Popular Modules for Data Analytics

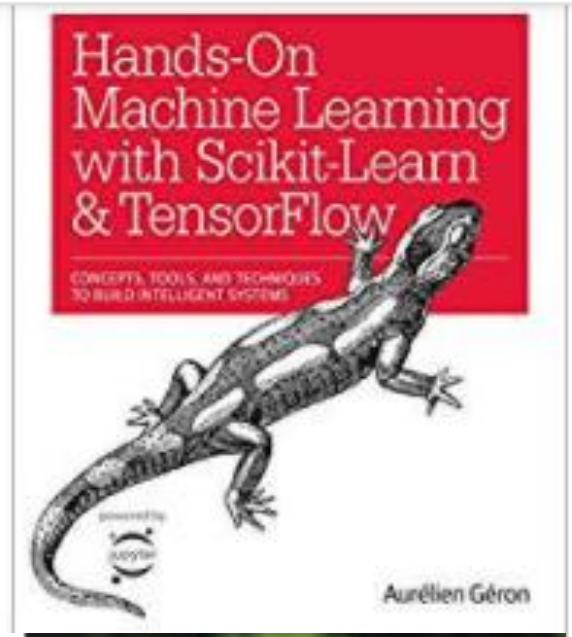
- **(Ch 30) Numpy**
- **(Ch 31) Pandas**
- **(Ch 32) Matplotlib, Seaborn**
- **(Ch 33) SciPy**
- **(Ch 34) Scikit learn**

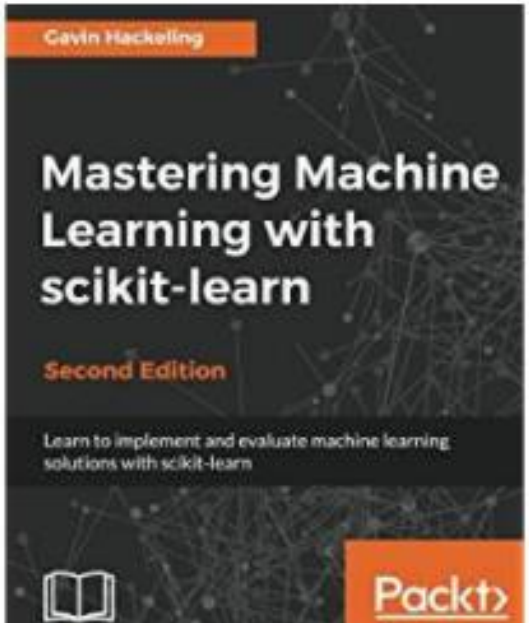
(Ch 34) Scikit Learn Module

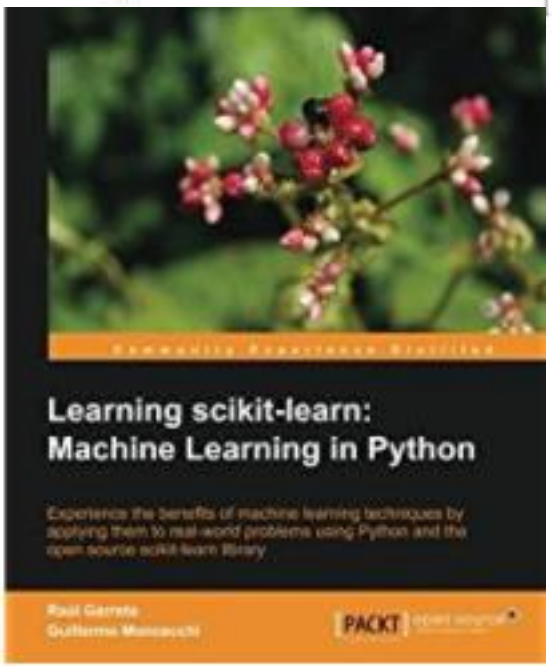
Table of Contents

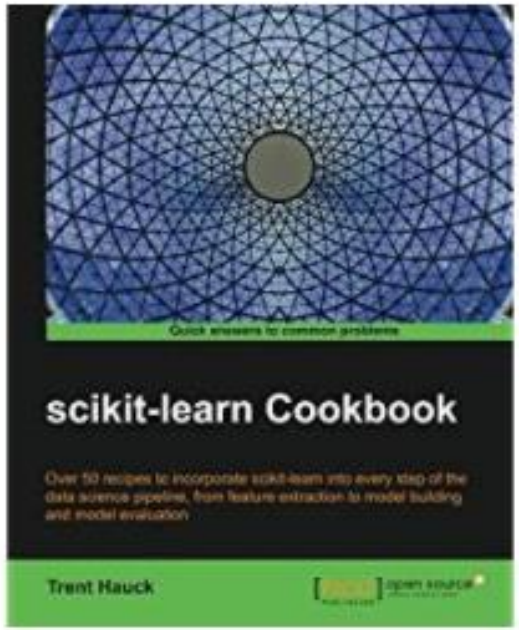
- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbor (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

Many SciKit-Learn Books









What is “sklearn” module?

- Scikit (SciPy Toolkit) –learn, SK-Learn
- (Open Source) Machine Learning Library for Python
- Built on top of SciPy
  - Designed to interoperate with Python numerical and scientific library
- Dependency
  - NumPy
  - SciPy
  - Matplotlib (for running examples)

- Open source (http://scikit-learn.org)
  - Initially developed by David Cournapeau as a “Google Summer of Code” project in 2007
  - Still under active development (V 0.19.0 as of August, 2017)

참고

“sklearn” Installation guide [1/3]

1. http://www.lfd.uci.edu/~gohlke/pythonlibs/ 에서 PC 환경에 맞는 Numpy+MKL와 Scipy whl 파일 다운로드

NumPy, a fundamental package needed for scientific c

Numpy+MKL is linked to the Intel® Math Kernel

numpy-1.11.3+mkl-cp27-cp27m-win32.whl
numpy-1.11.3+mkl-cp27-cp27m-win_amd64.whl
numpy-1.11.3+mkl-cp34-cp34m-win32.whl
numpy-1.11.3+mkl-cp34-cp34m-win_amd64.whl
numpy-1.11.3+mkl-cp35-cp35m-win32.whl
numpy-1.11.3+mkl-cp35-cp35m-win_amd64.whl
numpy-1.11.3+mkl-cp36-cp36m-win32.whl
numpy-1.11.3+mkl-cp36-cp36m-win_amd64.whl
numpy-1.13.0+mkl-cp27-cp27m-win32.whl
numpy-1.13.0+mkl-cp27-cp27m-win_amd64.whl
numpy-1.13.0+mkl-cp34-cp34m-win32.whl
numpy-1.13.0+mkl-cp34-cp34m-win_amd64.whl
numpy-1.13.0+mkl-cp35-cp35m-win32.whl
numpy-1.13.0+mkl-cp35-cp35m-win_amd64.whl
numpy-1.13.0+mkl-cp36-cp36m-win32.whl
numpy-1.13.0+mkl-cp36-cp36m-win_amd64.whl

SciPy is software for mathematics, science, and engineering.

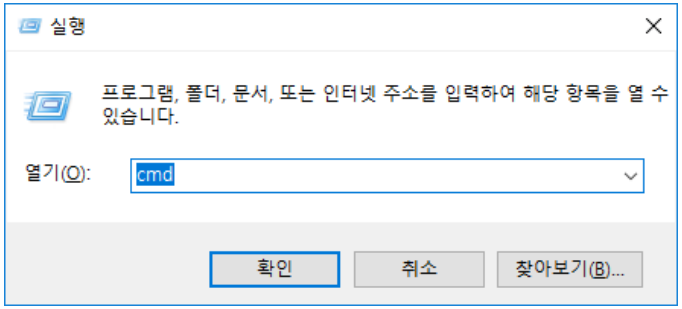
Install numpy+mkl before installing scipy.

scipy-0.19.1-cp27-cp27m-win32.whl
scipy-0.19.1-cp27-cp27m-win_amd64.whl
scipy-0.19.1-cp34-cp34m-win32.whl
scipy-0.19.1-cp34-cp34m-win_amd64.whl
scipy-0.19.1-cp35-cp35m-win32.whl
scipy-0.19.1-cp35-cp35m-win_amd64.whl
scipy-0.19.1-cp36-cp36m-win32.whl
scipy-0.19.1-cp36-cp36m-win_amd64.whl

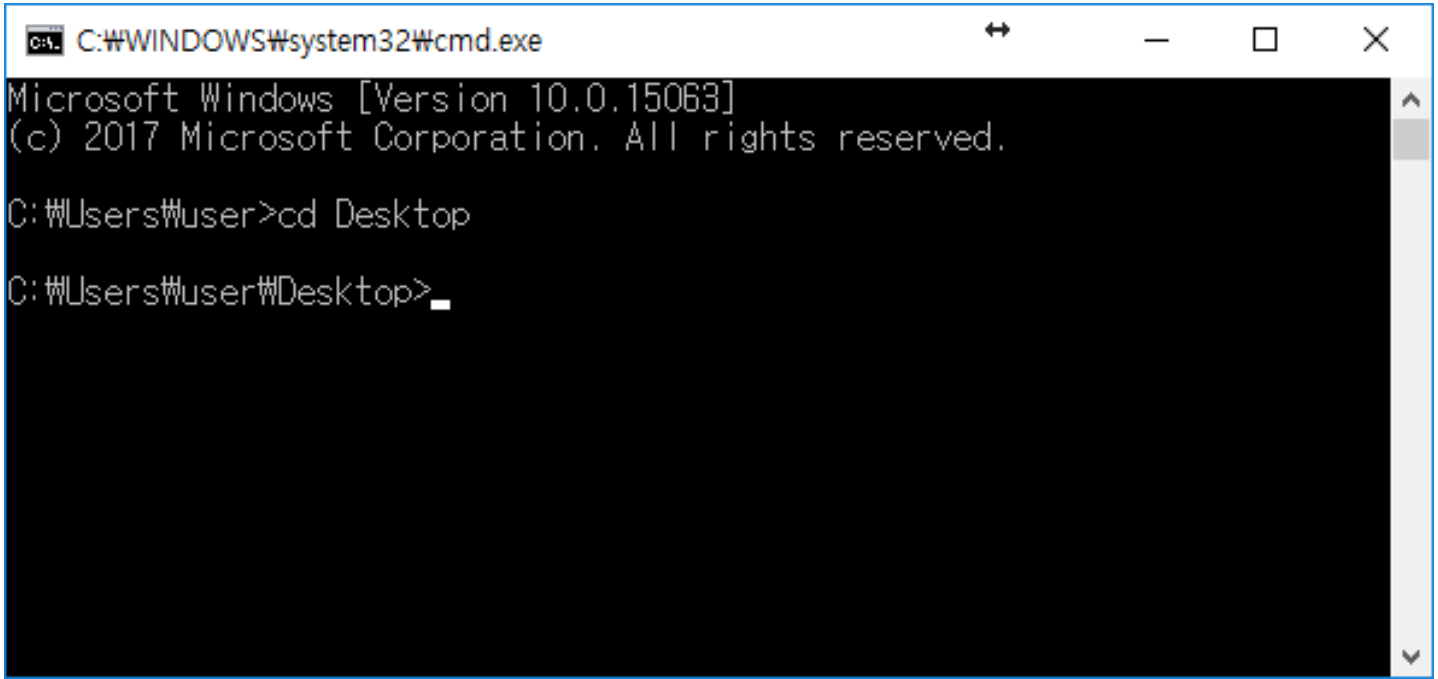
참고

“sklearn” Installation guide [2/3]

2. 명령 프롬프트 실행: “윈도우 키” + R → cmd 실행



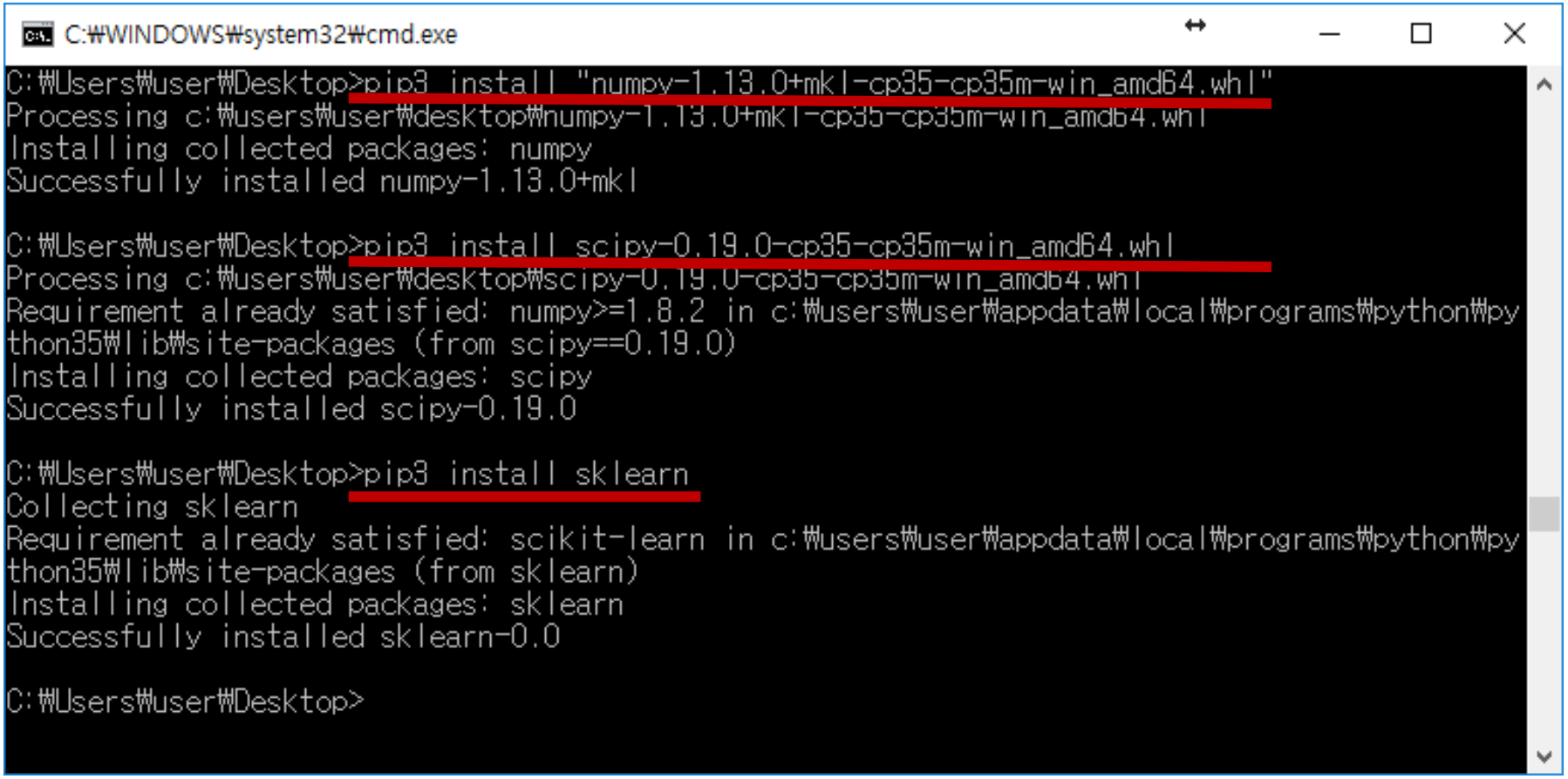
3. whl 파일 저장 경로로 이동: cd + “디렉토리 경로”



참고

“sklearn” Installation guide [3/3]

4. **[pip install *.whl]**: Numpy, Scipy 설치
5. **[pip install sklearn]**: sklearn 설치



“sklearn” module: Contents

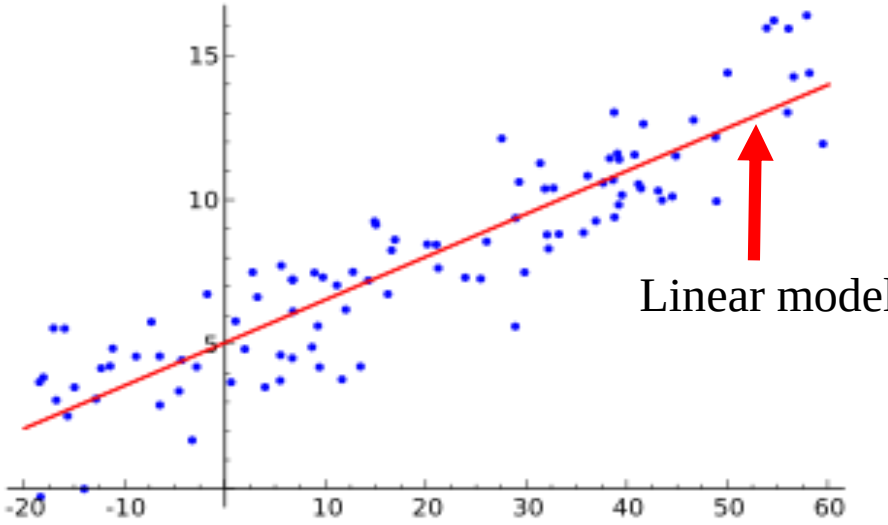
- Classification: identify to which category an object belongs to
  - Regression: predict continuous-valued attribute
    - Linear regression, logistic regression, etc …
  - SVM, Decision tree, Neural Nets, Nearest neighbors, …
- Clustering: grouping of similar objects
  - k-means, hierarchical clustering, etc …
- Model selection: validate and choosing parameters and model
  - Cross validation, metrics, etc …
- Preprocessing: feature extraction & normalization
  - Feature extraction, etc …
- Dimensionality reduction: reducing number of variables
  - PCA, feature selection, etc …
- Datasets

Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbor (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

Regression

- Finding an equation which explains the data
  - Explain ←→ Predict
- Started from 1800s
  - Legendre 1805, Gauss 1809
- Various regression models
  - Linear regression
  - Non-linear regression
  - Logistic regression



Linear Regression Concept [1/2]

- Modeling relationship between continuous dependent variable **y** and one or more independent variables **X** using linear predictor function

$$\mathbf{Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n}$$

$\mathbf{x_n}$: arbitrary input, independent variable

Y: output based on $\mathbf{x_n}$, dependent variable

β: coefficients for accurate predictor function

$\boldsymbol{\beta_0}$: intercept

- **Linear Regression Classifier**

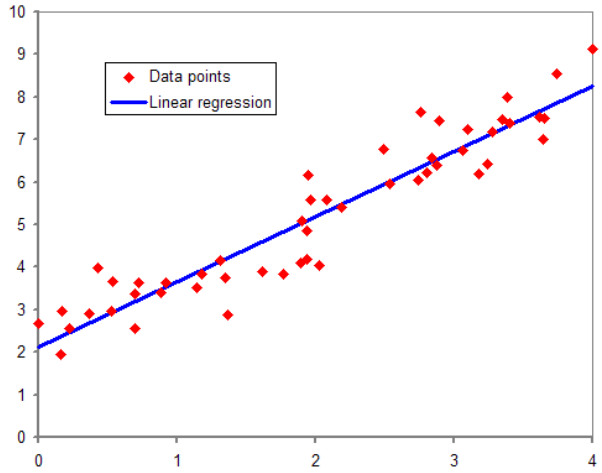
Linear Regression Concept [2/2]

- Find the linear line minimizing distance from all points
- For new data with x values and unknown Y, predict Y with the linear predictor function
- Logistic regression for classification (returns class number)

X	Y
0.5	2
0.7	2.5
1.2	3.4
…	….
3.6	7.5
3.8	8.2
4	9

$$Y = \beta_0 + \beta_1 X$$

New X	Predict Y
3.1	?



x1	x2	x3	y
0.3	0.4	2.44	3.2
1.2	1.67	2.22	2.8
…	….	….	
1.1	1.9	3.8	2.9

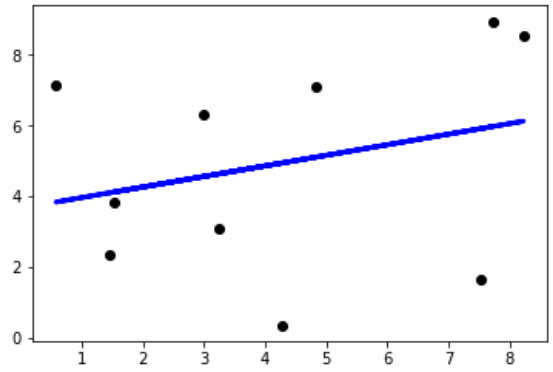
$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

Simple Linear Regression using Python (10x1) 2D List

```
 9 from random import *
10 import numpy as np
11 from sklearn import linear_model
12 import matplotlib.pyplot as plt
13
14 x = []
15 y = []
16 for i in range(10):
17     x.append([round((random( )*10), 2)])
18     y.append( round((random( )*10), 2) )
19 print("Python 2D List x: \n",  x)
20 print("Python 1D List y: \n",  y)
21
22 regr = linear_model.LinearRegression( )
23 regr.fit(x, y)
24
25 plt.scatter(x, y, color="black")
26 plt.plot(x, regr.predict(x), color="blue", linewidth=3)
27 plt.show()
```

regr.fit(x, y)
- x 는 Python 2D array
- y 는 Python 1D array
- x & y 는 same length

regr.predict(x)
- x 는 Python 2D array




```
Python 2D List x:
 [[4.84], [2.99], [0.58], [4.27], [3.24], [1.54], [7.73], [1.47], [8.23], [7.52]]
Python 1D List y:
 [7.1, 6.32, 7.13, 0.36, 3.11, 3.84, 8.95, 2.33, 8.52, 1.66]
```

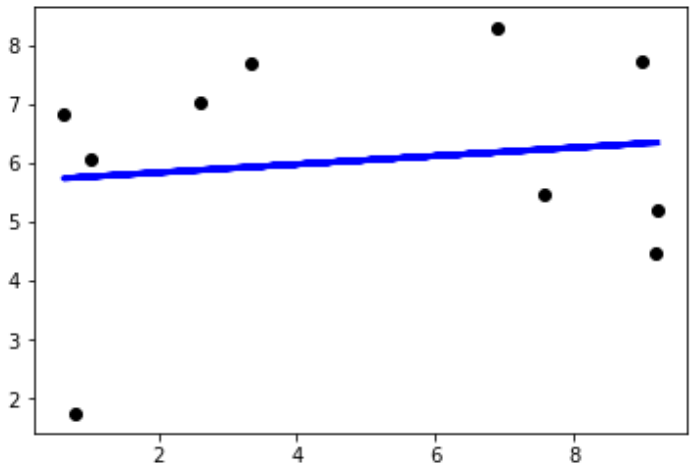
Simple Linear Regression using Numpy (10x1) 2D Array

```
from random import *
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

x = []
y = []
for i in range(10):
    x.append([round((random( )*10), 2)])
    y.append( round((random( )*10), 2) )
x = np.array(x)
y = np.array(y)
print("Numpy 2D Array x: \n",  x)
print("Numpy 1D Array y: \n",  y)

regr = linear_model.LinearRegression( )
regr.fit(x, y)

plt.scatter(x, y, color="black")
plt.plot(x, regr.predict(x), color="blue", linewidth=3)
plt.show()
```

regr.fit(x, y)
- x 는 Numpy 2D array
- y 는 Numpy 1D array
- x & y 는 same length

regr.predict(x)
- x 는 Numpy 2D array

```
Numpy 2D Array x:
 [[9.19]
 [3.34]
 [6.9 ]
 [0.8 ]
 [7.59]
 [8.99]
 [0.63]
 [1.01]
 [2.6 ]
 [9.21]]
Numpy 1D Array y:
 [4.46 7.7  8.3  1.76 5.45 7.72 6.83 6.05 7.02 5.19]
```



Simple Linear Regression using Pandas

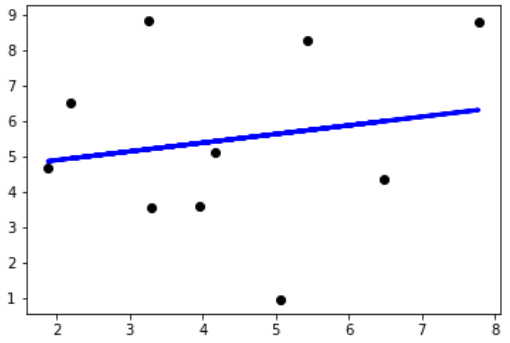
```
from random import *
import numpy as np
import pandas as pd
from sklearn import linear_model
import matplotlib.pyplot as plt

x = []
y = []
for i in range(10):
    x.append([round((random( )*10), 2)])
    y.append( round((random( )*10), 2) )
p_data_x = pd.DataFrame(x)
p_data_y = pd.Series(y)

print("Pandas DataFrame x: \n",  p_data_x)
print("Pandas Series y: \n",  p_data_y)

regr = linear_model.LinearRegression( )
regr.fit(p_data_x, p_data_y)

plt.scatter(p_data_x, p_data_y,  color="black")
plt.plot(p_data_x, regr.predict(p_data_x), color="blue", linewidth=3)
plt.show()
```

regr.fit(x, y)
- x 는 Pandas DataFrame
- y 는 Pandas Series
- x & y 는 same length

regr.predict(x)
- x 는 Pandas DataFrame

```
Pandas DataFrame x:
      0
0  3.26
1  1.88
2  7.78
3  3.95
4  3.30
5  5.07
6  6.48
7  5.44
8  2.18
9  4.17
```

```
Pandas Series y:
 0    8.86
1    4.69
2    8.80
3    3.59
4    3.57
5    0.98
6    4.37
7    8.28
8    6.52
9    5.13
dtype: float64
```



Linear Regression Class (at linear_model submodule)

sklearn.linear_model.LinearRegression (*fit_intercept*=True, *normalize*=False, *copy_X*=True, *n_jobs*=1)

4 parameters

- **fit_intercept**: Default "True" → Calculate intercept
  If False, no intercept (e.g. data is already centered).
- **normalize**: Default "False" (Or if '**fit_intercept**' is False) → No normalization
  If True, normalize $(X - \mu) / \sqrt{\sum_{k=1}^{n} |x_k|^2}$ (L2 norm) before regression.
- **copy_X**: Default "True" → original input data 로 regression계산
  If false, original input data X를 X – μ (mean centering, 표준점수)로 변환후 regression 계산 (regression equation산출에는 차이가 없고 data해석을 할때에 유익할수 있음)
- **n_jobs**: number of CPUs for computation. (1: default, -1: use all CPUs). 규모가 작은 작업의 경우 job 개수를 늘리면 오히려 더 많은 시간 소요

2 attributes

- **coef_** : array, shape (n_features,) or (n_targets, n_features)
- **intercept_** : array

$$Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

5 methods

Methods

fit (X, y[, sample_weight])	Fit linear model.
get_params ([deep])	Get parameters for this estimator.
predict (X)	Predict using the linear model
score (X, y[, sample_weight])	Returns the coefficient of determination R^2 of the prediction.
set_params (**params)	Set the parameters of this estimator.

Linear Regression Example [1/7]

- Using sklearn diabetes data (442 instances)
  - 10 attributes: Age, Sex, Body mass index (BMI), Average blood pressure (ABP) , Six blood serum (S1-S6)
  - Target: quantitative measure of disease progression one year after baseline
  - All the attributes are numeric, mean centered and scaled by standard deviation
  - datasets.load_diabetes() ➔ datasets submodule에 있는 load_diabetes( )

diabetes.data

diabetes.target

442 개

Age	Sex	BMI	ABP	S1	S2	S3	S4	S5	S6
0.038	0.051	0.062	0.022	-0.044	-0.035	-0.043	-0.003	0.020	-0.018
-0.002	-0.045	-0.051	-0.026	-0.001	-0.019	0.074	-0.039	-0.068	-0.092
0.085	0.051	0.044	-0.006	-0.046	-0.034	-0.032	-0.003	0.002	-0.026
-0.089	-0.045	-0.012	-0.037	0.012	0.025	-0.036	0.034	0.023	-0.009

식전혈당
151
75
141
206

Blood : 혈병 (Cruor) + 혈청 (Serum)
혈청성분 → 면역관련물질, 항체…

Linear Regression Example: Data Preprocessing [2/7]

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
```

```
# Load the diabetes dataset
diabetes = datasets.load_diabetes()

diabetes
```

[…..] 가 442개

```
{ 'DESCR': 'Diabetes dataset\n================\……
  'data': array([   [ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226, 0.01990842, -0.01764613],
                    [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,  -0.06832974, -0.09220405],
                  …]),
  'feature_names':  ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6'],
  'target': array([151.,  75., 141., 206., 135.,  97., …., 220.,  57.]) }
```

442개

load_diabetes( ) does not return Python Dictionary, but dictionary-like object, the interesting attributes are: 'data', the data to learn and 'target', the regression target for each sample.

Linear Regression Example : Data Preprocessing [3/7]

- Use only 3rd column values (i.e., BMI)

regr.fit(x, y)
x 는 2D array
y 는 1D or 2D array
x & y 는 same length

```
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]

diabetes_X
```

```
array([[ 0.06169621],
       [-0.05147406],
       [ 0.04445121],
       [-0.01159501],
       [-0.03638469],
       [-0.04069594],
```

- diabetes.data[:, 2] 하면 numpy 1D array 생성 :
  array([0.06169621, -0.05147406, 0.04445121, ….])
- np.newaxis 로 (Nx1) 2D numpy array를 생산
- Regression의 X 값이 여러 개 있을수 있으므로 X data는 2D Numpy Array 로 만들어야 함!
- Target 값은 single 값들의 집합이므로 1D Numpy Array

diabetes.data which is Numpy 2D Array

```
{ 'DESCR': 'Diabetes dataset\n================\……
 'data': array([  [ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226, 0.01990842, -0.01764613],
                  [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,  -0.06832974, -0.09220405],
                 …]),
 'feature_names':  ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6'],
 'target': array([151.,  75., 141., 206., 135.,  97., …., 220.,  57.]) }
```

참고자료

Numpy newaxis

Simply put, the `newaxis` is used to ***increase the dimension*** of the existing array by *one more dimension*, when used *once*.

Thus, **1D** array will become **2D** array, **2D** array will become **3D** array and so on

```
x1 = np.array([1, 2, 3, 4, 5])
x2 = np.array([5, 4, 3])
```

```
In [2]: x1_new = x1[:, np.newaxis]
# now, the shape of x1_new is (5, 1)
# array([[1],
#        [2],
#        [3],
#        [4],
#        [5]])
```

```
x1_new = x1[:]
# array([1, 2, 3, 4, 5])
```

Linear Regression Example : Data Preprocessing [4/7]

- Split dataset into train data set and test data set

```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

diabetes_X

```
array([[ 0.06169621],
       [-0.05147406],
       [ 0.04445121],
       [-0.01159501],
       [-0.03638469],
       [-0.04069594],
```

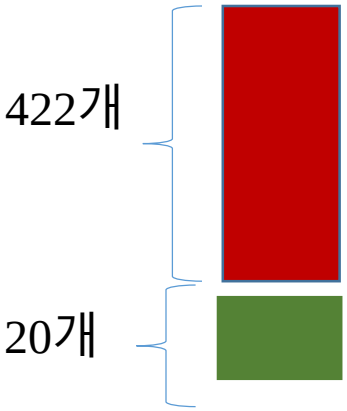
Numpy 2D Array

diabetes.target

```
array([151.,
        75.,
       141.,
       206.
       ,…] )
```

Numpy 1D Array

- 0 ~ 421: training data set (diabetes_X_train, diabetes_Y_train)
- 422 ~ 441: test data set (diabetes_X_test, diabetes_Y_test)



diabetes_X 은 Numpy (Nx1) 2D Array 이므로
diabetes_X[:-20] => diabetes_X[:-20, 0]

regr.fit(x, y)
x 는 2D array
y 는 1D or 2D array
x & y 는 same length

Linear Regression Example: Learning from Data [5/7]

- Create & train linear regression model with training data set

```
# Create linear regression object
regr = linear_model.LinearRegression(copy_X=0)

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

regr
```

False ➔original input data X를 X – μ (표준점수)로 변환후 regression 계산

```
LinearRegression(copy_X=0, fit_intercept=True, n_jobs=1, normalize=False)
```

After training the data, we can do the followings

- print(regr.coef_)
- print(regr.intercept_)

$$Y = \beta_0 + \beta_1 x_1$$

- regr.predict(xi) # xi in test_X
- np.mean ((regr.predict(test_X) – test_Y) ** 2) ➔ mean squared error
  (정답과 예측값의 차이를 2승한 값들의 평균)
- regr.score(test_X, test_Y) ➔ variance score (뒷페이지)

참고사항

regr.fit(X, y) : Learning from Data

fit (*X*, *y*, *sample_weight=None*)

Fit linear model.

Parameters:

X : numpy array or sparse matrix of shape [n_samples,n_features]

Training data

Must be 2D Array!

y : numpy array of shape [n_samples, n_targets]

1D or 2D Array!

Target values. Will be cast to X's dtype if necessary

sample_weight : numpy array of shape [n_samples]

Individual weights for each sample

New in version 0.17: parameter *sample_weight* support to LinearRegression.

Returns:

self : returns an instance of self.

참고사항

Mean Square Error (MSE, 평균제곱오차)

MSE: The average of the squares of the errors

The Smaller
The Better

$$\text{MSE} = \frac{1}{n}\underbrace{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}_{\text{u}} = \frac{\text{u}}{n}$$

MSE ➔ RMSE

무슨의미?

RMSE: Root Mean Square Error

추정치 θ에 대한 추정량 $\hat{\theta}$의 평균 제곱근 편차를 평균 제곱 오차의 제곱근으로 정의할때:

$$\text{RMSE}(\hat{\theta}) = \sqrt{\text{MSE}(\hat{\theta})} = \sqrt{\text{E}((\hat{\theta} - \theta)^2)}.$$

$$\theta_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{bmatrix} \quad \text{and} \quad \theta_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{bmatrix}.$$

이고, 식은:

$$\text{RMSE}(\theta_1, \theta_2) = \sqrt{\text{MSE}(\theta_1, \theta_2)} = \sqrt{\text{E}((\theta_1 - \theta_2)^2)} = \sqrt{\frac{\sum_{i=1}^{n}(x_{1,i} - x_{2,i})^2}{n}}$$

참고사항

regr.score( ) : Variance Score

R^2 = R square = R squared

score (*X*, *y*, *sample_weight=None*)

Returns the coefficient of determination R^2 of the prediction.

The coefficient R^2 is defined as (1 - u/v), where u is the residual sum of squares ((y_true - y_pred) ** 2).sum() and v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Variance Score (0 ~ 1) : 1 – u/v
u = ((y_true – y_pred) ** 2).sum()
v = ((y_true – y_true.mean()) ** 2).sum()

$negative < R^2 < 1$

R^2 controversial even in statistics community!

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

The most general definition of the coefficient of determination is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Linear Regression Example: Validation [6/7]

- Print the results of trained regression

In [39]:
```
# The coefficients
print('Coefficients: \n', regr.coef_)

# The intercept
print('Intercept: \n', regr.intercept_)

# The mean squared error
print('Mean squared error: %.2f' %
      np.mean((regr.predict(diabetes_X_test) - diabetes_y_test) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score : %.2f' %
     regr.score(diabetes_X_test, diabetes_y_test))
```

Numpy 2D ndarray

Numpy 1D ndarray

Predict with test set and calculate the difference between predicted value and the answer

```
Coefficients:
 [ 938.23786125]
Intercept:
 152.918861826
Mean squared error: 2548.07
Variance score : 0.47
```

means

Regression Equation

y = 152.92 + 938.24 * BMI

intercept coefficient

Linear Regression Example : Result Plotting [7/7]

diabetes_X

```
array([[ 0.06169621],
       [-0.05147406],
       [ 0.04445121],
       [-0.01159501],
       [-0.03638469],
       [-0.04069594],
```

diabetes.target

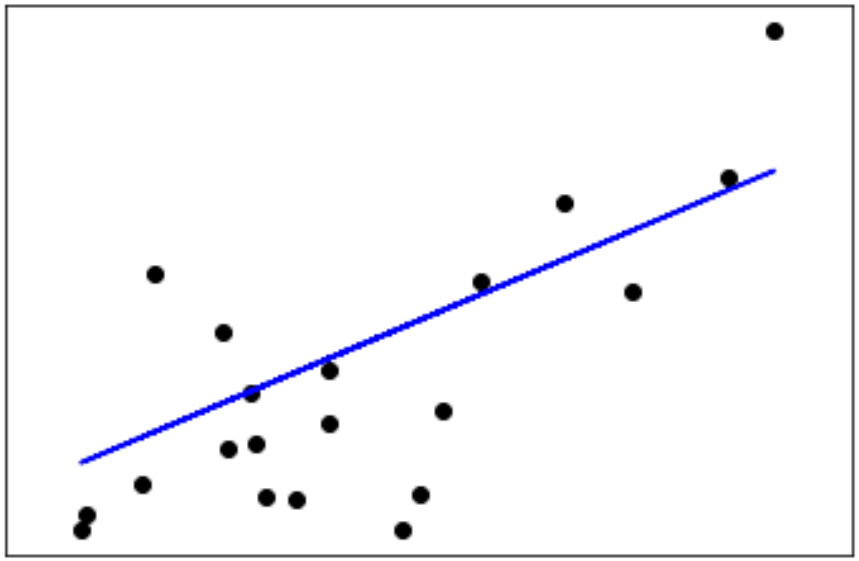
```
array([151.,
        75.,
       141.,
       206.
       ,…] )
```

- Plot the regression model with test data

```
In [46]: # Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color = 'black')
plt.plot(diabetes_X_test, regr.predict(diabetes_X_test), color = 'blue')

plt.xticks(())
plt.yticks(())
```

no ticks on axis
tick: (눈금) X, Y의 간격표시

```
Out[46]: ([], <a list of 0 Text yticklabel objects>)
```



scatter( ) – data points of test data
plot( ) – regression line

All-in-One Code (Regression Classifier)

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
```

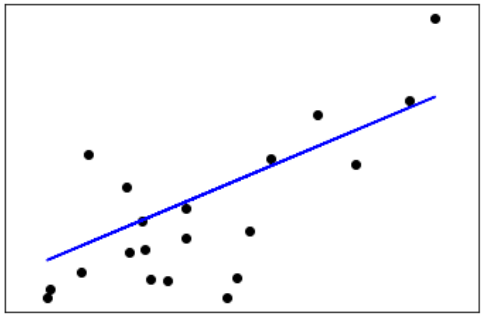
```
# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

```
# Create linear regression object
regr = linear_model.LinearRegression(copy_X=0)

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

```
# The mean squared error
print('Mean squared error: %.2f' %
      np.mean((regr.predict(diabetes_X_test) - diabetes_y_test) ** 2))

# Explained variance score: 1 is perfect prediction
print('Variance score : %.2f' %
     regr.score(diabetes_X_test, diabetes_y_test))
```

```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color = 'black')
plt.plot(diabetes_X_test, regr.predict(diabetes_X_test), color = 'blue')

plt.xticks(())
plt.yticks(())
```

diabetes_X	diabetes.target
array([[0.06169621], [-0.05147406], [0.04445121], [-0.01159501], [-0.03638469], [-0.04069594],	array([151., 75., 141., 206. ,…])



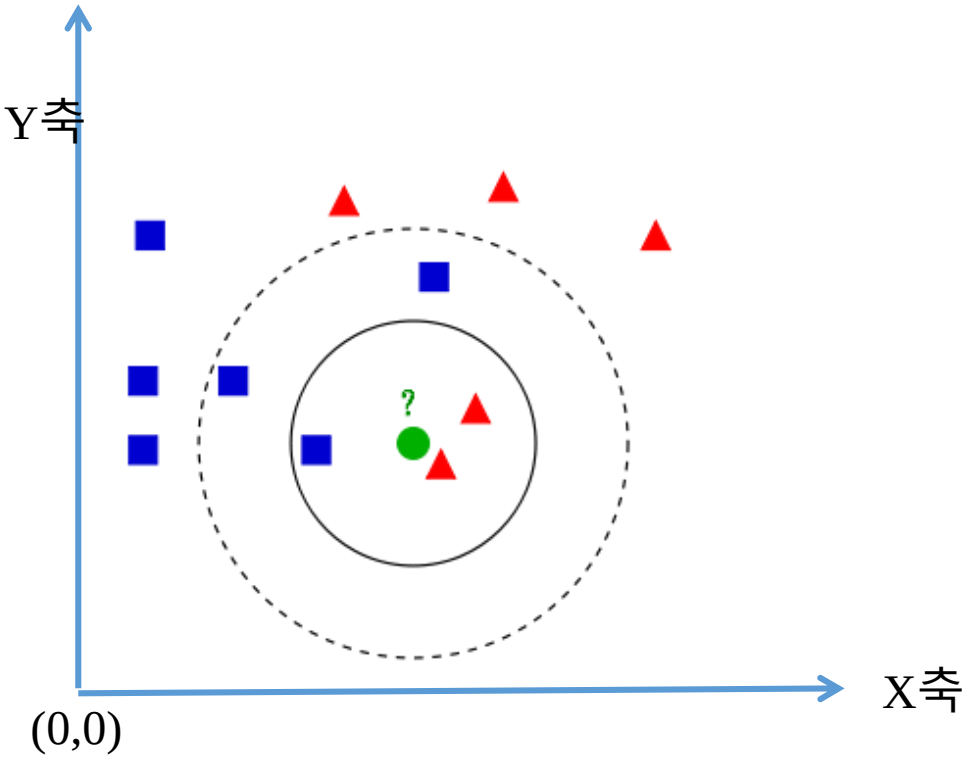
Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbors (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

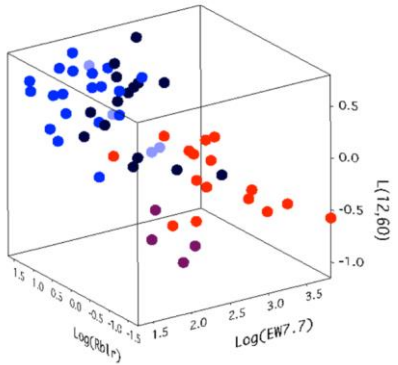
KNN Data

X	Y	Shape
1	3	Rectangle
1	4	Rectangle
1	7	Rectangle
…	…	
3	7.1	Triangle
5	7.2	Triangle
7	7.0	Triangle

New X	New Y	Predict Shape
4	3	?

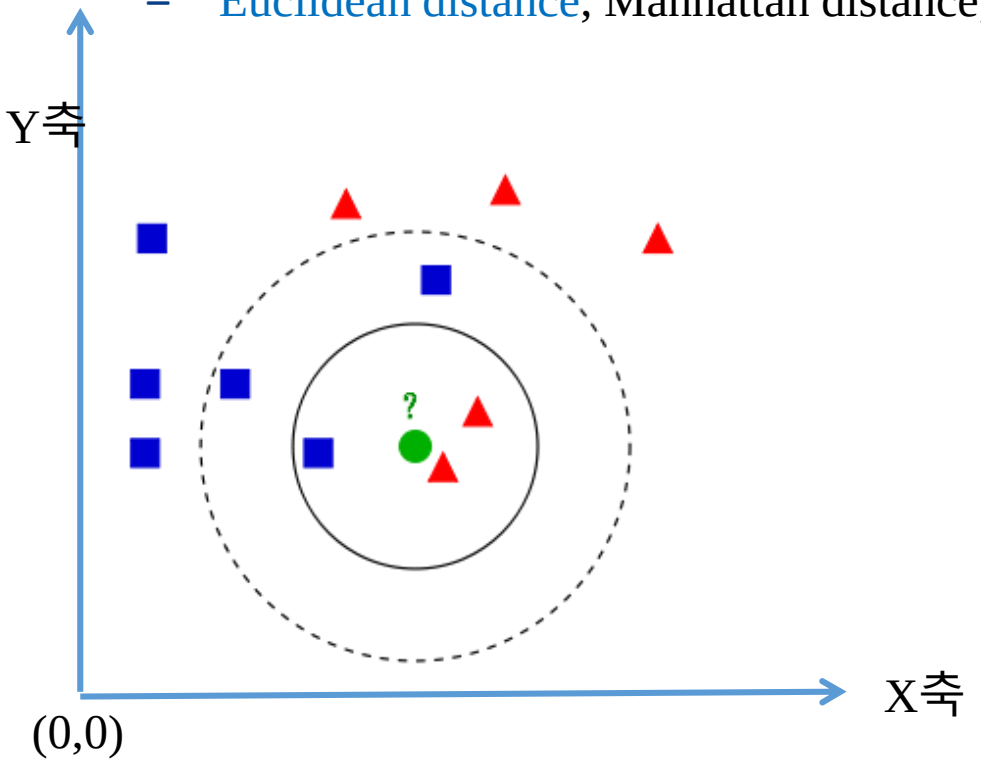


X	Y	Z	Color Ball
0.3	0.4	2.44	Blue
1.2	1.67	2.22	Red
…	….	….	
1.1	1.9	3.8	Black



“sklearn” module: k-Nearest Neighbors (KNN)

- Assume that similar data will be located closely
- Determine the class of new data based on **k** closest data
- No model (no fomula) is used, only data is used for KNN
- Various distance metrics
  - Euclidean distance, Manhattan distance, Mahalanobis distance, etc…



Want to classify new data circle

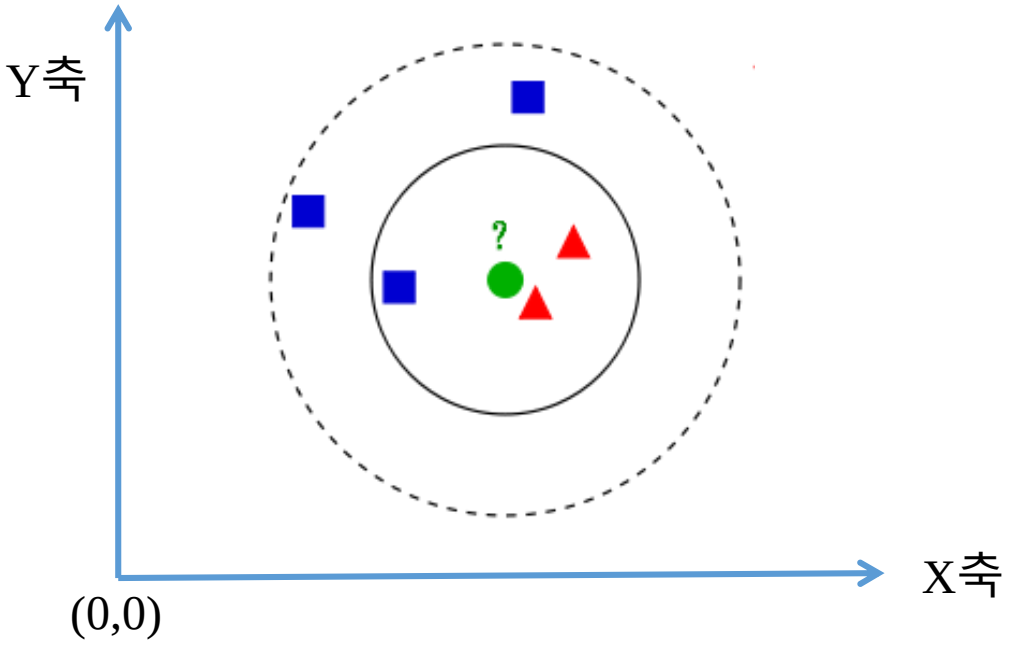
For **k=3**, 2 **triangles** & 1 **rectangle**
Result → classify new data as **red triangle**

For **k=5**, 2 **triangles** & 3 **rectangles**
Result → classify new data as **blue rectangle**

Altman, Naomi S. "An introduction to kernel and nearest-neighbor nonparametric regression." *The American Statistician* 46.3 (1992): 175-185.

Distance vs Count

- New comer의 주변에 5-NN을 했을때에 Blue는 3개, Red는 2개라면…..



- Count만 고려하면 Blue가 3개라서 New comer는 Blue로 classify
- 만약 distance를 고려하여 weight을 준다면 classification은 달라질수 있다

KNeighborsClassifier Class (at neighbors submodule)

sklearn.neighbors.KNeighborsClassifier (**kwargs)

8 parameters

- **n_neighbors (5)**: number of neighbors to use.
- **weights (uniform)**: weight function for prediction.
  {uniform, distance, user defined function}
- **Algorithm**: algorithm to compute nearest neighbors
  {auto, ball_tree, kd_tree, brute}
- **leaf_size**: leaf size for ball/KD tree
- **Metric(minkowski)**: distance metric
- Metric_params: dict, optional
- **p**: power parameter for minkowski metric
  p = 1, Manhattan distance (L1)/ p = 2, Euclidean distance (L2)
- **n_jobs**: number of jobs for computation. (1: default, -1: use all CPUs)

8 methods

Methods

fit (X, y)	Fit the model using X as training data and y as target values
get_params ([deep])	Get parameters for this estimator.
kneighbors ([X, n_neighbors, return_distance])	Finds the K-neighbors of a point.
kneighbors_graph ([X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbors for points in X
predict (X)	Predict the class labels for the provided data
predict_proba (X)	Return probability estimates for the test data X.
score (X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params (**params)	Set the parameters of this estimator.

KNN Example: Iris Classifier [1/5]

- Using sklearn iris dataset (150 lines)
  - 3 classes, 50 instance for each class
  - 4 column data : Sepal length, Sepal width, Petal length, Petal width
  - Target: Classifying 0: iris-setosa, 1: iris-versicolour, 2: iris-virginica
- 뒷산에서 붓꽃을 채집해서 4 column data를 측정하여 입력하면 붓꽃의 종류를 분류해주는 KNN Classifier를 만들려 한다

150개

sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
6.3	3.3	6	2.5	2
5.8	2.7	5.1	1.9	2
7.1	3	5.9	2.1	2

Original iris dataset

- iris: 붓꽃
- sepal: 꽃받침
- petal: 꽃잎

KNN Example : Data Preprocessing [2/5]

- Load **iris** dataset

```
In [54]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
y = iris.target

h = .02  # step size in the mesh
```

determine value of ***k***

iris.data Numpy 2D Array

sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
6.3	3.3	6	2.5	2
5.8	2.7	5.1	1.9	2
7.1	3	5.9	2.1	2

Original iris dataset

2D Numpy

모든 row에서

0,1번째 column만

load_iris( ) returns Dictionary-like object, the interesting attributes are: 'data', the data to learn, 'target', the classification labels, 'target_names', the meaning of the labels, 'feature_names', the meaning of the features, and 'DESCR', the full description of the dataset.

>>> iris

참고

```
{
   'DESCR': 'Iris Plants Database\n ==================== \n \n
            Notes\n -----\n  Data Set Characteristics:\n   y Statistics:\n\n
            ...  ...

   'data': array([[5.1, 3.5, 1.4, 0.2],
          [4.9, 3. , 1.4, 0.2],
          [4.7, 3.2, 1.3, 0.2],
          [4.6, 3.1, 1.5, 0.2],
          [5. , 3.6, 1.4, 0.2],
          [5.4, 3.9, 1.7, 0.4],
          [4.6, 3.4, 1.4, 0.3],
          [5. , 3.4, 1.5, 0.2],
          [4.4, 2.9, 1.4, 0.2],
           ...
           ...
          [5.9, 3. , 5.1, 1.8]]),

   'feature_names': [ 'sepal length (cm)',
                      'sepal width (cm)',
                      'petal length (cm)',
                      'petal width (cm)'],

   'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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]),

    'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

}
```

iris.data (numpy 2D array)

iris.target (numpy 1D array)

plt.scatter( ) with iris dataset "iris.csv"

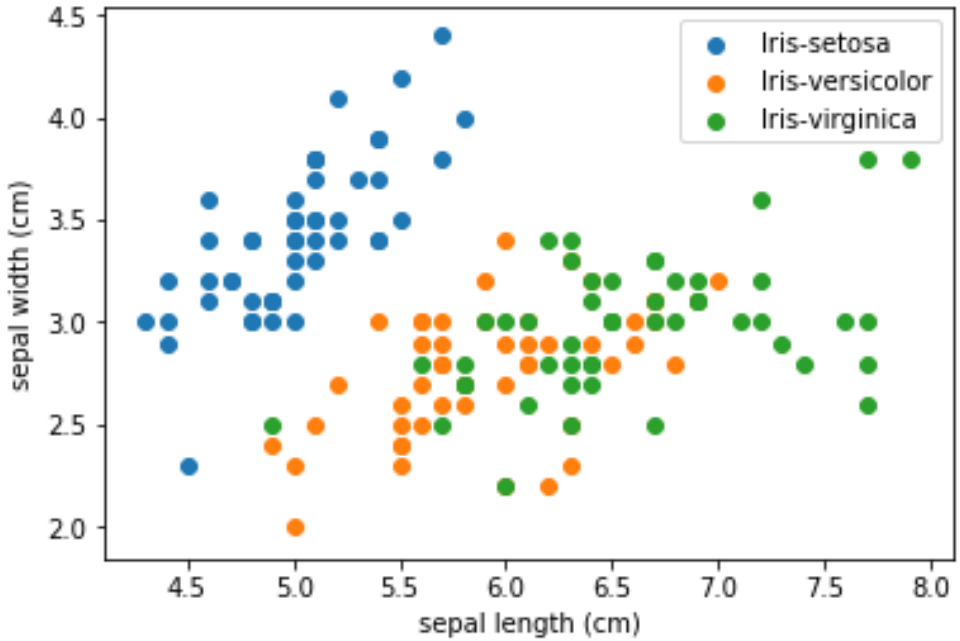
참고

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

iris = pd.read_csv('iris.csv')
groups = iris.groupby('Class')

for cls, group in groups:
    plt.scatter(group.SepalLength, group.SepalWidth, label=cls)
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.legend()
```

Internet에서 iris.csv를 구해서

Class별로 group 나누기

(group key column, 나머지 columns) 형식으로 반환



KNN Example : Data Preprocessing [3/5]

clf.fit(x, y)
x 는 2D array
y 는 1D or 2D array
x & y 는 same length

iris.data

```
'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
```

150개

Numpy 2D Array

```
X = iris.data[:, :2]
```

variable **X**

```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6],
       [5.4, 3.9],
       [4.6, 3.4],
       [5. , 3.4],
       [4.4, 2.9],
       [4.9, 3.1],
       [5.4, 3.7],
       [4.8, 3.4],
       [4.8, 3. ],
       [4.3, 3. ],
       [5.8, 4. ],
       [5.7, 4.4],
```

Numpy 2D ndarray

variable **y**

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Numpy 1D ndarray

KNN Example : Plotting the Training Data [4/5]

variable **X**
Numpy 2D ndarray

150개

sepal length	sepal width
5.1	3.5
4.9	3
4.7	3.2
7	3.2
6.4	3.2
6.9	3.1

variable **y**
Numpy 1D ndarray

target
0
0
0
1
1

- Create color maps for each class
  - 3 colors because data set has 3 classes

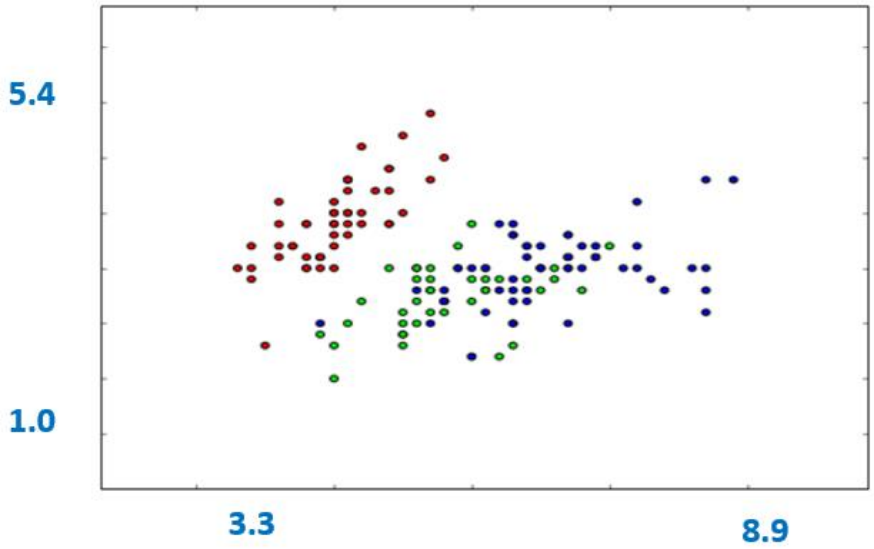
matplotlib.colors.ListedColormap

class matplotlib.colors.ListedColormap(*colors*, *name='from_list'*, *N=None*)

Colormap object generated from a list of colors.

```
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
```

r g b
#FF0000



Scatter Plot Result of dataset X & label data y

KNN Example : Train & Predict KNN Classifier [5/5]

- Create & train KNN model with training set

```
# we create an instance of Neighbours Classifier and fit the data.
clf = neighbors.KNeighborsClassifier(n_neighbors, weights='uniform')
clf.fit(X, y)

clf
```

'uniform': distance를 고려안함.

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=15, p=2,
           weights='uniform')
```

clf.fit(X, y) 의 결과는 150개의 case를 learning 했다는 것이며,
NewComer가 들어오면 k-NN classifier에서 NewComer의 class를 결정

- Predict the class of NewComer with clf.predict( )

```
>>> new_ comer = np.array( [ [3.7,  4.5] ])
>>> iris_class =  clf.predict(new_comer)
>>> print(" The iris_class for new_point : ",  iris_class)
>>> The iris_class for new_point :  0
```

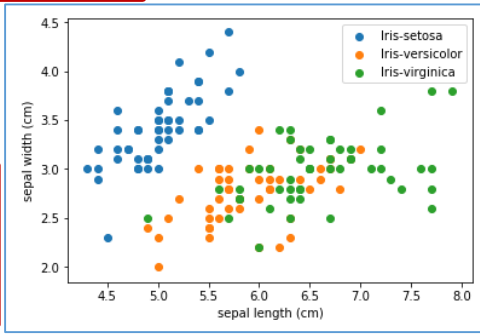
2D Numpy object 로
NewComer를 표현해야

All-in-One Code (KNN Classifier)

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
y = iris.target
```

variable X (Numpy 2D ndarray)	sepal length	sepal width	variable y (Numpy 1D ndarray)	target
	5.1	3.5		0
	4.9	3		0
	4.7	3.2		0
150개	7	3.2		1
	6.4	3.2		1
	6.9	3.1		

```
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
```



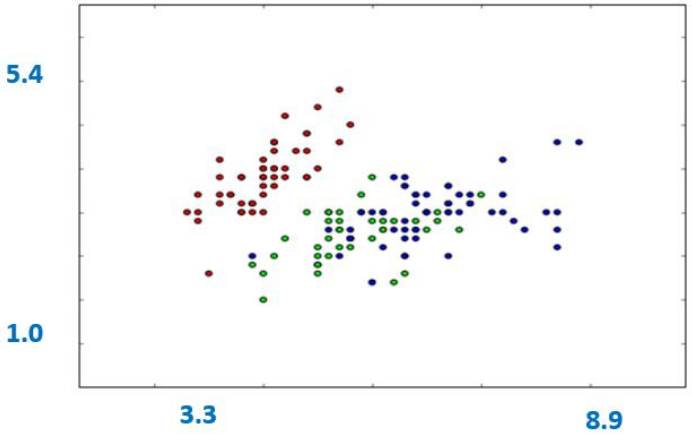

```
# we create an instance of Neighbours Classifier and fit the data.
clf = neighbors.KNeighborsClassifier(n_neighbors, weights='uniform')
clf.fit(X, y)
```

```
>>> new_ comer = np.array( [ [3.7,  4.5] ])
>>> iris_class =  clf.predict(new_comer)
>>> print(" The iris_class for new_point : ",  iris_class)
>>> The iris_class for new_point :  0
```

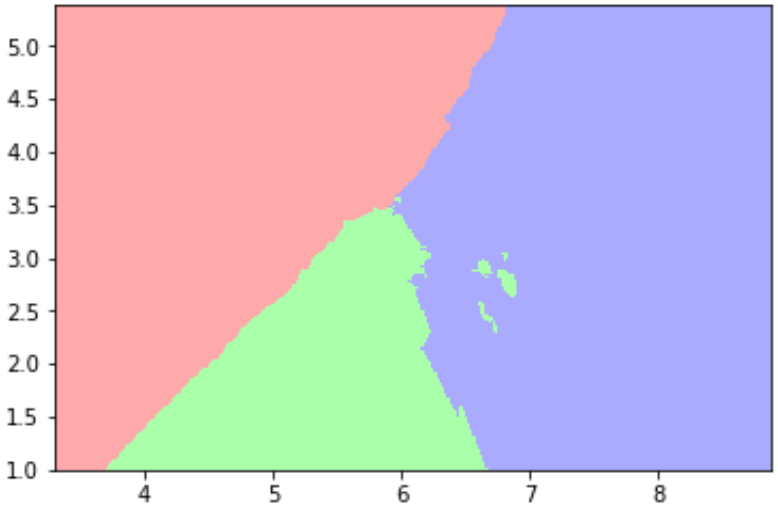
Now we are done?

Better Idea!

- clf.predict( )
- NewComer point의 좌표가 들어오면 color decision

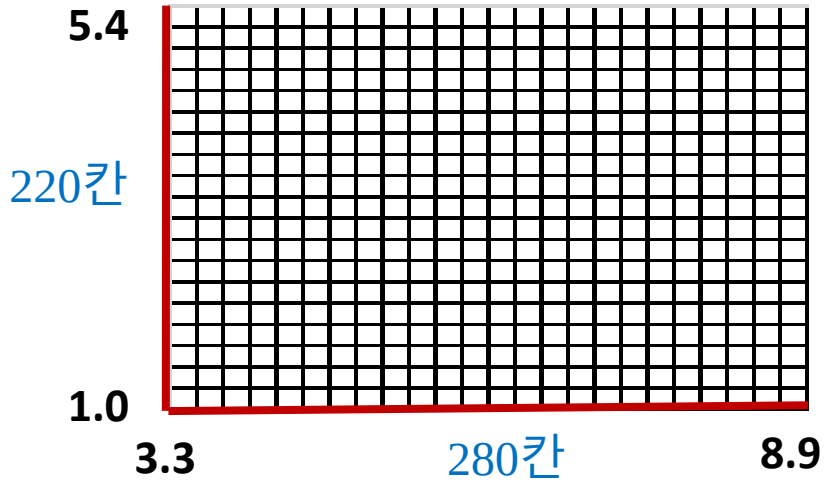


- 전체 region 에 있는 point들을 k-NN classifier 에 입력한다면!
- 전체 region이 몇 개의 sub-area로 나누어져 coloring 될것이고
- Sub-area coloring이 전체 region을 훨씬 intuitive하게 보일것!



전체 Region을 Grid of Points로 표현

- 0.02 간격으로 grid를 만들었다면 point의 개수는 61600개 (280 x 220)



모든 point들은 Numpy 2D array로 표현하려면?

np.meshgrid( ) 가 xx, yy를 생성

xx ➔ Grid의 모든 point들의 x 좌표들를 220 X 280 의 Numpy 2D array로 표현

xx = array([[3.3, 3.32, 3.34, 3.36, 3.38, …… 8.88, 8.9], [....], …, [….]])

280개 list가 220번 반복

yy ➔ Grid의 모든 point들의 y 좌표들를 220 X 280 의 Numpy 2D array로 표현

yy = array ([[1.0, 1.0, 1.0, 1.0, 1.0, ……., 1.0, 1.0], [....], …, [….]])

280개 list가 220번 반복

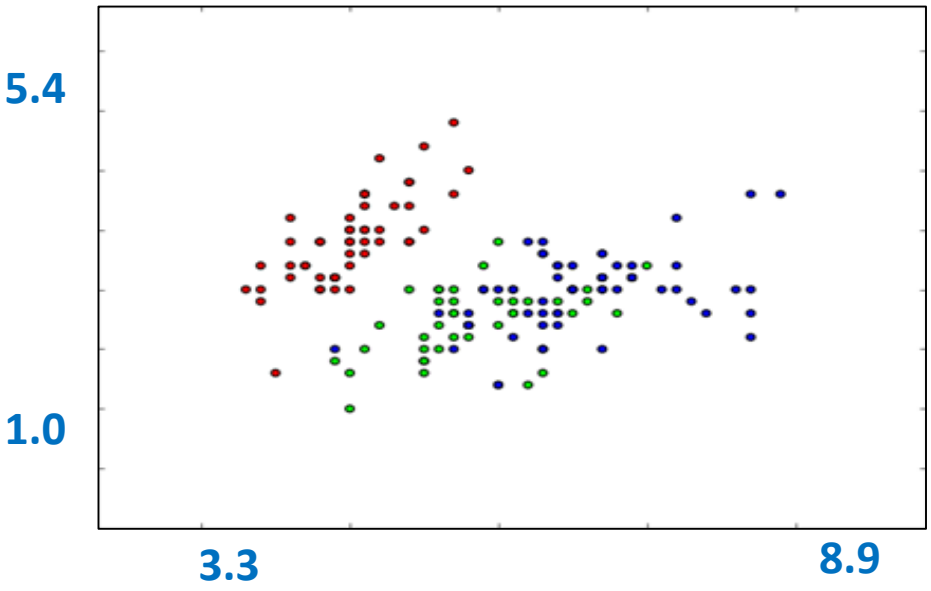
KNN MeshGrid Coloring: Setting Plot Boundary [1/6]

- Configure boundary of plot with maximum & minimum value of each axis

```
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

x_min, x_max, y_min, y_max
```

모든 row에서 1번째 column만

```
(3.2999999999999998, 8.9000000000000004, 1.0, 5.4000000000000004)
```



sepal length	sepal width	target
5.1	3.5	0
4.9	3	0
4.7	3.2	0
7	3.2	1
6.4	3.2	1
6.9	3.1	1
6.3	3.3	2
5.8	2.7	2
7.1	3	2

variable **X** variable **y**

150개 input data를 plot() 하고 x축, y축에 range를 기입했다면..

KNN MeshGrid Coloring: Prepare xx, yy [2/6]

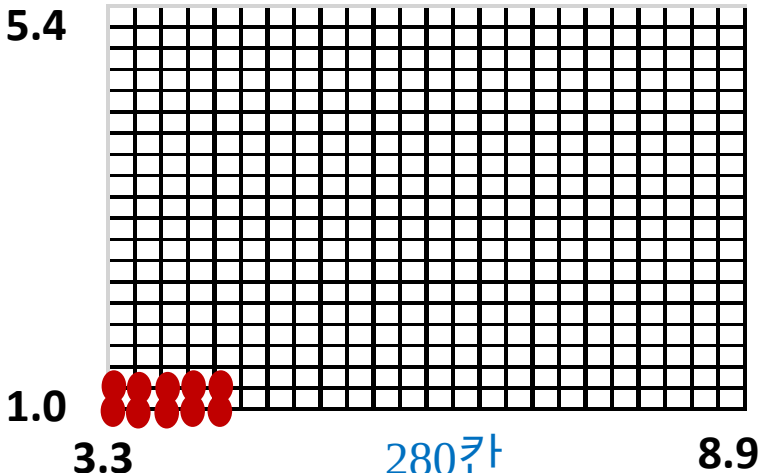
Create a grid with test data points of [min ~ max] with step size **0.02**

x_min : 3.3, x_max: 8.9, y_min: 1.0, y_max: 5.4, h = 0.02

```
# Assign a color to each
# point in the mesh [x_min, x_max] x [y_min, y_max].
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

xx[:2,:5], xx.shape, yy[:2,:5], yy.shape
```

```
(array([[ 3.3 ,  3.32,  3.34,  3.36,  3.38],
        [ 3.3 ,  3.32,  3.34,  3.36,  3.38]]),
 (220, 280),
 array([[ 1.  ,  1.  ,  1.  ,  1.  ,  1.  ],
        [ 1.02,  1.02,  1.02,  1.02,  1.02]]),
 (220, 280))
```

- xx = array([[3.3, 3.32, 3.34, 3.36, 3.38, …… 8.88, 8.9], ….[…]]) → 280개 list가 220번 반복되는 Numpy 2D array
- grid의 220x280 (66180개)의 point들의 x좌표를 Numpy 2D array로 표현
- yy도 마찬가지



xx[:2, :5] → grid에서 2x5 만큼의 subgrid를 구성하는 point들의 xx값

yy[:2, :5] → grid에서 2x5 만큼의 subgrid를 구성하는 point들의 yy값

KNN MeshGrid Coloring: Prepare Input Format [3/6]

xx = array([[3.3, 3.32, 3.34, 3.36, 3.38, …… 8.88, 8.9], […],…, […]])
grid의 220 x 280 (66180개)의 point들의 x좌표를 Numpy 2D array로 표현.

yy는 point들의 y좌표를 Numpy 2D array로 표현

- 이제 우리가 필요한것은 cclf.predict( ) 에 입력할 66180개의 point들의 Numpy 2D Array.
- (예) array([[3.5, 4.2], [4.4, 5.8], …., [..]])
- xx, yy 를 flatten 시키고, xxx, yyy에서 각각 1개의 element를 모아서 point를 만든다

- numpy_ ndarray.ravel() flattens multi-dimensional array

```
In [76]: xr = xx.ravel()
         yr = yy.ravel()

         xr, xr.shape, yr, yr.shape

Out[76]: (array([ 3.3 ,  3.32,  3.34, ...,  8.84,  8.86,  8.88]),
          (61600,),
          array([ 1.  ,  1.  ,  1.  , ...,  5.38,  5.38,  5.38]),
          (61600,))
```

220 X 280 ➔ 61600개로 나열

KNN MeshGrid Coloring: Prepare Coordinates of Points [4/6]

- Numpy.c_ concatenate with 2 axes

```
In [79]: xy = np.c_[xr, yr]

         xy, xy.shape

Out[79]: (array([[ 3.3 ,  1.  ],
                 [ 3.32,  1.  ],
                 [ 3.34,  1.  ],
                 ...,
                 [ 8.84,  5.38],
                 [ 8.86,  5.38],
                 [ 8.88,  5.38]]), (61600, 2))
```

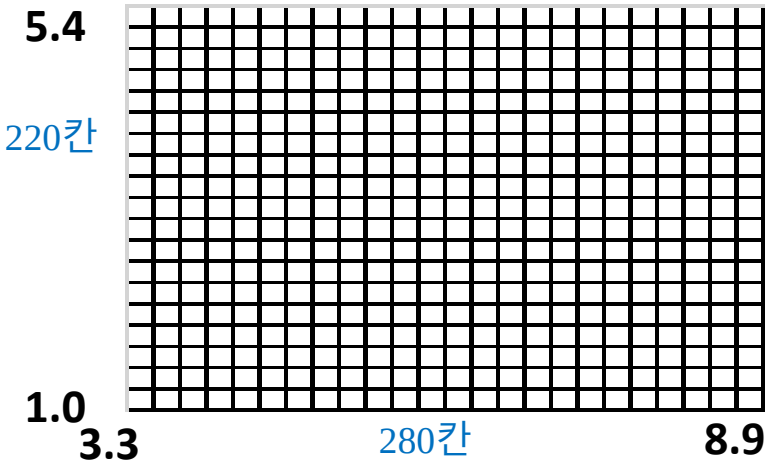
61600개의 points

xr: 모든 point 들의 x 좌표를 모아둔 Numpy 1D array

```
array([ 3.3 ,  3.32,  3.34, ...,  8.84,  8.86,  8.88])
```

yr: 모든 point 들의 y 좌표를 모아둔 Numpy 1D array

```
array([ 1.  ,  1.  ,  1.  , ...,  5.38,  5.38,  5.38])
```



Now, xy 완성!

Grid의 모든 point들을 clf.predict( ) 에 입력가능한 numpy 2D ndarray로 완성!

Numpy.c_

Examples

```
>>> np.c_[np.array([1,2,3]), np.array([4,5,6])]
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> np.c_[np.array([[1,2,3]]), 0, 0, np.array([[4,5,6]])]
array([[1, 2, 3, 0, 0, 4, 5, 6]])
```

np.c_ is a shorthand for numpy.concatenate( )

```
 8 length = 10
 9 x = np.array(range(0,length)).reshape(length,1)
10 y = np.array([random()*10 for i in range(length)]).reshape(length,1)
15 xy_data = np.concatenate((x, y), axis=1)
```

x: Numpy (10x1) 2D Array

```
array(
[[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]] )
```

y: Numpy (10x1) 2D Array

```
array(
[[6.22444081]
[2.39957452]
[6.01572263]
[8.94770864]
[6.8150673 ]
[0.13597106]
[7.80840772]
[3.71623543]
[3.20308959]
[5.42052624]] )
```

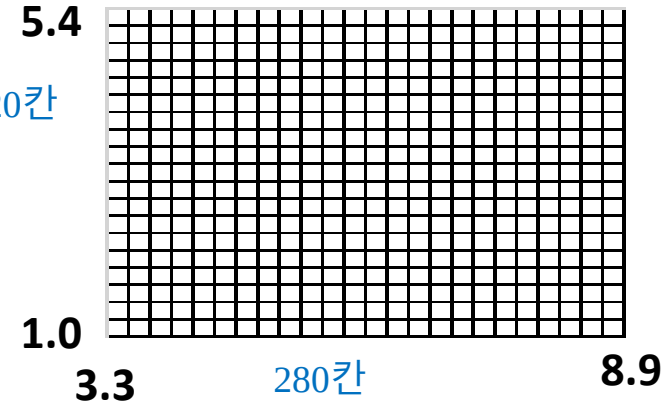
xy_data: Numpy (10x2) 2D Array

```
array(
[[0.         6.22444081]
[1.         2.39957452]
[2.         6.01572263]
[3.         8.94770864]
[4.         6.8150673 ]
[5.         0.13597106]
[6.         7.80840772]
[7.         3.71623543]
[8.         3.20308959]
[9.         5.42052624]] )
```

KNN MeshGrid Coloring: Predict Colors of All Points (Z) [5/6]

- Predict class of each data points

```
In [81]: Z = clf.predict(xy)

         Z

Out[81]: array([0, 0, 0, ..., 2, 2, 2])
```



Learning된 KNN classifier로 grid의 61600개점을 predict하여 class label (0,1,2 값)들을 생산

- Reshape the ndarray of predicted values from 1D to 2D for plotting

```
In [84]: Z = Z.reshape(xx.shape)

         Z, Z.shape

Out[84]: (array([[0, 0, 0, ..., 2, 2, 2],
                 [0, 0, 0, ..., 2, 2, 2],
                 [0, 0, 0, ..., 2, 2, 2],
                 ...,
                 [0, 0, 0, ..., 2, 2, 2],
                 [0, 0, 0, ..., 2, 2, 2],
                 [0, 0, 0, ..., 2, 2, 2]]), (220, 280))
```

220X280

Z

220

280

Grid의 61600 point들에 들어갈 Predict된 class label (0,1,2 값) 들을 280 x 220 shape의 Numpy 2D array로 Z에 저장

KNN MeshGrid Coloring: Colored Subregions [6/6]

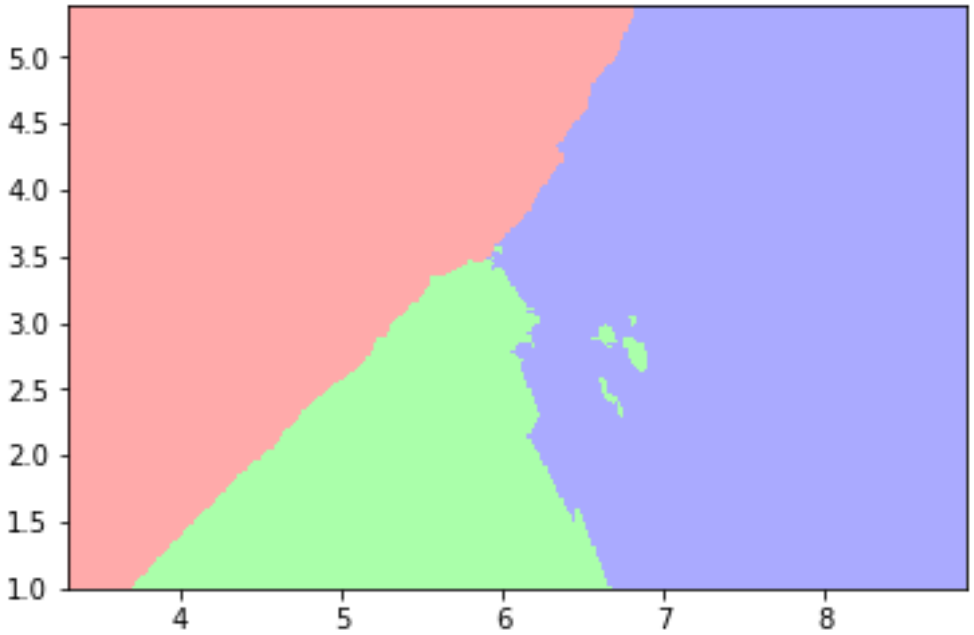
- Print the result of prediction with grid test set
  - Color of each region means which class a data point classified by k-NN classifier

light red light green light blue

```
# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
```

```
: plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

```
: <matplotlib.collections.QuadMesh at 0x21c37e70908>
```



X: Numpy 2D Array
y: Numpy 1D Array
xx: Numpy 2D Array
yy: Numpy 2D Array
Z: Numpy 2D Array

Z

```
(array([[0, 0, 0, ..., 2, 2, 2],
        [0, 0, 0, ..., 2, 2, 2],
        [0, 0, 0, ..., 2, 2, 2],
        ...,
        [0, 0, 0, ..., 2, 2, 2],
        [0, 0, 0, ..., 2, 2, 2],
        [0, 0, 0, ..., 2, 2, 2]]), (220, 280))
```

참고자료

matplotlib.pyplot.pcolormesh(*args, **kwargs)

Plot a quadrilateral mesh.

Call signatures:

여러점으로 구성된 지역전체를 coloring

```
pcolormesh(C)
pcolormesh(X, Y, C)
pcolormesh(C, **kwargs)
```

C에 있는 값에 cmap을 대입하거나
default로 가진 color순서로 적용할수도

Create a pseudocolor plot of a 2-D array.

pcolormesh is similar to `pcolor()`, but uses a different mechanism and returns a different object; pcolor returns a `PolyCollection` but pcolormesh returns a `QuadMesh`. It is much faster, so it is almost always preferred for large arrays.

C may be a masked array, but *X* and *Y* may not. Masked array support is implemented via *cmap* and *norm*, in contrast, `pcolor()` simply does not draw quadrilaterals with masked colors or vertices.

Returns: matplotlib.collections.QuadMesh

Other Parameters:

cmap : Colormap, optional

A `matplotlib.colors.Colormap` instance. If `None`, use rc settings.

norm : Normalize, optional

A `matplotlib.colors.Normalize` instance is used to scale luminance data to 0,1. If `None`, defaults to `normalize()`.

vmin, vmax : scalar, optional

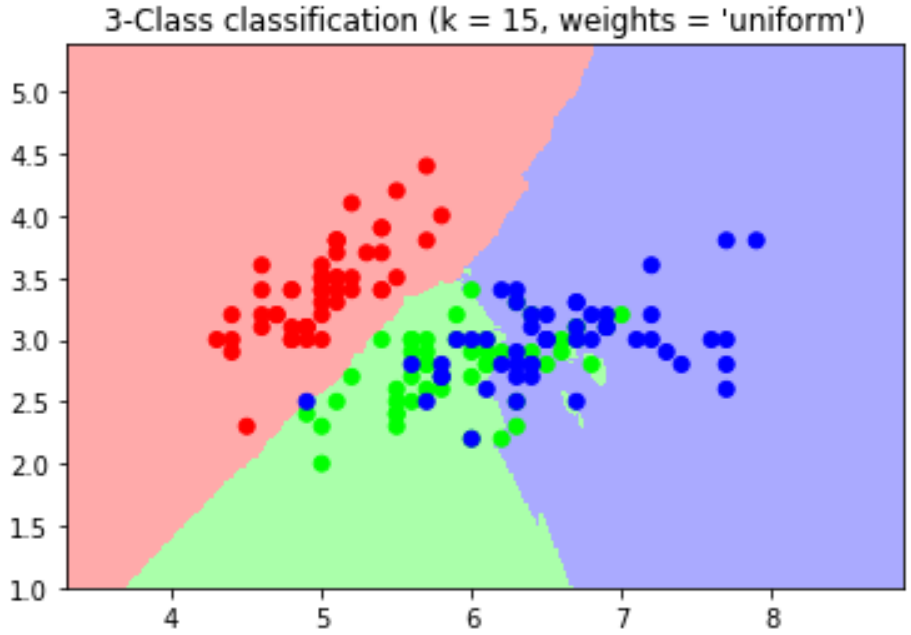
MeshGrid Coloring with Plotting Training Data

Region of each class and distribution of original data are similar

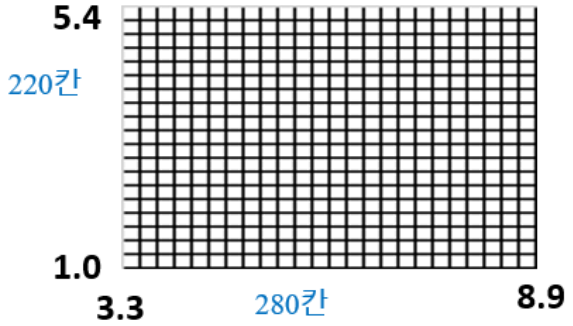
```
# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

```
: plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = %i, weights = 'uniform')"
          % (n_neighbors))
```

```
: <matplotlib.text.Text at 0x21c3961c048>
```



X: Numpy 2D Array
y: Numpy 1D Array
xx: Numpy 2D Array
yy: Numpy 2D Array
Z: Numpy 2D Array

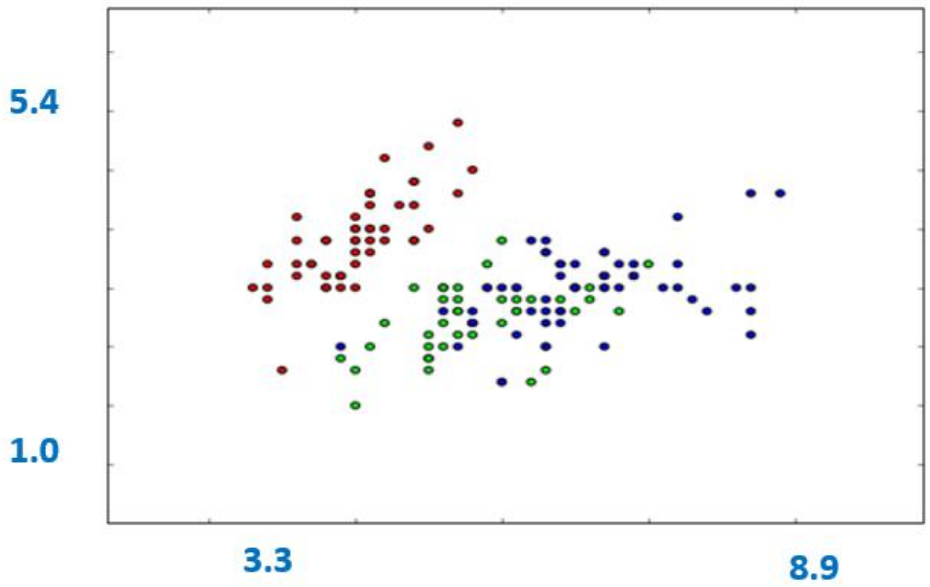


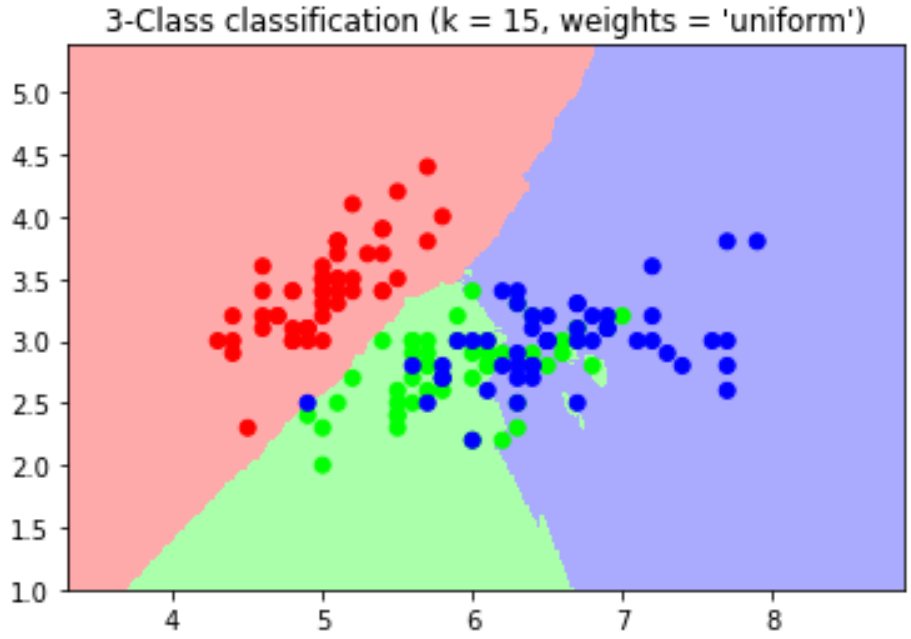
variable **X** variable **y**

150개

sepal length	sepal width	target
5.1	3.5	0
4.9	3	0
4.7	3.2	0
7	3.2	1
6.4	3.2	1
6.9	3.1	1
6.3	3.3	2
5.8	2.7	2
7.1	3	2

Which Visualization Do You Like Better?



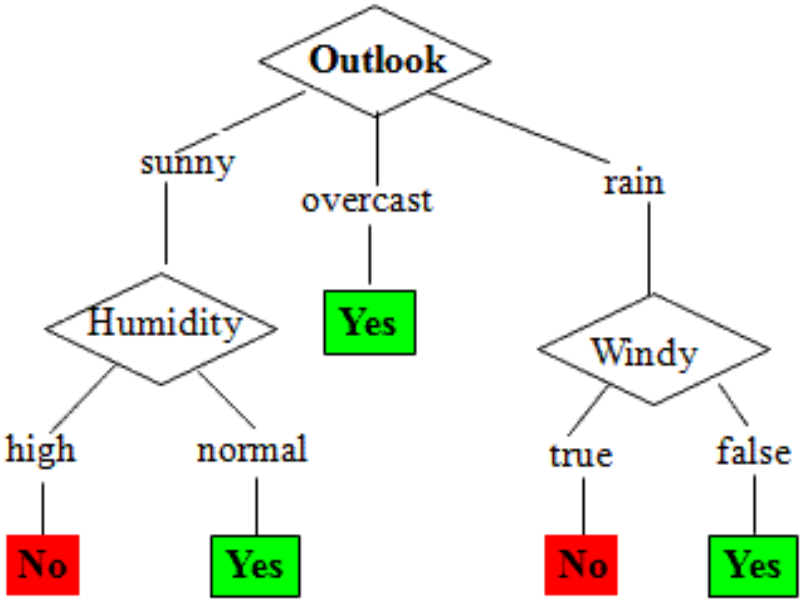


Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbors (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

Decision Tree

- Classifier using Tree



- Early Decision Trees
  - CHAID(1980), CHART(1984)
- Current Decision Trees
  - ID3(1986) → C4.5(1993) → C5.0
    - C5.0 : commercial

- References

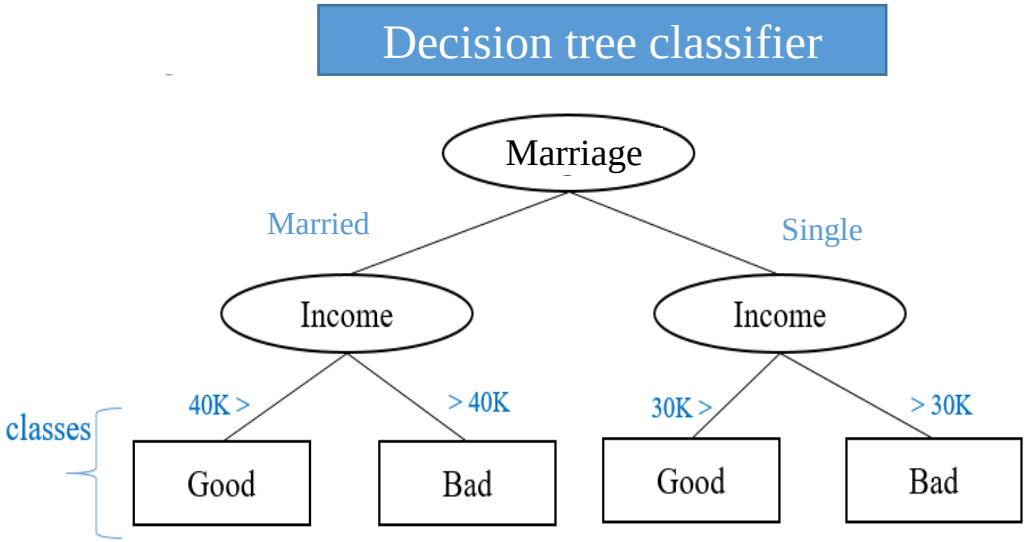
  - Kass, Gordon V. "An exploratory technique for investigating large quantities of categorical data. (CHAID)" *Applied statistics,* 1980
  - Breiman, L. “Classification and Regression Trees (CHART)” New York: Routledge. 1984.
  - Quinlan, J. Ross. "Induction of decision trees (ID3)" *Machine learning*1.1 1986.
  - Quinlan, J. R. “C4.5: Programs for Machine Learning.” Morgan Kaufmann Publishers, 1993.

What is Decision Tree [1/3]

- Supervised learning model
- Flowchart-like structure to classify an outcome based on a set of predictors
  - Ellipse node: split condition
  - Rectangle node: classified class (= Leaf node)

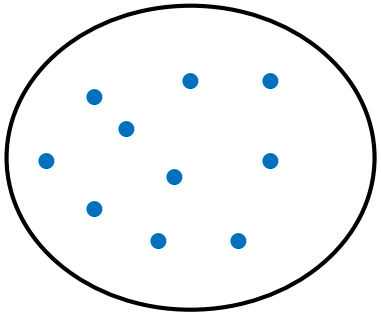
Degree를 check하고 그다음에 Income을 check해서 class를 정하는 방법

Name	Marriage	Income	Credit Status
H.Kim	Married	$50000	Good
P. Lee	Single	$35000	Good
J. Hong	Married	$18000	Bad
W.Sawn	Married	$39000	Good
J. Doe	Single	$55000	Good
W. Son	Single	$25000	Bad
Q. Li	Single	$15000	Bad

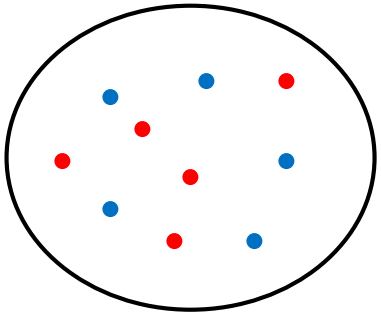


What is Decision Tree? [2/3]

- Split dataset based on an attribute with highest purity
  - Purity: proportion of data in a split that belong to class k
  - Maximum purity: each split has data of same class
  - Gini Index, Entropy



Max Purity



Min Purity

Lawn Mower vs Riding Lawn Mower

$100 ~ $300

$1000 ~ $3000

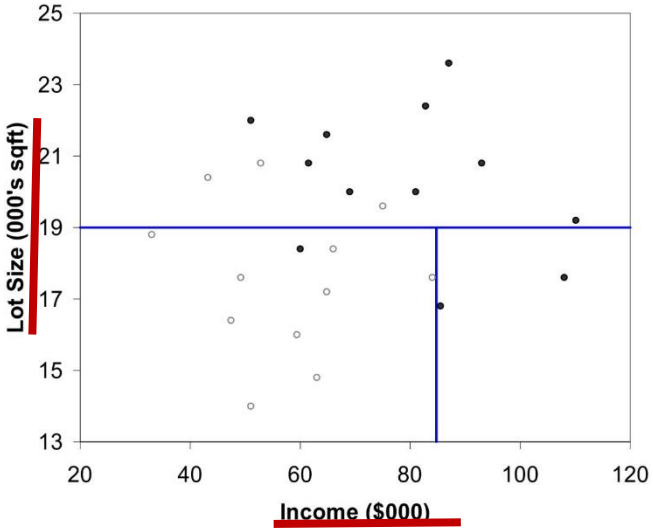
What is Decision Tree? [3/3]

Ownership of Riding Lawn Mower

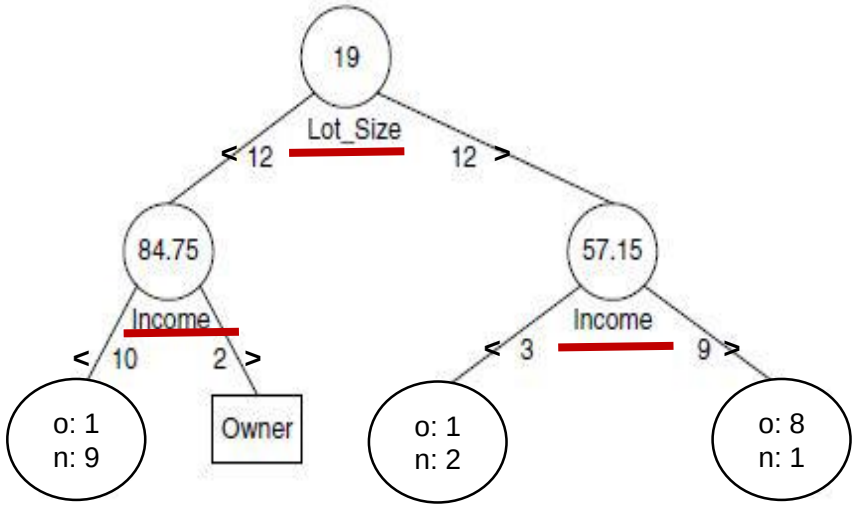
($1000) (1000 Sq_Ft)

Income	Lot_Size	Ownership
60.0	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87.0	23.6	owner
110.1	19.2	owner
108.0	17.6	owner
82.8	22.4	owner
69.0	20.0	owner
93.0	20.8	owner
51.0	22.0	owner
81.0	20.0	owner
75.0	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84.0	17.6	non-owner
49.2	17.6	non-owner
59.4	16.0	non-owner
66.0	18.4	non-owner
47.4	16.4	non-owner
33.0	18.8	non-owner
51.0	14.0	non-owner
63.0	14.8	non-owner

Visual representation



Tree structure



Recursive Partitioning in Decision Tree

- Pick one of the predictor variables, x_i
- Pick a value of x_i, say s_i, that **divides the training data** into two (not necessarily equal) portions
- Measure how "**pure**" or homogeneous each of the resulting portions are
  - "Pure" = containing records of mostly one class
- Algorithm tries different values of x_i, and s_i to maximize purity in initial split
- After you get a "**maximum purity**" split, **repeat** the process for a second split, and so on

Decision Tree for Ownership of Riding Mowers

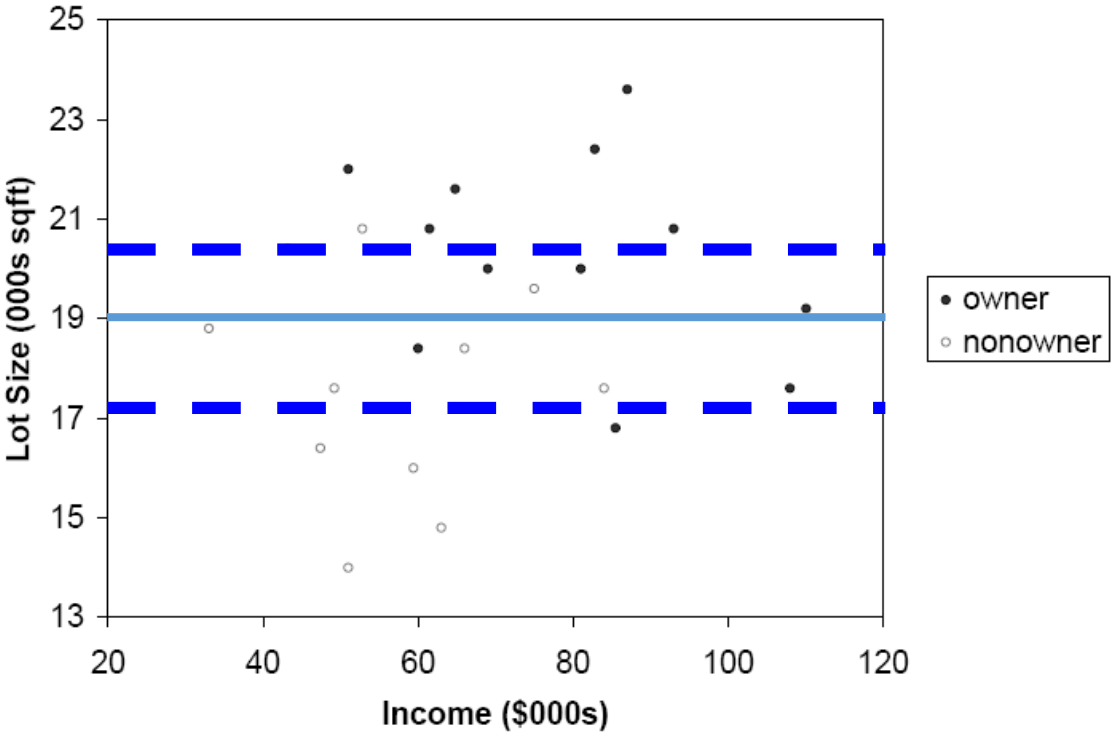
- Goal: Classify 24 households as owning or not owning riding lawn-mowers
- Predictors = Income, Lot Size

Income	Lot_Size	Ownership
60.0	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87.0	23.6	owner
110.1	19.2	owner
108.0	17.6	owner
82.8	22.4	owner
69.0	20.0	owner
93.0	20.8	owner
51.0	22.0	owner
81.0	20.0	owner
75.0	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84.0	17.6	non-owner
49.2	17.6	non-owner
59.4	16.0	non-owner
66.0	18.4	non-owner
47.4	16.4	non-owner
33.0	18.8	non-owner
51.0	14.0	non-owner
63.0	14.8	non-owner

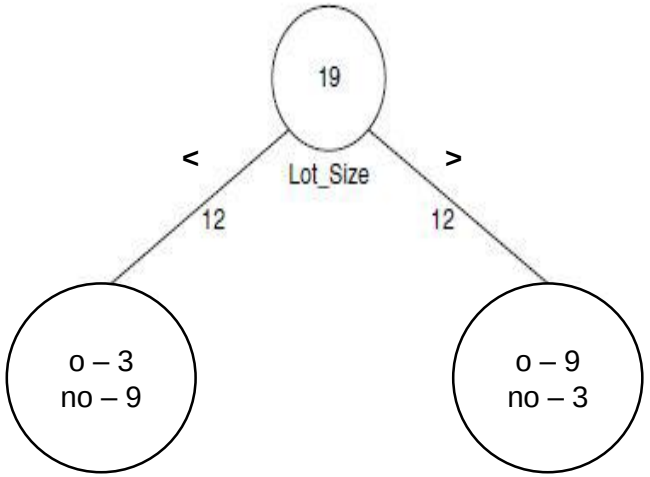
- **How to split the values of continuous variable!**
- Sort records according to one variable (say, lotSize)
- Find the splitpoint of lotSize (halfway between 14.0 and 23.6 ➔ 19) using Gini Index calculation
- Divide records into those with lotsize > 19 and those with lotsize < 19
- For each splitted area, try the next variable (say, income), which is $84,000 and $57,000

The first split: Lot Size ➔ 19,000 sqrt

1st Round = Drawing Blue Line



Tree structure

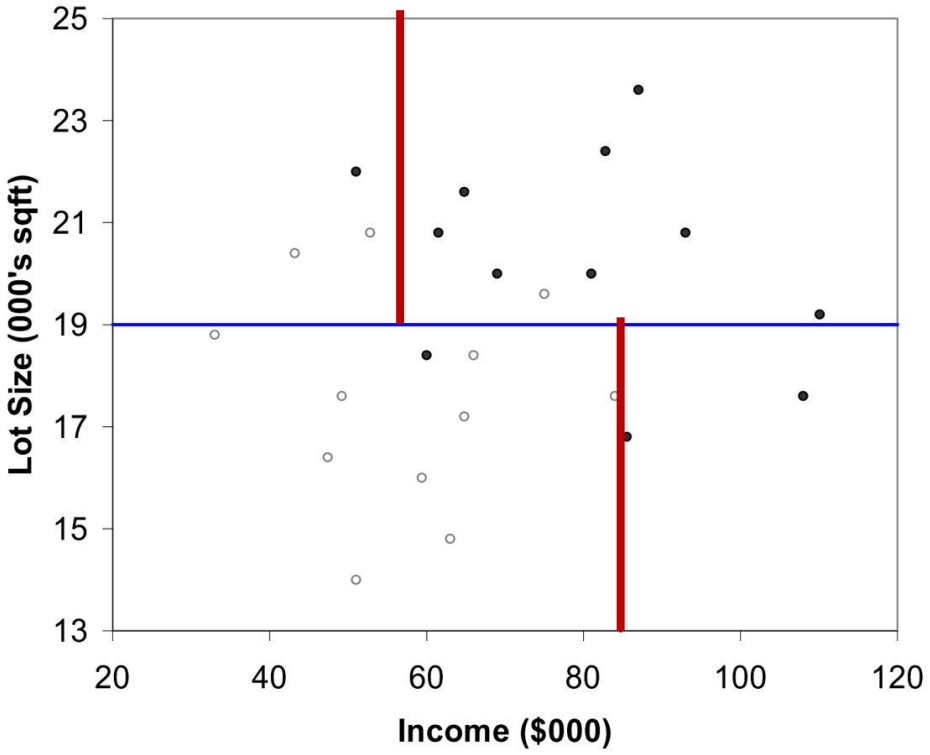


** Purity를 measure하는 Gini Index 계산으로 split point를 결정

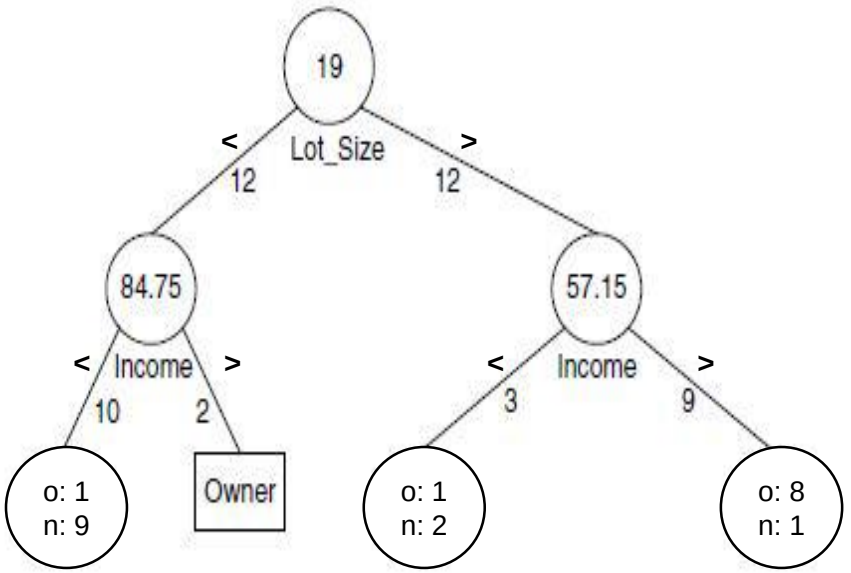
** 모든 dotted line으로 생성된 split에서 Gini Index를 가지고 Information-Gain Ratio를 계산해서 가장 큰값이 되는 dotted line을 next split point로 결정

Second Split: Income ➔ $84,000 & $57,000

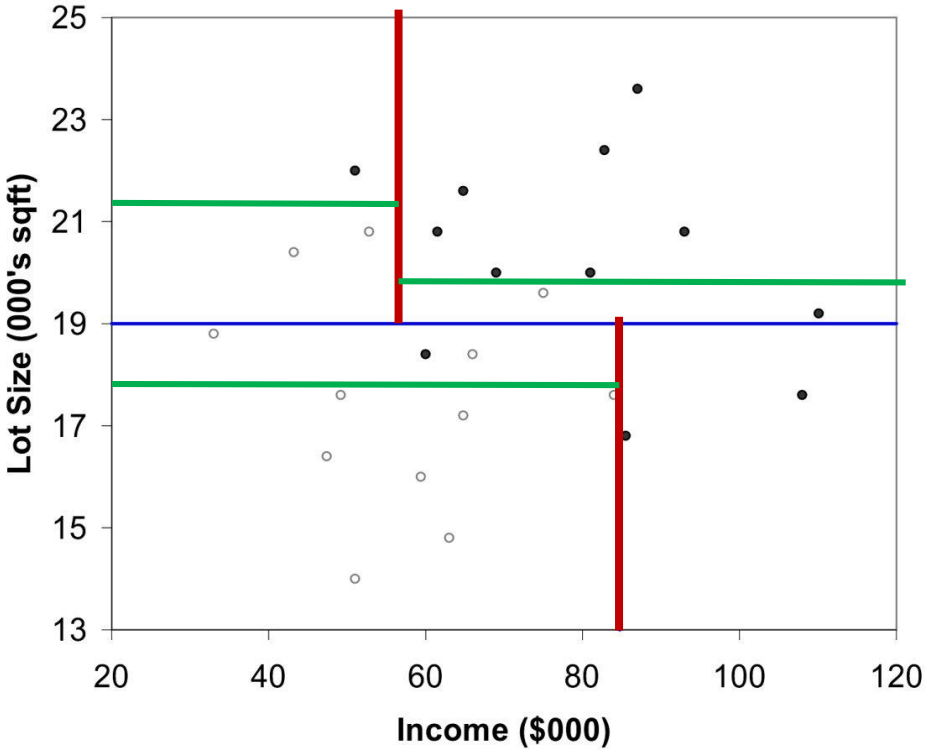
2nd Round = Drawing Red Line



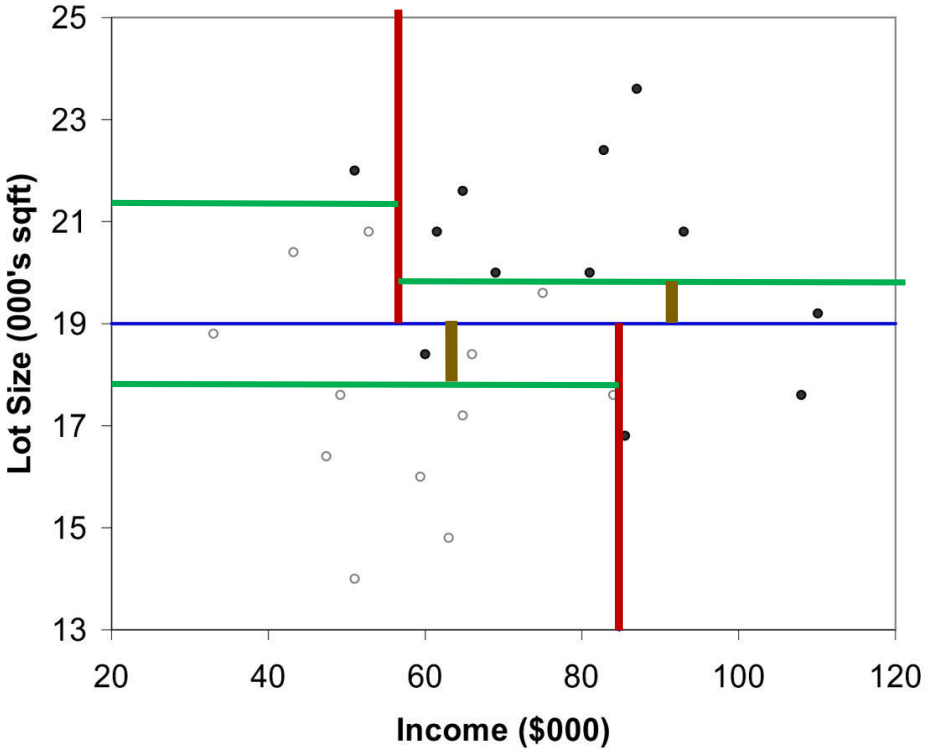
Tree structure



3rd Round
= Drawing Green Line



4th Round
= Drawing Black Line



언제까지 Line을 만들어가나!

모든 Box속에 있는 동질의 내용물이 될때까지! (Pure)

After All Splits

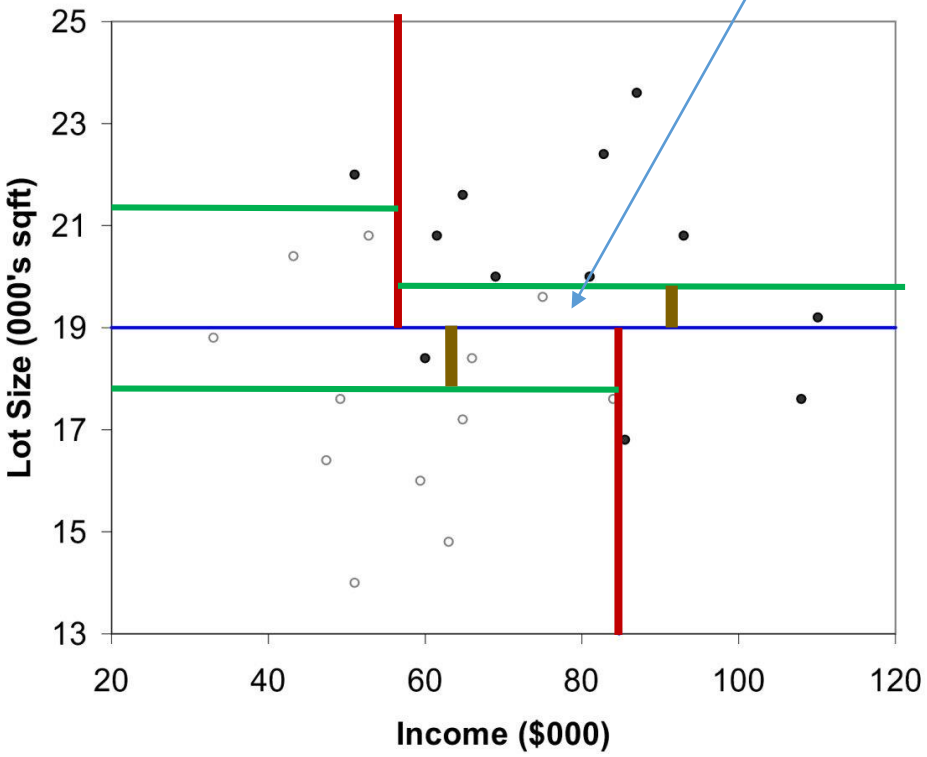
- Result of full grown tree:

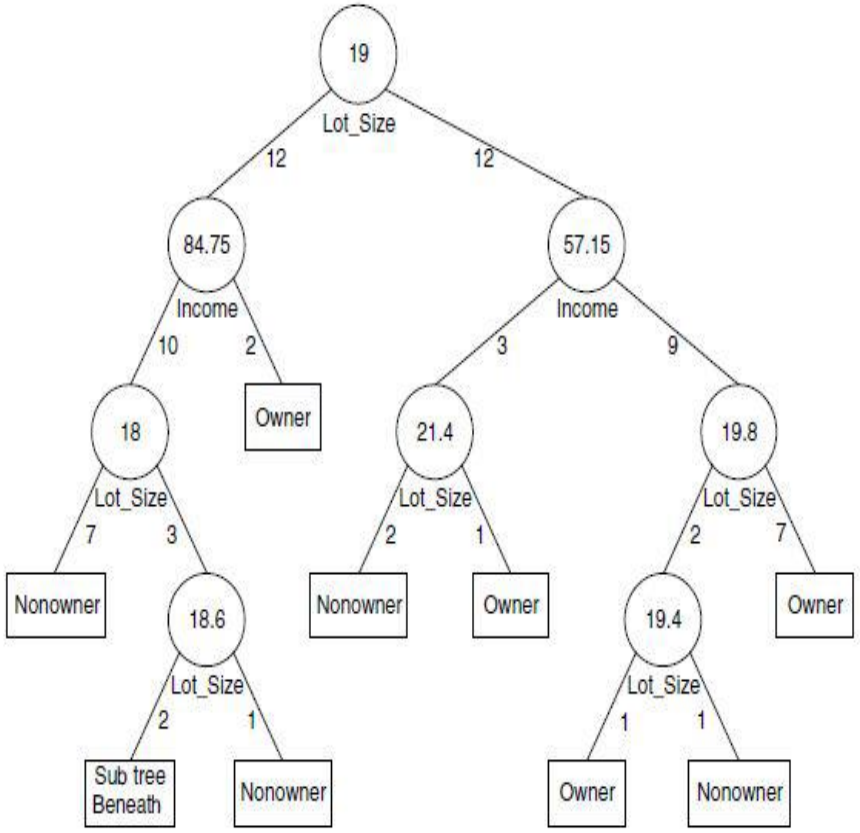
  Each rectangle is completely pure

- Danger of overfitting problem

Overfitting: Training Data에서 general property를 찾아야 하는데, 특정 Training Data에 심하게 의존하는 방식으로 learning이 이루어지면 Training Data Instance 에 too much fit한 pattern이 나오는 현상!

이곳에 new data가 있으면 white로 classify하게 되나, 그것은 overfitting으로 인한 부적절한 classification

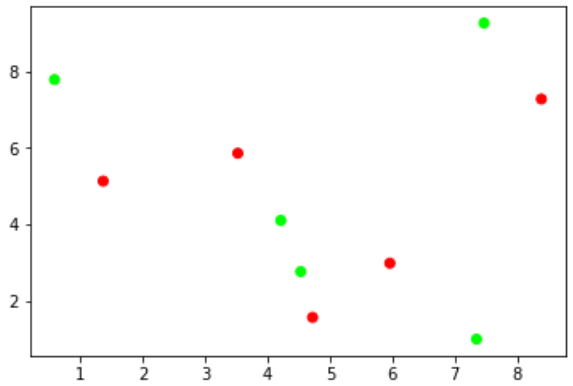




Simple Decision Tree using Python 2D Lists

- Predict z from xy using Decision Tree

```
from random import *
import numpy as np
import pandas as pd
from sklearn import tree
import matplotlib.pyplot as plt

xy = []
z  = []
for i in range(10):
    xy.append([round((random()*10), 2), round((random()*10), 2)])

z = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

print("Python 2D List xy: \n", xy)
print("Python 1D List z: \n",  z)
print("--------")
clf = tree.DecisionTreeClassifier( )
clf.fit(xy, z)

test_data = [3, 2.45]
print("Decison Tree Test_Data [%d, %d]" % (test_data[0], test_data[1]))
print("The decision for the test data;" , clf.predict([test_data]))
```



Training decision tree model with 2 Python Lists

Print predicted results

```
Python 2D List xy:
 [[3.85, 3.98], [2.54, 7.95], [5.9, 1.48], [2.53, 1.07], [6.74, 1.05],
[0.92, 1.87], [6.2, 8.1], [7.93, 3.16], [4.92, 5.15], [3.05, 3.0]]
Python 1D List z:
 [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

```
--------
Decison Tree Test_Data [3, 2]
The decision for the test data; [2]
```

Simple Decision Tree using Numpy 2D Arrays

- Predict z from xy using Decision Tree

```
from random import *
import numpy as np
import pandas as pd
from sklearn import tree
import matplotlib.pyplot as plt

xy = []
z  = []
for i in range(10):
    xy.append([round((random()*10), 2), round((random()*10), 2)])

z = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

xy = np.array(xy)
z  = np.array(z)
print("Python Numpy 2D Array xy: \n", xy)
print("Python Numpy 1D Array z: \n",  z)
print("--------")
clf = tree.DecisionTreeClassifier( )
clf.fit(xy, z)

test_data = np.array([3, 2.45])
print("Decison Tree Test_Data:", test_data)
print("The decision for the test data;" , clf.predict([test_data]))
```

Training decision tree model with 2 Numpy 2D arrays

```
Python Numpy 2D Array xy:
 [[4.74 5.89]
 [7.11 9.83]
 [4.93 0.85]
 [3.52 1.77]
 [7.47 9.7 ]
 [5.51 1.88]
 [3.66 8.  ]
 [8.12 5.24]
 [7.99 0.96]
 [1.44 8.4 ]]
Python Numpy 1D Array z:
 [1 1 1 1 1 2 2 2 2 2]
```

```
--------
Decison Tree Test_Data: [3.   2.45]
The decision for the test data; [2]
```

Simple Decision Tree using Pandas DataFrames

```
from random import *
import numpy as np
import pandas as pd
from sklearn import tree
import matplotlib.pyplot as plt

xy = []
z  = []
for i in range(10):
    xy.append([round((random()*10), 2), round((random()*10), 2)])

z = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

xy = pd.DataFrame(xy)
z  = pd.Series(z)
print("Python Pandas DataFrame  xy: \n", xy)
print("Python Pandas Series  z: \n",  z)
print("--------")
clf = tree.DecisionTreeClassifier( )
clf.fit(xy, z)

test_data = pd.DataFrame([[3, 2.45]])
print("Decison Tree Test_Data:\n", test_data)
print("The decision for the test data: " , clf.predict(test_data))
```

Training decision tree model with 2 Pandas DataFrame

```
Python Pandas DataFrame  xy:
      0     1
0  4.36  4.46
1  0.94  9.11
2  8.04  4.56
3  6.29  4.58
4  4.81  6.36
5  5.68  8.90
6  0.79  7.57
7  3.34  1.67
8  5.77  6.60
9  3.25  5.21
Python Pandas Series  z:
 0    1
1    1
2    1
3    1
4    1
5    2
6    2
7    2
8    2
9    2
dtype: int64
```

```
--------
Decison Tree Test_Data:
    0     1
0  3  2.45
The decision for the test data:  [2]
```

DecisionTreeClassifier Class (at tree submodule)

- sklearn.tree.DecisionTreeClassifier (***kwargs*)
  - **criterion (gini)**: function to measure the quality of a split {gini, entropy}
  - **splitter (best)**: strategy to choose split {best, random}
  - **max_features**: number of features when looking for best split
  - **max_depth**: maximum depth of tree
  - **min_samples_split**: minimum number of samples required to split
  - **min_impurity_split**: node will split if its impurity is above this threshold for early stopping in tree growth
  - **random_state**: seed for initial random numbers for coefficients.

12개 parameters …

Iris Decision Tree Classifier (All-in-One Code) [1/3]

- Requirements: pydotplus, sklearn, GraphViz
- Load data and train decision tree classifier

```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.externals.six import StringIO
import pydotplus

iris = load_iris()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)
```

sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
6.3	3.3	6	2.5	2
5.8	2.7	5.1	1.9	2
7.1	3	5.9	2.1	2

Original iris dataset

- **StringIO** module implements file-like class '**StringIO**' that reads and writes a string buffer
- **Pydotplus** module provides Python interface to create, handle, modify and process graph in Graphviz DOT language

clf

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

12개 parameters

clf.predict(array[[4.8, 3.1, 1.5, 0.2]])
[0]

2D Numpy object 로 NewComer를 표현해야

Iris Decision Tree Classifier : Drawing [2/3]

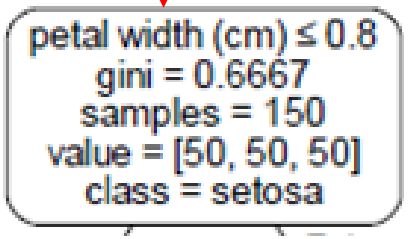
Export classifier **clf**'s information to **dot_data** (StringIO instance) in Graphviz DOT language

Graphviz의 dot language code 결과물이 dot_data에 write된다

```
In [104]: dot_data = StringIO()
          tree.export_graphviz(clf, out_file=dot_data,
                               feature_names=iris.feature_names,
                               class_names=iris.target_names,
                               filled=True, rounded=True,
                               special_characters=True)

          dot_data.getvalue()
```

iris table의 heading 정보

```
Out[104]: 'digraph Tree {\nnode [shape=box, style="filled, rounded", color="black", fontn
          ame=helvetica] ;\nedge [fontname=helvetica] ;\n0 [label=<petal width (cm) &le;
          0.8<br/>gini = 0.6667<br/>samples = 150<br/>value = [50, 50, 50]<br/>class = se
          tosa>, fillcolor="#e5813900"] ;\n1 [label=<gini = 0.0<br/>samples = 50<br/>valu
```



Decision tree의 특정 node

Transform DOT language code file to graph with pydotplus module's function

- Transform DOT data to graph format & write the result as a PDF file

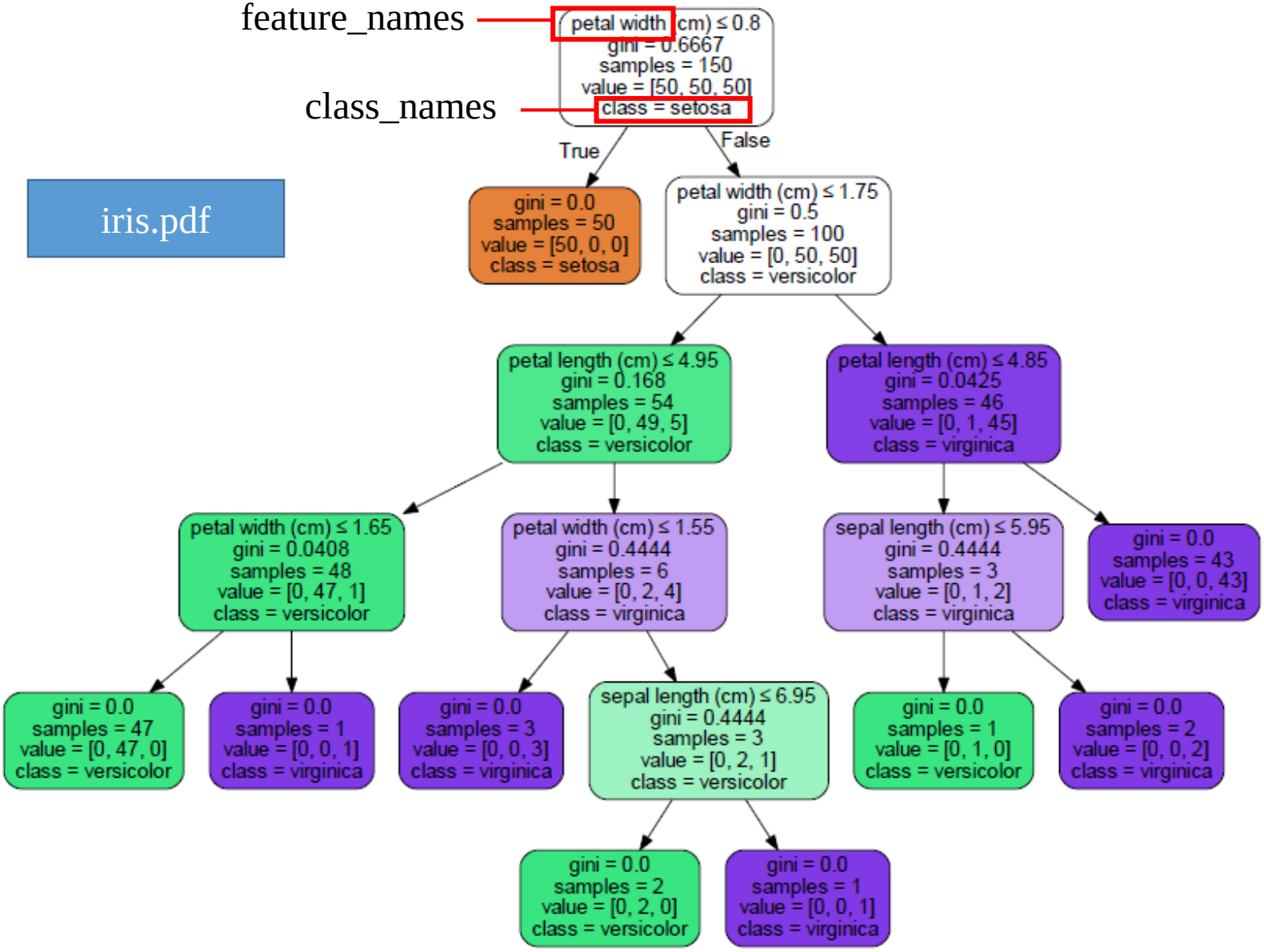
```
In [106]: graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
          graph.write_pdf('iris.pdf')
```

Iris Decision Tree Classifier : Drawing [3/3]

feature_names

class_names

iris.pdf



Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbor (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

Conditional Probability (조건부 확률)

- 사건 A가 주어졌을 때 사건 B의 조건부확률은 $P(B|A)$로 나타내고 $P(A) > 0$이라는 가정하에 다음과 같이 정의

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

P(B | A) = P(A ∩ B) / P(A)
P(A | B) = P(B ∩ A) / P(B)
P(B | A) * P(A) = P(A | B) * P(B)
P(B | A) = P(A | B) * P(B) / P(A)

- 사건 A를 새로운 축소된 표본공간으로 간주했을때, 사건 B가 일어날 확률
- 3개의 동전을 차례로 던지는 경우, 앞면이 나온수가 2번일때 (사건 A) , 첫번째 던지기가 앞면 (사건 B) 이 나올 확률은 ?
  - P(B|A) = P(A | B) * P(B) / P(A)
    - P(A) ➔ P(HHT, HTH, THH) ➔ (½ * ½ * ½) * 3 ➔ 3/8
    - P(A | B) ➔ P (H가 나온상황에서 HHT or HTH 인 경우) ➔ ½ * (1/8 * 2) ➔ 1/8
    - P(B) ➔ 1/2
    - P(A | B) * P(B) / P(A) ➔ (1/8) * (1/2) / (3/8) ➔ 1/6

Prior Probability vs Posterior Probability

- Prior Probability (사전확률)
  - 확률시행전에 이미 가지고 있는 지식을 통해 정해지는 확률
  - 예, 동전을 던져서 앞면이 나올확률은 ½
- Posterior Probability (사후확률)
  - 사건발생후에 추가된정보로 결정되는 확률
  - Bayes Theorem으로 얻어지는 확률

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

- Likelihood (우도, 가능도) : **직관적인 정의 → 확률분포함수의 y값**
  - **셀 수 있는 사건: 가능도 = 확률**
  - **연속 사건: 가능도 ≠ 확률, 가능도 = PDF값**
- Evidence (증거, 관찰값)

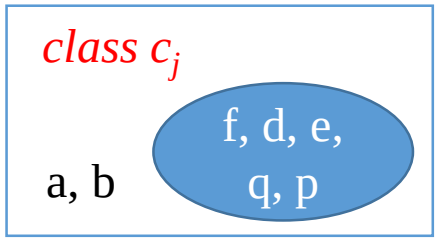
$$P(A \mid B) * P(B) = P(B \mid A) * P(A)$$
$$P(A \mid B) = P(B \mid A) * P(A) / P(B)$$

- P(A | B) = B가 관측됐을때 그 원인이 A일 조건부 확률 (Posterior)
- P(B | A) = A하에서의 B의 발생 조건부확률 (Likelihood)
- P(A) = A의 사전 확률 (Prior)
- P(B) = B의 증거 (Evidence)
  - 사후확률의 계산에는 필요한 값이나, 추론/결정/판정에 영향을 미치지 않는 정규화된 상수로 취급됨

Concept of Bayesian Classifiers

- Bayes' theorem is named after Rev. Thomas Bayes (1701–1761), who first provided an equation that allows new evidence to update beliefs
- Bayesian classifiers use **Bayes theorem**, which is based on Conditional Probability

$$p(c_j \mid d) = \frac{p(d \mid c_j)\, p(c_j)}{p(d)}$$



d가 존재할때 class c_j 에 있을 확률
= class c_j 가 존재할때 d 가 포함될 확률

where $p(c_j \mid d)$ = probability of instance d being in class c_j,

$p(d \mid c_j)$ = probability of generating instance d given class c_j,

$p(c_j)$ = probability of occurrence of class c_j

$p(d)$ = probability of instance d occurring

- The class with the maximum probability becomes the predicated class of instance d

- Bayesian classifiers require
  - computation of $p(d \mid c_j)$
  - precomputation of $p(c_j)$
  - $p(d)$ can be ignored since it is the same for all classes

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

Example of Bayesian Classifiers

- Example: Suppose there are 2 credit categories: Excellent and Good
  - The following facts are given for people having the income range (75000 ~ 80000)
    - Suppose the ratio of "Excellent" in the income range is 0.1 → $p(d \mid c_j) = 0.1$
    - Suppose the ratio of "Good" in the income range is 0.05 → $p(d \mid c_j) = 0.05$
    - Suppose the 10 % of person are classified as Excellent → $p(c_j) = 0.1$
    - Suppose the 30% of person are classified as Good → $p(c_j) = 0.3$
  - Given d : John's income is 76000
  - What is the credit category for John?
    - $p(d \mid c_j) * p(c_j)$ for class Excellent is : 0.1 X 0.1 ➔ 0.01
    - $p(d \mid c_j) * p(c_j)$ for class Good is : 0.05 X 0.3 ➔ 0.015
    - So, John would be more likely classified in class Good

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

class 가 여러개 있을 때 d 가 각각의 class 에 속할 확률을 계산해서 가장 큰 확률에 해당하는 class를 답으로 찾는 방식

Naïve Bayesian Classifiers

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

Salary	Class
$10K	A
$12K	A+
$9K	A
$3K	B
$13	A+

Salary	House	Class
$10K	Yes	A
$12K	Yes	A+
$9K	Yes	A
$3K	No	B
$13	No	A+

Number값 Boolean 값

- *Suppose d consists of attributes d_1 d_2 …. d_n*
- To simplify the task, **naïve Bayesian classifiers** assume attributes have independent distributions, and thereby estimate

$$p(d \mid c_j) = p(d_1 \mid c_j) * p(d_2 \mid c_j) * \ldots * p(d_n \mid c_j)$$

  - Bayes' theorem with the "naïve" assumption of independence between every pair of features
  - Each of the $p(d_i \mid c_j)$ can be estimated from a histogram on d_i values for each class c_j
    - the histogram is computed from the training instances

Intuition: Naïve Bayesian Classifiers

< 문제점 > Bayesian Classifier 는 Hypothesis에 완벽하게 matching되는 record들이 존재하여야만 classification을 할수 있다

X1	X2	X3	Y
α1	β	δ2	1
α	β2	δ	0
α3	β1	δ3	1

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

Bayes' theorem으로 (α, β, δ, ?)를 추론하려면 trained data에 (α, β, δ, 1) 이나 (α, β, δ, 0)같은 tuple 들이 있어야 위의 공식에 따라 계산을 해서 높은 확률값을 가지는 class를 확정하는데…

- Naïve Bayes Classifier: 기존 Bayes 방식에서 attribute들간에 결합하는 확률이 서로 독립 (independence)라는 가정에 의존하므로 Hypothesis 에 정확히 matching되는 record가 없어도 추론이 가능

(α, β, δ, ?)를 추론을 Naïve Bayes Classifier로 하면

$P(X_1=\alpha, X_2=\beta, X_3=\delta \mid y=1) = P(X_1=\alpha \mid y=1) * P(X_2=\beta \mid y=1) * P(X_3=\delta \mid y=1)$

Naïve Bayesian Classifier: Computer Sales [1/3]

Class:

C1:buys_computer = 'yes'

C2:buys_computer = 'no'

Data sample

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

X 는 computer 를 살것인가 안살것인가?

age	income	student	Credit _Rating	Buy_ Computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no
30	**medium**	**yes**	**fair**	**???**

Category값 Boolean 값

Naïve Bayesian Classifier: Computer Sales [2/3]

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**
  - $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
    P(buys_computer = "no") = 5/14= 0.357
  - Compute $P(X|C_i)$ for each class (Naïve Bayesian)

P(x1| C1) / P(x1| C2):
P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6

P(x2| C1) / P(x2| C2):
P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

P(x3| C1) / P(x3| C2):
P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

P(x4| C1) / P(x4| C2):
P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028 **
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

Naïve Bayesian Classifier: Computer Sales [3/3]

age	income	student	Credit _Rating	Buy_ Computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Category값 Boolean 값

- Learning
  - Class Ci = { c1, c2}
  - Attribute X = { x1, x2, x3, x4 }
  - 각 class별로 P(Ci) 계산
  - 각 attribute 의 discrete 값 (di) 별로 P(di | Ci) 계산

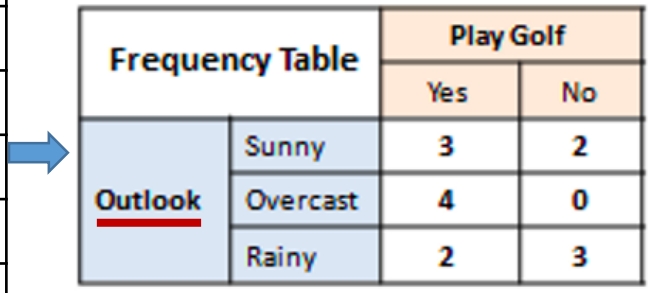
- Prediction
  - N (v1, v2, v3, v4) 는 Ci 중 어디에 속할것인가?
    P(Ci | N) == P(N | Ci) * P(Ci)

P(N | c1) ➔ P(v1 | c1) * P(v2 | c1) * P(v3 | c1) * P(v4 | c1)
P(N | c2) ➔ P(v1 | c2) * P(v2 | c2) * P(v3 | c2) * P(v4 | c2)

Naïve Bayes Classifier: Golf Playing [1/3]

Weather dataset

Outlook	Temp	Humidity	windy	Play Golf
Windy	Hot	High	False	No
Windy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



Frequency Table		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

Feature (column) 별로 frequency table을 생성!

Naïve Bayes Classifier: Golf Playing [2/3]

- Calculating the posterior probability (사후 확률)
  - Constructing a frequency table for each attribute against the target
  - Then, transforming the frequency tables to likelihood tables
  - Finally use the Naive Bayesian equation to calculate the posterior probability for each class
  - The class with the highest posterior probability is the outcome of prediction

Frequency Table		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

Likelihood Table		Play Golf		
		Yes	No	
Outlook	Sunny	3/9	2/5	5/14
	Overcast	4/9	0/5	4/14
	Rainy	2/9	3/5	5/14
		9/14	5/14	

$P(x \mid c) = P(Sunny \mid Yes) = 3/9 = 0.33$

$P(x \mid c) = P(sunny \mid No) = 2/5 = 0.4$

$P(x) = P(Sunny) = 5/14 = 0.36$

$P(c) = P(No) = 5/14 = 0.36$

$P(c) = P(Yes) = 9/14 = 0.64$

Posterior Probability:

$P(c \mid x) = P(Yes \mid Sunny) = 0.33 \times 0.64 \div 0.36 = 0.60$

$P(c \mid x) = P(No \mid Sunny) = 0.40 \times 0.36 \div 0.36 = 0.40$

Naïve Bayes Classifier: Golf Playing [3/3]

- Prediction

Outlook	Temp	Humidity	windy	Play Golf
Rainy	Cool	High	True	?

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

$$p(d \mid c_j) = p(d_1 \mid c_j) * p(d_2 \mid c_j) * \ldots * p(d_n \mid c_j)$$

$$P(Yes \mid X) = P(Rainy \mid Yes) \times P(Cool \mid Yes) \times P(High \mid Yes) \times P(True \mid Yes) \times P(Yes)$$

$$P(Yes \mid X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.00529$$

$$P(No \mid X) = P(Rainy \mid No) \times P(Cool \mid No) \times P(High \mid No) \times P(True \mid No) \times P(No)$$

$$P(No \mid X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.02057$$

➡ *Play Golf* ➔ *No*

- The final posterior probabilities can be standardized between 0 and 1
  - Standardized Probability of Yes

$$0.2 = \frac{0.00529}{0.02057 + 0.00529}$$

  - Standardized Probability of No

$$0.8 = \frac{0.02057}{0.02057 + 0.00529}$$

Pros/Cons of Naïve Bayes Classifiers

- Naïve Bayes classifiers have worked quite well in many real-world situations
  - Document classification, Spam filtering
  - They require a small amount of training data to estimate the necessary parameters
- *Pros*
  - Computationally fast & Simple to implement
  - Works well with small datasets
  - Perform well even if the *Naive Assumption* is not perfectly met
- *Cons*
  - Require to remove correlated features (assumption of independent features!)
    - because they are voted twice in the model and it can lead to over inflating importance
  - "Zero Frequency" Category
    - If a categorical variable has a category in test data set which was not observed in training data set, then the model will assign a zero probability
    - Not be able to make a prediction

Naïve Bayes Classifiers (at naïve_bayes SubModule)

- `sklearn.naive_bayes` module
  - Supervised learning methods based on applying Bayes' theorem with strong (naïve) feature independence assumptions
  - The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$
  - `GaussianNB([priors])` class
    - Gaussian Naive Bayes (GaussianNB)
    - 모든 feature (column) 이 Gaussian 분포를 따르는 Data이던지 혹은 따른다고 가정을 해야 이 방법을 쓸수 있다!
  - `BernoulliNB([alpha, binarize, …])` class
    - Naive Bayes classifier for multivariate Bernoulli models
    - 모든 feature (column) 이 Bernoulli 분포를 따르는 Data이던지 혹은 따른다고 가정을 해야 이 방법을 쓸수 있다!
  - `MultinomialNB([alpha, …])` class
    - Naive Bayes classifier for multinomial models
    - 모든 feature (column) 이 Categorial Data일때 이 방법을 쓸수 있다!

Feature들이 다양한 분포를 가진 경우에는 User Coding 해야함

Gaussian Naïve Bayes

- 데이터들이 실수이고 연속적인 값을 지닌 실수이고 특정값주변에서 발생한다라고 가정
- 전형적으로 각 클래스의 연속적인 값들이 Gaussian Distribution를 따른다고 가정
- 예를 들어, 트레이닝 데이터가 연속적인 속성을 포함하는 것으로 가정하면, 먼저 클래스에 따라 데이터를 나눈 뒤에, 각 클래스에서의 평균과 분산을 계산한다
- 클래스 *c* 와 연관된 x의 평균을 μ_c 이라고 하고, 분산을 δ_c 라고 하자

- 주어진 클래스 *c* 의 확률분포는 (Prior p(c)) 정규분포식을 통해 아래와 같이 계산이 될수 있다

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

모든 feature (column) 이 Gaussian 분포를 따르는 Data이던지 혹은 따른다고 가정을 해야 이 방법을 쓸수 있다!

Bernoulli Naïve Bayes

- 다변수 베르누이 이벤트 모델에서, Feature들은 독립적인 Boolean value (이진 변수)이다
- 다항 모델의 특성벡터가 이벤트의 빈도수를 나타내는 반면, 이 모델은 이벤트 발생 여부를 나타내는 Boolean value 을 가진다
- 이진변수의 발생이 특성으로 사용되는 문서 분류 작업에 대하여 널리 이용된다
- 만일 x_i 가 어휘들 중 i번째 용어의 발생유무를 표현하는 부울값일 경우 주어진 클래스에 C_k대한 문서의 우도 (확률분포)는 다음 식으로 주어진다

$$p(\mathbf{x}|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}(1 - p_{ki})^{(1-x_i)}$$

모든 feature (column) 이 Bernoulli 분포를 따르는 Data이던지 혹은 따른다고 가정을 해야 이 방법을 쓸수 있다!

Gaussian Naïve Bayes Classifier Example

모든 feature (column)이 Gaussian 분포를 따르던가 혹은 따른다고 가정

Original iris dataset

sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
6.3	3.3	6	2.5	2
5.8	2.7	5.1	1.9	2
7.1	3	5.9	2.1	2

iris.data

iris.target

$$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$$

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>>
>>> from sklearn.naive_bayes import GaussianNB
>>>
>>> gnb = GaussianNB( )
>>> clf = gnb.fit(iris.data, iris.target)
>>> y_pred = clf.predict(iris.data)
>>>
>>> print("Number of mislabeled points out of a total %d points: %d"
          % (iris.data.shape[0], (iris.target != y_pred).sum()))
Number of mislabeled points out of a total 150 points: 6
```

$p(d \mid c_j)$ 와 $p(c_j)$ 를 계산

True의 갯수

```
>>> iris.target != y_pred
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
```

Bernoulli Naïve Bayes Classifier

- `sklearn.naive_bayes.BernoulliNB(**kwarg)`
  - **alpha**: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing)
  - **binarize**: Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors
  - **fit_prior**: Whether to learn class prior probabilities or not. If false, a uniform prior will be used
  - **class_prior**: Prior probabilities of the classes. If specified the priors are not adjusted according to the data

BernoulliNB Classifier: Example

- All features are binary such that they take only two values
  - e.g. a nominal categorical feature that has been one-hot encoded

```
 9 import numpy as np
10 from sklearn.naive_bayes import BernoulliNB
11
12 # Create three binary features
13 X = np.random.randint(2, size=(100, 3))
14 # Create a binary target vector
15 y = np.random.randint(2, size=(100, 1)).ravel()
16
17 print("   X:     y:",  "   X:      y:", \
18       "   X:     y:",  "   X:      y:" )
19
20 for i in range(25):
21       print(X[i], " ", y[i],   " " ,\
22             X[i + 25], " ", y[i + 25], " ",  \
23             X[i + 50], " ", y[i + 50], " ",  \
24             X[i + 75], " ", y[i + 75])
25
26 # Create Bernoulli Naive Bayes object
27 # with prior probabilities of each class
28 clf = BernoulliNB(class_prior=[0.25, 0.5])
29 # Train model
30 model = clf.fit(X, y)
31 print()
32 print("Test Data: [0, 0, 0]) ==> ", clf.predict([[0,0,0]]))
```

```
   X:     y:    X:      y:    X:      y:    X:      y:
[1 1 0]   0   [0 0 0]   1   [0 0 1]   1   [0 1 1]   1
[0 0 0]   1   [0 0 1]   1   [0 0 1]   0   [0 0 1]   1
[0 1 0]   0   [0 1 1]   1   [0 0 1]   1   [1 1 0]   0
[0 1 1]   0   [0 1 1]   0   [0 1 1]   1   [1 1 0]   1
[0 1 0]   0   [1 0 1]   0   [1 1 1]   0   [0 1 0]   0
[1 1 1]   1   [1 1 1]   1   [0 0 1]   0   [0 1 1]   1
[0 0 1]   0   [0 1 1]   0   [0 0 1]   1   [1 1 0]   1
[1 1 1]   0   [1 0 1]   0   [0 0 1]   0   [1 1 1]   1
[1 0 1]   1   [1 1 1]   1   [1 0 1]   1   [1 0 0]   1
[0 1 1]   1   [0 1 1]   1   [1 0 1]   1   [1 0 1]   0
[1 1 1]   0   [1 1 0]   1   [0 0 0]   0   [1 0 1]   0
[0 0 0]   1   [1 0 0]   1   [0 1 1]   0   [0 1 1]   1
[0 0 1]   1   [1 1 0]   1   [1 1 1]   0   [1 0 1]   1
[0 1 1]   1   [1 1 1]   0   [0 0 1]   0   [0 1 0]   1
[1 0 0]   0   [1 1 1]   0   [1 0 1]   1   [1 1 0]   0
[0 0 0]   0   [0 1 1]   1   [0 1 1]   1   [0 1 0]   0
[1 1 1]   1   [0 1 0]   1   [1 0 1]   1   [1 0 0]   0
[1 0 1]   1   [1 1 0]   0   [1 1 1]   1   [1 1 1]   0
[0 1 1]   0   [1 1 0]   0   [0 0 0]   0   [1 1 0]   0
[1 1 0]   1   [1 1 0]   1   [0 0 1]   0   [0 0 0]   0
[1 1 1]   0   [1 1 0]   1   [0 0 1]   0   [0 1 1]   1
[0 1 1]   1   [0 0 1]   0   [0 1 1]   1   [0 1 0]   0
[1 1 0]   1   [1 0 0]   1   [1 0 0]   0   [0 0 1]   0
[0 0 1]   0   [0 0 1]   1   [0 1 0]   0   [0 0 1]   1
[0 0 0]   0   [0 1 1]   0   [0 0 1]   1   [0 0 1]   1

Test Data: [0, 0, 0]) ==>  [1]
```

Priors: P(c0) ➔ 0.25, P(c1) ➔ 0.5

$p(c_j \mid d) = p(d \mid c_j)\, p(c_j)$

$p(d \mid c_j)$ 와 $p(c_j)$ 를 계산

[0,0,0] 소속은 c0 or c1?

skip

Multinomial Naïve Bayes Classifier

- `sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)`
  - **alpha**: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing)
  - **fit_prior**: Whether to learn class prior probabilities or not. If false, a uniform prior will be used
  - **class_prior**: Prior probabilities of the classes. If specified the priors are not adjusted according to the data

skip

MultinomialNB classifier: Example [1/4]

* 20 newsgroups dataset: A collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups

```
>>> # Work on a partial dataset with only 4 categories
>>> categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
>>>
>>> # Load the list of files matching those categories
>>> from sklearn.datasets import fetch_20newsgroups
>>> twenty_train = fetch_20newsgroups(subset = 'train', categories = categories, shuffle
= True, random_state = 42)
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)


>>> twenty_train.target_names
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
>>> len(twenty_train.data)
2257
>>> # Print the first lines of the first loaded file
>>> print("\n".join(twenty_train.data[0].split("\n")[:2]))
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
>>>
>>> twenty_train.target[:10]
array([1, 1, 3, 3, 3, 3, 3, 2, 2, 2])
```

0 = alt.atheism
1 = comp.graphics
2 = sci.med
3 = soc.religion.christian

skip

MultinomialNB classifier: Example [2/4]

- Extracting features from text files

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> count_vec = CountVectorizer()
>>> X_train_counts = count_vec.fit_transform(twenty_train.data)
>>> X_train_counts.shape
(2257, 35788)
>>> count_vec.vocabulary_.get('algorithm')
4690
```

* `sklearn.feature_extraction.text.CountVectorizer(**kwargs)`

: convert a collection of text documents to a matrix of token counts

- **`fit_transform(raw_documents[,y])`**: learn the vocabulary dictionary and return term-document matrix
- **`vocabulary_`**: a mapping of terms to feature indices

skip

MultinomialNB classifier: Example [3/4]

- Occurrence count has an issue
  - Longer documents will have higher average count values than shorter documents, even though they might talk about the same topics
- TF-IDF (Term Frequency – Inversed Document Frequency)
  - A numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus

```
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> tfidf_transformer = TfidfTransformer()
>>> X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
>>> X_train_tfidf.shape
(2257, 35788)
```

* `sklearn.feature_extraction.text.TfidfTransformer(**kwargs)`

: transform a count matrix to a normalized tf or tf-idf representation

- **`fit_transform(X[,y])`**: fit to data, then transform it

skip

MultinomialNB classifier: Example [4/4]

- Training a classifier

```
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)
```

- Predict the outcome on a new document
  - Call `transform()` instead of `fit_transform()`, since transformers have already been fit to the training set

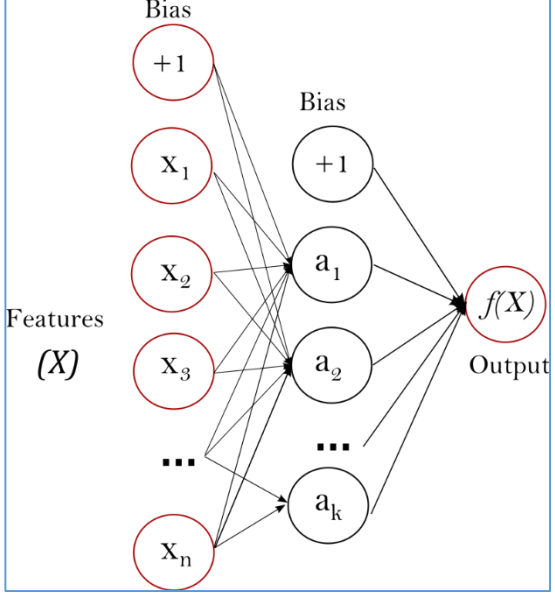
```
>>> docs_new = ['God is love', 'OpenGL on the GPU is fast']
>>> X_new_counts = count_vec.transform(docs_new)
>>> X_new_tfidf = tfidf_transformer.transform(X_new_counts)
>>>
>>> predicted = clf.predict(X_new_tfidf)
>>>
>>> for doc, category in zip(docs_new, predicted):
        print('%r => %s' % (doc, twenty_train.target_names[category]))


'God is love' => soc.religion.christian
'OpenGL on the GPU is fast' => comp.graphics
```

Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbor (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

Neural Network

- (Artificial) Neural Network
- Concept was driven from 1940s
- Not enough computation power
  - AI Winter



1958 Perceptron

1969 Perceptron criticized

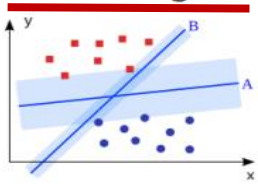
1974 Backpropagation

awkward silence (AI Winter)

Convolution Neural Networks for Handwritten Recognition

1998

1995 SVM reigns



2006 Restricted Boltzmann Machine

Google Brain Project on 16k Cores

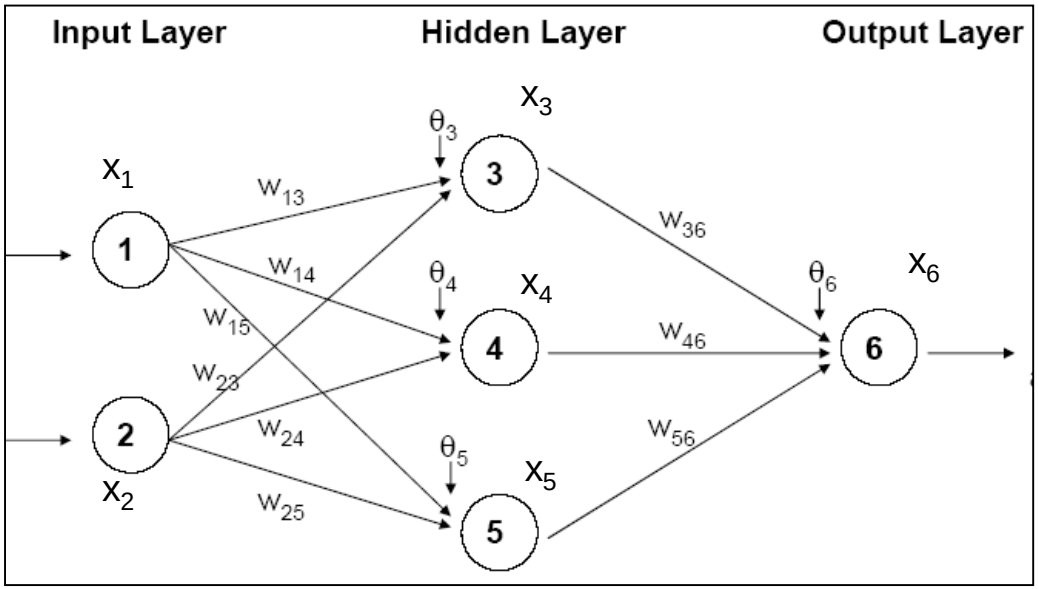
2012

2012 AlexNet wins ImageNet

IMAGENET

Neural-Net: Schematic Diagram – 3 layers

- input layer: input nodes = input or independent variables x
- output layer: output node = output or dependent variable y
- hidden layer: hidden nodes



θ_j: node bias value
w_{ij}: output from node ***i*** to node ***j***
x_i: input to node ***i***

$$output_j = g(\Theta_j + \sum_{i=1}^{p} w_{ij}\, x_i)$$

$$output_3 = g(\Theta_3 + w_{13}{*}x_1 + w_{23}{*}x_2)$$

$$output_4 = g(\Theta_4 + w_{14}{*}x_1 + w_{24}{*}x_2)$$

$$output_5 = g(\Theta_5 + w_{15}{*}x_1 + w_{25}{*}x_2)$$

$$output_6 = g(\Theta_6 + w_{36}{*}x_3 + w_{46}{*}x_4 + w_{56}{*}x_5)$$

Transfer Function g( )

- Hidden Layer와 Output Layer의 노드에서 하는 같은종류의 계산

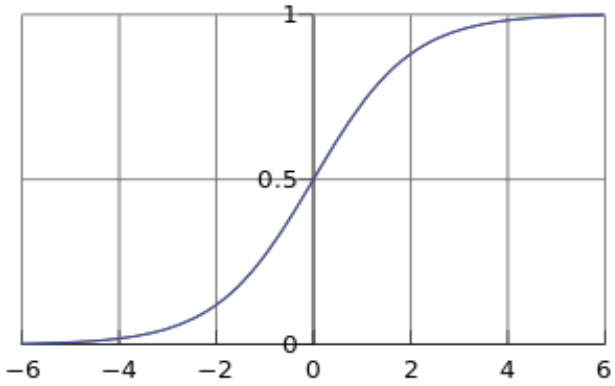
  뉴론의 활성화 함수 (activation function) 또는 학습 함수 (learning function)

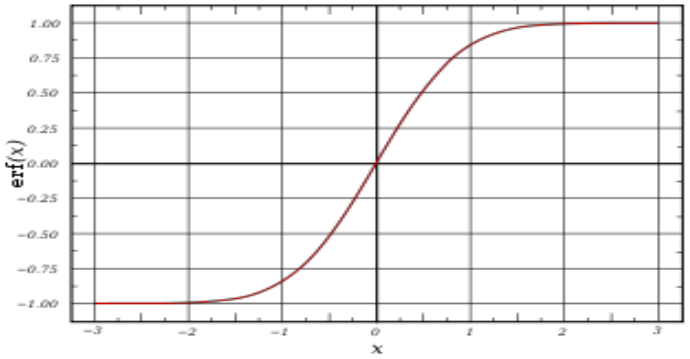
$$output_j = g(\Theta_j + \sum_{i=1}^{p} w_{ij}\, x_i)$$

θ_j: node bias value
w_{ij}: output from node ***i*** to node ***j***
x_i: input to node ***i***

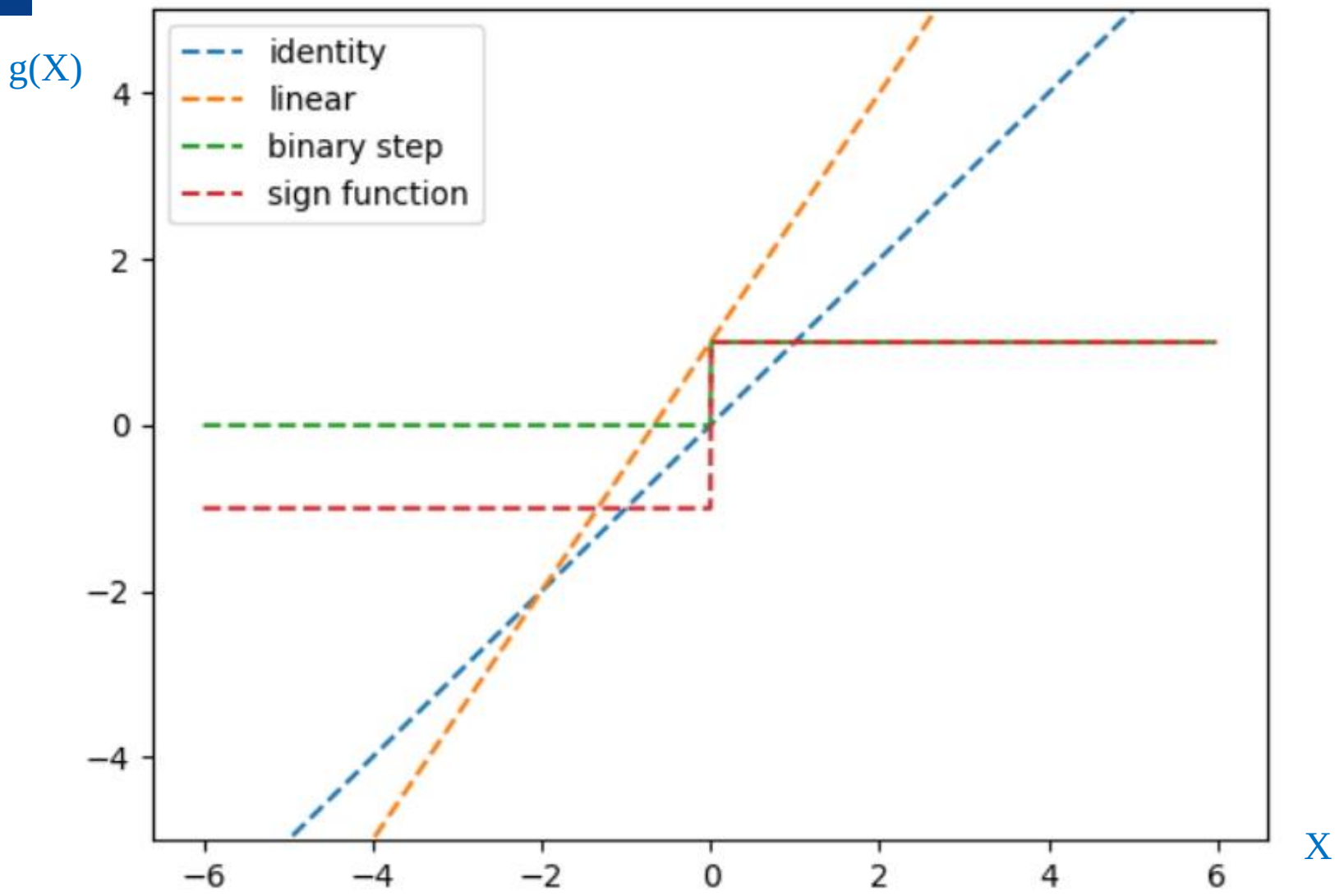
- If g is sigmoid function (= logistic function)
  g(x) = 1/(1+exp(−x))

Logistic function
Relu function
Identity function
Tanh function

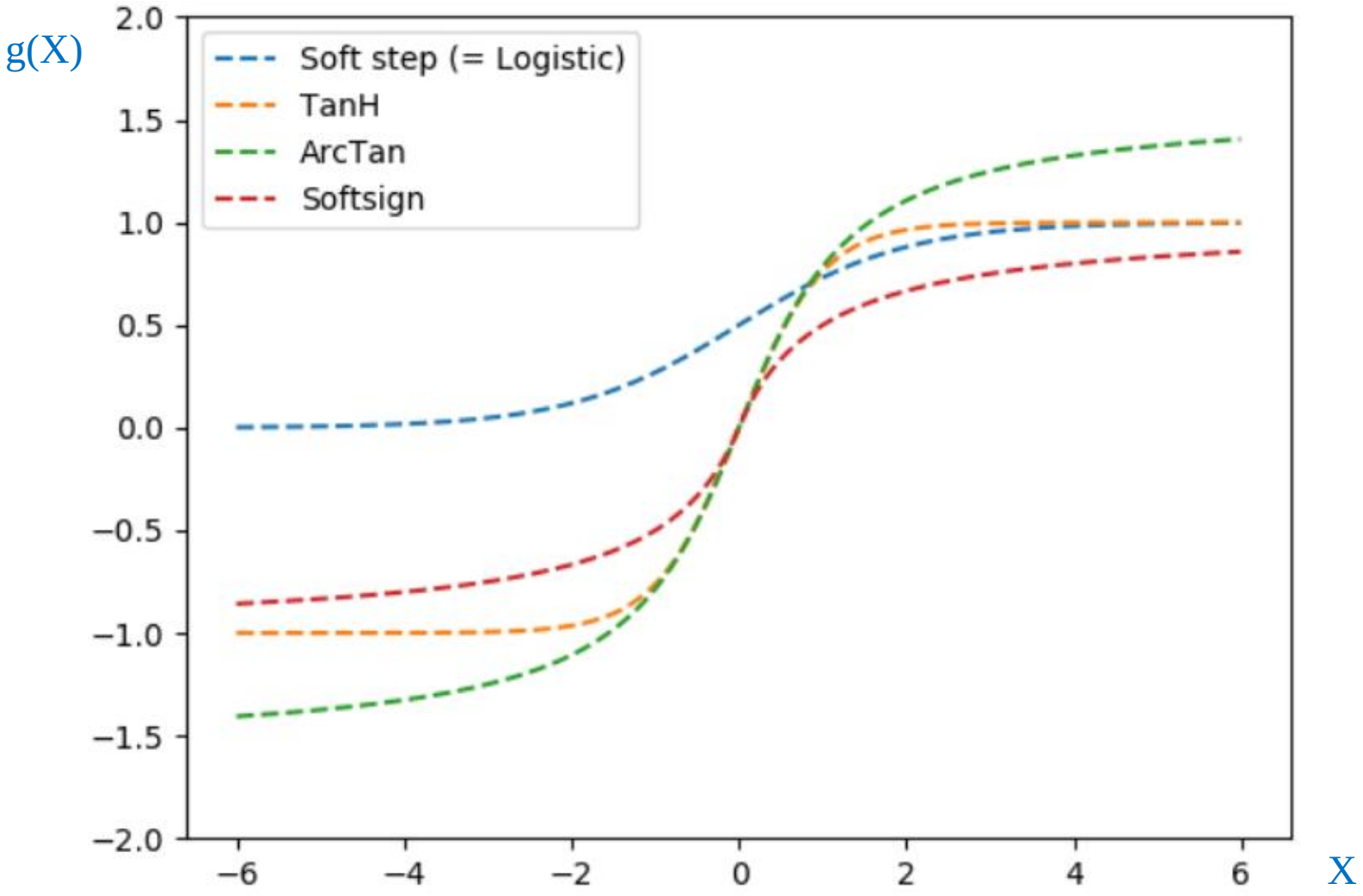




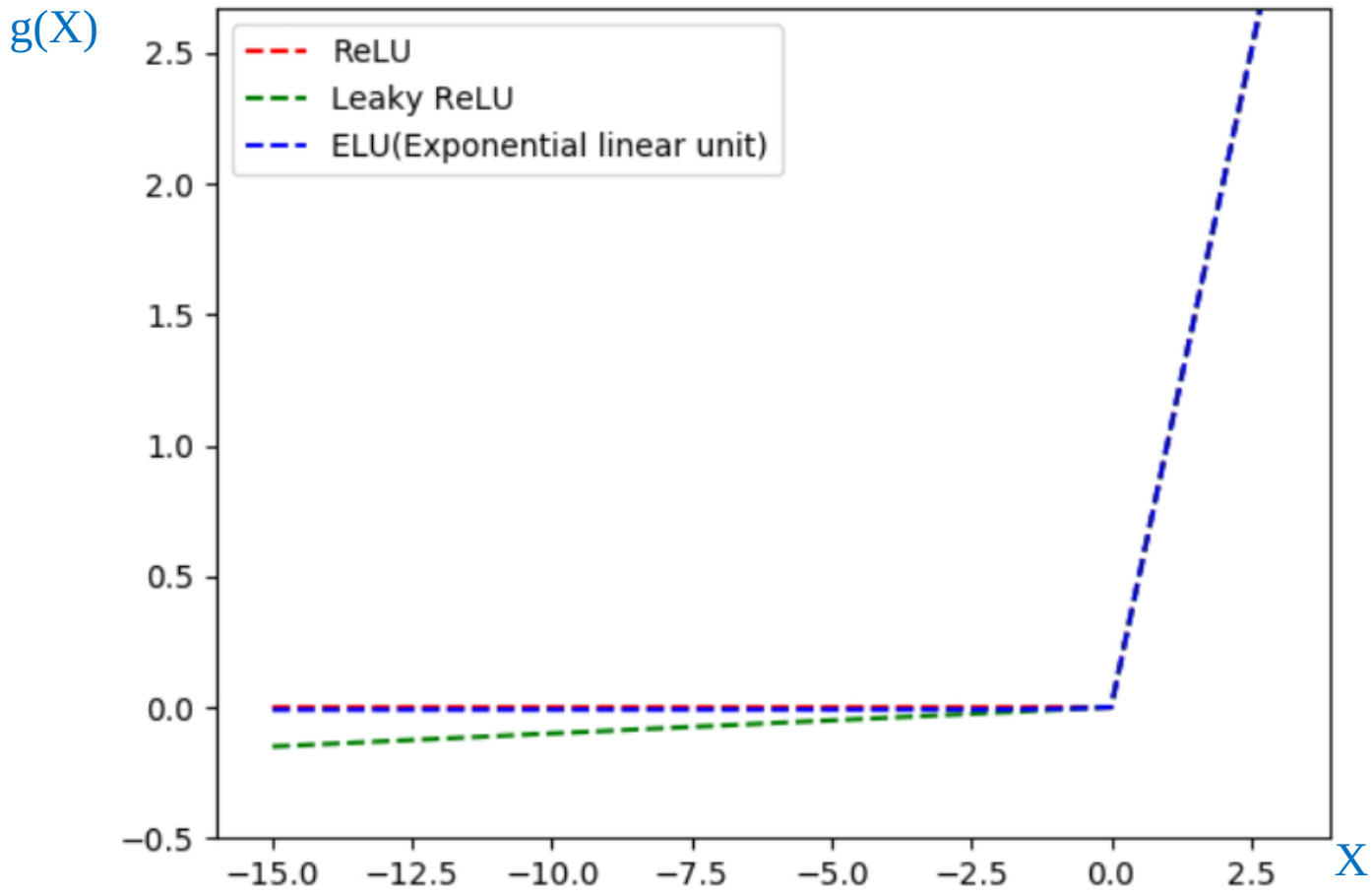
활성함수 (선형함수, 계단함수 계열) [1/3]



활성함수(Sigmoid 계열) [2/3]



활성함수(Relu계열) [3/3]



Neural Net Example

– Using Fat & Salt content to predict consumer acceptance of cheese

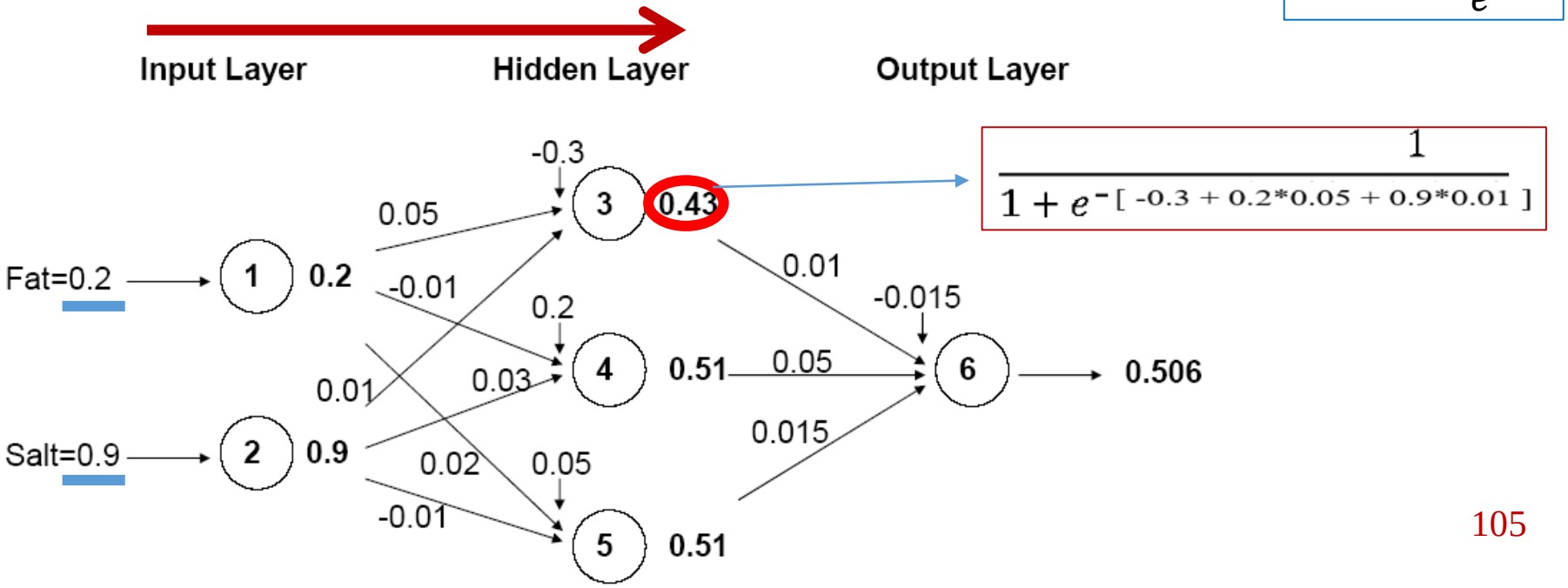
Training Data

Obs.	*Fat Score*	*Salt Score*	*Acceptance*
1	0.2	0.9	1
2	0.1	0.1	0
3	0.2	0.4	0
4	0.2	0.5	0
5	0.4	0.5	1
6	0.3	0.8	1

Initial Pass of the Network

처음상태에서 모든 edge의 weight값과 theta값은 사용자가 random하게 주어도 되고, 시스템이 random하게 정해주고 시작을 해도 된다. 대부분 처음에는 0.01과 같은 작은값들로 시작함.

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



Output Layer

- The output of the last hidden layer becomes input for the output layer
- Uses same function as above, i.e. a function g of the weighted average

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

$$output_6 = \frac{1}{1 + e^{-[-0.015 + (0.01)(0.43) + (0.05)(0.507) + (0.015)(0.511)]}} = 0.506$$

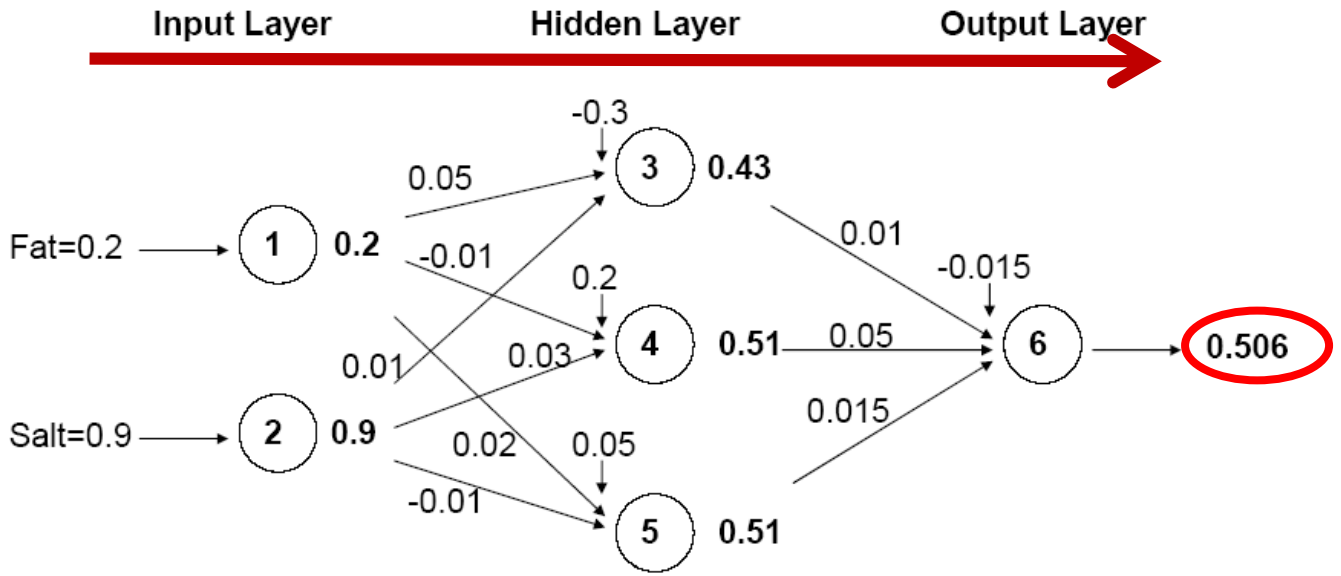
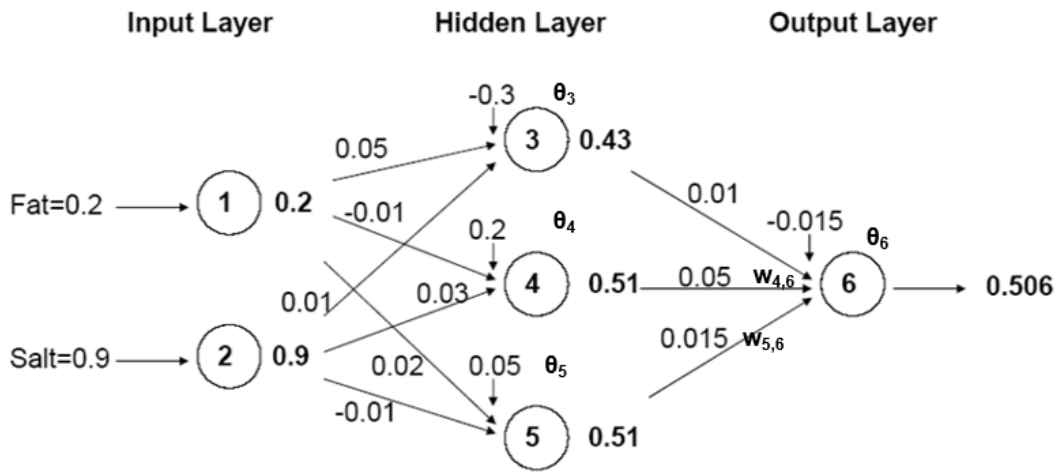


Figure 11.3: Computing node outputs (in boldface type) using the first observation in the tiny example and a logistic function

Error is used to Update Weights: Backpropagation [1/2]



Obs.	Fat Score	Salt Score	Acceptance
1	0.2	0.9	1
2	0.1	0.1	0
3	0.2	0.4	0
4	0.2	0.5	0
5	0.4	0.5	1
6	0.3	0.8	1

정답은 1

- Output from output node k: $\hat{y}_k$ 정답은 y_k
- Error associated with that node: $err_k = \hat{y}_k(1-\hat{y}_k)(y_k - \hat{y}_k)$

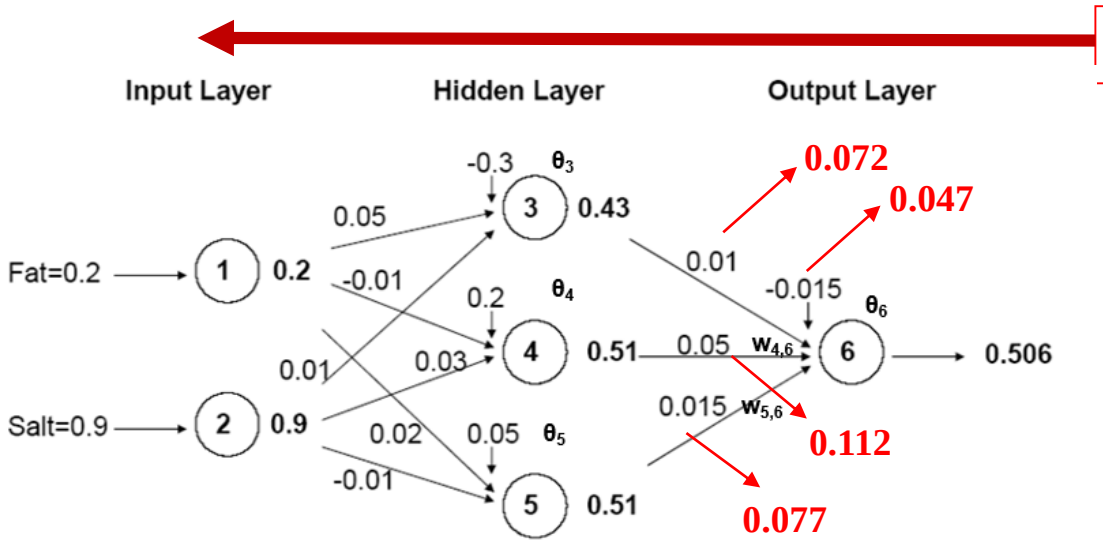
$$\theta_j^{new} = \theta_j^{old} + l(err_j)$$

$$w_j^{new} = w_j^{old} + l(err_j)$$

l = a constant between 0 and 1, reflects the "learning rate" or "weight decay parameter"
(값은 designer가 결정)

- 1에 가깝게 정하면 error를 중요하게 보는것이고 0에 가깝게 정하면 error를 가볍게 보는것

Error is used to Update Weights: Backpropagation [2/2]



Obs.	Fat Score	Salt Score	Acceptance
1	0.2	0.9	1
2	0.1	0.1	0

정답은 1

$$err_k = \hat{y}_k(1-\hat{y}_k)(y_k - \hat{y}_k)$$

$$\theta_j^{new} = \theta_j^{old} + l(err_j)$$

$$w_j^{new} = w_j^{old} + l(err_j)$$

y_k = 1
$\hat{y}_k$ = 0.506
err = 0.506(1-0.506)(1-0.506)
= 0.123

with learning rate (etha) η (or l) = 0.5

θ6 = -0.015 + 0.5 * 0.123 = **0.047**
w3,6 = 0.01 + 0.5 * 0.123 = **0.072**
w4,6 = 0.05 + 0.5 * 0.123 = **0.112**
w5,6 = 0.015 + 0.5 * 0.123 = **0.077**

MLPClassifier Class (at neural_network submodule)

- sklearn.neural_network.MLPClassifier (**kwargs)
  - **hidden_layer_sizes ((100,))**: tuple, set the structure of the network.
    i^{th} element represents the number of neurons
    in i^{th} hidden layer
  - **activation (relu)**: {identity, logistic, tanh, relu}
  - **solver (adam)**: solver for weight optimization. {lbfgs, sgd, adam}
  - **alpha**: L2 regularization term
  - **batch_size**: size of minibatches for stochastic optimizer
  - **max_iter**: maximum number of iteration. Iterate until convergence determined by '**tol**' or this number
  - **tol**: tolerance for optimization. Determine whether training step converges
  - **random_state**: seed for initial random numbers for coefficients.
  - **verbose**: whether to print progress messages
  - **momentum**: whether to use momentum for gradient descent update only when solver='sgd'

21 parameters…..

Neural-Net for MNIST: Training Data [1/6]

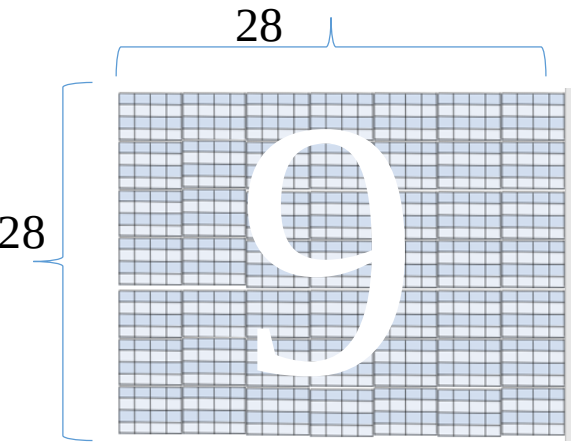
yann.lecun.com/exdb/mnist/

yann.lecun.com **THE MNIST DATABASE**

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

- MNIST original data set
  - Handwritten digits
  - 28x28 image data (= 784 pixels)
    - One pixel (8 bit) has a value in range (0 ~ 255)
  - 70,000 data (60,000 train data + 10,000 test data)



검은바탕에 하얀글씨

One image = Numpy 1D Array

array([0, 0, 0, …. 37, 42, 0, 0, 0, 49, 232, 220, …, 0, 0, 0] }

784 numbers

Neural-Net for MNIST: Data Loading [2/6]

- Import modules & load dataset

fetch_mldata( ) : fetch dataset from mldata.org.
If file does not exist, download from mldata.org

```
In [109]: %matplotlib inline
          import matplotlib.pyplot as plt
          from sklearn.datasets import fetch_mldata
          from sklearn.neural_network import MLPClassifier

          mnist_data = datasets.fetch_openml('mnist_784', cache=True)

          mnist
```

MultiLayerPerceptron Classifier

```
Out[109]: {'COL_NAMES': ['label', 'data'],
           'DESCR': 'mldata.org dataset: mnist-original',
           'data': array([[0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
           'target': array([ 0.,  0.,  0., ...,  9.,  9.,  9.])}
```

Dictionary-like object

← One image (784 pixels)

70000개 images

70000개 labels

Neural-Net for MNIST: Data Preprocessing [3/6]

- Rescale the value of dataset to [0, 1)
  - As each pixel has value of 0~255 (8 bits 정보), it should be divided by 255
  - Each pixel의 값이 0 ~ 1 사이의 값으로 normalize시킴

```
X, y = mnist.data / 255., mnist.target
```

- Separate training data set and test data set from X and y

```
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

mnist

Numpy 2D Array Numpy 1D Array

data	target
784 pixels	0
784 pixels	2
784 pixels	7
……	…..
784 pixels	6
784 pixels	1

70000개

One image (784 pixels) :

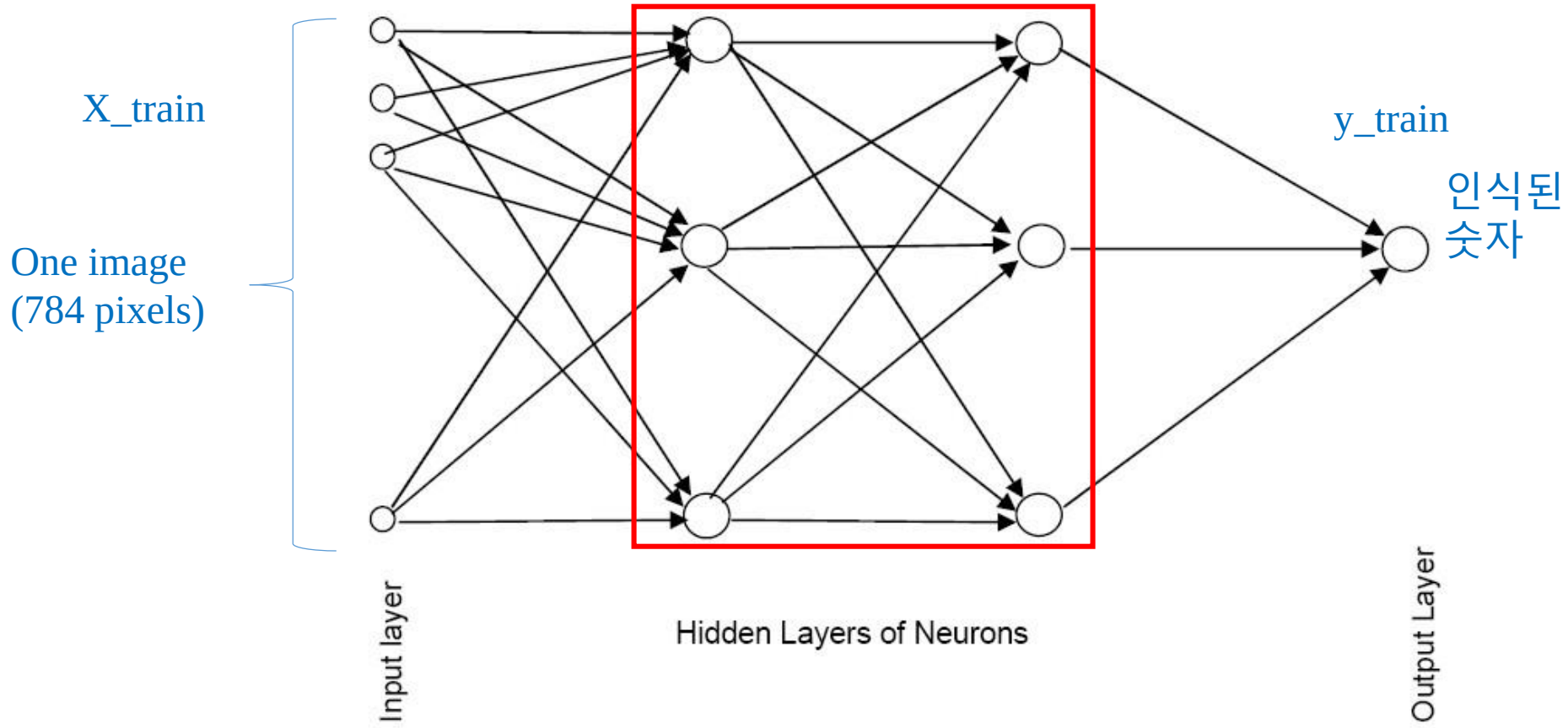
array([32/255, 39/255, 211/255, …… 11/255, 245/255])

➔

array([0.125, 0.152, 0.827, …… 0.043, 0.960])

Neural-Network Example: Data Preprocessing [4/6]

Multi-Layer-Perceptron Neural-Net Classifier For MNIST Database



One image (784 pixels) :
array([32/255, 39/255, 211/255, …… 11/255, 245/255])
➔
array([0.125, 0.152, 0.827, …… 0.043, 0.960])

Neural-Net for MNIST: Learning from Training Data [5/6]

- Train neural-network classifier

Hidden Layer의 Node숫자 = 50

```
In [120]: mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                              solver='sgd', verbose=10, tol=1e-4, random_state=1,
                              learning_rate_init=.1)

          mlp.fit(X_train, y_train)
```

Stochastic gradient descent

한 개의 input case에 대해서 forward – backward iteration을 10번이내로 수행

```
Iteration 1, loss = 0.32212731
Iteration 2, loss = 0.15738787
Iteration 3, loss = 0.11647274
Iteration 4, loss = 0.09631113
Iteration 5, loss = 0.08074513
Iteration 6, loss = 0.07163224
Iteration 7, loss = 0.06351392
Iteration 8, loss = 0.05694146
Iteration 9, loss = 0.05213487
Iteration 10, loss = 0.04708320
```

```
Out[133]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                        beta_2=0.999, early_stopping=False, epsilon=1e-08,
                        hidden_layer_sizes=(50,), learning_rate='constant',
                        learning_rate_init=0.1, max_iter=10, momentum=0.9,
                        nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
                        solver='sgd', tol=0.0001, validation_fraction=0.1, verbose=1,
                        warm_start=False)
```

21 parameters

Neural-Net for MNIST: Checking the Performance [6/6]

- Print the mean accuracy of training/test data sets

```
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

```
Training set score: 0.985733
Test set score: 0.971000
```

score (*X*, *y*, *sample_weight=None*)

R^2 = R square = R squared

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

The most general definition of the coefficient of determination is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

All-in-One Code (Neural Net Classifier)

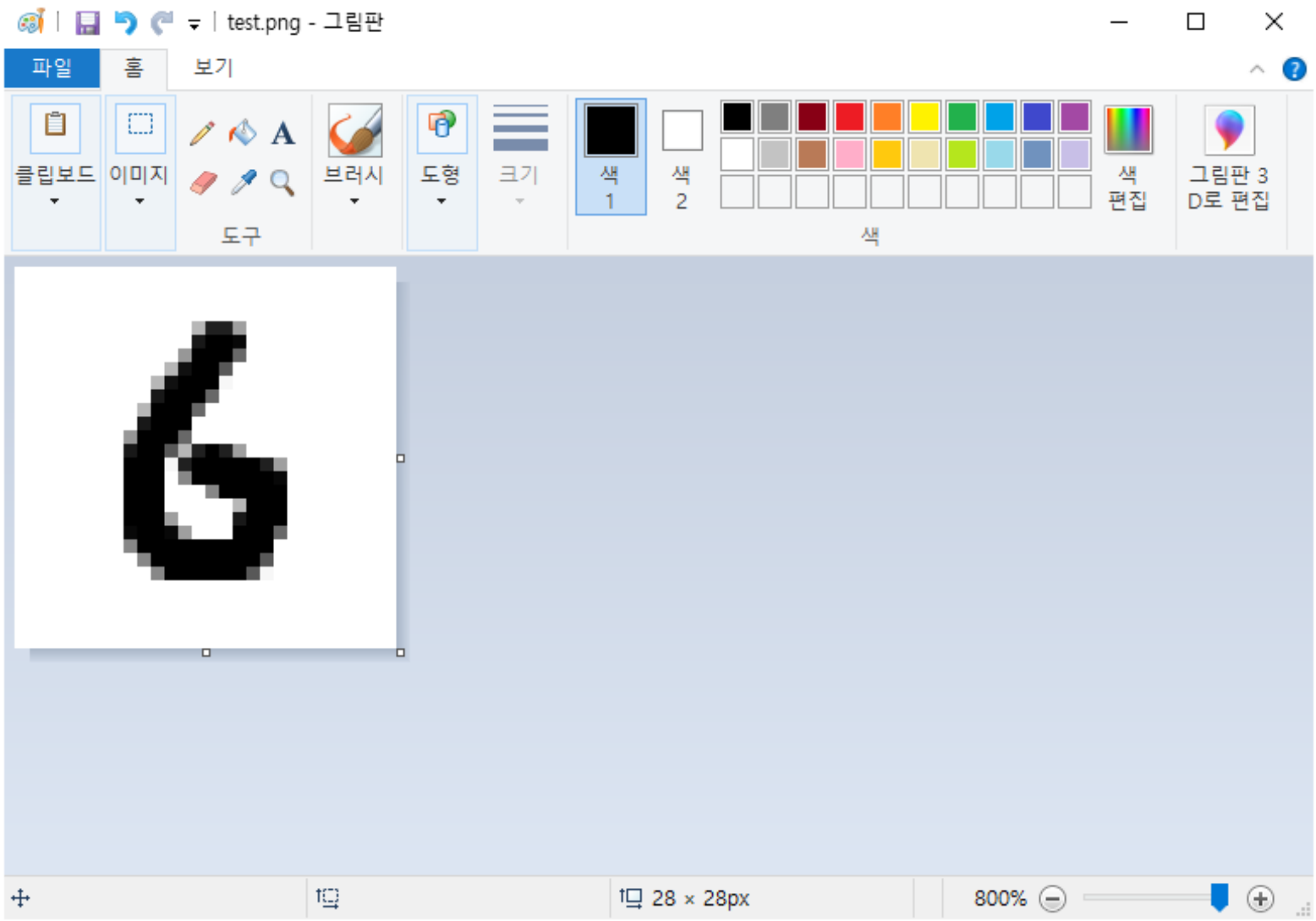
** fetch_openml( ) instead of fetch_mldata( )

One image (784 pixels) :
array([32/255, 39/255, 211/255, …… 11/255, 245/255])
➔
array([0.125, 0.152, 0.827, …… 0.043, 0.960])

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.neural_network import MLPClassifier



mnist_data = datasets.fetch_openml('mnist_784', cache=True)

X, label = mnist_data.data/255., mnist_data.target

X_train, X_test = X[:60000], X[60000:]
label_train, label_test = label[:60000], label[60000:]

mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1, learning_rate_init=.1)

mlp.fit(X_train, label_train)

print("Training set score: %f" %mlp.score(X_train, label_train))
print("Test set score: %f" %mlp.score(X_test, label_test))
```

손으로 쓴 글씨를 Test [1/2]

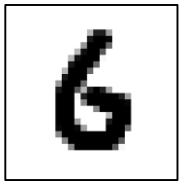
- 그림판에서 28*28 크기의 숫자 이미지를 test.png로 저장



손으로 쓴 글씨를 Test [2/2]

- opencv module: computer vision library
- opencv 설치:
  - Command 창에서 ➔ pip install opencv-python
- >>> import cv2
- cvtColor ( ) : convert color

Mnist



그림판

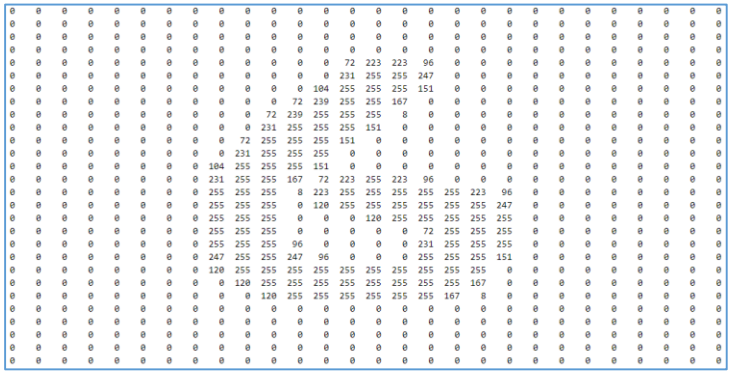
Mnist는 검은바탕에 하얀글씨를 가정

그림판은 하얀바탕에 검은글씨로 생성

- 아래 코드를 통해 그림을 입력 데이터로 변환

```
1   import cv2
2
3   im = cv2.imread("test.png")
4
5   im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
6
7   for i in range(len(im_gray)):
8       for j in range(len(im_gray[0])):
9           im_gray[i][j] = 255 - im_gray[i][j]
10
11  print(im_gray)
```

White ➔ Black
Blakc ➔ White



mlp.predict([im_grey]) ⟶ 6

Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbor (KNN) Classifier
- Decision Tree Classifier
- Bayesian Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

K-Means History

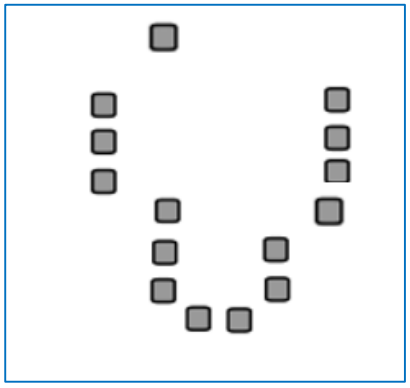
- History
  - 1956년에 프랑스에서 Hugo Steinhaus 에 의해 개념 등장
  - 1967년에 James MacQueen 에 의해 용어 등장
  - S. Lloyd가 1957년에 펄스 부호 변조를 위해 고안한 것을 1982년에 Publish
- References

  - Steinhaus, Hugo. "Sur la division des corp materiels en parties." *Bull. Acad. Polon. Sci* 1.804 (1956)
  - MacQueen, James. "Some methods for classification and analysis of multivariate observations." *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. No. 14. 1967.
  - *Lloyd, S. P. (1957). "Least square quantization in PCM". 《Bell Telephone Laboratories Paper》.*
  - *Lloyd., S. P. (1982). "Least squares quantization in PCM" (PDF). 《IEEE Transactions on Information Theory》* ***28*** *(2)*

K-Means Clustering [1/2]

- Unsupervised learning model
- Similar with K Nearest-Neighbor algorithm, assume that similar data will be located closely
- Based on such assumption, k-means algorithm aims to partition n data into k clusters
- Each observation belongs to the cluster with nearest centroid (mean)

Given Data

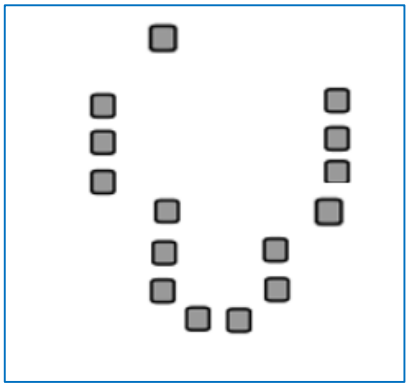


How to cluster the points?

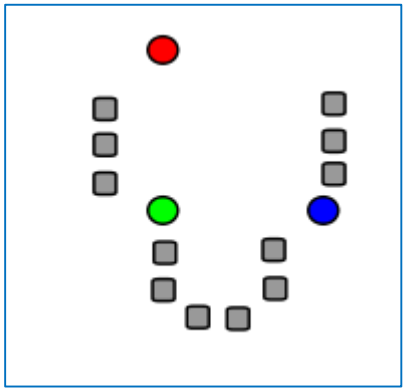
K-Means Clustering [2/2]

- Procedure

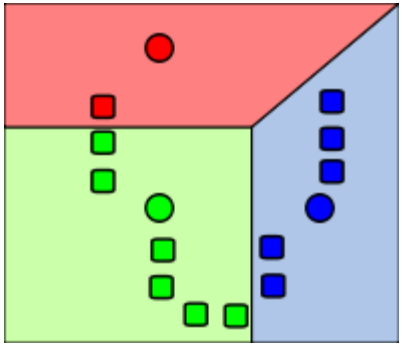
Given Data



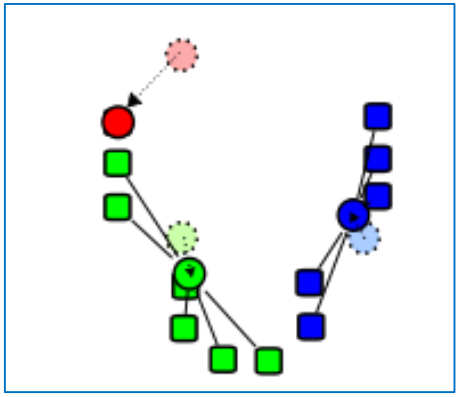
Step 1:
pick k centroids randomly



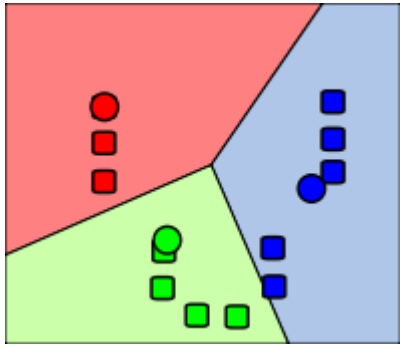
Step 2: Calculate distance between all data and centroids ; Assign each data point to the closest centroid's cluster



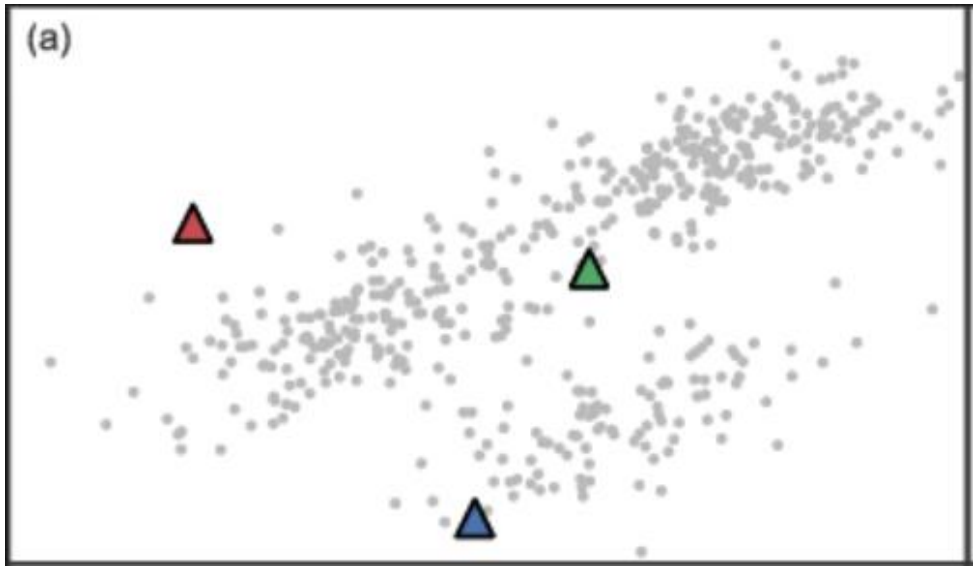
Step 3: pick the mean points as new k centroids



Step 4: repeat Step 2 & 3 until convergence

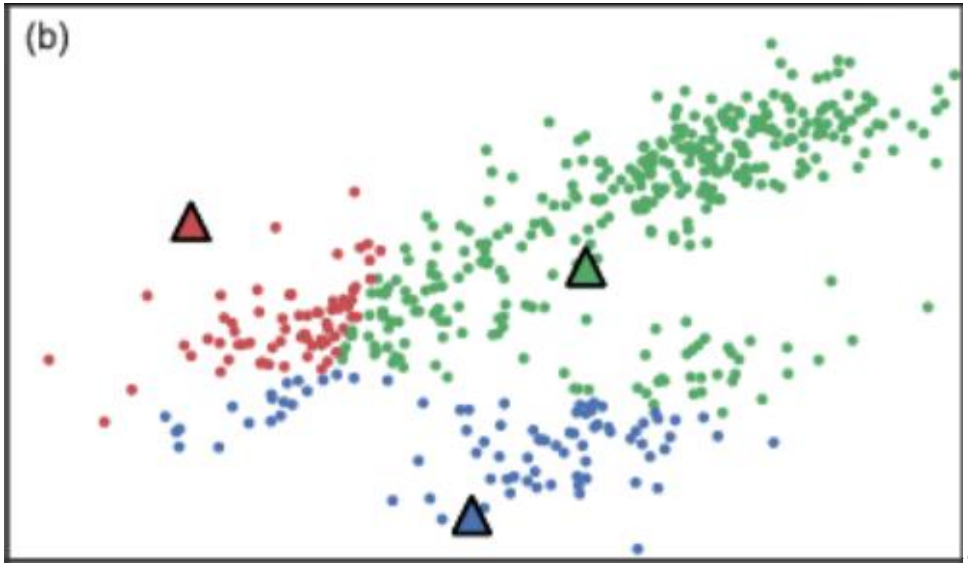


Step-by-Step K-means Example [1/3]



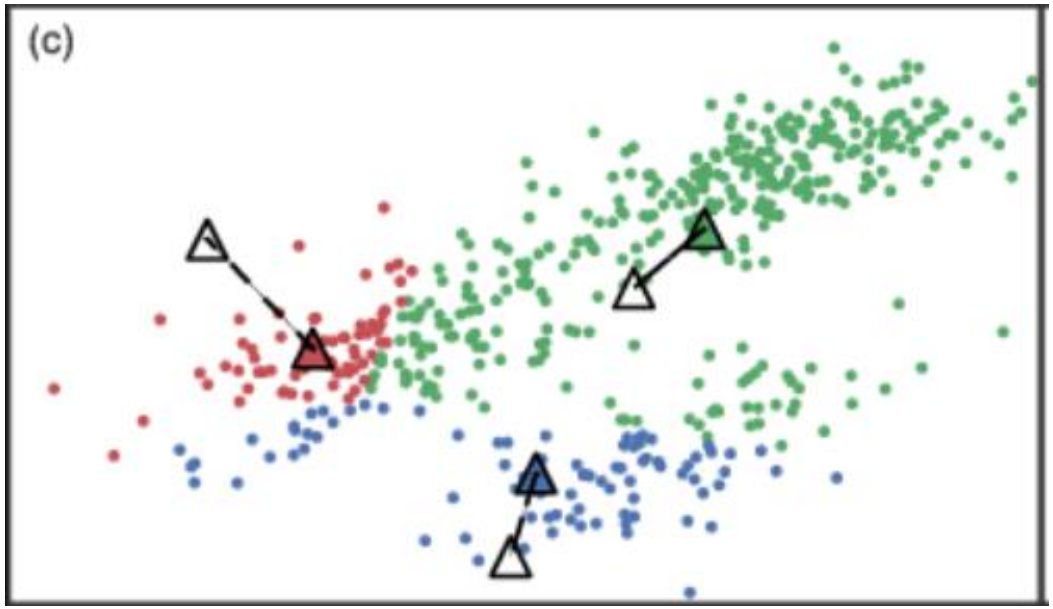
임의로 3개의 centroid 를 정한다

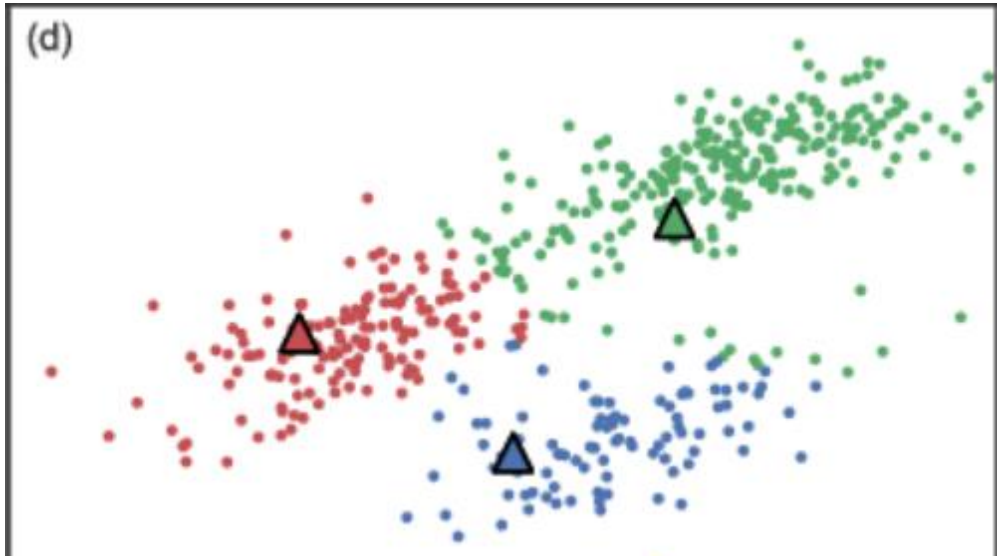
모든 point (p) 에 대해서 3개의 centroid 까지의 거리를 계산하여 가장 가까운 centroid의 color를 p의 color를 결정



Step-by-Step K-means Example [2/3]

같은 color의 point들에서 x좌표의 평균과 y좌표의 평균을 계산하여 새로운 centroid 를 정한다

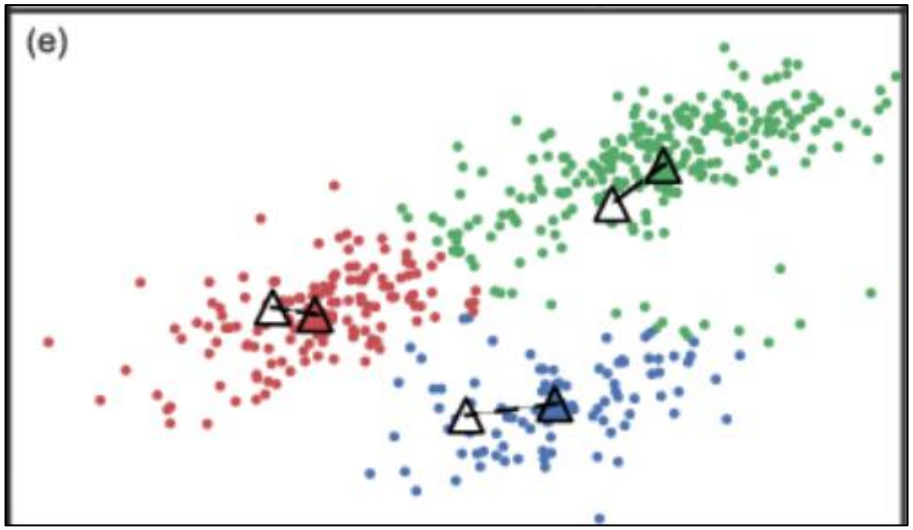


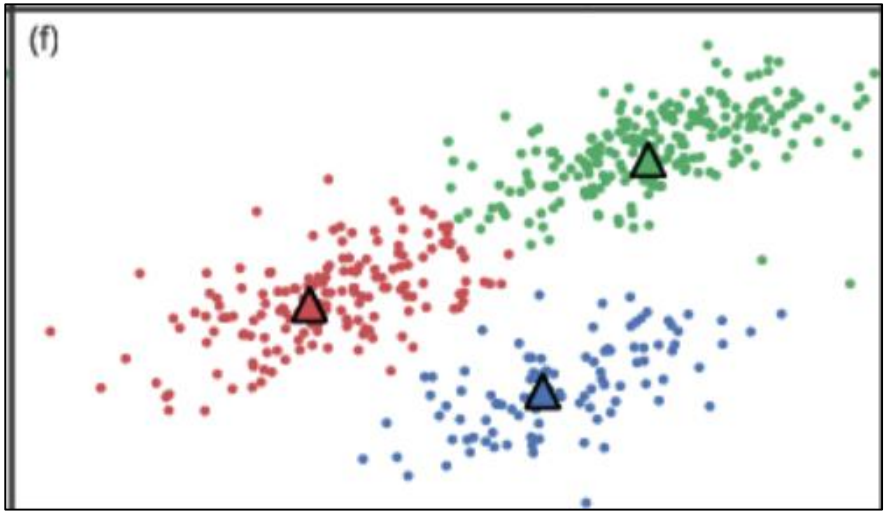


모든 point (p) 에 대해서 3개의 centroid 까지의 거리를 계산하여 가장 가까운 centroid의 color를 p의 color를 결정

Step-by-Step K-means Example [3/3]

같은 color의 point들에서 x좌표의 평균과 y좌표의 평균을 계산하여 새로운 centroid 를 정한다

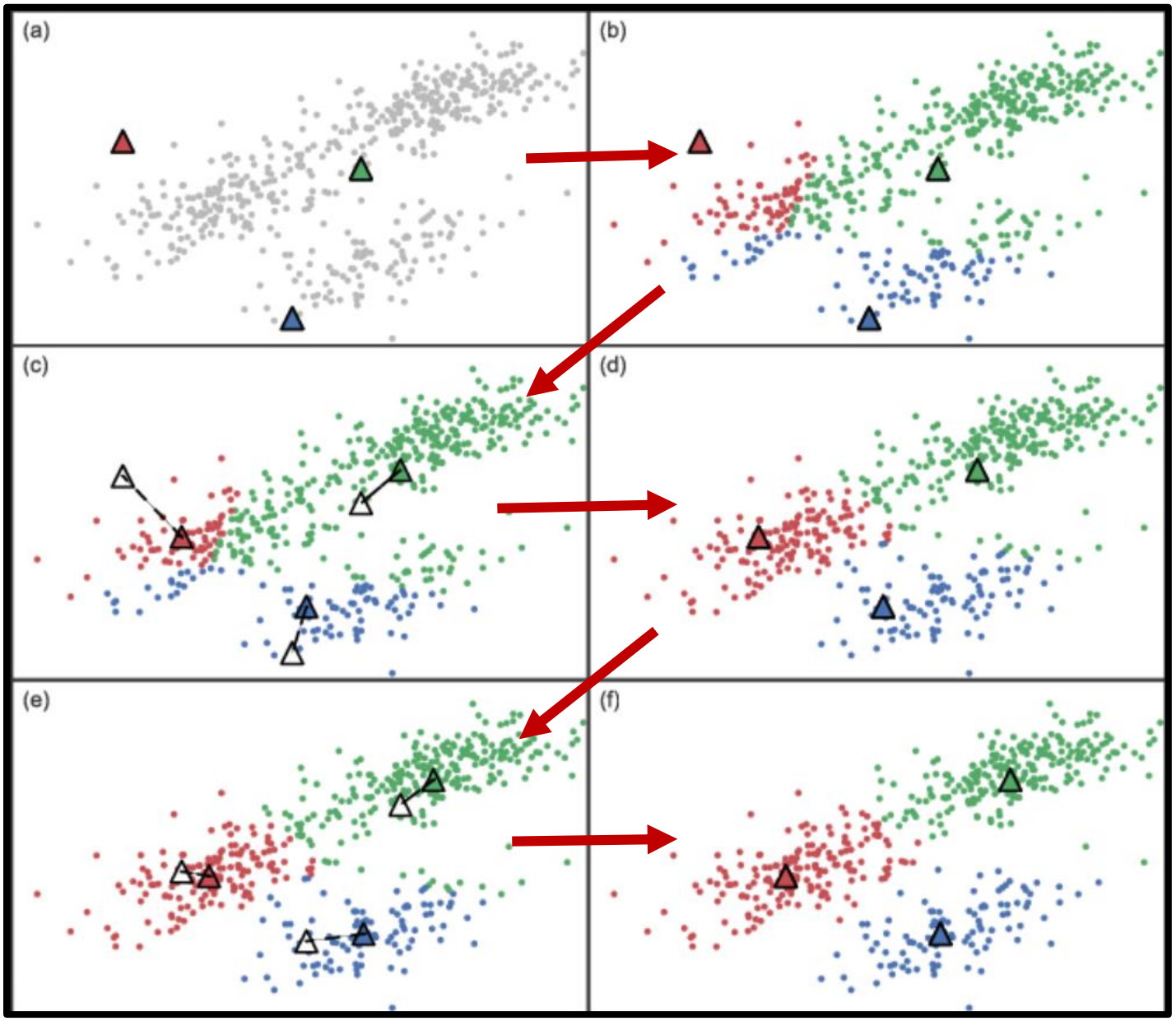




모든 point (p) 에 대해서 3개의 centroid 까지의 거리를 계산하여 가장 가까운 centroid의 color를 p의 color를 결정

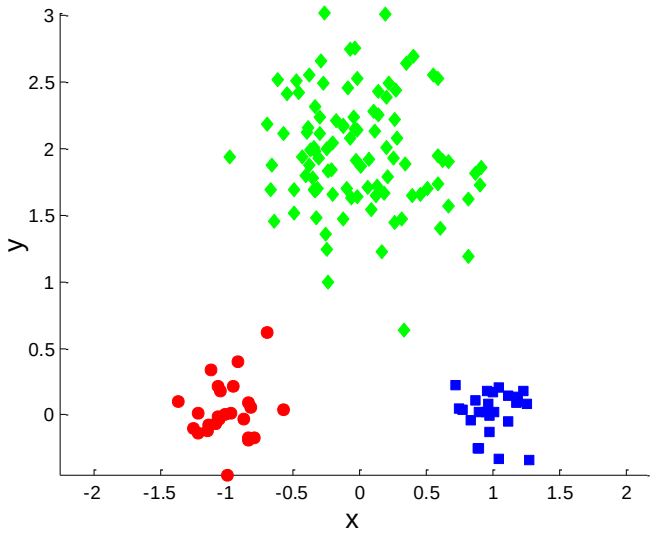
Centriod들의 위치에 변화가 없을때까지 계속 반복!

Full Sequence of K-means Clustering



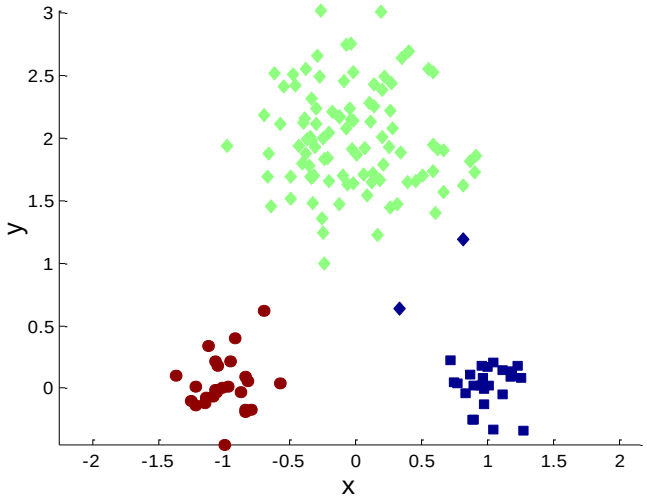
2 different K-means Clustering according to Initial Centroids

Original Points

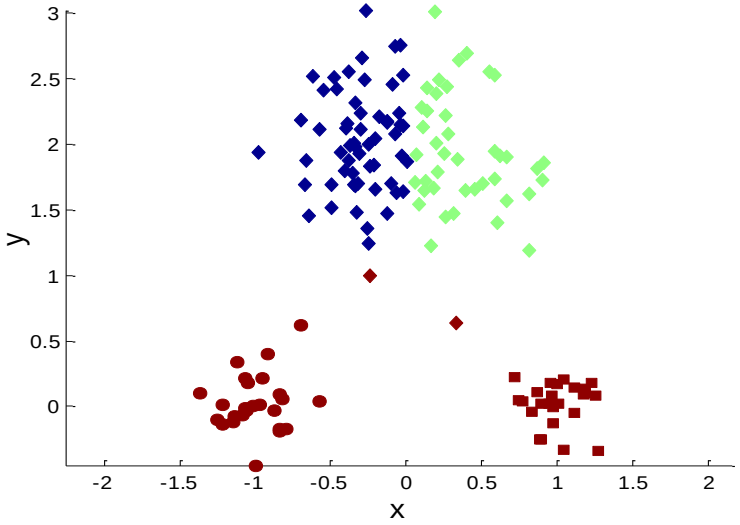


X	Y	Color
1	2	Green
-0.5	2.1	Green
0.5	2.2	Green
…	…	
-1	0	Red
-1.2	0.7	Red
1	0	Blue

No Use



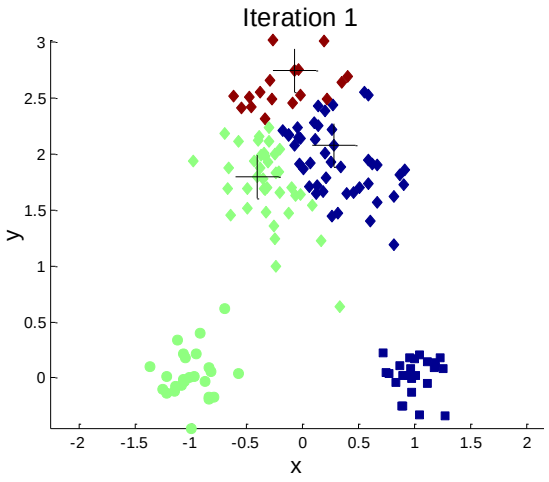
Optimal Clustering

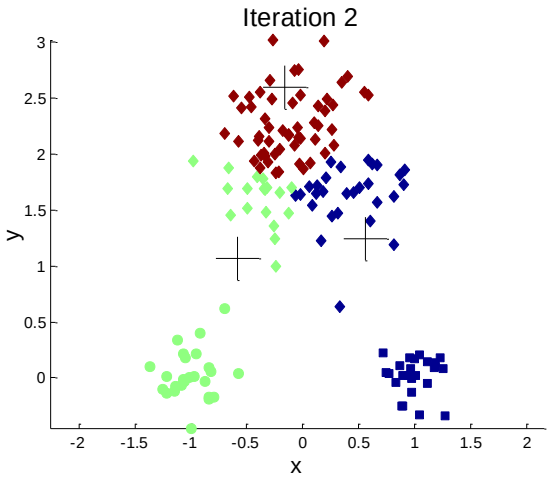


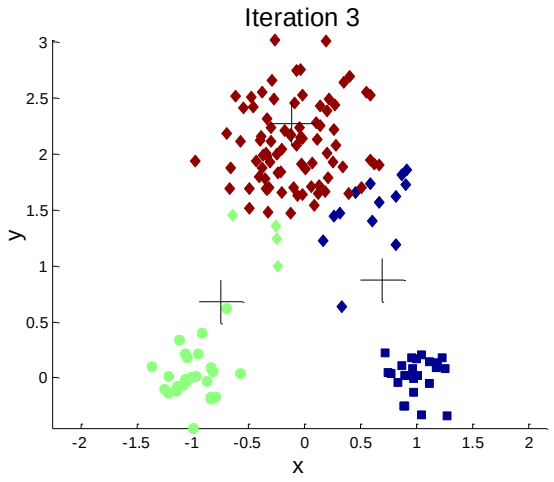
Sub-optimal Clustering

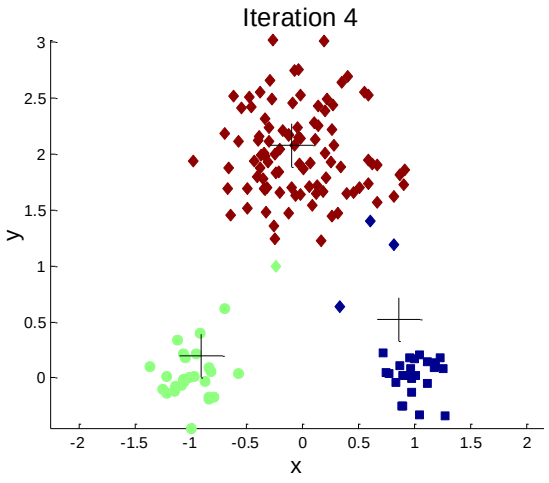
Importance of Choosing Initial Centroids: Good Case

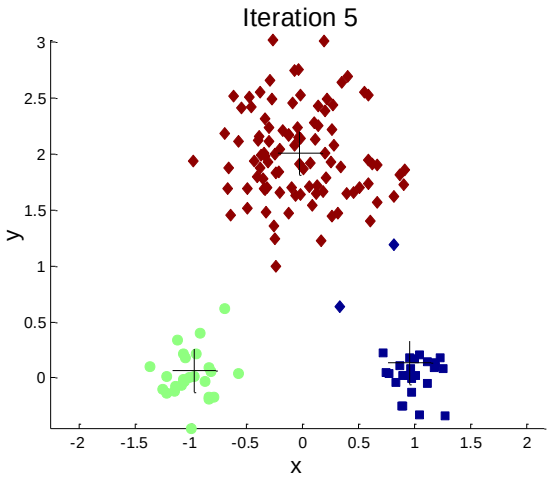
Cluster 중심점 3개을 random하게 찍고, 모든 point들에 대해서 3개의 cluster 중심점과의 거리계산을 하고, 거리가 가장 짧은 cluster 중심점의 member로 labeling하고, 각 cluster별로 center point를 다시 설정하여 Cluster 중심점을 update하고, cluster들이 변화가 없을때까지 repeatedly 작업

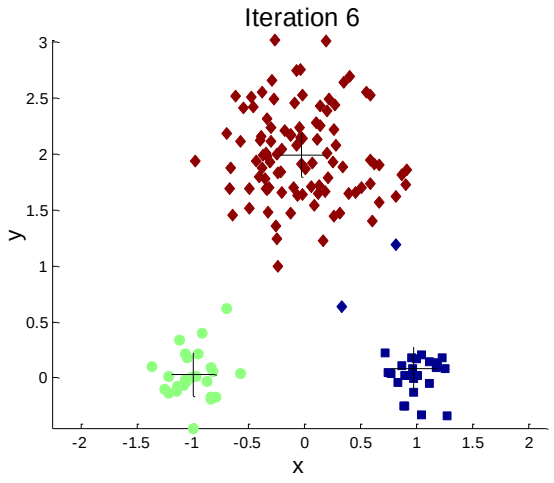












Kmeans Class (at cluster submodule)

- sklearn.cluster.KMeans (***kwargs*)
  - **n_clusters (8)**: number of clusters
  - **max_iter (300)**: maximum number of iteration for a single run
  - **n_init (10)**: number of run time with different centroid
  - **Init (k-means++)**: method for centroid initialization.
    {k-means++, random}
  - **tol**: tolerance for optimization. Determine whether training step converges
  - **random_state**: seed for initial random numbers for coefficients.
  - **n_jobs**: number of jobs for computation. (1: default, -1: use all CPUs).
  - **copy_X**: if True, original data is not modified

11 Parameters & 5 attributes

K-Means Clustering Example: Data Preprocessing [1/3]

- Import external modules & load **iris** dataset
- Create subset from iris data with 2nd, 3rd columns

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

from sklearn.cluster import KMeans
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data[:, 2:4], iris.target

X.shape, X
```

```
((150, 2), array([[ 1.4,  0.2],
                  [ 1.4,  0.2],
                  [ 1.3,  0.2],
                  [ 1.5,  0.2],
                  [ 1.4,  0.2],  ...)
```

X — Numpy 2D Array y — Numpy 1D Array

sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
6.3	3.3	6	2.5	2
5.8	2.7	5.1	1.9	2
7.1	3	5.9	2.1	2

150개

Original iris dataset

No use!

Clustering iteration은 converge할때까지 계속!

K-Means Clustering Example: Learning [2/3]

Numpy 2D Array

petal length	petal width
1.4	0.2
1.4	0.2
1.3	0.2
4.7	1.4
4.5	1.5
4.9	1.5
6	2.5
5.1	1.9
5.9	2.1

- Learning k-means clustering model with dataset **X**

```
In [7]: est = KMeans(n_clusters=3)
        est.fit(X)

        est
```

```
Out[7]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
            n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
            random_state=None, tol=0.0001, verbose=0)
```

- Result of the clustering as list

학습의 결과물

```
In [8]: labels = est.labels_

        labels
```

```
Out[8]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 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, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
               2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

150개의 0, 1, 2 값

K-Means Clustering Example: Plot The Result [3/3]

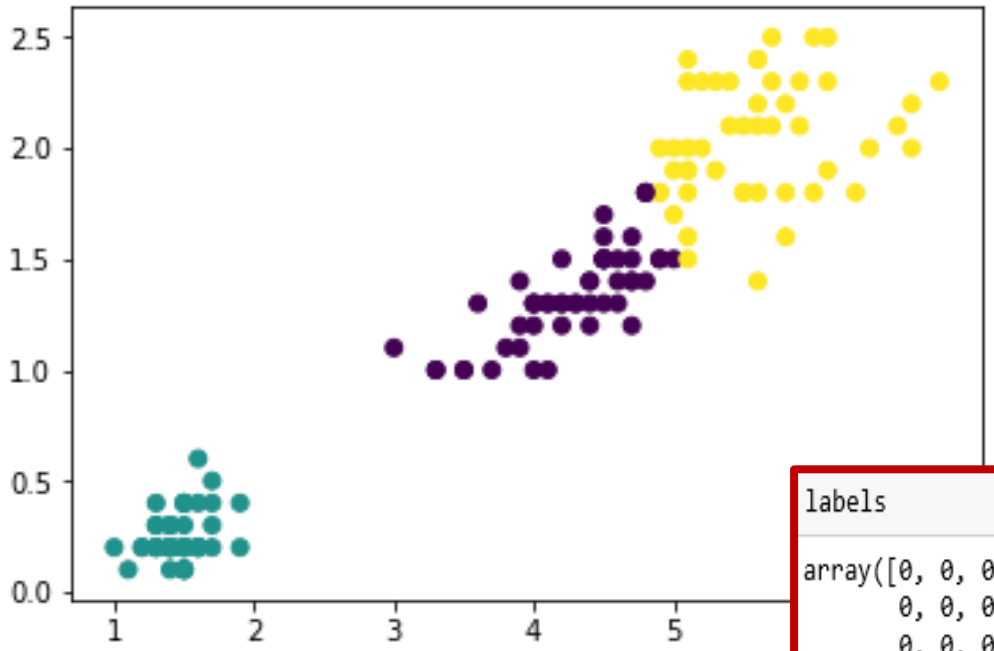
- Plot the result of clustering as scatter plot
  - Each data point is colored by the cluster number in the '**labels**'

color

Default colormap 이용

```
: plt.scatter(X[:, 0], X[:, 1], c=labels)
: <matplotlib.collections.PathCollection at 0x1ebd50efcf8>
```

Label 0 ➔ Violet
Label 1 ➔ Green
Label 2 ➔ Yellow



X

petal length	petal width
1.4	0.2
1.4	0.2
1.3	0.2
4.7	1.4
4.5	1.5
4.9	1.5
6	2.5
5.1	1.9
5.9	2.1

150개

labels

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

Clustering에서는 Test Data 개념이 없음!
Clustering 자체로 목적완성!

All-in-One Code (K-means Clustering)

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

from sklearn.cluster import KMeans
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data[:, 2:4], iris.target
```

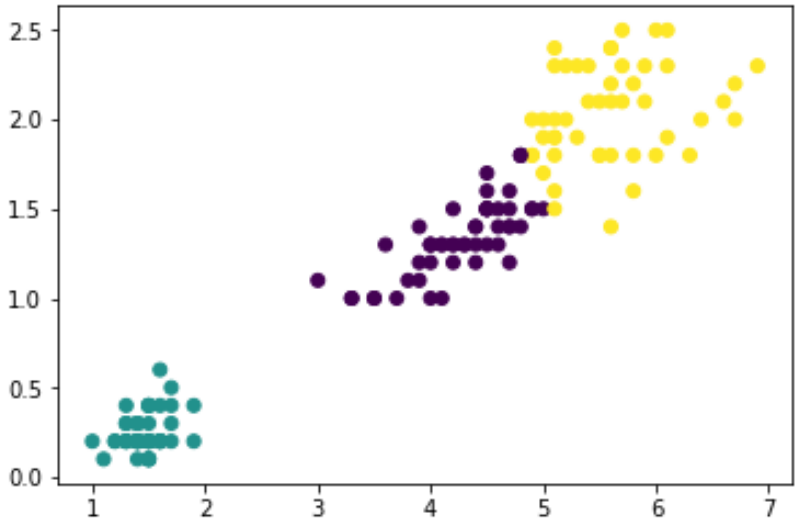
```
est = KMeans(n_clusters=3)
est.fit(X)
labels = est.labels_
```

```
plt.scatter(X[:, 0], X[:, 1], c=labels)
```

```
plt.scatter(X[:,0], X[:,1], c=y)
```

X Numpy 2D Array

petal length	petal width
1.4	0.2
1.4	0.2
1.3	0.2
4.7	1.4
4.5	1.5
4.9	1.5
6	2.5
5.1	1.9
5.9	2.1



K-Means Result vs. Real Data Plotting

```
iris = datasets.load_iris()
X, y = iris.data[:, 2:4], iris.target

est = KMeans(n_clusters=3)
est.fit(X)
labels = est.labels_

plt.scatter(X[:, 0], X[:, 1], c=labels)
```

VS

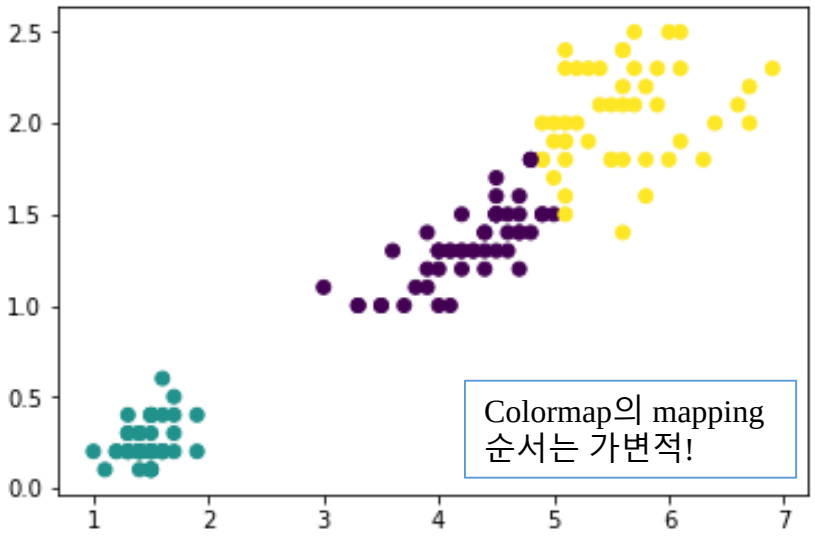
```
iris = datasets.load_iris()
X, y = iris.data[:, 2:4], iris.target
plt.scatter(X[:,0], X[:,1], c=y)
```

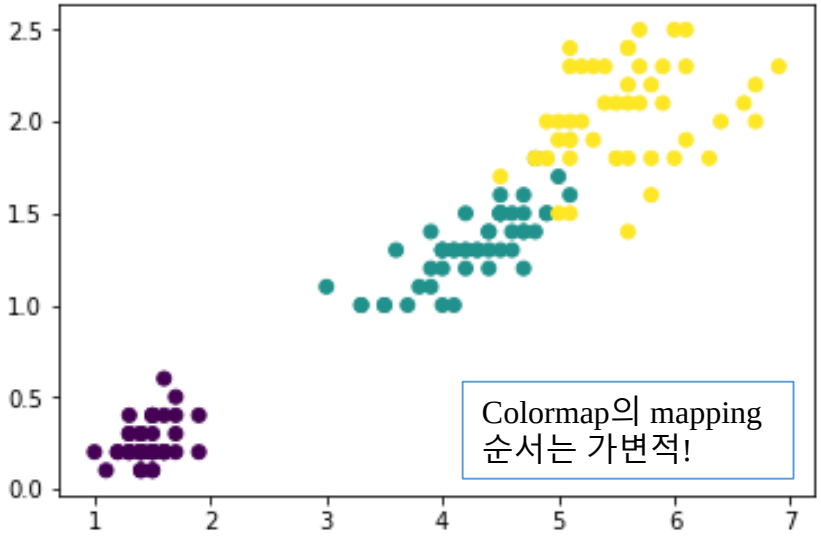
X: Numpy 2D Array, y: Numpy 1D Array

sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3	1.4	0.2	0
4.7	3.2	1.3	0.2	0
7	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
6.3	3.3	6	2.5	2
5.8	2.7	5.1	1.9	2
7.1	3	5.9	2.1	2

150개

Original iris dataset





matplotlib.pyplot.scatter ¶

참고자료

matplotlib.pyplot.scatter(*x*, *y*, *s=None*, *c=None*, *marker=None*, *cmap=None*, *norm=None*, *vmin=None*, *vmax=None*, *alpha=None*, *linewidths=None*, *verts=None*, *edgecolors=None*, *hold=None*, *data=None*, ***kwargs*)

A scatter plot of *y* vs *x* with varying marker size and/or color.

Parameters:

x, y : array_like, shape (n,)

The data positions.

s : scalar or array_like, shape (n,), optional

The marker size in points**2. Default is `rcParams['lines.markersize'] ** 2`.

c : color, sequence, or sequence of color, optional, default: 'b'

The marker color. Possible values:

- A single color format string.
- A sequence of color specifications of length n.
- A sequence of n numbers to be mapped to colors using *cmap* and *norm*.
- A 2-D array in which the rows are RGB or RGBA.

Note that *c* should not be a single numeric RGB or RGBA sequence because that is indistinguishable from an array of values to be colormapped. If you want to specify the same RGB or RGBA value for all points, use a 2-D array with a single row.

marker : `MarkerStyle`, optional, default: 'o'

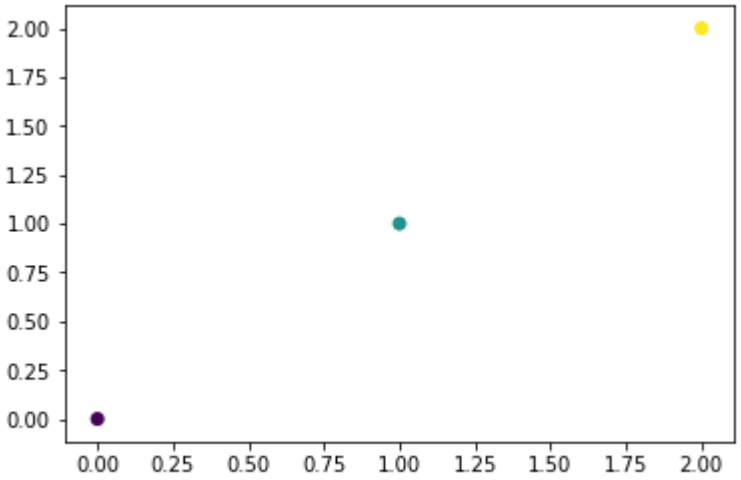
The marker style. *marker* can be either an instance of the class or the text shorthand for a particular marker. See `markers` for more information marker styles.

cmap : `Colormap`, optional, default: None

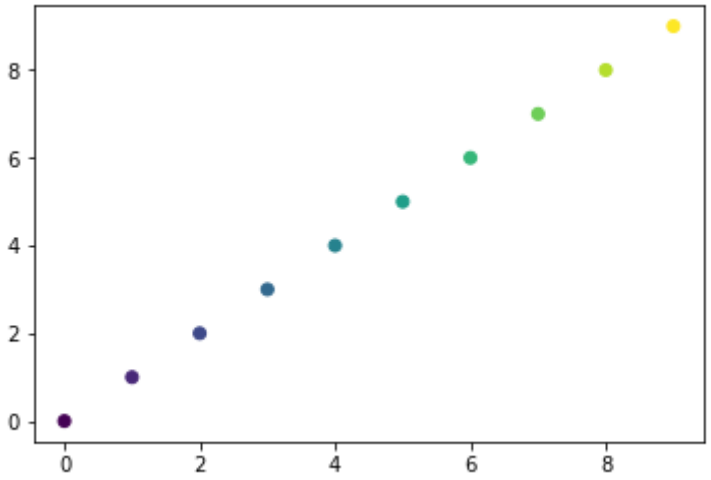
A `Colormap` instance or registered colormap name. *cmap* is only used if *c* is an array of floats. If `None`, defaults to rc `image.cmap`.

Default Color Map in plt.scatter( )

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(3)
y = np.arange(3)
t = np.arange(3)

plt.scatter(x, y, c=t)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(10)
y = np.arange(10)
t = np.arange(10)

plt.scatter(x, y, c=t)
plt.show()
```

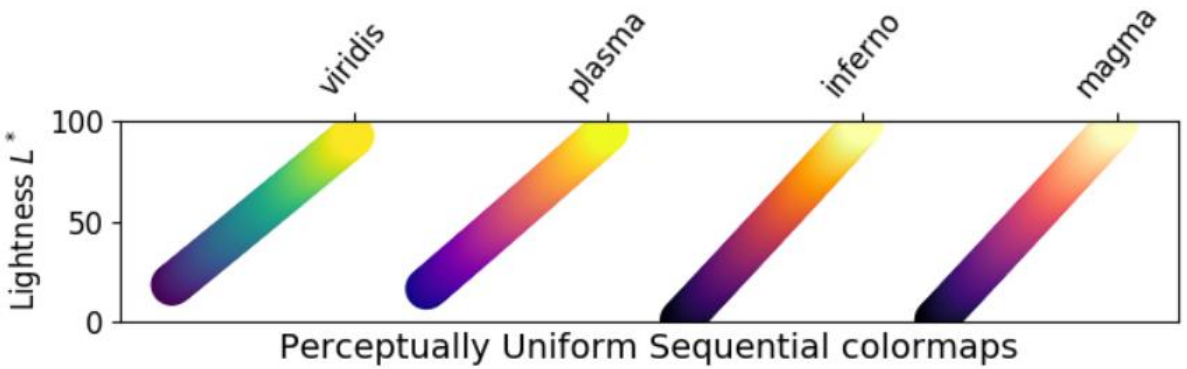


Default Color Map

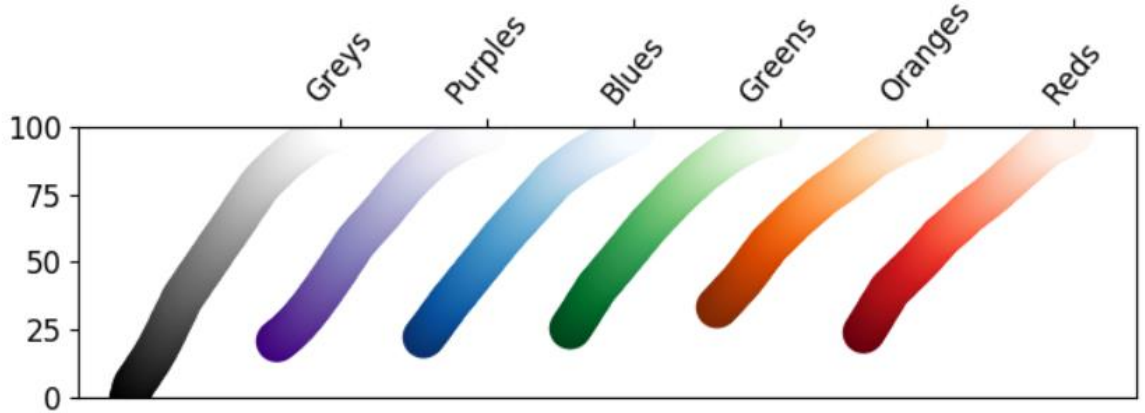
t = 0 ➔ Violet
t = 1 ➔ Green
t = 2 ➔ Yellow

Matplotlib: image.cmap

(Source code)



(png, pdf)



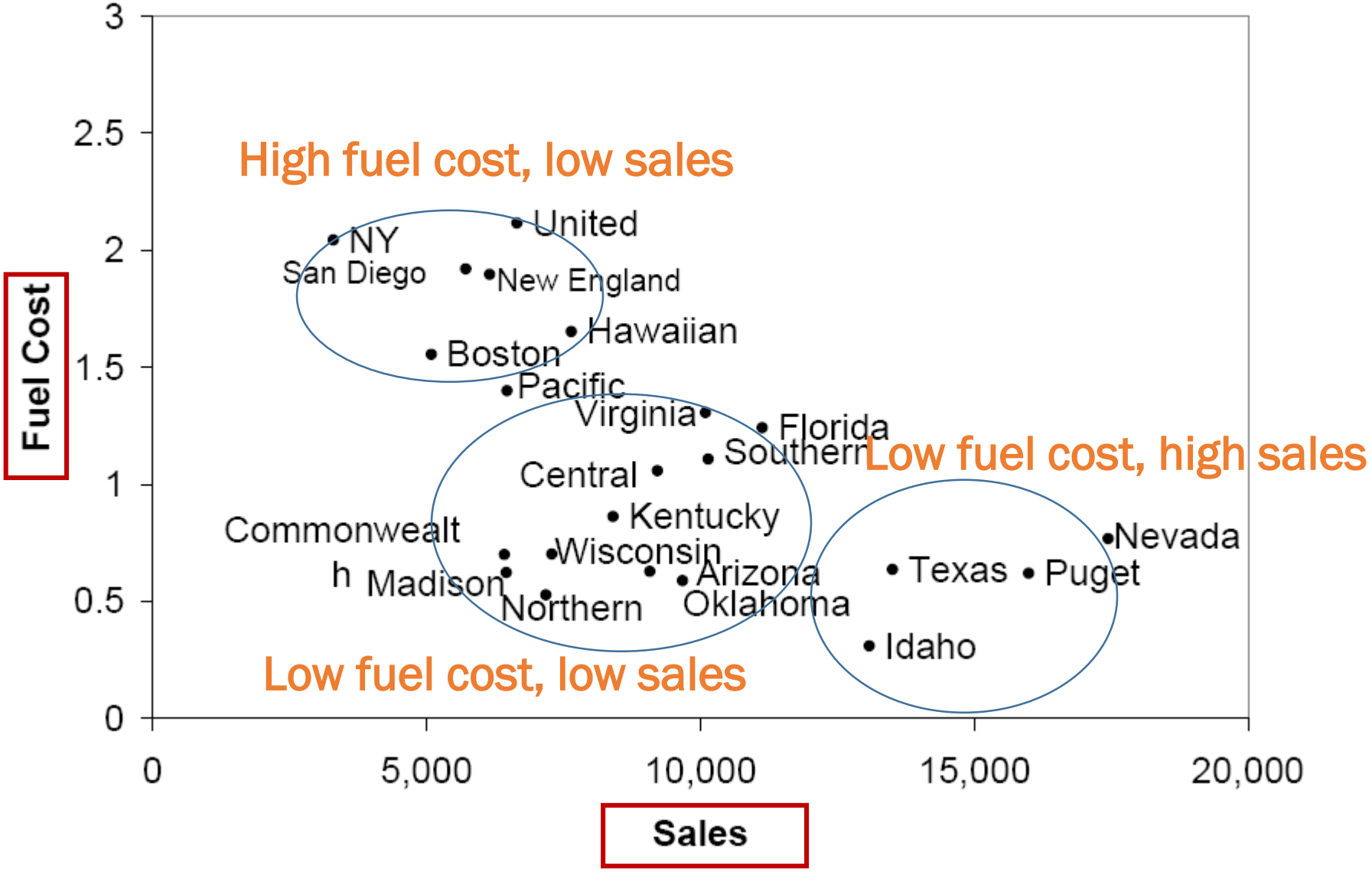
Applications of K-Means [1/3]

Data of Fuel Efficiency of Energy Companies

Company	Fixed_charge	RoR	Cost	Load	Δ Demand	Sales	Nuclear	Fuel_Cost
Arizona	1.06	9.2	151	54.4	1.6	9077	0	0.628
Boston	0.89	10.3	202	57.9	2.2	5088	25.3	1.555
Central	1.43	15.4	113	53	3.4	9212	0	1.058
Commonwealth	1.02	11.2	168	56	0.3	6423	34.3	0.7
Con Ed NY	1.49	8.8	192	51.2	1	3300	15.6	2.044
Florida	1.32	13.5	111	60	-2.2	11127	22.5	1.241
Hawaiian	1.22	12.2	175	67.6	2.2	7642	0	1.652
Idaho	1.1	9.2	245	57	3.3	13082	0	0.309
Kentucky	1.34	13	168	60.4	7.2	8406	0	0.862
Madison	1.12	12.4	197	53	2.7	6455	39.2	0.623
Nevada	0.75	7.5	173	51.5	6.5	17441	0	0.768
New England	1.13	10.9	178	62	3.7	6154	0	1.897
Northern	1.15	12.7	199	53.7	6.4	7179	50.2	0.527
Oklahoma	1.09	12	96	49.8	1.4	9673	0	0.588
Pacific	0.96	7.6	164	62.2	-0.1	6468	0.9	1.4
Puget	1.16	9.9	252	56	9.2	15991	0	0.62
San Diego	0.76	6.4	136	61.9	9	5714	8.3	1.92
Southern	1.05	12.6	150	56.7	2.7	10140	0	1.108
Texas	1.16	11.7	104	54	-2.1	13507	0	0.636
Wisconsin	1.2	11.8	148	59.9	3.5	7287	41.1	0.702
United	1.04	8.6	204	61	3.5	6650	0	2.116
Virginia	1.07	9.3	174	54.3	5.9	10093	26.6	1.306

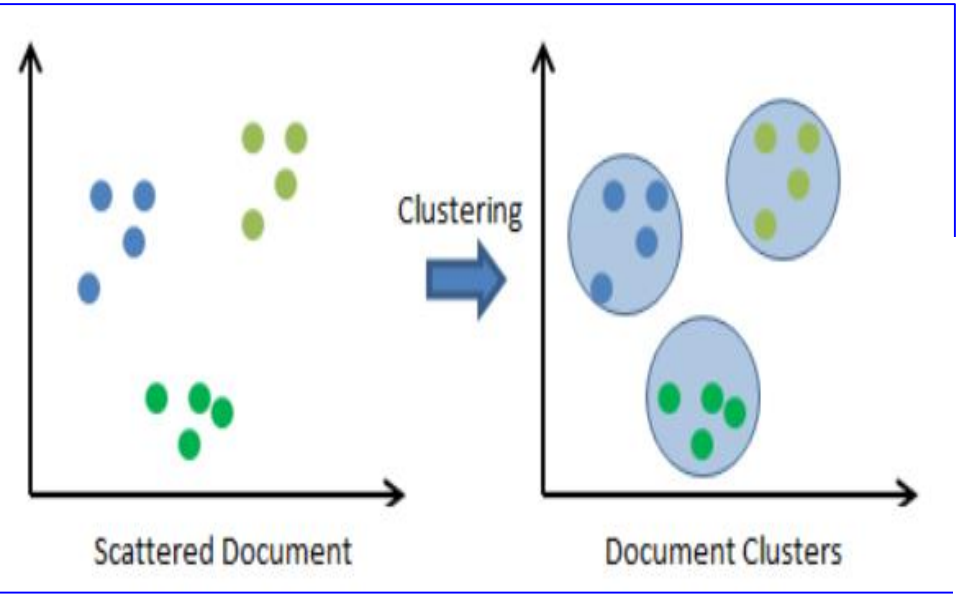
Applications of K-Means [2/3]

Sales & Fuel Cost: 3 rough clusters can be seen

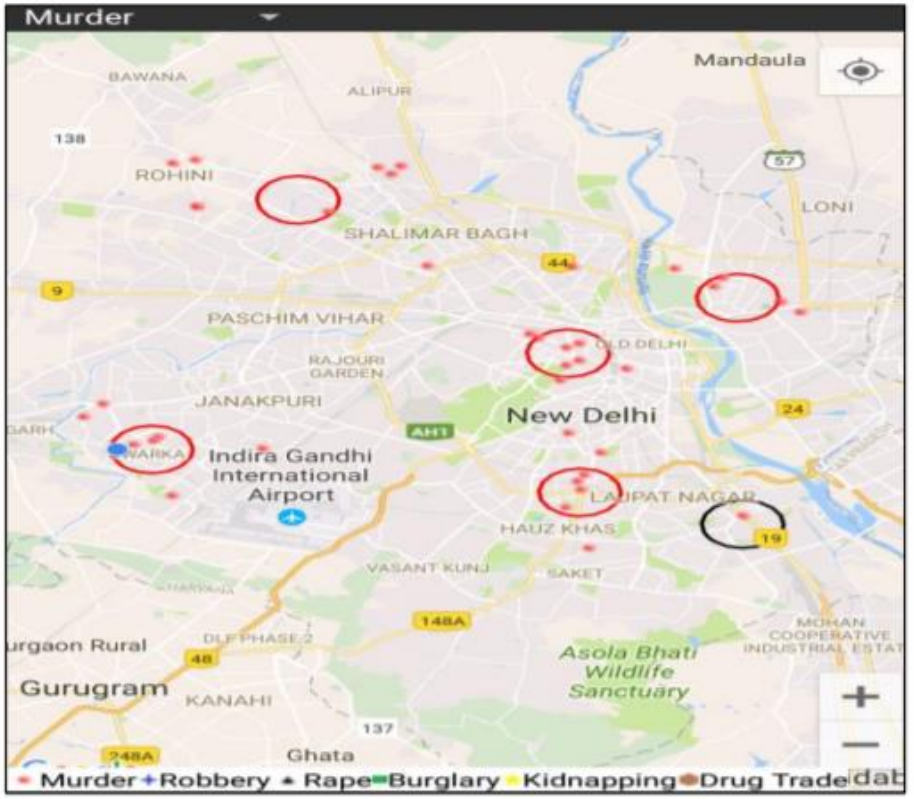


Applications of K-Means [3/3]

Document Clustering



Identifying Crime Localities



Sk-Learn: Table of Contents

- What is SK Learn?
- Linear Regression Classifier
- K-Nearest Neighbor (KNN) Classifier
- Decision Tree Classifier
- Neural Network Classifier
- K-Means Clustering
- SK-Learn Submodules and Their Functions

Scikit-learn Submodules [1/2]

`sklearn.base` : Base classes and utility functions
- Base classes
- Functions

`sklearn.calibration` : Probability Calibration

`sklearn.cluster` : Clustering
- Classes
- Functions

`sklearn.cluster.bicluster` : Biclustering
- Classes

`sklearn.covariance` : Covariance Estimators

`sklearn.cross_decomposition` : Cross decomposition

`sklearn.datasets` : Datasets
- Loaders
- Samples generator

`sklearn.decomposition` : Matrix Decomposition

`sklearn.discriminant_analysis` : Discriminant Analysis

`sklearn.dummy` : Dummy estimators

`sklearn.ensemble` : Ensemble Methods
- partial dependence

`sklearn.exceptions` : Exceptions and warnings

`sklearn.feature_extraction` : Feature Extraction
- From images
- From text

`sklearn.feature_selection` : Feature Selection

`sklearn.gaussian_process` : Gaussian Processes

`sklearn.isotonic` : Isotonic regression

`sklearn.kernel_approximation` Kernel Approximation

`sklearn.kernel_ridge` Kernel Ridge Regression

`sklearn.linear_model` : Generalized Linear Models

`sklearn.manifold` : Manifold Learning

`sklearn.metrics` : Metrics
- Model Selection Interface
- Classification metrics
- Regression metrics
- Multilabel ranking metrics
- Clustering metrics
- Biclustering metrics
- Pairwise metrics

`sklearn.mixture` : Gaussian Mixture Models

Scikit-learn Submodules [2/2]

`sklearn.model_selection` : Model Selection

- Splitter Classes
- Splitter Functions
- Hyper-parameter optimizers
- Model validation

`sklearn.multiclass` : Multiclass and multilabel classification

- Multiclass and multilabel classification strategies

`sklearn.multioutput` : Multioutput regression and classification

`sklearn.naive_bayes` : Naive Bayes

`sklearn.neighbors` : Nearest Neighbors

`sklearn.neural_network` : Neural network models

`sklearn.pipeline` : Pipeline

`sklearn.preprocessing` : Preprocessing and Normalization

`sklearn.random_projection` : Random projection

`sklearn.semi_supervised` Semi-Supervised Learning

`sklearn.svm` : Support Vector Machines

- Estimators
- Low-level methods

`sklearn.tree` : Decision Trees

`sklearn.utils` : Utilities

Scikit-learn: Datasets Submodule

sklearn.datasets : Datasets

- datasets.clear_data_home ([data_home])
- datasets.dump_svmlight_file (X, y, f[, ...])
- datasets.fetch_20newsgroups ([data_home, ...])
- datasets.fetch_20newsgroups_vectorized ([...])
- datasets.fetch_california_housing ([...])
- datasets.fetch_covtype ([data_home, ...])
- datasets.fetch_kddcup99 ([subset, data_home, ...])
- datasets.fetch_lfw_pairs ([subset, ...])
- datasets.fetch_lfw_people ([data_home, ...])
- datasets.fetch_mldata (dataname[, ...])
- datasets.fetch_olivetti_faces ([data_home, ...])
- datasets.fetch_rcv1 ([data_home, subset, ...])
- datasets.fetch_species_distributions ([...])
- datasets.get_data_home ([data_home])
- datasets.load_boston ([return_X_y])
- datasets.load_breast_cancer ([return_X_y])
- datasets.load_diabetes ([return_X_y])
- datasets.load_digits ([n_class, return_X_y])
- datasets.load_files (container_path[, ...])
- datasets.load_iris ([return_X_y])
- datasets.load_linnerud ([return_X_y])
- datasets.load_mlcomp (*args, **kwargs)
- datasets.load_sample_image (image_name)
- datasets.load_sample_images ()
- datasets.load_svmlight_file (f [, n_features, ...])
- datasets.load_svmlight_files (files[, ...])
- datasets.load_wine ([return_X_y])
- datasets.mldata_filename (dataname)

Scikit-learn: Clustering Submodule

sklearn.cluster: Clustering

The `sklearn.cluster` module gathers popular unsupervised clustering algorithms.

User guide: See the Clustering section for further details.

Classes

`cluster.AffinityPropagation` ([damping, ...])	Perform Affinity Propagation Clustering of data.
`cluster.AgglomerativeClustering` ([...])	Agglomerative Clustering
`cluster.Birch` ([threshold, branching_factor, ...])	Implements the Birch clustering algorithm.
`cluster.DBSCAN` ([eps, min_samples, metric, ...])	Perform DBSCAN clustering from vector array or distance matrix.
`cluster.FeatureAgglomeration` ([n_clusters, ...])	Agglomerate features.
`cluster.KMeans` ([n_clusters, init, n_init, ...])	K-Means clustering
`cluster.MiniBatchKMeans` ([n_clusters, init, ...])	Mini-Batch K-Means clustering
`cluster.MeanShift` ([bandwidth, seeds, ...])	Mean shift clustering using a flat kernel.
`cluster.SpectralClustering` ([n_clusters, ...])	Apply clustering to a projection to the normalized laplacian.

Scikit-learn: Generalized Linear Model Submodule [1/2]

`sklearn.linear_model` : Generalized Linear Models

`linear_model.ARDRegression` ([n_iter, tol, ...])	Bayesian ARD regression.
`linear_model.BayesianRidge` ([n_iter, tol, ...])	Bayesian ridge regression
`linear_model.ElasticNet` ([alpha, l1_ratio, ...])	Linear regression with combined L1 and L2 priors as regularizer.
`linear_model.ElasticNetCV` ([l1_ratio, eps, ...])	Elastic Net model with iterative fitting along a regularization path
`linear_model.HuberRegressor` ([epsilon, ...])	Linear regression model that is robust to outliers.
`linear_model.Lars` ([fit_intercept, verbose, ...])	Least Angle Regression model a.k.a.
`linear_model.LarsCV` ([fit_intercept, ...])	Cross-validated Least Angle Regression model
`linear_model.Lasso` ([alpha, fit_intercept, ...])	Linear Model trained with L1 prior as regularizer (aka the Lasso)
`linear_model.LassoCV` ([eps, n_alphas, ...])	Lasso linear model with iterative fitting along a regularization path
`linear_model.LassoLars` ([alpha, ...])	Lasso model fit with Least Angle Regression a.k.a.
`linear_model.LassoLarsCV` ([fit_intercept, ...])	Cross-validated Lasso, using the LARS algorithm
`linear_model.LassoLarsIC` ([criterion, ...])	Lasso model fit with Lars using BIC or AIC for model selection
`linear_model.LinearRegression` ([...])	Ordinary least squares Linear Regression.
`linear_model.LogisticRegression` ([penalty, ...])	Logistic Regression (aka logit, MaxEnt) classifier.
`linear_model.LogisticRegressionCV` ([Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.

Scikit-learn: Generalized Linear Model Submodule [2/2]

sklearn.linear_model : Generalized Linear Models

linear_model.MultiTaskLasso ([alpha, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer
linear_model.MultiTaskElasticNet ([alpha, ...])	Multi-task ElasticNet model trained with L1/L2 mixed-norm as regularizer
linear_model.MultiTaskLassoCV ([eps, ...])	Multi-task L1/L2 Lasso with built-in cross-validation.
linear_model.MultiTaskElasticNetCV ([...])	Multi-task L1/L2 ElasticNet with built-in cross-validation.
linear_model.OrthogonalMatchingPursuit ([...])	Orthogonal Matching Pursuit model (OMP)
linear_model.OrthogonalMatchingPursuitCV ([...])	Cross-validated Orthogonal Matching Pursuit model (OMP)
linear_model.PassiveAggressiveClassifier ([...])	Passive Aggressive Classifier
linear_model.PassiveAggressiveRegressor ([C, ...])	Passive Aggressive Regressor
linear_model.Perceptron ([penalty, alpha, ...])	Read more in the User Guide.
linear_model.RANSACRegressor ([...])	RANSAC (RANdom SAmple Consensus) algorithm.
linear_model.Ridge ([alpha, fit_intercept, ...])	Linear least squares with l2 regularization.
linear_model.RidgeClassifier ([alpha, ...])	Classifier using Ridge regression.
linear_model.RidgeClassifierCV ([alphas, ...])	Ridge classifier with built-in cross-validation.
linear_model.RidgeCV ([alphas, ...])	Ridge regression with built-in cross-validation.
linear_model.SGDClassifier ([loss, penalty, ...])	Linear classifiers (SVM, logistic regression, a.o.) with SGD training

Scikit-learn: Nearest Neighbors Submodule

sklearn.neighbors : Nearest Neighbors

neighbors.BallTree	BallTree for fast generalized N-point problems
neighbors.DistanceMetric	DistanceMetric class
neighbors.KDTree	KDTree for fast generalized N-point problems
neighbors.KernelDensity ([bandwidth, ...])	Kernel Density Estimation
neighbors.KNeighborsClassifier ([...])	Classifier implementing the k-nearest neighbors vote.
neighbors.KNeighborsRegressor ([n_neighbors, ...])	Regression based on k-nearest neighbors.
neighbors.LocalOutlierFactor ([n_neighbors, ...])	Unsupervised Outlier Detection using Local Outlier Factor (LOF)
neighbors.RadiusNeighborsClassifier ([...])	Classifier implementing a vote among neighbors within a given radius
neighbors.RadiusNeighborsRegressor ([radius, ...])	Regression based on neighbors within a fixed radius.
neighbors.NearestCentroid ([metric, ...])	Nearest centroid classifier.
neighbors.NearestNeighbors ([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
neighbors.kneighbors_graph (X, n_neighbors[, ...])	Computes the (weighted) graph of k-Neighbors for points in X
neighbors.radius_neighbors_graph (X, radius)	Computes the (weighted) graph of Neighbors for points in X

Scikit-learn: Neural Net Submodule

sklearn.neural_network : Neural network models

The `sklearn.neural_network` module includes models based on neural networks.

User guide: See the Neural network models (supervised) and Neural network models (unsupervised) sections for further details.

`neural_network.BernoulliRBM` ([n_components, ...])	Bernoulli Restricted Boltzmann Machine (RBM).
`neural_network.MLPClassifier` ([...])	Multi-layer Perceptron classifier.
`neural_network.MLPRegressor` ([...])	Multi-layer Perceptron regressor.

Scikit-learn: Decision Tree Submodule

sklearn.tree : Decision Trees

The `sklearn.tree` module includes decision tree-based models for classification and regression.

User guide: See the Decision Trees section for further details.

`tree.DecisionTreeClassifier` ([criterion, ...])	A decision tree classifier.
`tree.DecisionTreeRegressor` ([criterion, ...])	A decision tree regressor.
`tree.ExtraTreeClassifier` ([criterion, ...])	An extremely randomized tree classifier.
`tree.ExtraTreeRegressor` ([criterion, ...])	An extremely randomized tree regressor.
`tree.export_graphviz` (decision_tree[, ...])	Export a decision tree in DOT format.