

빅데이터분석

방향성 데이터 분석 기법 I -  
데이터 전처리와 분류 모형과 성과 척도

# 1/사이킷런(scikit-learn)(1)

- 1) 사이킷런 개요
- 2) 첫 번째 머신러닝 만들어보기
- 3) 사이킷런의 기반 프레임워크
- 4) Model Selection 모듈
- 5) 데이터 전처리

# 1. 사이킷런 개요

---



## 사이킷런이란?

“

파이썬 머신러닝 라이브러리 중  
가장 많이 사용되는 라이브러리

”



# 1. 사이킷런 개요



## 사이킷런이란?

The screenshot shows the scikit-learn homepage. At the top, there's a navigation bar with links: Install, User Guide, API, Examples, and More. Below this, the 'scikit-learn' logo is displayed with the tagline 'Machine Learning in Python'. A 'Getting Started' button is visible. To the right, a list of features is provided:

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

The main content area is divided into several sections, each with a title, description, applications, algorithms, and an example visualization:

- Classification**: Identifying which category an object belongs to. Applications: Spam detection, image recognition. Algorithms: SVM, nearest neighbors, random forest, and more. Example: A 4x4 grid of handwritten digits (MNIST) classified into two classes (red and blue).
- Regression**: Predicting a continuous-valued attribute associated with an object. Applications: Drug response, Stock prices. Algorithms: SVR, nearest neighbors, random forest, and more. Example: A line plot showing a non-linear relationship between two variables, with training and testing data points.
- Clustering**: Automatic grouping of similar objects into sets. Applications: Customer segmentation, Grouping experiment outcomes. Algorithms: k-Means, spectral clustering, mean-shift, and more. Example: A scatter plot showing data points grouped into four clusters (red, yellow, green, and blue).
- Dimensionality reduction**: Reducing the number of random variables to consider. Applications: Visualization, Increased efficiency. Algorithms: k-Means, feature selection, non-negative matrix factorization, and more. Example: A 3D scatter plot showing data points in a high-dimensional space, with axes labeled 'Vernica', 'Versicour', and 'Seton'.
- Model selection**: Comparing, validating and choosing parameters and models. Applications: Improved accuracy via parameter tuning. Algorithms: grid search, cross validation, metrics, and more. Example: A line plot showing the performance of a model as a function of a parameter, with a green line indicating the optimal performance.
- Preprocessing**: Feature extraction and normalization. Applications: Transforming input data such as text for use with machine learning algorithms. Algorithms: preprocessing, feature extraction, and more. Example: A grid of images showing various preprocessing steps, such as normalization and feature extraction.



홈페이지에서 제공되는 기능

- ▶ Classification
- ▶ Regression
- ▶ Clustering
- ▶ Dimensionality reduction
- ▶ Model selection
- ▶ Preprocessing

## 2. 첫 번째 머신러닝 만들어보기

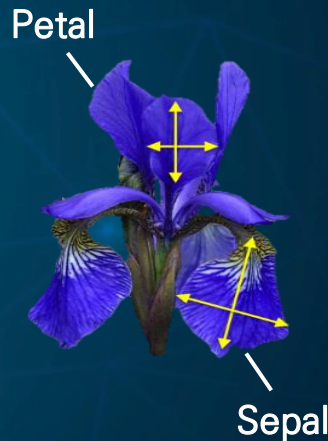
### iris 데이터 집합

→ 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)



## 2. 첫 번째 머신러닝 만들어보기

 필요 모듈 import

```
from sklearn.datasets import load_iris
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split
```

## 2. 첫 번째 머신러닝 만들어보기

### 데이터 가져오기

```
import pandas as pd
```

```
iris = load_iris()
```

```
iris_data = iris.data
```

```
iris_label = iris.target
```

```
print('iris target값:', iris_label)
```

```
print('iris target명:', iris.target_names)
```

#### Bunch

data ← input 값

target ← target variable 값

feature\_names ← input 변수의 이름

target\_names ← target 변수의 이름

DESCR ← dataset에 대한 description

```
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)
```

```
iris_df['label'] = iris.target
```

```
iris_df.head(3)
```

## 2. 첫 번째 머신러닝 만들어보기

### 데이터 가져오기

```
import pandas as pd
```

```
iris = load_iris()
```

```
iris_data = iris.data
```

```
iris_label = iris.target
```

```
print('iris target값:', iris_label)
```

```
print('iris target명:', iris.target_names)
```

```
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)
```

```
iris_df['label'] = iris.target
```

```
iris_df.head(3)
```

	sepal length (cm)	sepal width (cm)	...	petal width (cm)	label
0	5.1	3.5	...	0.2	0
1	4.9	3.0	...	0.2	0
2	4.7	3.2	...	0.2	0

input 변수 4개를 가지는 DataFrame 생성

5번째 column 생성



## 2. 첫 번째 머신러닝 만들어보기

### 훈련용, 테스트용 데이터 분할

#### sklearn.model\_selection 모듈

```
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label,  
                                                    test_size=0.2, random_state=11)
```

이 경우 데이터의 20%를 test로 사용됨  
(80%는 train dataset으로 사용)

값을 지정해 주면 항상 일정하게  
testset이 선정됨

## 2. 첫 번째 머신러닝 만들어보기

### 훈련용, 테스트용 데이터 분할

#### sklearn.model\_selection 모듈

```
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label,  
                                                    test_size=0.2, random_state=11)
```

함수 호출 후, return 값이 4개의 원소를 갖는 튜플로 반환됨

- ▶  $X_{train}$  : train data의 input 변수
- ▶  $y_{train}$  : target의 train 값
- ▶  $X_{test}$  : test data의 input 변수
- ▶  $y_{test}$  : target의 test 값

## 2. 첫 번째 머신러닝 만들어보기

### 모형 객체 생성과 학습, 예측

```
dt_clf = DecisionTreeClassifier(random_state=11)
```

*DecisionTreeClassifier Object  
Instance를 생성*

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

*훈련 data의 input 변수 값과 target 값을 주고  
학습을 시켜 DecisionTree를 생성*

```
pred = dt_clf.predict(X_test)
```

*X\_test에 대한 예측 값이 pred라는 변수에 할당*

- ▶ 다른 Classifier들도 fit와 훈련 data를 줘서 훈련시키고 predict에서 예측하는 동일한 메커니즘을 통해서 예측할 수 있음

## 2. 첫 번째 머신러닝 만들어보기

### 성능 평가

```
from sklearn.metrics import accuracy_score  
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

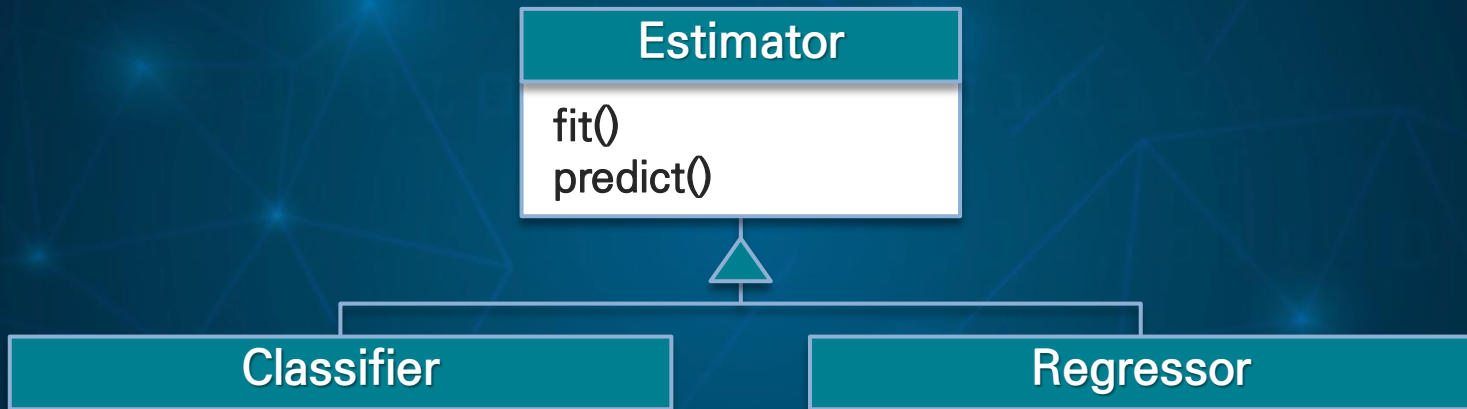
첫 번째 인자는 정답, 두 번째 인자는 *model*의 예측 값을 제시

예측 정확도: 0.9333



### 3. 사이킷런의 기반 프레임워크

 Estimator – 지도 학습의 모든 알고리즘의 부모 클래스



target 변수가 범주형인 경우 사용

**예**

Iris의 종 몇 개 중 하나를 가질 때

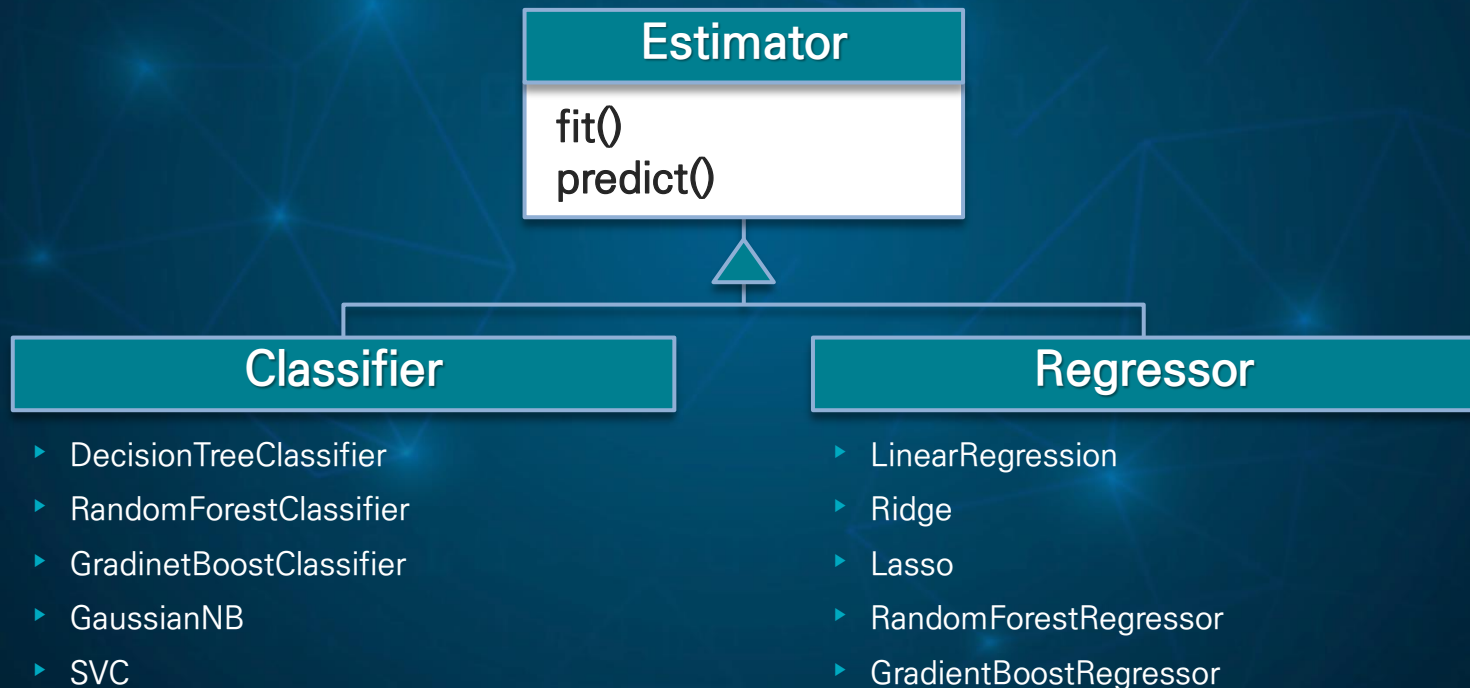
목표변수가 연속형인 경우 사용

**예**

집값을 예측할 때

### 3. 사이킷런의 기반 프레임워크

 Estimator – 지도 학습의 모든 알고리즘의 부모 클래스



### 3. 사이킷런의 기반 프레임워크

#### 분류나 회귀 연습용 예제 데이터

`datasets.load_boston()` : *boston* 집값을 예측

`datasets.load_breast_cancer()` : *유방암에 대한 여부를 예측*


`datasets.load_diabetes()` : *당뇨병에 대한 예측*

`datasets.load_digits()`

`datasets.load_iris()`

### 3. 사이킷런의 기반 프레임워크

#### fetch 계열 명령

 패키지에 처음부터 저장되어 있지 않고 처음 호출 시 인터넷에서 다운로드  
(최초 사용 시 인터넷 연결 필요)

- ▶ `fetch_covtype()`: 회귀분석용 토지 조사
- ▶ `fetch_20newsgroup()`: 뉴스 그룹 텍스트 데
- ▶ `fetch_olivetti_faces()`: 얼굴 이미지
- ▶ `fetch_lfw_people()`: 얼굴 이미지
- ▶ `fetch_lfw_pairs()`: 얼굴 이미지
- ▶ `fetch_rvc1()`: 로이터 뉴스 말뭉치
- ▶ `fetch_mldata()`: ML 웹사이트에서 다운로드



### 3. 사이킷런의 기반 프레임워크

#### 내장 데이터 집합

“ dataset은 Bunch Class 형태로 되어 있어서  
딕셔너리 형태로 접근할 수가 있음 ”

### 3. 사이킷런의 기반 프레임워크



#### Bunch 객체

```
from sklearn.datasets import load_iris

iris_data = load_iris()
print(type(iris_data))

keys = iris_data.keys()
print('붓꽃 데이터 세트의 키들:', keys)
```

<class 'sklearn.utils.Bunch'>

붓꽃 데이터 세트의 키들: dict\_keys(['data', 'target', 'frame',  
'target\_names', 'DESCR', 'feature\_names', 'filename'])

### 3. 사이킷런의 기반 프레임워크



#### Bunch 객체

```
print('\n feature_names 의 type:', type(iris_data.feature_names))  
print(' feature_names 의 shape:', len(iris_data.feature_names))  
print(iris_data.feature_names)
```

feature\_names 의 type: `<class 'list'>`

feature\_names 의 shape: 4

`['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']`

### 3. 사이킷런의 기반 프레임워크



#### Bunch 객체

```
print('\n target_names 의 type:', type(iris_data.target_names))  
print(' feature_names 의 shape:', len(iris_data.target_names))  
print(iris_data.target_names)
```

target\_names 의 type: <class 'numpy.ndarray'>

feature\_names 의 shape: 3

['setosa' 'versicolor' 'virginica']



### 3. 사이킷런의 기반 프레임워크



#### Bunch 객체

```
print('\n data 의 type:',type(iris_data.data))  
print(' data 의 shape:',iris_data.data.shape)  
print(iris_data['data'])
```

data 의 type: `<class 'numpy.ndarray'>`

data 의 shape: `(150, 4)`

`[[5.1 3.5 1.4 0.2]`

`[4.9 3. 1.4 0.2]`

`.....`

150개의 Row와 4개의 column으로 이루어져 있다는 뜻

[illegible]

## 4. Model Selection 모듈

### train\_test\_split() - 훈련/테스트 데이터 세트 분리

잘못된 예

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
iris = load_iris()
dt_clf = DecisionTreeClassifier()
train_data = iris.data
train_label = iris.target
dt_clf.fit(train_data, train_label)
```

```
# 학습 데이터 셋으로 예측 수행
pred = dt_clf.predict(train_data)
print('예측 정확도:', accuracy_score(train_label, pred))
```

예측 정확도: 1.0

## 4. Model Selection 모듈

### train\_test\_split() - 훈련/테스트 데이터 세트 분리

잘못된 예

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
iris = load_iris()
dt_clf = DecisionTreeClassifier()
train_data = iris.data
train_label = iris.target
dt_clf.fit(train_data, train_label)
```

```
# 학습 데이터 셋으로 예측 수행
pred = dt_clf.predict(train_data)
print('예측 정확도: ', accuracy_score(train_label, pred))
```

Model을 만들 때 쓴 data와 Model의 성능을 평가하기 위해서는 쓴 data가 같음

갖고 있는 data 중 일부를 가지고 훈련 및 테스트를 해야 함

예측 정확도: 1.0



## 4. Model Selection 모듈

### train\_test\_split() 주요 인자

test\_size

- ▶ 디폴트는 0.25

shuffle

- ▶ 디폴트는 True

random\_state

- ▶ 지정하지 않으면 수행할 때마다 다른 학습/테스트 데이터를 생성

## 4. Model Selection 모듈

### train\_test\_split()의 반환값은 튜플 형태

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
dt_clf = DecisionTreeClassifier( )
iris_data = load_iris()
```

```
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target,
                                                    test_size=0.3, random_state=121)
```

```
dt_clf.fit(X_train, y_train)
pred = dt_clf.predict(X_test)
print('예 측 정 확 도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

예 측 정 확 도: 0.9556



# 정리하기

---

- 첫번째 머신러닝 만들기
- 사이킷런의 기반 프레임워크
- Model Selection 모듈

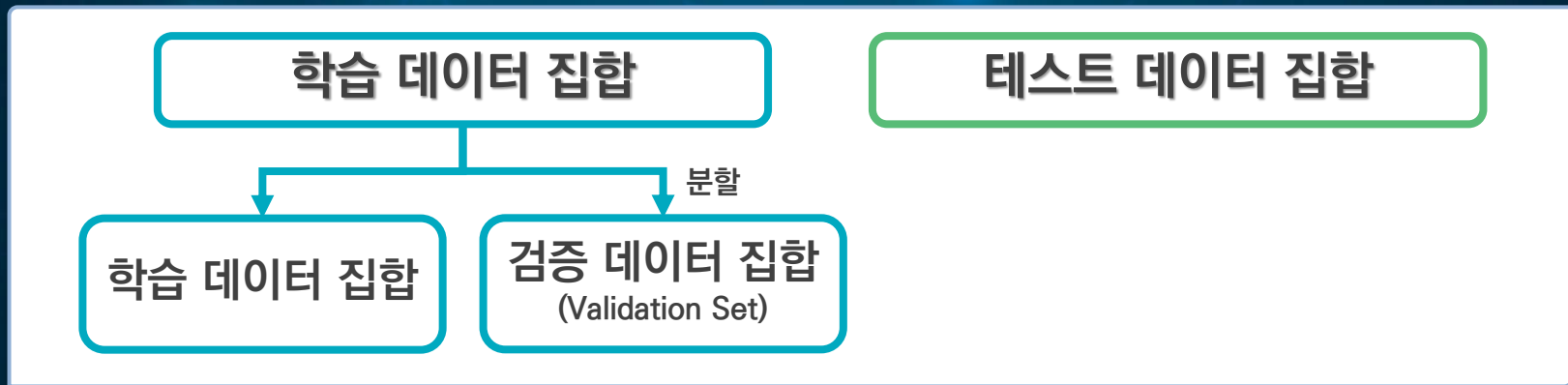
## 2/사이킷런(scikit-learn)(2)

- 1) 사이킷런 개요
- 2) 첫 번째 머신러닝 만들어보기
- 3) 사이킷런의 기반 프레임워크
- 4) Model Selection 모듈

# 1. Model Selection 모듈



모델을 만들 때 데이터를 **학습 데이터**와 **테스트 데이터**로 나눔





- ▶ 테스트를 할 때 훈련데이터를 쓰면 성능이 높게 나오기 때문에 데이터를 나눔
- ▶ 많은 기계학습 모델들이 검증 데이터 집합을 활용한 하이퍼파라미터 튜닝을 통하여 모델을 최적화함



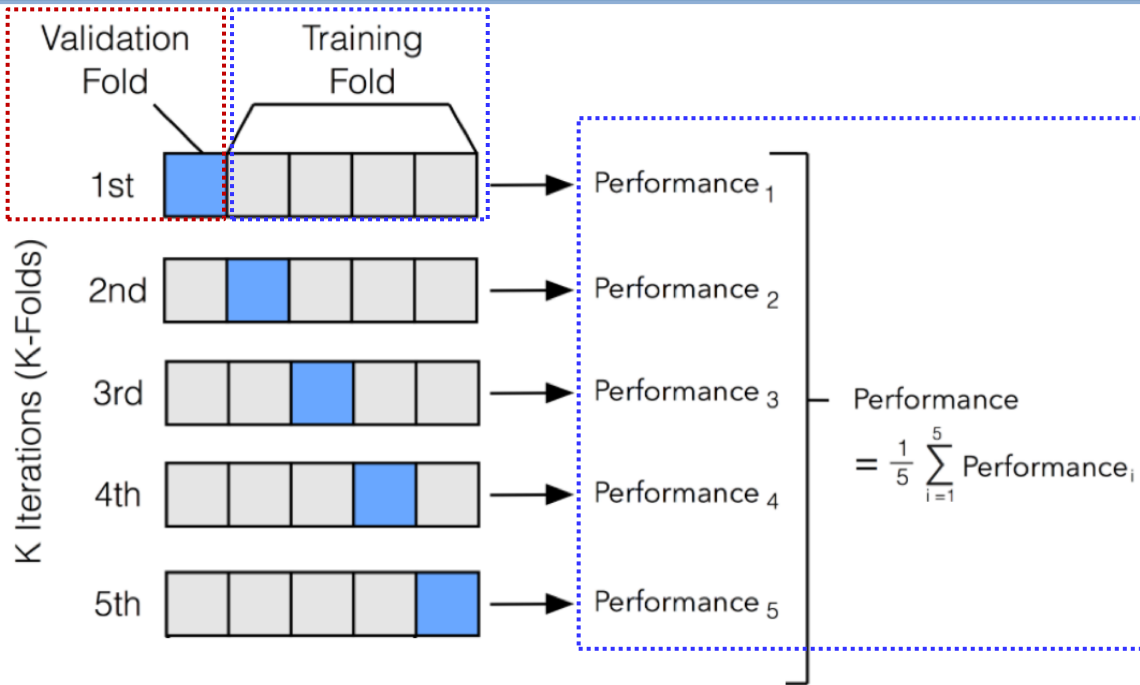
# 1. Model Selection 모듈

## K 폴드 교차 검증

-  검증 데이터 집합으로 하이퍼파라미터를 조정하기 위해서 많이 사용할 경우
-  데이터셋이 충분하지 않은 경우

# 1. Model Selection 모듈

## K 폴드 교차 검증



# 1. Model Selection 모듈

## K 폴드 교차 검증

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
import numpy as np

iris = load_iris()
features = iris.data
label = iris.target
dt_clf = DecisionTreeClassifier(random_state=156)
```

```
kfold = KFold(n_splits=5)
cv_accuracy = []
print('붓꽃 데이터 세트 크기:', features.shape[0])
```

5개의 세트로 분할하여 K 폴드 교차  
검증을 하겠다는 뜻

추후 5번의 성능평가 결과를 리스트에  
추가해서 저장

# 1. Model Selection 모듈



## K 폴드 교차 검증

```
n_iter = 0
```

*train\_index, test\_index가 반환됨*

```
for train_index, test_index in kfold.split(features):  
    X_train, X_test = features[train_index], features[test_index]  
    y_train, y_test = label[train_index], label[test_index]  
    #학습 및 예측  
    dt_clf.fit(X_train, y_train)  
    pred = dt_clf.predict(X_test)  
    n_iter += 1  
    accuracy = np.round(accuracy_score(y_test, pred), 4)  
    train_size = X_train.shape[0]  
    test_size = X_test.shape[0]  
    print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'  
        .format(n_iter, accuracy, train_size, test_size))  
    print('#{0} 검증 세트 인덱스:{1}'.format(n_iter, test_index))  
    cv_accuracy.append(accuracy)
```

# 1. Model Selection 모듈



## K 폴드 교차 검증

```
n_iter = 0
```

```
for train_index, test_index in kfold.split(features):
```

```
    X_train, X_test = features[train_index], features[test_index]
```

```
    y_train, y_test = label[train_index], label[test_index]
```

```
    #학습 및 예측
```

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

```
    pred = dt_clf.predict(X_test)
```

```
    n_iter += 1
```

```
    accuracy = np.round(accuracy_score(y_test, pred), 4)
```

```
    train_size = X_train.shape[0]
```

```
    test_size = X_test.shape[0]
```

```
    print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'
```

```
        .format(n_iter, accuracy, train_size, test_size))
```

```
    print('#{0} 검증 세트 인덱스:{1}'.format(n_iter, test_index))
```

```
    cv_accuracy.append(accuracy)
```

네 번째 자리에서 반올림 하라는 뜻

5번 반복



# 1. Model Selection 모듈

## K 폴드 교차 검증

*iris data는 레코드가 150개(30개씩 5폴드)*

```
n_iter = 0
```

```
for train_index, test_index in
```

```
    X_train, X_test =
```

```
    y_train, y_test =
```

```
    #학습 및 예측
```

```
    dt_clf.fit(X_train,
```

```
    pred = dt_clf.pre
```

```
    n_iter += 1
```

```
    accuracy = np.ro
```

```
    train_size = X_tr
```

```
    test_size = X_test.shape[0]
```

```
    print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'
```

```
        .format(n_iter, accuracy, train_size, test_size))
```

```
    print('#{0} 검증 세트 인덱스:{1}'.format(n_iter, test_index))
```

```
    cv_accuracy.append(accuracy)
```

#1 교차 검증 정확도 :1.0, 학습 데이터 크기: 120, 검증 데이터 크기: 30

#1 검증 세트 인덱스:[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
20 21 22 23 24 25 26 27 28 29]

#2 교차 검증 정확도 :0.9667, 학습 데이터 크기: 120, 검증 데이터 크기: 30

#2 검증 세트 인덱스:[30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47  
48 49 50 51 52 53 54 55 56 57 58 59]

...

# 1. Model Selection 모듈



## K 폴드 교차 검증

```
# 개별 iteration별 정확도를 합하여 평균 정확도 계산  
print('\n## 평균 검증 정확도:', np.mean(cv_accuracy))
```

```
## 평균 검증 정확도: 0.9
```

# 1. Model Selection 모듈

## Stratified K-폴드 교차 검증



저장되어 있는 데이터가 클래스별로 차례대로 저장된 경우 문제가 생길 수 있음

예

iris 데이터의 경우

- ▶ 3가지 종류 중 맨 앞에 있는 setosa 50개가 먼저 오고, 차례대로 50개, 50개, 50개가 와 있음
- ▶ 만약 교차 검증을 위해서 50개를 앞에 것만 썼다면, 한 종이 한꺼번에 Validation Set으로 들어오는 문제가 생기게 됨
- ▶ 이런 경우에 쓰는 것이 Stratified K 폴드 교차 검증임

# 1. Model Selection 모듈

## Stratified K-폴드 교차 검증

### 층화 K-폴드

```
import pandas as pd

iris = load_iris()

iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['label']=iris.target
iris_df['label'].value_counts()
```

```
2    50
1    50
0    50
Name: label, dtype: int64
```

# 1. Model Selection 모듈

## Stratified K-폴드 교차 검증

```
kfold = KFold(n_splits=3)
```

3 fold 교차 검증을 하겠다는 뜻

```
n_iter = 0
```

```
for train_index, test_index in kfold.split(iris_df):
```

```
    n_iter += 1
```

```
    label_train= iris_df['label'].iloc[train_index]
```

```
    label_test= iris_df['label'].iloc[test_index]
```

```
    print('## 교차 검증: {0}'.format(n_iter))
```

```
    print('학습 레이블 데이터 분포:\n', label_train.value_counts())
```

```
    print('검증 레이블 데이터 분포:\n', label_test.value_counts())
```

종별로 몇 개씩 있는지 출력



# 1. Model Selection 모듈

## Stratified K-폴드 교차 검증

```
kfold = KFold(n_splits=3)

n_iter = 0
for train_index, test_index in kfold.split(iris_df):
    n_iter += 1
    label_train = iris_df['label'].iloc[train_index]
    label_test = iris_df['label'].iloc[test_index]
    print('## 교차 검증: {0}'.format(n_iter))
    print('학습 레이블 데이터 분포:\n', label_train.value_counts())
    print('검증 레이블 데이터 분포:\n', label_test.value_counts())
```

## 교차 검증: 1

학습 레이블 데이터 분포:

2 50

1 50

Name: label, dtype: int64

검증 레이블 데이터 분포:

0 50

Name: label, dtype: int64

....

- ▶ 학습할 때는 label 1 2인 것만 가지고 하고, 테스트할 때는 label 0인 것만 가지고 해서 제대로 모델이 만들어지지 않으므로 **총화 K 폴드를 사용하는 것이 좋음**

# 1. Model Selection 모듈

## Stratified K-폴드 교차 검증

```
dt_clf = DecisionTreeClassifier(random_state=156)

skfold = StratifiedKFold(n_splits=3)
n_iter=0
cv_accuracy=[]

for train_index, test_index in skfold.split(features, label):
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]
    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_test)
```

# 1. Model Selection 모듈

## Stratified K-폴드 교차 검증

```
n_iter += 1
accuracy = np.round(accuracy_score(y_test, pred), 4)
train_size = X_train.shape[0]
test_size = X_test.shape[0]
print('\n#{0} 교차 검증 정확도: {1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'
      .format(n_iter, accuracy, train_size, test_size))
print('#{0} 검증 세트 인덱스: {1}'.format(n_iter, test_index))
cv_accuracy.append(accuracy)

print('\n## 교차 검증별 정확도:', np.round(cv_accuracy, 4))
print('## 평균 검증 정확도:', np.mean(cv_accuracy))
```

```
## 교차 검증별 정확도: [0.98 0.94 0.98]
## 평균 검증 정확도:
0.9666666666666667
```

# 1. Model Selection 모듈

 `cross_val_score()`



교차 검증을 보다 간편하게



폴드 집합 설정



for 루프를 통한 반복 추출과 학습, 정확도 예측



정확도 평균



# 1. Model Selection 모듈



## cross\_val\_score()

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
```

```
iris_data = load_iris()
dt_clf = DecisionTreeClassifier(random_state=156)
```

```
data = iris_data.data
label = iris_data.target
```

```
scores = cross_val_score(dt_clf, data, label, scoring='accuracy', cv=3)
print('교차 검증별 정확도:', np.round(scores, 4))
print('평균 검증 정확도:', np.round(np.mean(scores), 4))
```

교차 검증별 정확도: [0.98 0.94 0.98]  
평균 검증 정확도: 0.9667

폴드 수

\* 분류의 경우 Straified K-폴드 방식으로 분할

# 정리하기

---

- Model Selection 모듈

# 3/사이킷런(scikit-learn)(3)

## 1) 데이터 전처리

# 1. 데이터 전처리

---

 GIGO: Garbage-in garbage-out

“

데이터 전처리의 중요성

”

데이터  
인코딩

변수  
스케일링

# 1. 데이터 전처리



## 데이터 인코딩



문자형 변수를 **숫자**로 바꿔주는 작업



레이블 인코딩(Label encoding)




원-핫 인코딩(One Hot encoding)



# 1. 데이터 전처리

## 레이블 인코딩(Label encoding)

 범주가 5개가 있다면 5개에 대해서 **일련번호**를 부여하는 것

- ▶ 0 1 2 3 4 문자열에 대해서 하나의 숫자를 대응시키는 방법

# 1. 데이터 전처리

## 레이블 인코딩(Label encoding)

```
from sklearn.preprocessing import LabelEncoder
```

```
items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']
```

```
encoder = LabelEncoder()
```

```
encoder.fit(items)
```

Encoding 할 대상에 class가 몇 개인지 확인

```
labels = encoder.transform(items)
```

실제 인코딩 작업 수행

```
print('인코딩 변환값:',labels)
```

인코딩 변환값: [0 1 4 5 3 3 2 2]

```
print('인코딩 클래스:',encoder.classes_)
```

인코딩 클래스: ['TV' '냉장고' '믹서' '선풍기' '전자렌지' '컴퓨터']

# 1. 데이터 전처리

## 레이블 인코딩(Label encoding)

```
print('디코딩 원본 값:',encoder.inverse_transform([4, 5, 2, 0, 1, 1, 3, 3]))
```

디코딩 원본 값: ['전자렌지' '컴퓨터' '믹서' 'TV' '냉장고' '냉장고' '선풍기' '선풍기']

# 1. 데이터 전처리

## 원-핫 인코딩(One Hot encoding)

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50



3개의 컬럼이 만들어짐

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

# 1. 데이터 전처리

## 원-핫 인코딩(One Hot encoding)

### 사이킷런의 OneHotEncoder 사용



OneHotEncoder로 변환하기 전에 모든 문자열 값을 숫자형 값으로 변환되어야 함



입력 값으로 2차원 데이터가 필요함



# 1. 데이터 전처리

## 원-핫 인코딩(One Hot encoding)

```
from sklearn.preprocessing import OneHotEncoder  
import numpy as np
```

```
items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']
```

# 먼저 **숫자값으로 변환**을 위해 LabelEncoder로 변환합니다.

```
encoder = LabelEncoder()  
encoder.fit(items)  
labels = encoder.transform(items)
```

# **2차원 데이터로 변환**합니다.

```
labels = labels.reshape(-1,1)
```

	0
0	0
1	1
2	4
3	5
4	3
5	3
6	2
7	2

# 1. 데이터 전처리

## 원-핫 인코딩(One Hot encoding)

```
items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']
```

# 원-핫 인코딩을 적용합니다.

```
oh_encoder = OneHotEncoder()  
oh_encoder.fit(labels)  
oh_labels = oh_encoder.transform(labels)  
print('원-핫 인코딩 데이터')  
print(oh_labels.toarray())  
print('원-핫 인코딩 데이터 차원')  
print(oh_labels.shape)
```

(8, 6)

```
[[1. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 1. 0.]  
 [0. 0. 0. 0. 0. 1.]  
 [0. 0. 0. 1. 0. 0.]  
 [0. 0. 0. 1. 0. 0.]  
 [0. 0. 1. 0. 0. 0.]  
 [0. 0. 1. 0. 0. 0.]]
```

# 1. 데이터 전처리

## 원-핫 인코딩(One Hot encoding)

### Pandas에서 One Hot Encoding 하는 예

```
import pandas as pd

df = pd.DataFrame({'item': ['TV', '냉장고', '전자렌지', '컴퓨터',  
                           '선풍기', '선풍기', '믹서', '믹서']})
one_hot_result = pd.get_dummies(df)
```

one_hot_result - DataFrame						
Index	item_TV	item_냉장고	item_믹서	item_선풍기	item_전자렌지	item_컴퓨터
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	1	0	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0
7	0	0	1	0	0	0

- ▶ Pandas에서는 get\_dummies를 하면 쉽게 One Hot Encoding을 할 수 있음

# 1. 데이터 전처리



## 변수 스케일링

### 변수 스케일링(Feature scaling)

“

서로 다른 변수의 값 범위를 일정하게 조정하는 방법

”

# 1. 데이터 전처리



## 변수 스케일링

### StandardScaler

$$z = \frac{x_i - \mu}{\sigma}$$

### MinMaxScaler

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- ▶ min을 갖는 경우 0이 되고, max를 갖는 경우 1이 됨
- ▶ 즉, 0과 1 사이로 값을 바꿔주는 방법



# 1. 데이터 전처리



## StandardScaler

```
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
iris_data = iris.data
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)

print('feature 들의 평균 값')
print(iris_df.mean())
print('\nfeature 들의 분산 값')
print(iris_df.var())
```

# 1. 데이터 전처리



## StandardScaler

### feature 들의 평균 값

```
sepal length (cm)  5.843333  
sepal width (cm)   3.057333  
petal length (cm)  3.758000  
petal width (cm)   1.199333  
dtype: float64
```

### feature 들의 분산 값

```
sepal length (cm)  0.685694  
sepal width (cm)   0.189979  
petal length (cm)  3.116278  
petal width (cm)   0.581006  
dtype: float64
```

# 1. 데이터 전처리



## StandardScaler

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

```
scaler = StandardScaler()
```

```
scaler.fit(iris_df)
```

*Standard 스케일링을 하려면, 평균과 표준편차를 구해야 함*

```
iris_scaled = scaler.transform(iris_df)
```

```
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
```

```
print('feature 들의 평균 값')
```

```
print(iris_df_scaled.mean())
```

```
print('\nfeature 들의 분산 값')
```

```
print(iris_df_scaled.var())
```

# 1. 데이터 전처리



## StandardScaler

'10'을 의미

feature 들의 평균 값

```
sepal length (cm) -1.690315e-15  
sepal width (cm) -1.842970e-15  
petal length (cm) -1.698641e-15  
petal width (cm) -1.409243e-15  
dtype: float64
```

- ▶ 4개의 scaled된 값의 평균을 구해 보면 0에 가까운 값이 됨

feature 들의 분산 값

```
sepal length (cm) 1.006711  
sepal width (cm) 1.006711  
petal length (cm) 1.006711  
petal width (cm) 1.006711  
dtype: float64
```

- ▶ 4개의 scaled된 값의 평균을 구해 보면 1에 가까운 값이 됨



# 1. 데이터 전처리



## MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature들의 최소 값')
print(iris_df_scaled.min())
print('\nfeature들의 최대 값')
print(iris_df_scaled.max())
```



# 1. 데이터 전처리



## MinMaxScaler

**feature들의 최소 값**

sepal length (cm) 0.0

sepal width (cm) 0.0

petal length (cm) 0.0

petal width (cm) 0.0

dtype: float64

**feature들의 최대 값**

sepal length (cm) 1.0

sepal width (cm) 1.0

petal length (cm) 1.0

petal width (cm) 1.0

dtype: float64

# 정리하기

---

- 데이터 인코딩
- 변수 스케일링