

# 빅데이터분석 실습

4주 지도학습 알고리즘 1

데이터 사이언스 전공

담당교수: 곽철완

# 강의 내용

- 사이킷런 연습
- k-최근접 이웃 알고리즘
- 지도학습 기초

# 사이킷런을 이용한 머신 러닝 연습

## ■ 기본 라이브러리

- 분석을 위해 기본 라이브러리를 import한다

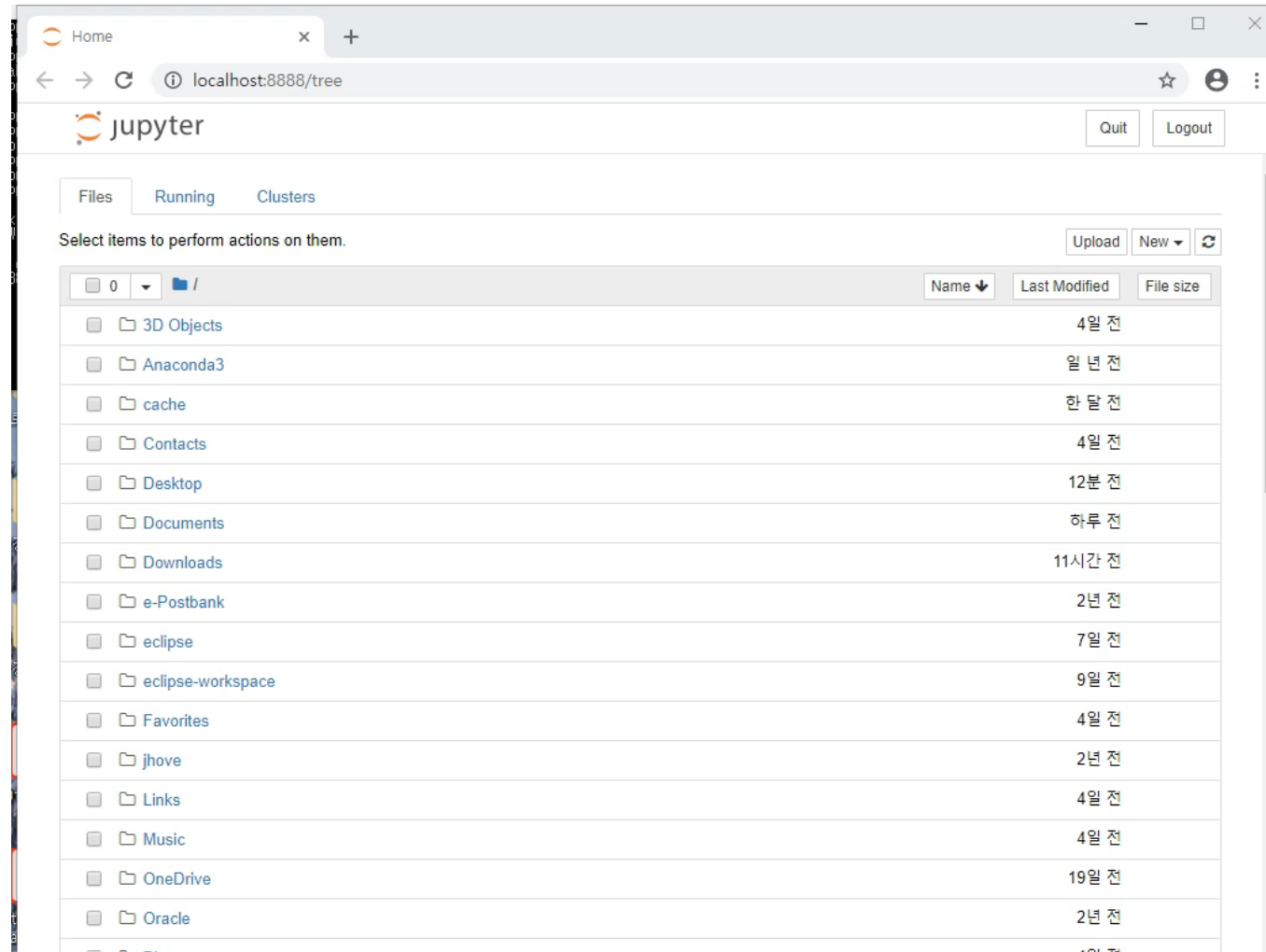
```
%matplotlib inline
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
import sklearn
```

## ■ 분석할 데이터 세트 적재

- iris 데이터 세트(scikit-learn의 datasets 모듈에 포함)
- load\_iris 함수 사용하여 데이터 세트 이름을 iris\_dataset 객체로 지정

```
from sklearn.datasets import load_iris  
iris_dataset = load_iris()
```

```
In [2]: from sklearn.datasets import load_iris  
iris_dataset = load_iris()
```



- 데이터 세트를 key( ) 함수를 이용하여 확인

```
print("iris_dataset의 키:\n", iris_dataset.keys())
```

```
In [3]: print("iris_dataset의 키:\n", iris_dataset.keys())
```

```
iris_dataset의 키:  
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

- DESCR이란?
  - 데이터 세트에 대한 설명

```
print(iris_dataset['DESCR'][:193]+ "\n...")
```

```
In [4]: print(iris_dataset['DESCR'][:193]+ "\n...")
```

```
.. _iris_dataset:
```

```
Iris plants dataset
```

```
-----
```

```
**Data Set Characteristics:**
```

```
 :Number of Instances: 150 (50 in each of three classes)
```

```
 :Number of Attributes: 4 numeric, pre
```

```
 ...
```

[ :193 ] 은 처음부터  
193번째 자리까지만 제시  
표시이다

- [ :193 ] ← 무엇일까?

```
In [21]: print(iris_dataset['DESCR'] + "\n...")
```

```
.. _iris_dataset:
```

```
Iris plants dataset
```

```
-----  
**Data Set Characteristics:**
```

click to scroll output; double click to hide

```
(50 in each of three classes)  
:Number of Attributes: 4 numeric, predictive attributes and the class  
:Attribute Information:  
  - sepal length in cm  
  - sepal width in cm  
  - petal length in cm  
  - petal width in cm  
  - class:  
    - Iris-Setosa  
    - Iris-Versicolour  
    - Iris-Virginica
```

```
:Summary Statistics:
```

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

```
:Missing Attribute Values: None  
:Class Distribution: 33.3% for each of 3 classes.  
:Creator: R.A. Fisher  
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)  
:Date: July, 1988
```

```
The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
```



- target\_names: iris 품종의 이름, 즉 타깃의 이름을 표시

```
print("타깃의 이름:", iris_dataset['target_names'])
```

```
In [7]: print("타깃의 이름:", iris_dataset['target_names'])  
타깃의 이름: ['setosa' 'versicolor' 'virginica']
```

- feature\_names: 피처(변수)를 설명하는 문자열

```
In [8]: print("피처의 이름:\n", iris_dataset['feature_names'])  
피처의 이름:  
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- data: 데이터에 대한 설명

```
In [9]: print("data의 타입:", type(iris_dataset['data']))  
data의 타입: <class 'numpy.ndarray'>
```

type(): 타입에 대한 질문, 사이킷런의 데이터세트는 NumPy 배열이다

```
In [10]: print("data의 크기:", iris_dataset['data'].shape)  
data의 크기: (150, 4)
```

.shape : 데이터 크기-행과 열(피쳐)

```
In [11]: print("data의 처음 다섯 행:\n", iris_dataset['data'][:5])  
  
data의 처음 다섯 행:  
[[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]]
```

[:5]의 의미는?

- target: 분류할 클래스(class)로 숫자로 표시(iris는 3가지 품종이므로, 0,1,2로 구성)

```
In [13]: print("target의 타입:", type(iris_dataset['target']))
```

```
target의 타입: <class 'numpy.ndarray'>
```

## type(): 타입에 대한 질문

```
In [14]: print("target의 크기:", iris_dataset['target'].shape)
```

$$\text{target}_{\exists} \models \exists \mathcal{I} : (150,)$$

```
In [15]: print("타겟:\n", iris_dataset['target'])
```

타깃:

[illegible]

## ■ 학습용 데이터 vs 평가용 데이터

- 모델 작성을 위해 학습용과 평가용으로 구분
- train\_test\_split( ) 함수는 학습용 75%, 평가용 25%로 구분하도록 세팅

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

```
In [17]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(
              iris_dataset['data'], iris_dataset['target'], random_state=0)
```

- 학습용 데이터 세트 크기 확인

```
In [18]: print("X_train 크기:", X_train.shape)
         print("y_train 크기:", y_train.shape)
```

```
X_train 크기: (112, 4)
y_train 크기: (112,)
```

X\_train : 훈련용 데이터 세트  
y\_train: 훈련용 레이블 데이터

- 평가용 데이터 세트 크기 확인

```
In [19]: print("X_test 크기:", X_test.shape)
         print("y_test 크기:", y_test.shape)
```

```
X_test 크기: (38, 4)
y_test 크기: (38,)
```

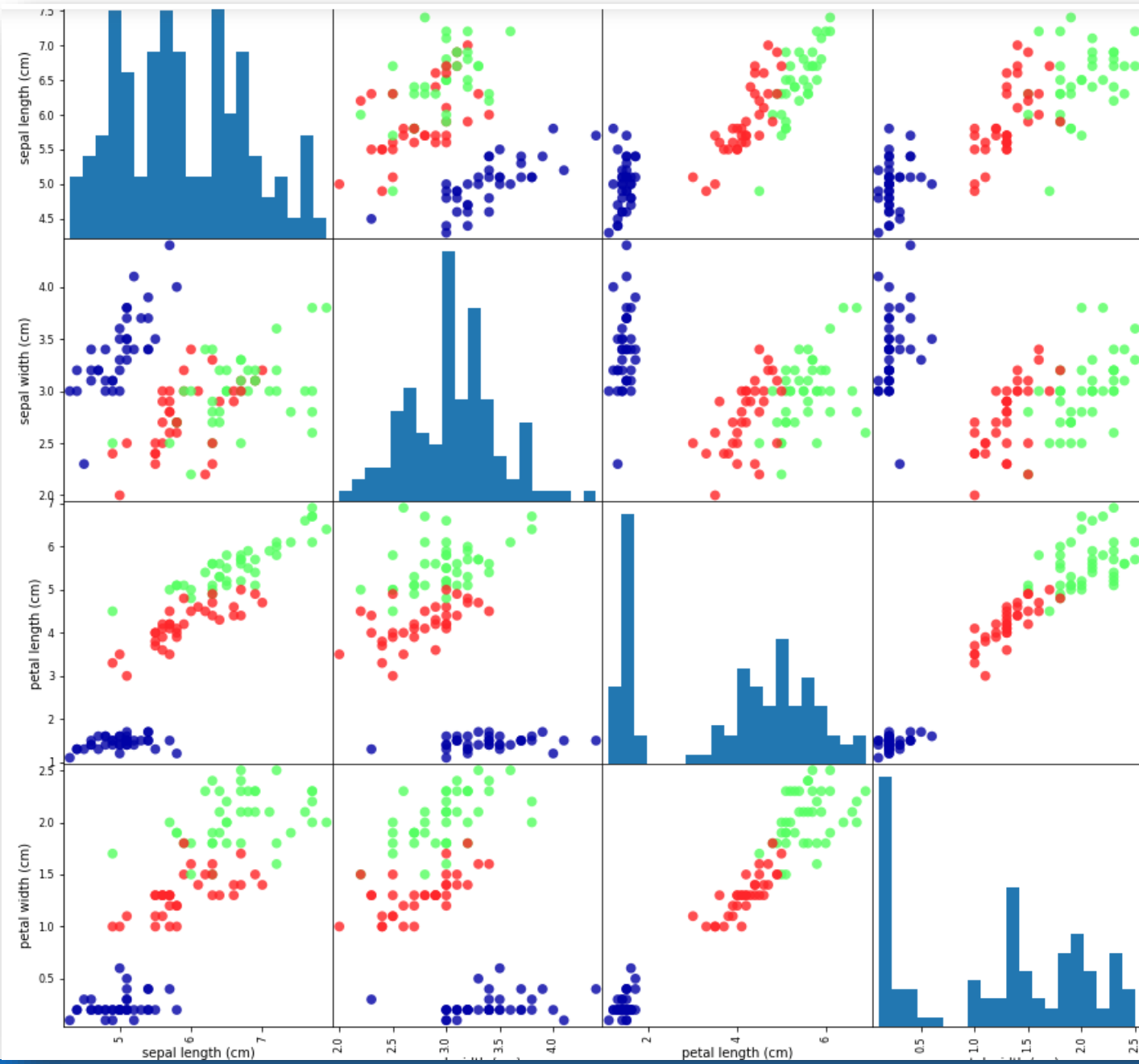
## ■ 데이터 세트 확인하기(상관관계)

- NumPy 배열을 pandas 데이터 프레임으로 변환: 그림을 그리기 위함
- scatter\_matrix( ): pandas를 이용하여 산점도 그리기(히스토그램 포함)

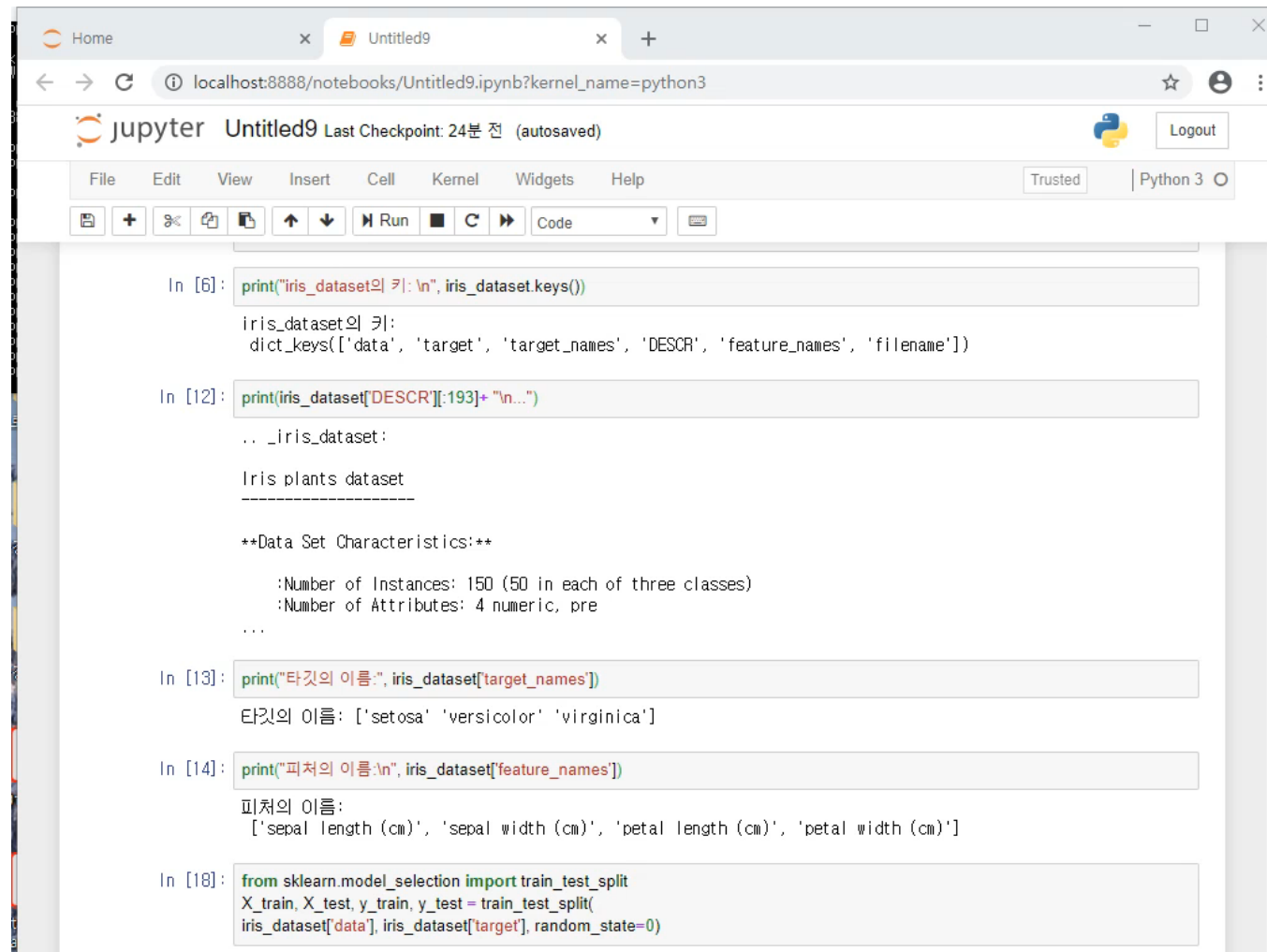
```
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
                           hist_kwds={'bins':20}, s=60, alpha=.8, cmap=mglearn.cm3)
```

alpha: 투명도  
c: 색  
s: 점의 크기  
marker = 'o' : 동그란 점

```
In [20]: # X_train 데이터를 사용해서 데이터 프레임 작성
# 열의 이름은 iris_dataset.feature_names에 있는 문자열 사용
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# 데이터프레임을 사용해 y_train에 따라 색으로 구분된 산점도 작성
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15,15), marker='o',
                           hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
```







The screenshot shows a Jupyter Notebook titled 'Untitled9' running on a local server at localhost:8888. The interface includes a top navigation bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help' menus. Below the menu is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook content consists of several code cells:

```
In [6]: print("iris_dataset의 키: \n", iris_dataset.keys())
iris_dataset의 키:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])

In [12]: print(iris_dataset['DESCR'][:193] + "\n...")
.. _iris_dataset:
Iris plants dataset
-----
**Data Set Characteristics:**

: Number of Instances: 150 (50 in each of three classes)
: Number of Attributes: 4 numeric, pre
...
```

```
In [13]: print("타겟의 이름:", iris_dataset['target_names'])
타겟의 이름: ['setosa' 'versicolor' 'virginica']

In [14]: print("피처의 이름:\n", iris_dataset['feature_names'])
피처의 이름:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

In [18]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

## ■ k-최근접 이웃 알고리즘

- k-최근접 이웃 알고리즘은 neighbors 모듈 아래 KNeighborsClassifier 클래스에 구현되어 있음

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=1)
```

```
In [22]: from sklearn.neighbors import KNeighborsClassifier  
         knn = KNeighborsClassifier(n_neighbors=1)
```

- 예측을 위해 KNeighborsClassification( ) 함수에 매개변수를 1로 지정한 후 knn 객체로 저장

- knn 객체의 fit 메서드를 이용하여 훈련 데이터 세트로 부터 모델을 만듦

```
knn.fit(X_train, y_train)
```

```
In [23]: knn.fit(X_train, y_train)
```

```
Out [23]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                                metric_params=None, n_jobs=None, n_neighbors=1, p=2,  
                                weights='uniform')
```

세부적인 매개변수의 설명은  
다음에 ...

- 예측하기
  - iris 꽃이 어떤 품종에 속할 것인지 예측하기(예시 데이터 사용)

```
X_new = np.array([[5,2.9,1,0]])  
print("X_new.shape", X_new.shape)
```

```
In [24]: X_new = np.array([[5, 2.9, 1, 0]])  
         print("X_new.shape", X_new.shape)  
         X_new.shape (1, 4)
```

사이킷런은 항상 데이터가  
2차원 배열로 예상

```
prediction = knn.predict(X_new)
print("예측", prediction)
print("예측한 타깃의 이름:",
      iris_dataset['target_names'][prediction])
```

```
In [27]: prediction = knn.predict(X_new)
print("예측", prediction)
print("예측한 타깃의 이름:",
      iris_dataset['target_names'][prediction])
```

```
예측 [0]
예측한 타깃의 이름: ['setosa']
```

이 예측을 신뢰할 수 있을까요?

- 모형 평가하기

- 평가용 데이터 세트를 이용하여 iris 품종이 얼마나 정확하게 맞았는지 모델 성능 평가

```
y_pred = knn.predict(X_test)
print("테스트 세트에 대한 예측값:\n", y_pred)
```

```
In [28]: y_pred = knn.predict(X_test)
print("테스트 세트에 대한 예측값:\n", y_pred)
```

테스트 세트에 대한 예측값:

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
2]
```

```
print("테스트 세트의 정확도: {:.2f}".format(np.mean(y_pred == y_test)))  
print("테스트 세트의 정확도: {:.2f}".format(knn.score(X_test, y_test)))
```

```
In [31]: print("테스트 세트의 정확도: {:.2f}".format(np.mean(y_pred == y_test)))
```

테스트 세트의 정확도: 0.97

주의

```
In [32]: print("테스트 세트의 정확도: {:.2f}".format(knn.score(X_test, y_test)))
```

테스트 세트의 정확도: 0.97

```
In [27]: from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)

In [28]: knn.fit(X_train, y_train)

Out[28]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=1, p=2,
weights='uniform')

In [29]: X_new = np.array([[5, 2.9, 1, 0]])
print("X_new.shape", X_new.shape)

X_new.shape (1, 4)

In [30]: prediction = knn.predict(X_new)
print("예측", prediction)
print("예측한 타겟의 이름:",
      iris_dataset['target_names'][prediction])

예측 [0]
예측한 타겟의 이름: ['setosa']

In [31]: y_pred = knn.predict(X_test)
print("테스트 세트에 대한 예측값:\n", y_pred)

테스트 세트에 대한 예측값:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]

In [32]: print("테스트 세트의 정확도: {:.2f}".format(np.mean(y_pred == y_test)))
print("테스트 세트의 정확도: {:.2f}".format(knn.score(X_test, y_test)))
```



# 지도 학습

## ■ 기본 개념

- 분류와 회귀
  - 분류(classification)는 미리 정의 내려진 여러 클래스 레이블(class label) 중 하나를 예측하는 것이다
    - 이진 분류(양성 클래스 or 음성 클래스) 양성 클래스: 학습 대상
    - 다중 분류
  - 회귀(regression)는 실수를 예측하는 것이다
    - 교육수준, 연령, 주거지 등을 근거로 개인의 소득을 예측한다

## ■ 일반화, 과대적합, 과소적합

### ◦ 일반화

- 학습용 데이터를 학습시켜 만든 모델이 새로운 데이터에 대해 정확하게 예측할 수 있다면, 학습용 데이터에서 평가용 데이터로 일반화(generalization) 되었다고 한다
- 머신러닝에서 모델은 정확하게 일반화 되도록 만들어야 한다
- 일반화를 고려하여, 복잡한 모델을 만들면 학습용 데이터에만 정확한 모델이 될 수 있다

### ◦ 과대적합 overfitting

- 주어진 모든 정보를 다 사용해서 모델을 만들어 학습용 데이터에만 정확한 모델이 된 것을 과대적합이라 한다

- 과소적합 underfitting
  - 학습용 데이터의 다양한 점을 발견하지 못하고 학습용 데이터에 잘 맞지 않고, 너무 간단한 모델을 과소적합이라 한다
- 우리가 원하는 모델
  - 과대적합도 아니며 과소적합도 아니고, 일반화 성능이 최대가 되는 최적점에 있는 모델을 말한다
  - 이를 위해서 다양한 데이터의 수집이 중요하다

## ■ 데이터 세트

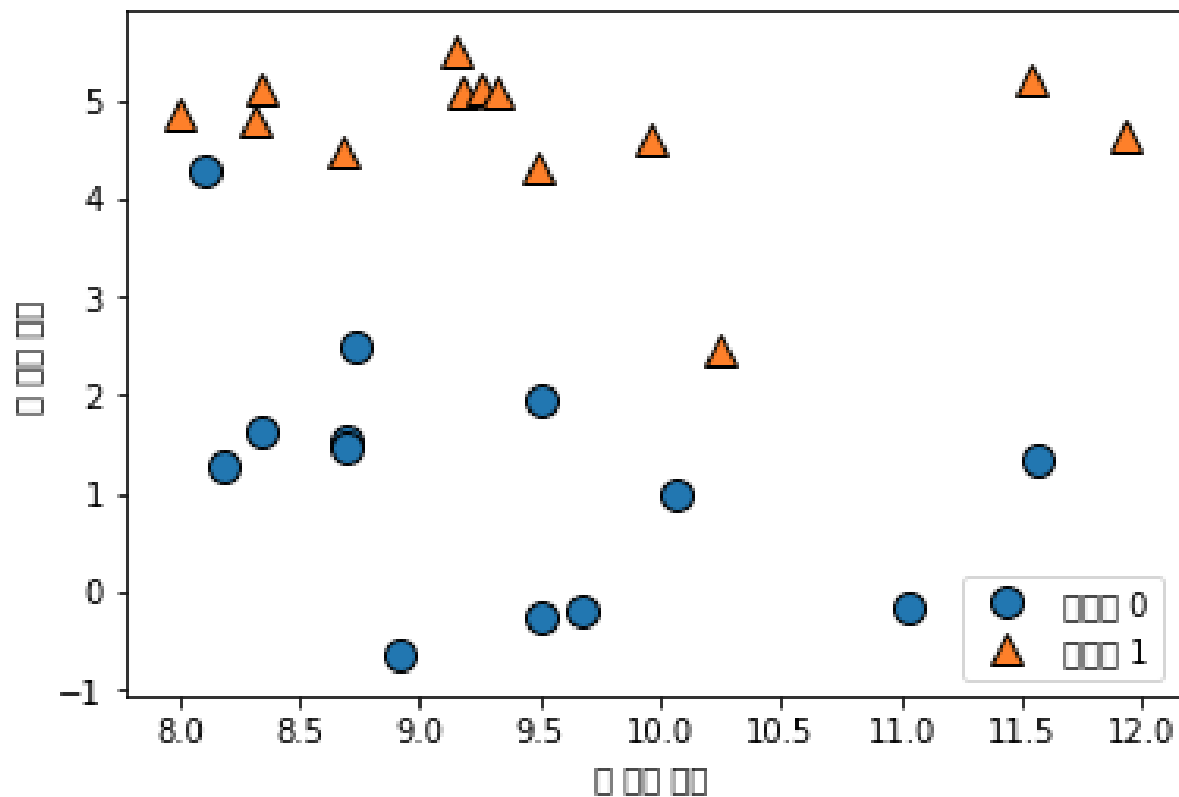
- forge 데이터 세트
  - 데이터 세트 불러오기 및 산점도 그리기

```
X, y = mglearn.datasets.make_forge()  
# 산점도 그리기  
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)  
plt.legend(["클래스 0", "클래스 1"], loc=4)  
plt.xlabel("첫 번째 특성")  
plt.ylabel("두 번째 특성")  
print("X.shape:", X.shape)
```

- 26가지 데이터와 2가지 종류
- warning 발생 및 한글 깨짐 현상

X.shape: (26, 2)

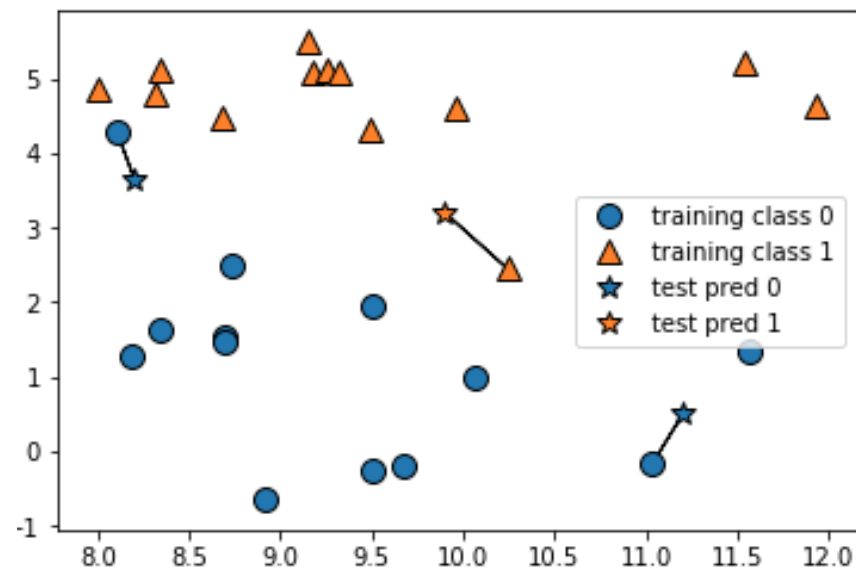
C:\Users\win 2\Anaconda3\lib\site-packages\sklearn\utils\make\_blobs is deprecated; Please import make\_blobs directly from sklearn.datasets  
warnings.warn(msg, category=DeprecationWarning)



## ■ k-최근접 이웃 분류

- 최근접 이웃 분류 알고리즘은 학습용 데이터 중, 분류할 대상과 가장 인접한 데이터 하나를 찾아 예측

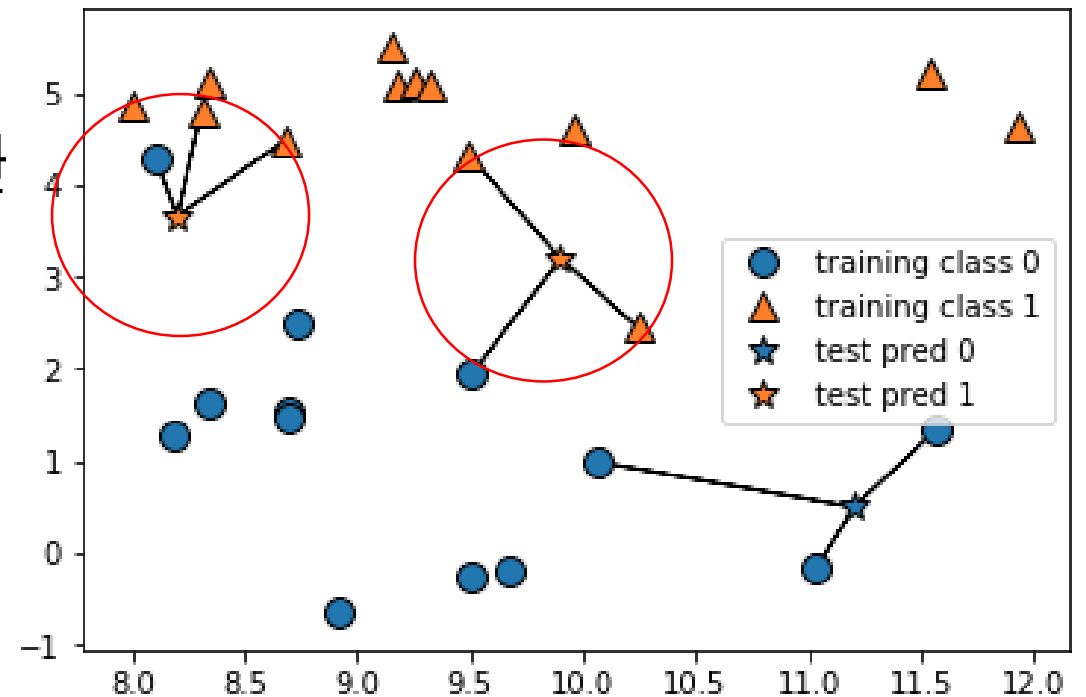
```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```



- k 값을 3으로 변경했을 때 결과

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```

- 평가용 데이터가 인접한 학습용 데이터와 인접한 3개 데이터 중, 2개가 속한 데이터로 분류



## ■ k-NN 분류 과정

- 학습용 데이터세트와 평가용 데이터세트로 구분

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

- KNeighborsClassifier를 import 하고 객체를 만듦(k=3)

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```



- 학습용 데이터세트를 이용하여 분류 모델 학습

```
clf.fit(X_train, y_train)
```

- 평가용 데이터세트에 대해 predict 메서드를 호출하여 예측
  - 학습용 데이터세트에서 가장 가까운 이웃을 계산하여 과반수 클래스에 분류

```
print("평가용 세트 예측:", clf.predict(X_test))
```

```
In [27]: print("평가용 세트 예측:", clf.predict(X_test))
```

```
평가용 세트 예측: [1 0 1 0 1 0 0]
```

- 모델의 일반화를 파악하기 위해 score 메서드에 평가용 데이터세트에 평가용 레이블을 포함시킨다

```
print("평가용 데이터세트 정확도: {:.2f}".format(clf.score(X_test, y_test)))
```

```
In [28]: print("평가용 데이터세트 정확도: {:.2f}".format(clf.score(X_test, y_test)))
```

```
평가용 데이터세트 정확도: 0.86
```

- 모델의 정확도 86%

## ■ KNeighborsClassifier 분석

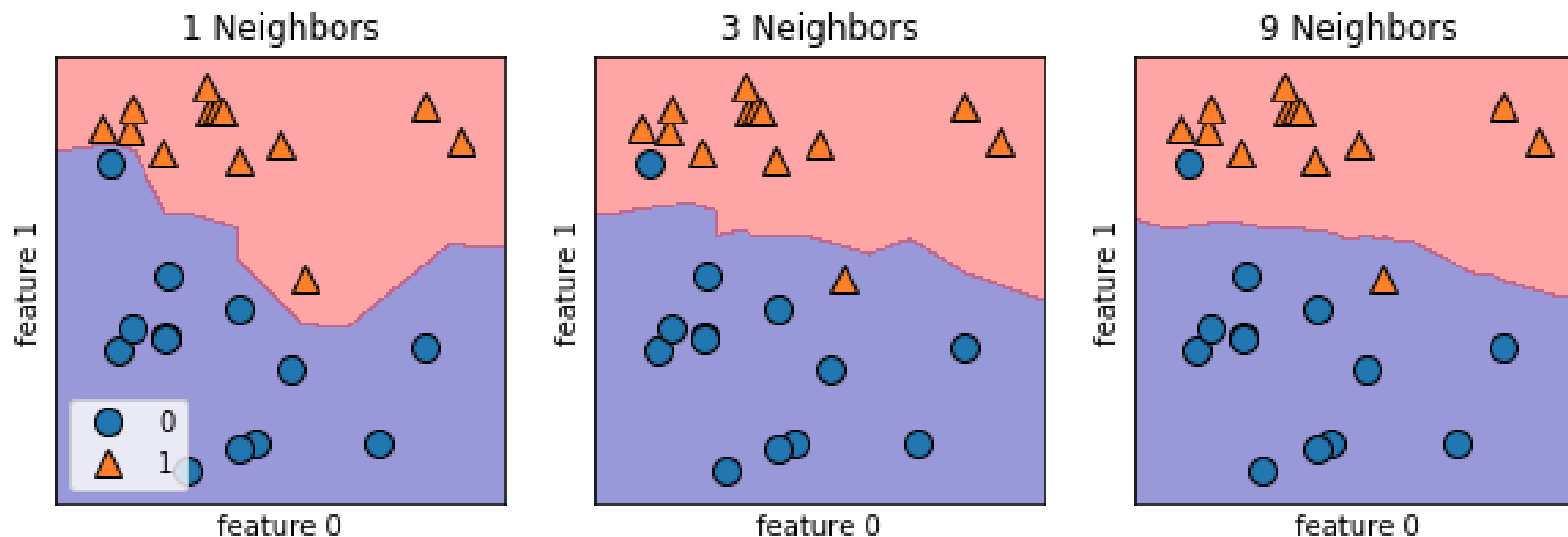
- xy 평면으로 그림을 그려서 분류기 작동을 확인(결정 경계 파악)

```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{ } Neighbors".format(n_neighbors))
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
    axes[0].legend(loc=3)
```

- $k=1$  인 경우, 결정 경계가 학습용 데이터에 인접하게 위치함
- $k=3$  인 경우, 결정 경계가 부드러워짐
- $k$  값이 커질 수록 모델의 예측은 정확하지 않을 수 있다

Out [30]: <matplotlib.legend.Legend at 0x22acd0e4d30>



- 모델 복잡도와 일반화 사이의 관계

```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{ } Neighbors".format(n_neighbors))
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
axes[0].legend(loc=3)
```

# 요약

- 사이킷런의 기본적인 함수
- 데이터 분석에 필요한 기본적인 라이브러리 import
- 데이터 세트 불러오는 방법 및 해석
- 학습용 데이터와 평가용 데이터로 구분
- 데이터 세트의 상관관계
- k-최근접 이웃 알고리즘
  - iris 데이터 세트
  - forge 데이터 세트

## 다음 시간

- k-최근접 이웃 회귀
- 선형 회귀 모델