

# 2강: 머신러닝 시작하기

인공지능 일반강좌 : 기계학습의 이해(L2-1)

## 강의 교재 사이트

- <https://github.com/hongsukyi/Lv2-1-MachineLearning>

# Contents

사이킷런 소개와 특징

사이킷런 프레임워크 배우기

모델 선택 모듈 소개

데이터 전처리

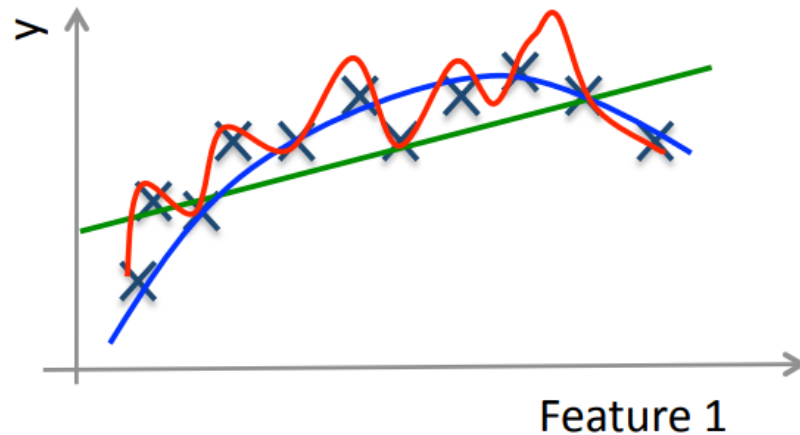
사이킷런으로 타이타닉 생존자 예측

실습 및 숙제

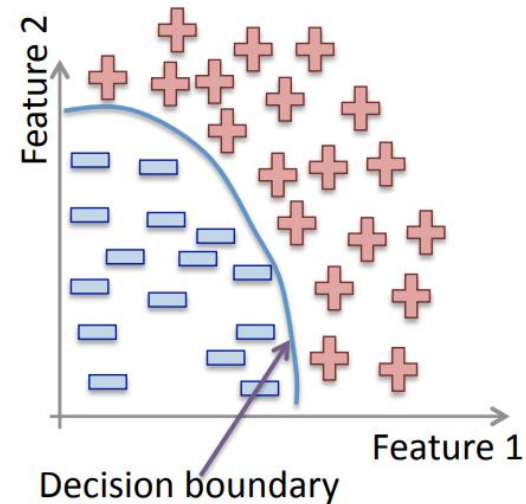
# 지도학습(Supervised Learning)

- 지도학습 알고리즘
- Support Vector Machines, neural networks,
- decision trees, K-nearest neighbors, naive Bayes
- 레이블이 있어야 함. 누가 레이블을 만드나?

회귀(Regression)



분류



# 사이킷런 소개와 특징

- 특징

- ✓ 파이썬 머신러닝 중에서 가장 많이 사용되는 라이브러리
- ✓ 예측 데이터 분석을 위해 간단하고 효과적인 툴 제공
- ✓ Numpy, SciPy, Matplotlib을 기반으로 구성
- ✓ 공개 소프트웨어, 상업용을 사용 불가 – BSD 라이선스
- ✓ 최근 텐서플로우, 케라스, 파이토치 등 딥러닝 전문 라이브러리와 경쟁 중

- 설치

- ✓ 아나콘다를 설치하면 기본적으로 사이킷런까지 설치가 완료 됨
- ✓ 가능하면 conda로 설치를 권장
  - `$ conda install scikit-learn`
- ✓ 버전을 확인
  - `$ print(sklearn.__version__)`

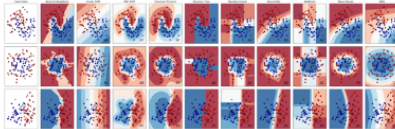
# 사이킷런(scikit-learn) 활용 예제

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...

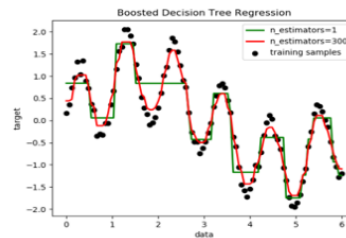


## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...



## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...

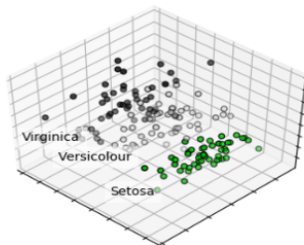


## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** k-Means, feature selection, non-negative matrix factorization, and more...

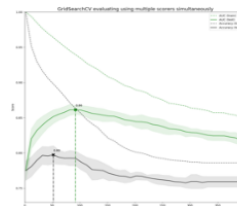


## Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning

**Algorithms:** grid search, cross validation, metrics, and more...

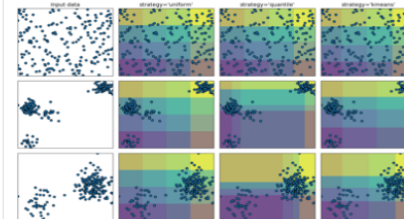


## Preprocessing

Feature extraction and normalization.


**Applications:** Transforming input data such as text for use with machine learning algorithms.

**Algorithms:** preprocessing, feature extraction, and more...



# 사이킷런(scikit-learn) 활용 예제

[https://scikit-learn.org/stable/auto\\_examples/index.html#classification](https://scikit-learn.org/stable/auto_examples/index.html#classification)

[Install](#) [User Guide](#) [API](#) [Examples](#) [More ▾](#)

[Prev](#) [Up](#) [Next](#)

**scikit-learn 0.22.1**  
[Other versions](#)


Please [cite us](#) if you use the software.

**Examples**  
Miscellaneous examples  
Biclustering  
Calibration  
Classification  
Clustering  
Covariance estimation  
Cross decomposition  
Dataset examples  
Decision Trees  
Decomposition

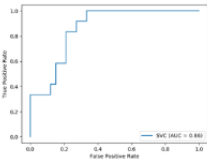
## Examples

### Miscellaneous examples

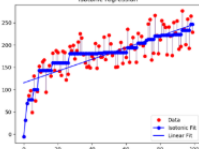
Miscellaneous and introductory examples for scikit-learn.



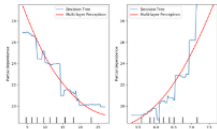
Compact estimator representations



ROC Curve with Visualization API



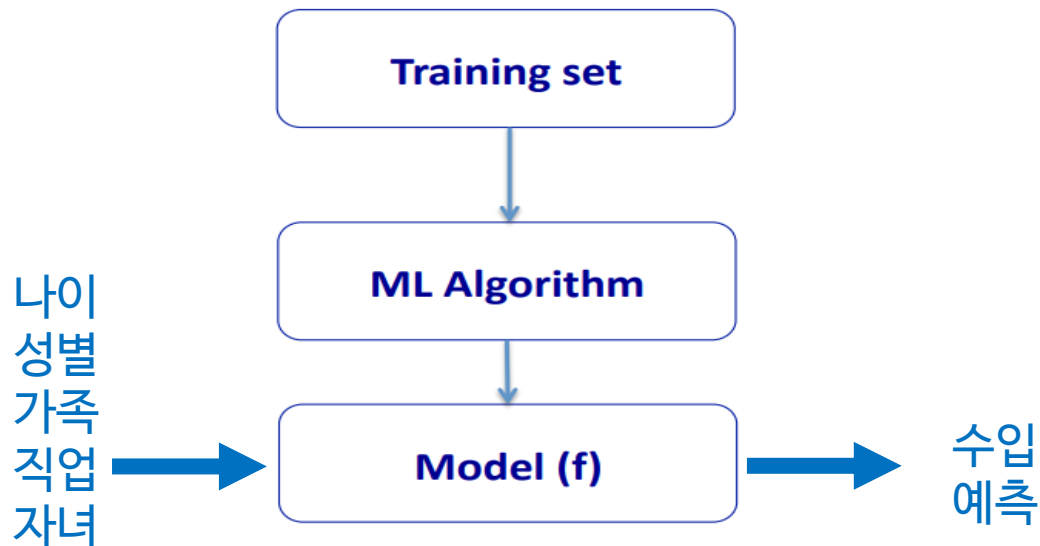
Isotonic Regression



Advanced Plotting With Partial Dependence

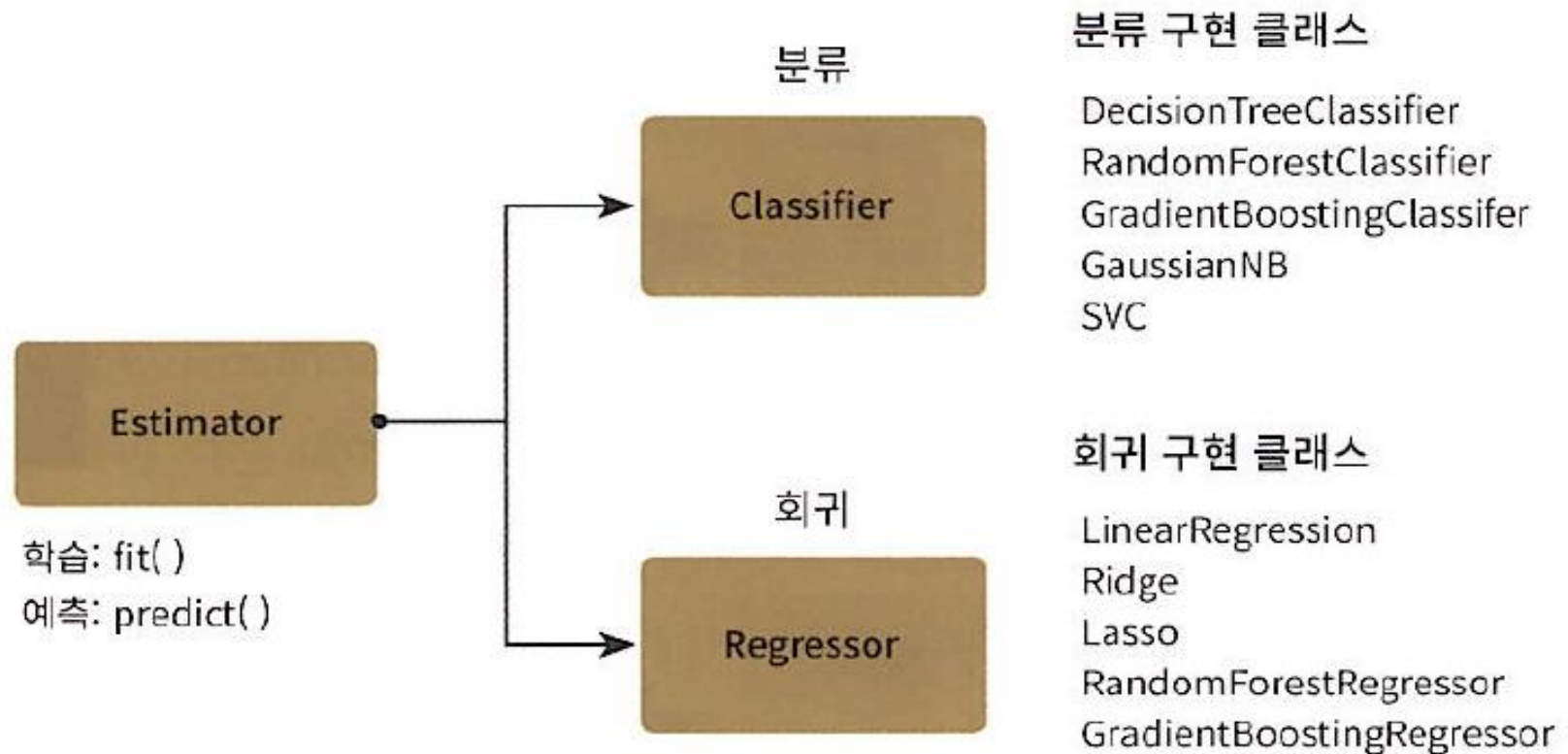
# 머신러닝 모델 구축 프로세스

- 사이킷런 제공해야 하는 모듈을 생각해보면
  - ✓ 피처 처리 (feature processing): 피처의 가공, 변경, 추출
  - ✓ 머신러닝 학습/테스트/예측 수행
  - ✓ 모델 평가





# 사이킷런 Estimator 구분



<그림: 파이썬 머신러닝 완벽가이드, 위키북스, 2019>

# Estimator 이해 및 fit(), predict()

- 지도학습의 분류와 회귀
  - ✓ 학습은 fit()으로 하고
  - ✓ 예측은 predict()를 이용
- Estimator란?
  - ✓ regressor + classifier 클래스로 지칭
- 비지도학습에서는
  - ✓ 차원축소, 클러스터링, 피처추출
  - ✓ fit()와 transform()을 적용
    - 여기서 fit()은 지도학습의 fit()과 다름.
    - 입력데이터 형태에 맞춰 데이터를 변환하기 위한 사전 구조를 맞추는 작업
    - 실제 fit()로 사전 구조를 맞추면 이후 입력데이터의 차원변환 등 transform()로 처리
  - ✓ fit\_transform()도 제공함.

# 사이킷런의 주요 모듈(1/2)

분류	모듈명	설명
예제 데이터	<code>sklearn.datasets</code>	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	<code>sklearn.preprocessing</code>	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자 형 코드 값으로 인코딩, 정규화, 스케일링 등)
	<code>sklearn.feature_selection</code>	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 선택 작업 수행하는 다양한 기능 제공
	<code>sklearn.feature_extraction</code>	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 Tf-Idf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 <code>sklearn.feature_extraction.text</code> 모듈에, 이미지 데이터의 피처 추출은 <code>sklearn.feature_extraction.image</code> 모듈에 지원 API가 있음.
피처 처리 & 차원 축소	<code>sklearn.decomposition</code>	차원 축소와 관련한 알고리즘을 지원하는 모듈임. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있음

# 사이킷런의 주요 모듈(2/2)

분류	모듈명	설명
데이터 분리, 검증 & 파라미터 튜닝	<code>sklearn.model_selection</code>	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	<code>sklearn.metrics</code>	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	<code>sklearn.ensemble</code>	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공
	<code>sklearn.linear_model</code>	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
	<code>sklearn.naive_bayes</code>	나이브 베이즈 알고리즘 제공. 가우시안 NB, 다항 분포 NB 등.
	<code>sklearn.neighbors</code>	최근접 이웃 알고리즘 제공. K-NN 등
	<code>sklearn.svm</code>	서포트 벡터 머신 알고리즘 제공
	<code>sklearn.tree</code>	의사 결정 트리 알고리즘 제공
유틸리티	<code>sklearn.cluster</code>	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등 )
	<code>sklearn.pipeline</code>	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

# 사이킷런 내장 데이터 세트

- 외부에서 별도로 다운로드 할 필요 없이, 예제로 제공하는 데이터

분류나 회귀 연습용 예제 데이터

1) 보스턴 주택 가격 예측을 위한 데이터

API 명	설명
<code>datasets.load_boston()</code>	회귀 용도이며, 미국 보스턴의 집 피쳐들과 가격에 대한 데이터 세트
<code>datasets.load_breast_cancer()</code>	분류 용도이며, 위스콘신 유방암 피쳐들과 악성/양성 레이블 데이터 세트
<code>datasets.load_diabetes()</code>	회귀 용도이며, 당뇨 데이터 세트
<code>datasets.load_digits()</code>	분류 용도이며, 0에서 9까지 숫자의 이미지 픽셀 데이터 세트
<code>datasets.load_iris()</code>	분류 용도이며, 붓꽃에 대한 피쳐를 가진 데이터 세트

2) 붓꽃의 `feature_names`는 4개로 꽃잎 (길이, 넓이) 꽃받침 (길이, 넓이)

# (실습1) 다른 데이터 불러오기(1)

## (연습) 다른 데이터 세트 불러오기

```
▶ import pandas as pd
```

```
▶ from sklearn.datasets import load_boston  
boston = load_boston()  
print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
▶ print('\n type:', type(boston.feature_names))  
print('\n shape:', len(boston.feature_names))  
print('\n name:', boston.feature_names)  
print('\n t_target:', type(boston.target))  
print('\n t_shape:', len(boston.target))
```

## (실습1) 다른 데이터 불러오기(2)

```
df_boston = pd.DataFrame(data=boston.data, columns=boston.feature_names)
df_boston['label'] = boston.target
df_boston.head(3)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	label
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

type: <class 'numpy.ndarray'>

shape: 13

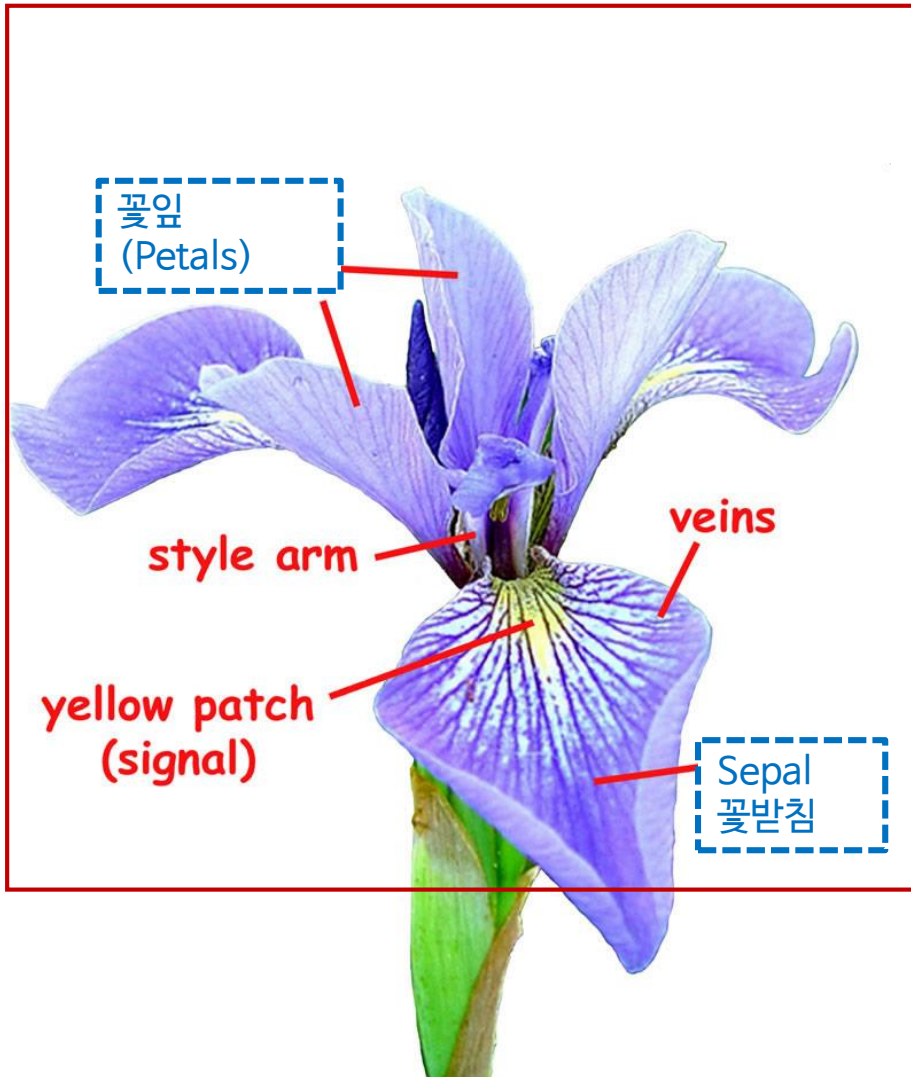
name: ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'  
'B' 'LSTAT']

t\_target: <class 'numpy.ndarray'>

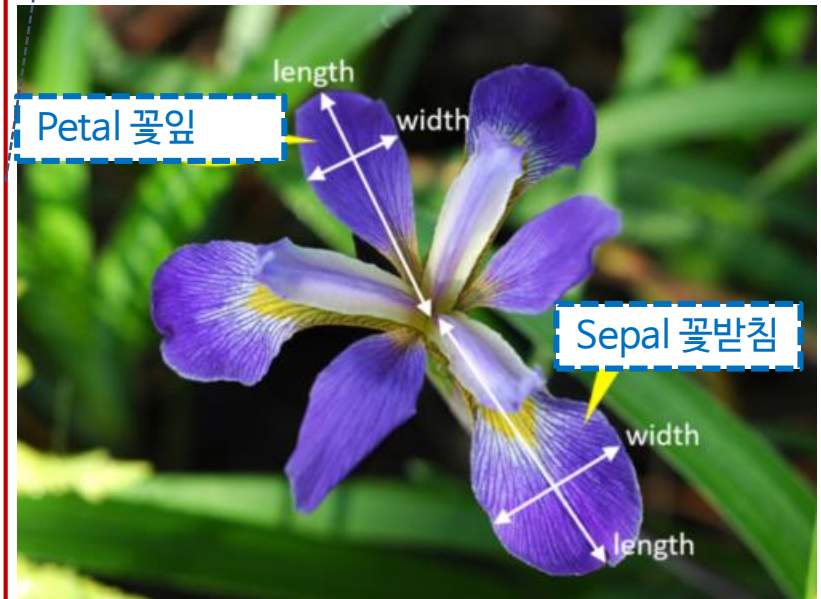
t\_shape: 506



## (실습2) 붓꽃 데이터를 불러오기



그림을 보면 직관적으로, (크기를 가름해보면) 꽃받침(sepal) 길이, 넓이, 꽃잎 길이, 넓이 순으로 데이터 값이 될 것이라고 추정해 볼 수 있다.





# 붓꽃(Iris) 품종 (레이블)

- 붓꽃 품종 3종 데이터 세트
  - ✓ 피쳐(Feature) : 꽃잎(petal)의 길이와 너비, 꽃받침(sepal)의 길이와 너비
- 지도학습(Supervised Learning)
  - ✓ 레이블데이터로 모델을 학습한 후 별도의 테스트 세트에서 미지의 레이블 예측.
  - ✓



**Iris Versicolor**



**Iris Setosa**



**Iris Virginica**

# 붓꽃 예측 프로세스 정리

- 데이터 세트 분리
  - ✓ 학습, 테스트, (검증) 데이터
- 모델 학습
  - ✓ 가장중요. 어떻게 학습하는 것인가?
  - ✓ 신경망은 이해하기 쉬운데. 블랙박스인가?
- 예측 수행
  - ✓ 예측 수행이란 무엇인가? 학습은 왜 했으며, 무엇을 하고자 하는 것인가?
  - ✓ 학습한 것이란 무엇인가? 왜 학습을 해야만 하는가?
  - ✓ 신경망에서는 웨이트 저장으로 이해할 수 있다.
- 평가
  - ✓ 예측의 의미는 무엇인가? 실제 예측하는 것인가?
  - ✓ 라벨은 이미 다 되어 있는데 무엇을 예측하는 것인가?

# Scikit-learn 첫 적용 : 붓꽃(1)

## 사이킷런을 이용하여 붓꽃(Iris) 품종 예측하기 붓꽃 데이터 가지고 오기

- sklearn.datasets에는 사이킷런에서 자체적으로 제공하는 데이터가 있음

(2020. 01. 24. 테스트)

```
In [1]: ▶ from sklearn.datasets import load_iris
```

sklearn.tree는 결정 트리 분류를 하는 모듈임

```
In [2]: ▶ from sklearn.tree import DecisionTreeClassifier
```

# Scikit-learn 첫 적용 : 붓꽃(2)

- model\_selection 모듈은 훈련, 시험, 검증 데이터 분리
- 최적의 하이퍼 파라미터로 평가
- 하이퍼 파라미터란 무엇인가?

```
In [3]:  ▶ from sklearn.model_selection import train_test_split
```

```
In [4]:  ▶ import pandas as pd
```

붓꽃 데이터 세트를 로딩합니다.

```
In [5]:  ▶ iris = load_iris()
```

Iris 데이터 세트에서 피쳐(feature)만으로 된 데이터를 numpy로 가지고 있습니다.

# Scikit-learn 첫 적용 : 붓꽃(3)

```
In [6]: ▶ iris_data = iris.data
```

- iris.target은 붓꽃 데이터 세트에서 레이블을 numpy로 가지고 왔음

```
In [7]: ▶ 1 iris_label = iris.target  
2 print('iris target값:', iris_label)
```

```
iris target값: [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 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]
```

라벨은 0 (setosa), 1, 2

# Scikit-learn 첫 적용 : 붓꽃(4)

```
In [8]: ▶ print('iris target명:', iris.target_names)
```

```
iris target명: ['setosa' 'versicolor' 'virginica']
```

```
In [9]: ▶ print('iris feature_names명:', iris.feature_names)
```

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

붓꽃의 feature\_names는 4개로  
꽃잎 (길이, 넓이) 꽃바침(길이, 넓이)

# Scikit-learn 첫 적용 : 붓꽃(5)

```
In [10]: ▶ 1 iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)
          2 iris_df['label'] = iris.target
          3 iris_df.head(3)
```

Out [10]:

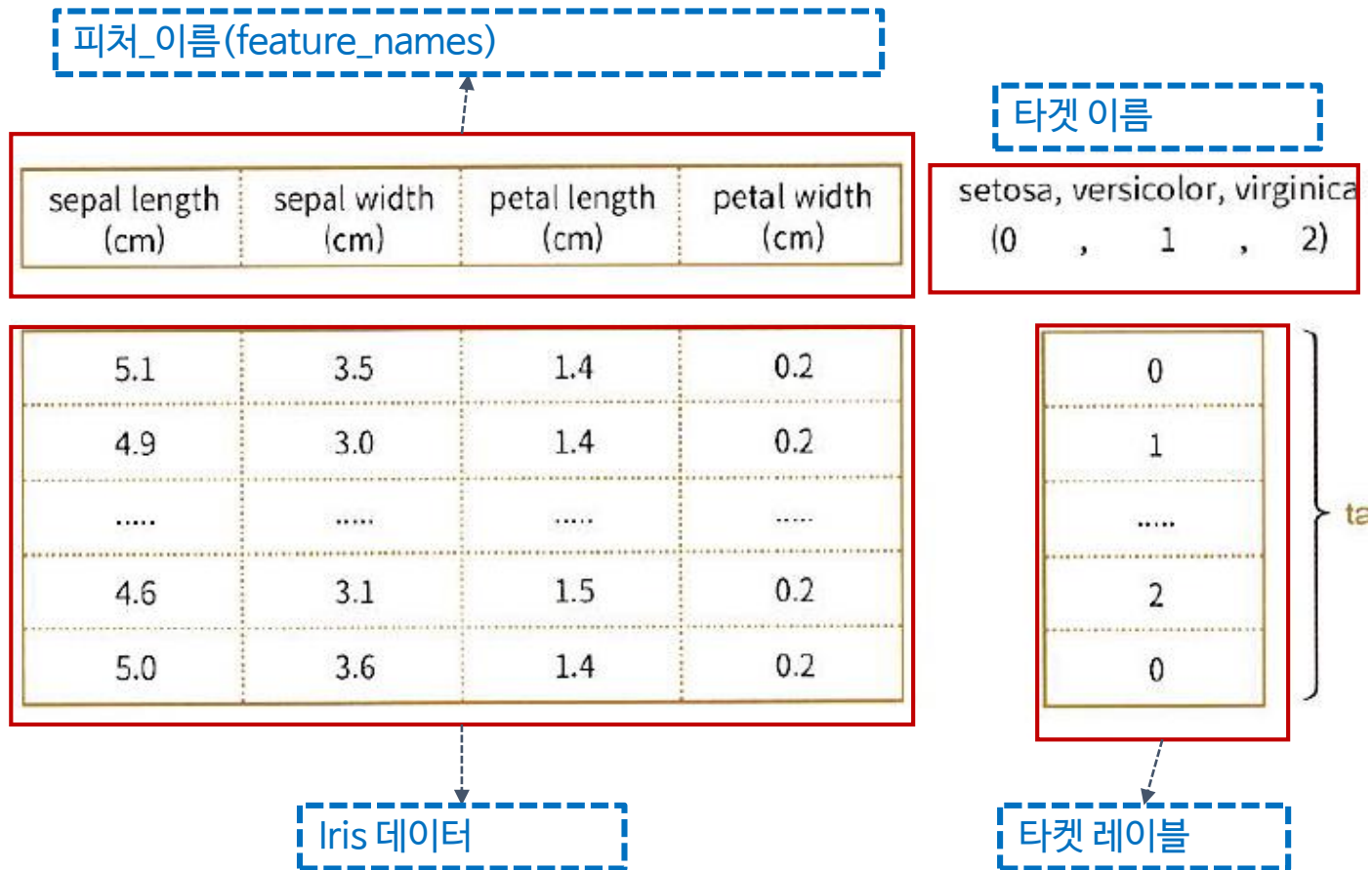
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

꽃받침(sepal)의  
길이가 가장 크고,  
다음은 width이다.

파다스에 label  
컬럼을 추가함.

(참고) 그림을 보면 직관적으로, (크기를 가름해보면) 꽃받침(sepal) 길이,  
넓이, 꽃잎 길이, 넓이 순으로 데이터 값이 될 것이라고 추정해 볼 수 있다.

# 붓꽃(iris) 데이터 구조 : load\_iris()





# Scikit-learn 첫 적용 : 붓꽃(6)

데이터 = 훈련(train) + 테스트(test)로 나누어야 한다

- 왜 나누는 것일까?
- 그럼 얼마의 비유로 나눌 것인가?
- 검증 데이터 셋은 무엇인가?

사이킷런에서는 붓꽃 원본 데이터를 학습하기 위하여  
훈련과 테스트로 자동으로 나누는 API를 제공한다.

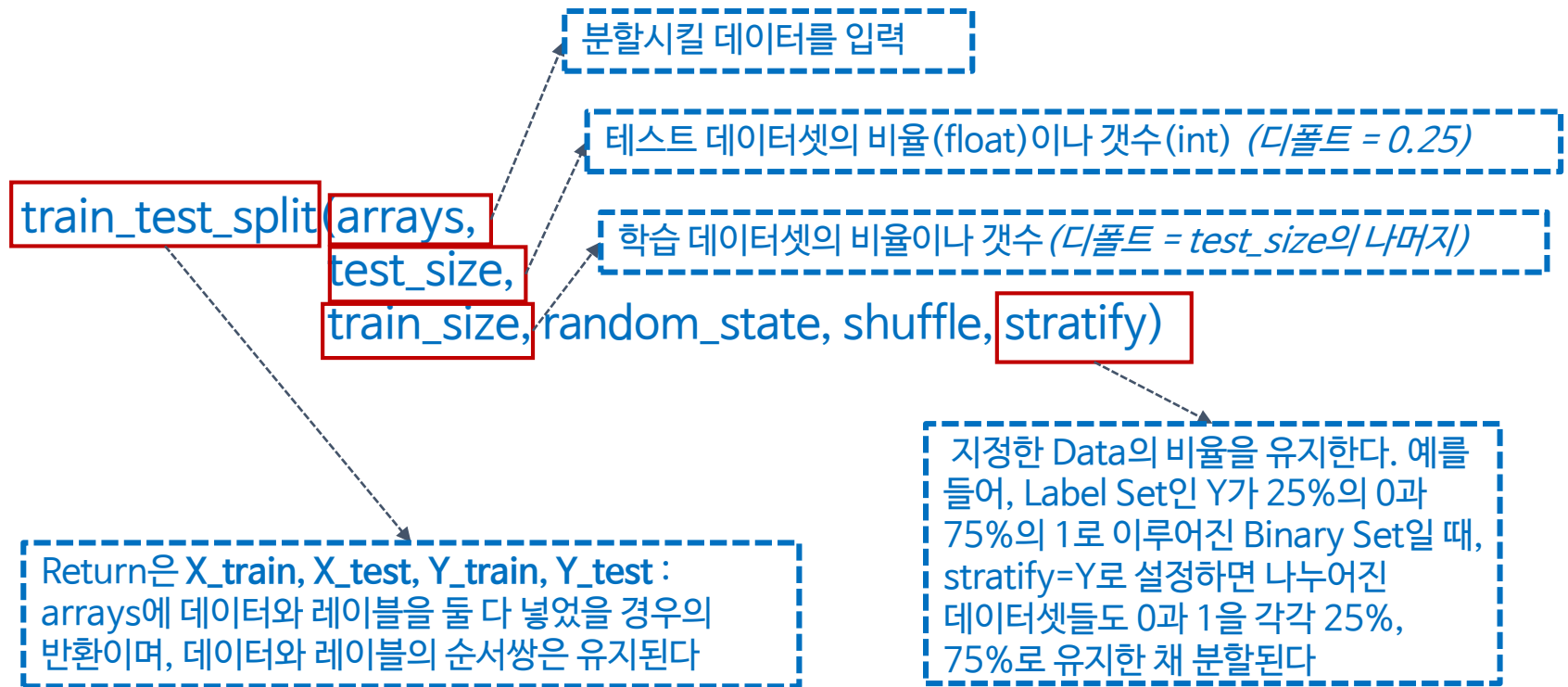
```
▶ X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label,  
                                                    test_size=0.2, random_state=99)
```

- DecisionTreeClassifier 객체 생성

```
▶ dt_clf = DecisionTreeClassifier(random_state=99)
```

랜덤 씨드는 임의의 값이  
가능하나, 특정 임의의 값을  
지정하면, 같은 계산에서  
같은 결과를 얻는다!

# Scikit-learn 첫 적용 : 붓꽃(7)



## Scikit-learn 첫 적용 : 붓꽃(8)

```
dt_clf = DecisionTreeClassifier(random_state=99)
```

학습 수행

의사 결정 트리 객체 생성하고, 생성된 의사결정 트리 객체의 fit()  
메서드에 학습용 피쳐 데이터 속성과 레이블 데이터를 입력해서 학습함

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

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',  
                        max_depth=None, max_features=None, max_leaf_nodes=None,  
                        min_impurity_decrease=0.0, min_impurity_split=None,  
                        min_samples_leaf=1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, presort='deprecated',  
                        random_state=99, splitter='best')
```

# Scikit-learn 첫 적용 : 붓꽃(9)

## sklearn.metrics.accuracy\_score

```
sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)
```

### 테스트 데이터로 예측하기

실제 레이블 세트, y\_pred는 예측 레이블 세트

- 학습이 완료된 DecisionTreeClassifier 객체에서 테스트 데이터 세트로 예측 수행.

▶ `pred = dt_clf.predict(X_test)` → X\_test는 학습에 참여하지 않은 순수한 데이터

▶ 

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

예측 정확도: 0.9333 → 정확도 93.3%

# Scikit-learn 첫 적용 : 붓꽃(10)

## 머신러닝 분류 예측 프로세스 (붓꽃)



붓꽃 데이터를 이용하여 머신러닝 분류 수행 프로세스

<그림: 파이썬 머신러닝 완벽가이드, 위키북스, 2019>

# 데이터 분리: train\_test\_split()

## 학습/테스트 데이터 셋 분리 – train\_test\_split()

- 학습 목표는 테스트 데이터 없이 학습 데이터를 사용하여
- 예측한 경우 정확도를 확인하고
- 무엇인 문제인가를 파악하여라

학습 데이터와 테스트 데이터 분리는 꼭 해야하는가?

- ✓ 이유는?
- ✓ 학습과 테스트는 70:30 비율로 분리하는 경우가 많다.

```
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)

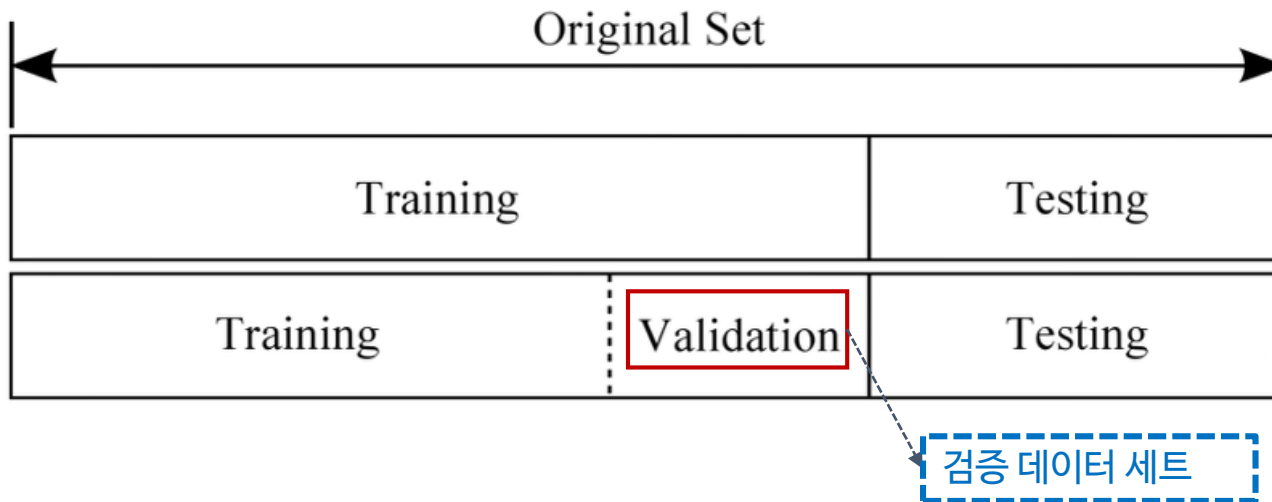
# 학습 데이터 셋으로 예측 수행
pred = dt_clf.predict(train_data)
print('예측 정확도:', accuracy_score(train_label, pred))
```

예측 정확도: 1.0

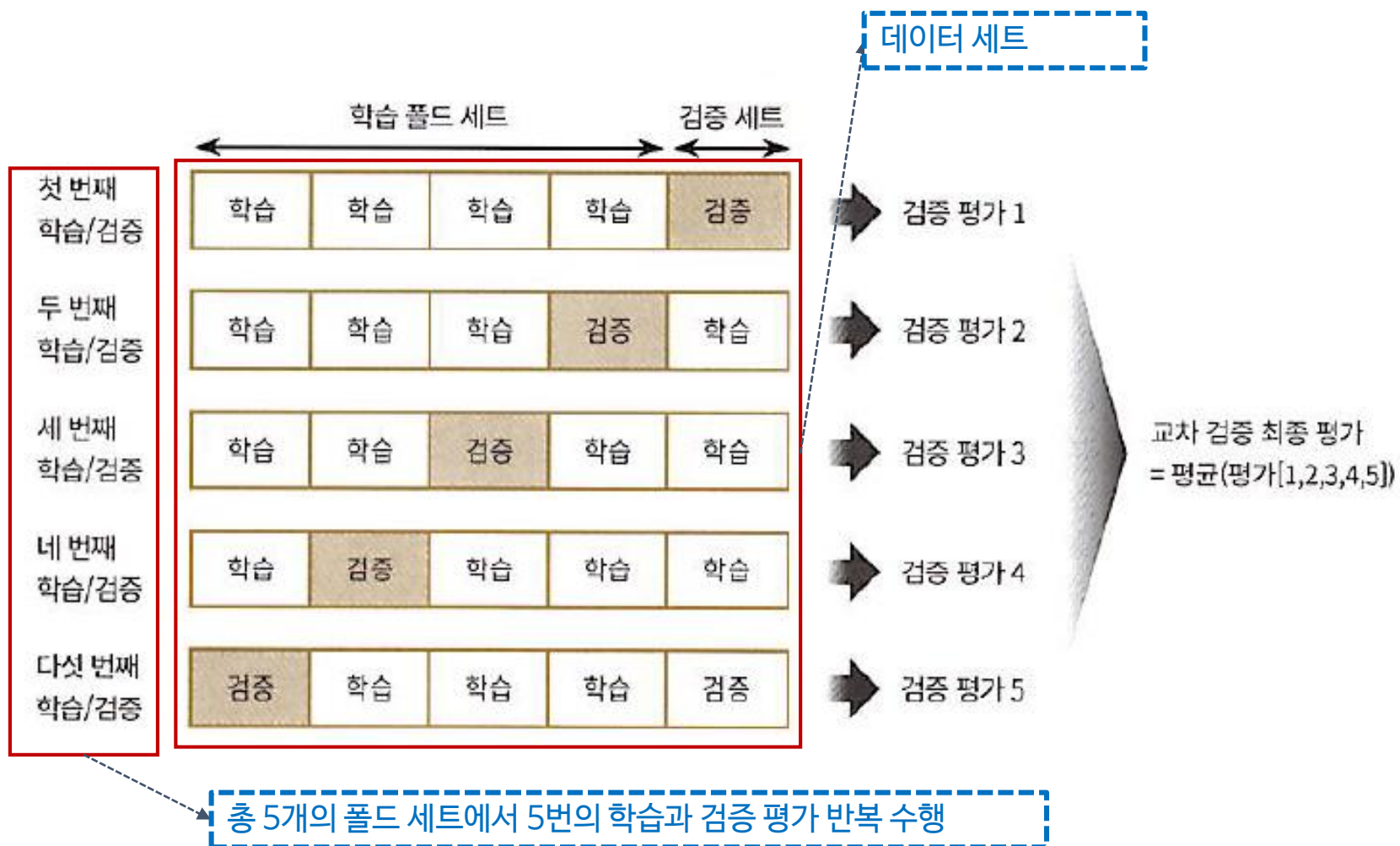
# 과적합(Overfitting)을 고민해보자.

- 과적합은 모델이 학습데이터에만 과도하게 최적화 됨
  - ✓ 실제 다른 데이터를 테스트하면 정확도가 과도하게 떨어지는 현상이 발생
  - ✓ 고정된 학습과 테스트 데이터로 평가하다 보면, 편향되게 모델을 유도
- 교차검증이 필요.
  - ✓ 학습/검증/테스트

(0.7/0.1/0.2) 혹은 (0.6/0.1/0.3)

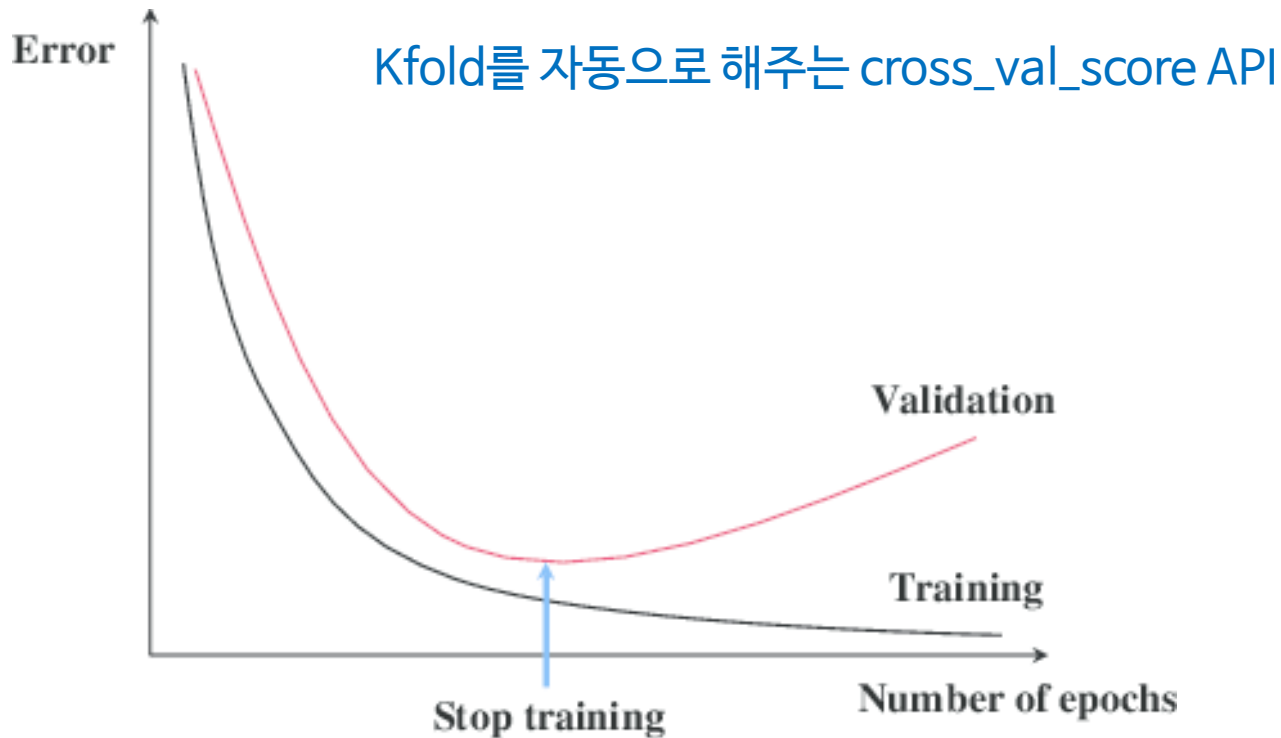


# 교차검증





# 머신러닝에서 검증 정확도의 의미는



```
sklearn.model_selection.cross_val_score(estimator, X, y=None, groups=None, scoring=None,  
cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)
```

교차 검증 개수 (디폴트=5개)

# 하이퍼 파라미터 튜닝

- 하이퍼 파라미터(Hyper-parameter)는 무엇인가?
  - ✓ 참고, 신경망에서 Learning Rate, Batch size, # Layers 등
- Hyper-parameter turning 방법
  - ✓ Grid Search
  - ✓ Random Search
  - ✓ Bayesian Optimization
- GridSeachCV API
  - ✓ 균일한 그리드 사용
  - ✓ 촘촘한 파라미터의 최적값을 구함
  - ✓ 가장 기초적이며 많은 계산시간을 요구한다.
  - ✓ High Throughput 계산에 적합

# 최적 하이퍼 파라미터 튜닝 (1)

## 실습03. GridSearchCV 파라미터 최적화

```
▶ 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

dtree = 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)
```

## 최적 하이퍼 파라미터 튜닝 (2)

### parameter 들을 dictionary 형태로 설정

```
parameters = {'max_depth':[1,2,3], 'min_samples_split':[2,3]}
```

```
▶ from sklearn.model_selection import GridSearchCV
```

```
### parameter 들을 dictionary 형태로 설정
```

```
parameters = {'max_depth':[1,2,3], 'min_samples_split':[2,3]}
```

- param\_grid의 하이퍼 파라미터들을 3개의 train, test set fold 로 나누어서 테스트 수행 설정.
- refit=True 가 default 임. True이면 가장 좋은 파라미터 설정으로 재 학습 시킴.

```
▶ grid_dtree = GridSearchCV(dtree, param_grid=parameters, cv=3, refit=True)
```

```
# 붓꽃 Train 데이터로 param_grid의 하이퍼 파라미터들을 순차적으로 학습/평가 .  
grid_dtree.fit(X_train, y_train)
```

## 최적 하이퍼 파라미터 튜닝 (3)

```
GridSearchCV(cv=3, error_score=nan,  
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,  
                                              criterion='gini', max_depth=None,  
                                              max_features=None,  
                                              max_leaf_nodes=None,  
                                              min_impurity_decrease=0.0,  
                                              min_impurity_split=None,  
                                              min_samples_leaf=1,  
                                              min_samples_split=2,  
                                              min_weight_fraction_leaf=0.0,  
                                              presort='deprecated',  
                                              random_state=None,  
                                              splitter='best'),  
             iid='deprecated', n_jobs=None,  
             param_grid={'max_depth': [1, 2, 3], 'min_samples_split': [2, 3]},  
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,  
             scoring=None, verbose=0)
```

## 최적 하이퍼 파라미터 튜닝 (4)

```
# GridSearchCV 결과 추출하여 DataFrame으로 변환
```

```
import pandas as pd
```

```
scores_df = pd.DataFrame(grid_dtree.cv_results_)
```

```
scores_df[['params', 'mean_test_score', 'rank_test_score', 'W',  
           'split0_test_score', 'split1_test_score', 'split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 1, 'min_samples_split': 2}	0.657143	5	0.657143	0.657143	0.657143
1	{'max_depth': 1, 'min_samples_split': 3}	0.657143	5	0.657143	0.657143	0.657143
2	{'max_depth': 2, 'min_samples_split': 2}	0.933333	3	0.942857	0.914286	0.942857
3	{'max_depth': 2, 'min_samples_split': 3}	0.933333	3	0.942857	0.914286	0.942857
4	{'max_depth': 3, 'min_samples_split': 2}	0.942857	1	0.971429	0.914286	0.942857
5	{'max_depth': 3, 'min_samples_split': 3}	0.942857	1	0.971429	0.914286	0.942857

## 최적 하이퍼 파라미터 튜닝 (5)

```
▶ print('GridSearchCV 최적 파라미터:', grid_dtrees.best_params_)
```

GridSearchCV 최적 파라미터: {'max\_depth': 3, 'min\_samples\_split': 2}

```
▶ print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_dtrees.best_score_))
```

GridSearchCV 최고 정확도: 0.9429

- GridSearchCV의 refit으로 이미 학습이 된 estimator 반환
- GridSearchCV의 best\_estimator\_는 이미 최적 하이퍼 파라미터로 학습이 됨

```
▶ estimator = grid_dtrees.best_estimator_  
pred = estimator.predict(X_test)  
print('테스트 데이터 세트 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

테스트 데이터 세트 정확도: 0.9556

# 데이터 전처리 (1)

- 사이킷런 머신러닝은 사용전에 데이터를 전처리 해야함
  - ✓ 결손값, NaN, Null는 허용하지 않음
  - ✓ 문자열 값을 입력으로 사용하지 않음
  - ✓ 숫자로 인코딩해서 변환 함
  - ✓ 피처는 카데고리형 피처는 코드 값으로 변환
  - ✓ 문자형 피처는 피처 벡터화(Vectorization)을 사용
  - ✓ 불필요한 피처는 삭제 가능
- 전처리의 영향은?
  - ✓ 전처리에 따라서 알고리즘의 복잡도가 정해지며, 예측 성능을 떨어뜨리기도 함



## 데이터 전처리 (2)

- 레이블 인코딩(Label encoding)

✓ 예를들면, 상품 데이터의 구분 : TV 1, 냉장고 2, 선풍기 3 등

원본 데이터		상품 분류를 레이블 인코딩한 데이터	
상품 분류	가격	상품 분류	가격
TV	1,000,000	0	1,000,000
냉장고	1,500,000	1	1,500,000
전자렌지	200,000	4	200,000
컴퓨터	800,000	5	800,000
선풍기	100,000	3	100,000
선풍기	100,000	3	100,000
믹서	50,000	2	50,000
믹서	50,000	2	50,000

# 데이터 전처리 (3) : 원-핫 인코딩

- ✓ 피처값 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 컬럼에만 1 표시
- ✓ 나머지 컬럼에는 0을 표시; (예) 0~9까지 이미지를 숫자 (MNIST)

원본 데이터

원-핫 인코딩

상품 분류	상품분류_ TV	상품분류_ 냉장고	상품분류_ 믹서	상품분류_ 선풍기	상품분류_ 전자렌지	상품분류_ 컴퓨터
TV	1	0	0	0	0	0
냉장고	0	1	0	0	0	0
전자렌지	0	0	0	0	1	0
컴퓨터	0	0	0	0	0	1
선풍기	0	0	0	1	0	0
선풍기	0	0	0	1	0	0
믹서	0	0	1	0	0	0
믹서	0	0	1	0	0	0

# 데이터 전처리 (4) : 원-핫 인코딩

## 데이터 인코딩

- 레이블 인코딩(Label encoding)

```
from sklearn.preprocessing import LabelEncoder

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

# LabelEncoder를 객체로 생성한 후, fit() 과 transform() 으로 label 인코딩 수행.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
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']

```
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']

# 데이터 전처리 (5) : 원-핫 인코딩

- 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 변환을 위해 LabelEncoder로 변환합니다.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
# 2차원 데이터로 변환합니다.
labels = labels.reshape(-1,1)

# 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)
```

원-핫 인코딩 데이터

```
[[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.]
```

원-핫 인코딩 데이터 차원  
(8, 6)

# 데이터 전처리 (6) : 원-핫 인코딩

판다스 API `get_dummies()`: 카테고리 값을 숫자 형으로 변화할 필요 없이 바로 변환 가능

```
import pandas as pd

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

	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

# 피쳐 스케일링과 정규화 (1)

- 피쳐 스케일링(feature scaling)

- ✓ 서로 다른 변수의 값 범위를 일정하게 맞추는 작업

- ✓ 표준화(Standardization)

- 데이터 피쳐 각각이 평균이 0이고 분산이 1인 가우시안 분포로 변환

- ✓ 정규화(No  $x_{i\_new} = \frac{x_i - \text{mean}(x)}{\text{stdev}(x)}$ )

- 서로 다른 피쳐의 크기를 통일하기 위해 크기를 변환해주는 개념
- 값이 0과 1 사이로 변환하는 것

$$x_{i\_new} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

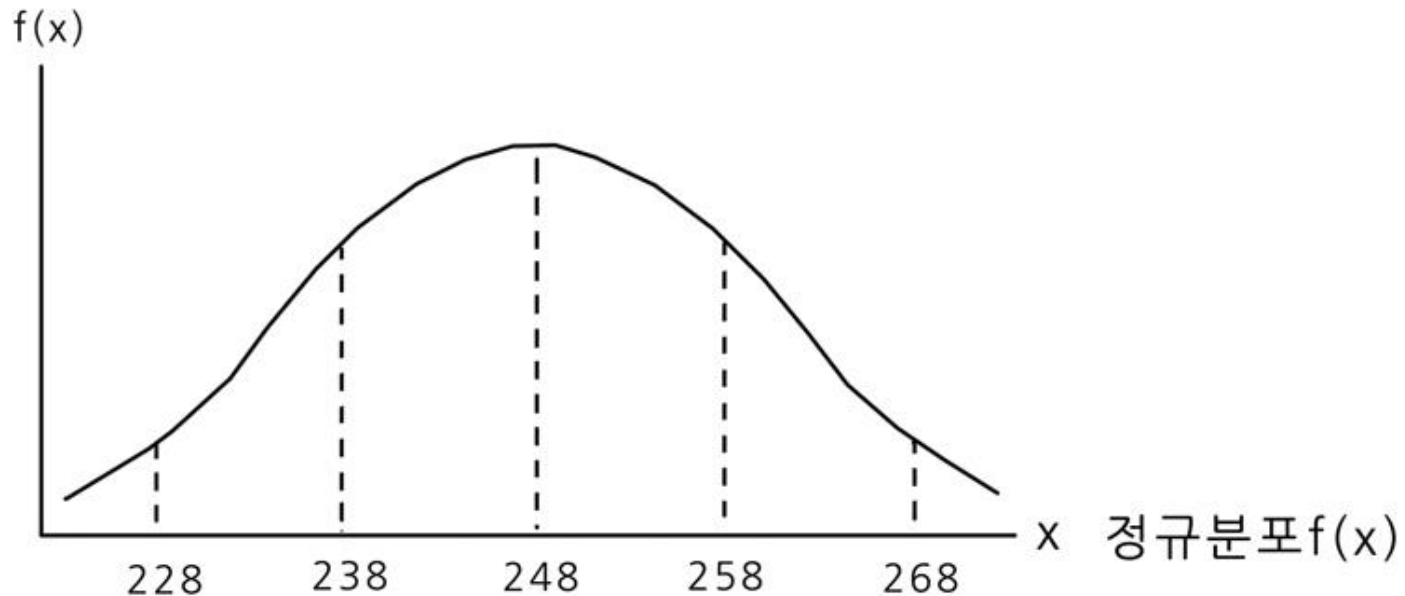
## 피쳐 스케일링과 정규화 (2)

- 사이킷런에서 제공하는 정규화 함수 Normalizer
- (벡터) 정규화
  - ✓ 선형대수의 정규화 개념이 적용
  - ✓ 개별 벡터의 크기에 맞추기 위해 변환
  - ✓ 개별 벡터를 모든 피쳐 벡터의 크기로 나눔

$$x_{i\_new} = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$$

## 피쳐 스케일링과 정규화 (3)

- StandardScaler는 표준화를 쉽게 지원하는 클래스
  - ✓ 가우시안 분포로 RBF 커널을 이용한 SVM, 선형회귀, 로지스틱 회귀는 가우시안 분포를 가지고 있다고 가정하고 구현됨
  - ✓ 따라서 표준화를 적용하는 것이 성능향상 예측에 도움이 됨





## 피처 스케일링과 정규화 (4)

- MinMaxScaler
  - ✓ 데이터 값이 0과 1사이로 변환
  - ✓ 음수 값이 있으면 (-1과 1 사이로 변환)
  - ✓ 데이터 분포가 가우시안이 아닐 경우 적용

## 피쳐 스케일링과 정규화 (5)

feature 들의 평균 값	원본	StandardScaler	MinMaxScaler
sepal length (cm)	5.843333	-1.690315e-15	0.0
sepal width (cm)	3.057333	-1.842970e-15	0.0
petal length (cm)	3.758000	-1.698641e-15	0.0
petal width (cm)	1.199333	-1.409243e-15	0.0
dtype: float64			
feature 들의 분산 값		:	
sepal length (cm)	0.685694	1.006711	1.0
sepal width (cm)	0.189979	1.006711	1.0
petal length (cm)	3.116278	1.006711	1.0
petal width (cm)	0.581006	1.006711	1.0
dtype: float64			

## 피쳐 스케일링과 정규화 (6)

```
from sklearn.preprocessing import StandardScaler

# StandardScaler 객체 생성
scaler = StandardScaler()
# StandardScaler 로 데이터 셋 변환. fit( ) 과 transform( ) 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

#transform( )시 scale 변환된 데이터 셋이 numpy ndarray로 반환되어 이를 DataFrame으로 변환
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature 들의 평균 값')
print(iris_df_scaled.mean())
print('feature 들의 분산 값')
print(iris_df_scaled.var())
```

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

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

## 피쳐 스케일링과 정규화 (7)

```
from sklearn.preprocessing import MinMaxScaler

# MinMaxScaler 객체 생성
scaler = MinMaxScaler()
# MinMaxScaler 로 데이터 셋 변환. fit() 과 transform() 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

# transform()시 scale 변환된 데이터 셋이 numpy ndarray로 반환되어 이를 DataFrame
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature들의 최소 값')
print(iris_df_scaled.min())
print('feature들의 최대 값')
print(iris_df_scaled.max())
```

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

# Thank You!

[www.ust.ac.kr](http://www.ust.ac.kr)