스마트 모빌리티 프로그래밍

Ch 17. 기계학습(ML), Scikit-learn, 회귀, 분류, 클러스터링

영남대학교 정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

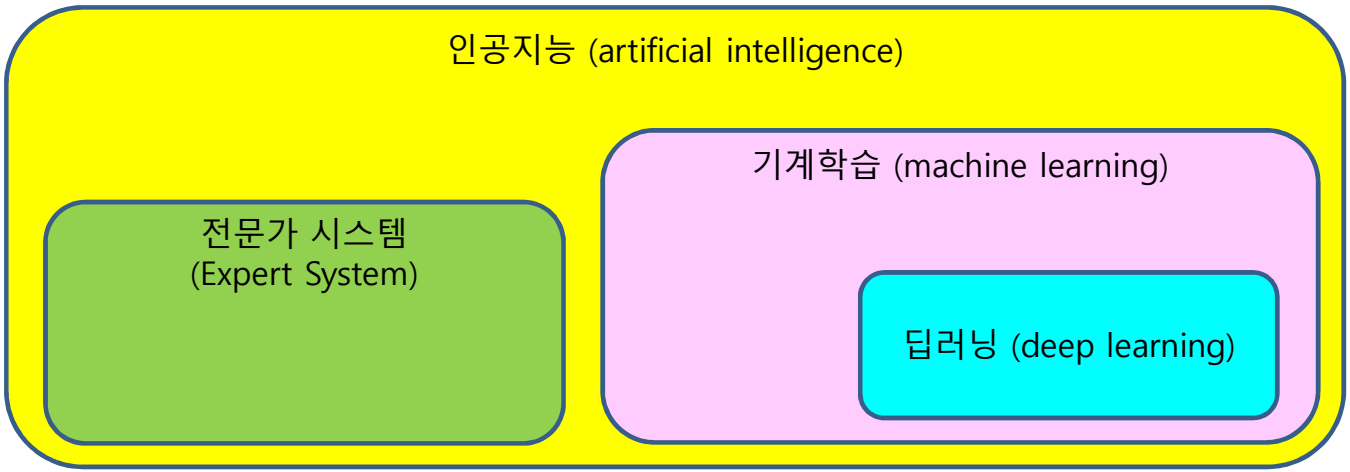
- 파이썬 기반 기계학습 (machine learning)
- Scikit-learning 기반 기계학습
- 선형 회귀 (Linear Regression)
- 분류 (Classification)
- 클러스터링 (Clustering)

파이썬 기반 기계학습

인공지능, 기계학습, 딥러닝 (심화학습)

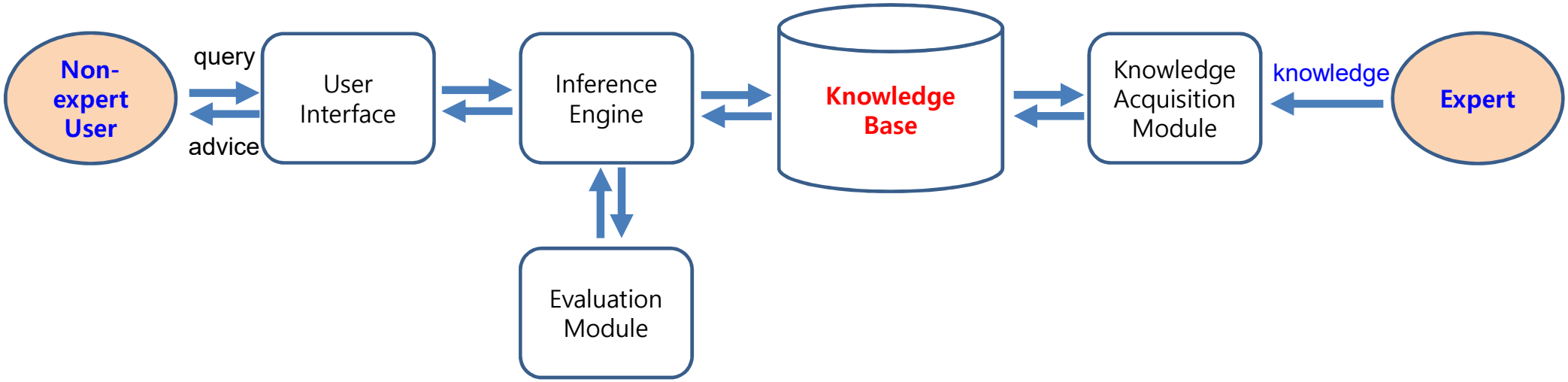
◆ 인공지능, 기계학습, 딥러닝 (심화학습)

- 인공지능 (artificial intelligence) : 인간처럼 학습하고 추론할 수 있는 소프트웨어 시스템 연구
- 기계학습 (machine learning) : 인공지능의 한 분야이며, 별도의 프로그래밍 없이 스스로 학습할 수 있는 소프트웨어 시스템 연구
- 딥러닝 (deep learning) : 인공 신경망 등을 사용하여 빅데이터로 부터 스스로 학습할 수 있는 소프트웨어 시스템 연구



전문가 시스템 (Expert System)

◆ 전문가 시스템의 기능 구조



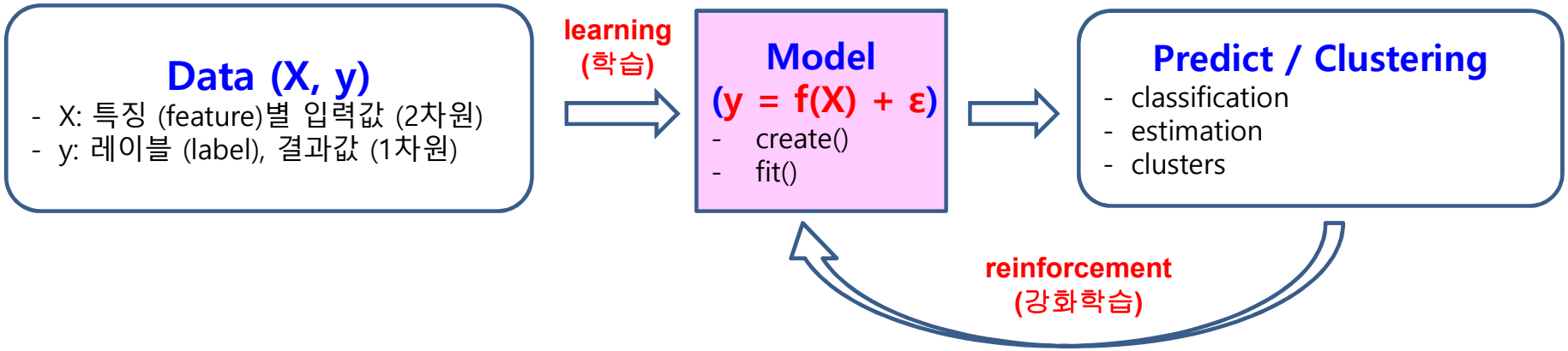
기계 학습 (machine learning)

◆ 기계 학습 (machine learning)

- 인공지능의 한 분야로 컴퓨터에 학습 기능을 부여하기 위한 연구
- 1959년 Arthur Samuel이 처음 사용
- 패턴 인식 및 계산 학습 이론에서 진화하여
  컴퓨터가 주어진 데이터를 학습하는 알고리즘을 연구
- 학습할 수 있는 데이터가 많아지면 알고리즘 성능이 향상됨
- 기계학습 알고리즘은 항상 고정적인 의사결정을 하는 프로그램과 달리,
  데이터 중심의 예측 또는 결정을 내릴 수 있음
- 기계 학습은 어떤 문제에 대하여 명시적 알고리즘을 설계하고,
  프로그래밍하는 것이 어렵거나 불가능한 경우에 주로 사용됨
- 사용분야: spam E-mail filtering, 네트워크 침입자 검출, 광학문자 인식 (OCR),
  필기체 인식, 컴퓨터 비전

기계학습의 모델

◆ 기계학습의 모델



기계 학습의 유형 - 지도학습, 비지도 학습, 강화학습

◆ 기계학습 유형별 특징

기계학습 유형	특징
지도학습 (supervised learning)	▪ 교사에 의하여 주어진 예제 (샘플)과 정답 (레이블)을 제공받음 ▪ 지도 학습의 목표는 입력을 출력에 매칭하는 일반적인 규칙(함수)를 학습 ▪ 예를 들어 강아지와 고양이를 구분하는 문제인 경우, 강아지와 고양이에 대한 영상을 제공한 후, 교사가 어떤 영상이 강아지인지, 어떤 영상이 고양이인지 구분하여 정답을 알려줌
비지도 학습 (unsupervised learning)	▪ 외부에서 정답(레이블)이 주어지지 않고, 학습 알고리즘이 스스로 입력에서 어떤 구조를 발견하는 학습 ▪ 비지도 학습을 사용하면 데이터에 숨겨져 있는 패턴을 발견할 수 있음
강화학습 (reinforcement learning)	▪ 보상 및 처벌 형태로 학습 데이터가 주어짐 ▪ 주로 차량 운전이나 상대방과의 경기와 같은 동적인 환경에서 프로그램의 행동에 대한 피드백만 제공되는 경우 ▪ 예를 들어 바둑에서 어떤 수를 두어서 승리하였다면 보상이 주어지며, 실패하였다면 처벌이 주어짐 ▪ 강화학습에서는 보상과 처벌을 통하여 학습이 이루어 짐

기계학습의 유형 및 주요 알고리즘

유형	지도 학습 (Supervised Learning)	비지도 학습 (Unsupervised Learning)	강화 학습 (Reinforcement Learning)
주요 기능	▪ 회귀 (regression) ▪ 분류 (Classification)	▪ 군집화 (Clustering) ▪ Dimensionality Reduction ▪ 패턴/구조 발견, Association ▪ Anomaly Detection	▪ Deep learning
주요 알고리즘	▪ Linear regression ▪ Logistic Regression ▪ Polynomial Regression ▪ Multivariate Adaptive Regression Splines (MARS) ▪ Decision Tree ▪ Random Forest ▪ k-NN (K-nearest neighbor) ▪ Neural Network ▪ Naïve-Bayes classifier ▪ SVM (Support Vector Machine) ▪ HMM (Hidden Markov Model)	▪ K-means clustering ▪ Spectral ▪ Hierarchical clustering ▪ AutoEncoder ▪ Expectation-Maximization (EM) ▪ PCA (Principal Component Analysis) ▪ Principal Component Regression (PCR) ▪ Multidimensional Scaling (MDS) ▪ Singular Vector Decomposition (SVD) ▪ DBSCAN (density-based spatial clustering of applications with noise)	▪ Convolutional Neural Network (CNN) ▪ Recurrent Neural Network (RNN) ▪ Stacked Auto-Encoders ▪ Deep Bolzmann Machine (DBM) ▪ Deep Belief Networks (DBN)

(Ref. List of Machine Learning Algorithms,
https://skilllx.com/list-of-machine-learning-algorithms/)

기계학습의 유형 및 주요 알고리즘

기계학습		관련 알고리즘
지도학습	분류 (classification)	▪ KNN (K-nearest neighbors) ▪ Naive Bayes ▪ Decision Tree ▪ Logistic Regression ▪ Random Forest ▪ Support Vector Machine (SVM) ▪ ANN (Artificial Neural Network)
	회귀 (regression)	▪ Linear Regression ▪ Regularized Linear Regression ▪ Ranom Forest Regression ▪ Support Vector Regression
비지도학습	군집화 (clustering)	▪ Hierarchical Clustering ▪ K-means Clustering, K-medios ▪ SOM (Self-organizing Map) ▪ DBSCAN (density-based spatial clustering of applications with noise)
	차원축소 (dimensionality reduction)	▪ PCA (Principal Component Analysis) ▪ Factor Analysis ▪ MDS (Multi-Dimensional Scaling)
	연관 규칙 학습	▪ MBA(Market Basket Analysis) ▪ Sequence Analysis ▪ Collaborative Filtering
강화 학습	강화 학습	▪ CNN (Convolutional Neural Network) ▪ Recurrent Neural Network (RNN)

대표적인 기계학습 알고리즘

◆ Top Machine Learning Algorithms You Should Know

- source: https://builtin.com/data-science/tour-top-10-algorithms-machine-learning-newbies
- Linear Regression
- Logistic Regression
- Linear Discriminant Analysis
- Classification and Regression Trees
- Naive Bayes
- K-Nearest Neighbors (KNN)
- Learning Vector Quantization (LVQ)
- Support Vector Machines (SVM)
- Random Forest
- Boosting
- AdaBoost

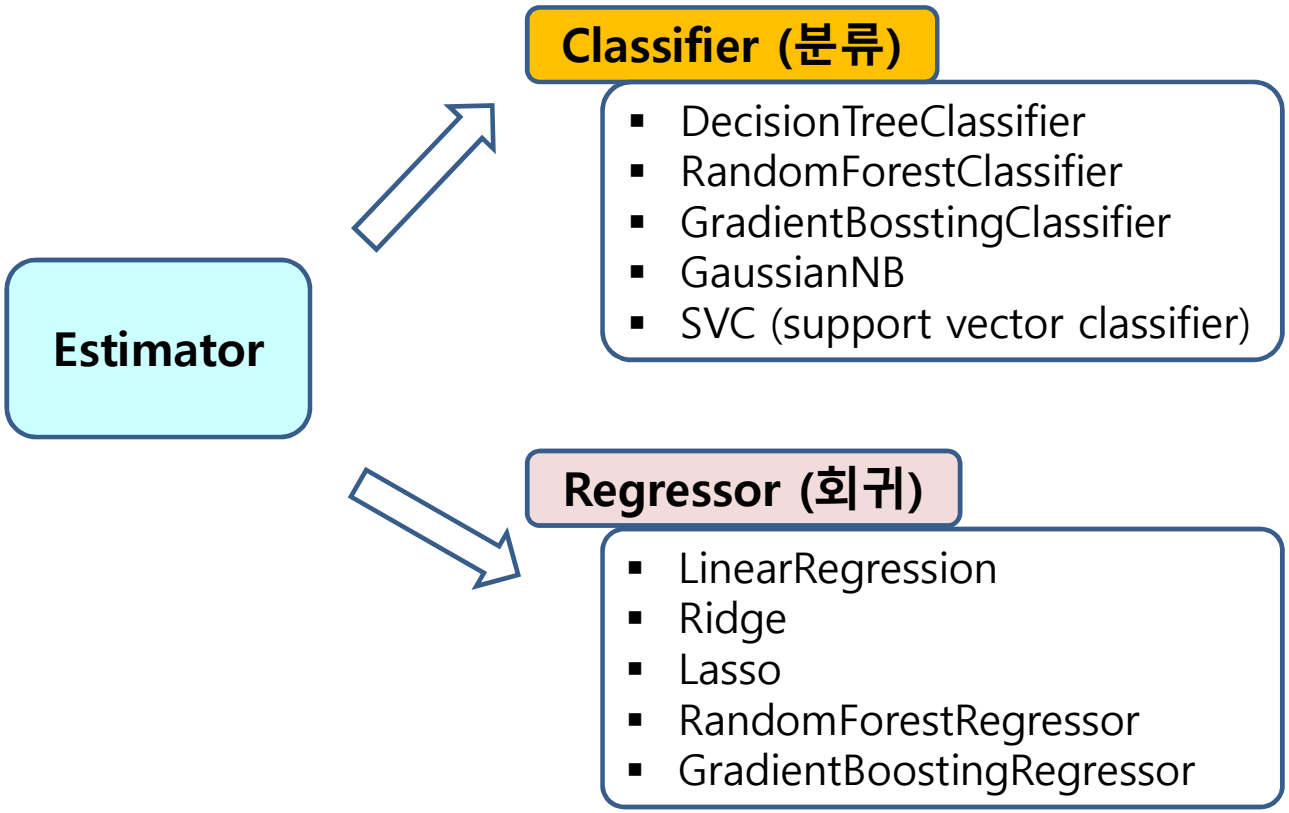
지도학습

◆ 지도학습 (supervised learning)

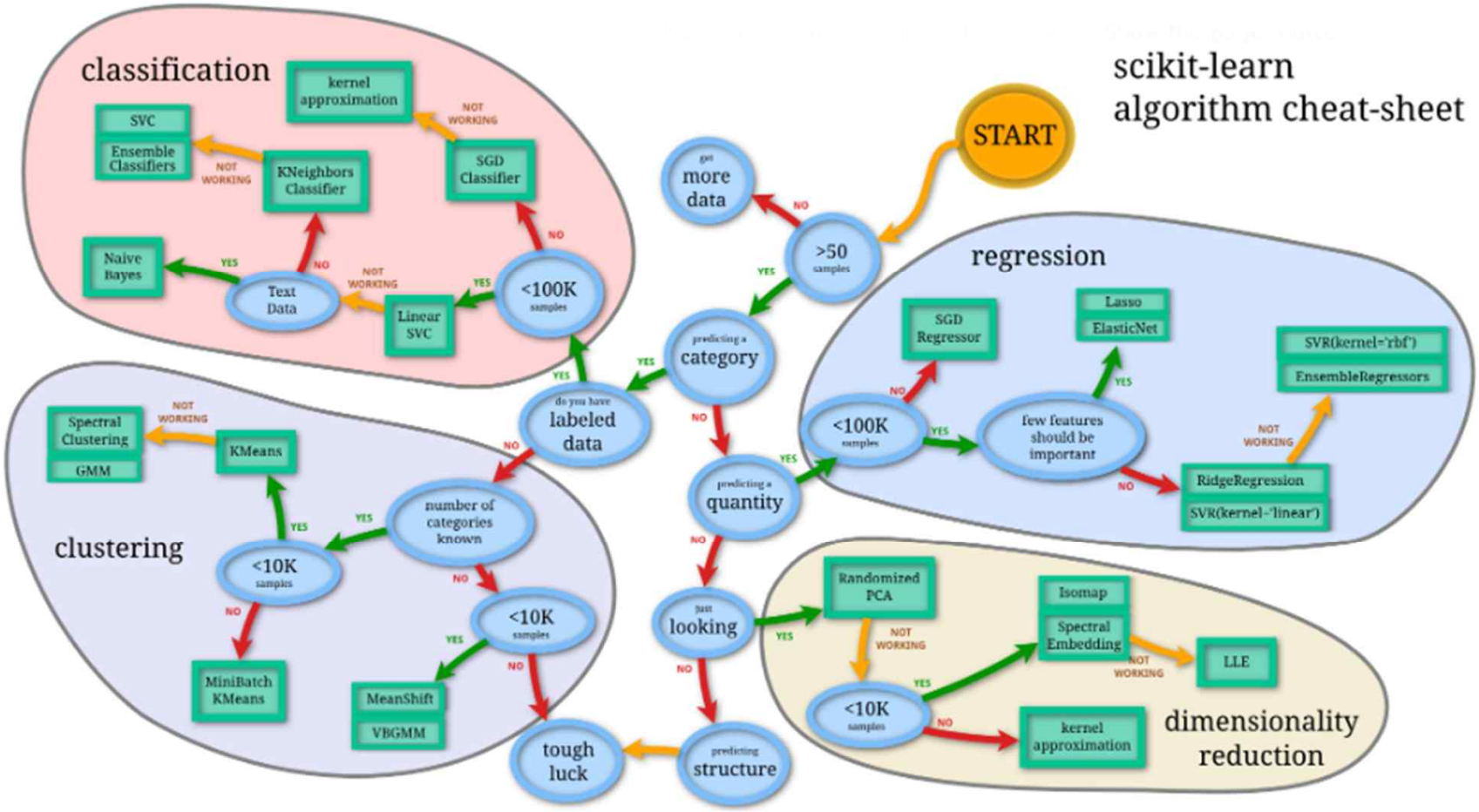
- 지도학습은 학습을 시키는 교사가 존재하는 학습방법
- 학습데이터에서 정답 (회귀: 출력값, 분류: 레이블)이 제공되므로 지도라는 용어를 사용함
- 지도학습에서는 입력을 결합하여 모델을 만들고,
  이전에 보지 못한 데이터도 적절히 예측하는 방법을 학습 시킴
- 지도학습은 크게 회귀 (regression)과 분류 (classification)으로 구분

지도학습 유형	특 징
회귀 (regression)	▪ 회귀는 주어진 입력-출력 값쌍을 학습한 후, 새로운 입력값이 들어왔을 때 합리적인 출력값을 예측 ▪ 회귀에서는 학습시키는 데이터가 이산적인 아니고 연속적이며, 입력과 출력이 모두 실수 (real number)로 표현되며, 연속적인 값을 예측 ▪ 예: 입력값 (x)에 대한 출력값을 y = f(x)의 방정식으로 예측
분류 (classification)	▪ 입력을 두 개 이상의 레이블 (유형)으로 분류하는 것 ▪ 해당 모델을 학습시킬 때 레이블을 제공하며, 올바른 레이블을 알려 줌 ▪ 학습이 끝나면 학습자가 한 번도 보지 못한 입력을 이들 레이블 중의 하나로 분류하는 시스템임 ▪ 예: 스팸 필터링, 필기체 숫자 인식

지도학습의 모든 알고리즘을 구현한 클래스 - Estimator



Choosing the Right Estimator



(참고: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

Scikit-Learn Library

◆ Scikit-Learn Library

- 분류 (classification), 회귀 (regression), 군집(clustering), 결정트리 (decision tree) 등의 다양한 기계학습 알고리즘을 적용할 수 있는 함수를 제공
- 필수 library
  - NumPy
  - SciPy
  - Matplotlib
  - Pandas

◆ Scikit-learn 설치

- pip install scikit-learn

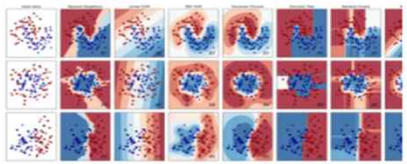
```
C:₩Users₩Owner>python -V
Python 3.9.9

C:₩Users₩Owner>pip install --upgrade scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-1.0.2-cp39-cp39-win_amd64.whl (7.2 MB)
     |████████████████████████████████| 7.2 MB 3.3 MB/s
Requirement already satisfied: numpy>=1.14.6 in c:₩users₩owner₩appdata₩local₩programs₩python₩python39₩lib₩site-packages (from scikit-learn) (1.22.1)
Requirement already satisfied: scipy>=1.1.0 in c:₩users₩owner₩appdata₩local₩programs₩python₩python39₩lib₩site-packages (from scikit-learn) (1.7.3)
Collecting joblib>=0.11
  Downloading joblib-1.1.0-py2.py3-none-any.whl (306 kB)
     |████████████████████████████████| 306 kB 6.4 MB/s
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-3.0.0-py3-none-any.whl (14 kB)
Installing collected packages: threadpoolctl, joblib, scikit-learn
Successfully installed joblib-1.1.0 scikit-learn-1.0.2 threadpoolctl-3.0.0
```

Applications of Scikit-Learn

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

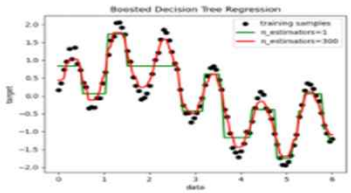


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Examples

Clustering

Automatic grouping of similar objects into sets.

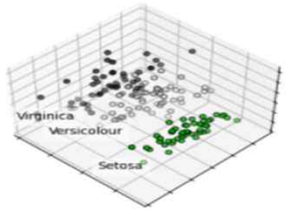
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

Examples

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...

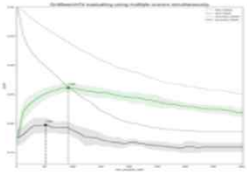


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

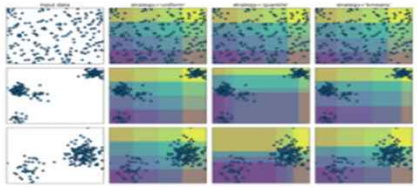


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

(source: https://scikit-learn.org/stable/)

Scikit-learn의 주요 응용 분야

분야	응용	알고리즘
분류 (classification)	▪ identifying which category an object belongs to ▪ spam detection, image recognition	▪ SVM (support vector machine) ▪ nearest neighbors ▪ random forest
회귀 (regression)	▪ predicting a continuous-valued attribute associated with an object ▪ drug response, stock prices	▪ SVR (support vector regression) ▪ nearest neighbors ▪ random forest
클러스터링 (clustering)	▪ automatic grouping of similar objects into sets ▪ customer segmentation, grouping experiment outcomes	▪ k-Means, spectral clustering, mean-shift
차원축소 (dimensionality reduction)	▪ reducing the number of random variables to consider ▪ visualization, increased efficiency	▪ k-Means, feature selection ▪ non-negative matrix factorization
모델 선정 (model selection)	▪ comparing, validating and choosing parameters and models ▪ improved accuracy via parameter tuning	▪ grid search, cross validation, metrics
전처리 (preprocessing)	▪ 특징 추출 (feature extraction) 및 정규화 (normalization) ▪ 입력 데이터를 기계학습 알고리즘에 사용할 수 있도록 전처리	▪ preprocessing ▪ feature extraction and normalization

Scikit-learn의 주요 모듈

분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업을 수행하는 다양한 기능 제공
	sklearn.feature_extraction	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 sklearn.feature_extraction.text 모듈에, 이미지 데이터의 피처 추출은 sklearn.feature_extraction.image 모듈에 지원 API가 있음
피처 처리 & 차원 축소	sklearn.decomposition	차원 축소와 관련한 알고리즘을 지원하는 모듈이다. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있다.
데이터 분리, 검증 & 파라미터 튜닝	sklearn.model_selection	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	sklearn.metrics	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	sklearn.ensemble	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공
	sklearn.linear_model	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Desccent) 관련 알고리즘도 제공
	sklearn.naïve_bayes	나이브 베이즈 알고리즘 제공. 가우시안 NB. 다항 분포 NB 등
	sklearn.neighbors	최 근접 이웃 알고리즘 제공. K-NN(K-Nearest Neighborhood) 등
	sklearn.svm	서포트 벡터 머신 알고리즘 제공
	sklearn.tree	의사 결정 트리 알고리즘 제공
	sklearn.cluster	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
유틸리티	sklearn.pipeline	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

Scikit-learn 제공 Datasets

◆ Scikit-learn 제공 데이터 세트

종류	download
iris (붓꽃)	load_iris(*[, return_X_y, as_frame])
diabetes(당뇨병)	load_diabetes(*[, return_X_y, as_frame, scaled])
digits(필기체 숫자)	load_digits(*[, return_X_y, as_frame])
Linnerud	load_linnerud(*[, return_X_y, as_frame])
wine(포도주)	load_wine(*[, return_X_y, as_frame])
breast_cancer(유방암)	load_breast_cancer(*[, return_X_y, as_frame])

지도학습기반
선형 회귀 (Linear Regression)

선형 회귀 (Linear Regression)

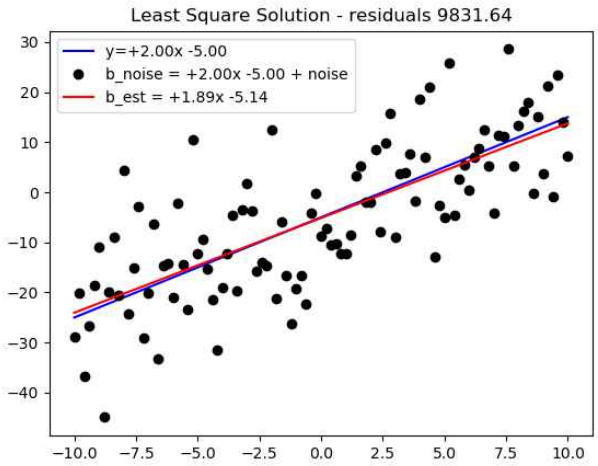
◆ 선형 회귀 (Linear Regression)

- https://ko.wikipedia.org/wiki/선형회귀
- 선형 회귀(線型回歸, linear regression)는 종속 변수 y와 한 개 이상의 독립 변수 (또는 설명 변수) X와의 선형 상관 관계를 모델링하는 회귀분석 기법
- 한 개의 설명 변수에 기반한 경우에는 단순 선형 회귀(simple linear regression), 둘 이상의 설명 변수에 기반한 경우에는 다중 선형 회귀라고 함
- 선형 회귀는 선형 예측 함수를 사용해 회귀식을 모델링하며, 알려지지 않은 파라미터는 데이터로부터 추정한다. 이렇게 만들어진 회귀식을 선형 모델이라고 한다.
- 선형 회귀는 깊이 있게 연구되고 널리 사용된 첫 번째 회귀분석 기법임
- 이는 알려지지 않은 파라미터에 대해 선형 관계를 갖는 모델을 세우는 것이,
  비선형 관계를 갖는 모델을 세우는 것보다 용이하기 때문이다.
- 선형 회귀는 여러 사용 사례가 있지만, 대개 아래와 같은 두 가지 분류 중 하나로 요약할 수 있다.
- 값을 예측하는 것이 목적일 경우, 선형 회귀를 사용해 데이터에 적합한 **예측 모형**을 개발한다.
  개발한 선형 회귀식을 사용해 y가 없는 x값에 대해 y를 예측하기 위해 사용할 수 있다.
- 종속 변수 y와 이것과 연관된 독립 변수 $X_1, \ldots, X_p$가 존재하는 경우에, 선형 회귀 분석을 사용해 X_j와 y의 관계를 정량화할 수 있다. X_j는 y와 전혀 관계가 없을 수도 있고, 추가적인 정보를 제공하는 변수일 수도 있다.
- 일반적으로 최소제곱법(least square method)을 사용해 선형 회귀 모델을 세운다. 최소제곱법 외에 다른 기법으로도 선형 회귀 모델을 세울 수 있다. 손실 함수(loss fuction)를 최소화 하는 방식으로 선형 회귀 모델을 세울 수도 있다. 최소제곱법은 선형 회귀 모델 뿐 아니라, 비선형 회귀 모델에도 적용할 수 있다.

선형 회귀 (linear regression)

◆ 선형 회귀

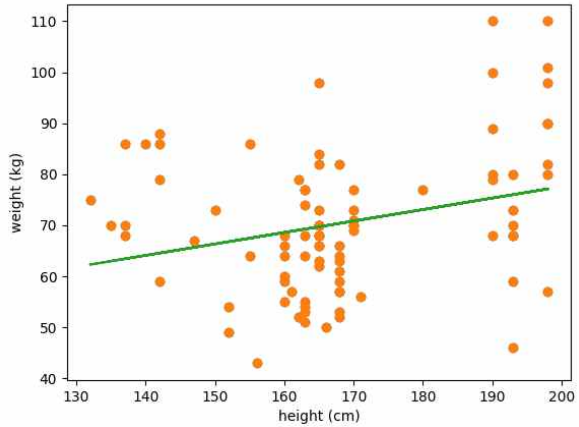
- 회귀는 입력 (x)와 출력 (y)값이 주어질 때,
  입력에서 출력으로의 매핑 함수 y = f(x)를 학습하는 것
- 입력 (x)는 다차원일 수 있음
- 선형 모델 (f(x) = mx + b)을 사용하여 회귀문제를 풀 때
  선형 회귀 (linear regression)이라 함
- 선형회귀의 사용 예
  - 부모의 키와 자녀의 키의 상관관계
  - 면적에 따른 주택의 가격
  - 나이 (연령)에 따른 실업률 예측
  - 공부시간과 학업 성적의 관계
  - CPU속도와 프로그램 실행 시간관계



Estimations with Linear Regression using Scikit-learn

```
# Estimations with Linear regression using scikit-learn
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("student_info.csv") # information of university students
print("df.head() => "); print(df.head())
x = df["height_cm"]
X = x.values.reshape(-1,1)
y = df["weight_kg"]
plt.plot(x, y, 'o')
plt.xlabel("height (cm)")
plt.ylabel("weight (kg)")

line_fitter = LinearRegression()
line_fitter.fit(X, y)
plt.plot(x, y, 'o')
py = line_fitter.predict(X)
plt.plot(x, py)
plt.show()

test_heights = ([[160]], [[170]], [[180]], [[190]])
for TH in test_heights:
    pw = line_fitter.predict(TH)
    h = TH[0][0]
    print("height {} cm => predicted_weight {:6.2f} kg".format(h, pw[0]))
```

no	gender	weight_kg	height_cm
1	m	98	198
2	m	77	170
3	m	70	170
4	m	90	198
. . . .			
88	f	100	190
89	f	54	163
90	f	57	161
91	f	101	198
92	f	110	190




```
df.head() =>
   no gender  weight_kg  height_cm
0   1      m         98        198
1   2      m         77        170
2   3      m         70        170
3   4      m         90        198
4   5      m         71        170
height 160 cm => predicted_weight  68.61 kg
height 170 cm => predicted_weight  70.85 kg
height 180 cm => predicted_weight  73.10 kg
height 190 cm => predicted_weight  75.35 kg
```

선형회귀에서 손실함수 최소화 방법

◆ 손실함수

- 선형 회귀에서 학습데이터 (x1, x2, x3, . . .)를 사용한 예측에서 선형 모델 f(x) = Wx + b을 사용할 때,
- 손실함수 (loss function, cost function)는 실제 데이터 x_i에 대한 $f(x_i)$와 예측된 선형 함수의 직선 방정식과의 차이로 표현

  Loss(W, b) = $\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$, n은 학습 데이터의 개수
- 학습에서는 손실함수 값이 최소가 되는 W와 b를 찾는 것

경사하강법 (Gradient Descent Method)

◆ 경사하강법 (gradient descent method)

- 손실함수의 경사 (gradient)를 사용하여 최적의 파라메터 값을 찾을 때 널리 사용
- 손실함수 (loss function)

$$\text{Loss(W, b)} = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$
$$= \frac{1}{n}\sum_{i=1}^{n}((Wx_i + b) - y_i)^2$$

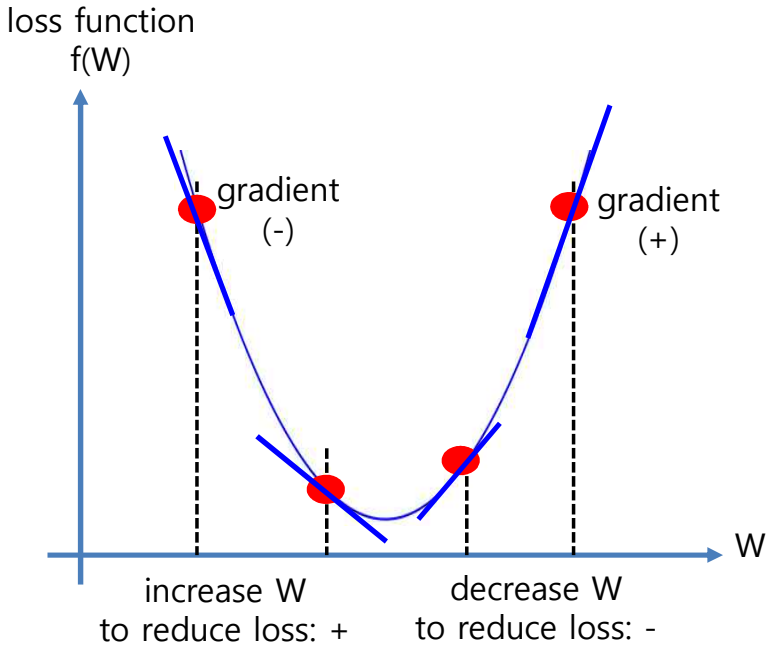
- 손실함수를 W에 대하여 미분

$$\frac{\partial \text{Loss(W, b)}}{\partial W} = \frac{2}{n}\sum_{i=1}^{n}((Wx_i + b) - y_i)$$

- 손실함수 결과값이 줄어드는 방향으로 W와 b를 update (ρ: learning rate, 학습률)

$$\text{W} = \text{W} - \rho * \frac{\partial \text{Loss(W, b)}}{\partial W}$$

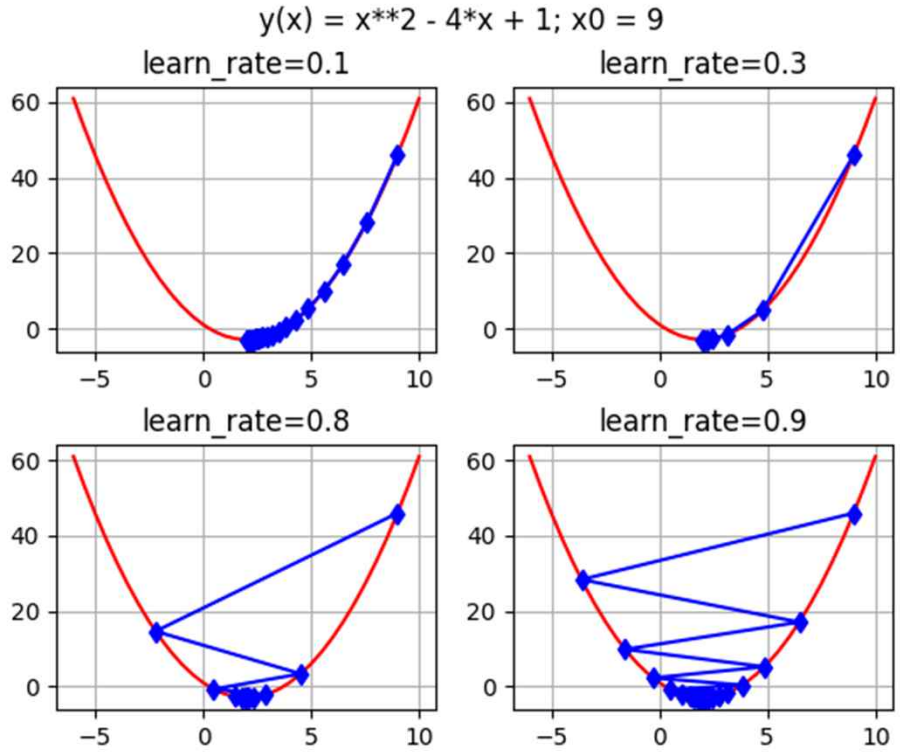
$$\text{b} = \text{b} - \rho * \frac{\partial \text{Loss(W, b)}}{\partial W}$$



경사 하강 알고리즘 (GDA)의 학습률에 따른 차이

◆ Gradient Descent Algorithm(GDA)의 학습률 (learn rate)

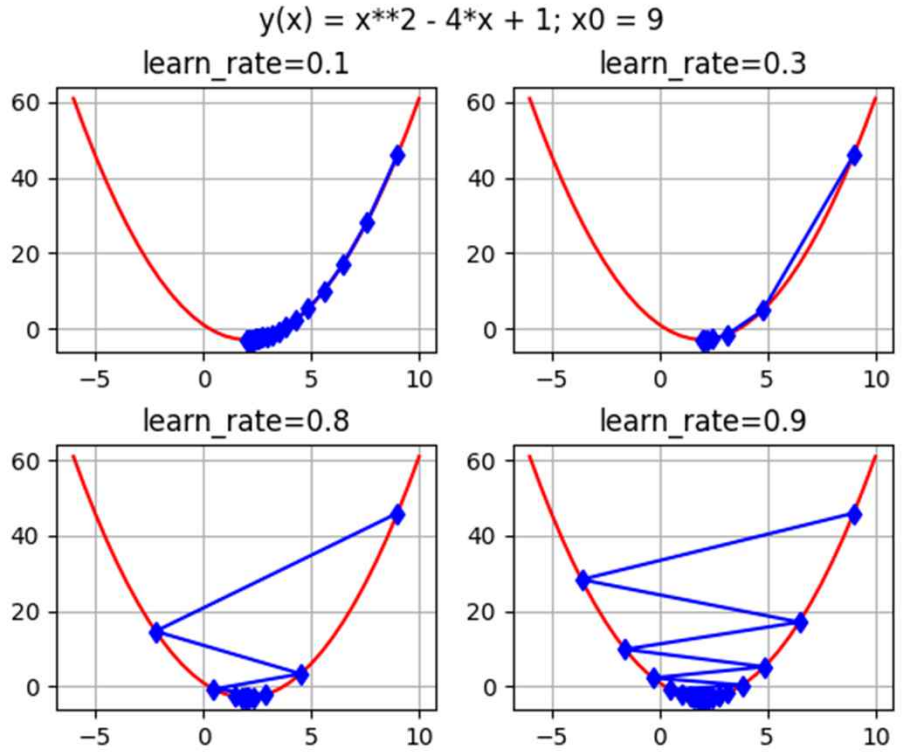
```
# gradient descent algorithm (GCA)  (1)
import numpy as np
import matplotlib.pyplot as plt

def gradient_descent(start, gradient, learn_rate,\
  max_iter, tol=0.01):
  steps = [start] # history tracking
  x = start
  for _ in range(max_iter):
    diff = learn_rate*gradient(x)
    if np.abs(diff)<tol:
      break
    x = x - diff
    steps.append(x) # history tracing
  return steps, x

def func1(x):
  return x**2-4*x+1

def gradient_func1(x):
  return 2*x - 4
```



경사 하강 알고리즘 (GDA)

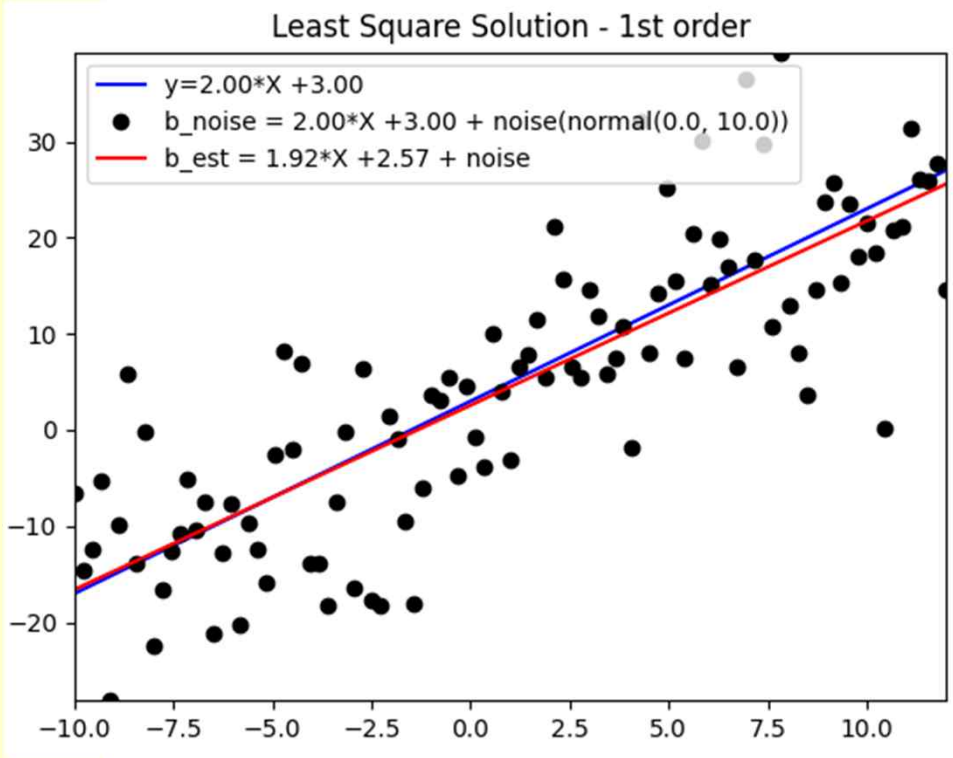
◆ Gradient Descent Algorithm(GDA) (계속)

```
# gradient descent algorithm (GCA)  (2)

#----------------
x = np.arange(-6, 10.1, 0.5)
y = func1(x)

learn_rates = [0.1, 0.3, 0.8, 0.9]
fig, ax = plt.subplots(2, 2)
plt.subplots_adjust(hspace=0.35)
plt.rc('figure', figsize=(8,8))
fig.suptitle("y(x) = x**2 - 4*x + 1; x0 = 9")

for i in range(2):
    for j in range(2):
        ax[i][j].plot(x, y, "r-")
        learn_rate = learn_rates[i*2 + j]
        history, result = gradient_descent(9, gradient_func1,\
          learn_rate, 100)
        print("history = ", history)
        y_h = list(map(func1, history))
        ax[i][j].plot(history, y_h, "bd-")
        #plt.title("learn_rate = {}".format(learn_rate))
        ax[i][j].set_title("learn_rate={}".format(learn_rate))
        ax[i][j].grid()
plt.show()
```



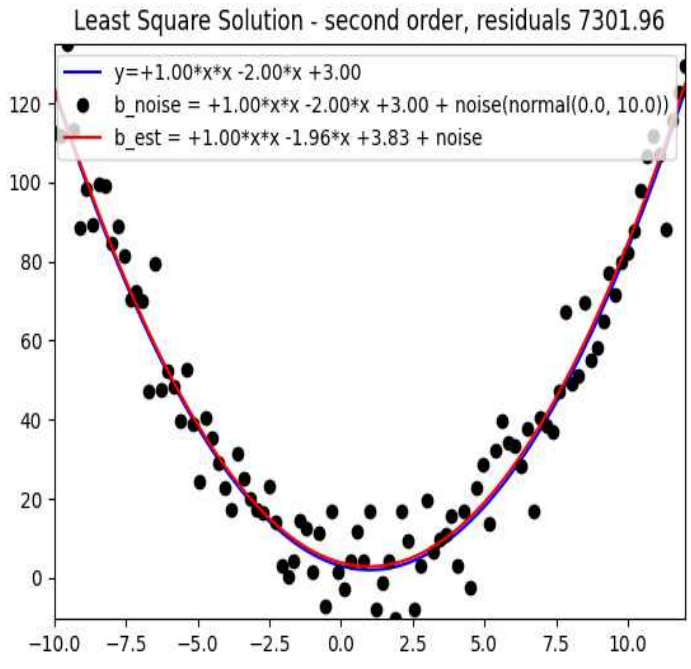
Numpy linalg.lstsq() - 1차 방정식

```
# Least square solution - 1st order
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-10.0, 12.0, num=101)
a, b = 2, 3
y = a*X + b
mu, sigma = 0.0, 10.0
noise = np.random.normal(mu, sigma, X.size) # preparation of noise
y_noise = y + noise # Y with noise

# y = Ax
A = np.vstack([X, np.ones(len(X))]).T
print("A =\n", A)

a_est, b_est = np.linalg.lstsq(A, y_noise, rcond=None)[0]
y_est =  a_est*X + b_est
print("a_est = {}, b_est = {}".format(a_est, b_est))

x_min, x_max = X[0], X[-1]
y_min, y_max = np.amin(y_noise), np.amax(y_noise)

plt.axis([x_min, x_max, y_min, y_max])
plt.plot(X, y, "b-", label="y={:.2f}*X {:+.2f}".format(a, b))
plt.plot(X, y_noise, "ko", label="b_noise = {:.2f}*X {:+.2f}
    + noise(normal({}, {}))".format(a, b, mu, sigma))
plt.plot(X, y_est, "r-", label="b_est = {:.2f}*X {:+.2f}
    + noise".format(a_est, b_est))

plt.title("Least Square Solution - 1st order")
plt.legend(loc="best")
plt.show()
```



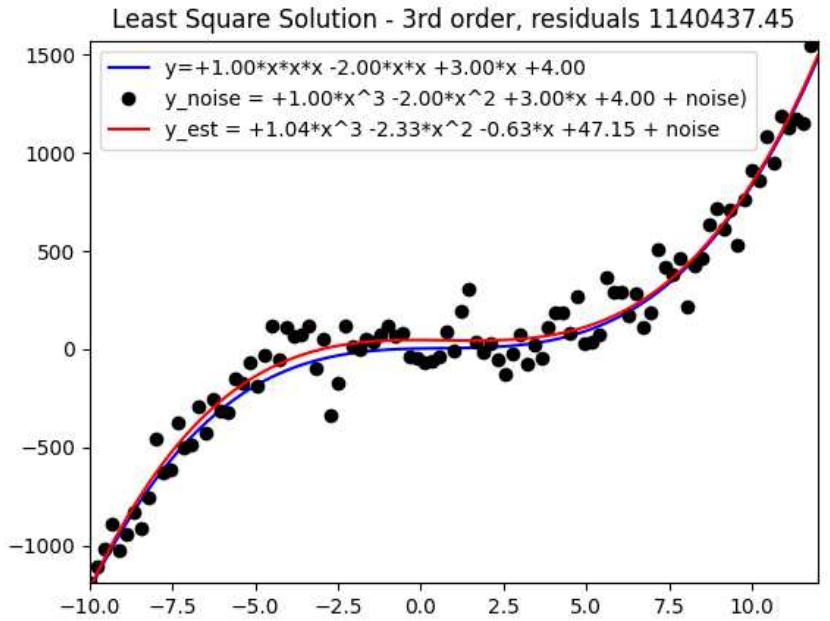
Numpy linalg.lstsq() - 2차 방정식

```
# Least square solution - 2nd order

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-10.0, 12.0, num=101)
a, b, c = 1, -2, 3
B = a*X*X + b*X + c
mu, sigma = 0.0, 10.0
noise = np.random.normal(mu, sigma, X.size) # preparation of noise
B_noise = B + noise # Y with noise
A = np.vstack([X*X, X, np.ones(len(X))]).T
print("A = ", A)

p, residuals, r, s = np.linalg.lstsq(A, B_noise, rcond=None)
a_est, b_est, c_est = p # estimated slop and offset
B_est = a_est*X*X + b_est*X + c_est

x_min, x_max = X[0], X[-1]
y_min, y_max = np.amin(B_noise), np.amax(B_noise)

plt.axis([x_min, x_max, y_min, y_max])
plt.plot(X, B, "b-", label="y={:+.2f}*x*x {:+.2f}*x {:+.2f}".format(a, b, c))
plt.plot(X, B_noise, "ko", label="b_noise = {:+.2f}*x*x {:+.2f}*x {:+.2f} + noise(normal({}, {}))".format(a, b, c, mu, sigma))
plt.plot(X, B_est, "r-", label="b_est = {:+.2f}*x*x {:+.2f}*x {:+.2f} + noise".format(a_est, b_est, c_est))

plt.title("Least Square Solution - second order, residuals {:.2f}".format(residuals[0]))
plt.legend(loc="best")
plt.show()
```



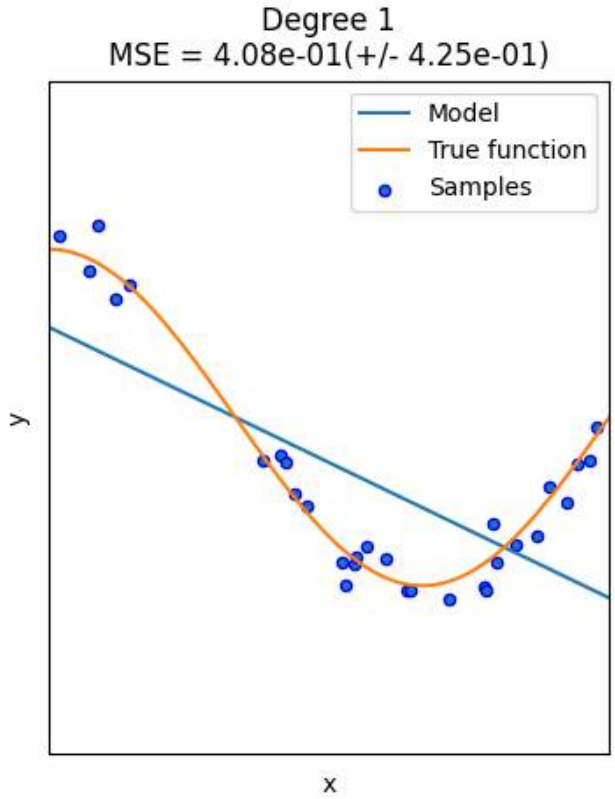
Numpy linalg.lstsq() - 3차 방정식

```
# Least square solution - 3rd order

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-10.0, 12.0, num=101)
a, b, c, d = 1, -2, 3, 4
B = a*X*X*X + b*X*X + c*X + d
mu, sigma = 0.0, 100.0
noise = np.random.normal(mu, sigma, X.size) # preparation of noise
B_noise = B + noise # Y with noise
A = np.vstack([X*X*X, X*X, X, np.ones(len(X))]).T
print("A = ", A)

p, residuals, r, s = np.linalg.lstsq(A, B_noise, rcond=None)
a_est, b_est, c_est, d_est = p # estimated slop and offset
B_est =  a_est*X*X*X + b_est*X*X + c_est*X + d_est

x_min, x_max = X[0], X[-1]
y_min, y_max = np.amin(B_noise), np.amax(B_noise)

plt.axis([x_min, x_max, y_min, y_max])
plt.plot(X, B, "b-", label="y={:+.2f}*x^3 {:+.2f}*x^2 {:+.2f}*x {:+.2f} ".format(a, b, c, d))
plt.plot(X, B_noise, "ko", label="b_noise = {:+.2f}*x^3 {:+.2f}*x^2 {:+.2f}*x {:+.2f}
    + noise(normal({}, {}))".format(a, b, c, d, mu, sigma))
plt.plot(X, B_est, "r-", label="b_est = {:+.2f}*x^3 {:+.2f}*x^2 {:+.2f}*x {:+.2f}
    + noise".format(a_est, b_est, c_est, d_est))

plt.title("Least Square Solution - 3rd order, residuals {:.2f}".format(residuals[0]))
plt.legend(loc="best")
plt.show()
```

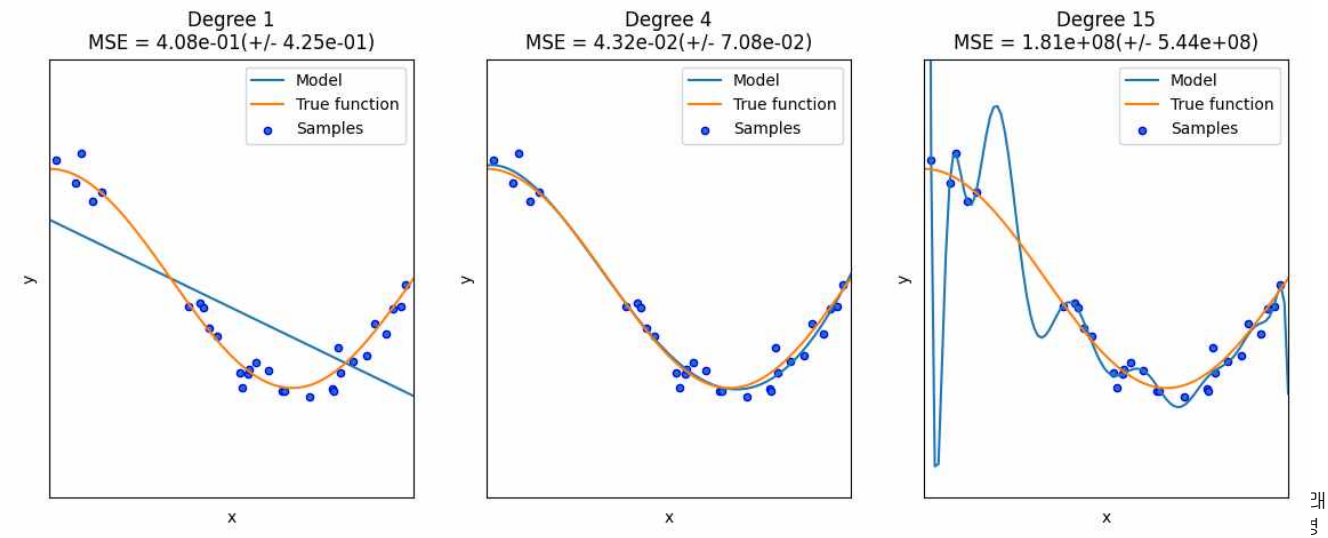


Scikit-learn LinearRegression, PolynomialFeatures, underfit, overfit

```
# Scikit-learn LinearRegression, PolynomialFeatures, underfitting, overfitting (1)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

def true_fun(X):
    return np.cos(1.5 * np.pi * X)

np.random.seed(0)

n_samples = 30
degrees = [1, 4, 15]

X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1

plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline(
        [
            ("polynomial_features", polynomial_features),
            ("linear_regression", linear_regression),
        ]
    )
```



```
# Scikit-learn LinearRegression, PolynomialFeatures, underfitting, overfitting (2)

    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using crossvalidation
    scores = cross_val_score(
        pipeline, X[:, np.newaxis], y, scoring="neg_mean_squared_error", cv=10
    )

    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
    plt.plot(X_test, true_fun(X_test), label="True function")
    plt.scatter(X, y, edgecolor="b", s=20, label="Samples")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title(
        "Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
            degrees[i], -scores.mean(), scores.std()
        )
    )
plt.show()
```



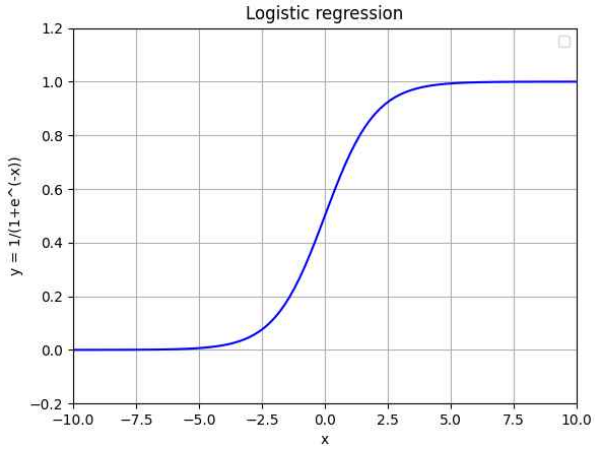
지도학습 기반 분류 (classification)

Scikit-learn Logistic Regression 모델

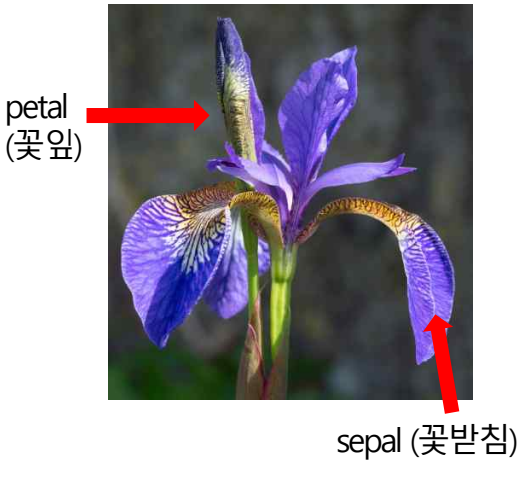
◆ class LogisticRegression

Method	Description
decision_function(X)	Predict confidence scores for samples.
densify()	Convert coefficient matrix to dense array format.
fit(X, y[, sample_weight])	Fit the model according to the given training data.
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict class labels for samples in X.
predict_log_proba(X)	Predict logarithm of probability estimates.
predict_proba(X)	Probability estimates.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.
sparsify()	Convert coefficient matrix to sparse format.

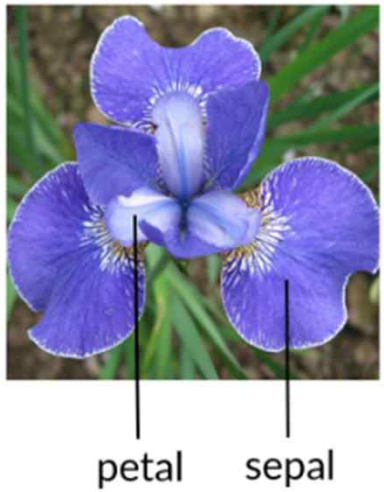
$$g(x) = \frac{\epsilon^x}{1+\epsilon^x} = \frac{1}{1+\epsilon^{-x}}$$



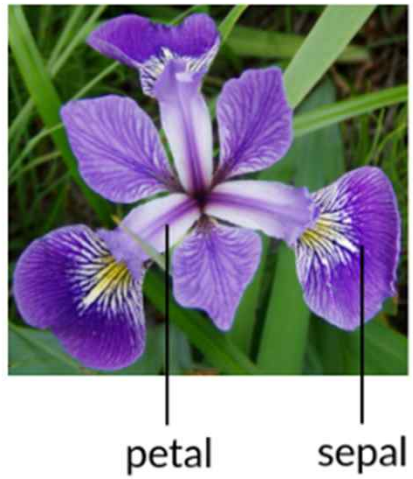
붓꽃 (iris) 꽃받침과 꽃잎 데이터 기반의 분류



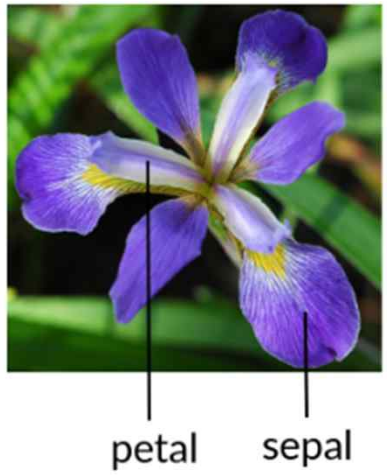
(a) 붓꽃(iris)의 구조



(b) Setosa



(c) Versicolor



(d) Virginica

참고자료: https://www.embedded-robotics.com/iris-dataset-classification/

Logistic Regression

```
# Scikit-learn LogisticRegress, iris classification (1)
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# preparation of dataset, train_set, test_set
dset_iris = load_iris()
print("type(dset_iris) =\n", type(dset_iris))
print("dset_iris['DESCR'] =\n", dset_iris['DESCR'])
print("dset_iris['data'] =\n", dset_iris['data'])
target_class = dset_iris['target']
print("dset_iris['target'] =\n", target_class)

target_names = dset_iris['target_names']
print("dset_iris['target_names'] = ", target_names)

X, y = load_iris(return_X_y=True)
print("Size of X (iris dataset) = ", len(X[:, 0]))
print("X.dimension = ", len(X[0, :]))
X_train, X_test, y_train, y_test = train_test_split(X, y,
   random_state=0, train_size=0.7)
X_train = X_train[:, :5]
X_test = X_test[:, :5]

print("X_train.shape = ", X_train.shape)
print("X_test.shape = ", X_test.shape)
print("Train_X/X, Test_X/X = {}, {}".format(len(X_train)/len(X), len(X_test)/len(X)))
```

```
type(dset_iris) =
 <class 'sklearn.utils._bunch.Bunch'>
dset_iris['DESCR'] =
  Squeezed text (63 lines).
dset_iris['data'] =
  Squeezed text (150 lines).
dset_iris['target'] =
 [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]
dset_iris['target_names'] =  ['setosa' 'versicolor' 'virginica']
Size of X (iris dataset) =  150
X.dimension =  4
X_train.shape =  (105, 4)
X_test.shape =  (45, 4)
Train_X/X, Test_X/X = 0.7, 0.3
lgst_reg.coef_ =  [[-0.39809112  0.83378437 -2.28943476 -0.97838342]
 [ 0.54449829 -0.29042004 -0.2325899  -0.65822871]
 [-0.14640717 -0.54336432  2.52202466  1.63661212]]
lgst_reg.intercept_ =  [  9.00116134   1.54300761 -10.54416895]
lgst_reg.score(X_train, y_train) =  0.9809523809523809
lgst_reg.score(X_test, y_test) =  0.9777777777777777
-------
Predict with X[1]: target_name=setosa
lgst_reg.predict(X[1:2, :5]) = [0] : setosa
lgst_reg.predict_proba(X[1:2, :5]) = [[9.63330755e-01 3.66690000e-02 2.45040789e-07]]
-------
Predict with X[51]: target_name=versicolor
lgst_reg.predict(X[51:52, :5]) = [1] : versicolor
lgst_reg.predict_proba(X[51:52, :5]) = [[0.00726048 0.80871333 0.18402619]]
-------
Predict with X[101]: target_name=virginica
lgst_reg.predict(X[101:102, :5]) = [2] : virginica
lgst_reg.predict_proba(X[101:102, :5]) = [[3.77420991e-04 1.63583189e-01 8.36039390e-01]]
```

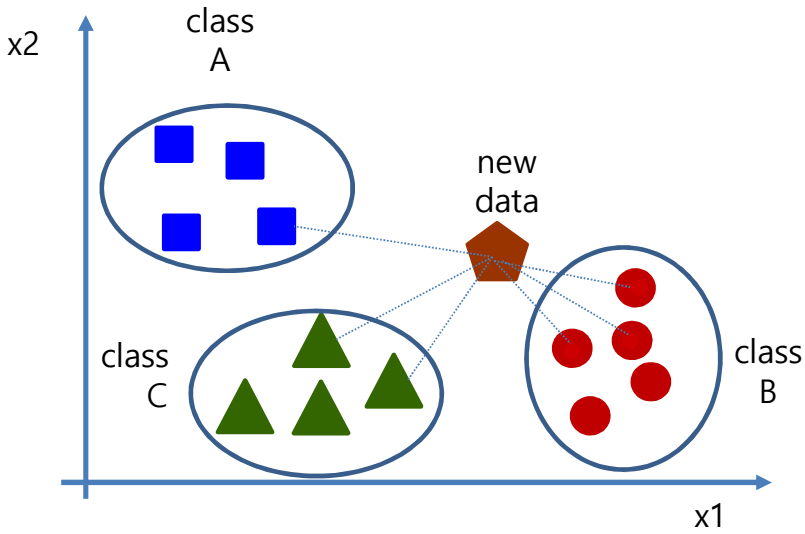
```
# Scikit-learn LogisticRegress, iris classification (2)

# training with LogisticRegresson
lgst_reg = LogisticRegression(random_state=0).fit(X_train, y_train)
print("lgst_reg.coef_ = ", lgst_reg.coef_)
print("lgst_reg.intercept_ = ", lgst_reg.intercept_)
print("lgst_reg.score(X_train, y_train) = ", lgst_reg.score(X_train, y_train))
print("lgst_reg.score(X_test, y_test) = ", lgst_reg.score(X_test, y_test))

# predictions with test cases
index_test_cases = [1, 51, 101]
for index in index_test_cases:
    print("-------")
    print("Predict with X[{}]: target_name={}".format(index,
        target_names[target_class[index]]))
    pred = lgst_reg.predict(X[index:index+1, :5])
    print("lgst_reg.predict(X[{}:{}, :5]) = {} : {}".format(index, index+1,
      lgst_reg.predict(X[index:index+1, :5]), target_names[pred[0]]))
    print("lgst_reg.predict_proba(X[{}:{}, :5]) = {}".format(index, index+1,
      lgst_reg.predict_proba(X[index:index+1, :5])))
```

```
type(dset_iris) =
 <class 'sklearn.utils._bunch.Bunch'>
dset_iris['DESCR'] =
  Squeezed text (63 lines).
dset_iris['data'] =
  Squeezed text (150 lines).
dset_iris['target'] =
 [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]
dset_iris['target_names'] =  ['setosa' 'versicolor' 'virginica']
Size of X (iris dataset) =  150
X.dimension =  4
X_train.shape =  (105, 4)
X_test.shape =  (45, 4)
Train_X/X, Test_X/X = 0.7, 0.3
lgst_reg.coef_ =  [[-0.39809112  0.83378437 -2.28943476 -0.97838342]
 [ 0.54449829 -0.29042004 -0.2325899  -0.65822871]
 [-0.14640717 -0.54336432  2.52202466  1.63661212]]
lgst_reg.intercept_ =  [  9.00116134   1.54300761 -10.54416895]
lgst_reg.score(X_train, y_train) =  0.9809523809523809
lgst_reg.score(X_test, y_test) =  0.9777777777777777
-------
Predict with X[1]: target_name=setosa
lgst_reg.predict(X[1:2, :5]) = [0] : setosa
lgst_reg.predict_proba(X[1:2, :5]) = [[9.63330755e-01 3.66690000e-02 2.45040789e-07]]
-------
Predict with X[51]: target_name=versicolor
lgst_reg.predict(X[51:52, :5]) = [1] : versicolor
lgst_reg.predict_proba(X[51:52, :5]) = [[0.00726048 0.80871333 0.18402619]]
-------
Predict with X[101]: target_name=virginica
lgst_reg.predict(X[101:102, :5]) = [2] : virginica
lgst_reg.predict_proba(X[101:102, :5]) = [[3.77420991e-04 1.63583189e-01 8.36039390e-01]]
```

k-Nearest Neighbor (kNN) 분류 (Classification) 알고리즘

◆ k-Nearest Neighbor (kNN) 알고리즘

- 학습데이터는 특징 공간 (feature space)에 클래스 (class)들로 분류
- 새로운 입력 데이터를 k개의 최근접 이웃 (nearest neighbor)들과 비교하여 어떤 클래스에 속하게 되는가를 결정
- 가장 가까운 k개의 이웃 중에서 가장 많은 표를 얻은 클래스로 분류



```
# k-nearest neighbor classification

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier

# preparation of iris data set
iris = load_iris()
iris_data = iris.data
iris_label = iris.target
print("iris_data =\n", iris_data)
print("iris_label = \n", iris_label)

X_train, X_test, y_train, y_test =\
   train_test_split(iris_data,  # feature data
        iris_label,  #label(answer) data
        test_size=0.2,  # adjust the size of test dataset (0.2=20% of total data)
        random_state=7) # random state in split of train and test data

print("X_train (size = {}) [:5] =\n{}".format(len(X_train), X_train[:5]))
print("y_train (size = {}) =\n{}".format(len(y_train), y_train))
print("X_test (size = {}) [:5]=\n{}".format(len(X_test), X_test[:5]))
print("y_test = ", y_test)

# Model learning and predict
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("y_pred = ", y_pred)
print(classification_report(y_test, y_pred))
```

```
iris_data =
 Squeezed text (150 lines).
iris_label =
 [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 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 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
 2 2]
X_train (size = 120) [:5] =
[[6.2 2.8 4.8 1.8]
 [5.7 2.6 3.5 1. ]
 [4.6 3.6 1.  0.2]
 [6.9 3.1 5.4 2.1]
 [6.4 2.9 4.3 1.3]]
y_train (size = 120) =
[2 1 0 2 1 0 0 0 0 2 2 1 2 2 1 0 1 1 2 0 0 0 2 0 2 1 1 1 0 0 0 1 2 1 1 0 2
 0 0 2 2 0 2 0 1 2 1 0 1 0 2 2 1 0 0 1 2 0 2 2 1 0 1 0 2 2 0 0 2 1 2 2 1 0
 0 2 0 0 1 2 2 1 1 0 2 0 0 1 1 2 0 1 1 2 2 1 2 0 1 1 0 0 0 1 1 0 2 2 1 2 0
 2 1 1 0 2 1 2 1 0]
X_test (size = 30) [:5]=
[[5.9 3.  5.1 1.8]
 [5.4 3.  4.5 1.5]
 [5.  3.5 1.3 0.3]
 [5.6 3.  4.5 1.5]
 [4.9 2.5 4.5 1.7]]
y_test =  [2 1 0 1 2 0 1 1 0 1 1 1 0 2 0 1 2 2 0 0 1 2 1 2 2 2 1 1 2 2]
y_pred =  [2 1 0 1 1 0 1 1 0 1 2 1 0 2 0 1 2 2 0 0 1 2 1 1 2 2 1 1 2 2]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.85      0.92      0.88        12
           2       0.90      0.82      0.86        11

    accuracy                           0.90        30
   macro avg       0.92      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

Decision-tree 기반 분류

```
# Scikit-learn iris classification - DecisionTreeClassifier (1)
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# preparation of iris data set
iris = load_iris()
iris_data = iris.data
iris_label = iris.target
print("iris_data =\n", iris_data)
print("iris_label = \n", iris_label)
# split train data and test data
# X:feature data only / y: correct answer label data only
# input X (feature data) into machine learning model
# learn with comparisons of the estimated label and compare with correct answer y
X_train, X_test, y_train, y_test =\
    train_test_split(iris_data,  # feature data
          iris_label,  #label(answer) data
          test_size=0.2,              # adjust the size of test dataset (0.2=20% of total data)
          random_state=7)      # random state in split of train and test data

print("X_train (size = {}) [:5] =\n{}".format(len(X_train), X_train[:5]))
print("y_train (size = {}) =\n{}".format(len(y_train), y_train))
print("X_test (size = {}) [:5]=\n{}".format(len(X_test), X_test[:5]))
print("y_test = ", y_test)
```

```
iris_data =
 Squeezed text (150 lines).
iris_label =
 [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 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 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 2 2
 2 2]
X_train (size = 120) [:5] =
[[6.2 2.8 4.8 1.8]
 [5.7 2.6 3.5 1. ]
 [4.6 3.6 1.  0.2]
 [6.9 3.1 5.4 2.1]
 [6.4 2.9 4.3 1.3]]
y_train (size = 120) =
[2 1 0 2 1 0 0 0 0 2 2 1 2 2 1 0 1 1 2 0 0 0 2 0 2 1 1 1 0 0 0 1 2 1 1 0 2
 0 0 2 2 0 2 0 1 2 1 0 1 0 2 2 1 0 0 1 2 0 2 2 1 0 1 0 2 2 0 0 2 1 2 2 1 0
 0 2 0 0 1 2 2 1 1 0 2 0 0 1 1 2 0 1 1 2 2 1 2 0 1 1 0 0 0 1 1 0 2 2 1 2 0
 2 1 1 0 2 1 2 1 0]
X_test (size = 30) [:5]=
[[5.9 3.  5.1 1.8]
 [5.4 3.  4.5 1.5]
 [5.  3.5 1.3 0.3]
 [5.6 3.  4.5 1.5]
 [4.9 2.5 4.5 1.7]]
y_test =  [2 1 0 1 2 0 1 1 0 1 1 1 0 2 0 1 2 2 0 0 1 2 1 2 2 2 1 1 2 2]
y_pred =  [2 1 0 1 2 0 1 1 0 1 2 1 0 2 0 2 2 2 0 0 1 2 1 1 2 2 1 1 2 2]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.91      0.83      0.87        12
           2       0.83      0.91      0.87        11

    accuracy                           0.90        30
   macro avg       0.91      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

Decision-tree 기반 분류

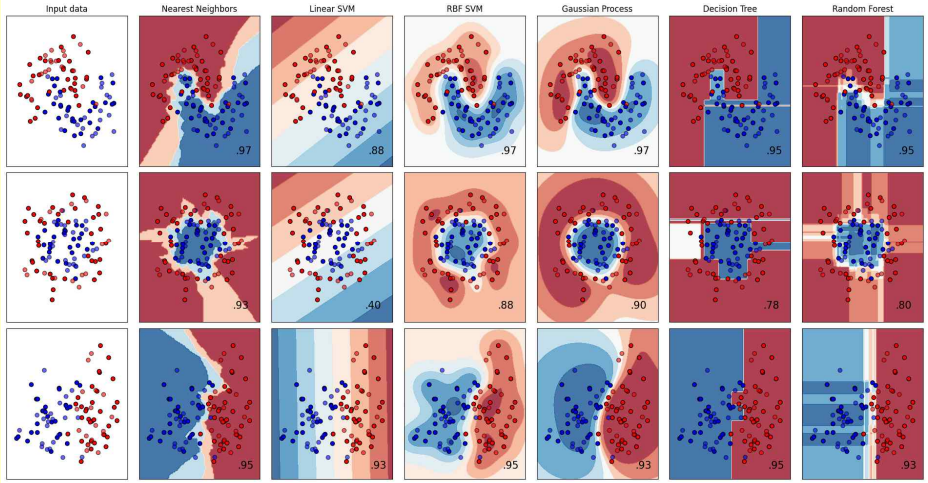
```
# Scikit-learn iris classification - DecisionTreeClassifier (2)

# Model learning and predict
decision_tree = DecisionTreeClassifier(random_state=32)
decision_tree.fit(X_train, y_train)
y_pred = decision_tree.predict(X_test)

print("y_pred = ", y_pred)
print(classification_report(y_test, y_pred))
```

```
iris_data =
  Squeezed text (150 lines).
iris_label =
 [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]
X_train (size = 120) [:5] =
[[6.2 2.8 4.8 1.8]
 [5.7 2.6 3.5 1. ]
 [4.6 3.6 1.  0.2]
 [6.9 3.1 5.4 2.1]
 [6.4 2.9 4.3 1.3]]
y_train (size = 120) =
[2 1 0 2 1 0 0 0 0 2 2 1 2 2 1 0 1 1 2 0 0 0 2 0 2 1 1 1 0 0 0 1 2 1 1 0 2
 0 0 2 2 0 2 0 1 2 1 0 1 0 2 2 1 0 0 1 2 0 2 2 1 0 1 0 2 2 0 0 2 1 2 2 1 0
 0 2 0 0 1 2 2 1 1 0 2 0 0 1 1 2 0 1 1 2 2 1 2 0 1 1 0 0 0 1 1 0 2 2 1 2 0
 2 1 1 0 2 1 2 1 0]
X_test (size = 30) [:5]=
[[5.9 3.  5.1 1.8]
 [5.4 3.  4.5 1.5]
 [5.  3.5 1.3 0.3]
 [5.6 3.  4.5 1.5]
 [4.9 2.5 4.5 1.7]]
y_test =  [2 1 0 1 2 0 1 1 0 1 1 1 0 2 0 1 2 2 0 0 1 2 1 2 2 2 1 1 2 2]
y_pred =  [2 1 0 1 2 0 1 1 0 1 2 1 0 2 0 2 2 2 0 0 1 2 1 1 2 2 1 1 2 2]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.91      0.83      0.87        12
           2       0.83      0.91      0.87        11

    accuracy                           0.90        30
   macro avg       0.91      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

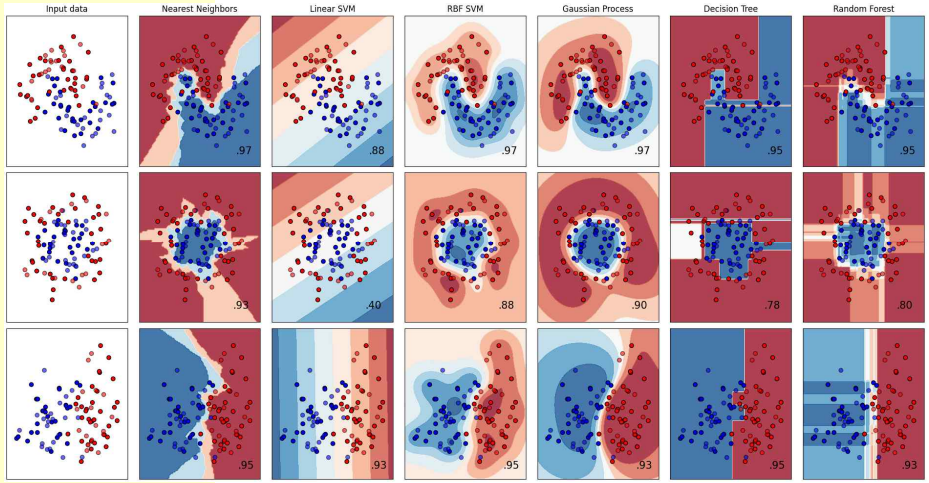
분류 (classification) 기법의 비교

Classifier들의 비교

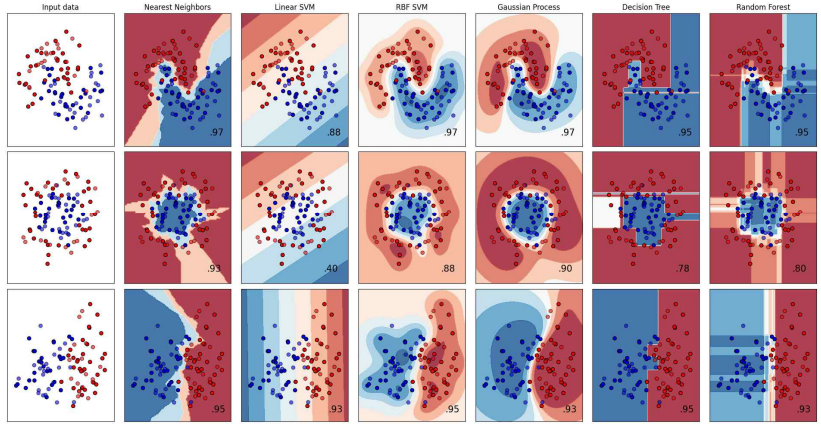
```
# Source: https://scikit-learn.org/stable/auto_examples/classification/  (1)
#    plot_classifier_comparison.html
# Code source: Gaël Varoquaux
#              Andreas Müller
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.inspection import DecisionBoundaryDisplay
```



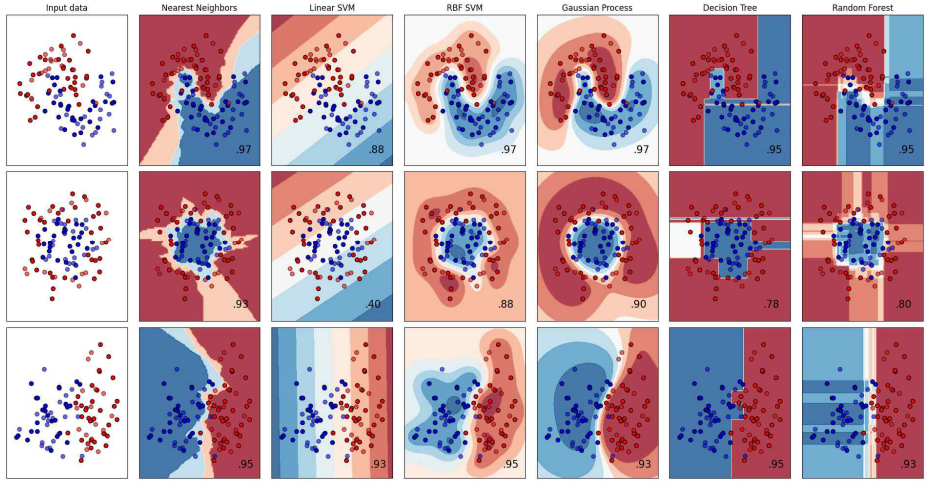
```
# Source: https://scikit-learn.org/stable/auto_examples/classification/  (2)

names = [
    "Nearest Neighbors",
    "Linear SVM",
    "RBF SVM",
    "Gaussian Process",
    "Decision Tree",
    "Random Forest",
]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
]

X, y = make_classification(
    n_features=2, n_redundant=0, n_informative=2, random_state=1, n_clusters_per_class=1
)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [
    make_moons(noise=0.3, random_state=0),
    make_circles(noise=0.2, factor=0.5, random_state=1),
    linearly_separable,
]
```



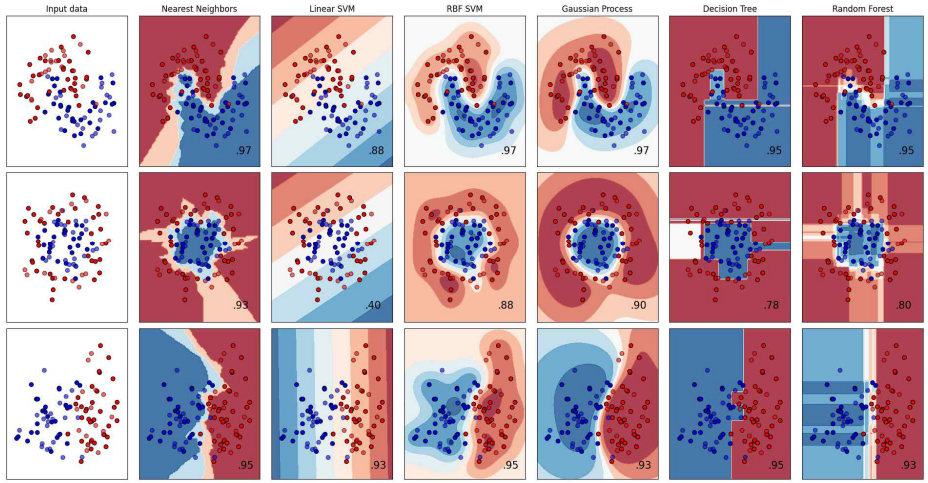
```
# Source: https://scikit-learn.org/stable/auto_examples/classification/  (3)

figure = plt.figure(figsize=(16, 16))
i = 1
# iterate over datasets
for ds_cnt, ds in enumerate(datasets):
    # preprocess dataset, split into training and test part
    X, y = ds
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.4, random_state=42
    )

    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(["#FF0000", "#0000FF"])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    if ds_cnt == 0:
        ax.set_title("Input data")
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors="k")
    # Plot the testing points
    ax.scatter(
        X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6, edgecolors="k"
    )
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1
```



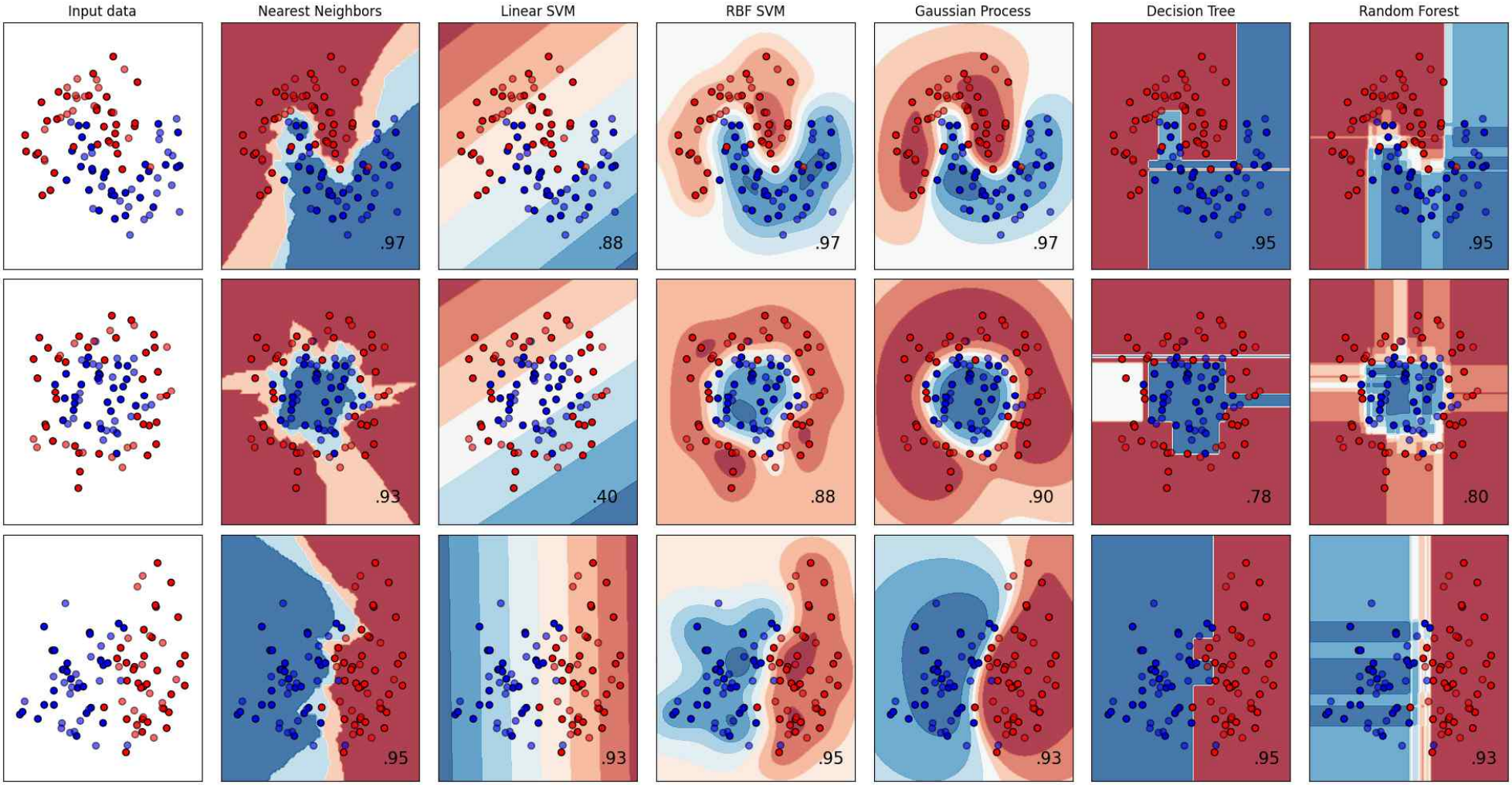
```
# Source: https://scikit-learn.org/stable/auto_examples/classification/  (4)

     # iterate over classifiers
    for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)

        clf = make_pipeline(StandardScaler(), clf)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)
        DecisionBoundaryDisplay.from_estimator(
            clf, X, cmap=cm, alpha=0.8, ax=ax, eps=0.5
        )

        # Plot the training points
        ax.scatter(
            X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors="k"
        )
        # Plot the testing points
        ax.scatter(
            X_test[:, 0],
            X_test[:, 1],
            c=y_test,
            cmap=cm_bright,
            edgecolors="k",
            alpha=0.6,
        )
```



```
# Source: https://scikit-learn.org/stable/auto_examples/classification/  (5)

        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.set_xticks(())
        ax.set_yticks(())
        if ds_cnt == 0:
            ax.set_title(name)
        ax.text(
            x_max - 0.3,
            y_min + 0.3,
            ("%.2f" % score).lstrip("0"),
            size=15,
            horizontalalignment="right",
        )
        i += 1

plt.tight_layout()
plt.show()
```



Classifier들의 비교 결과



비지도학습 클러스터링 (Clustering)

비 지도학습 (Unsupervised Learning)

◆ 비 지도학습 (Unsupervised Learning)

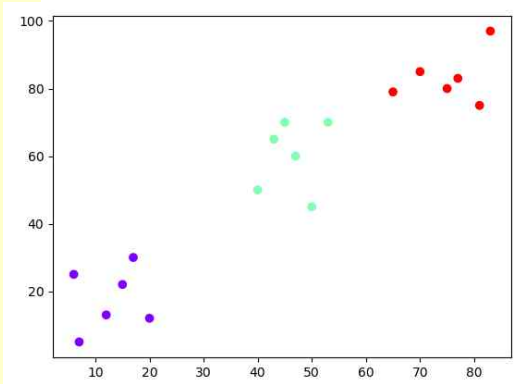
- 어떤 사전 지식도 없이 순수하게 입력 데이터에 대한 변환을 찾아 정보를 도출함
- Data visualization, data compression, data의 noise 제거, data의 상관 관계를 이해하기 위하여 사용됨
- 예: 차원 축소, 군집 (clustering)

K-means clustering

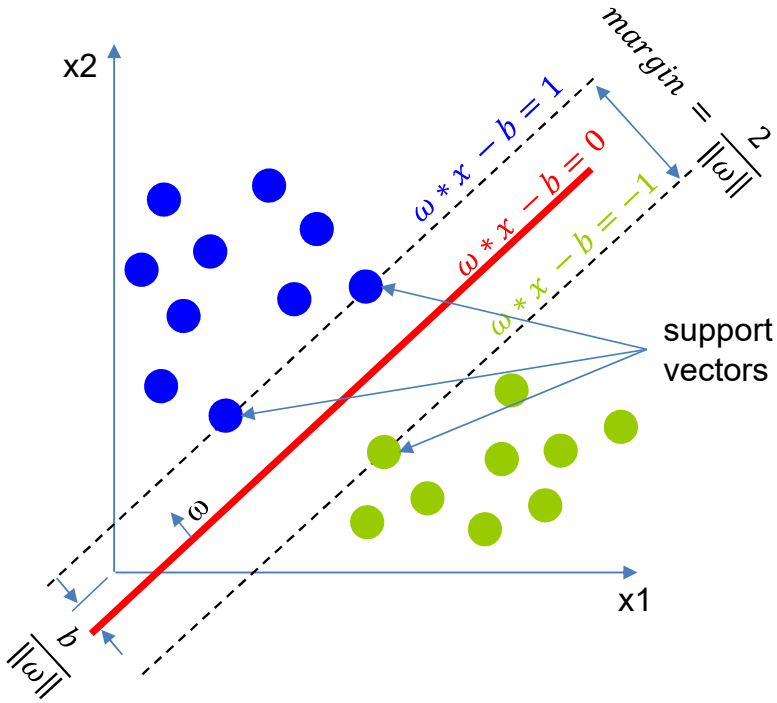
◆ K-means clustering

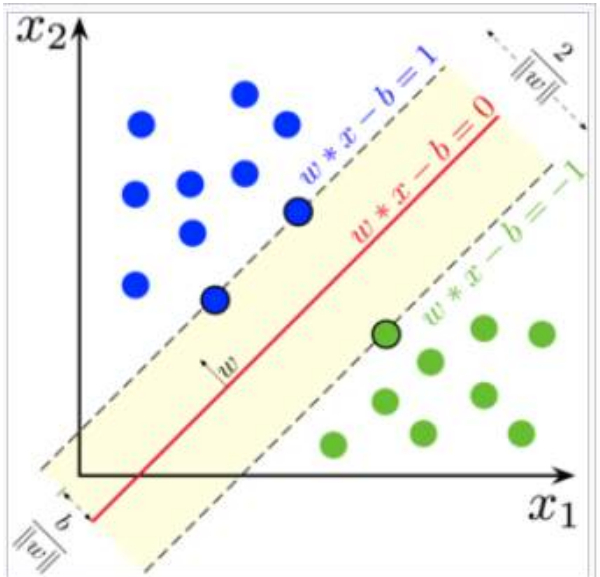
- 비지도 분할 학습
- 주어진 데이터를 k개의 그룹으로 클러스터링
- Sklearn 모듈의 Kmeans() 사용하여 구현

```
# K-means clustering
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

sample_data = np.array([[7, 5], [12, 13], [15, 22], [20, 12], [6, 25], [17, 30],
                        [53, 50], [43, 65], [40, 50], [45, 70], [50, 45], [47, 60],
                        [75, 80], [81, 75], [65, 79], [70, 85], [83, 97], [77, 83]])
#plt.scatter(sample_data[:, 0], sample_data[:, 1])
#plt.show()

kmeans = KMeans(n_clusters = 3)
kmeans.fit(sample_data)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
plt.scatter(sample_data[:, 0], sample_data[:, 1], c=kmeans.labels_, cmap='rainbow')
plt.show()
```



Support Vector Machine (SVM), Support Vector Classifier (SVC)





Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Scikit-learn SVC 기반 iris 분류

```
# Scikit-learn support vector machine (SVM) (1)
import pandas as pd
import numpy as np

from sklearn.svm import SVC, SVR
from sklearn.datasets import load_iris

# load data set iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
#print("iris.target=", iris.target)
df_target = pd.DataFrame(iris.target)
df['species'] = [iris.target_names[x] for x in iris.target]

species_to_labels = dict(zip(df['species'].unique(), range(len(df['species'].unique()))))
df['species'] = df['species'].map(species_to_labels) # convert species to labels
#print("df['species'] =\n", df['species'])

X = df.drop('species', axis=1)
y = df['species']

print("X =\n", X)
print("y =\n", y)
```

```
X =
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
y =
 0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: species, Length: 150, dtype: int64
SVC with linear kernel
intercept : [ 1.4528445   1.50771313 13.63764975]
precision : 0.98
SVC with rbf kernel
intercept : [ 0.08537971 -0.12101929 -0.14515776]
precision : 0.9866666666666667
SVC with poly kernel
intercept : [1.13460386 1.18408165 4.13277592]
precision : 0.98
SVC with sigmoid kernel
intercept : [3.64873385 3.66625196 0.33851057]
precision : 0.04
```

Scikit-learn SVC 기반 iris 분류

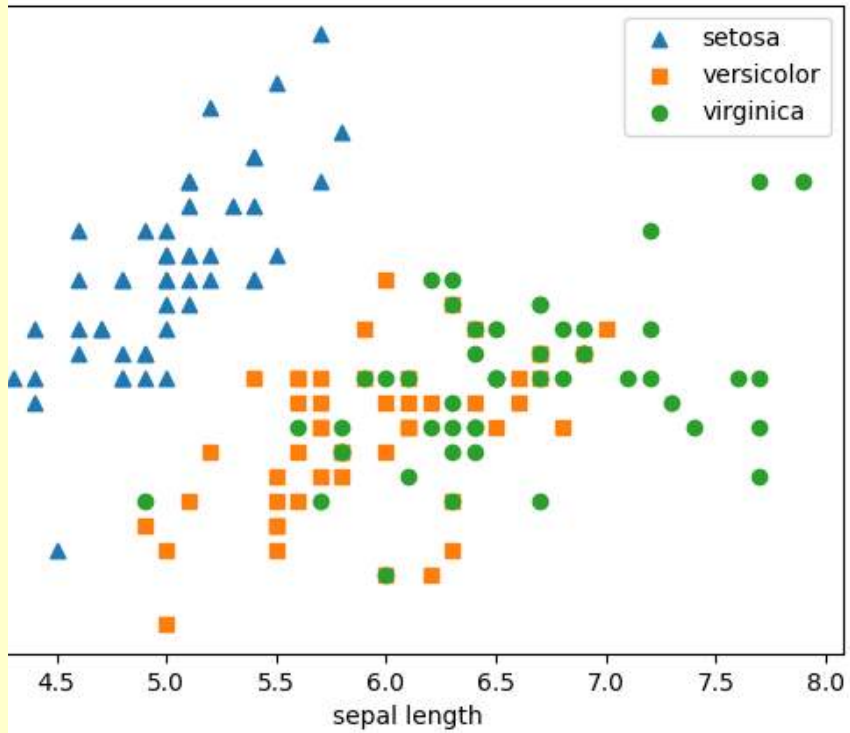
```
# Scikit-learn support vector machine (SVM) (2)

kernel_list = ['linear', 'rbf', 'poly', 'sigmoid']

for kernel in kernel_list:
    print('SVC with {} kernel'.format(kernel))
    clf = SVC(C=10, kernel=kernel, random_state=100) # create SVC classifier
    clf.fit(X, y) # model learning/training
    # Parameters
    print('intercept :', clf.intercept_)
    print('precision :', np.mean(y==clf.predict(X)))
```

```
X =
      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                   5.1               3.5                1.4               0.2
1                   4.9               3.0                1.4               0.2
2                   4.7               3.2                1.3               0.2
3                   4.6               3.1                1.5               0.2
4                   5.0               3.6                1.4               0.2
..                  ...               ...                ...               ...
145                 6.7               3.0                5.2               2.3
146                 6.3               2.5                5.0               1.9
147                 6.5               3.0                5.2               2.0
148                 6.2               3.4                5.4               2.3
149                 5.9               3.0                5.1               1.8

[150 rows x 4 columns]
y =
 0      0
1       0
2       0
3       0
4       0
       ..
145     2
146     2
147     2
148     2
149     2
Name: species, Length: 150, dtype: int64
SVC with linear kernel
intercept : [ 1.4528445   1.50771313 13.63764975]
precision : 0.98
SVC with rbf kernel
intercept : [ 0.08537971 -0.12101929 -0.14515776]
precision : 0.9866666666666667
SVC with poly kernel
intercept : [1.13460386 1.18408165 4.13277592]
precision : 0.98
SVC with sigmoid kernel
intercept : [3.64873385 3.66625196 0.33851057]
precision : 0.04
```

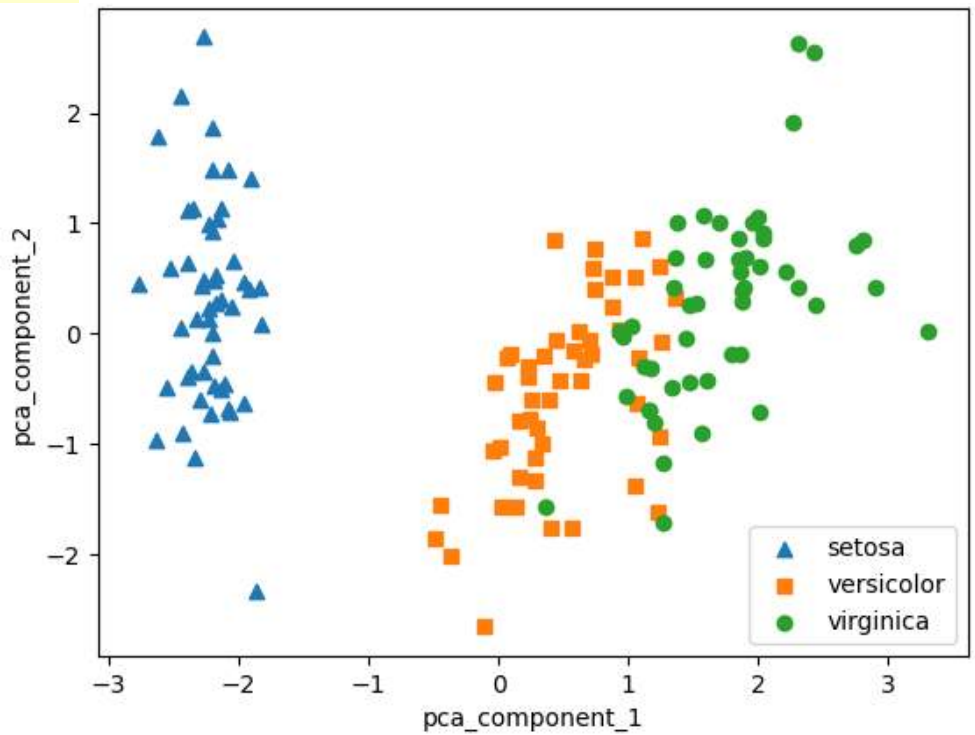
PCA 기반 차원축소 (Dimension Reduction) - iris

```
# dimension reduction - iris

from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
#matplotlib inline

iris=load_iris()
columns=['sepal_length','sepal_width','petal_length','petal_width']
df_iris=pd.DataFrame(iris.data,columns=columns)
df_iris['target']=iris.target
df_iris.head()

# markers: (setosa: triangle), (versicolor: square), (virginica: circle)
markers=['^','s','o']

for i, marker in enumerate(markers):
    x_axis_data=df_iris[df_iris['target']==i]['sepal_length']
    y_axis_data=df_iris[df_iris['target']==i]['sepal_width']
    plt.scatter(x_axis_data,y_axis_data,marker=marker,label=iris.target_names[i])

plt.legend()
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.show()
```

참고자료: **https://casa-de-feel.tistory.com/19**



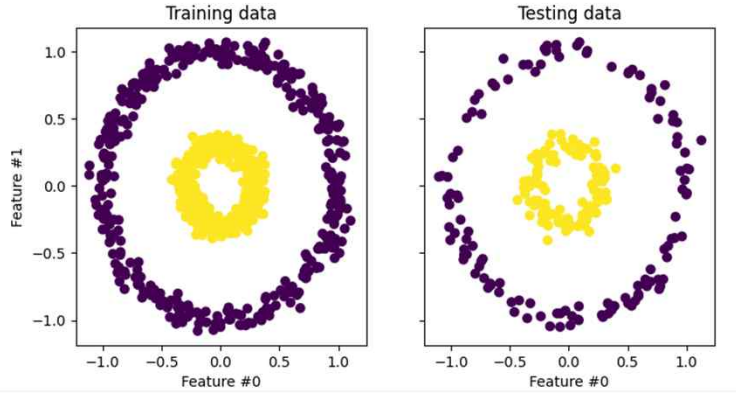
```
# dimension reduction by PCA, iris (1)
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

iris=load_iris()
columns=['sepal_length','sepal_width','petal_length','petal_width']
df_iris=pd.DataFrame(iris.data,columns=columns)
df_iris['target']=iris.target
df_iris.head()

scaler=StandardScaler()
iris_scaled=scaler.fit_transform(df_iris.iloc[:,:-1])

# Reduce to 2-dimensional
pca=PCA(n_components=2)
pca.fit(iris_scaled)
iris_pca=pca.transform(iris_scaled)

pca_columns=['pca_component_1','pca_component_2']
df_iris_pca=pd.DataFrame(iris_pca,columns=pca_columns)
df_iris_pca['target']=iris.target
df_iris_pca.head()

markers=['^','s','o']
for i, marker in enumerate(markers):
    x_axis_data=df_iris_pca[df_iris_pca['target']==i]['pca_component_1']
    y_axis_data=df_iris_pca[df_iris_pca['target']==i]['pca_component_2']
    plt.scatter(x_axis_data,y_axis_data,marker=marker,label=iris.target_names[i])

plt.legend()
plt.xlabel('pca_component_1')
plt.ylabel('pca_component_2')
plt.show()
```

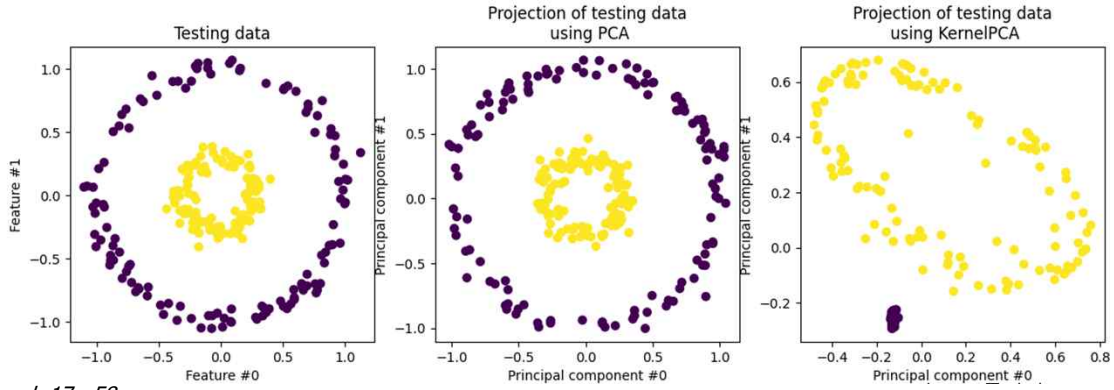


PCA (Principal Component Analysis)

```
# PCA and Kernel PCA (1)
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA, KernelPCA

X, y = make_circles(n_samples=1_000, factor=0.3, noise=0.05, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=0)

_, (train_ax, test_ax) = plt.subplots(ncols=2, sharex=True, sharey=True, figsize=(8, 4))

train_ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
train_ax.set_ylabel("Feature #1")
train_ax.set_xlabel("Feature #0")
train_ax.set_title("Training data")

test_ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
test_ax.set_xlabel("Feature #0")
_ = test_ax.set_title("Testing data")

pca = PCA(n_components=2)
kernel_pca = KernelPCA(n_components=None, kernel="rbf", gamma=10, fit_inverse_transform=True, alpha=0.1)

X_test_pca = pca.fit(X_train).transform(X_test)
X_test_kernel_pca = kernel_pca.fit(X_train).transform(X_test)

fig, (orig_data_ax, pca_proj_ax, kernel_pca_proj_ax) = plt.subplots(ncols=3, figsize=(14, 4))
```

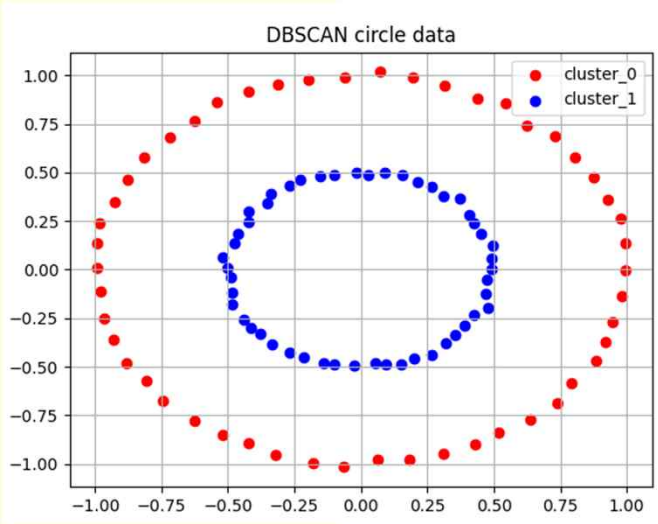
```
# PCA and Kernel PCA (2)

orig_data_ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
orig_data_ax.set_ylabel("Feature #1")
orig_data_ax.set_xlabel("Feature #0")
orig_data_ax.set_title("Testing data")

pca_proj_ax.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test)
pca_proj_ax.set_ylabel("Principal component #1")
pca_proj_ax.set_xlabel("Principal component #0")
pca_proj_ax.set_title("Projection of testing data\n using PCA")

kernel_pca_proj_ax.scatter(X_test_kernel_pca[:, 0], X_test_kernel_pca[:, 1], c=y_test)
kernel_pca_proj_ax.set_ylabel("Principal component #1")
kernel_pca_proj_ax.set_xlabel("Principal component #0")
_ = kernel_pca_proj_ax.set_title("Projection of testing data\n using KernelPCA")

plt.show()
```





원형 분포를 가지는 데이터에 대한 Clustering - DBSCAN

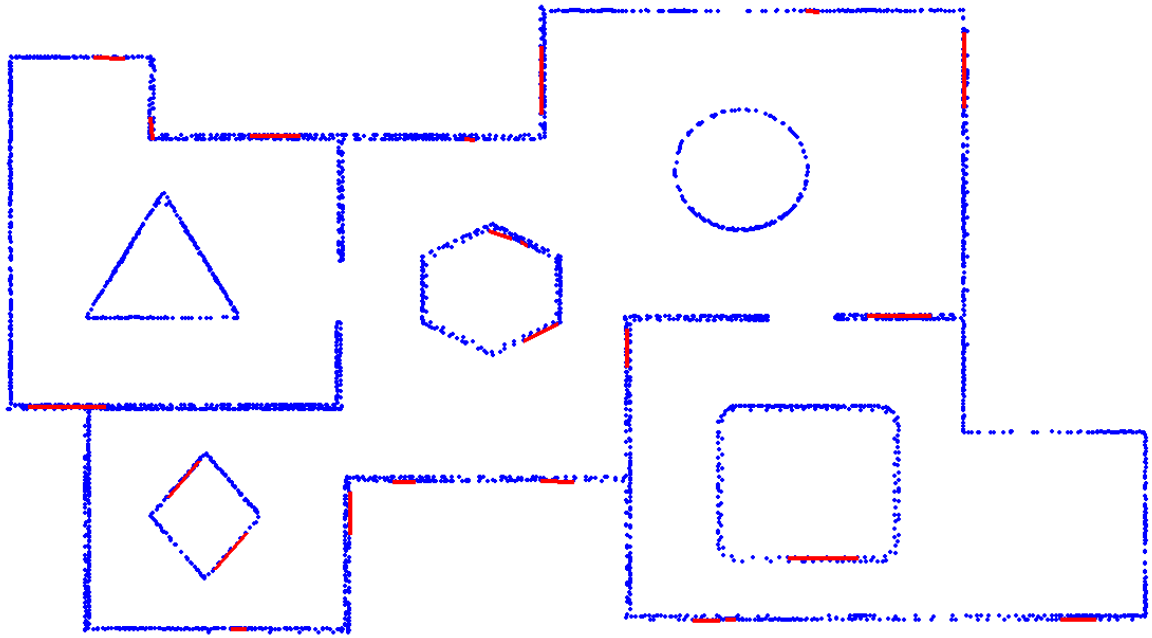
```
# Clustering with DBSCAN (density-based spatial clustering of applications with noise)

from sklearn.cluster import DBSCAN
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt

fig = plt.figure()
ax= fig.add_subplot(1, 1, 1)
color_dict = {0: 'red', 1: 'blue', 2: 'green', 3:'black',4:'yellow'}
# color dictionary of data color setting for n-th cluster data

circle_points, circle_labels = make_circles(n_samples=100, factor=0.5, noise=0.01)
# generates 100 points in circle distribution

epsilon, minPts = 0.2, 3 # initial epsilon, minPts
circle_dbscan = DBSCAN(eps=epsilon, min_samples=minPts) # DBSCAN setting
circle_dbscan.fit(circle_points)
n_cluster = max(circle_dbscan.labels_)+1

print(f'# of cluster: {n_cluster}')
print(f'DBSCAN Y-hat: {circle_dbscan.labels_}')

for cluster in range(n_cluster):
    cluster_sub_points = circle_points[circle_dbscan.labels_ == cluster]
    ax.scatter(cluster_sub_points[:, 0], cluster_sub_points[:, 1], c=color_dict[cluster], label='cluster_{}'.format(cluster))
ax.set_title('DBSCAN circle data')

ax.legend()
ax.grid()
plt.show()
```



군집화(clustering)의 응용 예

◆ 군집화(clustering)의 응용 예

- 자율주행 자동차의 차선 인식
- 자율주행 자동차의 LiDAR 센서 데이터를 사용한 장애물 인식
- 자율주행 청소기/로봇의 LiDAR 센서 데이터를 사용한 벽/통로, 장애물 종류 인식



Homework 17

Homework 17

17.1 K-means clustering 기능 구현

- 최근 10년간 기상청 기온 측정 데이터에서 서울, 인천, 강릉, 대전, 광주, 대구, 부산, 제주 8 지역의 월별 평균 기온 데이터를 정리하라. (8개 지역 x 12월 x 10년 평균 기온)의 데이터를 준비하라.
- 각 지역별 월별 평균 데이터 산출에서는 8장에서 배운 pandas 기반 데이터 분석 기법을 사용할 것.
- sklearn 모듈의 K-means를 사용하여 주어진 데이터를 K개의 cluster로 구성하는 프로그램을 강의 자료를 참조하여 구현하라.
- 위에서 준비한 8개 지역의 월평균 기온에 대하여 clustering (K: 3 ~ 6)을 하여 어떤 정보를 파악할 수 있는지 분석하라.

References

<Machine Learning>

[1] Aurelien Geron, Hands-on Machine Learning with Scikit-Learning, Keras & TensorFlow, O'Reilly, 2019.

[2] 사이킷런 scikit-learn 제대로 시작하기, https://www.youtube.com/watch?v=eVxGhCRN-xA.