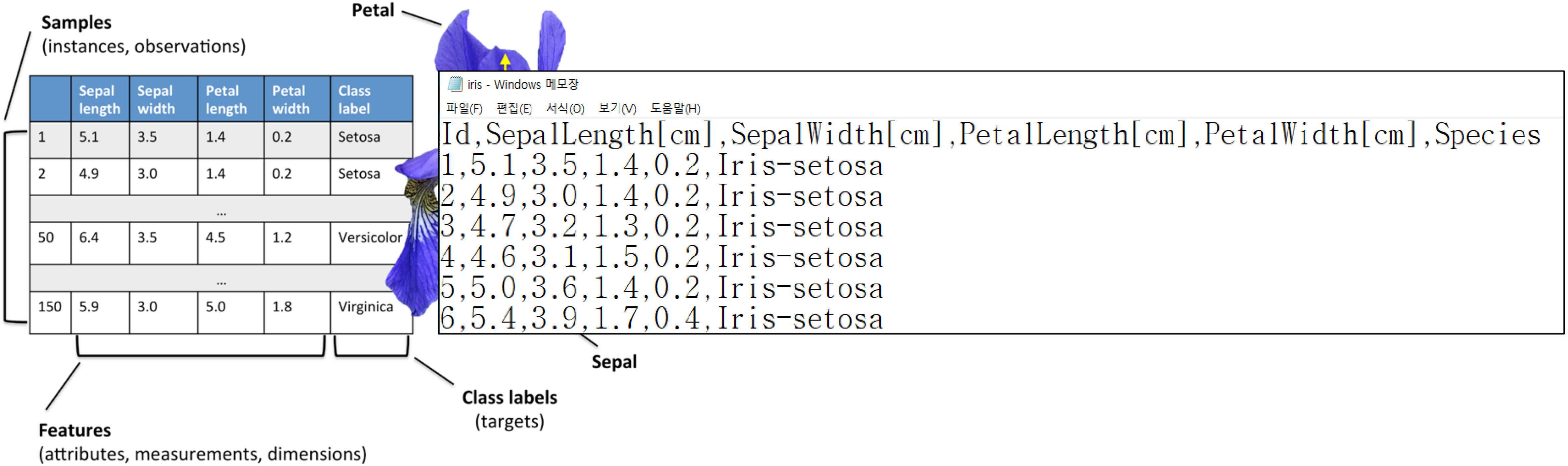
Class labels
(targets)



.CSV 파일

◆ 표 형태의 데이터를 저장하는 파일

- 각 줄은 하나의 행(row)에 해당하고, 각 열(column) 사이에는 **쉼표(,)**를 넣어 구분
- 모든 행은 같은 개수의 열을 가져야 한다.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

```
Id,SepalLength[cm],SepalWidth[cm],PetalLength[cm],PetalWidth[cm],Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
```

A DataFrame Library for Data Wrangling

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

https://pandas.pydata.org

(PANel DAta S)

McKinney, Wes. "Data structures for statistical computing in python." *Proceedings of the 9th Python in Science Conference*. Vol. 445. 2010.

Pandas

```
import pandas as pd


df = pd.read_csv('iris.csv')
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

DataFrame

```
df.shape
```

```
(150, 6)
```

Pandas

```
import pandas as pd
data = {  'A' : [1, 2, 3],
          'B' : [4, 5, 6],
          'C'  : [7, 8, 9]}
df = pd.DataFrame(data)
df
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

```
value = df.iloc[0, 1]
value
```

4

“iloc” : Pandas가 제공하는 DataFrame에서 행과 열을 선택하고 인덱싱하는 메서드, integer location의 약자

Basic Data Handling

Python Function

```python
def some_func(x):
    return 'Hello World ' + str(x)

some_func(123)
```

```
'Hello World 123'
```

Regular Function vs Lambda Function

◆ lambda 함수

- 이름 없이 정의, 주로 간단한 연산을 수행할 때 사용

```python
def some_func(x):
    return 'Hello World ' + str(x)

some_func(123)
```

```
'Hello World 123'
```

```python
f = lambda x: 'Hello World ' + str(x)
f(123)
```

```
'Hello World 123'
```

python Copy code

```python
lambda 매개변수: 표현식
```

여기서:

- `lambda`: 람다 함수를 정의하는 키워드입니다.
- `매개변수`: 함수에 전달되는 입력 매개변수(parameter)입니다.
- `표현식`: 매개변수를 이용하여 수행되는 연산이나 계산식(expression)입니다.

Column-based Data Processing

◆ Lambda Functions and ".apply"

Basic Data Handling

```
df['Species'] = df['Species'].apply(lambda x: 0 if x=='Iris-setosa' else x)
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

Species
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa

Column-based Data Processing

◆ Dictionaries and “.map”

```
d = {'Iris-setosa': 0,
     'Iris-versicolor': 1,
     'Iris-virginica': 2}

df = pd.read_csv('iris.csv')
df['Species'] = df['Species'].map(d)
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

Quick Inspections

◆ "**head**" and "**tail**"

```
df.tail()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

Accessing the Underlying NumPy Array(s)

◆ the "`.values`" attribute

NumPy Arrays

```
y = df['Species'].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Label Vector "y" and Design Matrix "X"

```
y = df['Species'].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
X = df.iloc[:, 1:5].values
X[:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

A Library with Additional Data Science-& Machine Learning-related Functions

MLXTEND

http://rasbt.github.io/mlxtend/

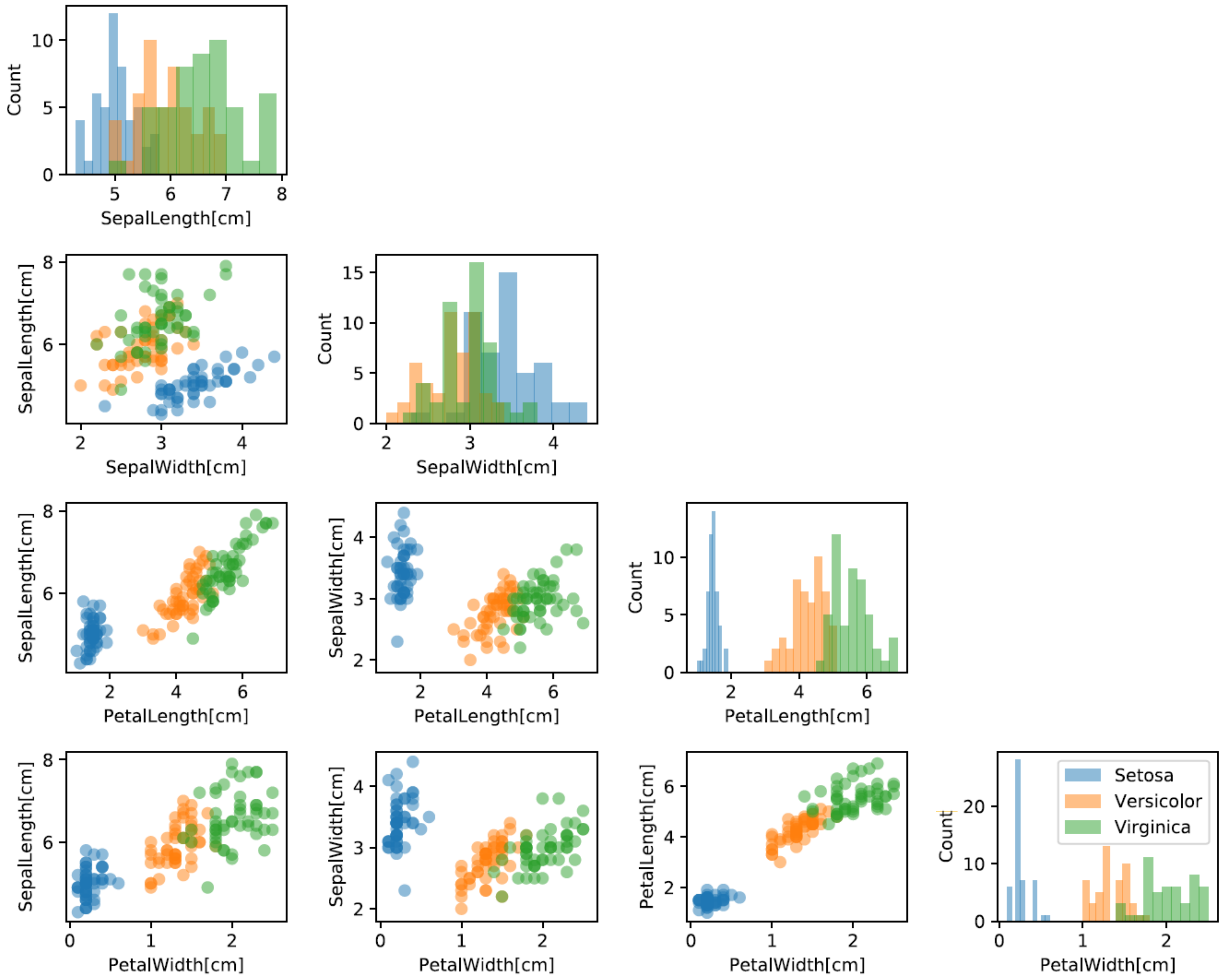
Raschka, Sebastian. "MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack." *The Journal of Open Source Software* 3.24 (2018).

Exploratory Data Analysis (EDA)

```
%matplotlib inline
import matplotlib.pyplot as plt
from mlxtend.data import iris_data
from mlxtend.plotting import scatterplotmatrix


names = df.columns[1:5]   # df.colums 데이터 프레임의 컬럼 이름

fig, axes = scatterplotmatrix(X[y==0], figsize=(10, 8), alpha=0.5)
fig, axes = scatterplotmatrix(X[y==1], fig_axes=(fig, axes), alpha=0.5)
fig, axes = scatterplotmatrix(X[y==2], fig_axes=(fig, axes), alpha=0.5, names=names)

plt.tight_layout()
plt.legend(labels=['Setosa', 'Versicolor', 'Virginica'])
plt.show()
```

```
pip install mlxtend
```

만약 상기 코드에서 mlxtend에서 오류나면 설치. 코랩에서는 설치 없이 가능


Count
SepalLength[cm]
SepalWidth[cm]
PetalLength[cm]
PetalWidth[cm]
Setosa
Versicolor
Virginica

Splitting a Dataset into Train, Validation, and Test Subsets

```
import numpy as np

indices = np.arange(X.shape[0])     # X.shape[0] = 150, indices = [0, 1, 2, …, 149]
rng = np.random.RandomState(123)
permuted_indices = rng.permutation(indices) # "permutation"은 원소나 객체들의 순서를 섞음
permuted_indices
```

```
array([ 72, 112, 132,  88,  37, 138,  87,  42,   8,  90, 141,  33,  59,
       116, 135, 104,  36,  13,  63,  45,  28, 133,  24, 127,  46,  20,
        31, 121, 117,   4, 130, 119,  29,   0,  62,  93, 131,   5,  16,
        82,  60,  35, 143, 145, 142, 114, 136,  53,  19,  38, 110,  23,
         9,  86,  91,  89,  79, 101,  65, 115,  41, 124,  95,  21,  11,
       103,  74, 122, 118,  44,  51,  81, 149,  12, 129,  56,  50,  25,
       128, 146,  43,   1,  71,  54, 100,  14,   6,  80,  26,  70, 139,
        30, 108,  15,  18,  77,  22,  10,  58, 107,  75,  64,  69,   3,
        40,  76, 134,  34,  27,  94,  85,  97, 102,  52,  92,  99, 105,
         7,  48,  61, 120, 137, 125, 147,  39,  84,   2,  67,  55,  49,
        68, 140,  78, 144, 111,  32,  73,  47, 148, 113,  96,  57, 123,
       106,  83,  17,  98,  66, 126, 109])
```

Splitting a Dataset into Train, Validation, and Test Subsets

```
import numpy as np

indices = np.arange(X.shape[0])
rng = np.random.RandomState(123)
permuted_indices = rng.permutation(indices)
permuted_indices
```

```
train_size, valid_size = int(0.65*X.shape[0]), int(0.15*X.shape[0])
test_size = X.shape[0] - (train_size + valid_size)
print(train_size, valid_size, test_size)
```

```
97 22 31
```

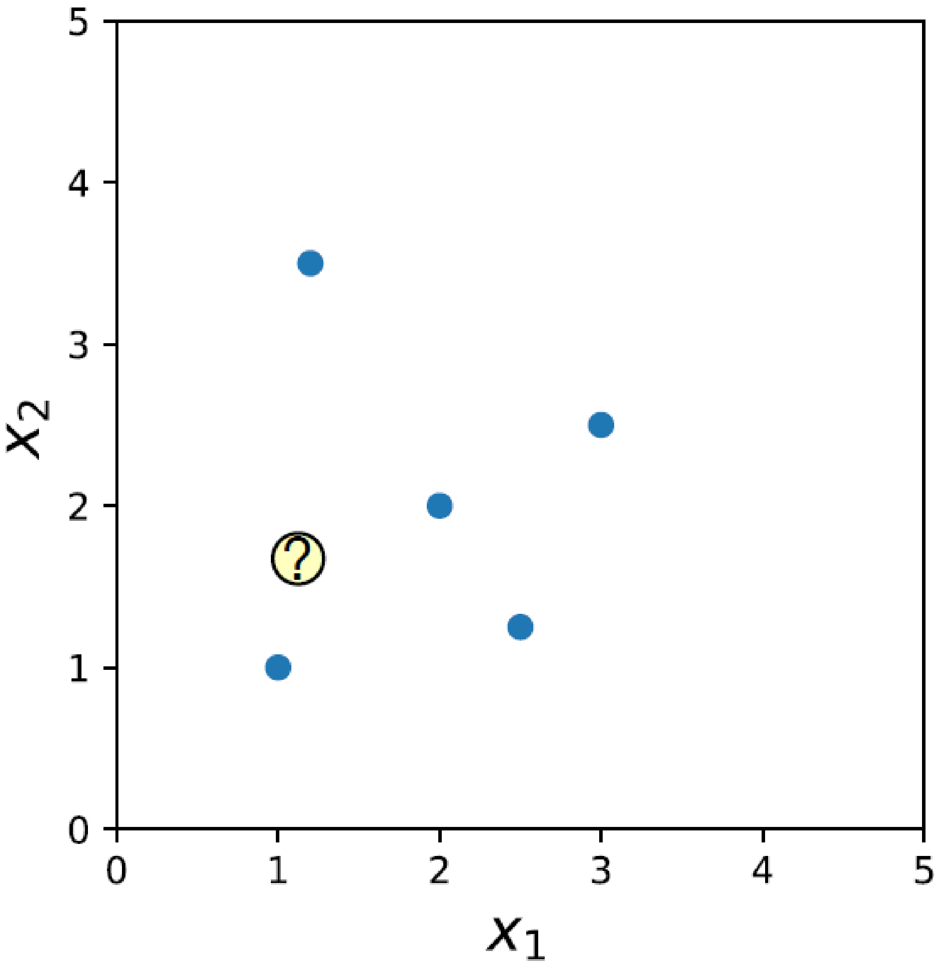
```
train_ind = permuted_indices[:train_size]
valid_ind = permuted_indices[train_size:(train_size + valid_size)]
test_ind = permuted_indices[(train_size + valid_size):]
```

```
X_train, y_train = X[train_ind], y[train_ind]
X_valid, y_valid = X[valid_ind], y[valid_ind]
X_test, y_test = X[test_ind], y[test_ind]
```

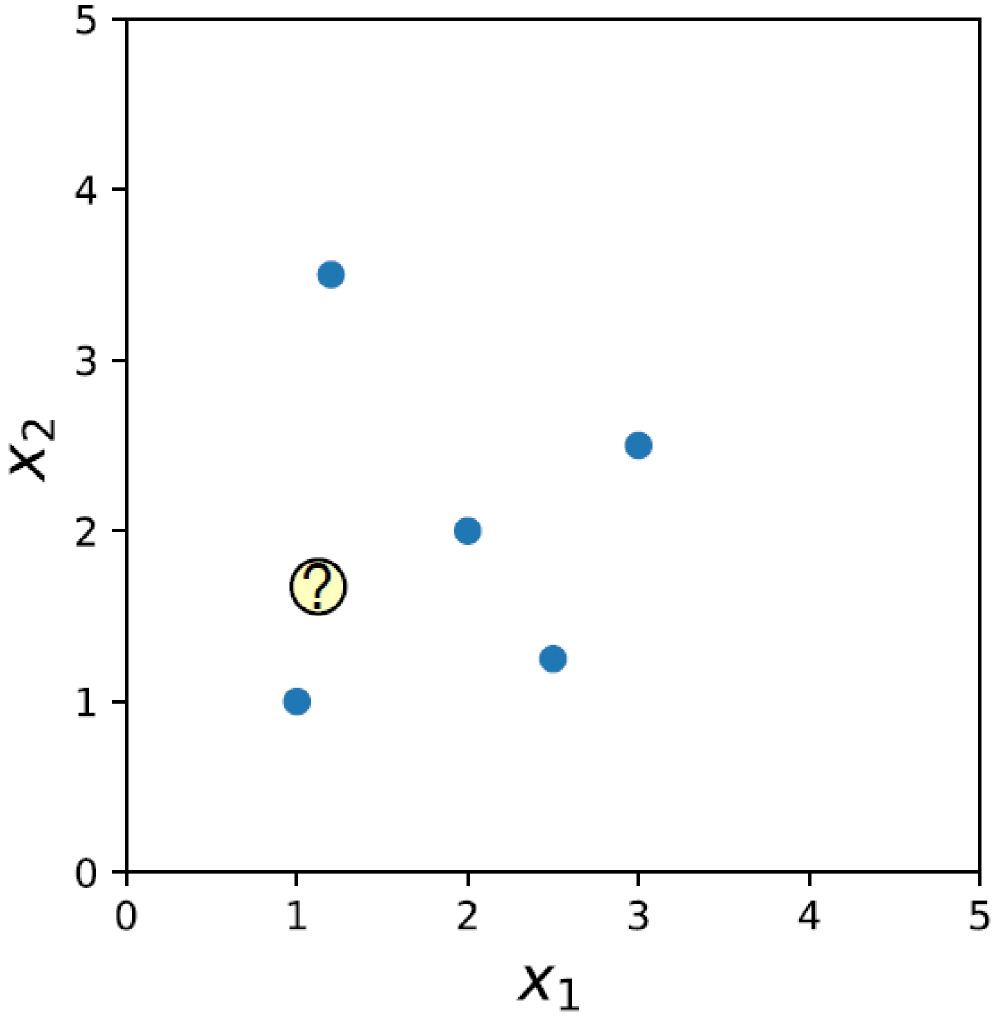
추후 더 쉬운 방법을 살펴봄

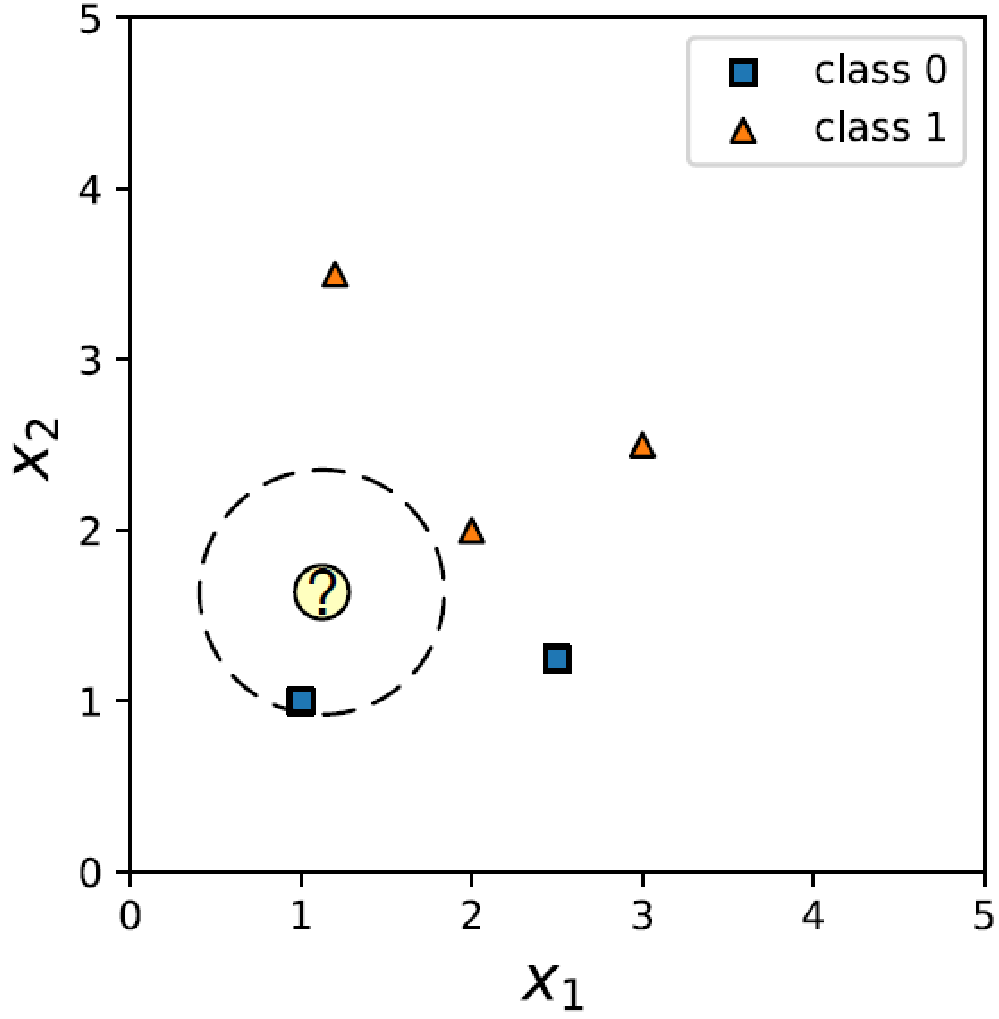
Nearest Neighbor Methods

1-Nearest Neighbor



1-Nearest Neighbor

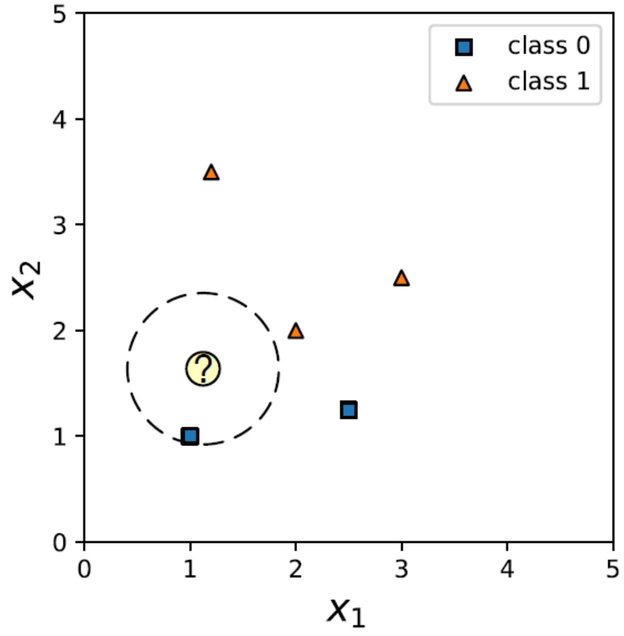




1-Nearest Neighbor

- Training Step :
  - 훈련 데이터셋을 메모리에 저장
    - A lazy learning algorithm
    - 비모수 모델(Nonparametric model) (vs. Parametric model?. See p.137)
      - ✓ 고정된 개수의 파라미터로 설명되지 않음
      - ✓ 훈련 데이터가 늘어남에 따라 파라미터 개수도 증가
    - 인스턴스(instance) 기반 모델

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (\,|\mathcal{D}| = n)$$



1-Nearest Neighbor

◆ Prediction Step

closest_point := None

closest_distance := ∞

- for $i = 1, \dots, n$:
  - current_distance := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
  - if current_distance < closest_distance:
    - closest_distance := current_distance
    - closest_point := $\mathbf{x}^{[i]}$
- return f(closest_point)

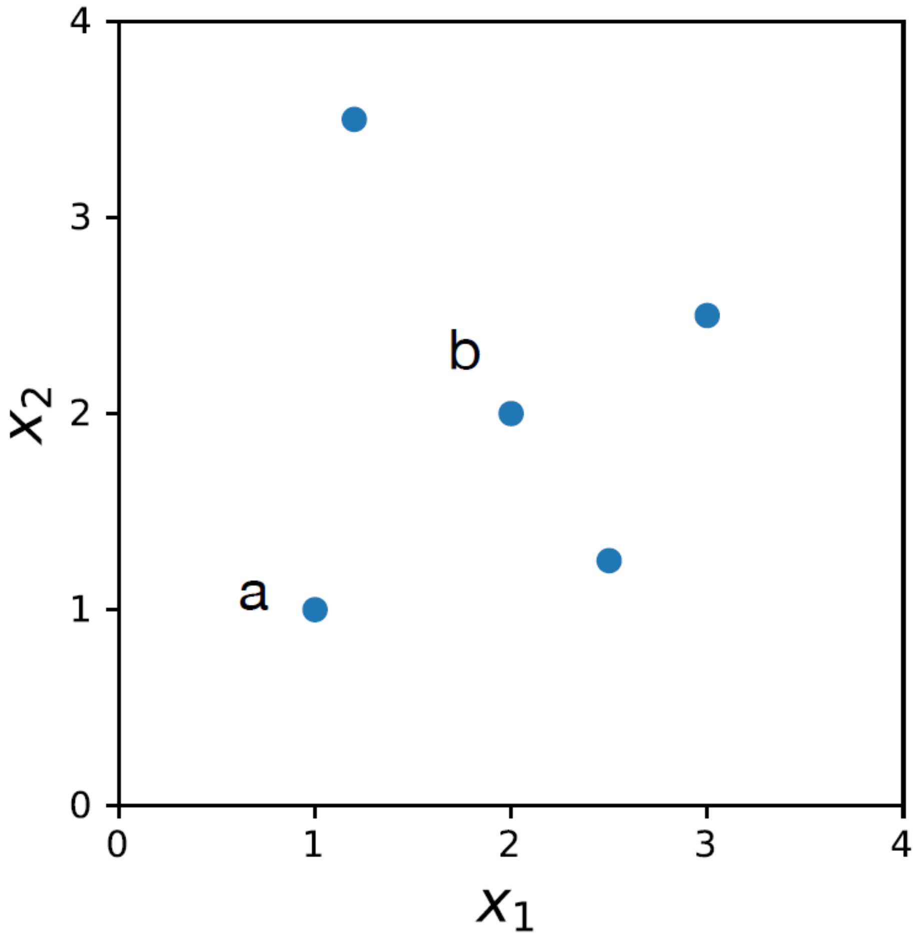
prediction $h(\mathbf{x}^{[q]})$ is the target value of `closest_point`

Commonly used: Euclidean Distance (L2)

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} \left(x_j^{[a]} - x_j^{[b]}\right)^2}$$

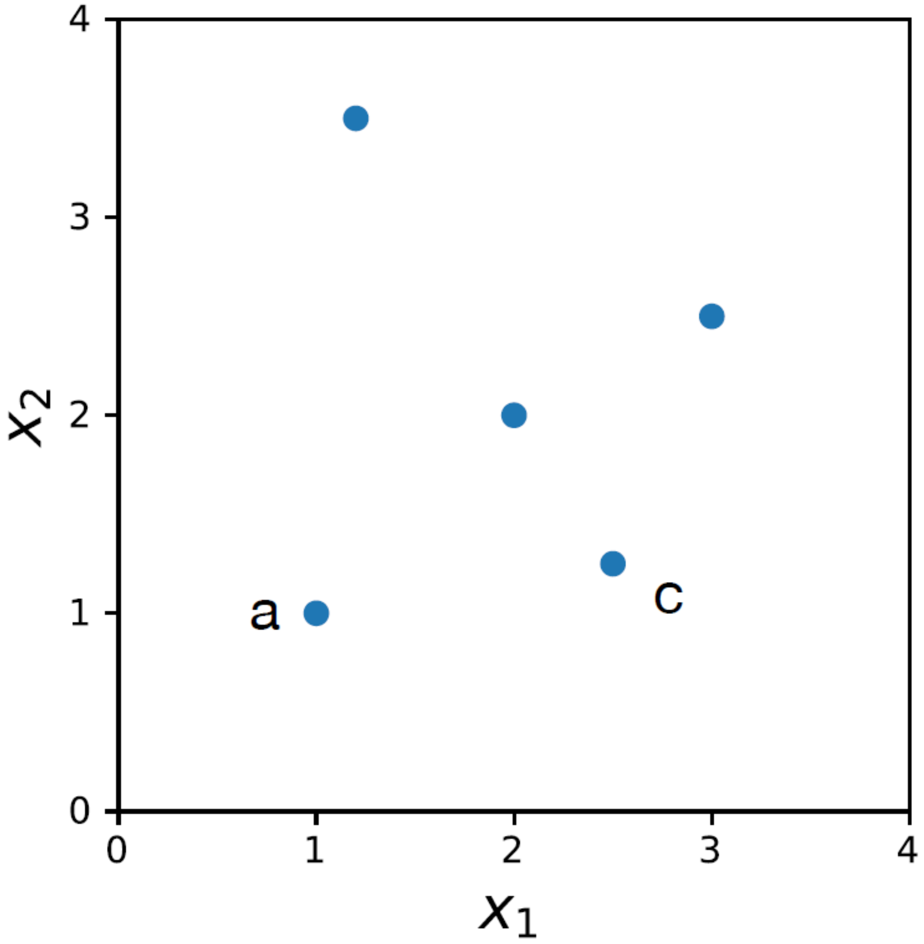
Nearest Neighbor Decision Boundary

◆ Decision Boundary Between (a) and (b)



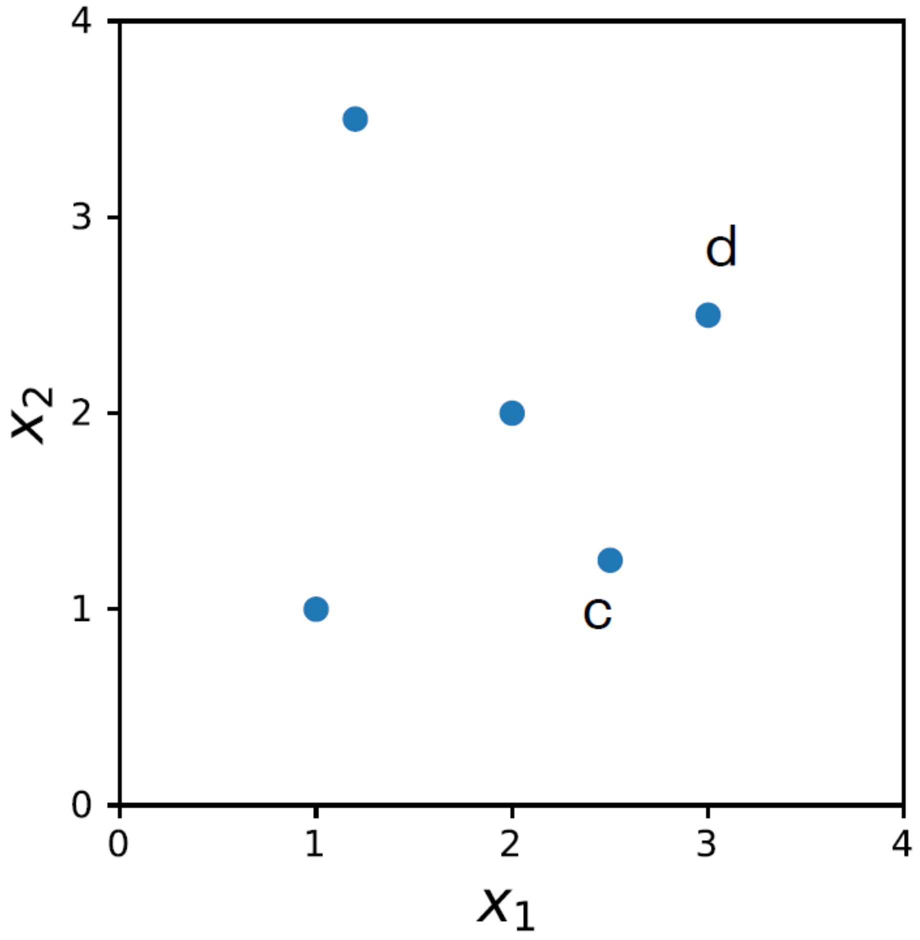
Nearest Neighbor Decision Boundary

◆ Decision Boundary Between (a) and (c)

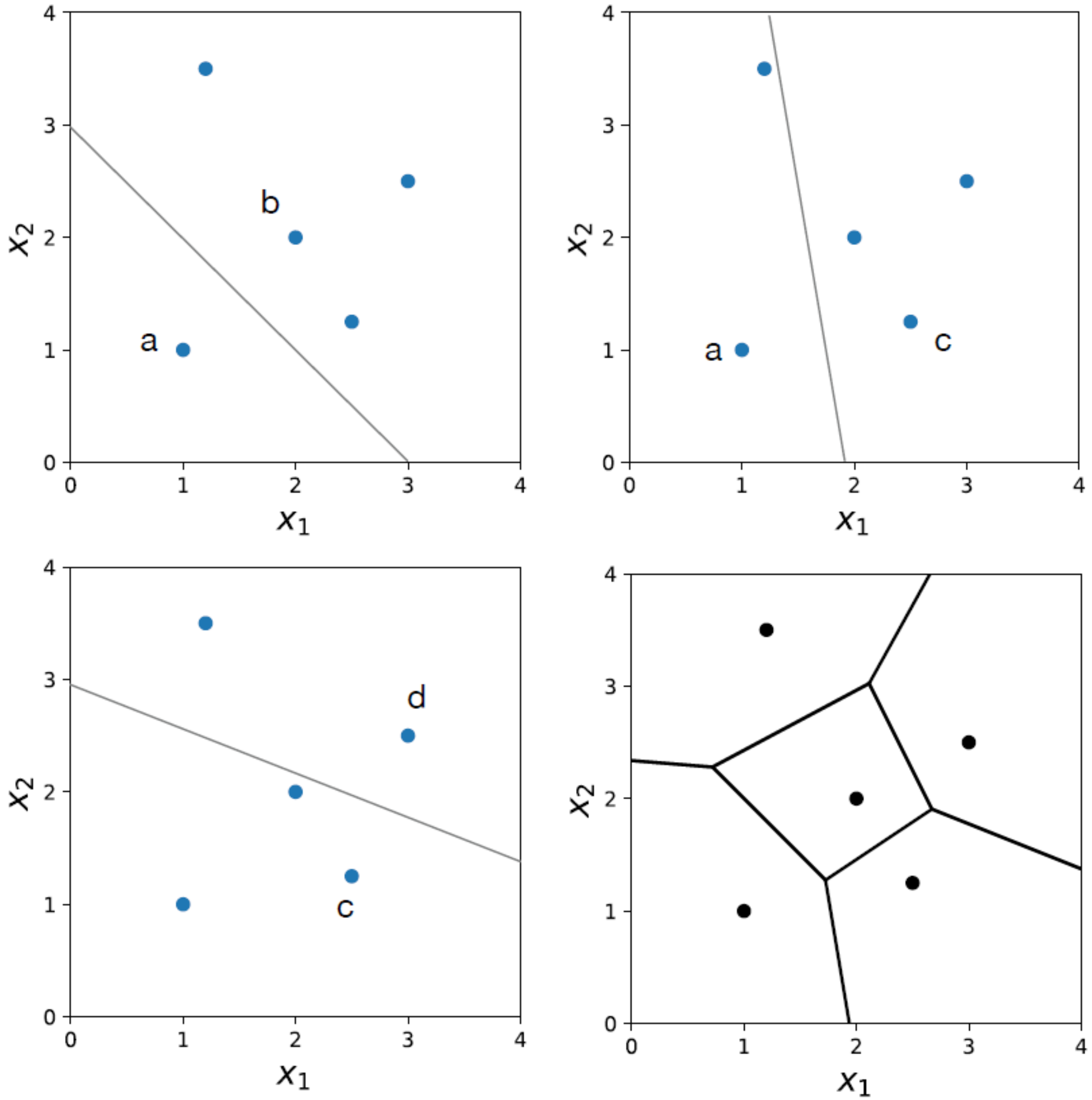


Nearest Neighbor Decision Boundary

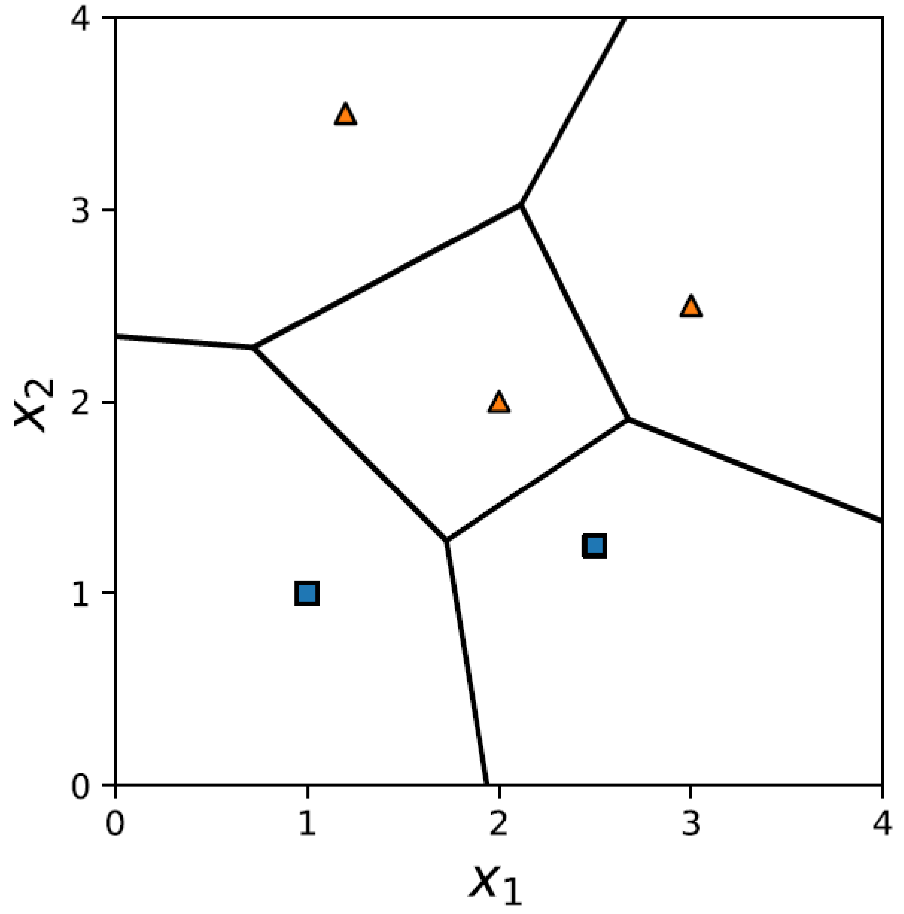
◆ Decision Boundary Between (c) and (d)

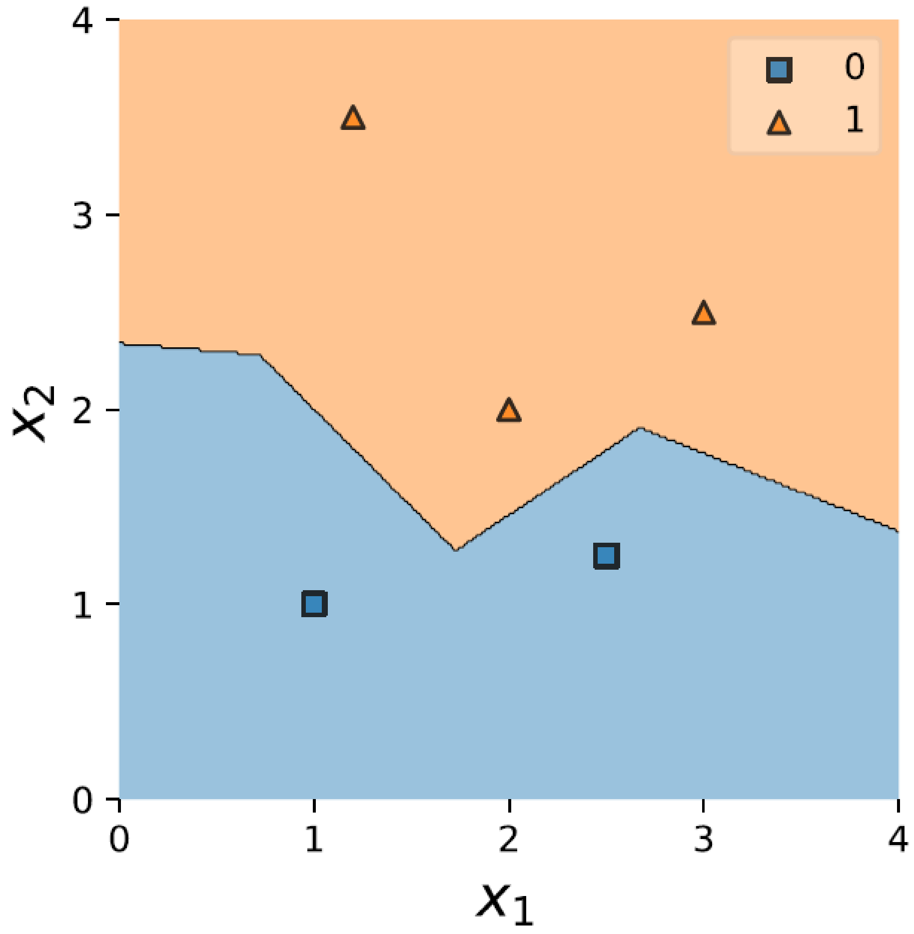


Decision Boundary 1NN



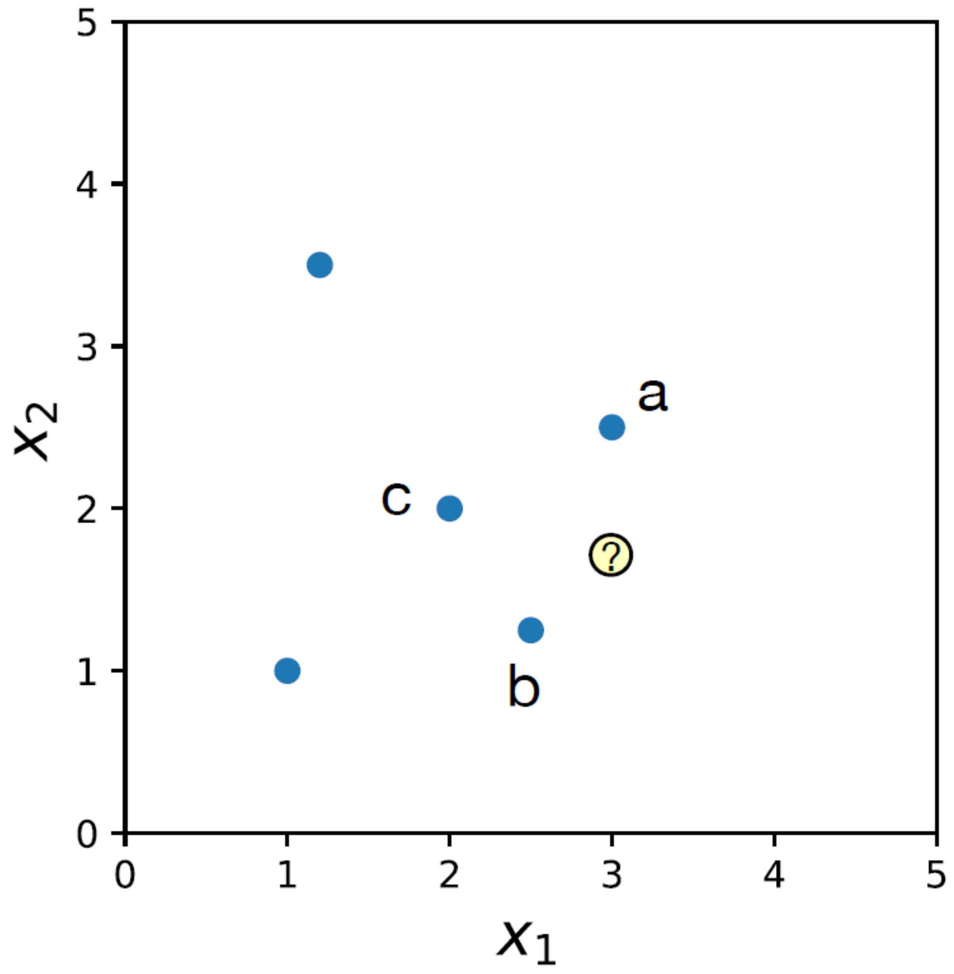
Decision Boundary 1NN





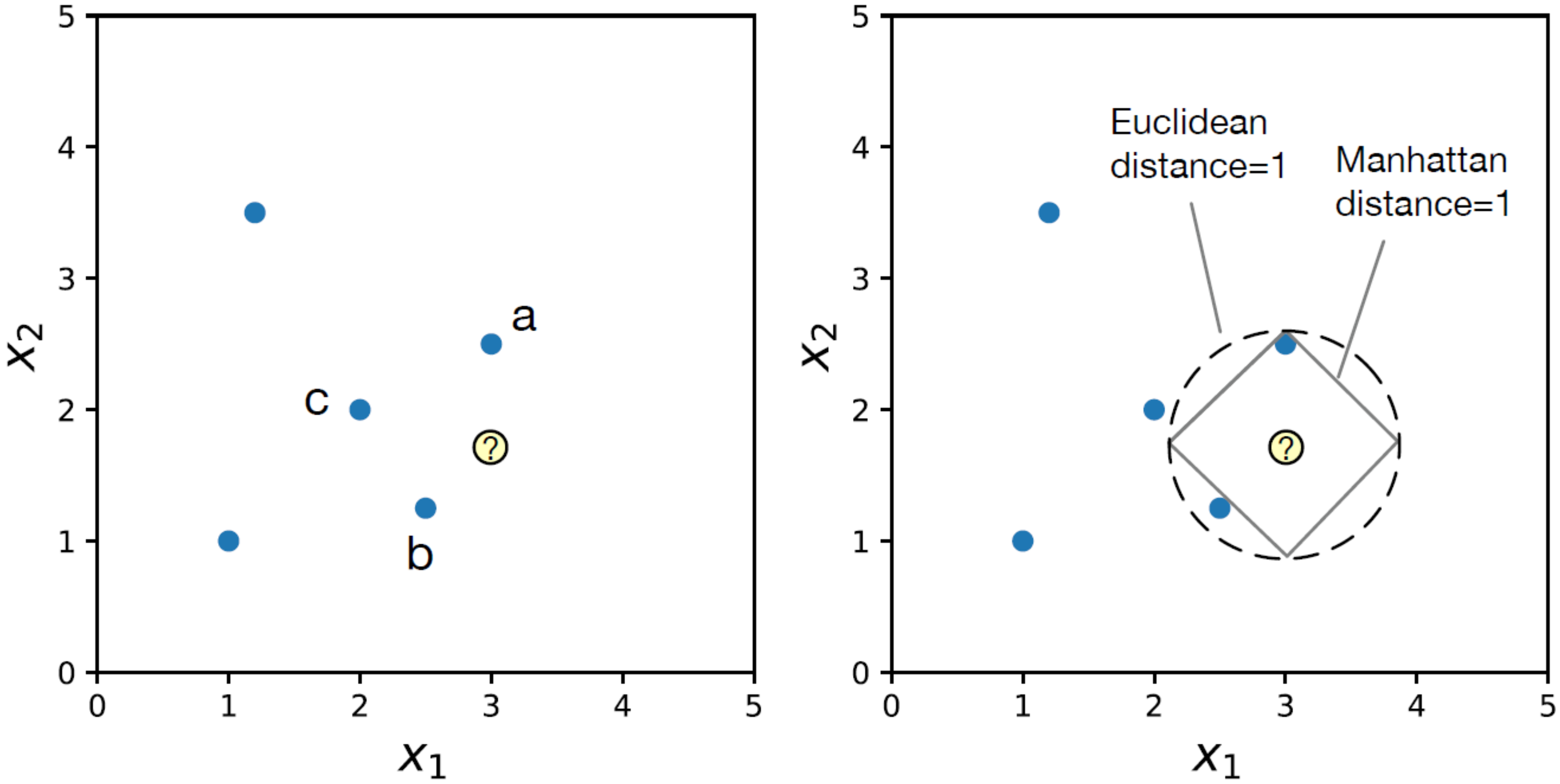
Distance Measure

◆ Which Point is Closest?



Distance Measure

- **Depends on the Distance Measure!**



Distance Measure

◆ Continuous Distance Measures

Euclidean

Manhattan

Minkowski: $d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \left[\sum_{j=1}^{m} \left(\left| x^{[a]} - x^{[b]} \right| \right)^{p} \right]^{\frac{1}{p}}$

Mahalanobis

...

◆ Discrete Distance Measures

Hamming: $d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sum_{j=1}^{m} \left| x^{[a]} - x^{[b]} \right|$

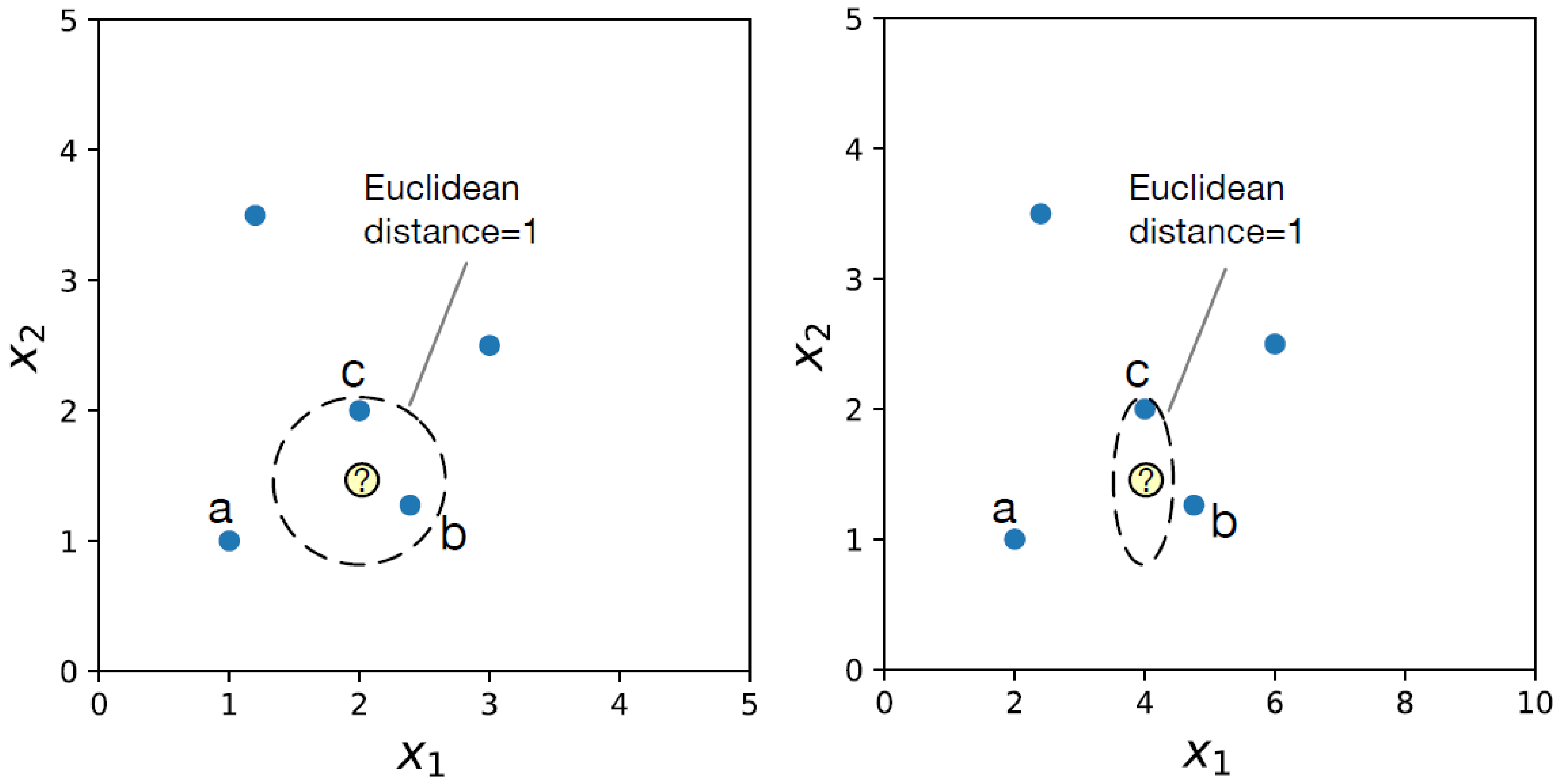
Jaccard/Tanimoto

Cosine similarity

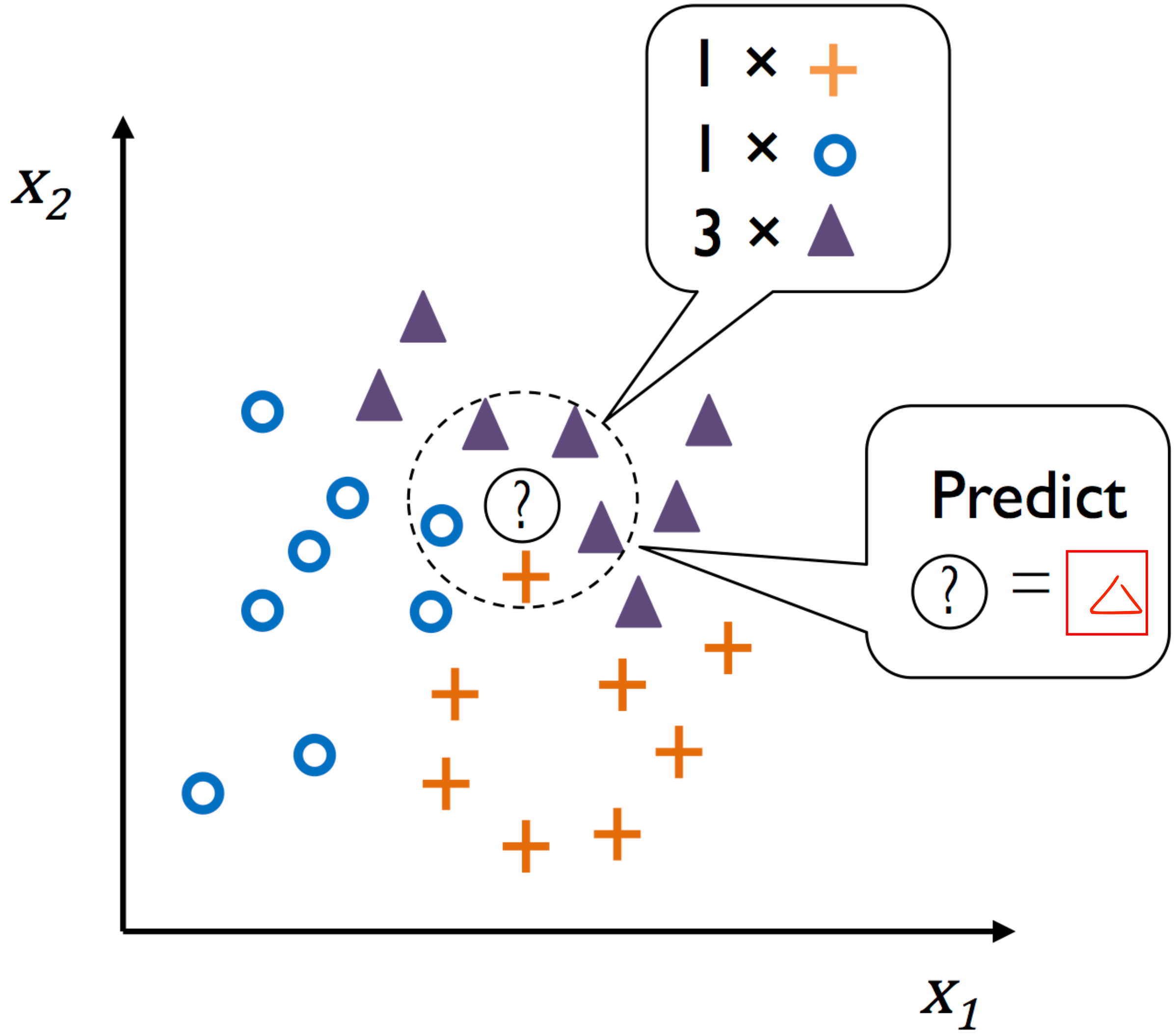
Dice

...

Feature Scaling



k-Nearest Neighbors



*k*NN for Classification

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]}) \rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]}) \rangle\} \qquad \mathcal{D}_k \subseteq \mathcal{D}$$

*k*NN for Classification

$$\mathscr{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]}) \rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]}) \rangle\} \qquad \mathscr{D}_k \subseteq \mathscr{D}$$

$$h(\mathbf{x}^{[q]}) = arg \max_{y \in \{1,\ldots,t\}} \sum_{i=1}^{k} \delta(y, f(\mathbf{x}^{[i]}))$$

$$\delta(a,b) = \begin{cases} 1, & \textbf{if } a = b, \\ 0, & \textbf{if } a \neq b\,. \end{cases}$$

Kronecker Delta function

kNN for Classification

$$\mathscr{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]}) \rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]}) \rangle\} \qquad \mathscr{D}_k \subseteq \mathscr{D}$$

$$h(\mathbf{x}^{[q]}) = arg \max_{y \in \{1,\ldots,t\}} \sum_{i=1}^{k} \delta(y, f(\mathbf{x}^{[i]}))$$

$$\delta(a,b) = \begin{cases} 1, & \textbf{if } a = b, \\ 0, & \textbf{if } a \neq b. \end{cases}$$

$$h(\mathbf{x}^{[t]}) = \textbf{mode}\left(\{f(\mathbf{x}^{[1]}), \ldots, f(\mathbf{x}^{[k]})\}\right)$$

kNN for Regression

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]})\rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]})\rangle\} \qquad \mathcal{D}_k \subseteq \mathcal{D}$$

$$h(\mathbf{x}^{[t]}) = \frac{1}{k}\sum_{i=1}^{k} f(\mathbf{x}^{[i]})$$

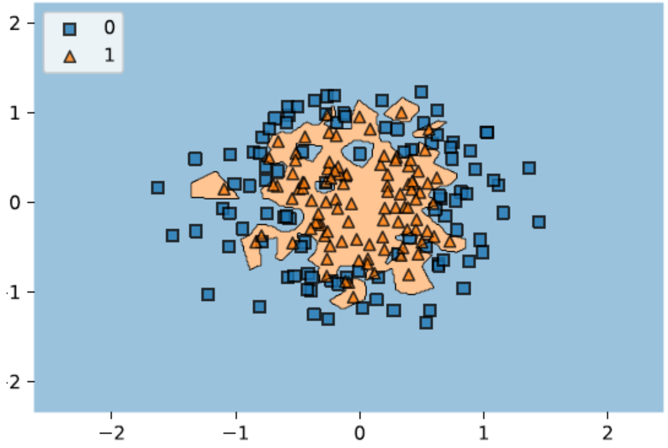
Hyperparameters

- Value of k
- Scaling of the feature axes
- Distance measure
- Weighting of the distance measure

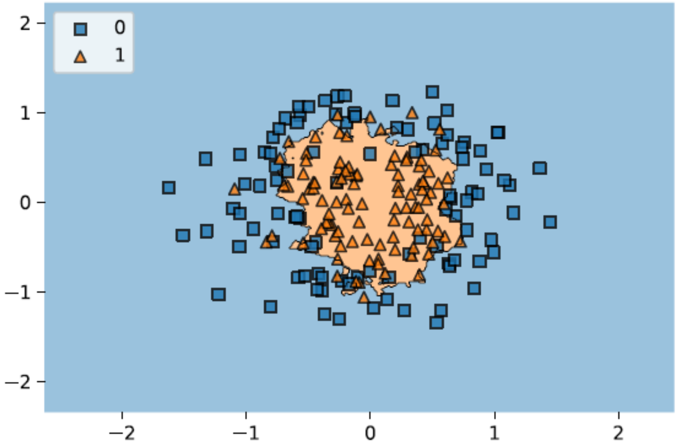
Hyperparameters

- ◆ $k \in \{1,3,7\}$

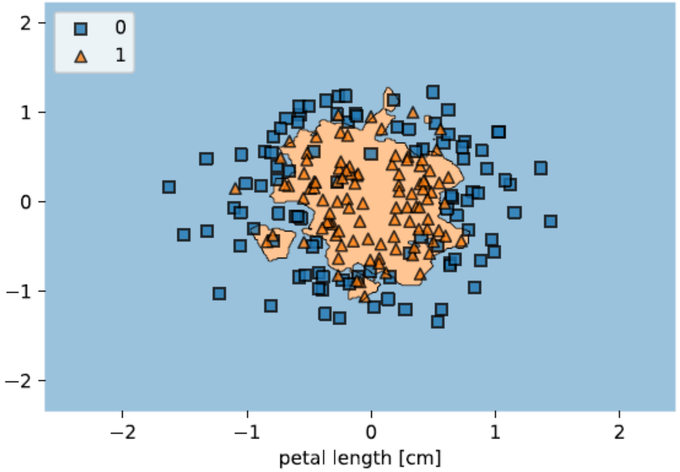
$k = \boxed{1}$



$k = \boxed{2}$



$k = \boxed{3}$



Feature-Weighting via Euclidean Distance

$$d_w(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} w_j \left(x_j^{[a]} - x_j^{[b]} \right)^2}$$

As a dot product:

$$\mathbf{c} = \mathbf{x}^{[a]} - \mathbf{x}^{[a]}, \quad (\mathbf{c}, \mathbf{x}^{[a]}\mathbf{x}^{[b]} \in \mathbb{R}^m)$$

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\mathbf{c}^t \mathbf{c}}$$

$$d_w(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \mathbf{c}^T \mathbf{W} \mathbf{c}, \qquad \mathbf{W} \in \mathbb{R}^{m \times m} = \textbf{diag}(w_1, w_2, \ldots, w_m)$$

Distance-weighted kNN

$$h(\mathbf{x}^{[t]}) = arg \max_{j \in \{1,\ldots,p\}} \sum_{i=1}^{k} w^{[i]} \delta(j, f(\mathbf{x}^{[i]}))$$

$$w^{[i]} = \frac{1}{d(\mathbf{x}^{[i]}, \mathbf{x}^{[t]})^2}$$

Small constant to avoid zero division
or set $h(\mathbf{x}) = f(\mathbf{x})$

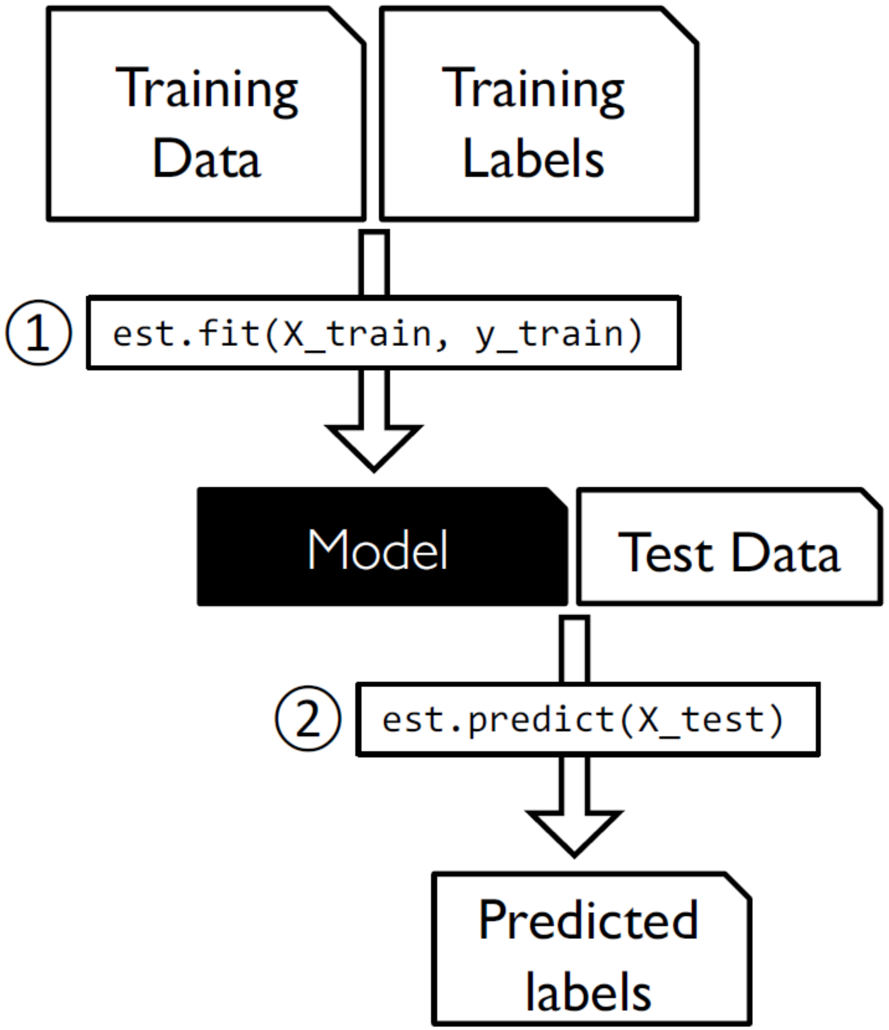
The "Main" Machine Learning Library for Python

http://scikit-learn.org

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *Journal of machine learning research* 12.Oct (2011): 2825-2830.

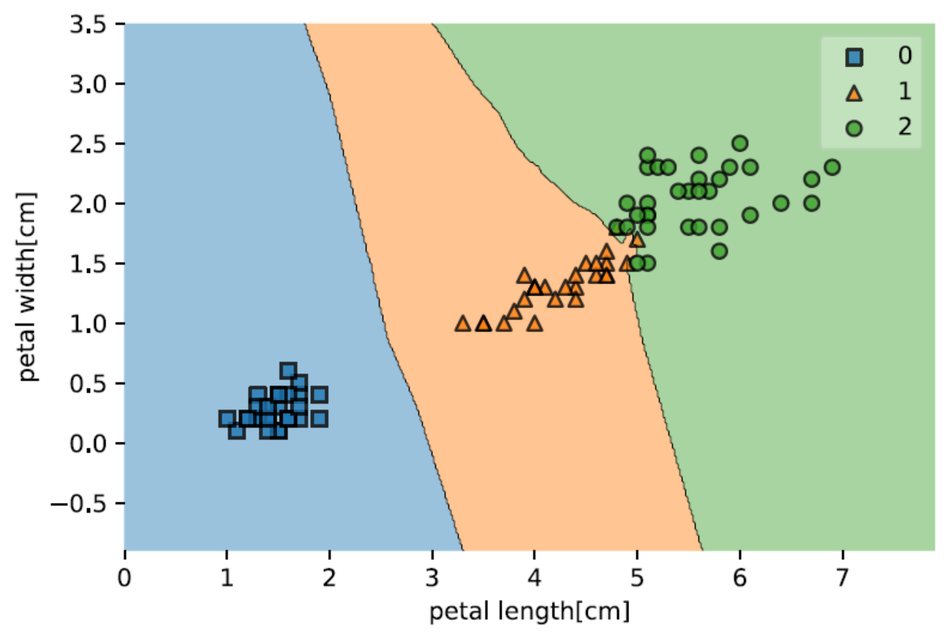
The Scikit-learn Estimator API (an OOP Paradigm)

```python
class SupervisedEstimator(...):

    def __init__(self, hyperparam_1, ...):
        self.hyperparm_1
        ...

    def fit(self, X, y):
        ...
        self.fit_attribute_
        return self

    def predict(self, X):
        ...
        return y_pred

    def score(self, X, y):
        ...
        return score

    def _private_method(self):
        ...
    ...
```

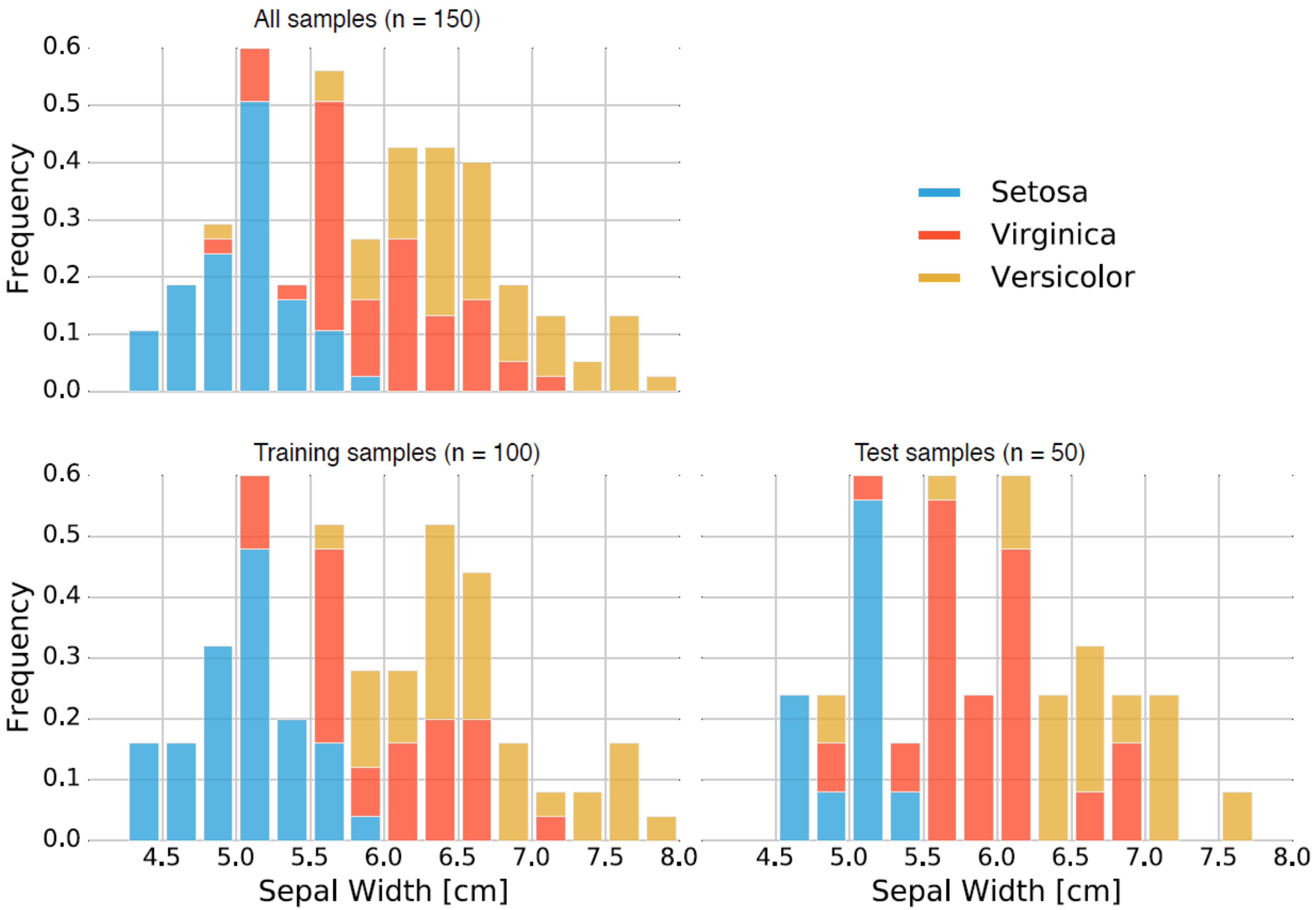


A 3-Nearest Neighbor Classifier & 2 Iris Features

```
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions

knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train[:, 2:], y_train)
plot_decision_regions(X_train[:, 2:], y_train, knn_model)
plt.xlabel('petal length[cm]')
plt.ylabel('petal width[cm]')
plt.savefig('images/decisionreg.pdf')
plt.show()
```



Issues with Random Subsampling …



Stratified Splits

◆ 각 분할에서 클래스 레이블(또는 카테고리)의 분포를 유지하려는 데이터 분할 방법

```
from sklearn.model_selection import train_test_split

X_temp, X_test, y_temp, y_test = \
        train_test_split(X, y, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y)
np.bincount(y_temp)
```

```
array([40, 40, 40])
```

```
X_train, X_valid, y_train, y_valid = \
        train_test_split(X_temp, y_temp, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y_temp)
X_train.shape
```

```
(96, 4)
```

```
np.bincount(y_train) ?
```

Normalization: Min-Max Scaling

◆ Normalization(정규화)

$$x_{norm}^{[i]} = \frac{x^{[i]} - x_{min}}{x_{max} - x_{min}}$$

- [0~1]
- 각 데이터 포인트의 상대적 크기 보존

Normalization: Min-Max Scaling

◆ Normalization(정규화)

$$x_{norm}^{[i]} = \frac{x^{[i]} - x_{min}}{x_{max} - x_{min}}$$

```
x = np.arange(6).astype(float)
x
```

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

```
x_norm = (x - x.min()) / (x.max() - x.min())
x_norm
```

```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

Normalization: Standardization

◆ Standardization (표준화)

$$x_{std}^{[i]} = \frac{x^{[i]} - \mu_x}{\sigma_x}$$

- 평균 0, 표준편차 1
- Z-score

Normalization: Standardization

◆ Standardization (표준화)

$$x_{std}^{[i]} = \frac{x^{[i]} - \mu_x}{\sigma_x}$$

```
x = np.arange(6).astype(float)
x
```

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

```
x_std = (x - x.mean()) / x.std()
x_std
```

```
array([-1.46385011, -0.87831007, -0.29277002,  0.29277002,  0.87831007,
        1.46385011])
```

Normalization: Standardization

◆ Standard ~~variation~~ (표준 편차)

$$x_{std}^{[i]} = \frac{x^{[i]} - \mu_x}{\sigma_x}$$

```
df = pd.DataFrame([1, 2, 1, 2, 3, 4])
df[0].std()
```

1.1690451944500122

```
df[0].values.std()
```

1.0671873729054748

Sample vs Population Standard Deviation

- ◆ 표본 표준편차 (sample standard deviation)

$$s_x = \sqrt{\frac{1}{n-1} \sum_{n}^{i=1} (x^{[i]} - \bar{x})^2}$$

- ◆ 모집단 표준편차 (population standard deviation)

$$\sigma_x = \sqrt{\frac{1}{n} \sum_{n}^{i=1} (x^{[i]} - \mu_x)^2}$$

Sample vs Population Standard Deviation

```
df = pd.DataFrame([1, 2, 1, 2, 3, 4])
df[0].std()
```

1.1690451944500122

```
df[0].values.std()
```

1.0671873729054748

```
df[0].values.std(ddof=1)
```

1.1690451944500122

$$s_x = \sqrt{\frac{1}{n-1}\sum_{n}^{i=1}(x^{[i]} - \bar{x})^2}$$

$$\sigma_x = \sqrt{\frac{1}{n}\sum_{n}^{i=1}(x^{[i]} - \mu_x)^2}$$

ddof : 자유도 감소(Delta Degrees of Freedom)를 나타내는 매개변수

Scaling Validation and Test Sets

◆ Do not recalculate “new” mean and standard deviation

```
mu, sigma = X_train.mean(axis=0), X_train.std(axis=0)

X_train_std = (X_train - mu) / sigma
X_valid_std = (X_valid - mu) / sigma
X_test_std = (X_test - mu) / sigma
```

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Standardize:

- example1: -1.21 -> class 2
- example2: 0.00 -> class 2
- example3: 1.21 -> class 1

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

Standardize:

- example1: -1.21 -> class 2
- example2: 0.00 -> class 2
- example3: 1.21 -> class 1

Assume you have the classification rule:

$$h(z) = \begin{cases} \text{class } 2 & \text{if } z \leq 0.6 \\ \text{class } 1 & \text{otherwise} \end{cases}$$

Scaling Validation and Test Sets

$$h(z) = \begin{cases} \text{class 2} & \text{if } z \leq 0.6 \\ \text{class 1} & \text{otherwise} \end{cases}$$

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

*Given 3 **NEW** examples:*

- example4: 5 cm -> class ?
- example5: 6 cm -> class ?
- example6: 7 cm -> class ?

Estimate "new" mean and std.: 6, 0.82

- *example5: -1.21 -> class 2*
- *example6: 0.00 -> class 2*
- *example7: 1.21 -> class 1*

Scaling Validation and Test Sets

Given 3 training examples:

- example1: 10 cm -> class 2
- example2: 20 cm -> class 2
- example3: 30 cm -> class 1

Estimate:

mean: 20 cm
standard deviation: 8.2 cm

$$h(z) = \begin{cases} \text{class 2} & \text{if } z \leq 0.6 \\ \text{class 1} & \text{otherwise} \end{cases}$$

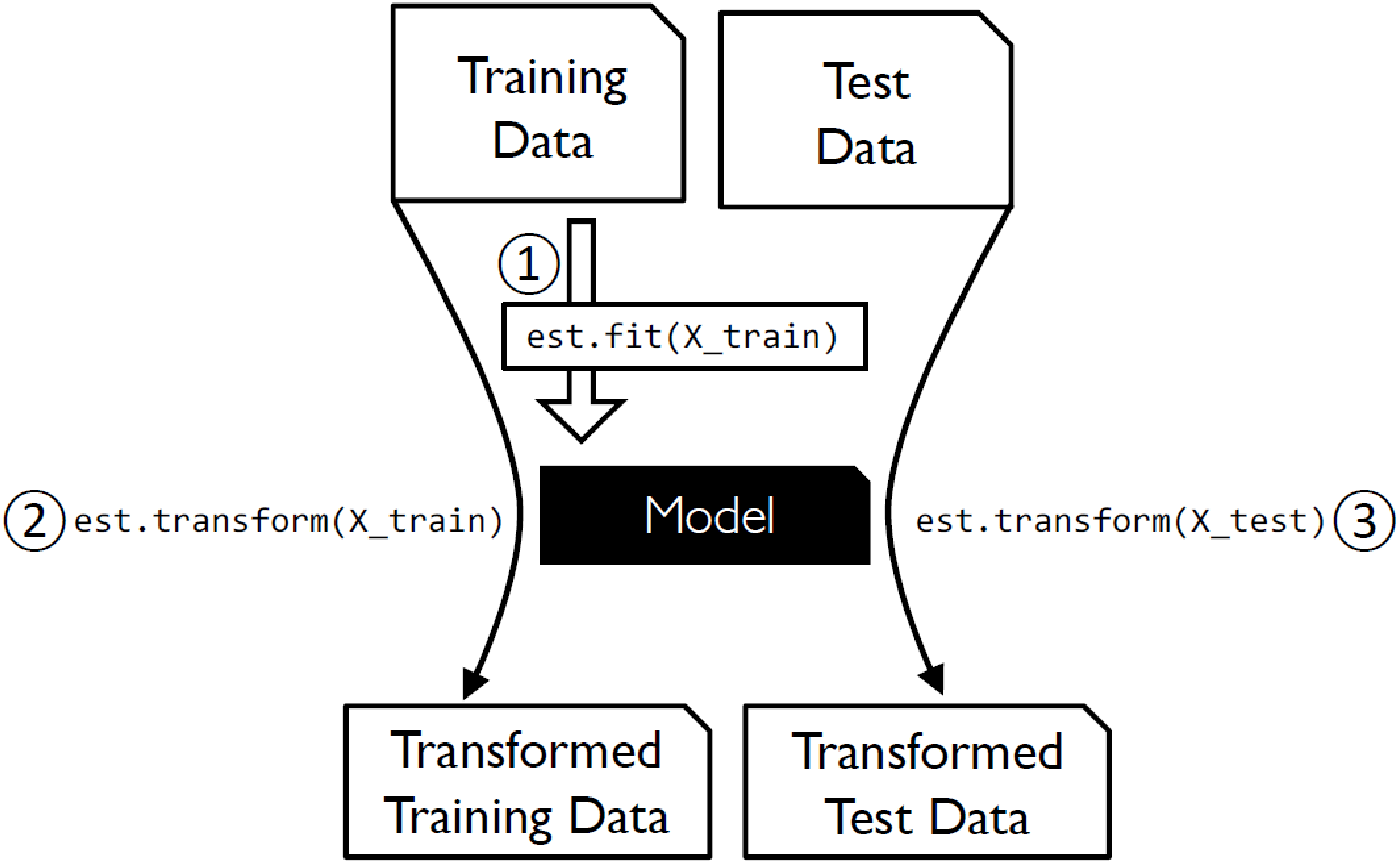
- example4: 5 cm -> class ?
- example5: 6 cm -> class ?
- example6: 7 cm -> class ?

~~*Estimate "new" mean and std.:*~~

- ~~*example5: -1.21 -> class 2*~~
- ~~*example6: 0.00 -> class 2*~~
- ~~*example7: 1.21 -> class 1*~~

- example5 : -1.84
- example6 : -1.72
- example7 : -1.60

The Scikit-Learn Transformer API



The Scikit-Learn Transformer API

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()    # StandardScaler 클래스의 인스턴스를 생성하고 scaler 변수에 할당
scaler.fit(X_train)          # X_train의 평균과 표준 편차 계산
X_train_std = scaler.transform(X_train) # Standardization 수행
X_valid_std = scaler.transform(X_valid)
X_test_std = scaler.transform(X_test)
```

Working with Categorical Data

◆ Categorical(범주형) Data

```
df = pd.read_csv('categoricaldata.csv')
df
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XXL	15.3	class1

Categorical Data -> Ordinal Data

◆ Ordinal(순서형) data : 상대적인 순서 또는 순위가 존재

```
mapping_dict = {'M': 2,
                'L': 3,
                'XXL': 5}

df['size'] = df['size'].map(mapping_dict)
df
```

	color	size	price	classlabel
0	green	2	10.1	class1
1	red	3	13.5	class2
2	blue	5	15.3	class1

Categorical Data -> Nominal Data (Class Labels)

◆ Nominal(명목형) Data : 범주 간에 순서나 순위가 없음. 범주간 동등 관계

```
from sklearn.preprocessing import LabelEncoder

                                  # 범주형 데이터를 정수형으로 인코딩

le = LabelEncoder()
df['classlabel'] = le.fit_transform(df['classlabel'])
df
```

	color	size	price	classlabel
0	green	2	10.1	class1
1	red	3	13.5	class2
2	blue	5	15.3	class1

→

	color	size	price	classlabel
0	green	2	10.1	0
1	red	3	13.5	1
2	blue	5	15.3	0

Categorical Data -> Nominal Data (Class Labels)

◆ `.get_dummies()` : 범주형 데이터를 가지고 있는 열(칼럼)을 원-핫(one-hot) 인코딩하는 데에 사용

	color	size	price	classlabel
0	green	2	10.1	0
1	red	3	13.5	1
2	blue	5	15.3	0

```
pd.get_dummies(df)            #pd.get_dummies(df, columns=['color'])
```

	size	price	classlabel	color_blue	color_green	color_red
0	2	10.1	0	0	1	0
1	3	13.5	1	0	0	1
2	5	15.3	0	1	0	0

Categorical Data -> Nominal Data (Class Labels)

```
pd.get_dummies(df)
```

	size	price	classlabel	color_blue	color_green	color_red
0	2	10.1	0	0	1	0
1	3	13.5	1	0	0	1
2	5	15.3	0	1	0	0

다중공선성
(collinearity)

```
pd.get_dummies(df, drop_first=True)
```

	size	price	classlabel	color_green	color_red
0	2	10.1	0	1	0
1	3	13.5	1	0	1
2	5	15.3	0	0	0

Dealing with Missing Data

```
df = pd.read_csv('missingdata.csv')
df
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

Dealing with Missing Data

```
df = pd.read_csv('missingdata.csv')
df
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

```
# drop rows with missing values:

df.dropna(axis=0)
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

```
# drop columns with missing values:

df.dropna(axis=1)
```

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

Dealing with Missing Data

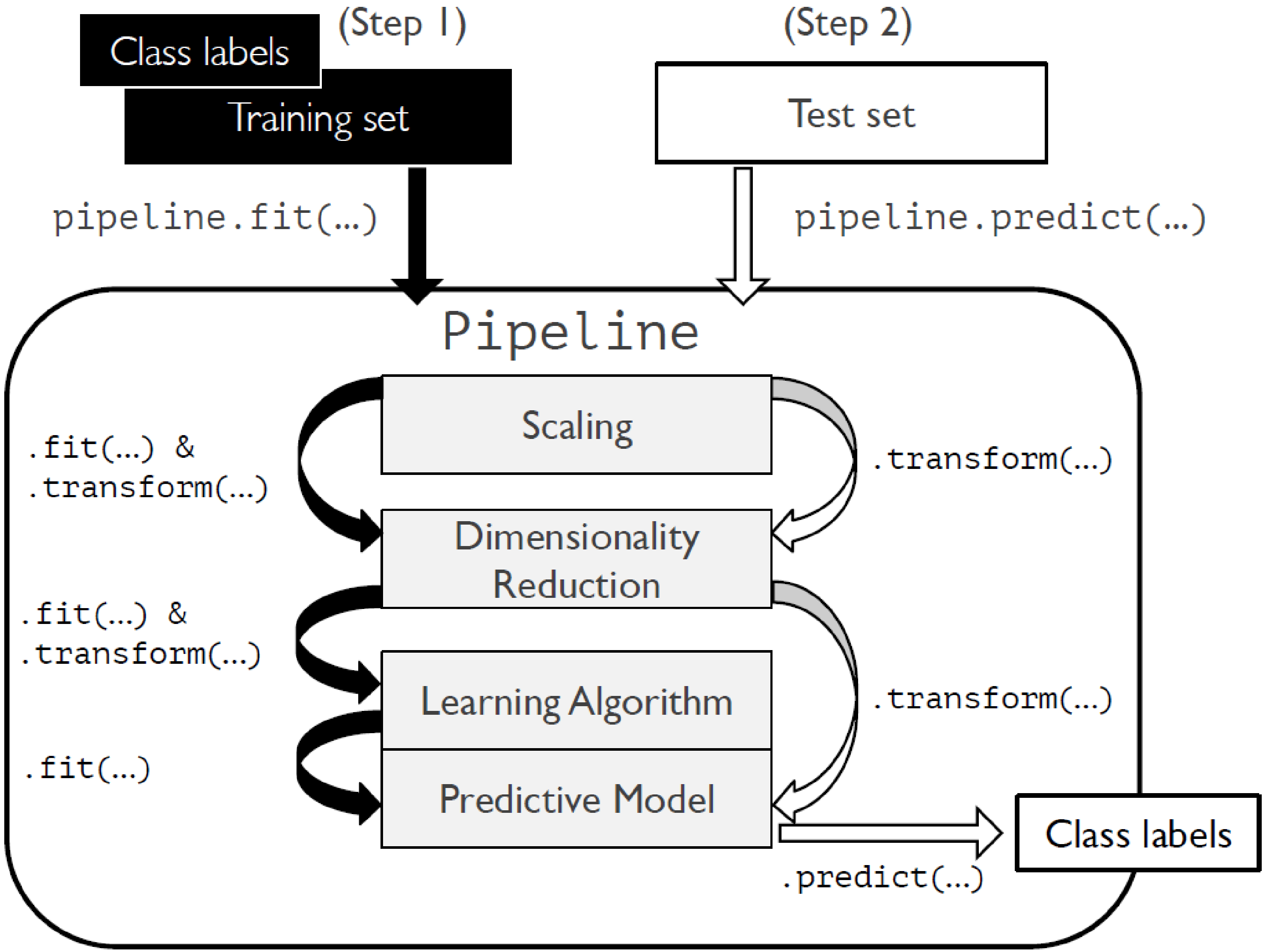
```
df = pd.read_csv('missingdata.csv')
df
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

```
from sklearn.impute import SimpleImputer


imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
X = df.values
X = imputer.fit_transform(df.values)
X
```

```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

Scikit-Learn Pipelines



Pipeline : 데이터전처리와 학습모델을 하나로 묶음

Scikit-Learn Pipelines

```
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier(n_neighbors=3))
```

```
pipe
```

```
Pipeline(memory=None,
     steps=[('standardscaler', StandardScaler(copy=True, with_mean=Tr
ue, with_std=True)), ('kneighborsclassifier', KNeighborsClassifier(al
gorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform'))])
```

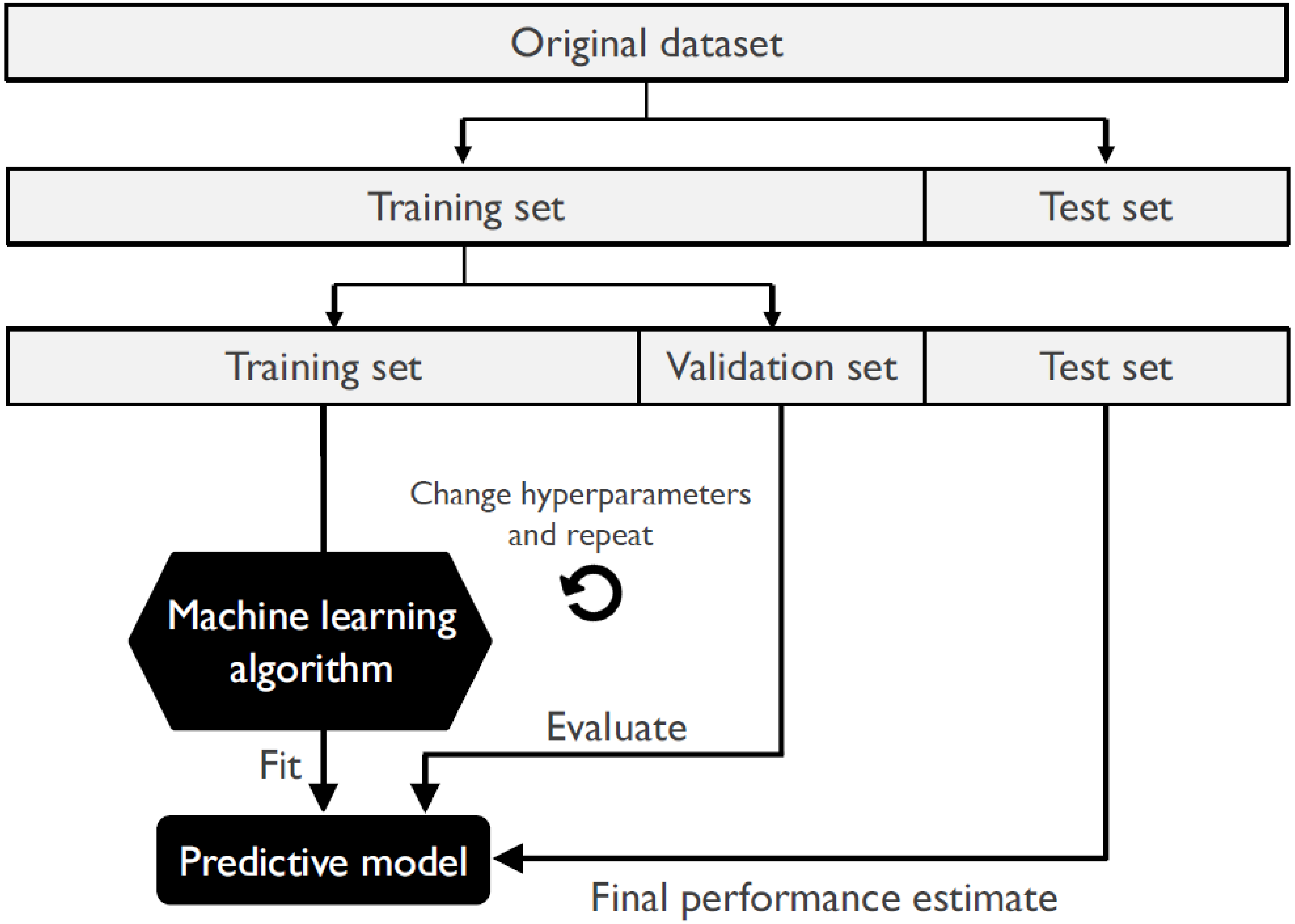
Scikit-Learn Pipelines

```
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier(n_neighbors=3))
```

```
pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

```
array([1, 0, 2, 2, 0, 0, 2, 1, 2, 0, 0, 2, 2, 1, 2, 1, 0, 0, 0, 0, 0,
2,
       2, 1, 2, 2, 1, 1, 1, 1])
```

Model Selection: Simple Holdout Method



Model Selection: Simple Holdout Method

```
from sklearn.model_selection import GridSearchCV
from mlxtend.evaluate import PredefinedHoldoutSplit
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris

####### 이전 cell을 사용하지 않으면, 아래 import 추가 #######
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
######################################################################
```

```
iris = load_iris()
X, y = iris.data, iris.target

X_train_valid, X_test, y_train_valid, y_test = train_test_split(X, y,
                                                                test_size=0.2, shuffle=True,
                                                                random_state=123, stratify=y)

train_ind, valid_ind = train_test_split(np.arange(X_train_valid.shape[0]),
                                        test_size=0.2, shuffle=True,
                                        random_state=123, stratify=y_train_valid)
```

참고) numpy version이 안 맞을 때
import numpy as np
np.bool = np.bool_

Model Selection: Simple Holdout Method

```
pipe = make_pipeline(StandardScaler(),
                     KNeighborsClassifier())

params = {'kneighborsclassifier__n_neighbors': [1, 3, 5],
          'kneighborsclassifier__p': [1, 2]}

split = PredefinedHoldoutSplit(valid_indices=valid_ind)

grid = GridSearchCV(pipe,
                    param_grid=params,
                    cv=split)
```

neighborsclassifier__p=1: 맨해튼 거리(Manhattan distance)
neighborsclassifier__p=2: 유클리디안 거리(Euclidean distance)
…

Model Selection: Simple Holdout Method

```
grid.cv_results_
```

```
{'mean_fit_time': array([0.00151896, 0.00076985, 0.00071883, 0.00068808, 0.00069523,
        0.00067973]),
 'std_fit_time': array([0., 0., 0., 0., 0., 0.]),
 'mean_score_time': array([0.00145102, 0.00129414, 0.00130701, 0.00129294, 0.00127792,
        0.0012753 ]),
 'std_score_time': array([0., 0., 0., 0., 0., 0.]),
 'param_kneighborsclassifier__n_neighbors': masked_array(data=[1, 1, 3, 3, 5, 5],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_kneighborsclassifier__p': masked_array(data=[1, 2, 1, 2, 1, 2],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'kneighborsclassifier__n_neighbors': 1,
   'kneighborsclassifier__p': 1},
  {'kneighborsclassifier__n_neighbors': 1, 'kneighborsclassifier__p': 2},
  {'kneighborsclassifier__n_neighbors': 3, 'kneighborsclassifier__p': 1},
  {'kneighborsclassifier__n_neighbors': 3, 'kneighborsclassifier__p': 2},
  {'kneighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 1},
  {'kneighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 2}],
 'split0_test_score': array([0.9       , 0.96666667, 0.96666667, 0.93333333, 0.9       ,
        0.9       ]),
 'mean_test_score': array([0.9       , 0.96666667, 0.96666667, 0.93333333, 0.9       ,
        0.9       ]),
 'std_test_score': array([0., 0., 0., 0., 0., 0.]),
 'rank_test_score': array([4, 1, 1, 3, 4, 4], dtype=int32)}
```

Model Selection: Simple Holdout Method

```
print(grid.best_score_)
print(grid.best_params_)
```

```
0.9666666666666667
{'kneighborsclassifier__n_neighbors': 1, 'kneighborsclassifier__p': 2}
```

```
clf = grid.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.2f%%' % (clf.score(X_test, y_test)*100))
```

```
Test accuracy: 100.00%
```

감사합니다.