



Samsung Innovation Campus

| Artificial Intelligence Course

Together for Tomorrow!
Enabling People

Education for Future Generations

Chapter 5.

Machine Learning 1

- Supervised Learning

Artificial Intelligence
Course

Chapter Description

◆ Chapter objectives

- ✓ Be able to introduce machine learning-based data analysis according to the business objective, strategy, and policy and manage the overall process.
- ✓ Be able to select and apply a machine learning algorithm that is the most suitable to the given problem and perform hyperparameter tuning.
- ✓ Be able to design, maintain, and optimize a machine learning workflow for AI modeling using structured and unstructured data.

◆ Chapter contents

- ✓ Unit 1. Machine Learning Based Data Analysis
- ✓ Unit 2. Application of Supervised Learning Model for Numerical Prediction
- ✓ Unit 3. Application of Supervised Learning Model for Classification
- ✓ Unit 4. Decision Tree
- ✓ Unit 5. Naïve Bayes Algorithm
- ✓ Unit 6. KNN Algorithm
- ✓ Unit 7. SVM Algorithm
- ✓ Unit 8. Ensemble Algorithm

Unit 1.

Machine Learning Based Data Analysis

- | 1.1. What is machine learning?
- | 1.2. Python scikit-learn library for machine learning
- | 1.3. Preparation and division of data set
- | 1.4. Data pre-processing for making a good training data set
- | 1.5. Practicing to find an optimal method to solve problems with scikit-learn

What is machine learning?

| What is machine learning?



- ▶ A statistical model that learns from **data**.
- ▶ A rather **simple** model can make **complex** predictions.

| Samuel's definition in the early phase of artificial intelligence

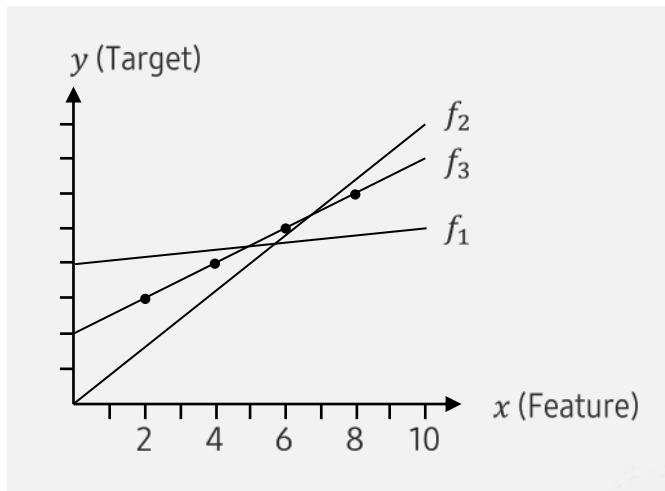
- ▶ "Programming Computers to learn from experience should eventually eliminate the need for much of this detailed programming effort." - Samuel, 1959

| Modern definition

- ▶ "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." – Mitchell, 1997 (p.2)
- ▶ "Programming computers to optimize a performance criterion using example data or past experience."
– Alpaydin, 2010
- ▶ "Computational methods using experience to improve performance or to make accurate predictions." – Mohri, 2012

Mathematical definition

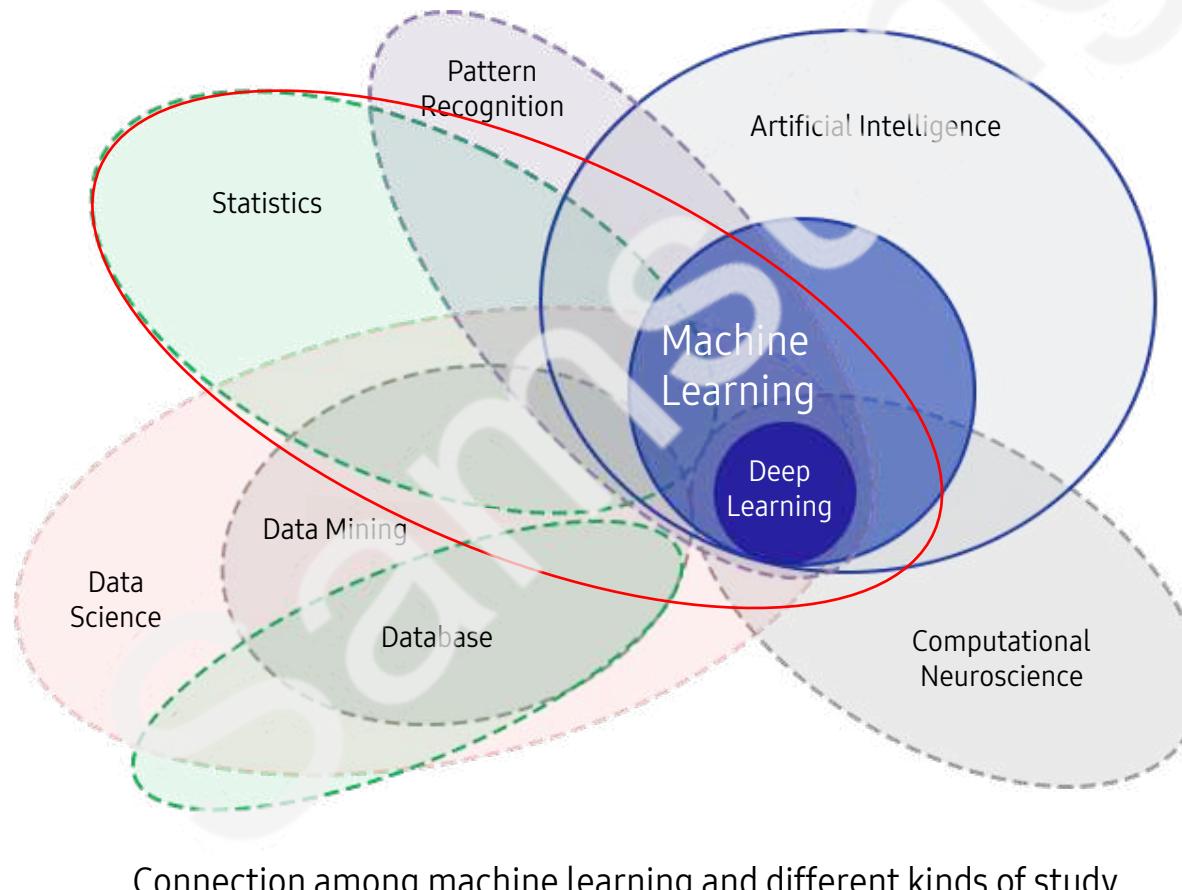
- Suppose that the x-axis is invested advertising expenses while the y-axis is sales.



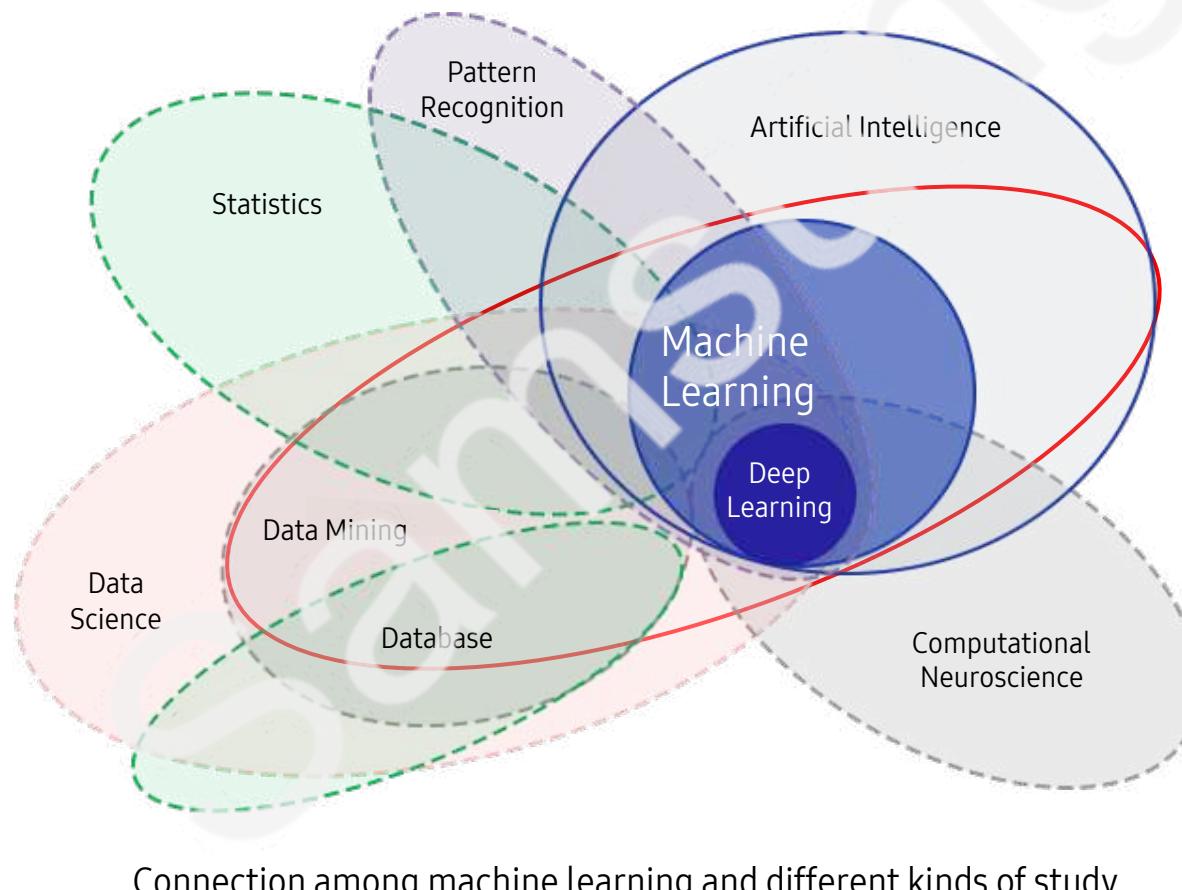
- Question about prediction – What are the sales when random advertising expenses are given?
- Linear regression
 - w and b as parameters
$$y = \underline{w}x + \underline{b}$$
- ‘w’ is commonly used as an abbreviation of ‘weigh.’

- Since the optimal value is unknown in the beginning, start with an arbitrary value and then reach the optimal value by gradually enhancing the performance.
 - From the graph, it starts from f_1 to continue as $f_1 \rightarrow f_2 \rightarrow f_3$.
 - The optimal value is f_3 where $w=0.5$ and $b=2.0$.

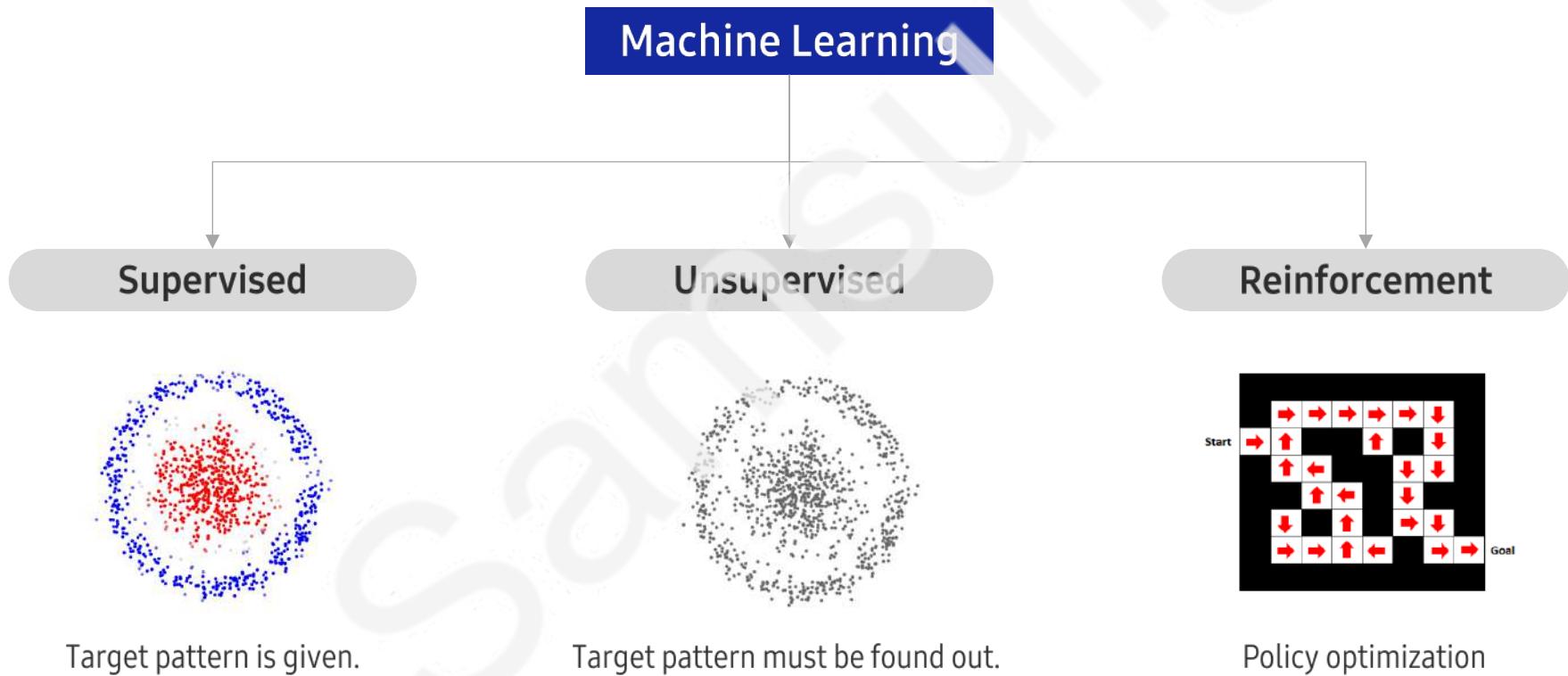
| Statistics and machine learning from a data analysis perspective



| Data mining and machine learning from a data analysis perspective



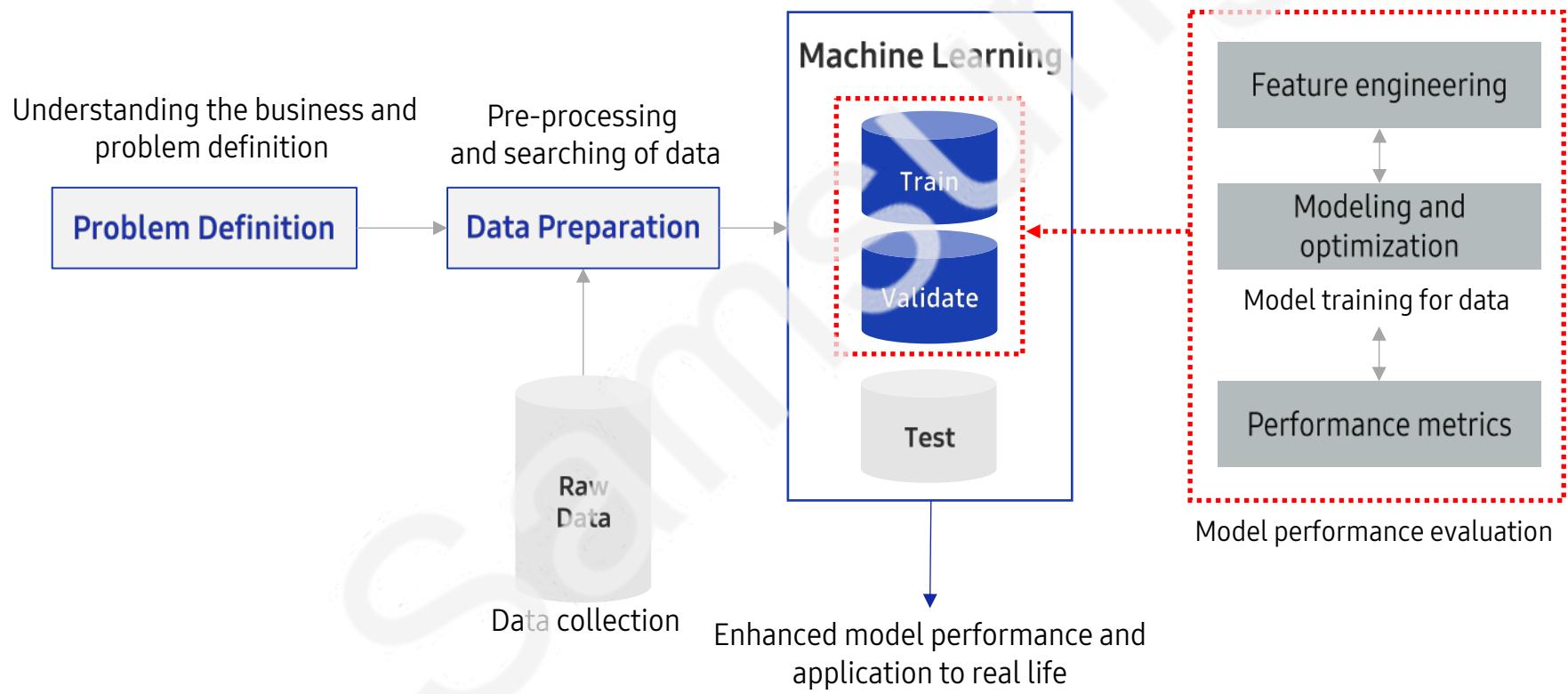
| Types of machine learning according to methods of supervision



1.1. What is machine learning?

UNIT 01

Machine learning workflow



| Machine learning types:

Type	Algorithm/Method
Unsupervised learning	Clustering
	MDS, t-SNE
	PCA, NMF
Supervised learning	Association analysis
	Linear regression
	Logistic regression
	Tree, Random Forest, Ada Boost, XGBoost
	Naïve Bayes
	KNN
	Support vector machine (SVM)
	Neural Network

Parameters vs. Hyperparameters

I Parameters

- ▶ Learned from data by training and not manually set by the practitioner.
- ▶ Contain the data pattern.

Ex Coefficients of linear regression

Ex Weights of neural network

I Hyperparameters

- ▶ **Can be set** manually by the practitioner.
- ▶ Can be tuned to optimize the machine learning performance.

Ex k in KNN algorithm

Ex Learning rate in neural network

Ex Maximum depth in Tree algorithm

Unit 1.

Machine Learning Based Data Analysis

- | 1.1. What is machine learning?
- | **1.2. Python scikit-learn library for machine learning**
- | 1.3. Preparation and division of data set
- | 1.4. Data pre-processing for making a good training data set
- | 1.5. Practicing to find an optimal method to solve problems with scikit-learn

Features of the scikit-learn library

| Features

- ▶ Integrated library interface by applying the façade design pattern
- ▶ Installed with various kinds of machine learning algorithms, model selection, and data pre-processing functions
- ▶ Simple and efficient tool to analyze predicted data
- ▶ Based on NumPy, SciPy and matplotlib
- ▶ Easily accessible and can be reused in many different situations
- ▶ Highly compatible with different libraries
- ▶ Does not support GPU
- ▶ Can be used as an open source and for commercial purposes

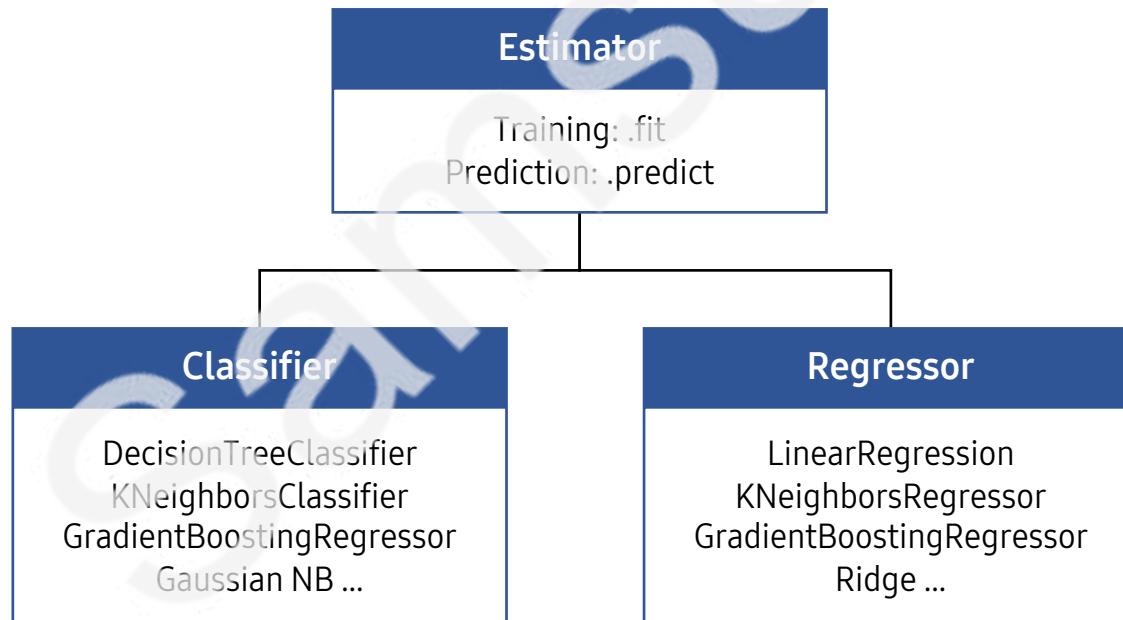
Mechanism of scikit-learn

- Scikit-learn is characterized by its intuitive and easy interface, complete with high-level API.



| Estimator, Classifier, Regressor

- ▶ Estimator refers to an object that can fit the model and deduce certain features of new data based on the training data.
- ▶ Classifier refers to a class that realizes a classifying algorithm, while regression refers to a class that realizes regressing algorithm.



Scikit-Learn Library

| About the Scikit-Learn library

- ▶ It is a representative Python machine learning library.
- ▶ To import a machine learning algorithm as a class:

```
from sklearn.<family> import <machine learning algorithm>
```

Ex from sklearn.linear_model import LinearRegression

- ▶ Hyperparameters are specified when the machine learning object is instantiated:

Ex myModel = KNeighborsClassifier(n_neighbors=10) # KNN with k = 10

| About the Scikit-Learn library

- ▶ To train a supervised learning model: `myModel.fit(X_train, Y_train)`
- ▶ To train a unsupervised learning model: `myModel.fit(X_train)`
- ▶ To predict using an already trained model: `myModel.predict(X_test)`
- ▶ To import a preprocessor as class: `from sklearn.preprocessing import <a preprocessor>`
- ▶ To split the dataset into a training set and a testing set:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=123)
```

- ▶ To calculate a performance metric (accuracy): `metrics.accuracy_score(Y_test, Y_pred)`
- ▶ To cross validate and do hyperparameter tuning at the same time:

```
myGridCV = GridSearchCV(estimator, parameter_grid, cv=k)
```

```
myGridCV.fit(X_train, Y_train)
```

| Practicing scikit-learn

- ▶ The sklearn.datasets module includes utilities to load datasets, including methods to load and fetch popular reference datasets. It also features some artificial data generators.

```
In [1]: from sklearn.datasets import load_breast_cancer
```

- ▶ Import data with the load_breast_cancer().

```
In [2]: type(load_breast_cancer())
```

```
Out[2]: sklearn.utils.Bunch
```

```
In [3]: data=load_breast_cancer()
```

- ▶ Container object exposing keys as attributes.

Bunch objects are sometimes used as an output for functions and methods.

They extend dictionaries by enabling values to be accessed by key, bunch["value_key"], or by an attribute, bunch.value_key.

Practicing scikit-learn

```
In [5]: pd.DataFrame(data['data'])
```

out[5]:

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0
...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6

569 rows × 30 columns



- This data becomes x (independent variable, data).

Practicing scikit-learn

```
In [6]: pd.DataFrame(data['target'])
```

```
Out[6]:
```

		0	
		0	564
1	0	565	0
2	0	566	0
3	0	567	0
4	0	568	1
...		569 rows × 1 columns	

Line 6

- This data becomes y (dependent variable, actual value).

| Practicing scikit-learn

```
In [1]: print(data.DESCR)
```

```
In [2]: help(train_test_split)
```

Line 1

- Provides details about the data.
- The help shows that the default value of test_size is 0.25.

| Practicing scikit-learn

```
In [7]: len(data.data)  
Out[7]: 569
```

Line 7

- From the total of 569 observed values, divide the data for training and evaluation into 7:3 or 8:2. 7.5:2.5 is the default value.

| Practicing scikit-learn

- ▶ Use train_test_split() to split the data for making and evaluating the model.

```
In [ ]: train_test_split
```

```
In [8]: 569*0.75
```

```
out[8]: 426.75
```

```
In [9]: from sklearn.model_selection import train_test_split
```

```
In [10]: X_train, X_test,y_train, y_test=train_test_split(data.data,data.target)
```

| Practicing scikit-learn

```
In [11]: len(X_train)
```

```
Out[11]: 426
```

```
In [12]: 569*0.25
```

```
Out[12]: 142.25
```

```
In [13]: len(X_test)
```

```
Out[13]: 143
```

Line 11

- 426 observed values (75%) out of total 569 observations are found.

Line 13

- 143 observed values (25%) out of total 569 observations are found.

Practicing scikit-learn

- ▶ For instancing, use the model's hyperparameter as an argument. Hyperparameter is an option that requires a human setting and greatly affects the model performance.

```
In [1]: 1 from sklearn.datasets import load_breast_cancer  
2 from sklearn.model_selection import train_test_split  
3 from sklearn.tree import DecisionTreeClassifier  
4  
5 data=load_breast_cancer()  
6 X_train, X_test, y_train, y_test=train_test_split(data.data, data.target, random_state=42)  
7  
8 model = DecisionTreeClassifier(criterion='entropy')  
9 model  
  
Out[1]: DecisionTreeClassifier(criterion='entropy')
```

Line 1-5

- Loading the test data set

Line 1-8

- Instancing the estimator and hyperparameter setting
- Model initialization by using the entropy for branching

| fit

- ▶ Use the fit method with an instance estimator for training. Send the training data and label data together as an argument to the supervised learning algorithm.

| predict

- ▶ The instance estimator that has completed training with fitting can be applied with the predict method. 'Predict' converts the estimated results of the model regarding the entered data.

```
In [2]: model.fit(X_train, y_train)
y_pred=model.predict(X_test)
y_pred
```

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

Line 2

- It is an estimated value, so the actual value for X_test may vary.
Measure the accuracy by comparing the two values.

| Practicing scikit-learn

```
In [18]: pred=model.predict(X_test)
```

```
In [19]: pred_2d=pred.reshape(len(pred),1)
```

In [20]: pred_2d

| Practicing scikit-learn

```
In [21]: import numpy as np  
  
In [22]: y_test_2d=y_test.reshape(len(pred),1)  
  
In [23]: df1=pd.DataFrame(pred_2d)  
  
In [24]: df2=pd.DataFrame(y_test_2d)
```

| Practicing scikit-learn

```
In [57]: pd.concat([df1,df2],axis=1)
```

```
Out[57]:
```

	0	0
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1
...

143 rows × 2 columns

 Line 57

- Data frame shows a result where predicted value and actual value differ.

| Practicing scikit-learn

```
In [26]: df_concat=pd.concat([df1,df2],axis=1)
```

```
In [27]: df_concat.columns=['pred','real']
```

```
In [28]: df_concat.columns
```

```
Out[28]: Index(['pred', 'real'], dtype='object')
```

| Practicing scikit-learn

```
In [65]: df_concat[df_concat['pred'] == df_concat['real']]
```

```
Out[65]:
```

	pred	real			
0	1	1	137	1	1
1	1	1	138	1	1
2	1	1	139	1	1
3	1	1	141	1	1
4	1	1	142	1	1
...	133 rows × 2 columns		

Practicing scikit-learn

```
In [66]: len(df_concat[df_concat['pred'] == df_concat['real']])
```

```
Out[66]: 133
```

```
In [68]: 133/143
```

```
Out[68]: 0.9300699300699301
```

Line 66

- 133/143

Practicing scikit-learn

```
In [66]: len(df_concat[df_concat['pred'] == df_concat['real']]) # 133/143
```

```
Out[66]: 133
```

```
In [68]: 133/143
```

```
Out[68]: 0.9300699300699301
```

- ▶ It showed 93% accuracy, which is quite a rare result. In fact, a process of increasing data accuracy is required during data pre-processing, and standardization is one of the options. The following is a brief summary of standardization.
 - Standardization can be done by calculating standard normal distribution. Another term for standardization is z-transformation, and the standardized value is also referred to as z-score. 94% accuracy would be obtained from KNN wine classification through standardization.
 - Standardization is widely used in data pre-processing in general other than KNN, and the following is the equation.

$$x = \frac{x-m}{\sigma} \quad (m: \text{average}, \sigma: \text{standard deviation})$$

- The standardization is available as StandardScaler class in the scikit-learn.

| Practicing scikit-learn

```
In [32]: from sklearn.preprocessing import StandardScaler  
  
In [33]: scaler=StandardScaler()  
  
In [34]: X_train  
  
Out[34]: array([[1.247e+01, 1.731e+01, 8.045e+01, ..., 1.053e-01, 3.035e-01,  
    7.661e-02],  
    [1.706e+01, 2.100e+01, 1.118e+02, ..., 1.827e-01, 2.623e-01,  
    7.599e-02],  
    [1.808e+01, 2.184e+01, 1.174e+02, ..., 9.181e-02, 2.369e-01,  
    6.558e-02],  
    ...,  
    [9.731e+00, 1.534e+01, 6.378e+01, ..., 1.571e-01, 3.108e-01,  
    1.259e-01],  
    [9.683e+00, 1.934e+01, 6.105e+01, ..., 3.846e-02, 2.552e-01,  
    7.920e-02],  
    [1.276e+01, 1.884e+01, 8.187e+01, ..., 8.312e-02, 2.744e-01,  
    7.238e-02]])
```

| Practicing scikit-learn

```
In [35]: df_scale_previous=pd.DataFrame(data['data'])
```

```
In [36]: scaler.fit(X_train)
```

```
Out[36]: StandardScaler()
```

```
In [37]: X_train=scaler.transform(X_train)
```

```
In [38]: df_scale_after=pd.DataFrame(X_train)
```

Line 35

- Data frame before standardization

| Practicing scikit-learn

In [39]: df_scale_previous

out[39]:

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0
...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6

569 rows × 30 columns

Line 39

- The differences among column values are huge before standardization.

| Practicing scikit-learn

In [40]: df_scale_after

out[40]:

	0	1	2	3	4	5	6	7	8	9	...
0	-0.473538	-0.470859	-0.475889	-0.498453	-0.491544	-0.531238	-0.666183	-0.653706	-1.019670	-0.327784	...
1	0.855004	0.382383	0.836277	0.779301	1.116642	0.038826	0.808755	1.306094	-0.275901	-0.291025	...
2	1.150236	0.576617	1.070667	1.086428	-1.598504	-0.334342	0.288007	0.229435	-0.116787	-1.365872	...
3	1.101030	0.387008	1.363655	0.999010	1.905805	2.996152	2.970181	1.836134	1.152431	1.244051	...
4	-0.378022	-0.436174	-0.367065	-0.416281	-1.120030	-0.285118	-0.180023	-0.209674	-0.842054	-0.763016	...
...
421	1.364423	0.606677	1.334356	1.337025	0.398575	0.686714	0.919334	0.997811	0.523373	-0.201332	...
422	-0.389599	-1.381908	-0.398875	-0.428519	-0.586812	-0.469367	-0.609222	-0.607334	-0.742145	-0.189568	...
423	-1.266321	-0.926384	-1.173618	-1.022668	0.782492	1.095292	4.151829	0.768024	2.762080	4.450949	...
424	-1.280214	-0.001460	-1.287883	-1.064919	-0.802232	-1.037096	-0.829736	-1.018335	-0.819852	-0.049882	...
425	-0.389599	-0.117076	-0.416454	-0.450374	0.040253	-0.468589	-0.784605	-0.806034	-0.157490	-0.126342	...

426 rows × 30 columns

Line 40

- After standardization, the column values do not significantly deviate from 0.
- Better performance would be possible compared to the performance before standardization.

| transform

- ▶ Feature processing is done with ‘transform’ to return the result.

```
In [3]: 1 from sklearn.preprocessing import StandardScaler  
2  
3 print(X_train)  
4  
5 scaler = StandardScaler()  
6 scaler.fit(X_train)  
7 X_train = scaler.transform(X_train)  
8  
9 X_train
```

```
[[1.289e+01 1.312e+01 8.189e+01 ... 5.366e-02 2.309e-01 6.915e-02]  
 [1.340e+01 2.052e+01 8.864e+01 ... 2.051e-01 3.585e-01 1.109e-01]  
 [1.296e+01 1.829e+01 8.418e+01 ... 6.608e-02 3.207e-01 7.247e-02]  
 ...  
 [1.429e+01 1.682e+01 9.030e+01 ... 3.333e-02 2.458e-01 6.120e-02]  
 [1.398e+01 1.962e+01 9.112e+01 ... 1.827e-01 3.179e-01 1.055e-01]  
 [1.218e+01 2.052e+01 7.722e+01 ... 7.431e-02 2.694e-01 6.878e-02]]
```

 Line 3-3

- Output before pre-processing

| transform

- ▶ Feature processing is done with ‘transform’ to return the result.

```
In [3]: 1 from sklearn.preprocessing import StandardScaler  
2  
3 print(X_train)  
4  
5 scaler = StandardScaler()  
6 scaler.fit(X_train)  
7 X_train = scaler.transform(X_train)  
8  
9 X_train
```

```
[[1.289e+01 1.312e+01 8.189e+01 ... 5.366e-02 2.309e-01 6.915e-02]  
 [1.340e+01 2.052e+01 8.864e+01 ... 2.051e-01 3.585e-01 1.109e-01]  
 [1.296e+01 1.829e+01 8.418e+01 ... 6.608e-02 3.207e-01 7.247e-02]  
 ...  
 [1.429e+01 1.682e+01 9.030e+01 ... 3.333e-02 2.458e-01 6.120e-02]  
 [1.398e+01 1.962e+01 9.112e+01 ... 1.827e-01 3.179e-01 1.055e-01]  
 [1.218e+01 2.052e+01 7.722e+01 ... 7.431e-02 2.694e-01 6.878e-02]]
```



Line 3-7

- Pre-processing – Apply scaling

| transform

- ▶ Feature processing is done with ‘transform’ to return the result.

```
In [3]: 1 from sklearn.preprocessing import StandardScaler  
2  
3 print(X_train)  
4  
5 scaler = StandardScaler()  
6 scaler.fit(X_train)  
7 X_train = scaler.transform(X_train)  
8  
9 X_train
```

```
[[1.289e+01 1.312e+01 8.189e+01 ... 5.366e-02 2.309e-01 6.915e-02]  
 [1.340e+01 2.052e+01 8.864e+01 ... 2.051e-01 3.585e-01 1.109e-01]  
 [1.296e+01 1.829e+01 8.418e+01 ... 6.608e-02 3.207e-01 7.247e-02]  
 ...  
 [1.429e+01 1.682e+01 9.030e+01 ... 3.333e-02 2.458e-01 6.120e-02]  
 [1.398e+01 1.962e+01 9.112e+01 ... 1.827e-01 3.179e-01 1.055e-01]  
 [1.218e+01 2.052e+01 7.722e+01 ... 7.431e-02 2.694e-01 6.878e-02]]
```



Line 3-9

- Result check after pre-processing

| transform

- ▶ Feature processing is done with ‘transform’ to return the result.

```
Out[3]: array([[-0.34913849, -1.43851335, -0.41172595, ..., -0.91671059,
   -0.92508585, -0.80841115],
   [-0.20468665,  0.31264011, -0.13367256, ...,  1.43655962,
    1.14955889,  1.56911143],
   [-0.32931176, -0.21507235, -0.31739376, ..., -0.7237126 ,
    0.53496977, -0.61934827],
   ...,
   [ 0.04739597, -0.56293662, -0.06529202, ..., -1.23262438,
    -0.68282718, -1.261137 ],
   [-0.04040808,  0.09966199, -0.03151368, ...,  1.08847951,
    0.48944465,  1.26159953],
   [-0.5502381 ,  0.31264011, -0.6040977 , ..., -0.59582424,
    -0.29911546, -0.82948141]])
```

| fit_transform

- ▶ Fit and Transform is combined as fit_transform.

```
In [4]: 1 print(X_test)
2
3 X_test = scaler.fit_transform(X_test)
4
5 X_test
```

[[1.247e+01 1.860e+01 8.109e+01 ... 1.015e-01 3.014e-01 8.750e-02]
[1.894e+01 2.131e+01 1.236e+02 ... 1.789e-01 2.551e-01 6.589e-02]
[1.546e+01 1.948e+01 1.017e+02 ... 1.514e-01 2.837e-01 8.019e-02]
...
[1.104e+01 1.683e+01 7.092e+01 ... 7.431e-02 2.998e-01 7.881e-02]
[1.981e+01 2.215e+01 1.300e+02 ... 2.388e-01 2.768e-01 7.615e-02]
[1.026e+01 1.222e+01 6.575e+01 ... 6.696e-02 2.937e-01 7.722e-02]]

Line 4-1 & 4-5

- Before & After

Line 4-3

- Combination of fit and transform

Major scikit modules

Classification	Module	Embedded functions
Data example	sklearn.datasets	Data set for practicing
Feature processing	sklearn.preprocessing	Pre-processing techniques (One-hot encoding, normalization, scaling, etc.)
	sklearn.feature_selection	Technique to search and select a feature that provides a significant impact to the model
	sklearn.feature_extraction	Feature extraction from source data The supporting API for feature extraction regarding image is present in the submodule image, while the supporting API for text data feature extraction is present in the submodule test.
Dimension reduction	sklearn.decomposition	Algorithms related to dimension reduction (PCA, NMF, Truncated SVD, etc.)
Validation, hyperparameter tuning, data separation	sklearn.model_selection	Validation, hyperparameter tuning, data separation, etc. (cross_validate, GridSearchCV, train_test_split, learning_curve, etc.)
Model evaluation	sklearn.metrics	Techniques to measure and evaluate model performance (accuracy, precision, recall, ROC curve, etc.)

Major scikit modules

Classification	Module	Embedded functions
Machine learning algorithm	sklearn.ensemble	Ensemble algorithms (Random forest, AdaBoost, bagging, etc.)
	sklearn.linear_model	Linear algorithms (Linear regression, logistic regression, SGD, etc.)
	sklearn.naïve_bayes	Naive Bayes algorithms (Bernoulli NB, Gaussian NB, multinomial distribution NB, etc.)
	sklearn.neighbors	Nearest neighbor algorithms (K-NN, etc.)
	sklearn.svm	Support Vector Machine algorithms
	sklearn.tree	Decision tree algorithms
	sklearn.cluster	Unsupervised learning (clustering) algorithms (Kmeans, DBSCAN, etc.)
Utility	sklearn.pipeline	Serial conversion of feature processing and machine learning algorithms, etc.

Unit 1.

Machine Learning Based Data Analysis

- | 1.1. What is machine learning?
- | 1.2. Python scikit-learn library for machine learning
- | **1.3. Preparation and division of data set**
- | 1.4. Data pre-processing for making a good training data set
- | 1.5. Practicing to find an optimal method to solve problems with scikit-learn

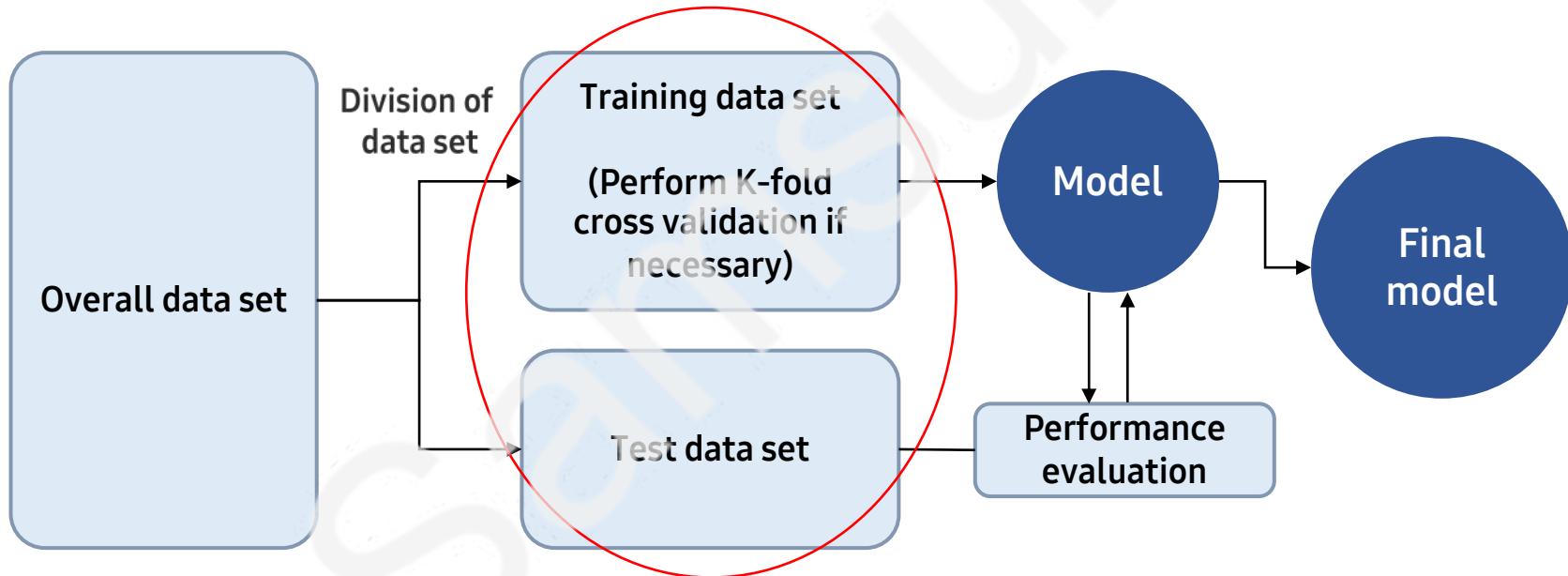
Preparation and division of data set

| Chapter objectives

- ▶ Be able to understand the meaning and ripple effect of overfitting and generalization and design data set division to solve issues.
- ▶ Be able to properly divide the training and test data sets for machine learning technique application according to the analysis purpose and data set features.
- ▶ Be able to divide the training and validation data sets and decide the appropriate k-value for cross-validation by deciding the necessity of cross-validation according to the issue and applied technique. Be able to divide the data sets and perform sampling by considering the prediction results based on data features and classified variable distribution.
- ▶ Be able to analyze differences among various sampling methods for data set division and apply appropriate sampling methods.

Necessity of data set division

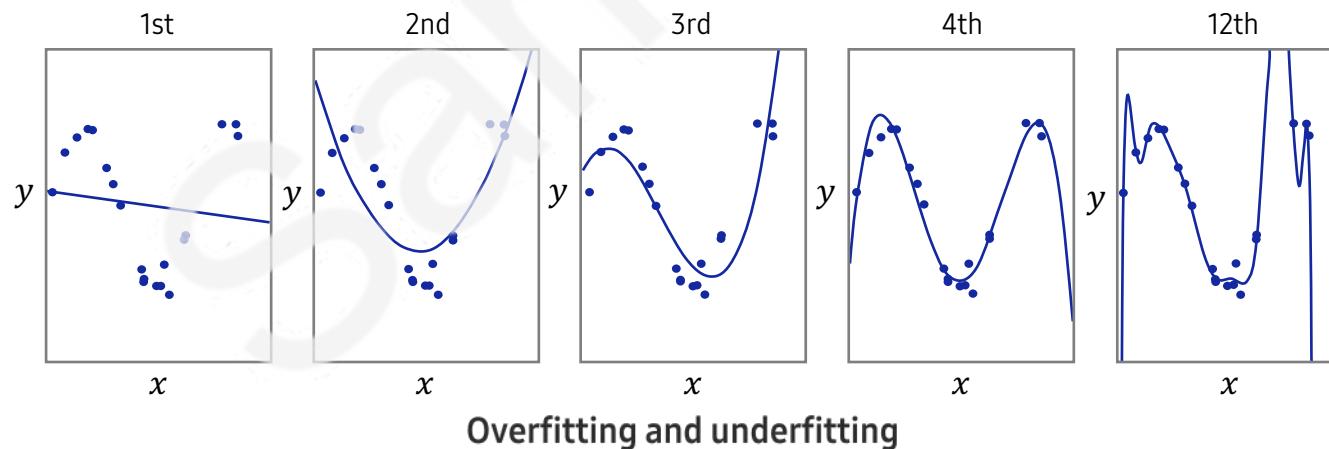
- When analyzing machine learning-based data, especially when applying a supervised learning-based model, do not analyze the overall data set but analyze by dividing the training and evaluation (test) data sets.



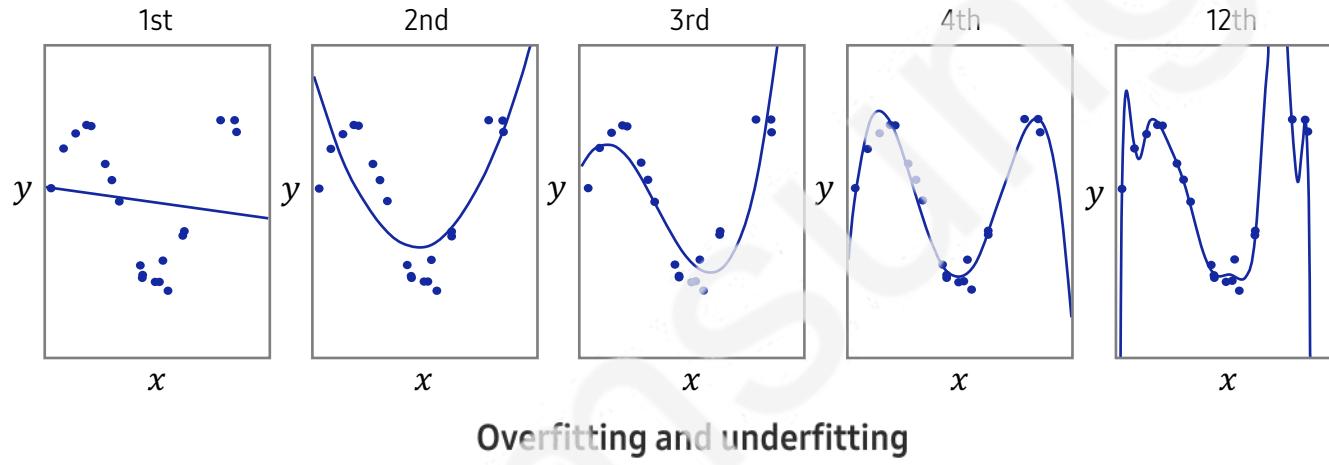
Machine learning modeling process through division of training and test data sets

Overfitting and generalization of modeling

- ▶ Strictly speaking, the data included in the provided training data set can be considered as the values obtained by chance. Hence, a new data set obtained to predict the values of new objective variables (or response variables) is not the same as the existing training data set.
- ▶ The chance that the patterns from the training data and new data to perfectly accord is extremely low. Thus, when learning the machine learning-based model, overfitting occurs in highlighting the training data set pattern when reflecting too much of the training data set. At the same time, generalization for accurate prediction of new data is underperformed.
- ▶ To prevent such issues, the data sets are generally divided into training and test data sets. Measure how the machine learning model learned with the training data set accurately predicts the test data set's objective variables (or response variables). The resulting standard will become the model performance evaluation standard.



| Overfitting and generalization of modeling

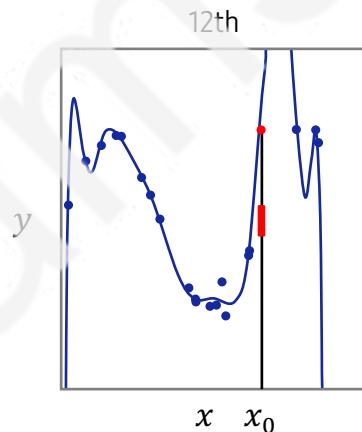


- ▶ Even if machine learning finds the optimal solution in data distribution, a wide margin of error occurs since the model has a small capacity. Such a phenomenon is referred to as underfitting; the linear equation model on the leftmost figure above is an example.
- ▶ An easy alternative is to use higher-degree polynomials, which are non-linear equations.
- ▶ The rightmost figure provided above is applied with the 12th-order polynomial.
- ▶ The model capacity got larger, and there are 13 parameters for estimation.

$$y = w_{12}x^{12} + w_{11}x^{11} + w_{10}x^{10} \dots w_1x^1 + w_0$$

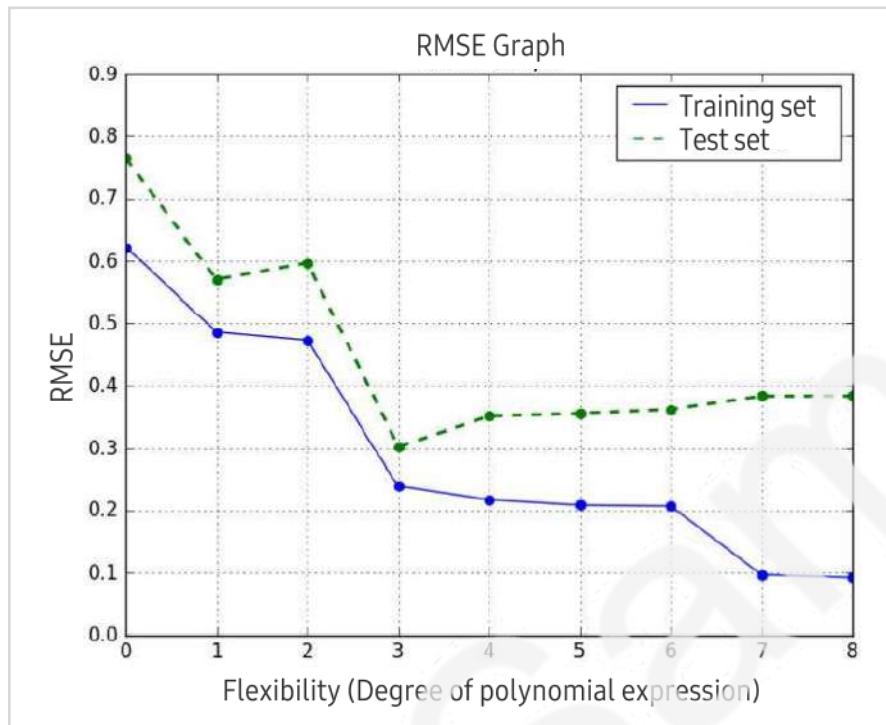
Overfitting

- When choosing a 12th-order polynomial curve, it approximates almost perfectly to the training set.
- However, an issue occurs when predicting new data.
 - The region around the red bar at x_0 should be predicted, but the red dot is predicted instead.
- The reason is because of the **large capacity of the model**.
 - Accepting the noise during the learning process → Overfitting**
- Model selection is required to select an adequate size model.



Inaccurate prediction in overfitting

Overfitting and generalization of modeling



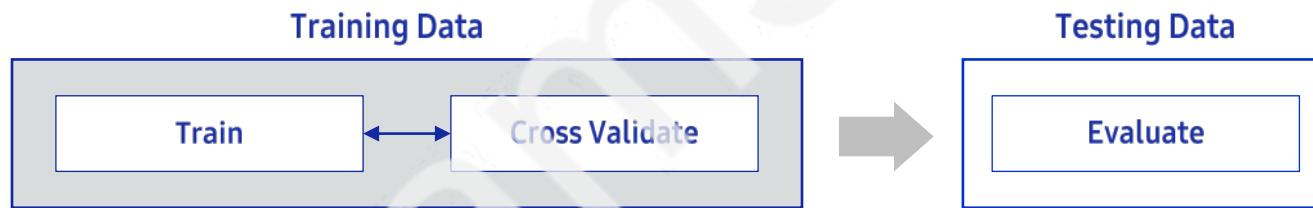
Comparison of the difference between the root means squared errors of training and test data

- ▶ As the flexibility of the machine learning technique increases (in other words, flexibility is increased as the possibility of the given data patterns accurately according with each other rises, followed by increased order of the polynomial.):
 - the root means squared error of the training data set shows a monotone decreasing trend.
 - the root means squared error of the test data set declines in the beginning as the order of polynomial rises, but it increases after a certain point.
- ▶ Summing up, the figure on the left shows an overfitting trend that reflects the training data set pattern too much after the 4th-order polynomial. If there was no root means squared error index calculated with test data, the root means squared error of the training data would decline as the order of polynomial rises, leading to selecting an overfitting model.
- ▶ Thus, test data is required.

Method and process of data set division

| Cross-Validation:

- ▶ The data should be split into a training set and a testing set.
- ▶ In principle, the testing set should be used only once! It should not be reused!
- ▶ If the training set is also used for evaluation, the errors can be unrealistically small.
- ▶ We would like to evaluate realistic errors while training by splitting the training data into two.



| Cross-Validation and Hyperparameter optimization:

- ▶ As we can repeatedly evaluate errors while training, it is also possible to tune the hyperparameters.

| Cross-Validation:

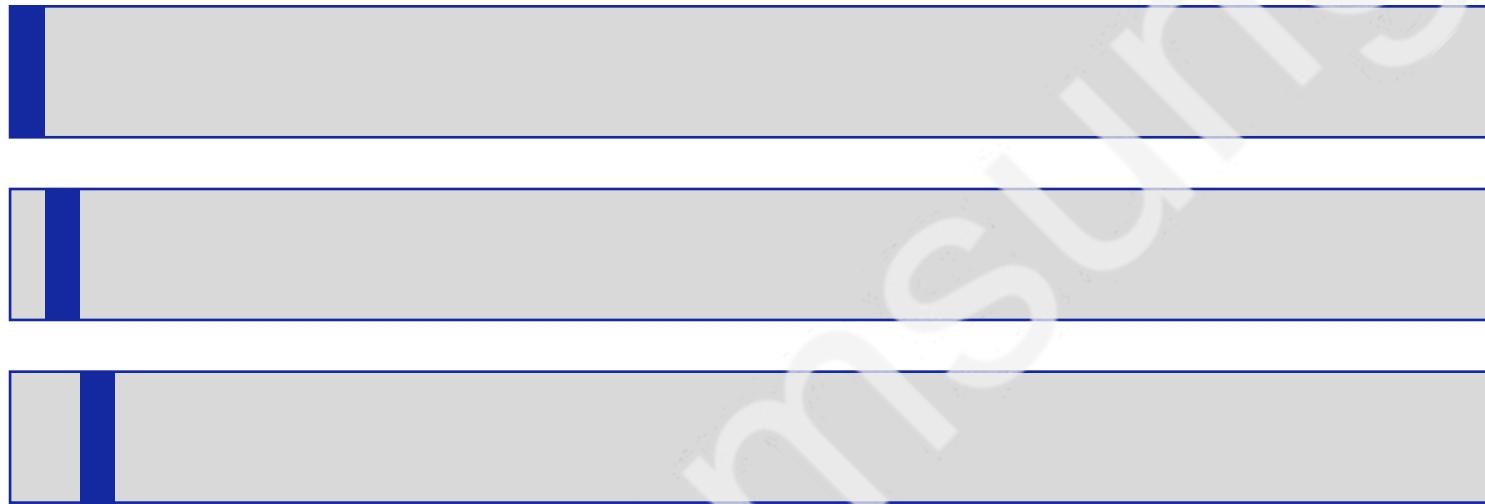
- 1) Split the data into a training set and a testing set.
- 2) Further subdivide the training set into a smaller training and a validation set.
- 3) Train the model with the smaller training set.
- 4) Evaluate the errors with the validation set.
- 5) Repeat from step 2) a few times.

| Cross-Validation method: k-Fold



- ▶ Subdivide the training dataset into k equal parts. Then, apply sequentially.

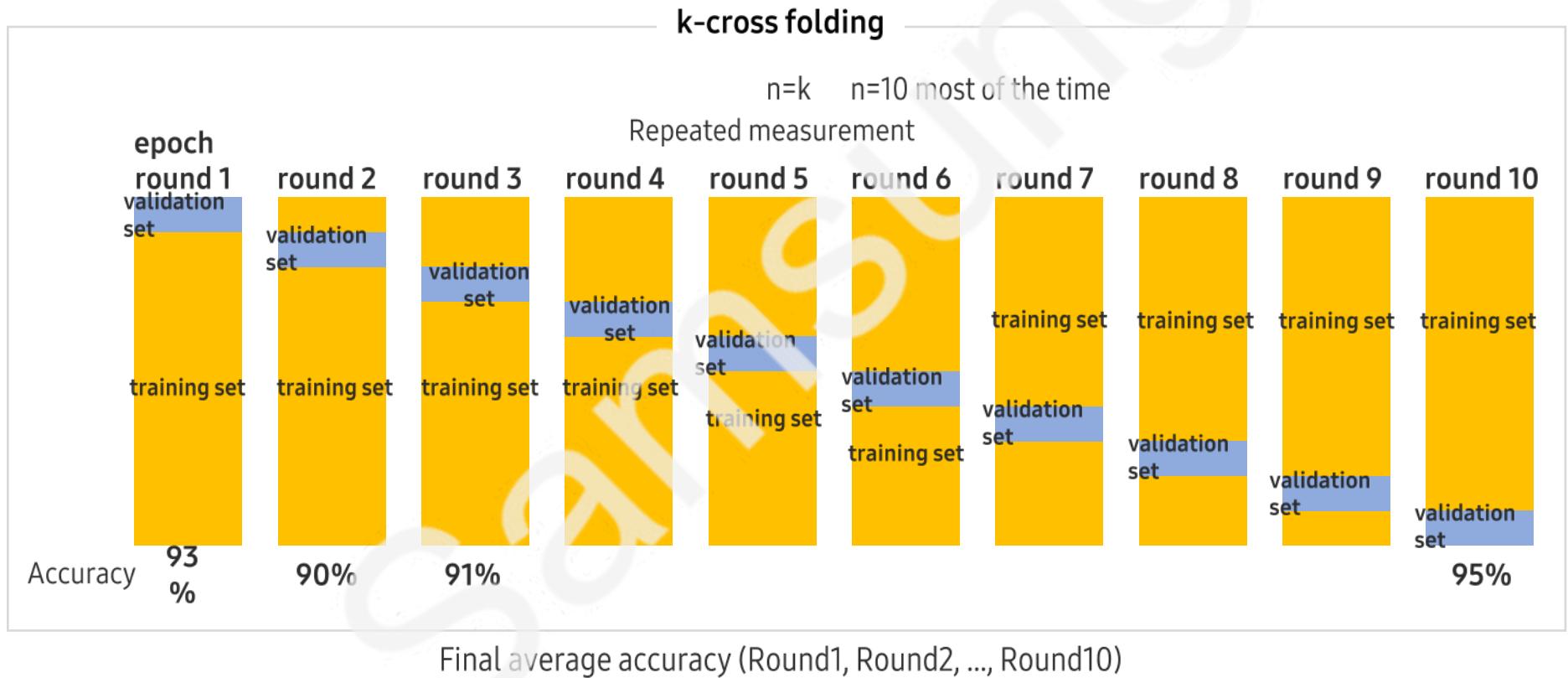
| Cross-Validation method: Leave One Out (LOO)



Validation
 Training

- ▶ Leave only one observation for validation. Apply sequentially. More time consuming.

| Cross-Validation method: k-cross folding



Unit 1.

Machine Learning Based Data Analysis

- | 1.1. What is machine learning?
- | 1.2. Phyton scikit-learn library for machine learning
- | 1.3. Preparation and division of data set
- | **1.4. Data pre-processing for making a good training data set**
- | 1.5. Practicing to find an optimal method to solve problems with scikit-learn

Missing value processing

- | Data cleansing for machine learning-based data analysis uses missing value and noise processing to eliminate discrepancies in collected data.
 - ▶ Missing value processing is done as follows.
 - ▶ First, import the iris data for quick examination.

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

```
In [2]: iris=load_iris()
```

```
In [3]: import pandas as pd
```

```
In [4]: iris['data']
```

```
out[4]: 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],
```

| Missing value processing

```
In [8]: iris_df.columns
```

```
out[8]: Index(['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'], dtype='object')
```

```
In [9]: iris_df['Sepal_Length']
```

```
Out[9]: 0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: Sepal_Length, Length: 150, dtype: float64
```

```
In [10]: import numpy as np
```

```
In [11]: np.mean(iris_df['Sepal_Length'])
```

```
out[11]: 5.843333333333335
```

Missing value processing

1) Ignore the record (row)

- In data classification, ignore the record if the class label is not distinguished.

Ex In the case of the iris data, ignore the fourth row as shown on the table below.

	x1	x2	x3	x4	y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

- 'Ignore the record' is extremely inefficient if missing values frequently occur.

Missing value processing

2) Insert the missing value

- Enter a certain value like 'unknown' for missing value. Or enter the average value of data, such as the overall average value, median value, or class that belongs to the same record.

	x1	x2	x3	x4	y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	unknown
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Missing value processing

2) Insert the missing value

- The average value of Sepal.Length for iris is 5.843 as provided earlier, so insert 5.843.

	x1	x2	x3	x4	y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4		3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

	x1	x2	x3	x4	y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	5.843	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

| Missing value processing

3) Manual entry

- A person in charge (or an expert) should check the data and modify it into an appropriate value.
- It requires a lot of time but provides high reliability.

Missing value processing

- ▶ Identify the missing value from the table type of data below and make appropriate processing.
 - In python, the missing value is specified as np.nan or null value.
 - 'nan' is an abbreviation of Not a Number.

```
In [12]: data=[[1.0,2.0,3.0,4.0],[5.0,6.0,np.nan,8.0],[10.0,11.0,12.0,np.nan]]
```

```
In [13]: np.array(data)
```

```
Out[13]: array([[ 1.,  2.,  3.,  4.],
   [ 5.,  6., nan,  8.],
   [10., 11., 12., nan]])
```

| Missing value processing

```
In [14]: pd.DataFrame(np.array(data) , columns=['A','B','C','D'])
```

```
Out[14]:
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

```
In [15]: df=pd.DataFrame(np.array(data) , columns=['A','B','C','D'])
```

- ▶ The omitted value is changed to NaN. In this case, it is not problematic due to the low number of data, but it is extremely inconvenient to manually find missing values from a huge data frame.

Missing value processing

- ▶ `isnull()` returns the data frame with a Boolean, which shows if the cell has a numerical value (`False`) or is omitted with a numerical value (`True`). Then, `sum()` is used to obtain the number of omissions.
- ▶ It is mandatory to check the number of missing values when importing data.

```
In [16]: df.isnull()
```

```
Out[16]:
```

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

```
In [17]: df.isnull().sum()
```

```
Out[17]: A    0  
          B    0  
          C    1  
          D    1  
          dtype: int64
```

Line 17

- The number of missing values can be counted.

| Removing the training sample or feature with missing value

- ▶ Use dropna() to completely delete a certain training sample (row) or feature (column).
- ▶ help(df.dropna) shows axis=0 is default.

```
In [18]: df.dropna()
```

```
Out[18]:
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

Line 18

- axis=0 is default, so the row with the NaN value is deleted.

| Removing the training sample or feature with missing value

- ▶ When trying to reflect the deleted result immediately to the object, do not omit `inplace=True` option.

```
In [19]: df.dropna(inplace=True)
```

```
In [20]: df
```

Out[20]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

| Removing the training sample or feature with missing value

- ▶ Delete the row with missing value.

```
In [21]: df=pd.DataFrame(np.array(data) , columns=['A','B','C','D'])
```

```
In [22]: df
```

```
out[22]:
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

| Removing the training sample or feature with missing value

```
In [23]: df.dropna(axis=1, inplace=True)
```

```
In [24]: df
```

```
Out[24]:
```

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

| Removing the training sample or feature with missing value

- ▶ If all rows have NaN, use how='all' to delete.

```
In [25]: df=pd.DataFrame(np.array(data) , columns=['A','B','C','D'])
```

```
In [26]: df
```

Out[26]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

| Removing the training sample or feature with missing value

```
In [27]: df.loc[3]=[np.nan,np.nan,np.nan,np.nan]
```

```
In [28]: df
```

Out[28]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN
3	NaN	NaN	NaN	NaN

| Removing the training sample or feature with missing value

```
In [29]: df.dropna(how='all', inplace=True)
```

```
In [30]: df
```

Out[30]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

| Removing the training sample or feature with missing value

- ▶ When deleting 3 or more NaN values, use thresh.

```
In [31]: df=pd.DataFrame(np.array(data) , columns=['A','B','C','D'])
df.loc[3]=[np.nan,np.nan,np.nan,3]
```

```
In [32]: df
```

Out[32]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN
3	NaN	NaN	NaN	3.0

| Removing the training sample or feature with missing value

```
In [33]: df.dropna(thresh=3, inplace=True)
```

```
In [34]: df
```

```
Out[34]:
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

| Removing the training sample or feature with missing value

- ▶ When deleting a row with NaN on a certain column, use subset.

```
In [35]: df
```

```
Out[35]:
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

```
In [36]: df.dropna(subset=['C'], inplace=True)
```

```
In [37]: df
```

```
Out[37]:
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
2	10.0	11.0	12.0	NaN

Imputation

- ▶ It is sometimes hard to delete a training sample or a certain column because it loses too much useful data. If so, estimate missing values from other training samples in the data set by kriging. The most commonly used method is to impute with average value, which is to change the missing value into the overall average of a certain column. In scikit-learn, use SimpleImputer class.

```
In [38]: from sklearn.impute import SimpleImputer
```

```
In [39]: df=pd.DataFrame(np.array(data) , columns=['A','B','C','D'])
```

```
In [40]: impt=SimpleImputer(missing_values=np.nan, strategy='mean')
```

- ▶ Impute with using df.values.

```
In [41]: df.values
```

```
out[41]: array([[ 1.,  2.,  3.,  4.],
   [ 5.,  6., nan,  8.],
   [10., 11., 12., nan]])
```

Imputation

```
In [42]: help(impt.fit)
```

Help on method fit in module sklearn.impute._base:

fit(X, y=None) method of sklearn.impute._base.SimpleImputer instance
Fit the imputer on X.

Parameters

X : {array-like, sparse matrix}, shape (n_samples, n_features)
Input data, where ``n_samples`` is the number of samples and
``n_features`` is the number of features.

Returns

self : SimpleImputer

Imputation

```
In [43]: imputed=impt.fit(df.values)
```

```
In [44]: imputed.transform(df.values)
```

```
Out[44]: array([[ 1. ,  2. ,  3. ,  4. ],
   [ 5. ,  6. ,  7.5,  8. ],
   [10. , 11. , 12. ,  6. ]])
```

```
In [45]: (3+12)/2
```

```
Out[45]: 7.5
```

Line 45

- Check it is the average of the column.
- ▶ For strategy parameters, median and most_frequent can also be set.

| Review on the scikit-learn estimator API

- ▶ Impute the missing value of the data set by using the SimpleImputer class of scikit-learn from the previous clause.
- ▶ The SimpleImputer class is a transformer class of scikit-learn that is used for data conversion.
- ▶ The two main methods for the estimator include fit and transform.
- ▶ Use the fit method to learn the model parameter in the training data.
- ▶ Use the transform method to convert the data into the learned parameter.
- ▶ The data array for conversion should be the same as the number of data features used in the model learning.

I Categorical data processing

- ▶ Data is generally classified into categorical and continuous scales depending on their features.
- ▶ In liberal arts and social science, questionnaires are mainly used to collect data.
- ▶ **Categorical scale**
 - It is a scale that can distinguish data into different categories and is classified into the nominal and ordinal scales.
- ▶ **Continuous scale**
 - It is a scale that divides linked data into the purpose of the survey. It is classified into interval and ratio scales.
- ▶ Actual data sets would include more than one categorical feature. As explained earlier, categorical data would classify sequential and non-sequential features. Ordinal scale can be referred to as sequential categorical scale that can array features with sequences.

| Categorical data processing

```
In [46]: df = pd.DataFrame([['green', 'M', 10.1, 'class2'],
                           ['red', 'L', 13.5, 'class1'],
                           ['blue', 'XL', 15.3, 'class2']])
```

```
In [47]: df
```

Out[47]:

	0	1	2	3
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

- ▶ The data on the table has features that have orders and do not have orders. The size is ordered, but the color is not ordered. Thus, the size is classified as the ordinal scale, while the color is the nominal scale.

| Categorical data processing

- ▶ Convert ordered data into numerical values. The reason why changing text data into numerical data is to allow a computer to process arithmetic operations.

```
In [48]: size_mapping = {'XL': 3,  
                      'L': 2,  
                      'M': 1}
```

```
In [49]: df.columns= ['color', 'size', 'price', 'classlabel']
```

```
In [50]: df
```

```
out[50]:
```

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

| Categorical data processing

```
In [51]: df['size']
```

```
Out[51]: 0    M  
1    L  
2    XL  
Name: size, dtype: object
```

```
In [52]: df['size']=df['size'].map(size_mapping)
```

```
In [53]: df['size']
```

```
Out[53]: 0    1  
1    2  
2    3  
Name: size, dtype: int64
```

```
In [54]: df
```

```
Out[54]:
```

	color	size	price	classlabel
0	green	1	10.1	class2
1	red	2	13.5	class1
2	blue	3	15.3	class2

| Class label encoding

- ▶ The class refers to the y value, a column with an actual value.
- ▶ Create a mapping to convert the class label from strings to integers.

```
In [55]: df['classlabel']
```

```
Out[55]: 0    class2  
1    class1  
2    class2  
Name: classlabel, dtype: object
```

```
In [56]: df['classlabel'].value_counts()
```

```
Out[56]: class2    2  
class1    1  
Name: classlabel, dtype: int64
```

```
In [57]: np.unique(df['classlabel'])
```

```
Out[57]: array(['class1', 'class2'], dtype=object)
```

| Class label encoding

- ▶ ‘enumerate’ creates an object with an index.

```
In [58]: enumerate(np.unique(df['classlabel']))
```

```
Out[58]: <enumerate at 0x1b17cb40140>
```

```
In [59]: for idx, label in enumerate(np.unique(df['classlabel'])):  
    print(idx, label)
```

```
0 class1  
1 class2
```

- ▶ ‘enumerate’ creates an object with an index.

```
In [60]: {label : idx for idx, label in enumerate(np.unique(df['classlabel']))}
```

```
Out[60]: {'class1': 0, 'class2': 1}
```

```
In [61]: class_mapping={label : idx for idx, label in enumerate(np.unique(df['classlabel']))}
```

| Class label encoding

- ▶ ‘enumerate’ creates an object with an index.

```
In [62]: df['classlabel'] = df['classlabel'].map(class_mapping)  
df
```

Out[62]:

	color	size	price	classlabel
0	green	1	10.1	1
1	red	2	13.5	0
2	blue	3	15.3	1

Line 62

- Change the class label from strings to integers.

| Class label encoding

- ▶ Since the method in the previous slide is rather inconvenient, scikit-learn supports LabelEncoder for easy conversion.

```
In [63]: df = pd.DataFrame([['green', 'M', 10.1, 'class2'],
                           ['red', 'L', 13.5, 'class1'],
                           ['blue', 'XL', 15.3, 'class2']])
df.columns= ['color', 'size', 'price', 'classlabel']
```

```
In [64]: df
```

```
out[64]:
```

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

| Class label encoding

```
In [65]: from sklearn.preprocessing import LabelEncoder  
  
In [66]: enc=LabelEncoder()  
  
In [67]: y=enc.fit_transform(df['classlabel'].values)  
  
In [68]: y  
Out[68]: array([1, 0, 1])
```

Application of one-hot encoding to unordered feature

There are cases when it is impossible to directly use categorical data in machine learning algorithms such as regression analysis, etc. If so, conversion is required to be recognized by a computer.

- ▶ In such cases, use a dummy variable expressed as 0 or 1. 0 or 1 does not represent how the number is large or small but shows whether a certain feature is present.
 - ▶ If a certain feature is present, it is expressed as 1; if it's not found, it is classified as 0. Likewise, one-hot encoding converts categorical data to a one-hot vector consisting of 0 or 1 that can be recognized by a computer.
 - ▶ Practice with the `iris.target` object.

```
In [69]: iris.target
```

| The encoding is done with integers, so insert the iris 'species' value.

```
In [70]: iris=pd.DataFrame(iris.target, columns=['species'])
```

```
In [71]: iris
```

```
Out[71]:
```

species			
0	0	145	2
1	0	146	2
2	0	147	2
3	0	148	2
4	0	149	2
...	...	150 rows × 1 columns	

| The encoding is done with integers, so insert the iris 'species' value.

```
In [72]: iris['species'].replace({0:'setosa',1:'versicolor', 2:'virginica'}, inplace=True)
```

```
In [73]: iris
```

```
Out[73]:
```

species	
0	setosa
1	setosa
2	setosa
3	setosa
4	setosa
...	...
	145 virginica
	146 virginica
	147 virginica
	148 virginica
	149 virginica
	150 rows × 1 columns

- | Use the get_dummies() function of pandas to convert every eigenvalue of categorical variables into new dummy variable.

```
In [74]: pd.get_dummies(iris['species'])
```

```
Out[74]:
```

	setosa	versicolor	virginica
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
...

- Use sklearn library to conveniently process one-hot encoding. The result is given as a sparse matrix in linear algebra. In the sparse matrix, the value of most matrices is 0. An opposite concept to the sparse matrix is a dense matrix.

Example of sparse matrix

$$\begin{pmatrix} 11 & 22 & 0 & 0 & 0 & 0 & 0 \\ 0 & 33 & 44 & 0 & 0 & 0 & 0 \\ 0 & 0 & 55 & 66 & 77 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 99 \end{pmatrix}$$

Only 9 out of the above 35 coefficients are not 0
in the above sparse matrix.

| OneHotEncoder

```
In [75]: from sklearn.preprocessing import OneHotEncoder  
In [76]: onehot_encoder=OneHotEncoder()  
In [77]: len_iris=len(iris['species'])
```

1.4. Data pre-processing for making a good training data set

UNIT 01

OneHotEncoder

```
In [78]: np.array(iris['species'])
```

OneHotEncoder

```
In [79]: iris_species_2d=np.array(iris['species']).reshape(len_iris,1)
```

```
In [80]: onehot_encoder.fit_transform(iris_species_2d)
```

```
Out[80]: <150x3 sparse matrix of type '<class 'numpy.float64'>'  
with 150 stored elements in Compressed Sparse Row format>
```

```
#Convert to a sparse matrix
```

```
onehot_reshaped=onehot_encoder.fit_transform(iris_species_2d)
```

Conversion to the sparse matrix

```
In [82]: print(onehot_reshaped[0:50])
```

(0, 0)	1.0	(17, 0)	1.0	(32, 0)	1.0
(1, 0)	1.0	(18, 0)	1.0	(33, 0)	1.0
(2, 0)	1.0	(19, 0)	1.0	(34, 0)	1.0
(3, 0)	1.0	(20, 0)	1.0	(35, 0)	1.0
(4, 0)	1.0	(21, 0)	1.0	(36, 0)	1.0
(5, 0)	1.0	(22, 0)	1.0	(37, 0)	1.0
(6, 0)	1.0	(23, 0)	1.0	(38, 0)	1.0
(7, 0)	1.0	(24, 0)	1.0	(39, 0)	1.0
(8, 0)	1.0	(25, 0)	1.0	(40, 0)	1.0
(9, 0)	1.0	(26, 0)	1.0	(41, 0)	1.0
(10, 0)	1.0	(27, 0)	1.0	(42, 0)	1.0
(11, 0)	1.0	(28, 0)	1.0	(43, 0)	1.0
(12, 0)	1.0	(29, 0)	1.0	(44, 0)	1.0
(13, 0)	1.0	(30, 0)	1.0	(45, 0)	1.0
(14, 0)	1.0	(31, 0)	1.0	(46, 0)	1.0
(15, 0)	1.0			(47, 0)	1.0
(16, 0)	1.0			(48, 0)	1.0
				(49, 0)	1.0

 Line 82

- (0, 0) is 1, thus setosa (setosa up to 50 matrices).

| Refer to the figure below for easier understanding.

```
In [83]: print(onehot_reshaped[50:100])
```

(0, 1)	1.0	(22, 1)	1.0	(40, 1)	1.0
(1, 1)	1.0	(23, 1)	1.0	(41, 1)	1.0
(2, 1)	1.0	(24, 1)	1.0	(42, 1)	1.0
(3, 1)	1.0	(25, 1)	1.0	(43, 1)	1.0
(4, 1)	1.0	(26, 1)	1.0	(44, 1)	1.0
(5, 1)	1.0	(27, 1)	1.0	(45, 1)	1.0
(6, 1)	1.0	(28, 1)	1.0	(46, 1)	1.0
(7, 1)	1.0	(29, 1)	1.0	(47, 1)	1.0
(8, 1)	1.0	(30, 1)	1.0	(48, 1)	1.0
(9, 1)	1.0	(31, 1)	1.0	(49, 1)	1.0
(10, 1)	1.0	(32, 1)	1.0		
(11, 1)	1.0	(33, 1)	1.0		
(12, 1)	1.0	(34, 1)	1.0		
(13, 1)	1.0	(35, 1)	1.0		
(14, 1)	1.0	(36, 1)	1.0		
(15, 1)	1.0	(37, 1)	1.0		
(16, 1)	1.0	(38, 1)	1.0		
(17, 1)	1.0	(39, 1)	1.0		
(18, 1)	1.0				
(19, 1)	1.0				
(20, 1)	1.0				
(21, 1)	1.0				

| Refer to the figure below for easier understanding.

```
In [84]: print(onehot_reshaped[101:150])
```

(0, 2)	1.0	(21, 2)	1.0	(40, 2)	1.0
(1, 2)	1.0	(22, 2)	1.0	(41, 2)	1.0
(2, 2)	1.0	(23, 2)	1.0	(42, 2)	1.0
(3, 2)	1.0	(24, 2)	1.0	(43, 2)	1.0
(4, 2)	1.0	(25, 2)	1.0	(44, 2)	1.0
(5, 2)	1.0	(26, 2)	1.0	(45, 2)	1.0
(6, 2)	1.0	(27, 2)	1.0	(46, 2)	1.0
(7, 2)	1.0	(28, 2)	1.0	(47, 2)	1.0
(8, 2)	1.0	(29, 2)	1.0	(48, 2)	1.0
(9, 2)	1.0	(30, 2)	1.0		
(10, 2)	1.0	(31, 2)	1.0		
(11, 2)	1.0	(32, 2)	1.0		
(12, 2)	1.0	(33, 2)	1.0		
(13, 2)	1.0	(34, 2)	1.0		
(14, 2)	1.0	(35, 2)	1.0		
(15, 2)	1.0	(36, 2)	1.0		
(16, 2)	1.0	(37, 2)	1.0		
(17, 2)	1.0	(38, 2)	1.0		
(18, 2)	1.0	(39, 2)	1.0		
(19, 2)	1.0				
(20, 2)	1.0				

| Refer to the figure below for easier understanding.

	Row index	0	1	2	
Column index	Species	setosa	versicolor	virginica	Sparse matrix expression
0	setosa	1	0	0	(0,0)
1	setosa	1	0	0	(1,0)
:	setosa	1	0	0	
49	setosa	1	0	0	(49,0)
50	versicolor	0	1	0	(50,1)
51	versicolor	0	1	0	(51,1)
:	versicolor	0	1	0	
100	versicolor	0	1	0	
101	virginica	0	0	1	(101,2)
	virginica	0	0	1	(102,2)
:	virginica	0	0	1	
150	virginica	0	0	1	(150,2)

| Using hold-out in real life that splits the data set into training and test data sets

- ▶ df_wine is the data that measure wines produced in Vinho Verde, which is adjacent to the Atlantic Ocean in the northwest of Portugal. It measured and analyzed the grade, taste, and acidity of 1,599 red wine and 4,898 white wine samples to create data. If the data is not found in the following route, it is possible to import the data from local by directly downloading it from the UCI repository.

| Using hold-out in real life that splits the data set into training and test data sets

```
In [85]: df_wine = pd.read_csv('https://archive.ics.uci.edu/'  
                           'ml/machine-learning-databases/wine/wine.data',  
                           header=None)  
  
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',  
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',  
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',  
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',  
                   'Proline']  
  
print('Class labels', np.unique(df_wine['Class label']))  
df_wine.head()  
  
Class labels [1 2 3]
```

Line 85

- When it is not accessible to the wine data set of the UCI machine learning repository, remove the remark of the following code and read the data set from the local route:
- `df_wine = pd.read_csv('wine.data', header=None)`

1.4. Data pre-processing for making a good training data set

UNIT 01

| Using hold-out in real life that splits the data set into training and test data sets

Out[85]:

Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64 1.04
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38 1.05
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68 1.03
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80 0.86
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32 1.04

I Using hold-out in real life that splits the data set into training and test data sets

- ▶ Data splitting is possible by using the `train_test_split` function provided in the `model_selection` module of scikit-learn. First, convert the features from index 1 to 13 to NumPy array and assign to variable X. With the `train_test_split` function, data conversion is done in four tuples, so assign by designating appropriate variables.

```
In [86]: from sklearn.model_selection import train_test_split  
  
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values  
  
X_train, X_test, y_train, y_test =\  
    train_test_split(X, y,  
                    test_size=0.3,  
                    random_state=0,  
                    stratify=y)
```

- ▶ Randomly split X and y into training and test data sets. `test_size=0.3`, so 30% of the sample is assigned to `X_test` and `y_test`.
- ▶ Regarding the `stratify` parameter, if the class label array `y` is sent, the class ratio found in the training and test data sets is identically maintained with the original data set.
- ▶ The most widely used ratios in real life are 6:4, 7:3, and 8:2, depending on the size of the data set. It is common and suitable for large data sets to split the training data set and test data set into the ratio of 9:1 or 9.9:0.1.

| Arranging the scale between features (variables)

- ▶ Refer to the practical code (Chapter5_Unit1_Machine Learning-Based Data Analysis 1 (Supervised Learning)) for detailed code.

```
In [16]: from sklearn.preprocessing import MinMaxScaler  
  
mms = MinMaxScaler()  
X_train_norm = mms.fit_transform(X_train)  
X_test_norm = mms.transform(X_test)
```

```
In [17]: from sklearn.preprocessing import StandardScaler  
  
stdsc = StandardScaler()  
X_train_std = stdsc.fit_transform(X_train)  
X_test_std = stdsc.transform(X_test)
```

| Arranging the scale between features (variables)

- ▶ Refer to the practical code (Chapter5_Unit1_Machine Learning-Based Data Analysis 1 (Supervised Learning)) for detailed code.

```
In [22]: from sklearn.preprocessing import RobustScaler  
rbs = RobustScaler()  
X_train_robust = rbs.fit_transform(X_train)  
X_test_robust = rbs.fit_transform(X_test)
```

| Arranging the scale between features (variables)

- ▶ Refer to the practical code (Chapter5_Unit1_Machine Learning-Based Data Analysis 1 (Supervised Learning)) for detailed code.

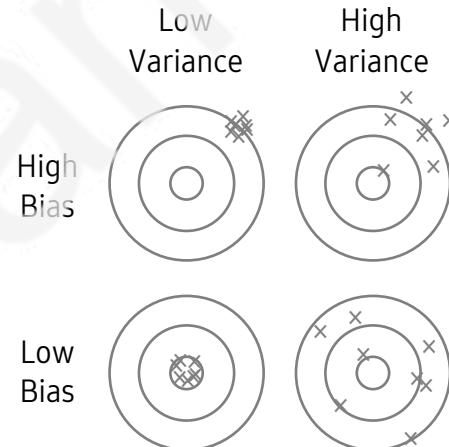
```
In [25]: #MaxAbsScaler divides the data into the maximum absolute value based on each feature. Thus, the  
#maximum value of each feature becomes 1.  
#The overall feature changes to [-1, 1] range.
```

```
In [26]: from sklearn.preprocessing import MaxAbsScaler  
mas = MaxAbsScaler()  
X_train_maxabs = mas.fit_transform(X_train)  
X_test_maxabs = mas.fit_transform(X_test)
```

Limiting model complexity through L1 and L2 regularizations

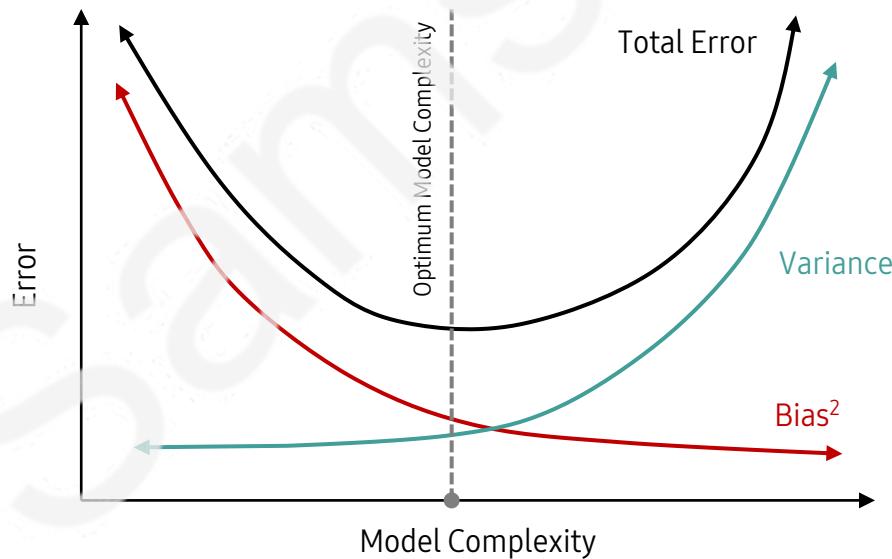
Bias and Variance

- If the predicted values are highly deviated from the actual target value in general, it is said that there's a high bias in the result. When the predicted values are scattered far away from one another, it is said that it has a high variance in the result.
- The following figure expresses the predicted results of the model on the target. High bias refers to the result when it significantly deviates from the target's center. On the other hand, if the predicted values are gathered around the target's center, it has a low bias. Bias refers to the overall similarity between the predicted values and actual values. In the figure, high variance is when the predicted values are greatly apart from one another. On the contrary, low variance is when the predicted values are closely gathered together. Thus, variance refers to the overall similarity among predicted values.



| trade-off

- ▶ Bias and variance have a trade-off relationship in which when one increases, the other falls, and vice versa. The model becomes complex at the beginning of learning; the overall error cost falls due to decreased bias. However, at some point, the model keeps learning and becomes much more complicated, which causes higher variance and increased overall error cost. In other words, the model gets overfitted to the training data. One way to prevent overfitting is to stop learning at the appropriate time. Regularization is a method to prevent overfitting by lowering variance. Still, it can increase bias instead due to the trade-off relationship.



Ridge Regression

- The ridge regression model is a technique to limit the L2norm of w, which is the regression coefficient vector. A constraint is added to minimize the sum of squares of weight in the cost function of linear regression. If the linear regression model is as follows:

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = Xw \quad (X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix})$$

- Then, the cost function of the ridge regression model is as follows. N is the number of data, and M is the number of elements of the regression coefficient vector. A constraint is added to the existing SSE (Sum of Squared Errors).

$$\hat{w}^{ridge} = \operatorname{argmin}_w \left\{ \sum_{i=1}^N (y_i - wX)^2 + \lambda \sum_{j=1}^M w_j^2 \right\}$$

- λ is a hyperparameter to adjust the weight of existing SSE and added constraint.
When the λ is large, regularization is greatly applied, and the regression coefficients become lower. When the λ becomes smaller, regularization gets weaker. When the λ equals 0, the constraint clause also becomes 0, the same as the general linear regression model.

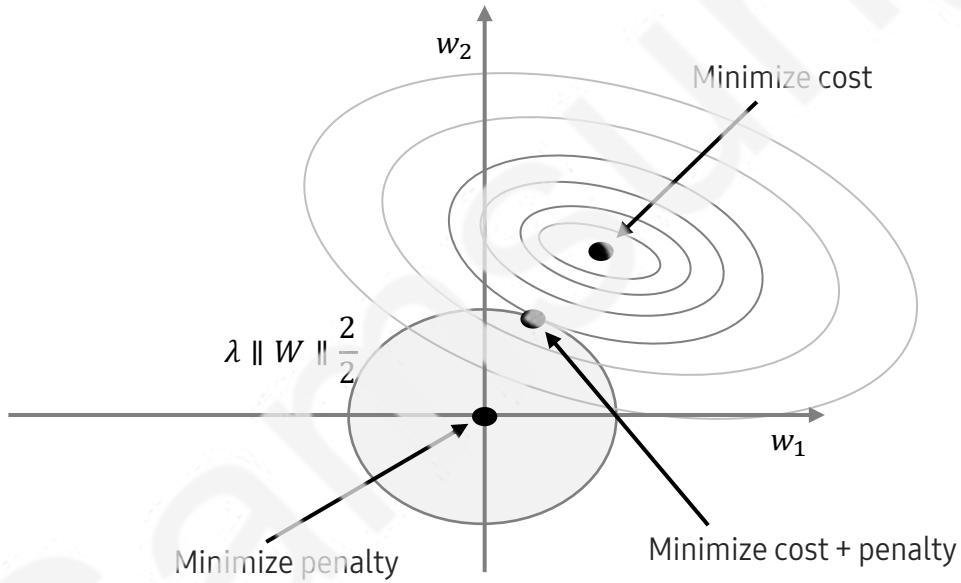
| Ridge Regression

- ▶ The following is an example of simple linear regression model equation:

$$\begin{aligned} SSE(w_1, w_2) &= \sum_{i=1}^n \{y_i - (w_1 x_i + w_2)\}^2 \\ &= \left(\sum_{i=1}^n x_i^2 \right) w_1^2 + (n) w_2^2 + 2 \left(\sum_{i=1}^n x_i^2 \right) w_1 w_2 \\ &\quad - 2 \left(\sum_{i=1}^n y_i \right) w_2 + \sum_{i=1}^n y_i^2 \end{aligned}$$

Ridge Regression

- When drawing the cost function SSE (w_1, w_2) on the coordinate with the x-axis and y-axis, an ellipse is created as provided in the following figure:



- In the figure above, the ellipse drawn in a solid line is the cost function, which is the combination of w_1 and w_2 with the same cost (SSE). The central point of the ellipse is when the cost becomes 0. Outward of the ellipse is the combination of w_1 and w_2 with higher cost, which is the model with higher error (consists of w_1 and w_2 weights). The colored circle refers to the constraint. The circle becomes smaller when the λ gets larger, and vice versa. The point where the cost function (ellipse) and constraint (colored circle) meet is the optimal solution where the cost of the ridge regression model is minimum.

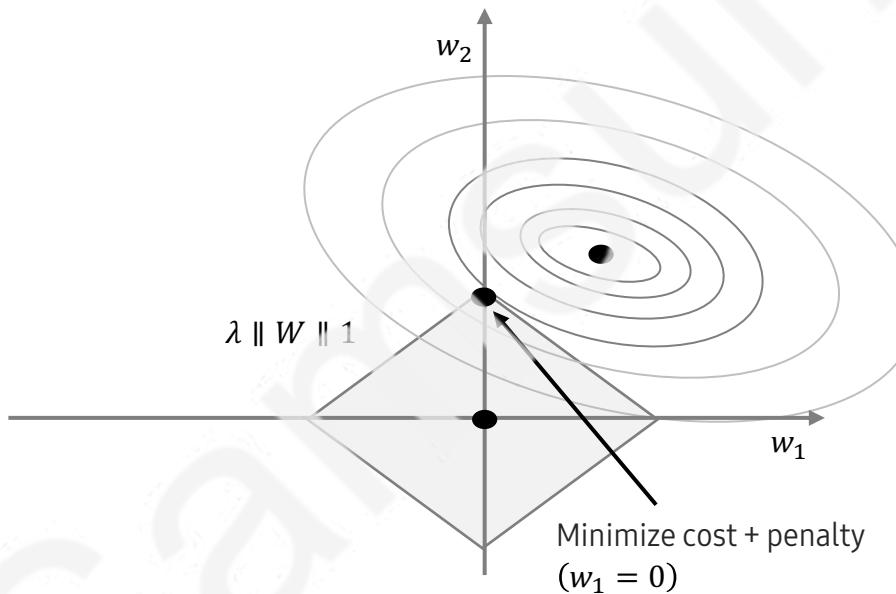
| Lasso Regression

- ▶ The Lasso (Least Absolute Shrinkage and Selection Operator) regression model is a technique to limit the L1norm of the regression coefficient vector w . A constraint is added to minimize the sum of the absolute values of the weights in the cost function of linear regression. The following figure shows the cost function of the Lasso regression model:

$$\hat{w}^{lasso} = \operatorname{argmin}_w \left\{ \sum_{i=1}^N (y_i - wX)^2 + \lambda \sum_{j=1}^M |w_j| \right\}$$

| Lasso Regression

- When drawing the cost function of the Lasso regression model (w_1, w_2) on the coordinate with the x-axis and y-axis, a rhombus is created as provided in the following figure:



- Since the constraint of the Lasso regression model is a rhombus, it is highly possible that the point meeting the cost function is the vertex of the rhombus. The vertexes of the rhombus are always the points where w_1 or w_2 are 0. Thus, the Lasso regression model results in 0 weight.

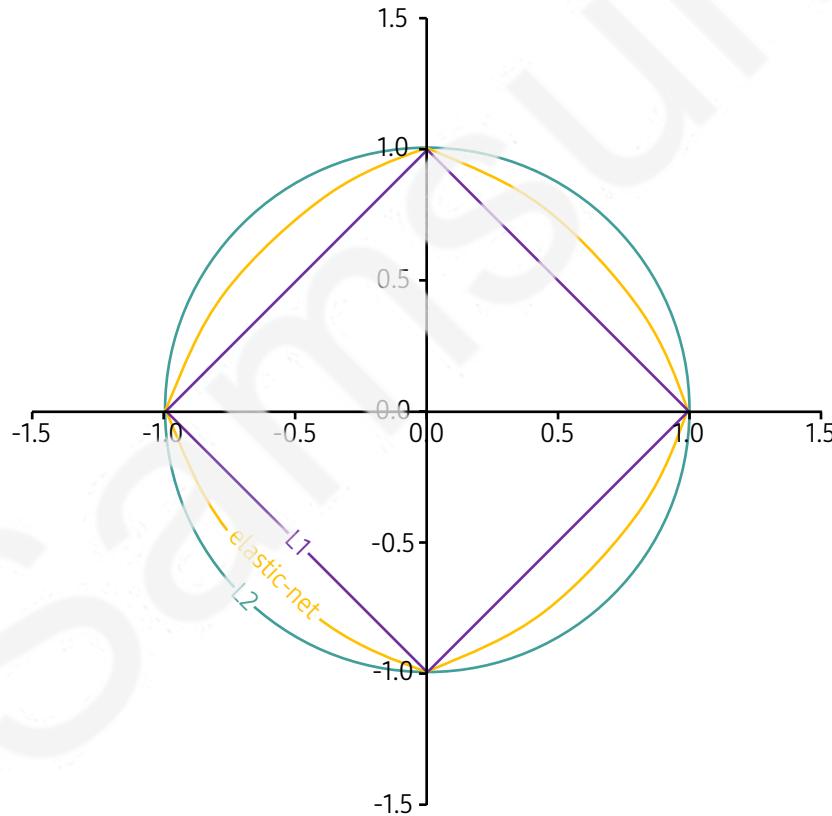
| Elastic-net regression

- ▶ The Elastic-net regression model applies both the L2norm and L1norm to the regression coefficient vector. The constraint is both the sum of weight squares and the sum of the absolute weight values. The following figure shows the cost function of Elastic-net. There are two hyperparameters of Elastic-net, which are λ_1 and λ_2 .

$$\hat{w}^{elastic} = \operatorname{argmin}_w \left\{ \sum_{i=1}^N (y_i - wX)^2 + \lambda_1 \sum_{j=1}^M w_j^2 + \lambda_2 \sum_{j=1}^M |w_j| \right\}$$

Elastic-net regression

- ▶ Elastic-net applies both the L2norm and L1norm at the same time, so the constraint is somewhere in the middle. It reduces a larger weight while making an unimportant weight 0.



Unit 1.

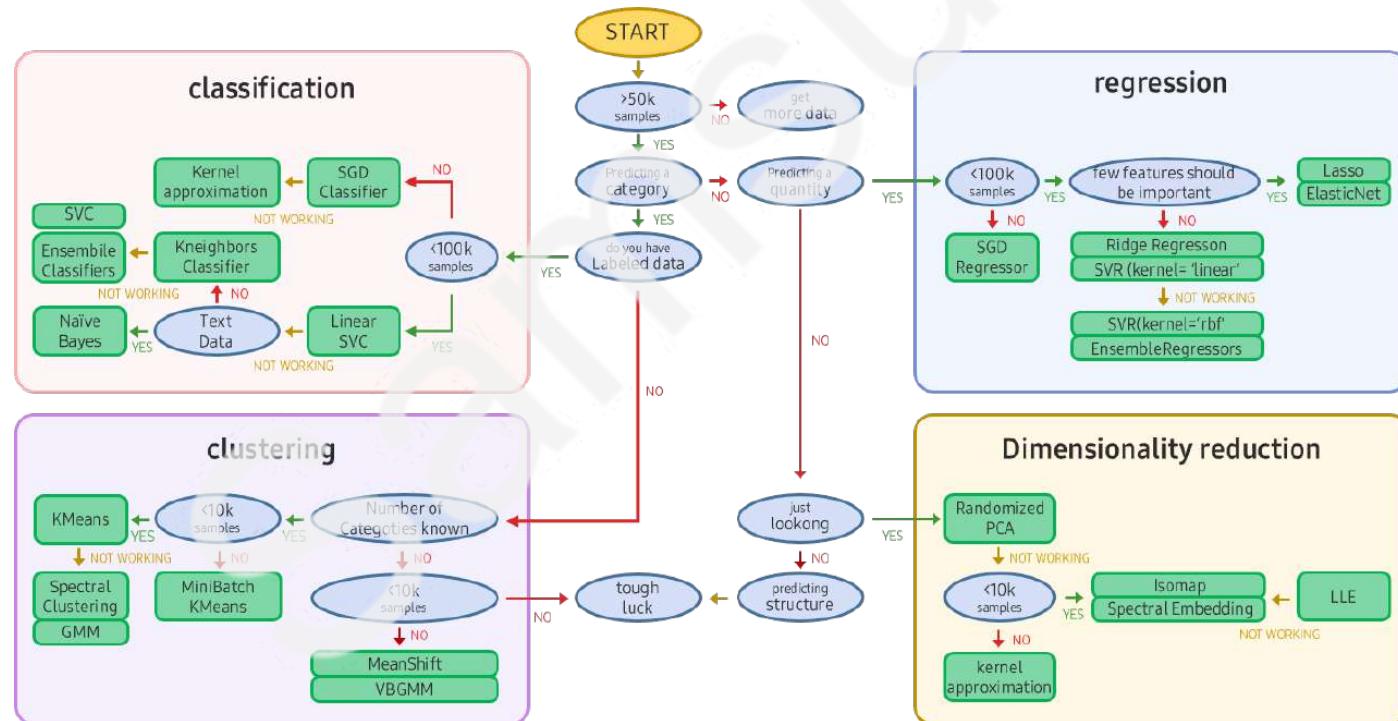
Machine Learning Based Data Analysis

- | 1.1. What is machine learning?
- | 1.2. Phyton scikit-learn library for machine learning
- | 1.3. Preparation and division of data set
- | 1.4. Data pre-processing for making a good training data set
- | **1.5. Practicing to find an optimal method to solve problems with scikit-learn**

Finding an optimal method to solve problems with scikit-learn

Practicing

- Use the following problem-solving methodology and consider which algorithm to apply. Perform pre-processing and overall process regarding the iris data and compare with the result code provided below.



1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

Practicing

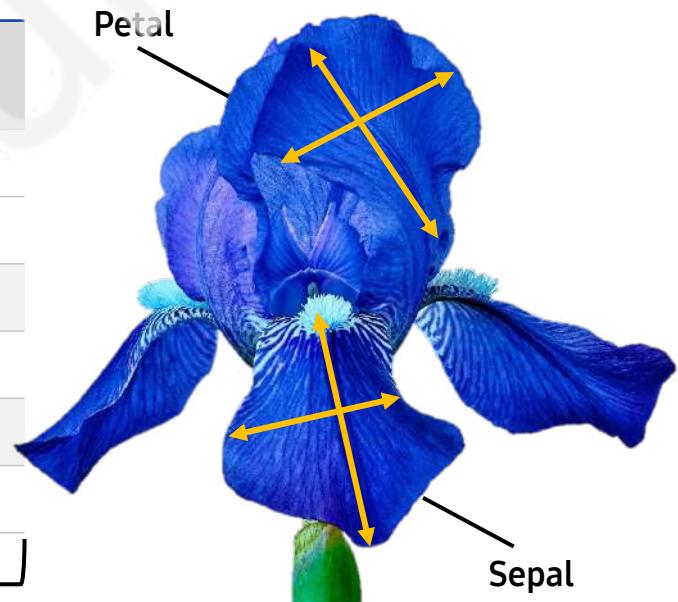
Sample

(Instance, observation)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Feature
(Property,
measurement value, dimension)

Class label
(Target)



| Considerations in machine learning

- 1) Define the problem of the business and check for solutions or best alternative plans.
 - 2) Check if it is possible to define it as a supervised or unsupervised problem.
 - 3) Check which method to use for measuring model performance.
 - 4) Check if the performance index is linked with the business objective and confirm if the project participants made an agreement regarding the performance.
- ▶ In this practicing problem, define the problem as iris species (supervised) and suppose that the result is satisfactory if the performance predicts 85% or higher classification accuracy.

| Understanding the iris data

```
In [1]: from sklearn.datasets import load_iris  
data=load_iris()  
data.keys()  
  
Out[1]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

| Domain knowledge of the iris data

- ▶ Data name: IRIS
- ▶ Number of data: 150
- ▶ Number of variables: 5
- ▶ Understanding variables

Sepal Length	Length information of the sepal
Sepal Width	Width information of the sepal
Petal Length	Length information of the petal
Petal Width	Width information of the petal
Species	Flower species, classified into setosa / versicolor / virginica

I Understanding the iris data

```
In [2]: print(data.DESCR)

.. _iris_dataset:

Iris plants dataset
-----
**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
    - Iris-Setosa
    - Iris-Versicolour
    - Iris-Virginica
```

| Understanding the iris data

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

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

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Iris data pre-processing and EDA

```
In [3]: 1 import pandas as pd  
2 import seaborn as sns  
3 import numpy as np  
4 import matplotlib.pyplot as plt
```

```
In [4]: 1 iris=pd.DataFrame(data=data.data, columns=data.feature_names)  
2 iris
```

Line 3-1 ~ 4

- Import the library required for practicing.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

Iris data pre-processing and EDA

```
In [3]: 1 import pandas as pd  
2 import seaborn as sns  
3 import numpy as np  
4 import matplotlib.pyplot as plt
```

```
In [4]: 1 iris=pd.DataFrame(data=data.data, columns=data.feature_names)  
2 iris
```

Line 4-1

- Convert the current variable data to ndarray and DataFrame of NumPy.

Iris data pre-processing and EDA

Out[4]:

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

150 rows × 4 columns

Iris data pre-processing and EDA

```
In [5]: feature=pd.DataFrame(data.data, columns=data.feature_names)
```

```
In [6]: data.target
```

```
In [7]: target=pd.DataFrame(data.target, columns=["species"])
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

Iris data pre-processing and EDA

In [8]: target

Out[8]:

species			
0	0	145	2
1	0	146	2
2	0	147	2
3	0	148	2
4	0	149	2
...	...	150 rows	x 1 columns

Iris data pre-processing and EDA

```
In [9]: 1 iris = pd.concat([feature, target], axis=1)
2
3 iris.rename({"sepal length (cm)": "sepal_length", "specal width (cm)": "specal_width",
4               "petal length (cm)": "petal_length", "petal width (cm)": "petal_width"}, 
5               axis=1, inplace=True)
6 iris.columns
```

```
Out[9]: Index(['sepal_length', 'sepal width (cm)', 'petal_length', 'petal_width',
   'species'],
   dtype='object')
```

Line 1

- Merge feature and target.

Iris data pre-processing and EDA

```
In [9]: 1 iris = pd.concat([feature, target], axis=1)
2
3 iris.rename({"sepal length (cm)": "sepal_length", "specal width (cm)": "specal_width",
4             "petal length (cm)": "petal_length", "petal width (cm)": "petal_width"}, 
5             axis=1, inplace=True)
6 iris.columns
```

```
Out[9]: Index(['sepal_length', 'sepal width (cm)', 'petal_length', 'petal_width',
   'species'],
   dtype='object')
```

Line 3 ~ 5

- Change the column name.

Iris data pre-processing and EDA

```
In [10]: iris["species"] = iris.species.map(lambda x: data.target_names[x])  
iris.head()
```

Out[10]:

	sepal_length	sepal width (cm)	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Line 10

- Change the target value.

Iris data pre-processing and EDA

```
In [11]: iris.isna().sum()
```

```
Out[11]: sepal_length      0  
         sepal width (cm)    0  
         petal_length        0  
         petal_width         0  
         species             0  
         dtype: int64
```

Line 11

- Check the missing value.

Iris data pre-processing and EDA

▶ Basic statistical analysis

- Perform basic statistical analysis to better understand the data by understanding the data size (numbers of data), the shape of data (matrix shape), data type, data distribution, and the relationship between features. It will also enhance the performance of machine learning.

```
In [12]: iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sepal_length     150 non-null    float64 
 1   sepal width (cm) 150 non-null    float64 
 2   petal_length     150 non-null    float64 
 3   petal_width      150 non-null    float64 
 4   species          150 non-null    object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

Iris data pre-processing and EDA

```
In [13]: iris.describe()
```

```
Out[13]:
```

	sepal_length	sepal width (cm)	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Line 13

- petal_length has the greatest standard deviation. Compared to other features, petal_width seems to have a narrower range of values. It would be better to perform regularization after checking the model performance due to the scale differences between features.

Iris data pre-processing and EDA

- ▶ Correlation analysis
 - Use corr to analyze the relationship among features.

```
In [14]: iris.corr()
```

```
Out[14]:
```

	sepal_length	sepal width (cm)	petal_length	petal_width
sepal_length	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal_length	0.871754	-0.428440	1.000000	0.962865
petal_width	0.817941	-0.366126	0.962865	1.000000

Line 14

- The correlation coefficient of petal_length and petal_width is 0.962865, which is extremely high. Since highly correlated features may induce multicollinearity problems, it is recommended to select one of the two variables to use.

Iris data pre-processing and EDA

▶ Aggregation analysis

```
In [15]: iris.groupby('species').size()
```

```
Out[15]: species
setosa      50
versicolor  50
virginica   50
dtype: int64
```

Line 15

- Number of data in each target was counted using the aggregation function ‘size,’ and it was confirmed that 50 data were equally found in each feature. Select between ‘size’ and ‘count’ depending on the purpose of the analysis. The ‘size’ counts the number of data, including missing values. On the other hand, the ‘count’ counts the number of data without missing values. In this case, there’s no difference using ‘size’ and ‘count’ because iris data does not have any missing values.

Iris data pre-processing and EDA

- ▶ Data visualization
 - Previously, basic statistics analysis was done on the data. Still, it is not easy to understand due to too many numbers, and incorrect reading of decimal points would result in significant errors. If so, visualization of data as a graph provides an intuitive understanding for the reader. Also, data visualization is efficient in explaining data analysis results.

Iris data pre-processing and EDA

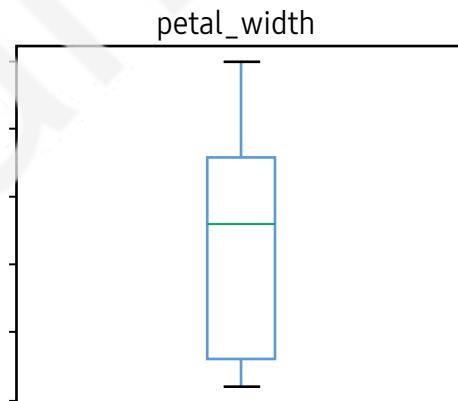
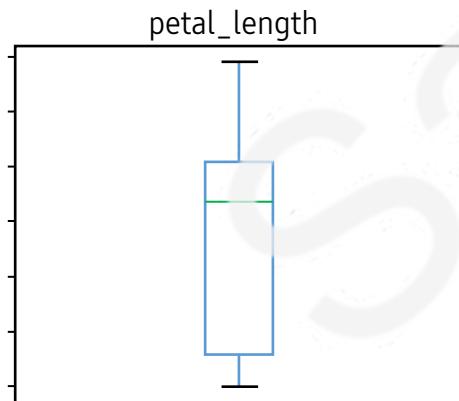
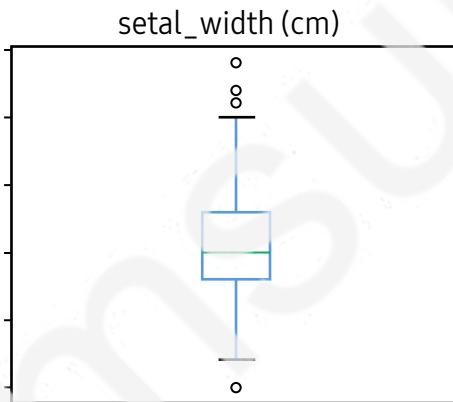
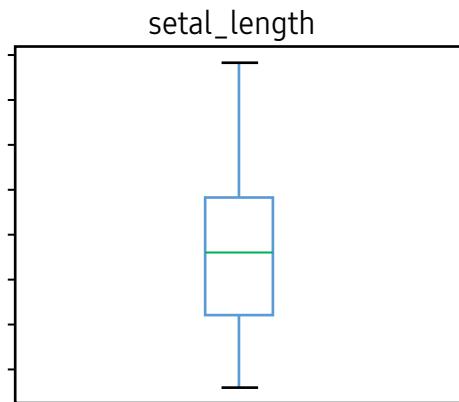
- ▶ Visualizing basic statistics and outlier

```
In [16]: def boxplot_iris(feature_names, dataset):  
    i=1  
    plt.figure(figsize=(11,9))  
    for col in feature_names:  
        plt.subplot(2,2,i)  
        plt.axis('on')  
        plt.tick_params(axis='both', left=True,  
                       top = False, right = False,  
                       bottom = True, labelleft = False,  
                       labeltop = False, labelright = False,  
                       labelbottom = False)  
        dataset[col].plot(kind='box', subplots = True, sharex = False)  
        plt.title(col)  
        i += 1  
    plt.show()
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

Iris data pre-processing and EDA

```
In [17]: boxplot_iris(iris.columns[:4],iris)
```



Iris data pre-processing and EDA

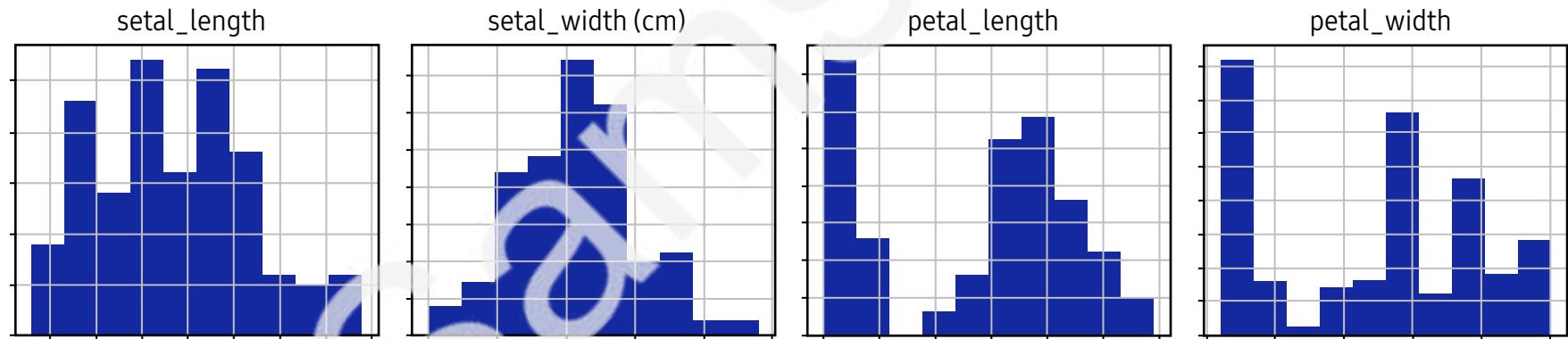
- ▶ Visualizing data distribution

```
In [18]: def boxplot_iris(feature_names, dataset):  
    i=1  
    plt.figure(figsize=(11,9))  
    for col in feature_names:  
        plt.subplot(2,2,i)  
        plt.axis('on')  
        plt.tick_params(axis='both', left=True,  
                       top = False, right = False,  
                       bottom = True, labelleft = False,  
                       labeltop = False, labelright = False,  
                       labelbottom = False)  
        dataset[col].hist()  
        plt.title(col)  
        i += 1  
    plt.show()
```

Iris data pre-processing and EDA

- The frequency of the median class interval is high for petal_width, and it becomes lower as it deviates far away from the center. In the box plot, the box length of sepal_width is short because a lot of data was aggregated at the median. In the case of petal_length, the frequency of the median class interval is high, but there are a lot of data on the left class interval. In the box plot, the box of petal_length is long to the bottom because there were a lot of data in lower values.

```
In [46]: histogram_iris(iris.columns[:-1], iris)
```



| Visualizing correlation

```
In [20]: corr=iris.corr()
cmap=sns.diverging_palette(220,10, as_cmap=True)
plt.figure(figsize=(11,9))
sns.heatmap(corr, cmap=cmap, vmax=1.0, vmin=-1.0, center=0, square=True,
            linewidths=.5, cbar_kws={"shrink":.5})
plt.show()
```



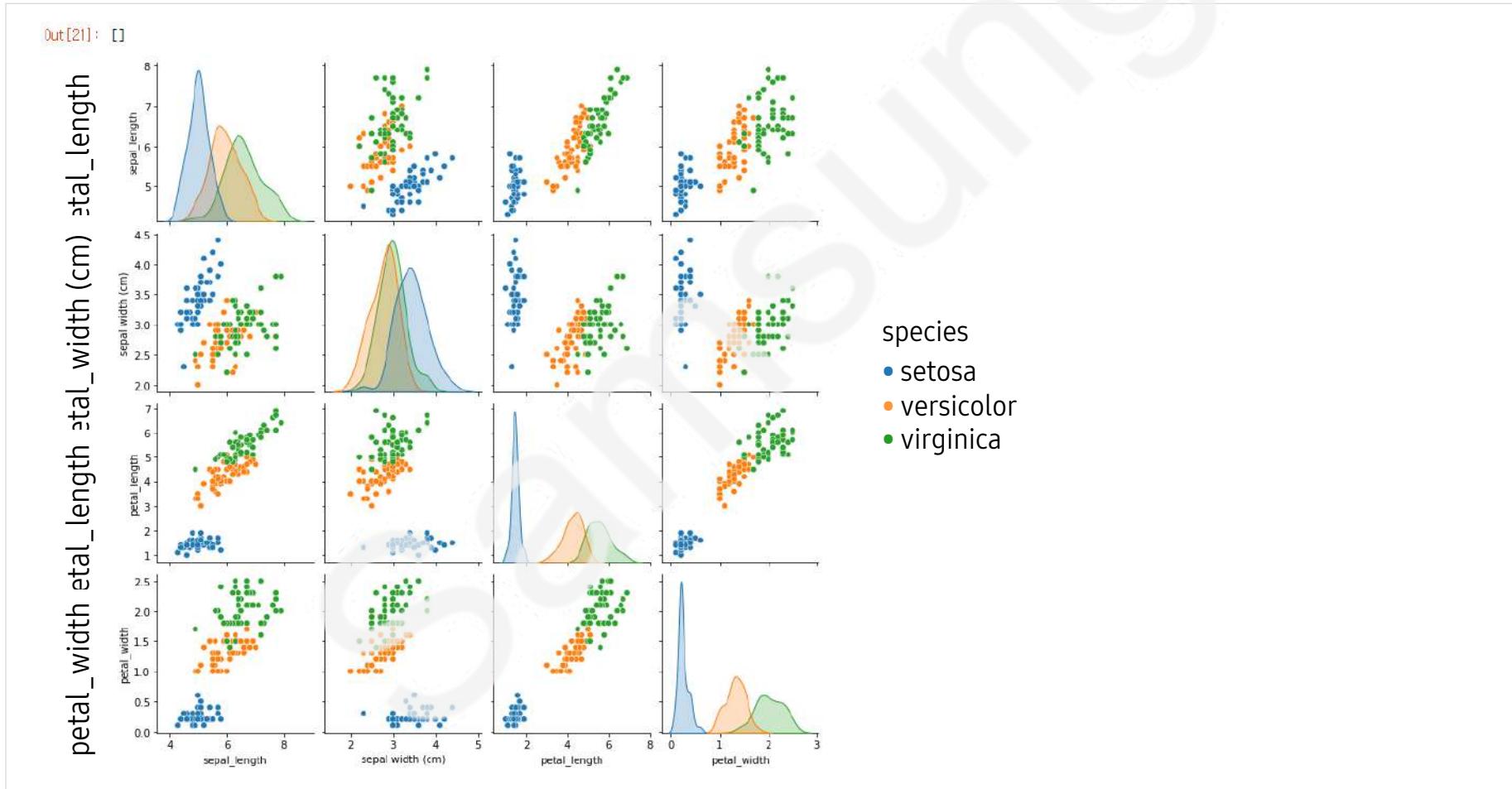
1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Visualizing the correlation between features and data distribution using pairplot

```
In [21]: sns.pairplot(iris, hue="species")
plt.plot()
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Visualizing the correlation between features and data distribution using pairplot



| Visualizing the correlation between features and data distribution using pairplot

- ▶ setosa is aggregated by clearly being deviated from other classes. Classification is possible by drawing an imaginary line, and setosa will be classified as a linear model. For versicolor and virginica, it seems difficult to classify them by drawing a line because they are mixed in the graph, complete with the sepal_width and sepal_length features. However, even if it seems a little vague, they can be classified in other graphs.

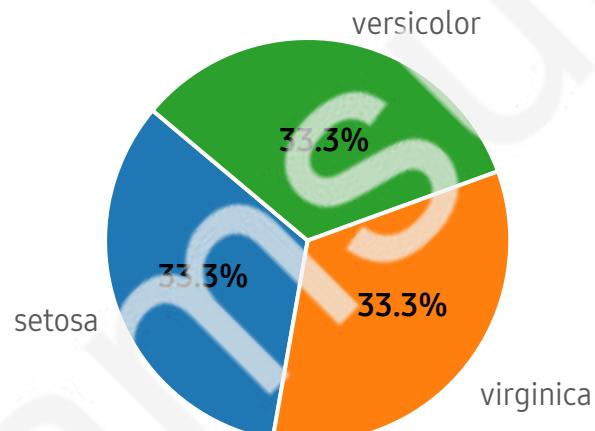
| Visualizing the class ratio of the target

```
In [22]: def piechar_iris(feature_names, target, dataset):
    i=1
    plt.figure(figsize=(11,9))
    for colName in [target]:
        labels=[];size=[];
        df=dataset.groupby(colName).size()
        for key in df.keys():
            labels.append(key)
            sizes.append(df[key])
        plt.subplot(2,2,i)
        plt.axis('on')
        plt.tick_params(axis='both',left=True,
                        top=False, right=False,
                        bottom=True, labelleft=True,
                        labeltop=True, labelright=False,
                        labelbottom=False)
        plt.pie(sizes, labels=labels, autopct='%1.1f%%',
                shadow=True,startangle=140)
        plt.axis('equal')
        i +=1
    plt.show()
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Visualizing the class ratio of the target

```
In [23]: piechar_iris(iris.columns[:-1], iris.species, iris)
```



Line 15

- The data is evenly arranged in each target class.

| Visualizing the class ratio of the target

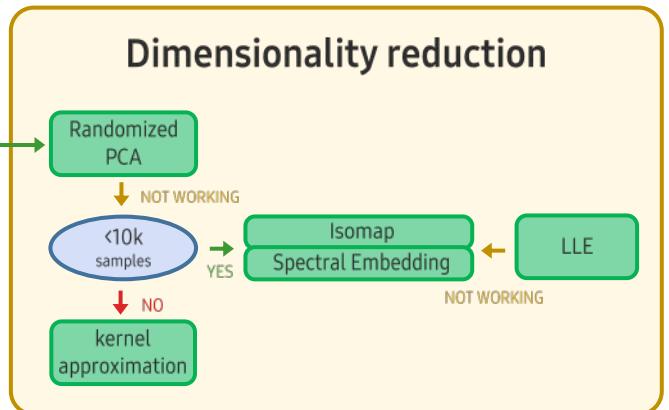
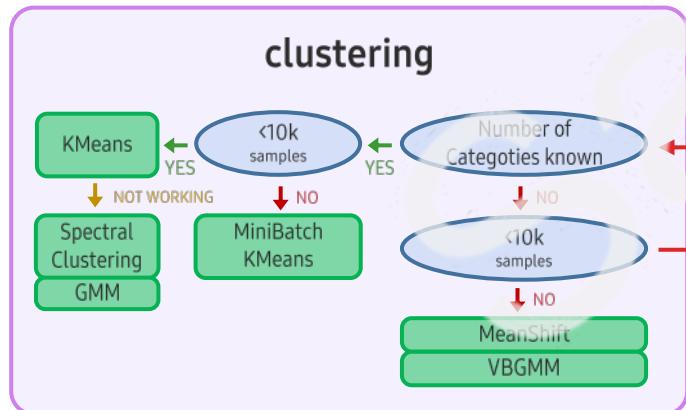
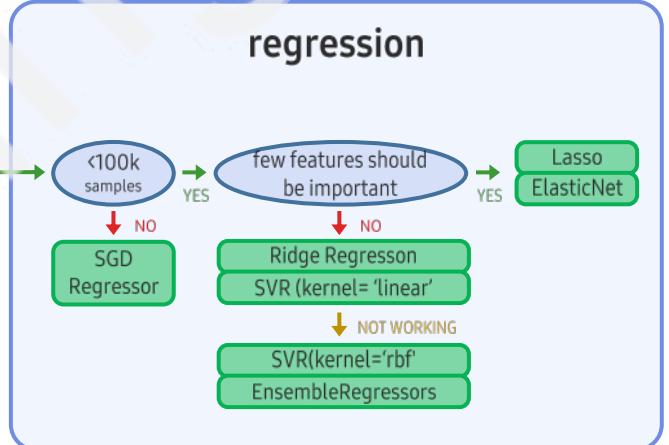
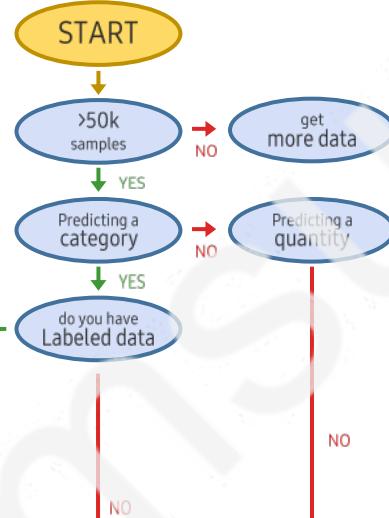
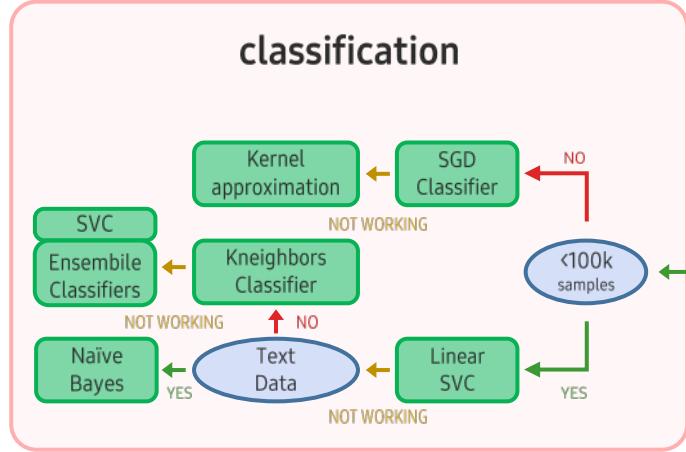
```
In [24]: from sklearn.model_selection import train_test_split
```

```
In [25]: X_train, X_test, y_train, y_test=train_test_split(iris.iloc[:, :-1], iris.iloc[:, -1],  
test_size=0.33, random_state=42)
```

- ▶ Before starting machine learning, split the data set into training and performance test data. The final objective of machine learning is to create a generalized model so that it can accurately predict new data. If evaluating the performance with data used in learning, the possibility of getting it right is high since the model is already familiar with the given data feature. For reliable evaluation, separate the performance test data set from the training data set. Because it is the separation of data, it is referred to as the hold out method.
- ▶ Split training and performance test data sets with the `train_test_split` function of `sklearn`. Classify the training data as 'train' and performance test data as 'test.' `X` is the feature of the data set, and `y` is the target. For structured data analysis, indicate `DataFrame` with capital letters and `Series` with lower cases. The `test_size=0.33` option separates 33% of the total data as a test set. `random_state=42` is an option used to induce reproducible results for the practicing problem. If not designating `random_state`, the data set for conversion will differ every time.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Algorithm selection



| Algorithm selection

- ▶ setosa is aggregated by clearly being deviated from other classes. Classification is possible by drawing an imaginary line, and setosa will be classified as a linear model. For versicolor and virginica, it seems difficult to classify them by drawing a line because they are mixed in the graph, complete with the sepal_width and sepal_length features. However, even if it seems a little vague, they can be classified in other graphs.

| Algorithm selection

```
In [26]: from sklearn.tree import DecisionTreeClassifier
```

```
In [27]: model = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
                                         min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
                                         random_state=42, max_leaf_nodes=None, min_impurity_decrease=0.0,
                                         min_impurity_split=None, class_weight=None)
```

- ▶ # regularization: Constraints the degree of freedom of the decision tree
- ▶ # Lowering max_depth would constrain the model and reduce the risk of overfitting.
- ▶ # min_samples_split: Minimum amount of sample required by the node for splitting
- ▶ # min_weight_fraction_leaf: Identical to the min_samples_leaf, but is the ratio with weight in the entire sample.
- ▶ # max_leaf_nodes: Maximum number of leaf nodes
- ▶ # max_features: The largest number of features that will be used for splitting by each node
- ▶ # Increasing the parameter that starts with min_ or lowering the parameter and that starts with max_ would increase the model constraint.

| Algorithm selection

- ▶ Gini impurity or entropy
- ▶ The difference between Gini impurity and entropy is vague in real life. Both of them create a similar tree.
- ▶ Calculation of Gini impurity is quicker, so it is recommended as a default.
- ▶ However, when creating a different tree, Gini impurity tends to isolate the most frequent class to one side, while entropy results in a more balanced tree.

| Model learning

- ▶ Perform model learning with the training data to check the model performance. The current model is set with default hyperparameter except for random_state.

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

```
Out[28]: DecisionTreeClassifier(random_state=42)
```

| Score

- ▶ Evaluate the performance using the performance test data set. In the Scikit-learn, score refers to accuracy. Since the iris data set is well-structured for practice, it generally shows high performance in any model.

```
In [29]: model.score(X_test,y_test)
```

```
Out[29]: 0.98
```

| Model generalization strategy

- ▶ As machine learning shows data-driven model performance, enough amount of data is required for good performance. An insufficient amount of data may result in overfitting, which means that the model shows lower prediction ability regarding unseen data because it is fit only to the training data features. The following is the generalization strategy so that the model would provide high performance regarding unseen data.

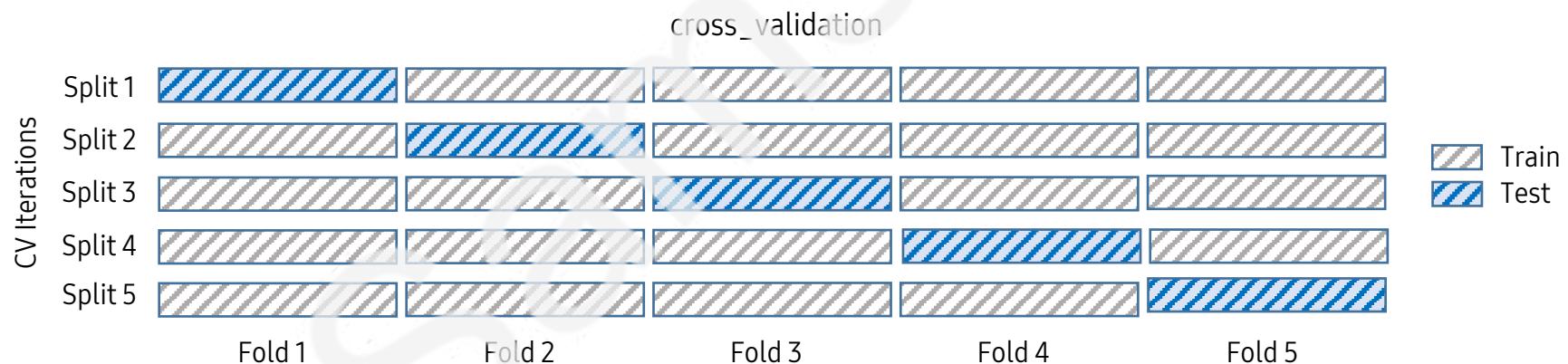
| Validation set

- ▶ The performance test data set to split with Train_test_split is for the final performance evaluation of the model. Since it is necessary to check model performance during model learning, hold out some of the data from the training data set and use it as the validation set. A chance of overfitting can be found during learning by using the validation set. It is also used for finding hyperparameters.



Cross validation

- ▶ This strategy is to make many validation sets so that every data can be included in learning once. Divide the data set into a random number $n=k$ (k-fold). Use the first fold as a validation set and other $k-1$ folds as a training set and measure the performance.
- ▶ Use the second fold as a test set and other folds as a training set for learning, and then measure the performance. Repeat the same process for all the other folds so that all data can be included in the training. Obtain k performance evaluation results and then average out to predict the model performance. The following figure is an example of when $k=5$.



Cross_val_score

- ▶ Cross validation can be easily performed using the cross_val_score function of scikit-learn.

```
In [30]: import numpy as np
from sklearn.model_selection import cross_val_score,KFold
cv=KFold(n_splits=10,shuffle=True,random_state=42)
results=cross_val_score(model, X_train, y_train, cv=cv)
fin_result=np.mean(results)
```

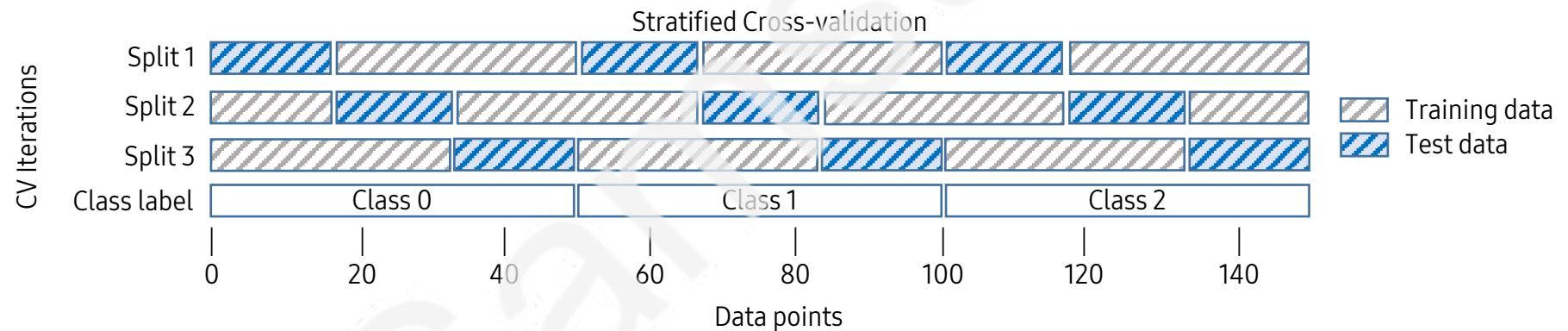
```
In [31]: for i, _ in enumerate(results):
    print("{}th cross validation score : {}".format(i,_))

print("Final cross validation final score: {}".format(fin_result))
```

```
0th cross validation score : 0.9
1st cross validation score : 1.0
2nd cross validation score : 0.8
3rd cross validation score : 1.0
4th cross validation score : 0.8
5th cross validation score : 0.9
6th cross validation score : 1.0
7th cross validation score : 0.9
8th cross validation score : 1.0
9th cross validation score : 1.0
Final cross validation score : 0.93
```

| stratified

- ▶ The random splitting of the train set and validation set would result in an inconstant target class when hold-out. If so, the data distribution differs in training and validation sets, affecting learning. For machine learning, a premise is present in which training data distribution and real-life data distribution are the same. If the premise is not followed, the learning model performance falls. So, to prevent such an issue, the stratified method is used to evenly distribute the target class ratio. The following figure provides an intuitive understanding of how the stratified method classifies data.



- ▶ Cross validation is possible by sending the instance of StratifiedKFold to the cv option of cross_val_score.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| stratified

```
In [32]: from sklearn.model_selection import StratifiedKFold  
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)  
results=cross_val_score(model, X_train, y_train, cv=cv)  
fin_result = np.mean(results)  
for i, _ in enumerate(results):  
    print("{}th stratified cross validation score: {}".format(i,_))
```

0th stratified cross validation score: 0.9
1st stratified cross validation score: 0.9
2nd stratified cross validation score: 0.8
3rd stratified cross validation score: 0.9
4th stratified cross validation score: 1.0
5th stratified cross validation score: 1.0
6th stratified cross validation score: 0.9
7th stratified cross validation score: 0.8
8th stratified cross validation score: 1.0
9th stratified cross validation score: 1.0

```
In [33]: print("Final stratified cross validation score: {}".format(fin_result))
```

Final stratified cross validation score: 0.9199999999999999

Learning Curve

- ▶ !pip install scikit-plot

```
In [34]: !pip install scikit-plot
Requirement already satisfied: scikit-plot in c:\users\emcast\anaconda3\lib\site-packages (0.3.7)
Requirement already satisfied: scipy>=0.9 in c:\users\emcast\anaconda3\lib\site-packages (from scikit-plot) (1.6.2)
Requirement already satisfied: matplotlib>=1.4.0 in c:\users\emcast\anaconda3\lib\site-packages (from scikit-plot) (3.3.4)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\emcast\anaconda3\lib\site-packages (from scikit-plot) (0.24.1)
Requirement already satisfied: joblib>=0.10 in c:\users\emcast\anaconda3\lib\site-packages (from scikit-plot) (1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\emcast\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\emcast\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\emcast\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (8.2.0)
Requirement already satisfied: numpy>=1.15 in c:\users\emcast\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (1.20.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\emcast\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (2.8.1)
Requirement already satisfied: cycler>=0.10 in c:\users\emcast\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (0.10.0)
Requirement already satisfied: six in c:\users\emcast\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib>=1.4.0->scikit-plot) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\emcast\anaconda3\lib\site-packages (from scikit-learn>=0.18->scikit-plot) (2.1.0)
```

| Learning Curve

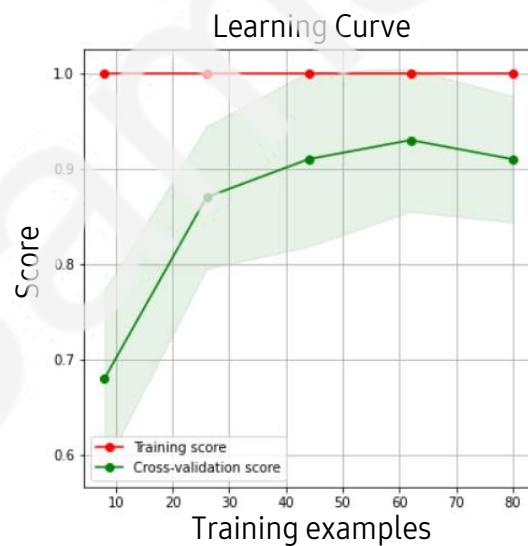
- ▶ !pip install scikit-plot
 - The green line is the result of cross-validation. Overfitting occurs when the green line goes upward to the right but then starts to fall. The red line is data used for training which is validated. The red line may momentarily fall if there are too much data. This phenomenon is only temporary, and the graph will converge in the long term.
 - The cv option is not designated, so 3fold is applied as default. There are 100 data in the training set, and 33% of it is used for cross-validation, so the maximum value of the x-axis is 66. The training curve is disconnected as the green line increases, so it is unknown if there is more data. At this point, it is impossible to know whether there is enough data. The training curve is drawn differently depending on the algorithm, even when using the same data. Thus, what is known from this training curve is that the performance of the current decision tree model would be better if there were more data.

| Learning Curve

- ▶ !pip install scikit-plot

```
In [35]: import scikitplot as skplt
```

```
In [36]: import matplotlib.pyplot as plt
skplt.estimators.plot_learning_curve(model,
                                      X_train, y_train,
                                      figsize=(6,6))
plt.show()
```



| Learning Curve

- ▶ If there is enough data, identical data distribution is maintained even when training and validation sets are randomly split. The cross-validation method is required when there is insufficient data. Draw a learning curve to determine whether there is enough data. The learning curve shows how performance changes when slightly increasing the amount of training data by setting the x-axis as the number of training data and the y-axis as the performance score. The test score is calculated by internal cross-validation.
- ▶ The learning curve can be drawn using the scikitplot library, which supports the scikit-learn. Install the library separately from scikit-learn to use.
- ▶ The scikitplot is not provided in anaconda as default, so it needs to be installed using the package management tools. Run the following code with the Jupyter Notebook to install the library. Be aware of different names when installing and importing the library.

| Model optimization strategy

```
In [37]: estimator=DecisionTreeClassifier()
from sklearn.model_selection import GridSearchCV
parameters={'max_depth' : [4,6,8,10,12],
            'criterion' : ['gini', 'entropy'],
            'splitter' : ['best', 'random'],
            'min_weight_fraction_leaf' : [0.0,0.1,0.2,0.3],
            'random_state' : [7,23,42,78,142],
            'min_impurity_decrease' : [0.0,0.05,0.1,0.2]}
model2=GridSearchCV(estimator=estimator,
                     param_grid = parameters,
                     cv=KFold(10), verbose=1,
                     n_jobs = -1, refit = True)
model2.fit(X_train, y_train)
```

Fitting 10 folds for each of 1600 candidates, totalling 16000 fits

```
Out[37]: GridSearchCV(cv=KFold(n_splits=10, random_state=None, shuffle=False),
                      estimator=DecisionTreeClassifier(), n_jobs=-1,
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [4, 6, 8, 10, 12],
                                  'min_impurity_decrease': [0.0, 0.05, 0.1, 0.2],
                                  'min_weight_fraction_leaf': [0.0, 0.1, 0.2, 0.3],
                                  'random_state': [7, 23, 42, 78, 142],
                                  'splitter': ['best', 'random']},
                      verbose=1)
```

| Model optimization strategy

▶ #Hyperparameter

- In machine learning, the machine learns the data and finds parameters by itself. Hyperparameters refer to parameters that need to be directly designated by a human as they cannot be found by the machine.
In the scikit-learn, it is possible to set hyperparameters to instance an algorithm.

▶ #Hyperparameter search using GridSearchCV

- In general, hyperparameters are found by the analyst's expertise.
The scikit-learn provides the GridSearchCV function that finds hyperparameters. It is a function that lists all cases regarding hyperparameter combinations on a grid and learns and performs performance measures for every combination. It may seem like working without any plans. However, the machine automatically performs the work as the range of hyperparameters is designated by the analyst. Although it takes some time, it eases finding hyperparameters.
- ▶ Similar to algorithms, instance GridSearchCV as well. When instancing, send the instanced algorithm model as an argument to the estimator option. For param_grid, send the dictionary containing hyperparameters for testing as an argument.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

The total number of cases that can possibly be made with the parameters from the practice problem is 1600. Since k=10 in the K-fold cross-validation, 10 cross-validations were performed for each case; a total of 16,000 training were done. The following table shows hyperparameter combinations in the practice problem.

- ▶ The optimal parameters and optimized performance found with GridSearchCV are recorded in best_params_ and best_score_ attributes.
- ▶ If the refit option is set to True, train the model with the optimal hyperparameters and record to the best_estimator_ attribute.

	0	1	2	3	4	...	1595	1596	1597	1598	1599
criterion	gini	gini	gini	gini	gini	...	entropy	entropy	entropy	entropy	entropy
max_depth	4	4	4	4	4	...	12	12	12	12	12
min_impurity_decrease	0	0	0	0	0	...	0.2	0.2	0.2	0.2	0.2
random_might_fraction_leaf	0	0	0	0	0	...	0.3	0.3	0.3	0.3	0.3
random_state	7	7	23	23	42	...	42	78	78	142	142
splitter	best	random	best	random	best	...	random	best	random	best	random

6 rows * 1600 columns

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

```
In [38]: model2.best_estimator_
Out[38]: DecisionTreeClassifier(max_depth=4, random_state=23, splitter='random')

In [39]: model2.best_params_
Out[39]: {'criterion': 'gini',
          'max_depth': 4,
          'min_impurity_decrease': 0.0,
          'min_weight_fraction_leaf': 0.0,
          'random_state': 23,
          'splitter': 'random'}

In [40]: model2.best_score_
Out[40]: 0.9700000000000001
```

Evaluation criteria and model evaluation

- ▶ Use the X_test and y_test from hold out for the final model evaluation. For an accurate evaluation, it is important to be aware of different kinds of evaluation criteria.

```
In [40]: model2.best_score_
```

```
Out[40]: 0.9700000000000001
```

```
In [41]: from sklearn.metrics import accuracy_score
pred=model2.predict(X_test)
pred
```

```
Out[41]: array(['versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'virginica', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'virginica', 'setosa',
       'virginica', 'setosa', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'setosa', 'setosa', 'setosa', 'setosa',
       'versicolor', 'setosa', 'setosa', 'virginica', 'versicolor',
       'setosa', 'setosa', 'setosa', 'virginica', 'versicolor',
       'versicolor', 'setosa', 'setosa', 'versicolor', 'versicolor',
       'virginica', 'versicolor', 'virginica'], dtype=object)
```

```
In [42]: accuracy_score(y_test,pred)
```

```
Out[42]: 0.98
```

| Evaluation criteria and model evaluation

- ▶ #Limitations of accuracy
 - So far, accuracy was the only criterion to validate the model, but a limitation is present. It requires more than accuracy to evaluate the model properly.
- Ex** If there's a model that predicts 'setosa' only even when entering various kinds of data, the model performance would be doubtful.
- However, let's assume that there are 48 setosas, 1 versicolor, and 1 virginica in the test set. When making an evaluation using this test set, a problem is that it would have 96% accuracy. Nevertheless, it's not because the model's performance is great. It would be necessary to check other evaluation criteria as well to accurately evaluate the model performance.

Confusion Matrix

- The following confusion matrix can be expressed with binary classification.

	Predicted positive class	Predicted negative class
Actual Positive	TP (True Positive)	FN (False Negative)
Actual Negative	FP (False Positive)	TN (True Negative)

- Evaluation scores, including precision, recall, f1-score, and others, can be made based on the abovementioned concepts (TP, FP, TN, FN).
- Use the confusion matrix to analyze both right and wrong predicted results. The confusion matrix can validate the performance differently to see how well the predicted and actual targets got right.

| Multi label classification

	Predicted setosa	Predicted versicolor	Predicted virginica
Actual setosa	Actual setosa and predicted setosa	Actual setosa but predicted versicolor	Actual setosa but predicted virginica
Actual versicolor	Actual versicolor but predicted setosa	Actual versicolor and predicted versicolor	Actual versicolor but predicted virginica
Actual virginica	Actual virginica but predicted setosa	Actual virginica but predicted versicolor	Actual virginica and predicted virginica

- ▶ Since the iris data is a multi-label classification problem, it cannot be expressed in four different concepts only as provided earlier. So, create three indices for each setosa, versicolor, virginica by considering each as a binary classification problem. Take setosa, for example.

| Confusion matrix of the iris data set

```
In [43]: from sklearn.metrics import confusion_matrix  
pred=model2.predict(X_test)  
confusion_matrix(y_test,pred)
```

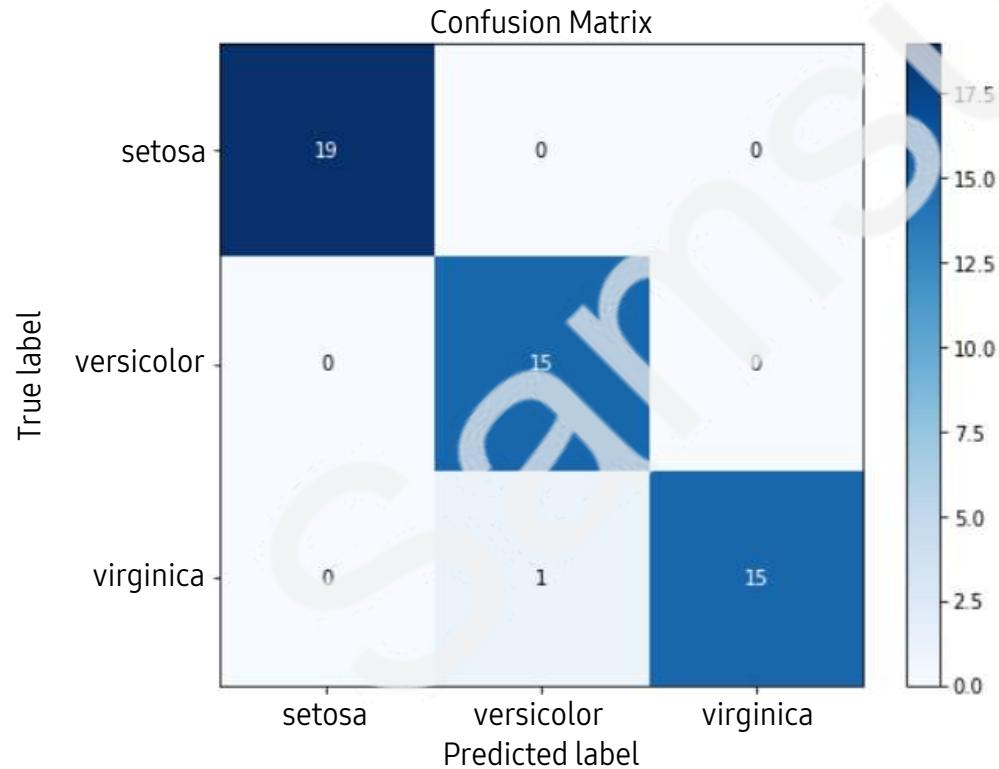
```
Out[43]: array([[19,  0,  0],  
                 [ 0, 15,  0],  
                 [ 0,  1, 15]], dtype=int64)
```

- ▶ With scikit-learn, it is possible to easily calculate the confusion matrix using `confusion_matrix` function. Send the arguments to the actual class and then to the predicted class.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Confusion matrix of the iris data set

```
In [44]: import scikitplot as skplt  
skplt.metrics.plot_confusion_matrix(y_test,pred,figsize=(8,6))  
plt.show()
```



| Confusion matrix of the iris data set

- ▶ Use the scikit-plot to visualize the confusion matrix into a more intuitive heatmap. The scikit-learn did not have labels for the x-axis and y-axis. On the other hand, scikit-plot has labels for the axis for easier result interpretation.

| precision / recall / fall-out / f-score

- Evaluation scores differ in each target class in a multi-label classification problem.

	Predicted setosa	Not predicted setosa (predicted versicolor or virginica)
Actual setosa	TP (True Positive)	FN (False Negative)
Not actual versicolor (versicolor or virginica)	FP (False Positive)	TN (True Negative)

- Score the evaluation results based on the confusion matrix's TP, TN, FP, and FN. These four concepts are only possible in binary classification problems. For multi-label classification problems having N target classes, such as iris data, consider each target class as binary classification and obtain N confusion matrixes.

Ex Iris data

Consider each setosa, versicolor, and virginica as a binary classification problem and create three confusion matrixes.

The following shows the confusion matrix of setosa.

I precision

- Precision is the ratio of the correct predicted class.

$$precision = \frac{TP}{TP + FP}$$

```
In [45]: from sklearn.metrics import precision_score
precisions=precision_score(y_test, pred, average=None)

for target, score in zip(data.target_names, precisions):
    print(f'{target} precision:{score}'")
```

setosa precision: 1.0
versicolor precision: 0.9375
virginica precision: 1.0

Line 45

- In multi-label classification, average cannot be “binary.”
- “binary” is the average default.

| recall

- ▶ Also called sensitivity, recall is the correct prediction ratio among the actual target class.

$$recall = \frac{TP}{TP + FN}$$

```
In [46]: from sklearn.metrics import recall_score
recalls=recall_score(y_test,model2.predict(X_test),average=None)
for target, score in zip(data.target_names, recalls):
    print(f'{target} sensitivity:{score}')
```

setosa sensitivity: 1.0
versicolor sensitivity: 1.0
virginica sensitivity : 0.9375

| fall-out

- ▶ Fall-out is the incorrect ratio among the actual class, not target. Also expressed as 1-specificity.

$$fall-out = \frac{FP}{FP + TN}$$

- ▶ The scikit-learn does not provide how to calculate fall-out.

| f-score

- Precision and recall have a trade-off relationship. The f-score is the weighted harmonic mean of precision and recall. If the f-score is less than 1, more weight is provided to precision; if it is greater than 1, more weight is provided to recall. The f-score is used to accurately understand the model performance when the data class is imbalanced.

$$F_{\beta} = (1 + \beta^2) \frac{(precision \times recall)}{\beta^2 precision + recall}$$

- For even weight of precision and recall, β is set 1 most of the time, which is specifically referred to as f1-score.

$$F_1 = 2 \cdot \frac{precision \times recall}{precision + recall}$$

| f-score

- ▶ #F1 measure – both precision and recall are equally weighted. F1 score is used to calculate the average from the harmonic mean of precision and recall (sensitivity) and weighting precision and recall.
- ▶ # a (precision), b (recall) $2ab/a+b$
- ▶ #F0.5 measure – Precision is more weighted than recall. 0.5 times greater weight is applied to the recall compared to precision.
- ▶ #F2 measure – Recall is more weighted. Recall is 2 times more weighted than precision.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| f-score

```
In [47]: from sklearn.metrics import fbeta_score, f1_score  
  
fbetas=fbeta_score(y_test,pred,beta=1,average=None)  
  
for target, score in zip(data.target_names, fbeta):  
    print(f'{target}fbeta score:{score}')  
  
f1s=f1_score(y_test,pred,average=None)  
  
for target, score in zip(data.target_names, f1s):  
    print(f'{target}f1 score:{score}')
```

```
setosa fbeta score : 1.0  
versicolor fbeta score : 0.967741935483871  
virginica fbeta score : 0.967741935483871  
setosa f1 score : 1.0  
versicolor f1 score : 0.967741935483871  
virginica f1 score : 0.967741935483871
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| accuracy

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

| classification_report

- ▶ Use the classification_report function of scikit-learn to batch calculate precision, recall, and f1-score.

```
In [48]: from sklearn.metrics import classification_report  
print(classification_report(y_test,pred))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	19
versicolor	0.94	1.00	0.97	15
virginica	1.00	0.94	0.97	16
accuracy			0.98	50
macro avg	0.98	0.98	0.98	50
weighted avg	0.98	0.98	0.98	50

| ROC curve

- ▶ ROC curve has TPR (True Positive Rate) on the y-axis and FPR(False Positive Rate) on the x-axis.
TPR is recall, and FPR refers to fall-out.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

ROC curve

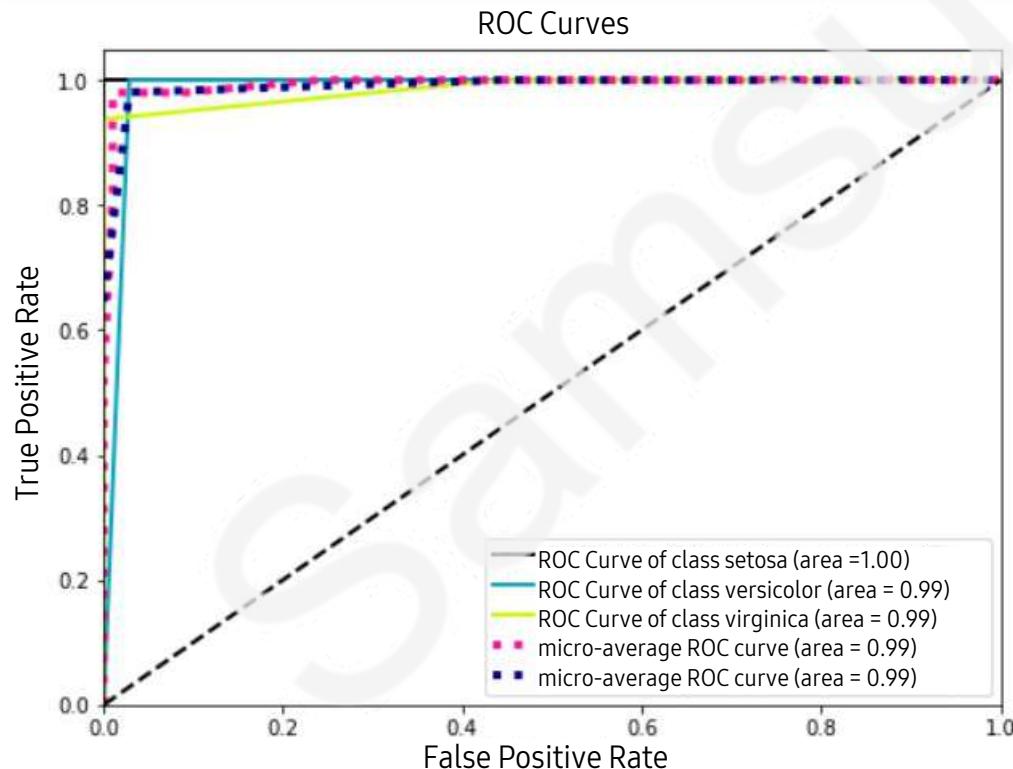
```
In [49]: pred_proba=model2.predict_proba(X_test)
```

```
In [50]: pred_proba
```

```
Out[50]: array([[0.          , 0.93939394, 0.06060606],
       [1.          , 0.          , 0.          ],
       [0.          , 0.          , 1.          ],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.93939394, 0.06060606],
       [1.          , 0.          , 0.          ],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.          , 1.          ],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.0625     , 0.9375     ],
       [1.          , 0.          , 0.          ],
       [1.          , 0.          , 0.          ],
       [1.          , 0.          , 0.          ],
       [1.          , 0.          , 0.          ],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.0625     , 0.9375     ],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.93939394, 0.06060606],
       [0.          , 0.0625     , 0.9375     ],
       [1.          , 0.          , 0.          ],
       [0.          , 0.0625     , 0.9375     ],
       [1.          , 0.          , 0.          ],
       [0.          , 0.0625     , 0.9375     ],
       [0.          , 0.          , 1.          ],
       [0.          , 0.          , 1.          ],
       [0.          , 0.          , 1.        ]],
```

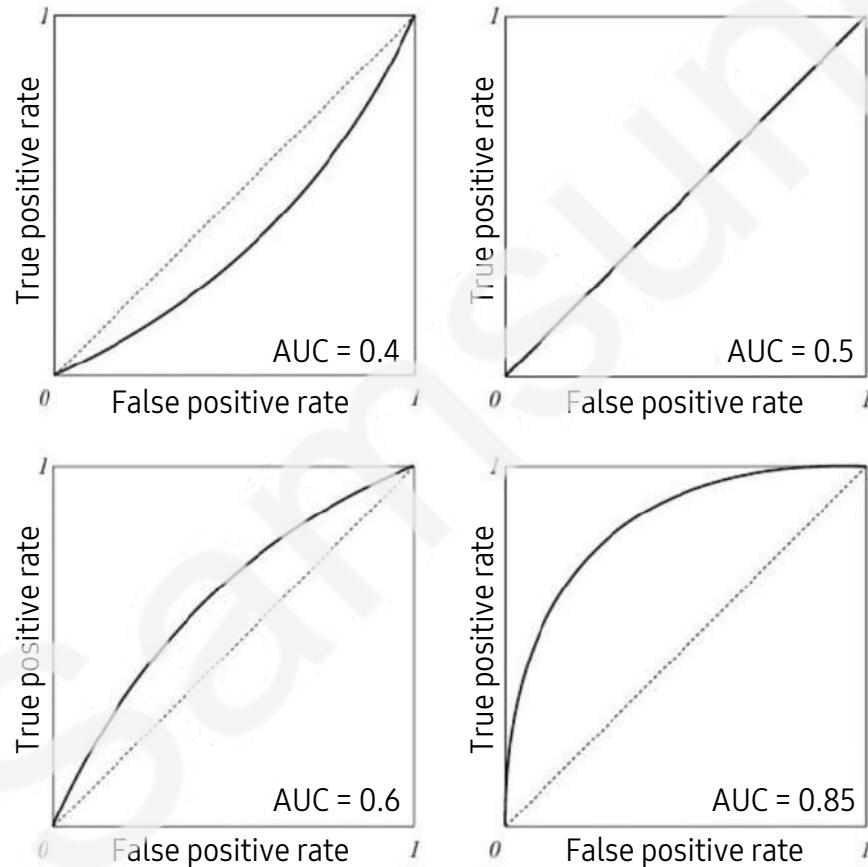
ROC curve

```
In [51]: import scikitplot as skplt  
skplt.metrics.plot_roc(y_test, pred_proba, figsize=(8,6))  
plt.show()
```



1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| AUC (Area Under Curve)



The Hundred-Page Machine Learning Book

Poor performance: Return to the previous step

- ▶ Final model
- ▶ Save model – It takes days to learn if there is a lot of data. It is extremely inefficient to train the model every time for prediction, so the best way is to save the model for reuse. Use ‘pickle’ to save the model.

```
In [52]: model.fit(iris.iloc[:, :-1], iris.iloc[:, -1])
```

```
Out[52]: DecisionTreeClassifier(random_state=42)
```

```
In [53]: import pickle
```

```
In [54]: with open('final_model.pickle', 'wb') as fp:  
    pickle.dump(model2, fp)
```

```
In [55]: f=open('final_model.pickle', 'rb')  
model3=pickle.load(f)
```

```
In [56]: predicted_species=model3.predict(iris.iloc[:, :-1])
```

```
In [57]: iris['predicted_species']=predicted_species  
iris.to_csv('FinalResult.csv', index=False)
```

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Poor performance: Return to the previous step

- ▶ Final model

```
In [52]: model.fit(iris.iloc[:, :-1], iris.iloc[:, -1])
```

```
Out[52]: DecisionTreeClassifier(random_state=42)
```

```
In [53]: import pickle
```

```
In [54]: with open('final_model.pickle', 'wb') as fp:  
    pickle.dump(model2, fp)
```

```
In [55]: f=open('final_model.pickle', 'rb')  
model3=pickle.load(f)
```

```
In [56]: predicted_species=model3.predict(iris.iloc[:, :-1])
```

```
In [57]: iris['predicted_species']=predicted_species  
iris.to_csv('FinalResult.csv', index=False)
```

Line 55

- Import model.

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Poor performance: Return to the previous step

- ▶ Final model

```
In [52]: model.fit(iris.iloc[:, :-1], iris.iloc[:, -1])
```

```
Out[52]: DecisionTreeClassifier(random_state=42)
```

```
In [53]: import pickle
```

```
In [54]: with open('final_model.pickle', 'wb') as fp:  
    pickle.dump(model2, fp)
```

```
In [55]: f=open('final_model.pickle', 'rb')  
model3=pickle.load(f)
```

```
In [56]: predicted_species=model3.predict(iris.iloc[:, :-1])
```

```
In [57]: iris['predicted_species']=predicted_species  
iris.to_csv('FinalResult.csv', index=False)
```

Line 56

- Final prediction

1.5. Practicing to find an optimal method to solve problems with scikit-learn UNIT 01

| Poor performance: Return to the previous step

- ▶ Final model

```
In [52]: model.fit(iris.iloc[:, :-1], iris.iloc[:, -1])
```

```
Out[52]: DecisionTreeClassifier(random_state=42)
```

```
In [53]: import pickle
```

```
In [54]: with open('final_model.pickle', 'wb') as fp:  
    pickle.dump(model2, fp)
```

```
In [55]: f=open('final_model.pickle', 'rb')  
model3=pickle.load(f)
```

```
In [56]: predicted_species=model3.predict(iris.iloc[:, :-1])
```

```
In [57]: iris['predicted_species']=predicted_species  
iris.to_csv('FinalResult.csv', index=False)
```

Line 57

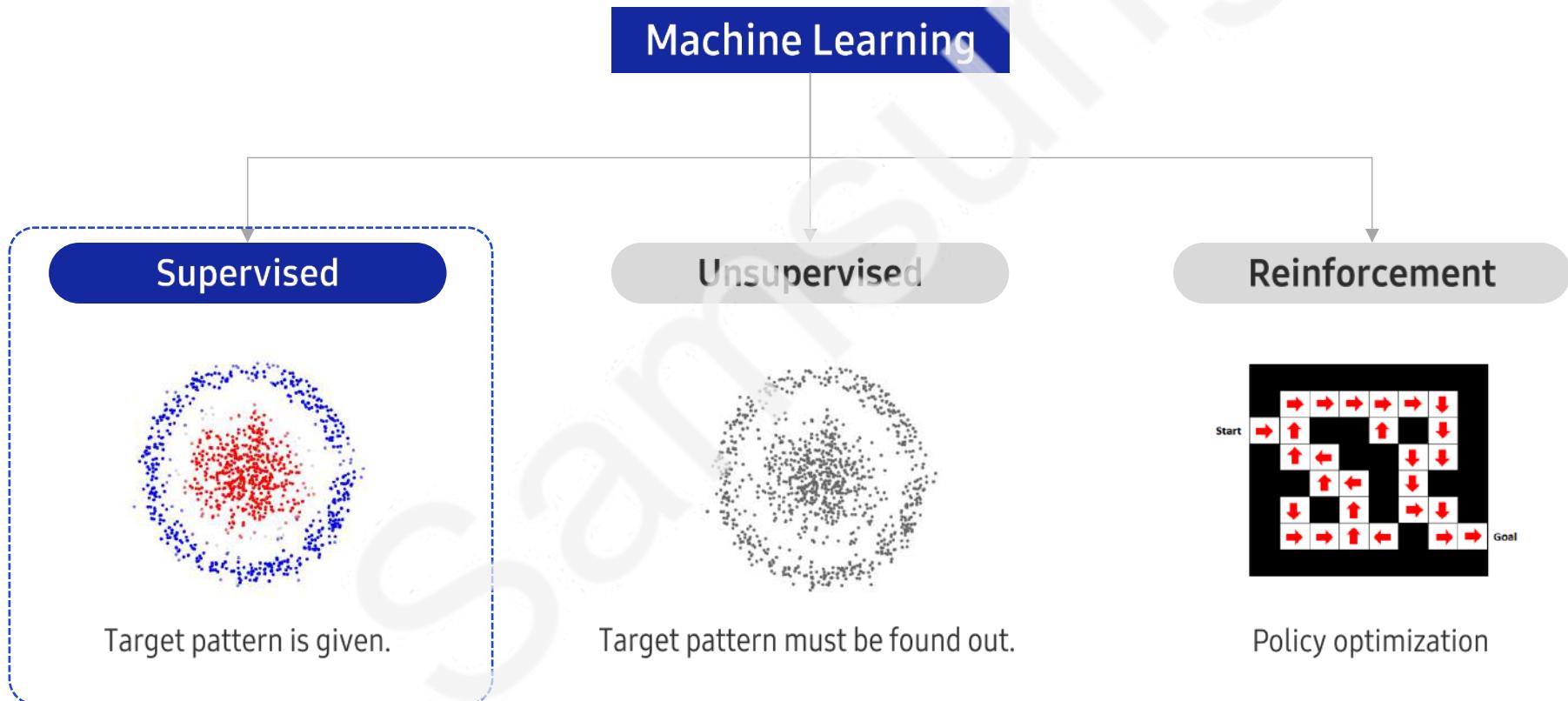
- Save csv.

Unit 2.

Application of the Supervised Learning Model for Numerical Prediction

- | **2.1. Training and Testing in Machine Learning**
- | 2.2. Linear Regression Basics
- | 2.3. Linear Regression Diagnostics
- | 2.4. Other Regression Types
- | 2.5. Practicing the Supervised Learning Model for Numerical Prediction

Machine Learning Types



Training and Testing in Machine Learning

| What is learning (human)?

- ▶ True learning is not just having a good memory.
- ▶ Learned material must be put to test.

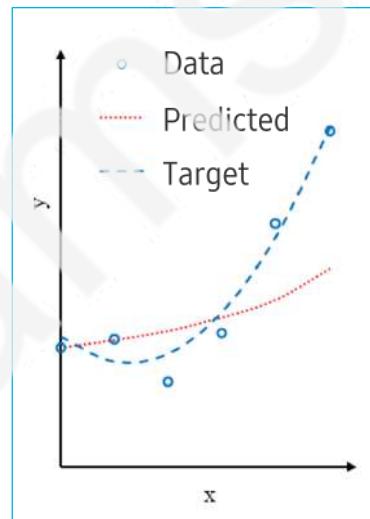
| Training and testing in machine learning

- ▶ A trained model must be put to test to generalize.
- ▶ Testing result is measured by errors.

Error Types

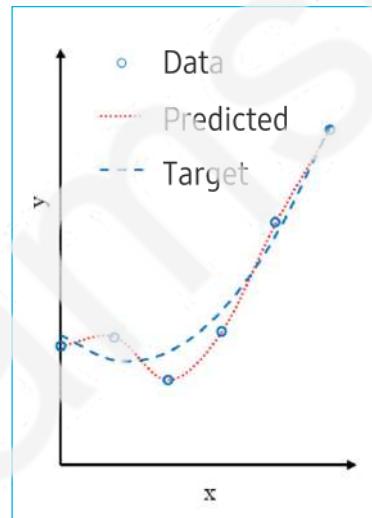
| Bias error (Underfitting error)

- ▶ Associated with simple/rigid/biased models.
- ▶ Prediction cannot account for the detailed data pattern.
- ▶ To lower this error type, increase the model complexity.



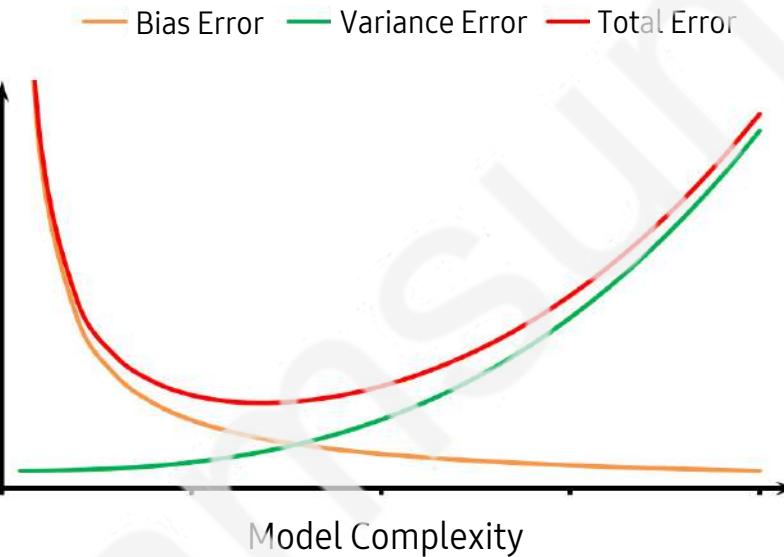
| Variance error (Overfitting error)

- ▶ Associated with models overly complex and sensitive to noise.
- ▶ Prediction performance is good while training but worsens when testing with a different dataset.
- ▶ To lower this error type, increase the amount of training data (*) or decrease the model complexity.



(*) By data augmentation method, for example.

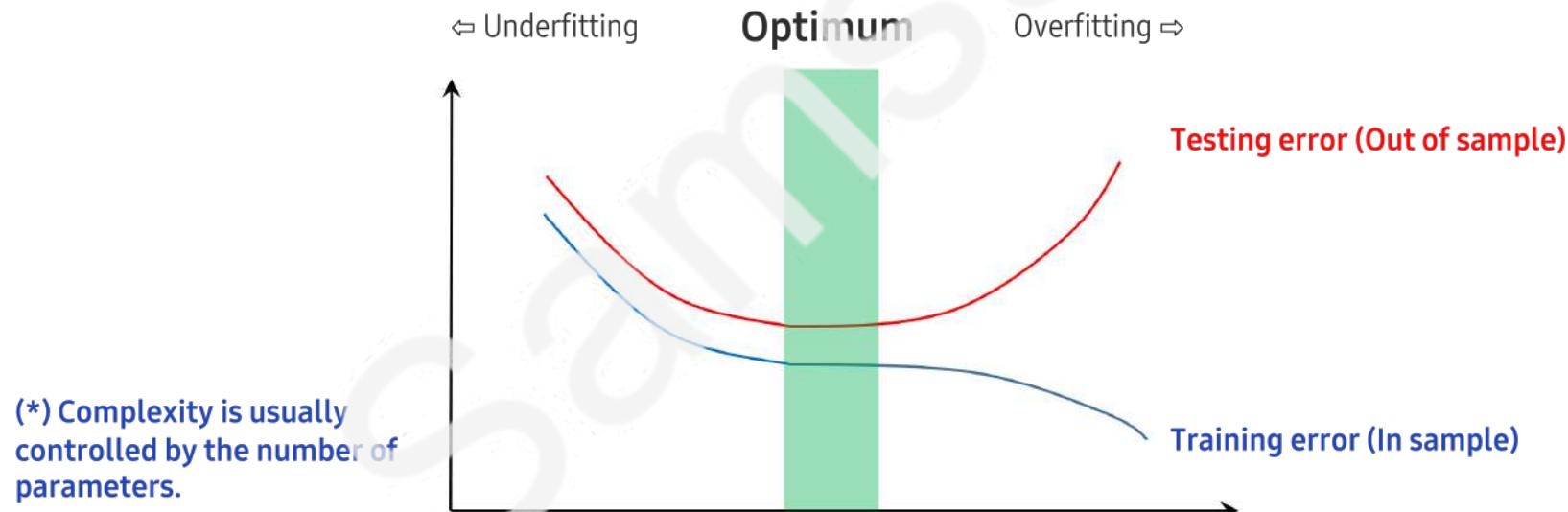
| Total error



- ▶ The goal is to minimize the **Total error** = **Bias error** + **Variance error**.
- ▶ Just enough complexity is required to “optimize” the model.

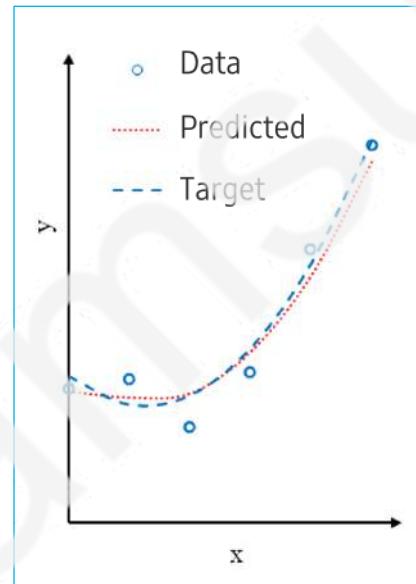
Minimizing Errors

- | Optimized machine learning model
 - ▶ Prediction performance should be good in both training and testing.
 - ▶ Given a machine learning algorithm, there is a model with just enough complexity (*).



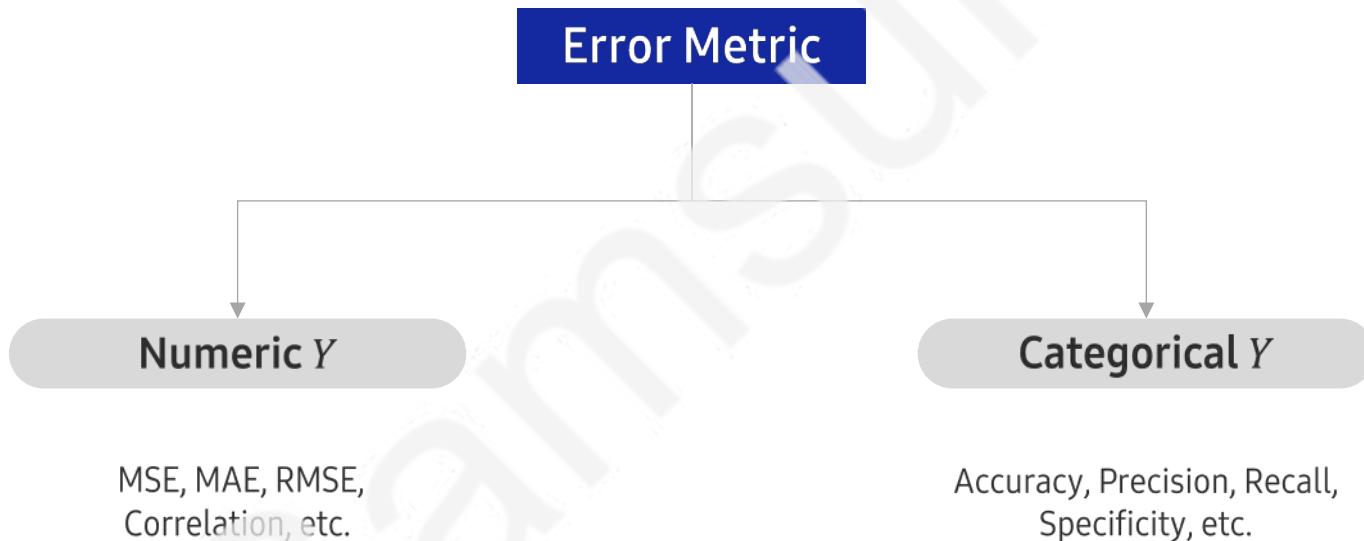
Optimized machine learning model

- ▶ Prediction performance should be good in both training and testing.
- ▶ Given a machine learning algorithm, there is a model with just enough complexity (*).



(*) Complexity is usually controlled
by the number of parameters.

Machine Learning Types

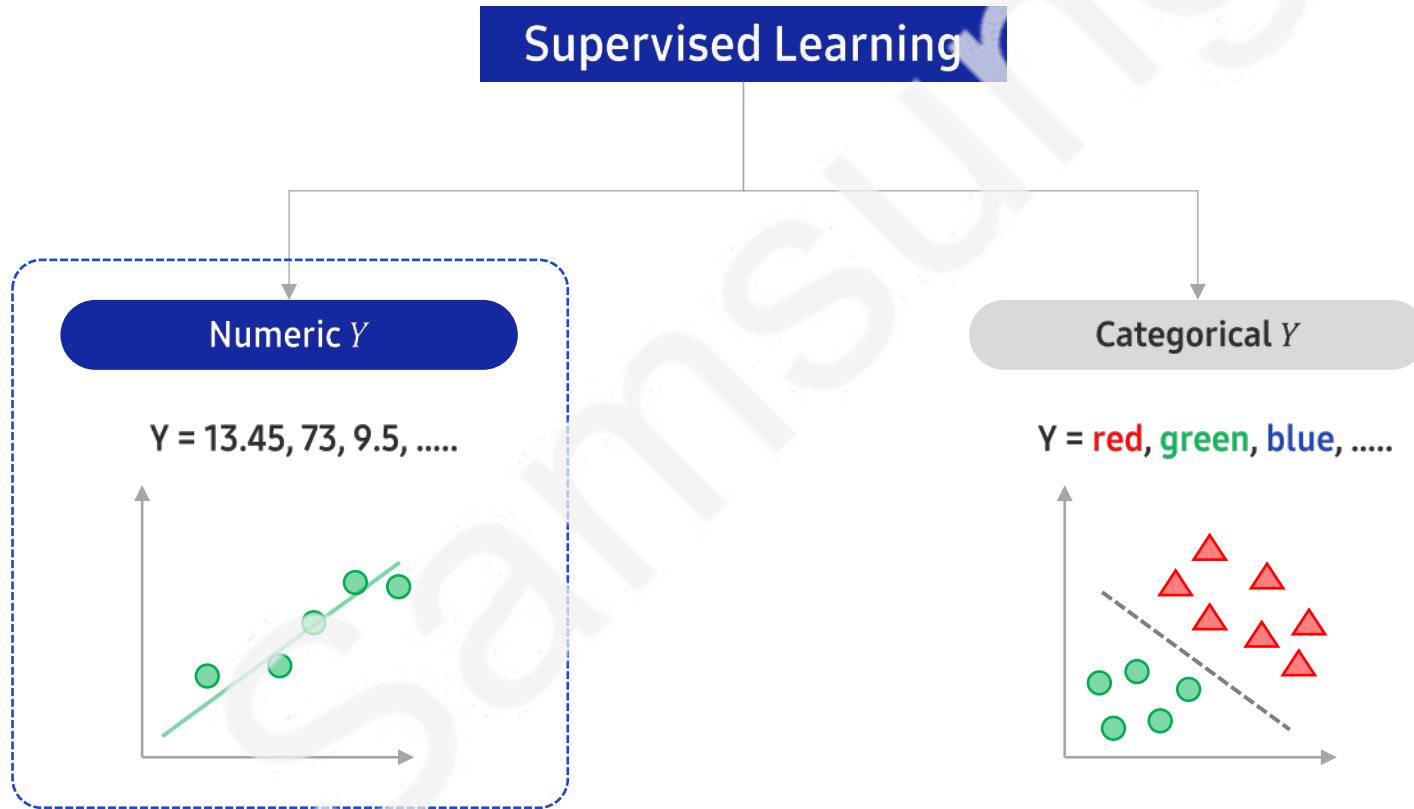


Unit 2.

Application of the Supervised Learning Model for Numerical Prediction

- | 2.1. Training and Testing in Machine Learning
- | **2.2. Linear Regression Basics**
- | 2.3. Linear Regression Diagnostics
- | 2.4. Other Regression Types
- | 2.5. Practicing the Supervised Learning Model for Numerical Prediction

Linear Regression Basics



| About linear regression

- ▶ There is one or more explanatory variables: X_1, X_2, \dots, X_k
- ▶ There is one response variable: Y
- ▶ The variables X_i and Y are connected by a linear relation: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k + \varepsilon$

Purpose of linear regression

a) By modeling, find out which explanatory variables have the most impact on the response variable.

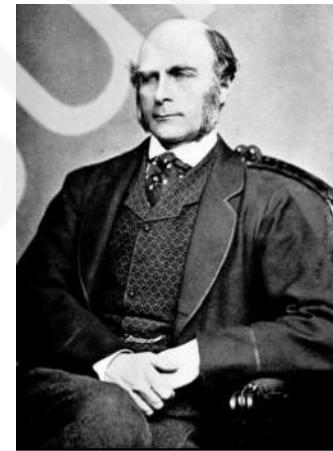
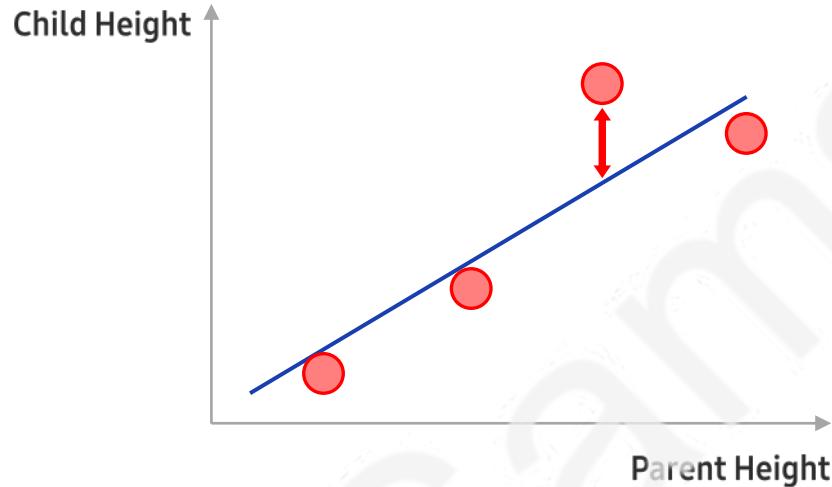
Ex If real estate price is the response variable Y , which are the most statistically meaningful explanatory variables? Area, location, age, distance to business center, etc.

b) Predict the response given the conditions for the explanatory variables.

Ex What is the price of a 10-year-old apartment with an area of $100 m^2$ and located 3 km away from the business center?
← “predict” the value that is not open to the public yet.

| Historical background

- ▶ Term “regression” was coined by Francis Galton, 19th-century biologist.
- ▶ The heights of the descendants tend to regress towards the mean.



Francis Galton

Pros

- ▶ Solid statistical and mathematical background
- ▶ Source of insights
- ▶ Fast training

Cons

- ▶ Many assumptions: linearity, normality, independence of the explanatory variables, etc.
- ▶ Sensitive to outliers
- ▶ Prone to multi-collinearity

Assumptions

- ▶ The response variable can be explained by a linear combination of the explanatory variables.
- ▶ There should be no multi-collinearity.
- ▶ Residuals should be normally distributed centered around 0.
- ▶ Residuals should be distributed with a constant variance.
- ▶ Residuals should be randomly distributed without a pattern.

} **Residual analysis**

| Linear model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$$

↑ ↑ ↑
Response variable Explanatory variables

- ▶ Variable values are given by data.

| Linear model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$$


Regression coefficients

- ▶ Regression coefficients are **model parameters**: capture the data patterns.

| Linear model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$$

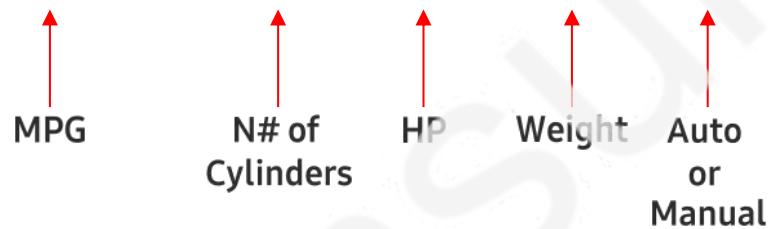
$$\begin{aligned}E[\varepsilon] &= 0 \\Var(\varepsilon) &= \sigma_\varepsilon^2\end{aligned}$$

- ▶ The error term ε should have zero mean and constant variance.

| Linear model

Ex

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \varepsilon$$



MPG can be explained by other variables.

| Interpreting the regression coefficients

$$\Delta Y = \beta_1 \Delta X_1 + \cdots + \beta_i \Delta X_i + \cdots + \beta_k \Delta X_k$$

- ▶ If X_1, X_2, \dots, X_k change by $\Delta X_1, \Delta X_2, \dots, \Delta X_k$, then the change in Y is ΔY .

| Interpreting the regression coefficients

$$\Delta Y = \beta_1 \Delta X_1 + \cdots + \beta_i \Delta X_i + \cdots + \beta_k \Delta X_k$$


- ▶ β_i can be interpreted as the change in Y when the X_i is increased by a unit ($\Delta X_i=1$).

| Interpreting the regression coefficients

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$$

↑ ||
Intercept 0

- ▶ The intercept β_0 is the value of Y when all the $X_i = 0$. It's like a "base line."

| Interpreting the regression coefficients

Ex Wage survey of a company's employees

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$$

↑ ↑ ↑
Wage Experience Qualification

- ▶ β_0 can be interpreted as the base wage when there is no experience or qualification.
- ▶ β_1 can be interpreted as the change in wage when the experience is increased by a unit.
- ▶ β_2 can be interpreted as the change in wage when the qualification is increased by a unit.

| Ordinary Least Squares (OLS) solution

$$y_j = \beta_0 + \beta_1 x_{j,1} + \beta_2 x_{j,2} + \cdots + \beta_K x_{j,k} + \varepsilon_j$$

$j \in [1, n]$ $n \gg k$
Over-determined!

- ▶ Now, we can write the linear relation in term of the actual data values.

| Ordinary Least Squares (OLS) solution

$$Y = X \beta + \varepsilon$$

- ▶ A compact notation using matrices

| Ordinary Least Squares (OLS) solution

$$\mathbf{Y} = \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- ▶ A compact notation using matrices

| Ordinary Least Squares (OLS) solution

$$Y = \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,k} \\ 1 & x_{2,1} & \cdots & x_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,k} \end{bmatrix}$$

- ▶ A compact notation using matrices

| Ordinary Least Squares (OLS) solution

$$Y = X \beta + \varepsilon$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}$$

- ▶ A compact notation using matrices

| Ordinary Least Squares (OLS) solution

$$\mathbf{Y} = \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\varepsilon}$$
$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

- ▶ A compact notation using matrices

| Ordinary Least Squares (OLS) solution

- ▶ As we have an overdetermined system of linear equations, the exact solution **does not exist**.
- ▶ We can minimize $|\boldsymbol{\epsilon}|^2$ and get the “best” solution $\boldsymbol{\beta}$.
- ▶ The minimization condition for $|\boldsymbol{\epsilon}|^2$ is given by the derivative:

$$\frac{d|\boldsymbol{\epsilon}|^2}{d\beta} = 0$$

| Ordinary Least Squares (OLS) solution

- ▶ The minimization condition for $|\boldsymbol{\epsilon}|^2$ is given by the derivative that can be expanded as following:

$$\begin{aligned}\frac{d|\boldsymbol{\epsilon}|^2}{d\beta} &= \frac{d|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}|^2}{d\beta} \\ &= \frac{d\{(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^t(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta)}\}}{d\beta} \\ &= \frac{d\{\mathbf{Y}^t\mathbf{Y} - \boldsymbol{\beta}^t\mathbf{X}^t\mathbf{Y} - \mathbf{Y}^t\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^t\mathbf{X}^t\mathbf{X}\boldsymbol{\beta}\}}{d\beta} \\ &= -2\mathbf{X}^t\mathbf{Y} + 2\mathbf{X}^t\mathbf{X}\boldsymbol{\beta} = 0\end{aligned}$$

| Ordinary Least Squares (OLS) solution

- ▶ The solution β from the previous slide is given by the following expression:

$$\beta = [(X^t X)^{-1} X^t] Y$$

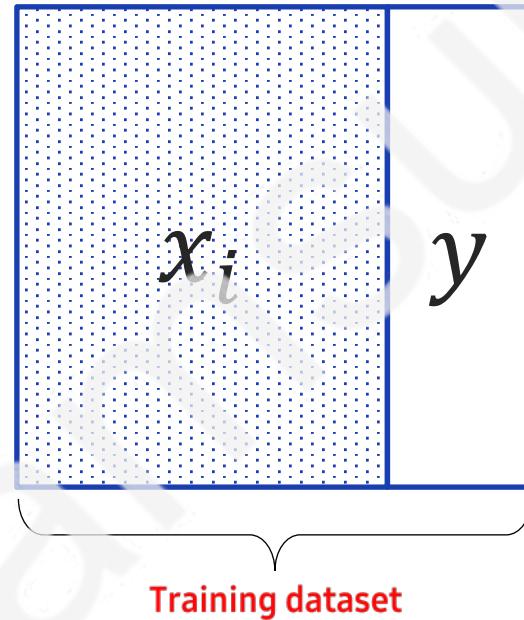


Pseudo-inverse

- ▶ The matrix expression within the square parentheses is called “pseudo-inverse.”

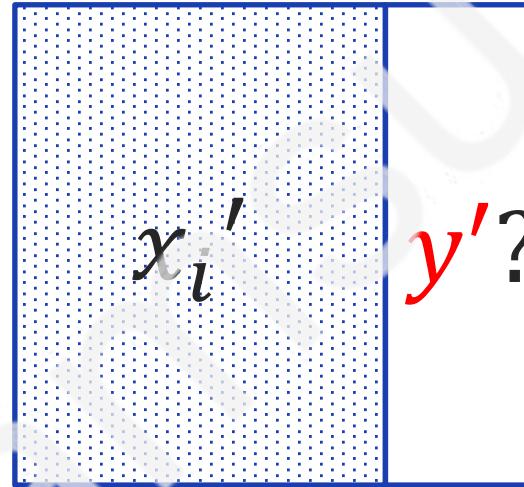
| Regression training and prediction (testing)

- 1) Training step: use the training dataset and get a set of model parameters $\{\beta_i\}$.



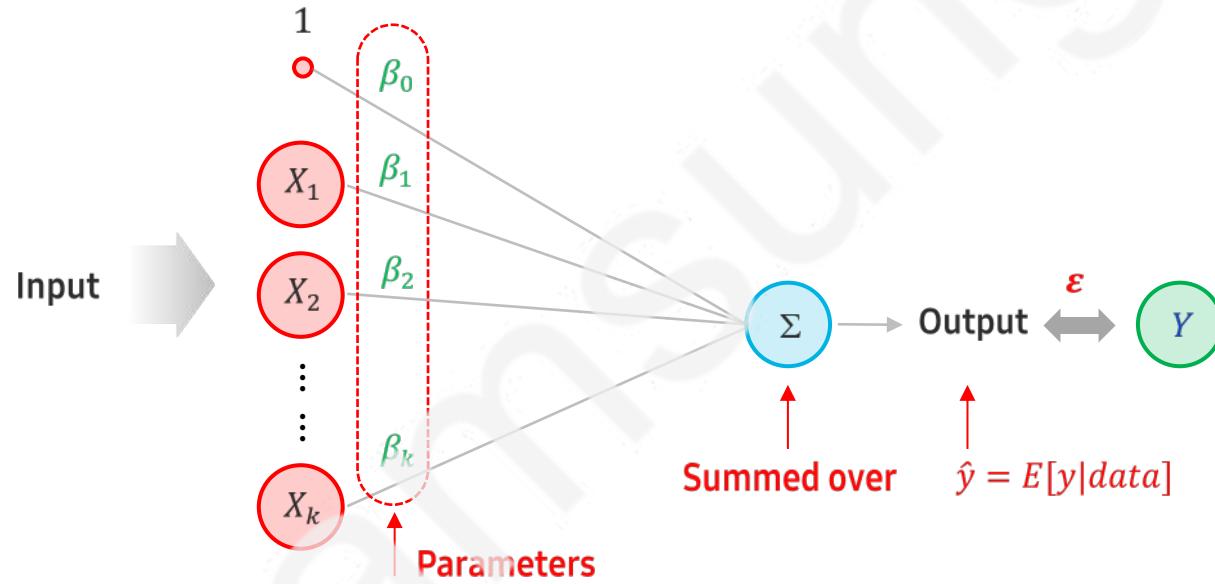
| Regression training and prediction (testing)

2) Prediction step: when a new set of $\{x_i'\}$ is given, calculate the value of y' , which was unknown.



- ▶ The predicted value of y' is denoted as \hat{y} , which is a conditional expectation $\hat{y} = E[y|data]$.
- ▶ Given the values x_1', x_2', \dots, x_k' , calculate $\hat{y} = \beta_0 + \beta_1 x_1' + \beta_2 x_2' + \dots + \beta_K x_k'$.

| Regression training and prediction (testing)



- ▶ Schematic view of the prediction step
- ▶ Notice some disagreement between the predicted \hat{y} and the true y .

| Error metrics

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

$$RMSE = \sqrt{MSE}$$

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

$$MAPE = \frac{100}{n} \times \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

| Confidence interval (95%) when there is only one explanatory variable

$$[\hat{y} - \text{quantile}(0.975, n - 2) \sigma_{\hat{y}}, \hat{y} + \text{quantile}(0.975, n - 2) \sigma_{\hat{y}}]$$

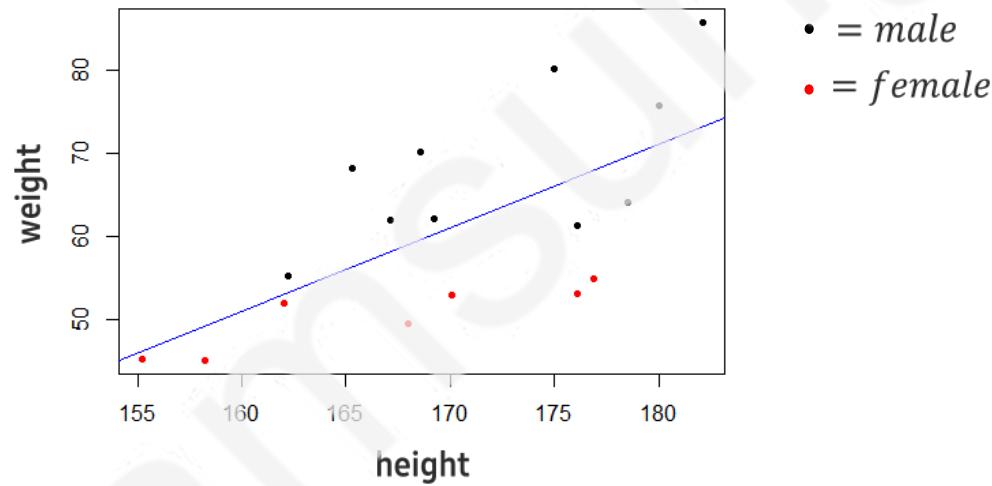
with

$$\sigma_{\hat{y}} = RMSE \times \sqrt{\left(\frac{1}{n} + \frac{(x' - \bar{x})^2}{\sum(x - \bar{x})^2} \right)}$$

- ▶ $\text{quantile}(\alpha, k)$ is Student-t quantile of degree of freedom k .

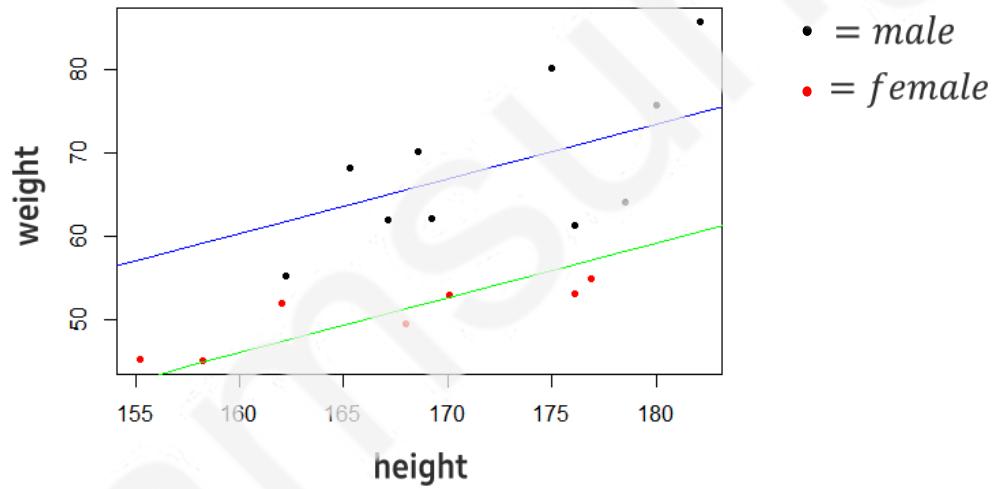
| Role of categorical variables

- ▶ Without categorical variable: weight ~ height



Role of categorical variables

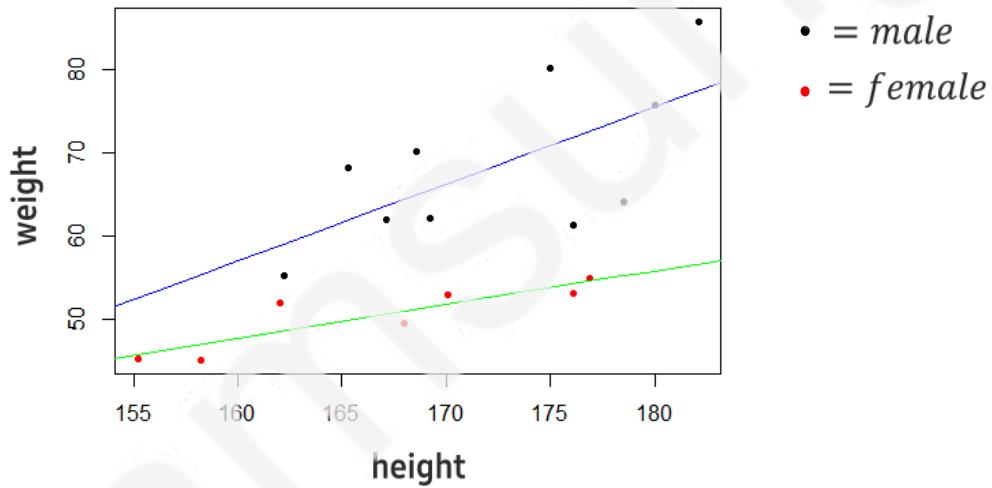
- With a categorical variable **gender**: weight ~ height + **gender**



- The categorical variable must be turned into dummy variable(s) first.
- Effectively raises or lowers the intercept.

Role of categorical variables

- With a categorical variable **gender** that “interacts”: weight ~ height + **gender** + height \times **gender**



- Both the intercept and slope are dependent on the categorical variable.
- Further improves the error metrics.

Coding Exercise #0301



Follow practice steps on 'ex_0301.ipynb' file.

Unit 2.

Application of the Supervised Learning Model for Numerical Prediction

- | 2.1. Training and Testing in Machine Learning
- | 2.2. Linear Regression Basics
- | **2.3. Linear Regression Diagnostics**
- | 2.4. Other Regression Types
- | 2.5. Practicing the Supervised Learning Model for Numerical Prediction

Linear Regression Diagnostics

| Linear regression diagnostic methods

- 1) Error metrics: MSE, RMSE, MAE, MAPE, etc.
- 2) Coefficient of determination or “r-squared” R^2
- 3) F-test for overall significance of the linear model
- 4) t-test for significance of individual regression coefficients
- 5) Correlation between Y and \hat{Y}
- 6) Variance inflation factor (VIF)

| Modelling: optimization of the information criteria: AIC or BIC

| Residual and leverage analysis

| Error metrics

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

$$RMSE = \sqrt{MSE}$$

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

$$MAPE = \frac{100}{n} \times \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

- ▶ The smaller, the better!

| Coefficient of determination or R^2

$$R^2 = 1 - \frac{SSE}{SST}$$

with $SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ and $SST = \sum_{i=1}^n (y_i - \bar{y})^2$

- ▶ R^2 is bounded above and below: $0 < R^2 < 1$.
- ▶ R^2 close to one means that the response variable is well explained.
- ▶ As more explanatory variables are added, R^2 tends to increase spuriously: adjusted R^2 introduced.
- ▶ If there is only one explanatory variable X , then:

$$R^2 = Cor(X, Y)^2$$

| F-test for overall significance of the linear model

- ▶ Null hypothesis $H_0: \beta_1 = \beta_2 = \dots = \beta_K = 0$
- ▶ Alternate hypothesis H_1 : At least a β_i is non zero.

⇒ F-test statistic =

$$\frac{\text{Variance that } \textcolor{blue}{\text{can be explained}}}{\text{Variance that } \textcolor{red}{\text{cannot be explained}}}$$

⇒ If the p-value is below a reference (say 0.05), then H_0 is rejected in favor of H_1 .

In this case, the linear model has an overall significance.

| t-test for significance of the individual regression coefficients

- ▶ Null hypothesis **H0**: $\beta_i = 0$
- ▶ Alternate hypothesis **H1**: $\beta_i \neq 0$

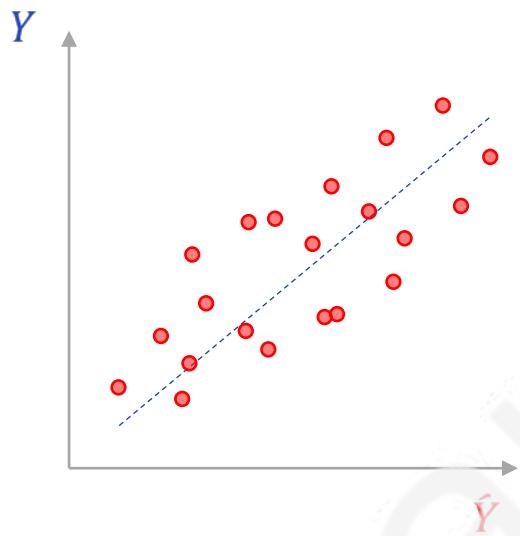
↳ t-test statistic =

$$\frac{\widehat{\beta}_i}{\text{Standard error of } \beta_i}, \text{ where } \widehat{\beta}_i = \text{estimated coefficient.}$$

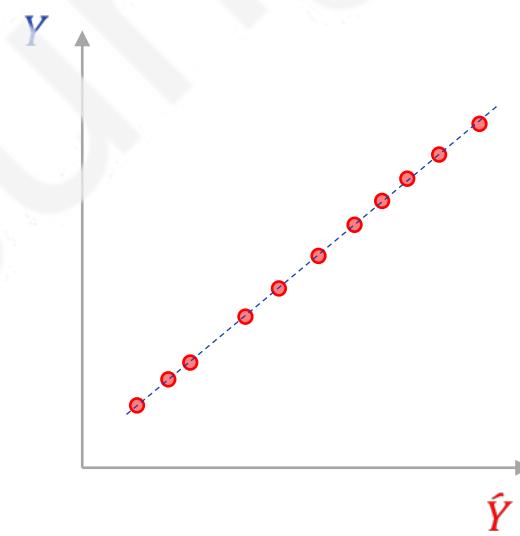
↳ If the p-value is below a reference (say 0.05), then **H0** is rejected in favor of **H1**.

In this case, inclusion of the explanatory variable X_i is justified.

| Correlation between Y and \hat{Y}



Weak positive correlation



Strong positive correlation

| Variance inflation factor (VIF)

- ▶ Can measure the “seriousness” of multi-collinearity.
- ▶ In general, $VIF > 10$ is considered serious. However, this reference point is rather subjective.
- ▶ In case serious multi-collinearity is detected, model simplification is required.

| Variance inflation factor (VIF)

- ▶ VIF is calculated individually for each explanatory variable X_i .
 - 1) Place X_i as the response and then regress using the expression:

$$X_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_{i-1} X_{i-1} + \beta_{i+1} X_{i+1} + \cdots + \varepsilon$$

- 2) Using the corresponding R^2 calculate VIF_i :

$$VIF_i = \frac{1}{1 - R_i^2}$$

| Information criteria and modelling

- ▶ Akaike information criteria (AIC) with p =number of parameters:

$$AIC = -2 \frac{\text{Log likelihood}}{n} + 2 \frac{p}{n}$$

- ▶ Bayes information criteria (BIC) with p =number of parameters:

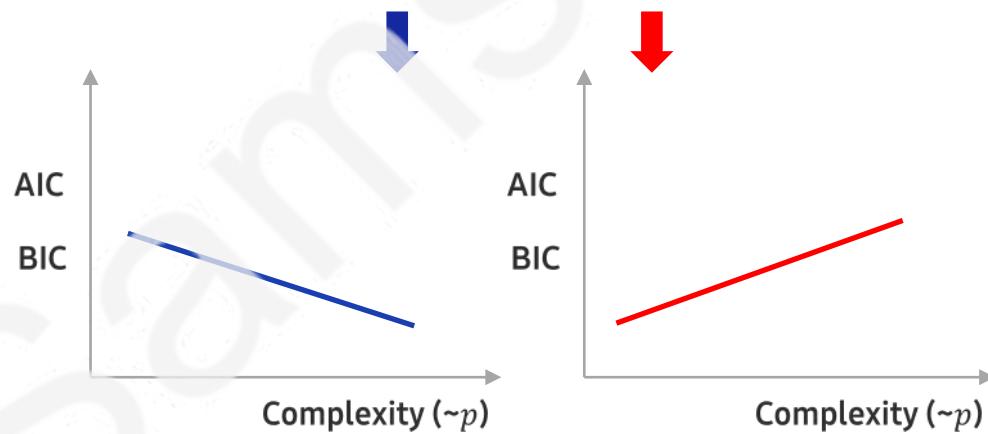
$$BIC = -2 \frac{\text{Log likelihood}}{n} + p \frac{\ln(n)}{n}$$

- ▶ In the expressions above, we have:

$$\text{Log Likelihood} = -\frac{n}{2} \left(1 + \ln(2\pi) + \ln\left(\frac{SSE}{n}\right) \right)$$

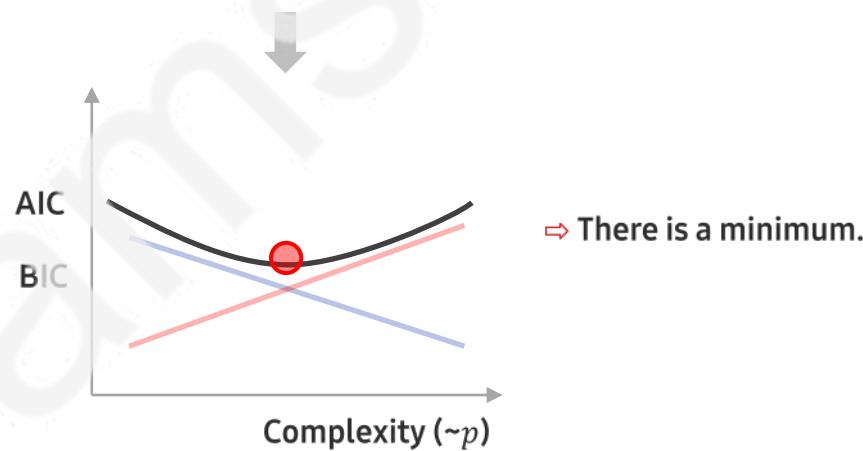
| Information criteria and modelling

$$AIC = -2 \frac{\text{Log likelihood}}{n} + 2 \frac{p}{n}$$
$$BIC = -2 \frac{\text{Log likelihood}}{n} + p \frac{\ln(n)}{n}$$

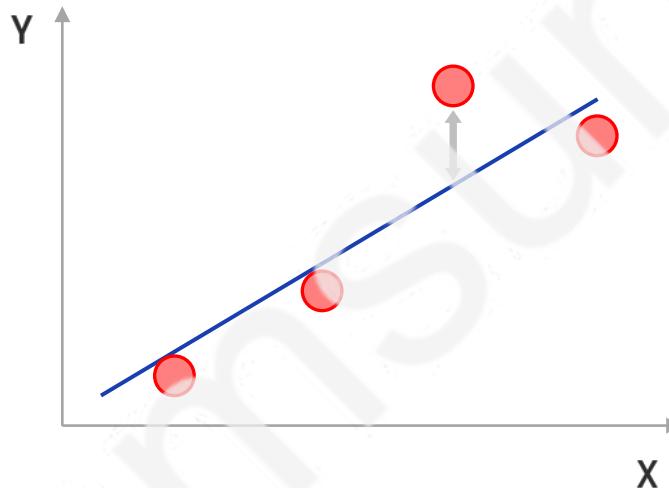


| Information criteria and modelling

$$AIC = -2 \frac{\text{Log likelihood}}{n} + 2 \frac{p}{n}$$
$$BIC = -2 \frac{\text{Log likelihood}}{n} + p \frac{\ln(n)}{n}$$



| Residual analysis



- ▶ Residual is the difference between the predicted \hat{y} and the real y .
- ▶ We can easily detect outliers in Y that deviate substantially from the main trend.

Residual analysis

- ▶ Reasons for residual analysis:

- 1) To detect outliers in Y .
- 2) To verify the assumptions of linear regression.

Residuals should be normally distributed centered around 0.

Residuals should be distributed with a constant variance.

Residuals should be randomly distributed without a pattern.

| Leverage analysis

- ▶ Leverage of the i-th observation

$$\text{Leverage} = H_{ii} \ , \ \text{where } \tilde{H} = \tilde{X}(\tilde{X}^t\tilde{X})^{-1}\tilde{X}^t$$

- ▶ There is a “sum rule.”

$$\sum_{i=1}^n H_{ii} = p$$

Mean leverage $\cong \frac{p}{n}$
Large leverage $> \frac{p}{n}$
Small leverage $< \frac{p}{n}$

- ▶ Leverage tells how distant X is from the center \Rightarrow Detection of outliers in X .

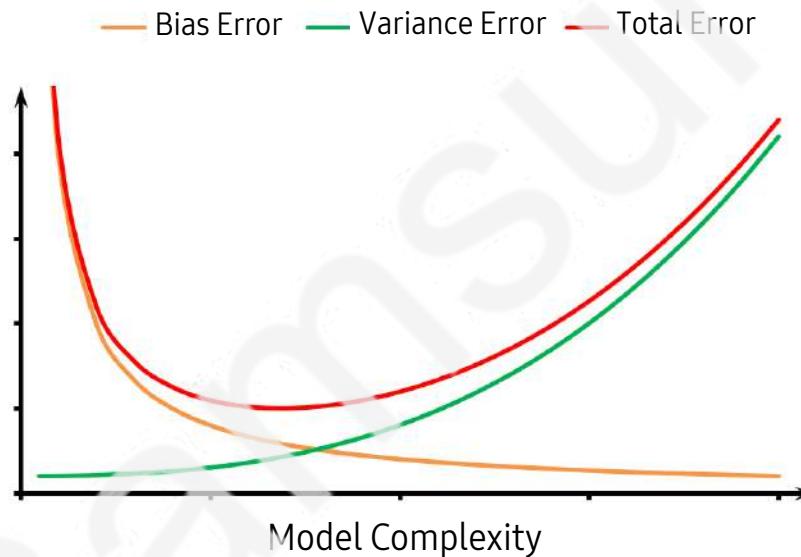
Unit 2.

Application of the Supervised Learning Model for Numerical Prediction

- | 2.1. Training and Testing in Machine Learning
- | 2.2. Linear Regression Basics
- | 2.3. Linear Regression Diagnostics
- | **2.4. Other Regression Types**
- | 2.5. Practicing the Supervised Learning Model for Numerical Prediction

Regularized Regression

| Bias-Variance trade off



- ▶ Tradeoff relation between the **Bias error** and the **Variance error**.
- ▶ The goal should be to minimize the **Total error** = **Bias error** + **Variance error**.

| Ridge regression

- ▶ Useful when the usual linear regression overfits (bias error <> variance error).
- ▶ We remember that the OLS solution consists in minimizing $|\boldsymbol{\epsilon}|^2$.
- ▶ In the Ridge regression, we minimize the following “loss function”:

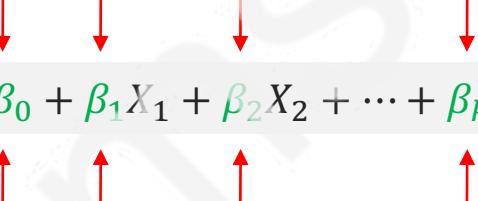
$$L = |\boldsymbol{\epsilon}|^2 + \lambda \sum_{i=0}^k \beta_i^2$$

- ▶ In the loss function, $\lambda \sum_{i=0}^k \beta_i^2$ is included as penalty: “L₂ regularization.”

| Ridge regression

- ▶ Positive and larger λ further constrains the coefficients β_i , decreasing the variance (overfitting) error.

$$L = |\boldsymbol{\varepsilon}|^2 + \lambda \sum_{i=0}^k \beta_i^2$$

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$$


- ▶ However, too large λ can make the model too "biased."
- ▶ Even when λ is large, the coefficients β_i never become exactly equal to zero.

| Lasso regression

- ▶ Useful when the usual linear regression overfits (bias error << variance error).
- ▶ In the Lasso regression, we minimize the following “loss function”:

$$L = |\boldsymbol{\varepsilon}|^2 + \lambda \sum_{i=0}^k |\beta_i^2|$$

- ▶ In the loss function, $\lambda \sum_{i=0}^k |\beta_i|$ is included as penalty: “L1 regularization.”
- ▶ Positive and larger λ further constraints the coefficients β_i decreasing the variance (overfitting) error.
- ▶ However, too large λ can make the model too “biased.”
- ▶ When λ is large, the coefficients β_i can become exactly equal to zero.

Polynomial Regression

| Polynomial regression

- ▶ Useful when the usual linear regression underfits (bias error \gg variance error).
- ▶ We can model the relationship between X and Y using the polynomials:

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

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon$$

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon$$

- ▶ We notice that there is **only one** explanatory variable X .
- ▶ When the polynomial power is too high, overfitting may incur.

Poisson Regression

| Poisson regression

- ▶ Useful when we would like to model the response Y that represents counts or frequencies.

$$\text{Log}(\lambda) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_K X_K + \varepsilon$$

- ▶ We are assuming that Y follows the Poisson distribution.

$$P(y) = \frac{\lambda^y e^{-\lambda}}{y!}$$

- a) Mean = λ
- b) Variance = λ
- c) Standard deviation = $\sqrt{\lambda}$

Unit 2.

Application of the Supervised Learning Model for Numerical Prediction

- | 2.1. Training and Testing in Machine Learning
- | 2.2. Linear Regression Basics
- | 2.3. Linear Regression Diagnostics
- | 2.4. Other Regression Types
- | 2.5. Practicing the Supervised Learning Model for Numerical Prediction**

Practicing the Supervised Learning Model for Numerical Prediction

| Import modules.

```
In [ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

Coding Exercise #0302



Follow practice steps on 'ex_0302.ipynb' file.

Coding Exercise #0303



Follow practice steps on 'ex_0303.ipynb' file.

Coding Exercise #0304



Follow practice steps on 'ex_0304.ipynb' file.

Coding Exercise #0305



Follow practice steps on 'ex_0305.ipynb' file.

Coding Exercise #0306



Follow practice steps on 'ex_0306.ipynb' file.

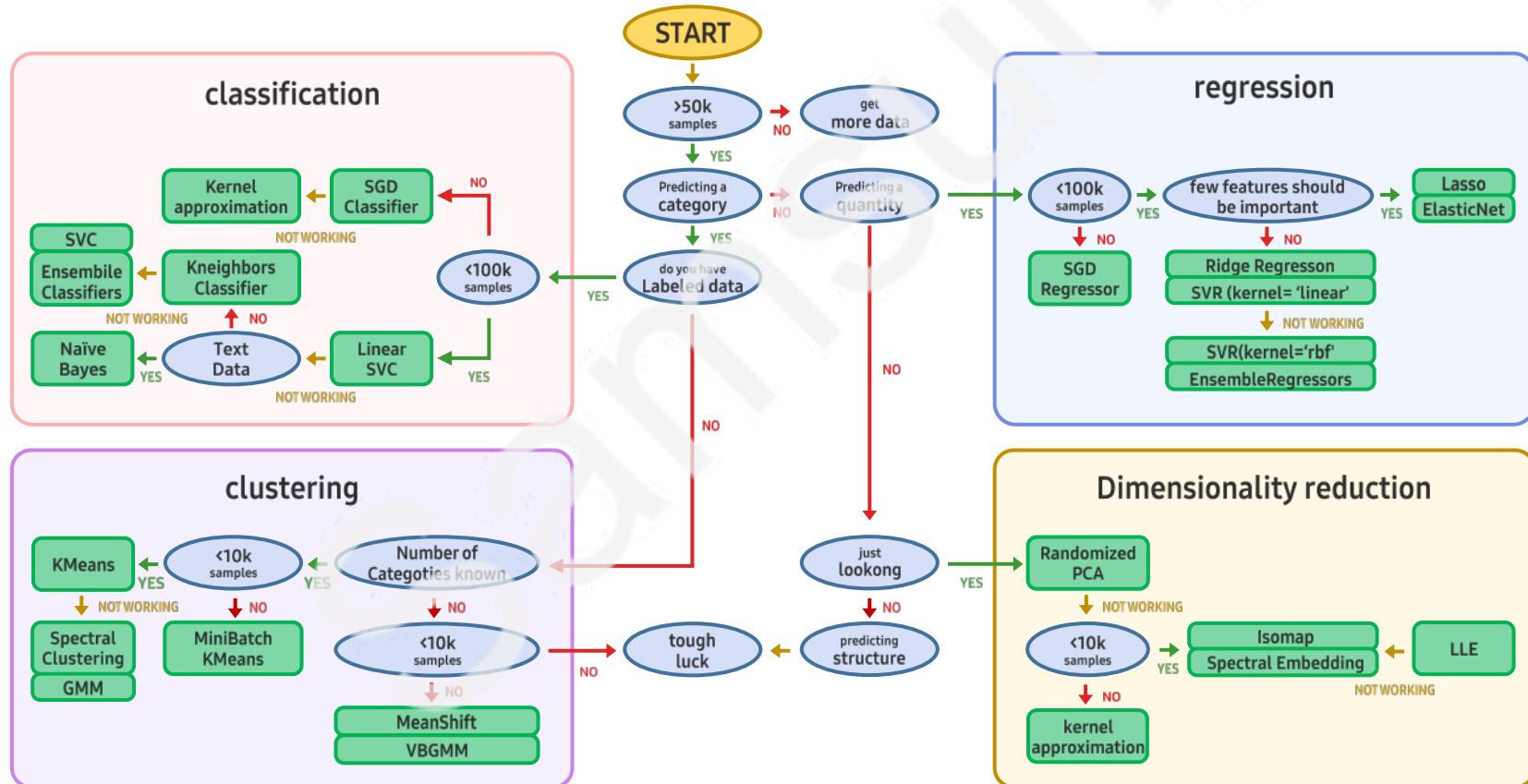
Unit 3.

Application of Supervised Learning Model for Classification

- | **3.1. Training and Testing in Machine Learning**
- | 3.2. Logistic Regression Basics
- | 3.3. Logistic Regression Performance Metrics

Training and Testing in Machine Learning

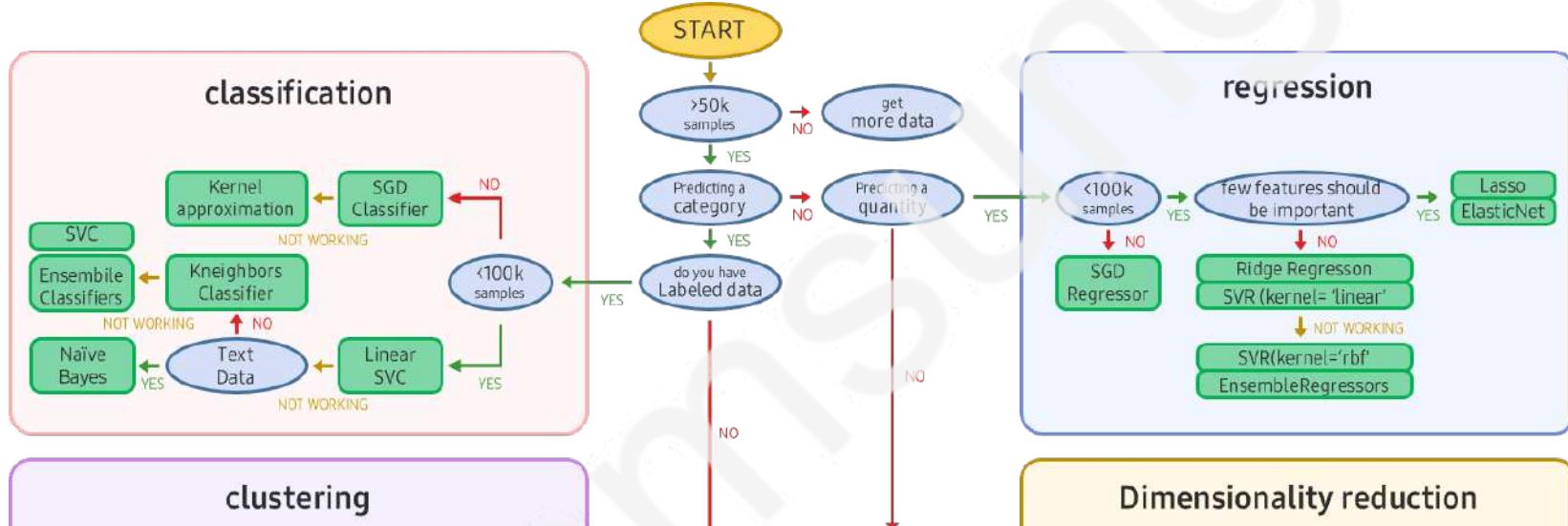
| Select a machine learning algorithm suitable for the data.



>Select a machine learning algorithm suitable for the data.

- ▶ Supervised learning is machine learning tasks that try to predict or classify the values of objective variables (or response variables) in unseen data as the training data set has the objective variables (or response variables) to be predicted (Y).
- ▶ It is called supervised learning because the algorithm is told to predict a certain target.
- ▶ Supervised learning is mainly classified into classification and regression problems depending on the types of objective variables (or response variables).
- ▶ In other words, classification is applied if the objective variables (or response variables) are discrete or nominal, such as 'male/female,' 'spam mail/ham mail,' 'positive/neutral/negative,' 'Seoul/Busan/Gyeonggi/Gangwon,' etc. On the other hand, regression is applied if the objective variables (or response variables) are numerical, such as 0~10, -500~500, -3.5~3.5, - $\infty\sim\infty$, etc.
- ▶ When the given data does not have classification marks or objective variables (or response variables), and if it's not trying to predict target value, unsupervised learning is applied.
- ▶ Unsupervised learning is classified into clustering, association, dimension reduction, and others depending on the analysis purpose and methods. Further details will be explained in other chapters.

Select a machine learning algorithm suitable for the data.



- As shown in the figure above, neural networks can be applied if there are correct answers and it is for classification purposes or has a lot of data. If not, it is possible to use a decision tree or SVC algorithm.
- Also, the Naïve Bayes algorithm can be used if the data is in text.
- The KNN method is used if it's not text data. It is also possible to use the ensemble method with a better performance.

Machine learning for classification

- ▶ Classification is used if the objective variables (or response variables) can be classified into certain categories, such as discrete or nominal type. It is the most common and frequently found in machine learning-based data analysis.
- ▶ Machine learning algorithms for classification can be applied to extensive daily and business problems.

Ex Frequently used examples

- (1) Classifying spam mails
 - (2) Prediction of corporate bankruptcy
 - (3) Prediction of customer loss
 - (4) Classifying customers' credit rating
 - (5) Prediction of occurrence of a certain disease (e.g. cancer, heart disease, etc.)
 - (6) Prediction of customer reaction to a specific marketing event
 - (7) Prediction of customer's purchase
- ▶ The examples provided above are extremely few real-life applications; there are infinite fields of business that classification type machine learning is applied. This type of machine learning has been continuously applied with new R&D and business methods.

Types of machine learning algorithm for classification (1/2)

- There are many different classification-type machine learning algorithms. Some of those can be used for regression problems. The following table provides descriptions of frequently used classification methods.

Type	Concept	Note
K-Nearest Neighbor	A classification method that uses the majority rule on the closest k objective variable values (or response variables) based on the distance between data coordination other than a certain data.	Lazy Learning
Naïve Bayes	A method based on Bayes' theorem that makes classification towards the higher probability by expressing the conditional probability of objective variables (or response variables) as a multiplication of the prior probability and likelihood function. All observed values are assumed to occur statistically independently from other observed values. (Referred to as a Naïve model since the assumption is given without confidence.)	Probability model (Bayes' theorem based conditional probability)
Logistic Regression	A method to estimate the probability of objective variables through the maximum likelihood estimation by assuming that the probability of the objective variable value being in a certain category is in the logistic function shape when the explanatory value is given.	Probability model (maximum likelihood estimation)
Decision Tree	A method to create classification rules by splitting the branches towards lower impurity or entropy in the order of variables most associated with objective variables.	Divide & Conquer

Types of machine learning algorithm for classification (2/2)

- There are many different classification-type machine learning algorithms. Some of those can be used for regression problems. The following table provides descriptions of frequently used classification methods.

Type	Concept	Note
Artificial Neural Network	A method inspired by the human neuron network. Comprised of the input node, hidden node, and output node, this analysis method is used to solve complicated classification or black box value prediction problems.	Black box test
Support Vector Machine	A method to classify certain data by finding a plane that maximizes the margin between data in different categories.	Linear and non-linear (Kernel trick)
Random Forest	An ensemble method to decide the final classification result by making various decision trees based on given data and aggregating the predicted results of each decision tree through voting.	Ensemble model

Unit 3.

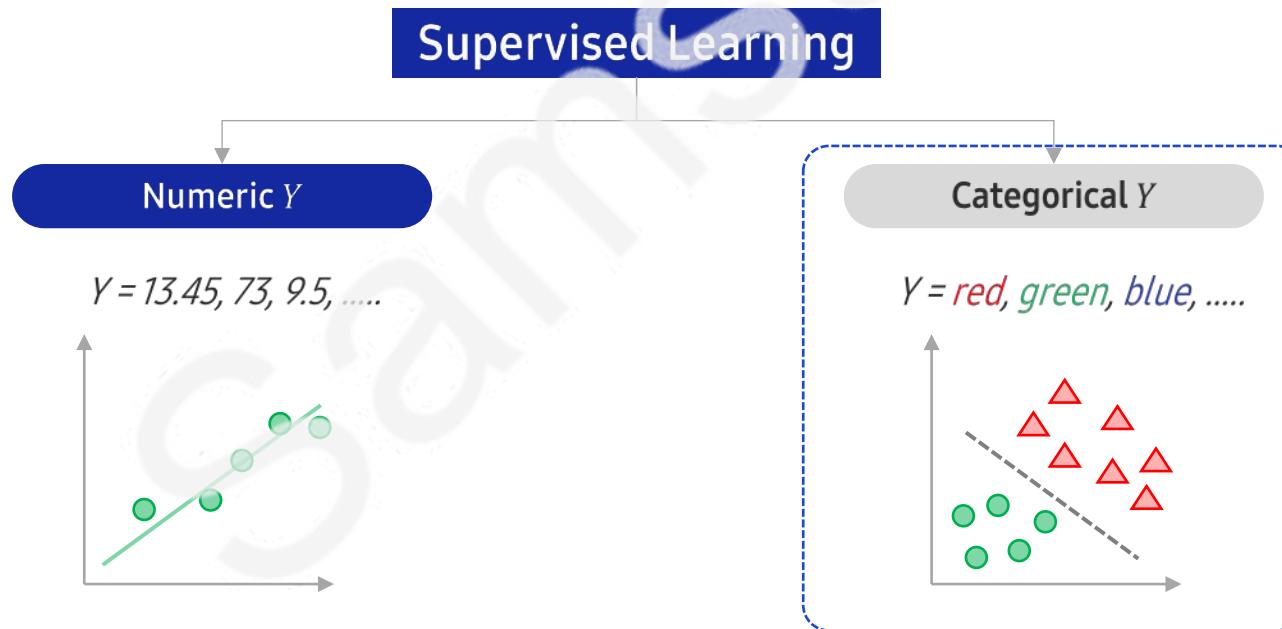
Application of Supervised Learning Model for Classification

- | 3.1. Training and Testing in Machine Learning
- | **3.2. Logistic Regression Basics**
- | 3.3. Logistic Regression Performance Metrics

Logistic Regression Basics

| What is logistic regression analysis?

- In contrast to general linear regression analysis used for numerical prediction, logistic regression analysis is used to classify the category of objective variables (y) to be predicted. In other words, what is being predicted is not the y value which is an objective variable. It is $P(Y=i)$, which is the probability of the objective variable y becoming a certain category (i).



| About linear regression

- ▶ There is one or more explanatory variables: X_1, X_2, \dots, X_k
- ▶ There is one response variable: Y
- ▶ The response variable Y is binary where possible values are {0,1}, {False, True}, {No, Yes}, etc.
- ▶ One of the most basic classification algorithms.

Pros

- ▶ Simple and relatively easy to implement
- ▶ Source of intuitive insights
- ▶ Fast training

Cons

- ▶ Not among the most accurate classification algorithms
- ▶ Assumes that the explanatory variables are independent without multi-collinearity

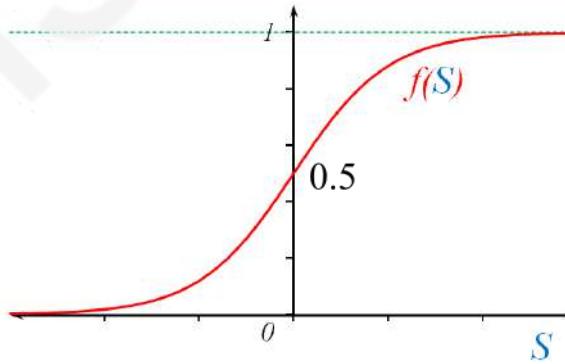
| About logistic regression

- ▶ The linear combinations of variables X_i is the so-called “Logit” denoted here as S .

$$S = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

- ▶ The conditional probability of Y being equal to 1 is denoted as $P(Y = 1 | \{x_i\})$.
- ▶ “Sigmoid” or “Logistic” function connects the probability with the logit.

$$f(S) = \frac{e^S}{1 + e^S}$$



| About logistic regression

- ▶ If $p = \text{probability of } Y=1$, we can define:

$$\text{Odds} = \frac{p}{1-p}$$

$$\text{Logit} = \text{Log(Odds)} = \text{Log}\left(\frac{p}{1-p}\right)$$

- ▶ The logistic function is the inverse of the logit (and vice versa):

$$S = \text{Log}\left(\frac{p}{1-p}\right)$$



$$p = \frac{e^S}{1 + e^S}$$

About logistic regression

- For logistic regression, the categories of objective variable (Y) to be predicted are 0 and 1 (binomial logistic regression model). The possibility of the objective variable Y category becoming 1 is expressed as $p(Y=1) = P(Y)$. Start by expressing the description as a regression equation. The left side of the equation is commonly referred to as 'odds.'
- The 'odds' signify dividend rate, which categorizes into success/failure probabilities. If the success probability is high, it becomes 1; if it is low, it becomes 0. In other words, it is a success rate. Adding a log to odds becomes logit, and this function is logistic.

$$\frac{P(Y)}{1 - P(Y)} = \exp(\beta_0 + \beta_1 X)$$

- The left side of the equation is a ratio of probability; the right side is an exponential function that has a $(0, \infty)$ range. Add a log on both sides of the equation to give a range with $(-\infty, \infty)$ values on both.

$$\log\left(\frac{P(Y)}{1 - P(Y)}\right) = \beta_0 + \beta_1 X$$

- When looking at the equation in detail, the $\beta_0 + \beta_1 X$ on the right side is a linear model with a range $(-\infty, \infty)$; the left side also has a range $(-\infty, \infty)$. The $\log(P(Y))/(1-P(Y))$ on the left side of the equation is called the logit function.
- Or add 'exp' to both sides of the equation and arrange it regarding $P(Y)$ to get the following equation.

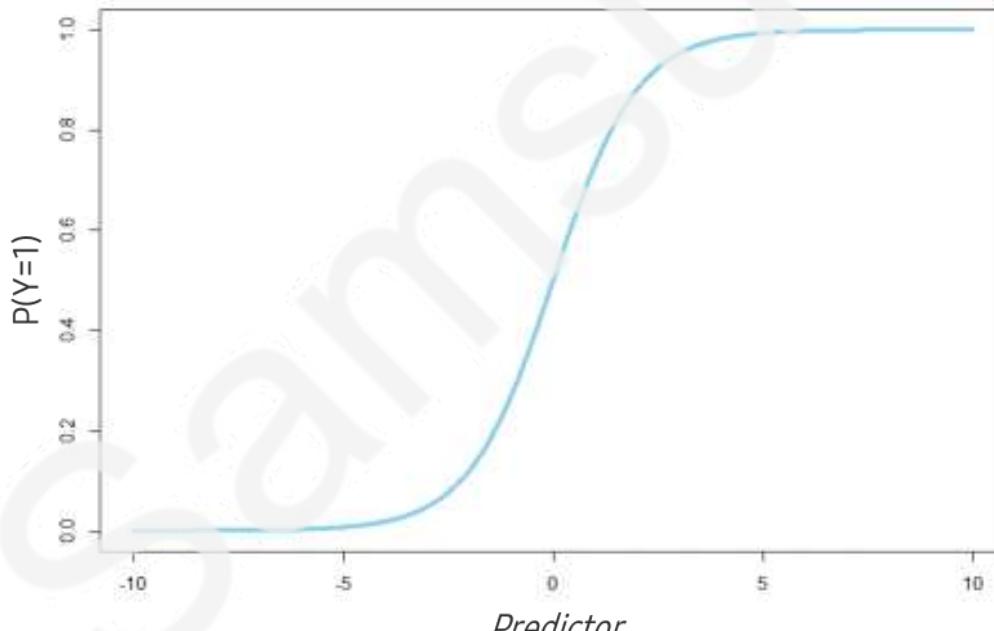
| About logistic regression

$$P(Y = 1) = \frac{e^{(\beta_0 + \beta_1 X)}}{1 + e^{(\beta_0 + \beta_1 X)}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

- ▶ To sum up, logistic regression analysis is an algorithm that establishes a model by using the logistic function from the above equation and predicting parameters β_0 , β_1 from the probability $P(Y=1) = P(Y)$ where objective variable Y having categorical value 1 from the training data. Maximum likelihood estimation is generally used for predicting parameters β_0 and β_1 . This is an analytical method that changes the formula. Since the direct calculation is difficult, estimation is done by giving a certain initial value and performing repeated calculations to adjust the value as a numerical calculation.

About logistic regression

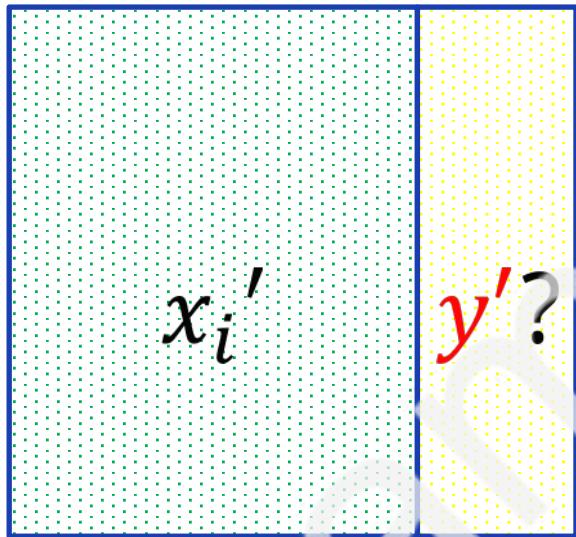
- The following figure shows a graph of the logistic function. As shown in the graph, the range of the X-axis is $-\infty$ to ∞ . The Y-axis is the probability of objective variable Y occurrence, which ranges between 0 and 1. Thus, the function $P(Y=1)$ shows an S-shaped curve from 0 to 1 as the x value increases.



Logistic regression graph

| Logistic regression training and prediction (testing)

2) Prediction step: when a new set of $\{x_i'\}$ is given, calculate the value of y' , which was unknown.

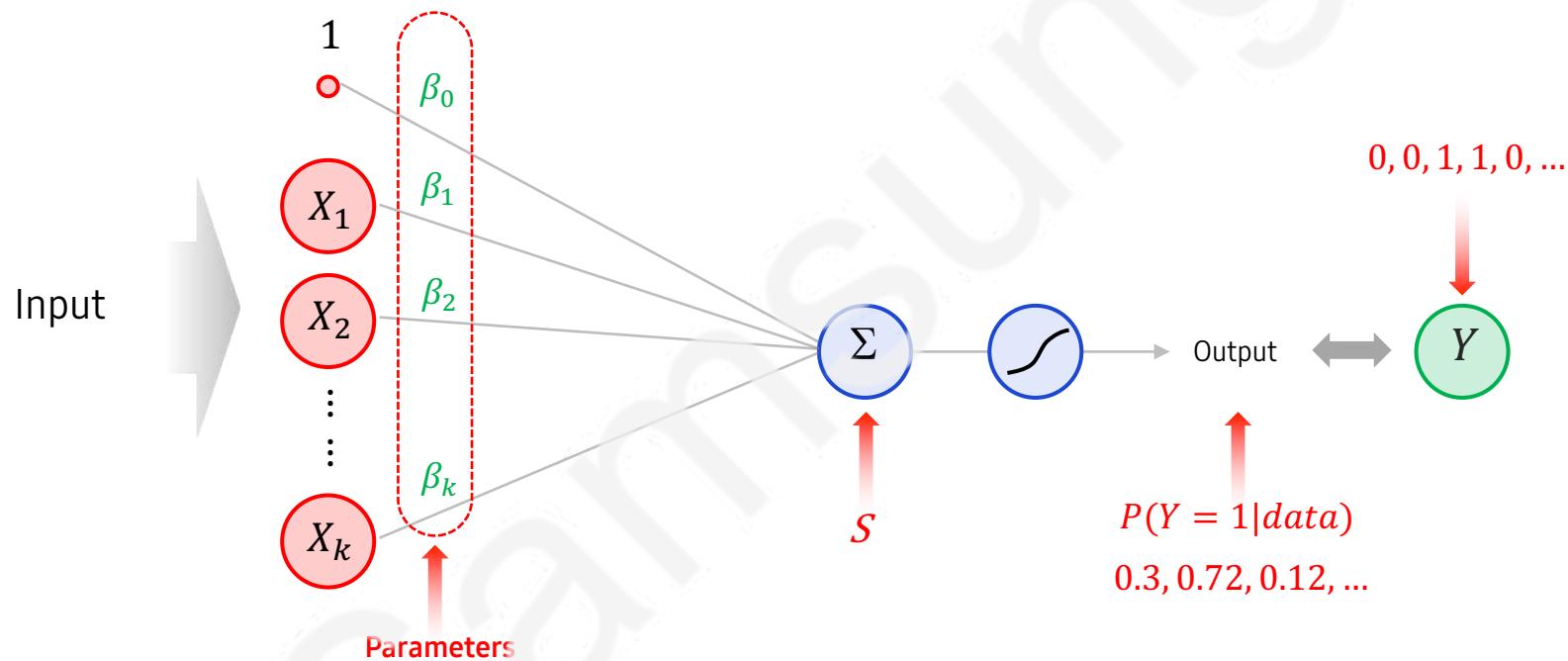


Compare the conditional probability with a cutoff.

$$P(Y' = 1 | \{x_i'\}) > \text{Cutoff} ?$$

$$\Rightarrow \hat{y} = 1 \text{ or } 0$$

| Logistic regression training and prediction (testing)



| Training by gradient descent algorithm

- ▶ We can get the parameter set $\{\beta_i\}$ by minimizing a target function $L(\beta)$.
We get to minimize the difference between the real Y and the predicted \hat{Y} .
- ▶ We can think of $L(\beta)$ as a “loss” or “cost.”
- ▶ $L(\beta)$ is a function of the parameter set $\{\beta_i\}$, which can be represented by a vector β .

| Training by gradient descent algorithm

- ▶ $L(\beta)$ is the **negative** of logarithmic likelihood defined as following:

$$L(\beta) = \sum_{i=1}^n \log(1 + e^{-y_i \beta^t x_i})$$

- ▶ In the above relation, x_i and y_i represent values given by the training dataset.
- ▶ Here, we assumed the conversion from $y_i = \{0,1\}$ to $y_i = \{-1, +1\}$.

| Training by gradient descent algorithm

- ▶ $L(\beta)$ is the **negative** of logarithmic likelihood defined as following:

$$L(\beta) = \sum_{i=1}^n \log(1 + e^{-y_i \beta^t x_i})$$

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,k} \end{bmatrix}, \quad \mathbf{x}_i^t = [1, x_{i,1}, x_{i,2}, \dots, x_{i,k}]$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \quad \beta^t = [\beta_0, \beta_1, \dots, \beta_k]$$

| Training by gradient descent algorithm

- ▶ The gradient of $L(\beta)$ is denoted as $\nabla L(\beta)$, which is also a function of the parameter set $\{\beta_i\}$ or β .
- ▶ The expression for the gradient $\nabla L(\beta)$ is:

$$\nabla L(\beta) = - \sum_{i=1}^N \frac{y_i x_i e^{-\beta^t x_i}}{1 + e^{-\beta^t x_i}}$$

- ▶ The growth of $L(\beta)$ is steepest along the direction of $\nabla L(\beta)$.
⇒ The descent of $L(\beta)$ is steepest along the direction of $-\nabla L(\beta)$.

| Training by gradient descent algorithm

- ▶ The expression for the gradient $\nabla L(\beta)$ is:

$$\nabla L(\beta) = - \sum_{i=1}^N \frac{y_i \mathbf{x}_i e^{-\beta^t \mathbf{x}_i}}{1 + e^{-\beta^t \mathbf{x}_i}}$$

- ▶ The gradient $\nabla L(\beta)$ is obtained by calculating the partial derivatives of $L(\beta)$:

$$\nabla L(\beta) = \begin{bmatrix} \frac{\partial L}{\partial \beta_0} \\ \frac{\partial L}{\partial \beta_1} \\ \vdots \\ \frac{\partial L}{\partial \beta_K} \end{bmatrix}$$

| Training by gradient descent algorithm

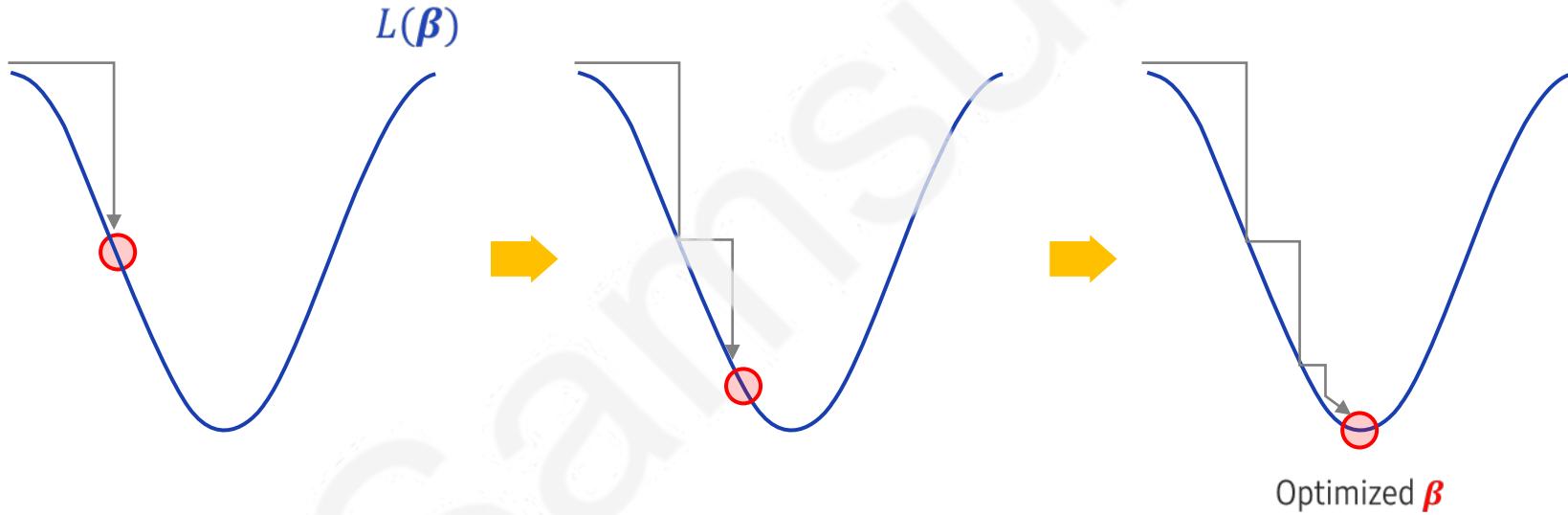
- ▶ $L(\beta)$ is minimized iteratively in small “steps” pushing β along the direction $-\nabla L(\beta)$.
 - 1) β is randomly initialized.
 - 2) Calculate the gradient $\nabla L(\beta)$.
 - 3) Update β by one step: $\beta \leftarrow \beta - \eta \nabla L(\beta)$.

Convergence speed is controlled by the “Learning rate” η .

- 4) Repeat from step 2) a fixed number of times (epochs).

| Training by gradient descent algorithm

- ▶ $L(\beta)$ is minimized iteratively in small “steps” pushing β along the direction $-\nabla L(\beta)$.



| Training by gradient descent algorithm

- ▶ The gradient function in Python

```
In [1]: def sigmoid(x):  
    s=1.0/(1.0 + np.exp(-x))  
    return s  
def gradient(X,Y,beta):  
    z = np.dot(X.beta.T)*Y  
    ds = -Y*(1-sigmoid(z))*X  
    return ds.sum(axis=0)
```

| Training by gradient descent algorithm

- ▶ The gradient function in Python

```
In [2]: def train(self, input_X, input_Y, n_epochs):
    ones_column = np.ones((input_X.shape[0], 1))
    X = np.concatenate((ones_column, input_X), axis=1)
    Y = (2 * input_Y - 1).reshape(-1, 1)
    for n in range(n_epochs):
        self.beta = self.beta - self.rate * gradient(X, Y, self.beta)
    return self.beta
```

Unit 3.

Application of Supervised Learning Model for Classification

- | 3.1. Training and Testing in Machine Learning
- | 3.2. Logistic Regression Basics
- | **3.3. Logistic Regression Performance Metrics**

Logistic Regression Performance Metrics

| Confusion matrix

- ▶ In the classification machine learning method, the most common method for evaluating the analysis model's result is the metric calculation, including classification accuracy, using a confusion matrix.
- ▶ The confusion matrix refers to the matrix that makes a crosstable of the predicted classification category from the analysis model and the actual classified category of data.

3.3. Logistic Regression Performance Metrics

UNIT 03

| Confusion matrix

		Predicted categorical value	
		Y	N
Actual categorical value	Y	O (TP: True Positive)	X (FN: False Negative)
	N	X (FP: False Positive)	O (TN: True Negative)

| Confusion matrix

- ▶ Confusion matrix can be made with a 2X2 crosstable, along with 3X3 and higher crosstables. For convenience, we will only use a 2X2 confusion matrix in this chapter.
- ▶ In the confusion matrix from the previous slide, the diagonally placed 'O' cases means the predicted and actual categorical values are the same. In other words, the classification machine learning predicted the results properly.
- ▶ On the other hand, if the predicted and actual categorical values differ, the machine learning model has made incorrect predictions.
- ▶ The categories that the analysis is mostly interested in are positive categories; the others are called negative categories. Depending on the accuracy of prediction (true or false) regarding positive and negative categories, the accurate classification of interested categories is called **TP (True Positive)**. The accurate classification of uninterested categories is called **TN (True Negative)**.
- ▶ The inaccurate classification of uninterested categories into interested categories is called **FP (False Positive)**. The inaccurate classification of interested categories into uninterested categories is called **FN (False Negative)**.
- ▶ There are various metrics based on different combinations of TP, TN, FP, and FN of the confusion matrix for evaluating the analysis result of classification machine learning methods.

Metric

- ▶ Major metrics calculated from the confusion matrix include accuracy, error rate = 1-accuracy, sensitivity (also referred to as recall, hit ratio, TP rate, etc.), specificity, FP rate, precision, and others. Among those, accuracy, sensitivity, and precision are the most frequently used metrics.
- ▶ Also, there are F-Measure (or F1-Score) that combines sensitivity, precision, and Kappa Statistics, where the predicted and actual values of the analysis model are exactly the same. The calculation formulas and definitions of various metrics are in the following table.

3.3. Logistic Regression Performance Metrics

UNIT 03

Metric

Metric	Calculation formula	Definition
accuracy	$(TP+TN) / (TP+TN+FP+FN)$	Ratio of accurate prediction of actual classification category (Ratio of TP and TN from the entire prediction)
error rate	$(FP+FN) / (TP+TN+FP+FN)$	Ratio of inaccurate prediction of actual classification category (Identical to 1-accuracy)
sensitivity = TP Rate	$(TP) / (TP+FN)$	Ratio of accurate prediction to 'positive' from actual 'positive' categories (True Positive – also referred to as Recall, Hit Ratio, and TP Rate)
specificity	$(TN) / (TN+FP)$	Ratio of accurate prediction to 'negative' from actual 'negative' categories (True Negative)
FP Rate	$(FP) / (TN+FP)$	Ratio of inaccurate prediction to 'positive' from actual 'negative' categories = 1-specificity
precision	$(TP) / (TP+FP)$	Ratio of actual 'True Positive' from the ratio of predicted 'positive'
F-Measure (F1-Score)	$\frac{\{2 * (precision) * (recall)\}}{(precision + recall)} = \frac{(2 * TP)}{2 * TP + FP + FN}$	Ranged between 0~1. It is the harmonic mean between precision and sensitivity (recall). If both precision and sensitivity are high, f-Measure also tends to have a larger value.
Kappa Statistic	$\frac{\{Pr(a) - Pr(e)\}}{(1 - Pr(e))}$	The value after eliminating coincidental agreement between predicted and actual values of the model (Ranged between 0~1. When the value is closer to 1, the predicted and actual values of the model accurately coincide. The values do not coincide when the value gets closer to 0.)

Metric

- ▶ Among the metrics from the previous slide, sensitivity signifies how well the actual ‘positive’ category is predicted ‘positive.’ The precision is the index showing the ratio of actual ‘positive’ from the predicted ‘positive’ categories. Thus, they are metrics that directly explain how well the classification machine learning analysis model classifies interested categorical values of objective variables.
- ▶ Sensitivity and precision are the most significant and frequently used metrics for classification machine learning results in real life.

| Confusion matrix

Ex		Actual 0	Actual 1
Predicted 0	120	5	
Predicted 1	15	20	

- ▶ Confusion matrix is a contingency table that counts the frequencies of the actual vs. predicted.

| Confusion matrix: Accuracy

Ex		Actual 0	Actual 1
Predicted 0	120	5	
Predicted 1	15	20	

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- ▶ Accuracy is the ratio between the diagonal sum and the total sum.

| Confusion matrix: Sensitivity

Ex		Actual 0	Actual 1
Predicted 0	120	5	
Predicted 1	15	20	

$$\text{Sensitivity} = \frac{\text{Number of correctly predicted 1s}}{\text{Total number of 1s}}$$

| Confusion matrix: Specificity

Ex		Actual 0	Actual 1
Predicted 0	120	5	
Predicted 1	15	20	

$$\text{Specificity} = \frac{\text{Number of correctly predicted 0s}}{\text{Total number of 0s}}$$

| Confusion matrix: Precision

Ex		Actual 0	Actual 1
Predicted 0		120	5
Predicted 1		15	20

$$\text{Precision} = \frac{\text{Number of correctly predicted 1s}}{\text{Total number of 1s}}$$

Note

- ▶ Accuracy alone is not sufficient for testing.
- Ex** If frauds constitute only 1% of all transactions, the accuracy of a fraud detection system (FDS) that predicts non-fraud in all transactions would be quite high at 99%.
However, such FDS would be useless because it misses the 1% that really matters.
- ▶ We should also consider metrics other than just the accuracy.

| Terminology

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$\text{Sensitivity} = \frac{\text{Number of correctly predicted 1s}}{\text{Total number of 1s}}$$

$$\text{Specificity} = \frac{\text{Number of correctly predicted 0s}}{\text{Total number of 0s}}$$

$$\text{Precision} = \frac{\text{Number of correctly predicted 1s}}{\text{Total number of 1s}}$$

$$\text{Cohen's kappa } \kappa = \frac{\text{Accuracy} - p_e}{1 - p_e}$$

where p_e is the probability of correct prediction by chance.

| Terminology

True Positive Rate = Sensitivity

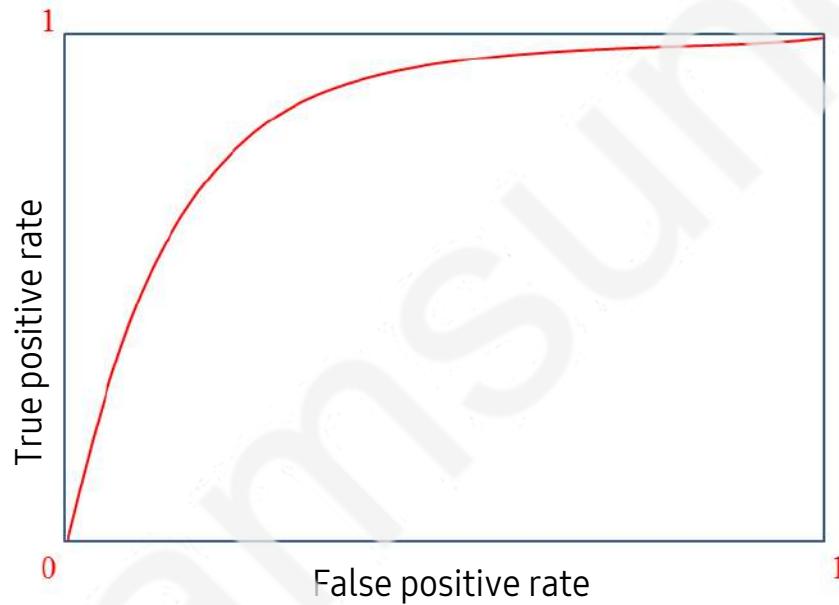
True Negative Rate = Specificity

False Positive Rate = $\frac{\text{Number of actual 0s that are incorrectly predicted as 1}}{\text{Total number of 0s}}$ = 1-Specificity

False Negative Rate = $\frac{\text{Number of actual 1s that are incorrectly predicted as 0}}{\text{Total number of 1s}}$ = 1-Sensitivity

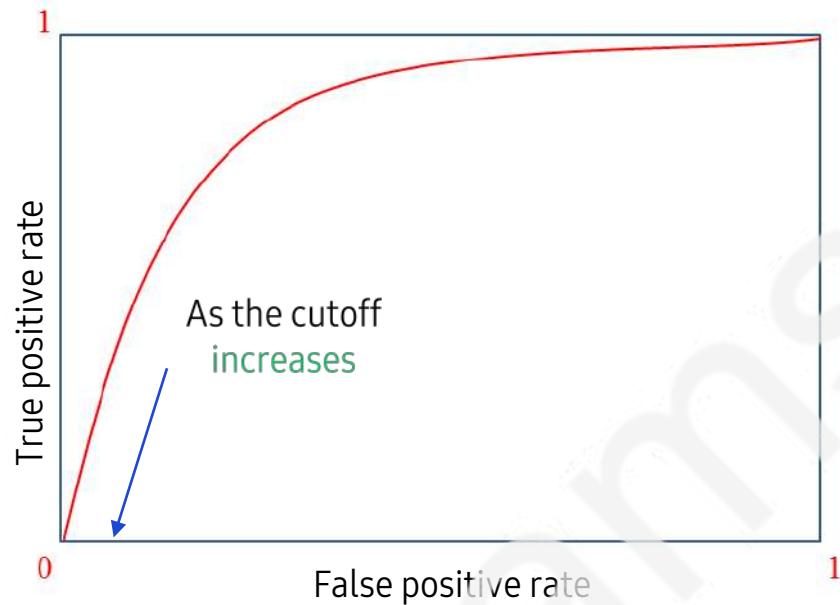
Positive Predicted Value = Precision

| ROC curve



- ▶ ROC curve is a parametric plot with respect to the *cutoff* probability.

| ROC curve

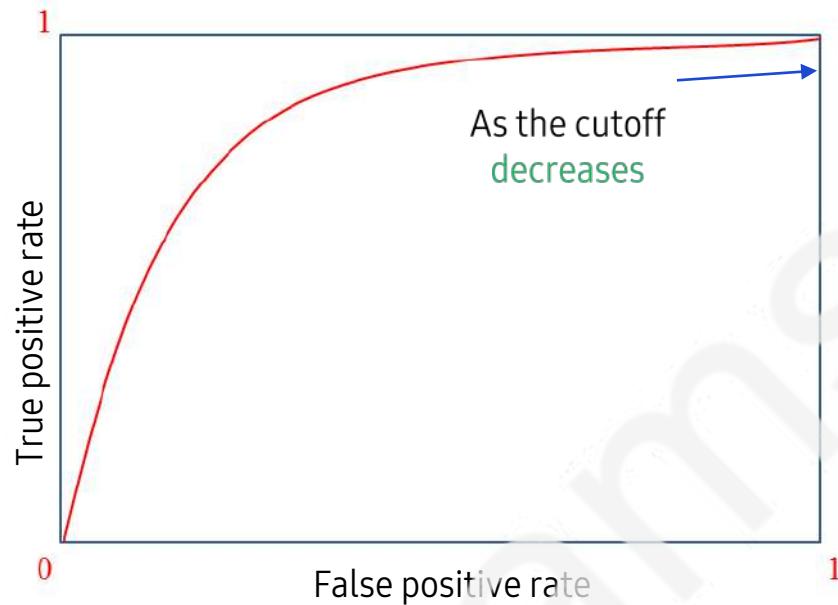


As the cutoff **increases** (closer to 1)

Performance Metric	Increase/ Decrease
True Positive Rate (Sensitivity)	↓
Specificity	↑
False Positive Rate (1-Specificity)	↓
Precision	↑

- ▶ ROC curve is a parametric plot with respect to the *cutoff* probability.

| ROC curve

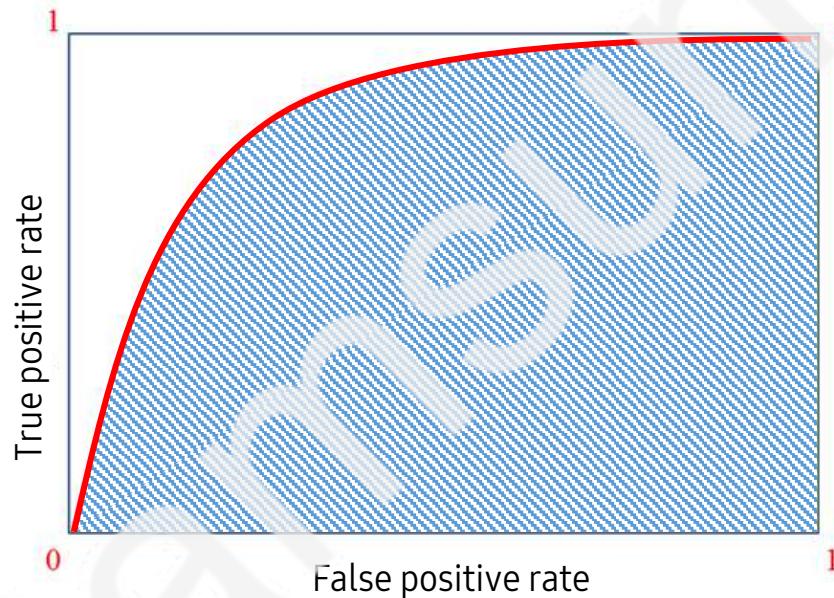


As the cutoff decreases (closer to 0)

Performance metric	Increase/ Decrease
True Positive Rate (Sensitivity)	↑
Specificity	↓
False Positive Rate (1-Specificity)	↑
Precision	↓

- ▶ ROC curve is a parametric plot with respect to the *cutoff* probability.

| ROC curve



- ▶ AUC stands for **Area Under the Curve**.
- ▶ AUC closer to 1 means good overall performance.

| Interpretation using Bayes' theorem

Ex There are 100 observations. In 6 cases, the actual response is 1.

In the rest of the 94 cases, the actual response is 0. It is known that for a given logistic regression model, the sensitivity = 0.92 and the specificity = 0.90.

For a new observation (only explanatory variables), this model predicts 1 as the response.

What is the probability of this prediction being correct?

- a) We have $P(1) = 0.06$, $P(0) = 0.94$.

$$P(\text{Predicted } 1 \mid \text{Actual } 1) = 0.92 \Leftrightarrow \text{"Sensitivity"}$$

$$P(\text{Predicted } 0 \mid \text{Actual } 0) = 0.90 \Leftrightarrow \text{"Specificity"}$$

$$\text{We can also derive } P(\text{Predicted } 1 \mid \text{Actual } 0) = 1 - P(\text{Predicted } 0 \mid \text{Actual } 0) = 0.1$$

| Interpretation using Bayes' theorem

Ex There are 100 observations. In 6 cases, the actual response is 1.

In the rest of the 94 cases, the actual response is 0. It is known that for a given logistic regression model, the sensitivity = 0.92 and the specificity = 0.90.

For a new observation (only explanatory variables), this model predicts 1 as the response.

What is the probability of this prediction being correct?

b) The answer we are seeking is given by

$$\begin{aligned} P(\text{Actual 1|Predicted 1}) &= \frac{P(\text{Predicted 1|Actual 1})P(1)}{P(\text{Predicted 1|Actual 1})P(1) + P(\text{Predicted 1|Actual 0})P(0)} \\ &= \frac{0.92 \times 0.06}{0.92 \times 0.06 + 0.1 \times 0.94} \cong 0.37 \end{aligned}$$

Coding Exercise #0307



Follow practice steps on 'ex_0307.ipynb' file.

Unit 4.

Decision Tree

| 4.1. Tree Algorithm

Overview of Decision Tree

| Definition

- ▶ A classification model that analyzes data collected from the past and expresses the patterns found (characteristics of each category) as a combination of features

| Purpose

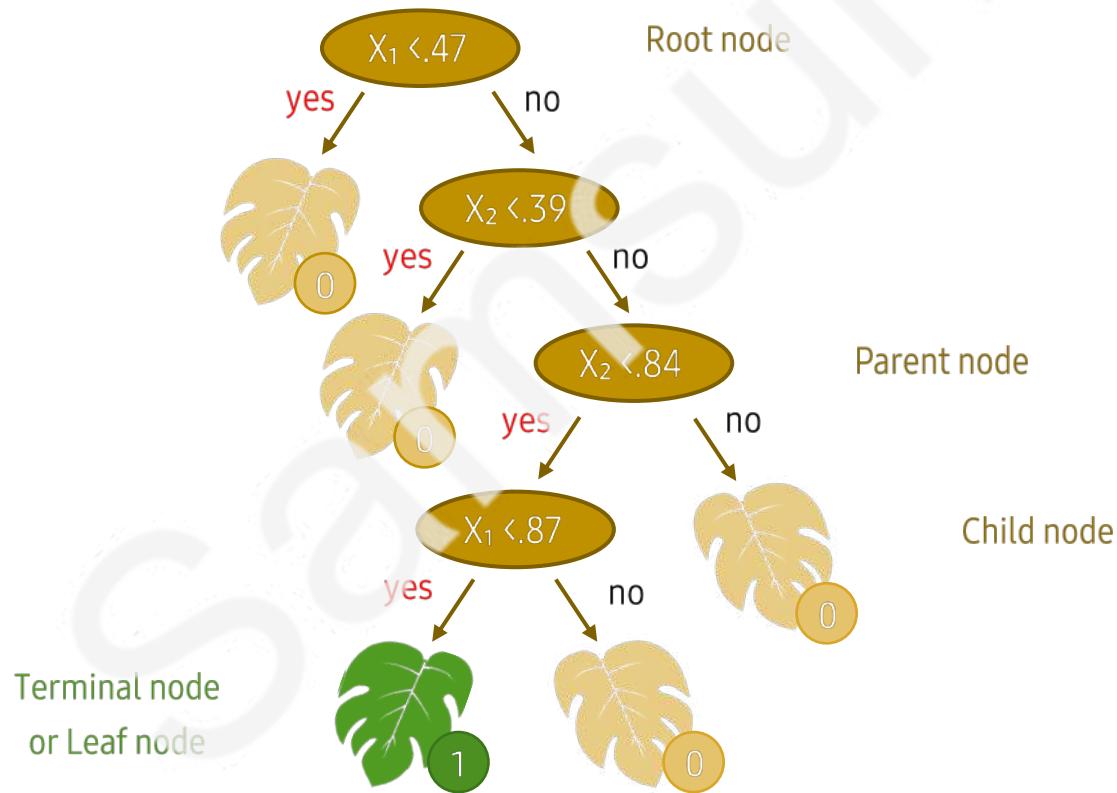
- ▶ Classification of unseen data and prediction of categorical values
- ▶ Extraction of generalized knowledge in a tree structure from the data

| Classification depending on the objective variable types

- ▶ Categorical variable: Classification Tree
- ▶ Continuous variable: Regression Tree

Composition

- Node, Branch, Depth



How to construct a decision tree model

Construction of a decision tree

Construction of a decision tree

Making a decision tree by designating an appropriate split criterion and stopping rules according to the purpose and data structure of analysis

Branching

Removing branches that have a high risk of error rate or inappropriate rules

Validity evaluation

Evaluation of the decision tree through cross-validation using the gain chart, risk chart, or test data

Interpretation and prediction

Interpretation of the decision tree and setting a prediction model

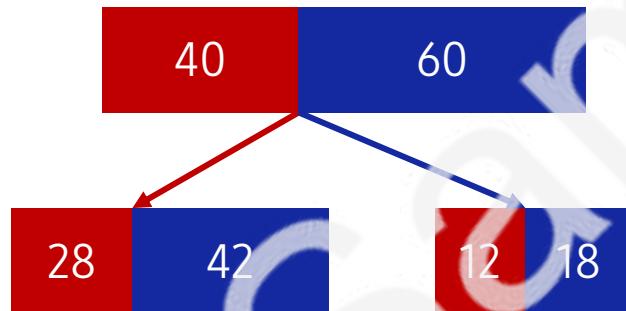
Split criterion of the decision tree

Analysis process of a decision tree

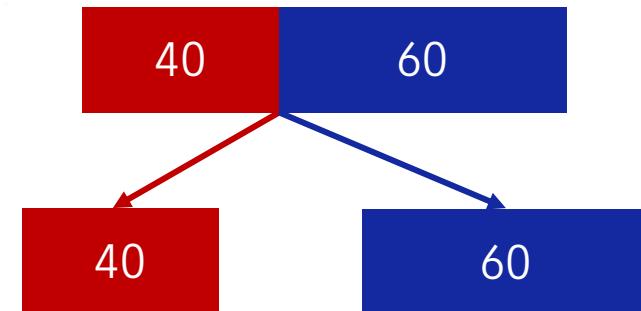
- ▶ Repetitive splitting: Repetitive splitting of the dimensional space of independent variables using training data
- ▶ Branching: Branching with evaluation data

Split criterion

- ▶ Create the classification tree so that the purity of the child node is greater than that of the parent node.



No Improvement

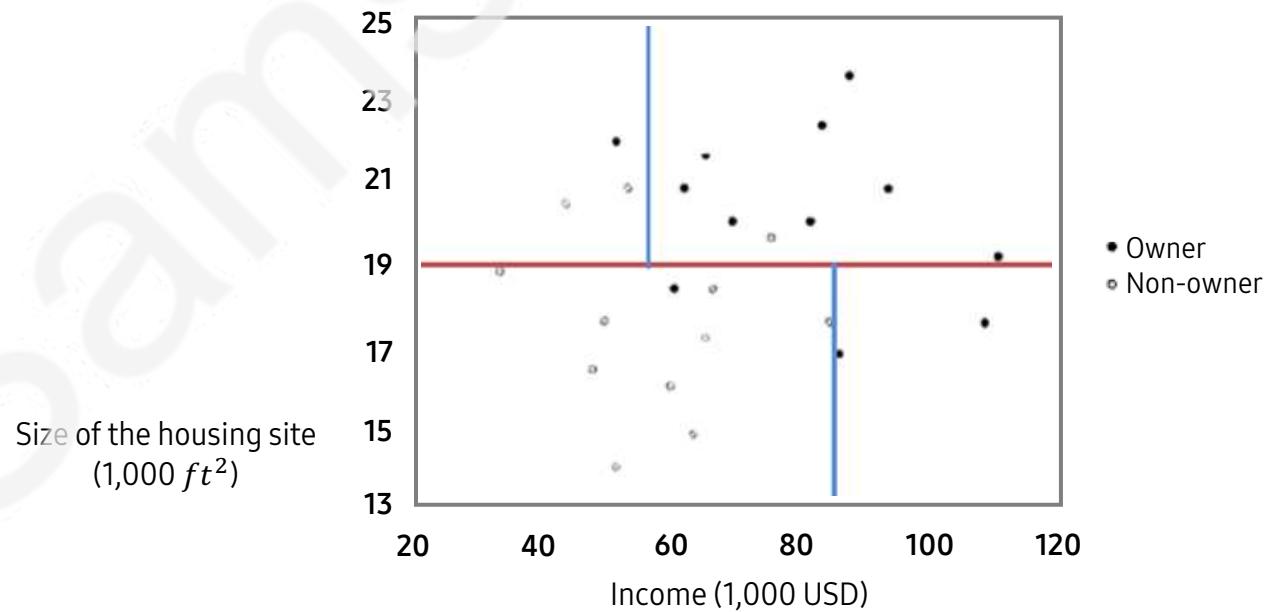


Perfect Split

Repetitive splitting process

Purpose

- ▶ Divide the entire space into rectangles and make each as pure or homogeneous as possible.
- ▶ Definition of 'pure':
 - Divide the area into pure or homogenous rectangular spaces as much as possible
 - All variables in the final rectangle belong to the same group.



Repetitive splitting process

- 1) Select x_i , one of the variables, and the x_i value (s_i) as a split criterion is designated to split the p -dimension space into two.
- 2) $x_i \rightarrow \{x_i \leq s_i\} \cup \{x_i > s_i\}$
- 3) Select a variable again and split it in the same way.
- 4) Repeat the process until it reaches desired purity.

Split criterion

- | Discrete objective variables
 - ▶ Chi squared statistic – p-value: Creates child nodes with a predictor variable with the lowest p-value and the optimal partitioning
 - ▶ Gini index: Selects child nodes with a predictor variable that reduces the Gini index and the optimal partitioning
 - ▶ Entropy measure: Creates child nodes with a predictor variable with the lowest entropy measure and the optimal partitioning

Split criterion

- I Continuous objective variables
 - ▶ F statistic in ANOVA: Creates child nodes with a predictor variable with the lowest p-value and the optimal partitioning
 - ▶ Variance reduction: Creates child nodes with the optimal partitioning that maximizes variance reduction

Selection of algorithms and classification variables

	Discrete objective variables	Continuous objective variables
CHAID (multi space partitioning)	Chi squared statistic	ANOVA F statistic
CART (binary space partitioning)	Gini index	Variance reduction
C4.5	Entropy measure	

Impurity measure

Gini index

- ▶ Selects child nodes with a predictor variable that reduces the Gini index and the optimal partitioning
- ▶ If the T data set is split into k categories and the category performance ratios are p₁, ..., p_k, it is expressed as the following equation.

$$Gini(T) = 1 - \sum_{l=1}^k p_l^2$$

high impurity(diversity), low purity



$$GI = 1 - (3/8)^2 - (3/8)^2 - (1/8)^2 - (3/8)^2 = .69$$

low impurity(diversity), high purity



$$GI = 1 - (6/7)^2 - (1/7)^2 = .24$$

Entropy measure

- ▶ In thermodynamics, entropy measures the degree of disorder.
- ▶ Creates child nodes with a predictor variable with the lowest entropy measure and the optimal partitioning.
- ▶ If the T data set is split into k categories and the category performance ratios are p₁, ..., p_k, it is expressed as the following equation.

$$Entropy(T) = - \sum_{l=1}^k p_l \log_2 p_l$$

Ex If 4 categories consist of ratios of 0.25, 0.25, 0.25, 0.25 (T0):

$$Entropy(T_0) = -(0.25 \log_2 0.25) * 4 = 1.39$$

Ex If 4 categories consists of ratios of 0.5, 0.25, 0.25, 0 (T1):

$$Entropy(T_1) = -(0.5 \log_2 0.5 + 0.25 \log_2 0.25 + 0.25 \log_2 0.25) = 1.04$$

Stopping criteria

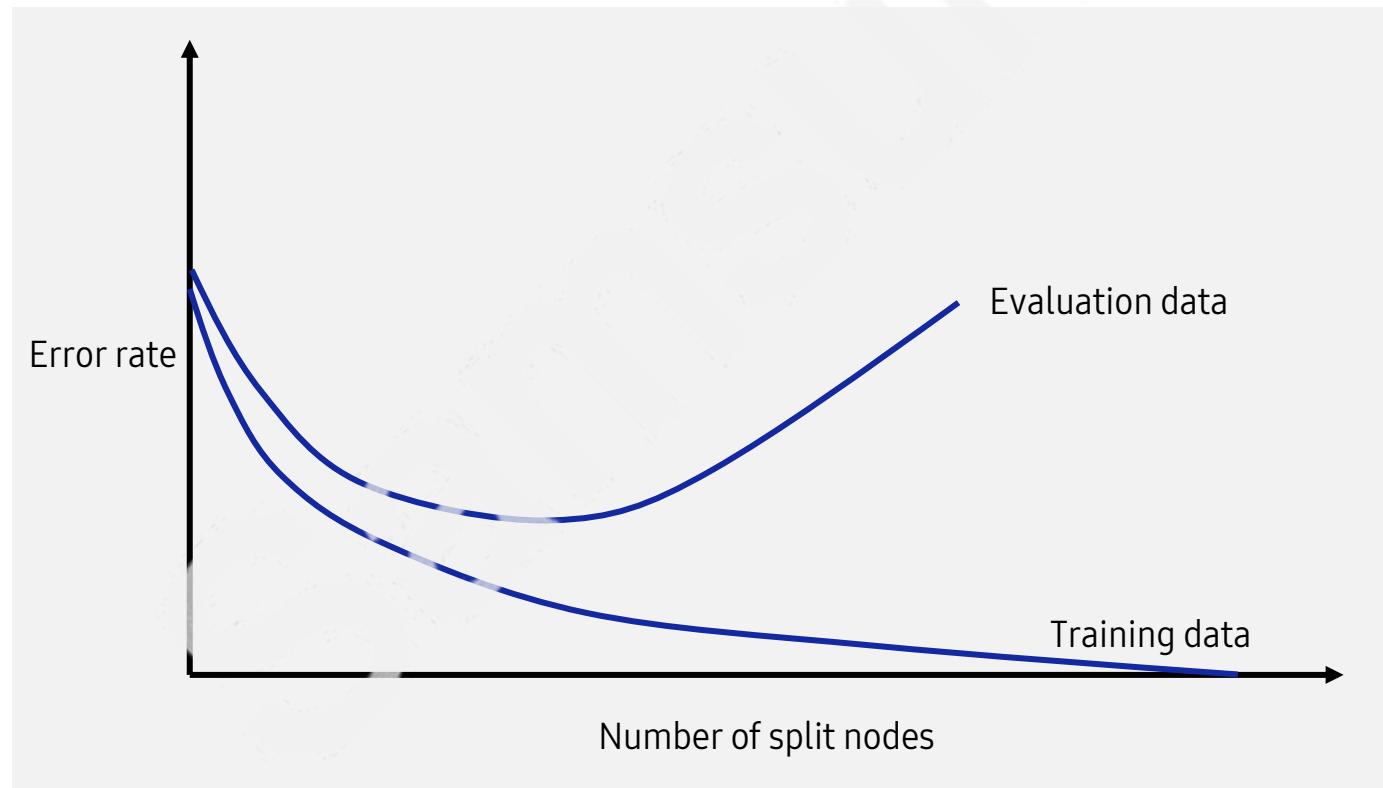
- | A rule to designate the current node as a terminal node without further splitting
 - ▶ Designates the depth of the decision tree
 - ▶ Designates the minimum number of records in the terminal node

Branching criteria

- | Application of test data
 - ▶ Application of the test data to the constructed model
 - ▶ Reviewing the predictive value of the constructed model through test data
 - ▶ Removing the branches that have a high risk of error rate or inappropriate rule of inference
- | By an expert
 - ▶ An expert reviewing the validity of rules suggested in the constructed model
 - ▶ Removing rules without validity

Overfitting problem

| Overfitting problem graph



Pros

- ▶ Creation of understandable rules (can be expressed with SQL)
- ▶ Useful in classification prediction
- ▶ Able to work with both continuous and discrete variables
- ▶ Shows a more relatively significant variable

Cons

- ▶ Not suitable to predict continuous variable values
- ▶ Unable to perform time series analysis
- ▶ Not stable

Tree Algorithm

| Pros

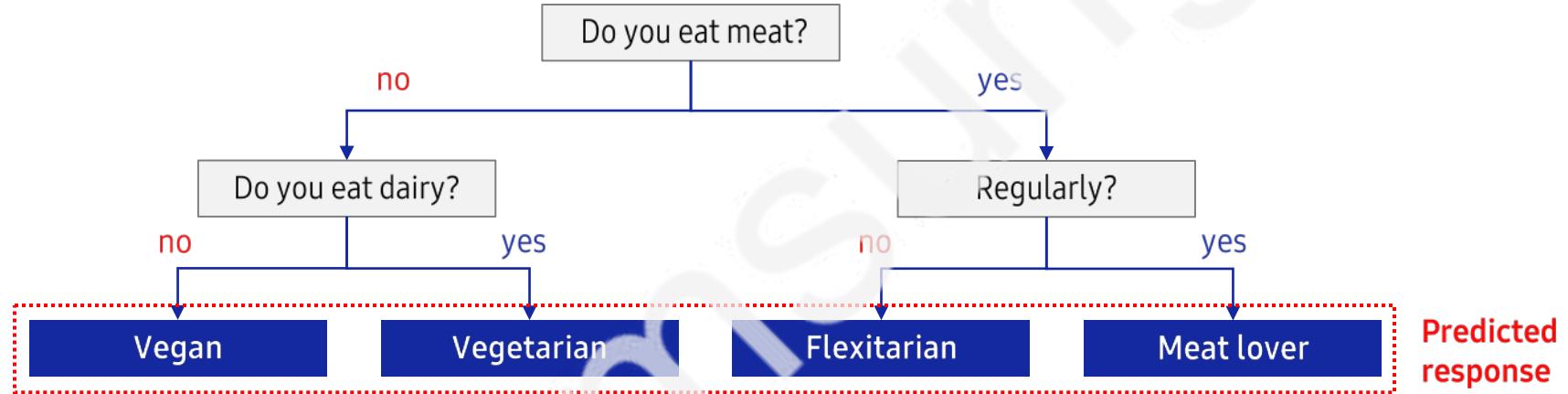
- ▶ Intuitive and easy to understand
- ▶ No assumptions about the variables
- ▶ No need to scale or normalize data
- ▶ Not that sensitive to the outliers

| Cons

- ▶ Not that powerful in the most basic form
- ▶ Prone to overfitting. Thus, “pruning” is often required.

Classification Tree

Ex



- ▶ Training step creates an inverted tree structure as above.
- ▶ Conditions are evaluated at the nodes and then branch out.
- ▶ Each leaf node corresponds to a **region** in the configurational space.

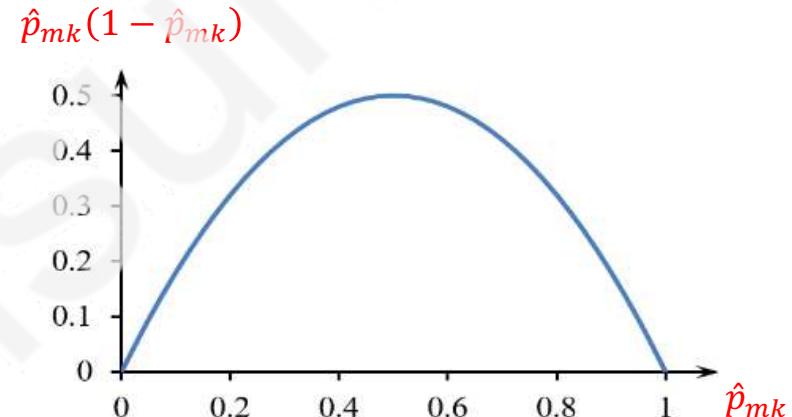
Classification Tree

- The tree structure is trained by minimizing the Gini impurity (or entropy).

$$G_m = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^K \hat{p}_{mk}^2$$

or

$$\text{Entropy}_m = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$$



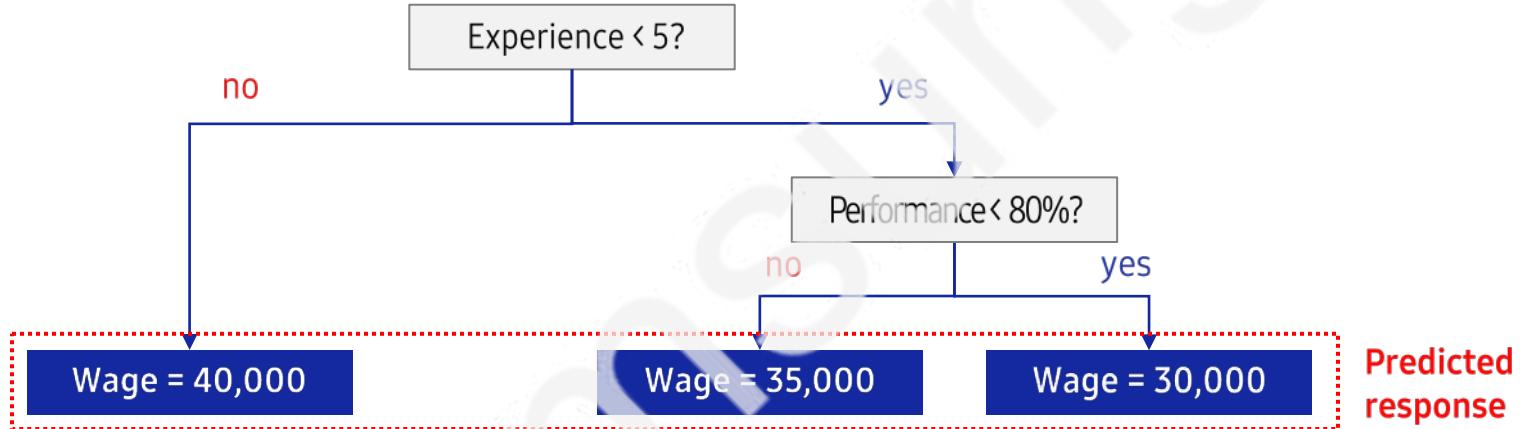
- G_m is the Gini impurity in the leaf node m .
 - Entropy_m is the entropy in the leaf node m .
 - Here, \hat{p}_{mk} is the proportion of the class k in the leaf node m .
 - K is the total number of possible classes.
 - The class with the largest proportion is the prediction at that leaf node.
- } The smaller, the better.

| Classification Tree: Procedure

- a) Make a basic tree.
- b) Prune branches that do not provide better performance.
Pruning can be done during the cross-validation step.
- c) Predict with the optimized tree.

Regression Tree

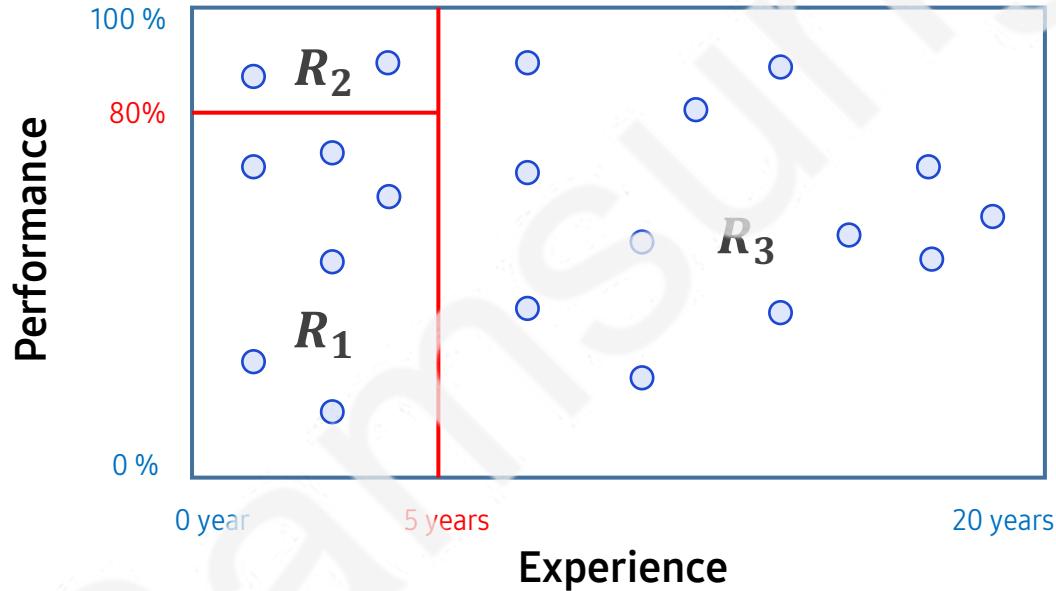
Ex



- ▶ Predicts numeric values rather than categories or classes.
- ▶ Conditions are evaluated at the nodes and then branch out.
- ▶ Each leaf node corresponds to a **region** in the configurational space.

Regression Tree

Ex



- ▶ Each leaf node corresponds to a **region** in the configurational space.

Regression Tree

- ▶ The configurational space is split into regions: $\{R_1, R_2, \dots, R_J\}$.
- ▶ The tree structure is trained by minimizing the RSS (residual sum of squares):

$$RSS = \sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

- J is the total number of leaf nodes.
- The predicted value in the j -th region \hat{y}_{R_j} is given by the average in that region.
- For the observations belonging to the region R_j , the same predicted response \hat{y}_{R_j} is assigned analogous to the classification Tree.

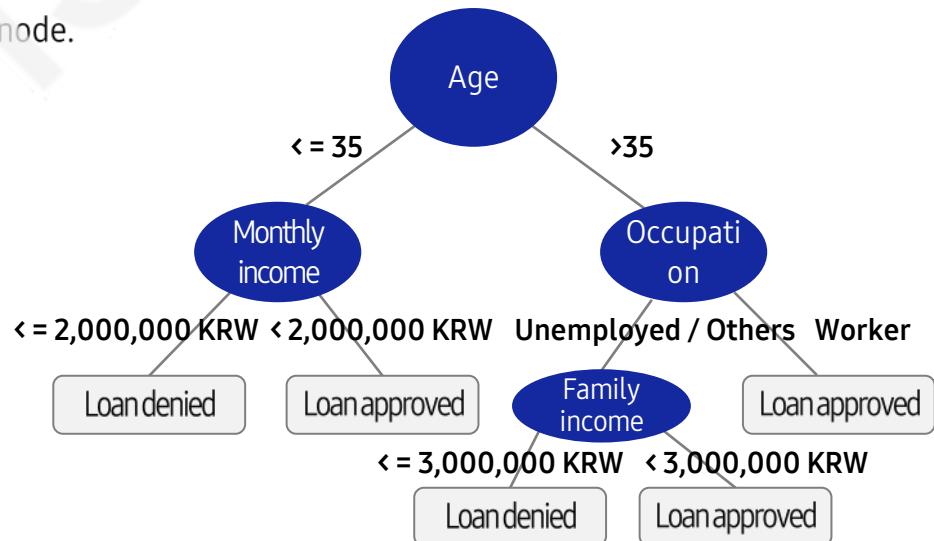
Scikit-Learn DecisionTreeClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
<code>max_depth</code>	The maximum depth of a tree
<code>min_samples_leaf</code>	The minimum number of sample points required to be at a leaf node
<code>min_samples_split</code>	The minimum number of sample points required to split an internal node
<code>max_features</code>	The number of features to consider when looking for the best split
<code>max_leaf_nodes</code>	The maximum number of leaf nodes in the tree

- ▶ Need to be tuned for optimized performance.
- ▶ More information can be found at:
Classifier: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>

Decision Tree

- ▶ Decision tree refers to a modeling technique as the shape of a tree branching out from root to leaf nodes. It depends on reference values of independent variables (explanatory or input variables) that affect the classification or prediction of objective variables.
- ▶ In the decision tree, each node is split in the form of if-then depending on explanatory variables' characteristics or reference values. When following the tree structure, it is possible to easily understand how the attribute value of data is classified into the category.
- ▶ The figure provided below is a typical form of the decision tree. From the example, 'age' is the root. It can be inferred that 'age' is the most significant variable when deciding the loan approval.
- ▶ The squared shape node at the end of each branch is the leaf node.



Split Criteria of Decision Tree

- ▶ Decision tree gives a question to each node and separates data by branching according to the response.
- ▶ To evaluate how well the data is separated, a specific criterion is required. In general, impurity is the evaluation criterion. The impurity becomes higher as various classifications are mixed in the node; it is the lowest when there's only one classification.
- ▶ Thus, if each node's impurity is low after node splitting, we can say that the decision tree is well classified.
- ▶ Impurity measures include Gini impurity and entropy.

$$\text{Gini Coff.} = 1 - \sum_{i=1}^K p_i^2, \quad \text{Entropy Coff.} = -\sum_{i=1}^K p_i \log_2 p_i$$

- ▶ The impurity index is 0 when $p_i = 0$ or $p_i = 1$, and it gets the largest when $p_i = \frac{1}{2}$, thus making a parabola. In other words, the impurity index is the lowest when there's a certain classification in the node or the node is completely free from any classification. In contrast, the impurity becomes the largest when many classifications are found in the same node.

Coding Exercise #0308



Follow practice steps on 'ex_0308.ipynb' file.

Unit 5.

Naïve Bayes Algorithm

I 5.1. Naïve Bayes Algorithm

Naïve Bayes Algorithm

| About Naïve Bayes algorithm

- ▶ It is a straightforward application of Bayes' theorem.

| Pros

- ▶ Intuitive and simple
- ▶ Not that sensitive to the noise and outliers
- ▶ Fast

| Cons

- ▶ Assumes that the features are independent, which may not be strictly true
- ▶ Not among the best-performing algorithms

| About Naïve Bayes algorithm

- ▶ Bayes' theorem:

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

- ▶ Now we take $A=Class$ and $B=Data$, then:

$$P_{post}(Class) = P(Class|Data) = \frac{P(Data|Class)P_{prior}(Class)}{P(Data)}$$

- ▶ We are more interested in comparing relative probabilities:

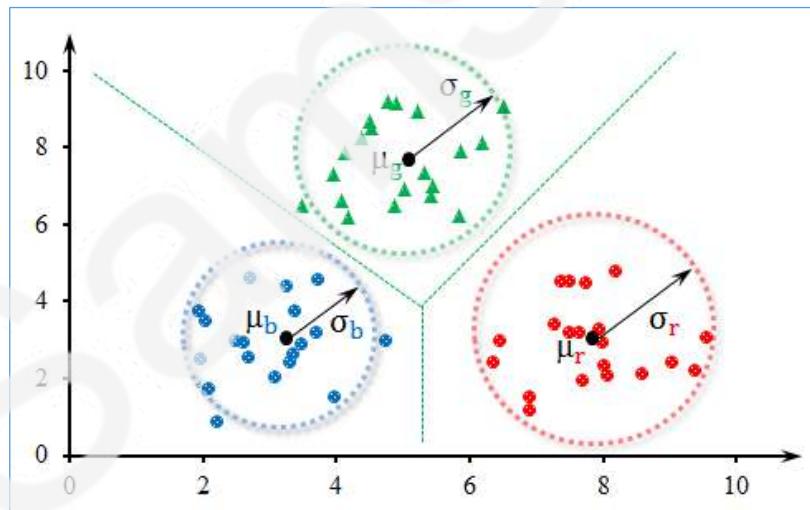
$$P_{post}(Class) \propto P(Data|Class)P_{prior}(Class)$$

| About Naïve Bayes algorithm

- ▶ We can approximate $P(Data \mid Class)$ by a Gaussian (normal distribution):

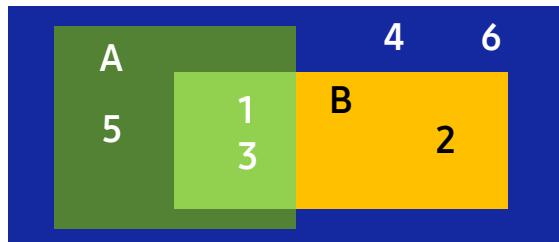
$$P_{post}(Class) \propto \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{1}{2\sigma_j^2}(x - \mu_j)^2\right) P_{prior}(Class)$$

where the parameters μ_j and σ_j are “learned” from the training data.



Random variable

- ▶ The variable whose values are unknown until the outcome
- ▶ Independent events
 - If the probability of the simultaneous occurrence of two cases is identical to the multiplication of the probabilities of each event to occur, then the two events are independent of each other.
 - $P(A \cap B) = P(A)P(B)$
- ▶ Dice



Conditional probability

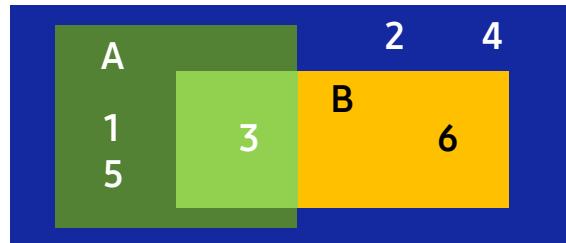
- Events of B (1,2,3) when the A events (1,3,5) occur
- 0.6666667



		Total	Event A	Event B	Probability A	Probability B
Event A	Odd number	1	1	1	0.5	0.5
Event B	Less than 3	2		2		
		3	3	3		
		4				
		5	5			
		6				

| Random variable

- ▶ The variable whose values are unknown until the outcome
- ▶ Independent events
 - Not affected
- ▶ Dice



		Total	Event A	Event B	Probability A	Probability B
Event A	Odd number	1	1		0.5	0.333
Event B	Multiple of 3	2				
		3	3	3		
		4				
		5	5	6		
		6				

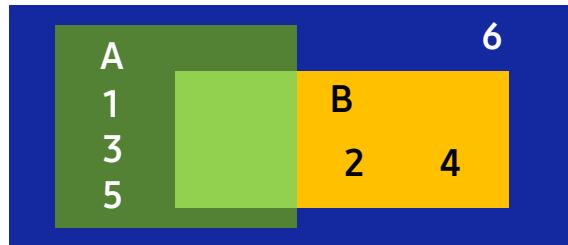


Conditional probability

- Events of B (3, 6) when the A events (1,3,5) occur
- 0.333333333

| Random variable

- ▶ The variable whose values are unknown until the outcome
- ▶ Exclusive events
 - Intersection is the null set.
- ▶ Dice



| Introduction to Bayesian statistics

1) Differentiate groups of people for 'shopping' and 'window shopping' through Bayesian interpretation.

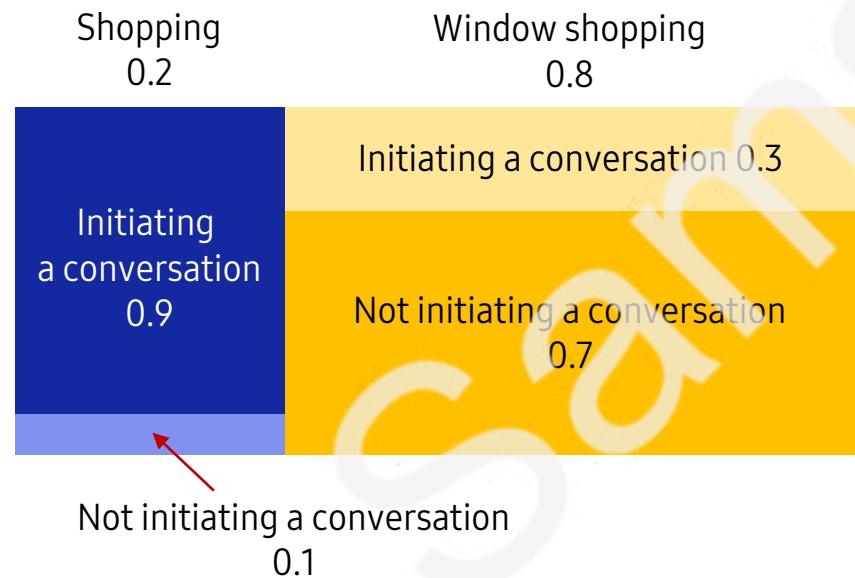
- Setting prior probability for different types
- Prior probability: Probability before getting certain information



| Introduction to Bayesian statistics

2) Set the conditional probability of 'initiating a conversation' for each type.

	Probability of initiating a conversation	Probability of not initiating a conversation	Total
Shopping	0.9	0.1	1
Window shopping	0.3	0.7	1

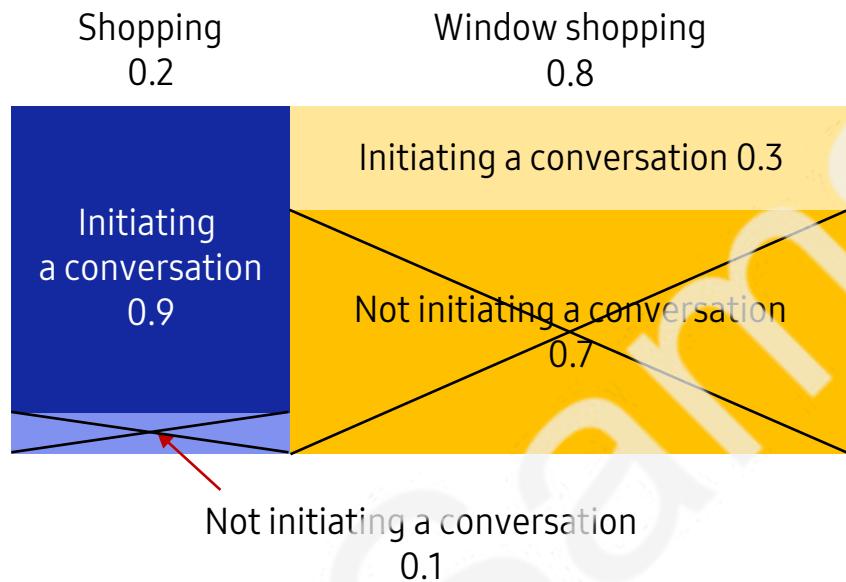


- The probability of each section is the area of each rectangle. (0.18, 0.02, 0.24, 0.56)

| Introduction to Bayesian statistics

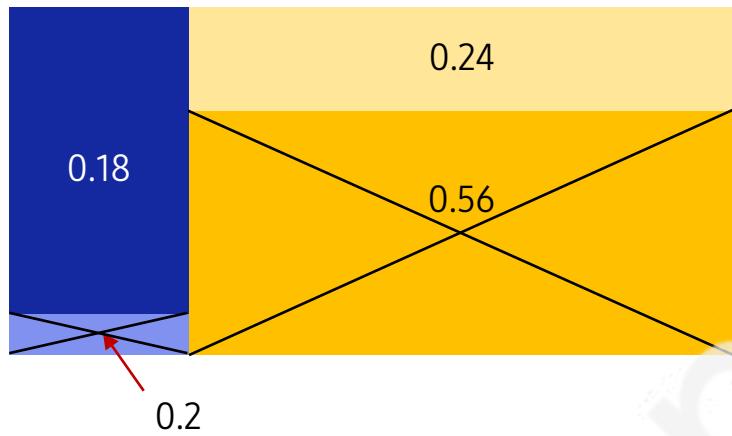
3) Remove 'impossible event' from the observations.

- The customer initiated a conversation. (additional information)



Introduction to Bayesian statistics

4) Calculate Bayesian probability of the shopping group.



	Shopping	Window shopping
Normalization	3/7	4/7
	43%	

Result
Initiating a conversation Cause
A certain group

- The result is 0.18:0.24, which is 3:4.
- Thus, it can be estimated that the probability of the customer who initiated a conversation being in the shopping group is 3/7 (43%). It is either called Bayesian statistics or posterior probability.
- Chooses if a person who initiated a conversation is in the shopping or window shopping group based on probability.

Introduction to Bayesian statistics

- ▶ The probability of a customer who initiated a conversation making a purchase is now different.
- ▶ Bayesian updating for customer type

Final summary The prior probability of being a shopping group
Observing to initiate a conversation
The posterior probability of being a shopping group

$$\begin{array}{ccc} 0.2 & \xrightarrow{\hspace{1cm}} & \\ 3/7 & \longrightarrow & 0.428571 \end{array}$$

- ▶ The customer wouldn't be in the shopping group for sure, but the probability is doubled.
- ▶ **Bayesian estimation** is '**performing Bayesian updating to the posterior probability based on the behavioral observation of the prior probability.**'
- ▶ Such an estimation method is called Bayesian statistics.
- ▶ **Bayes' theorem**
 - **Obtaining the posterior probability based on the prior probability**

Coding Exercise #0309



Follow practice steps on 'ex_0309.ipynb' file.

Unit 6.

KNN Algorithm

| 6.1. KNN Algorithm

KNN Algorithm

| About KNN (K-Nearest Neighbors)

- ▶ One of the simplest algorithms
- ▶ Prediction is based on the k nearest neighboring points.
- ▶ There are classification and regression variants.
 - Classification: prediction decided by the majority class of the nearest neighbors.
 - Regression: prediction given by the average of the nearest neighbors.

About KNN (K-Nearest Neighbors)

- ▶ KNN is a classification and prediction technique for a data set with the unknown category of the objective variable. It designates the most similar category of the surrounding data set.
- ▶ For KNN, specific criteria are required for measuring the ‘analogy’ between the certain data set and the surrounding one and how many data sets will be used for the final classification of the objective variable category.

1) Analogy measurement

- There are many ways to measure the analogy between data. In general, analogy calculation is done by inverting the squared Euclidean distance between two points or using the Pearson correlation coefficient. If the points are discrete variables, use the Jaccard coefficient.

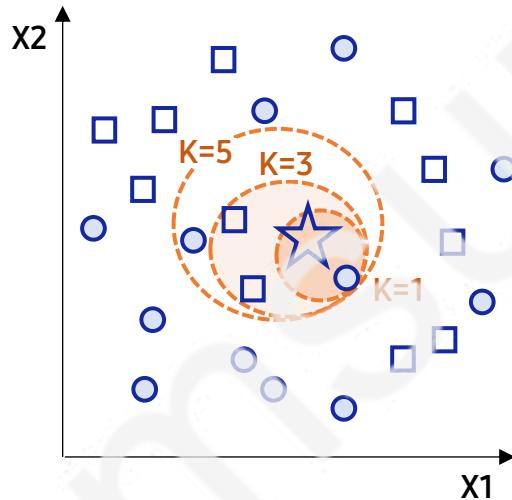
2) Objective variable classification criteria

- K in the KNN refers to the number of surrounding data points being used for classifying objective variables of the specific data point after measuring the analogy between the specific data point and other surrounding data sets.

Ex Take movie recommendations as an example. It recommends a movie that is similar to the customer’s favorite movie and that the customer hasn’t watched yet. When considering only one movie that is the most similar to the customer’s favorite, it is ‘1-nearest neighbor.’ When considering three similar movies, it would be ‘3-nearest neighbor.’

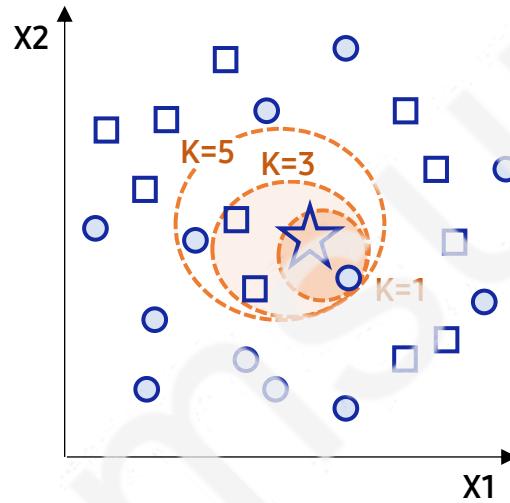
- Likewise, K-nearest neighbor is a technique to determine a new category according to the principle of majority rule. It finds k surrounding data points similar to the specific data point and considers the classification category of the specific data point.

| About KNN (K-Nearest Neighbors)



- ▶ The figure above conceptualizes the change in objective variables depending on the K value setting in the K-nearest neighbor method. The '★' point is the data value that needs to be classified; the points shaped in a square and circle are other data points present around.
- ▶ When setting K=1, since the closest data point to the star is a circle, the objective variable of '★' will be classified as 'o.' On the other hand, if K=3, three points closest to '★' will be considered, including two squares and one circle. In this case, '★' will be classified as '□.' Also, if K=5, since there are more circles around '★,' it will be again classified as 'o.'

| About KNN (K-Nearest Neighbors)



- ▶ Thus, different K values significantly change the K-nearest neighbor's predicted result of the objective variable category. This is why setting an appropriate K value is important in the K-nearest neighbor method.
- ▶ However, there are no clear theoretical or statistical standards for an appropriate K value. Different K values are usually set for repeated tests, and the final K value is set once it shows the optimal classification performance. Generally, a random initial K value is designated between K=3 and K=9, which is then used to test the classification performance with training and evaluation data to select the optimal K value.

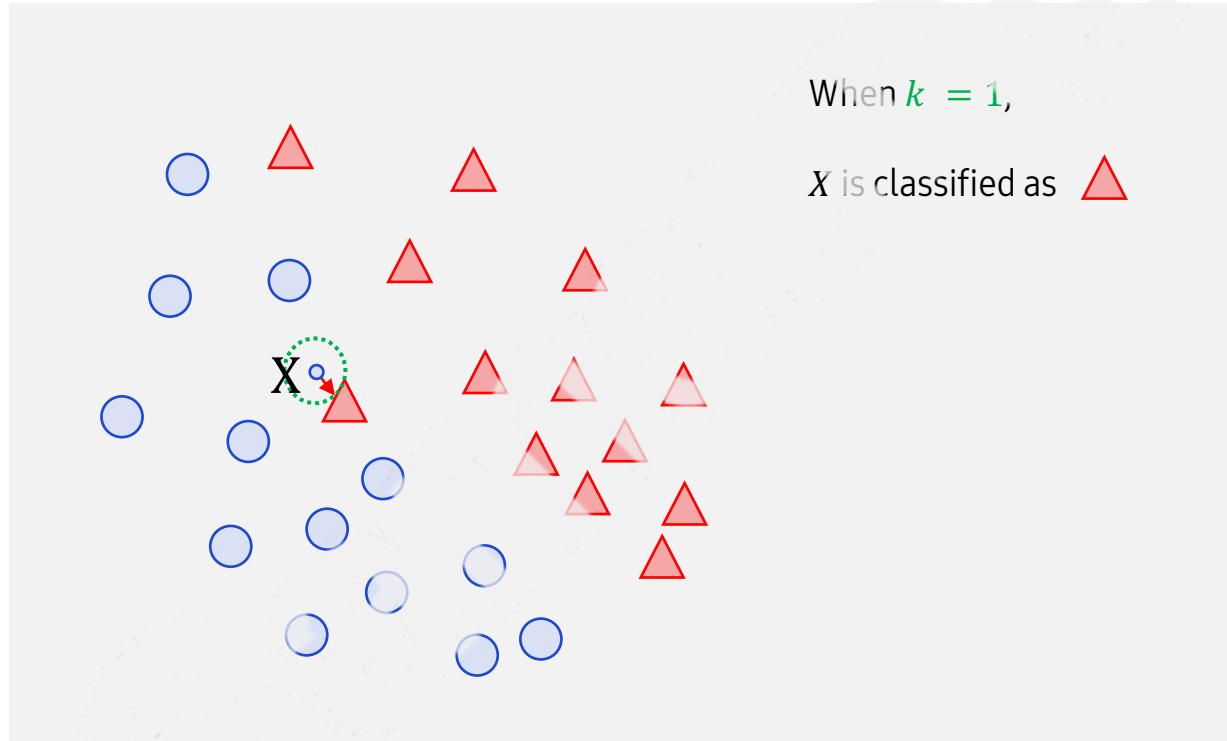
Pros

- ▶ Simple and intuitive
- ▶ No model parameters to calculate. So, there is no training step.

Cons

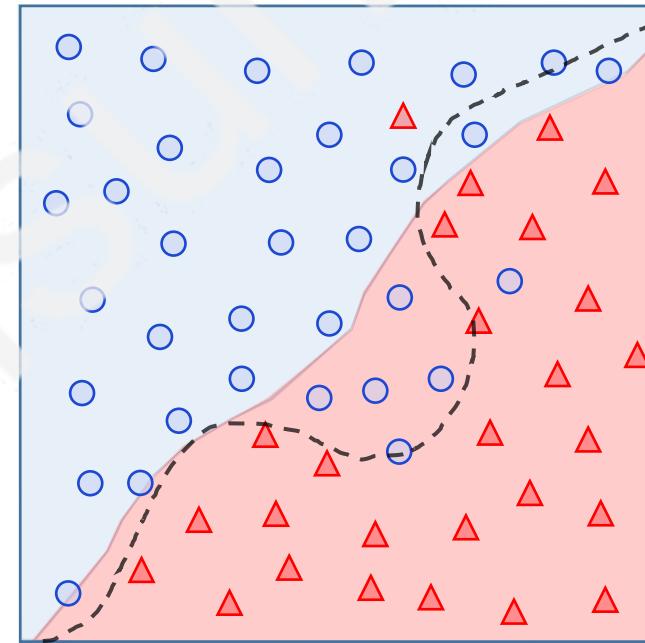
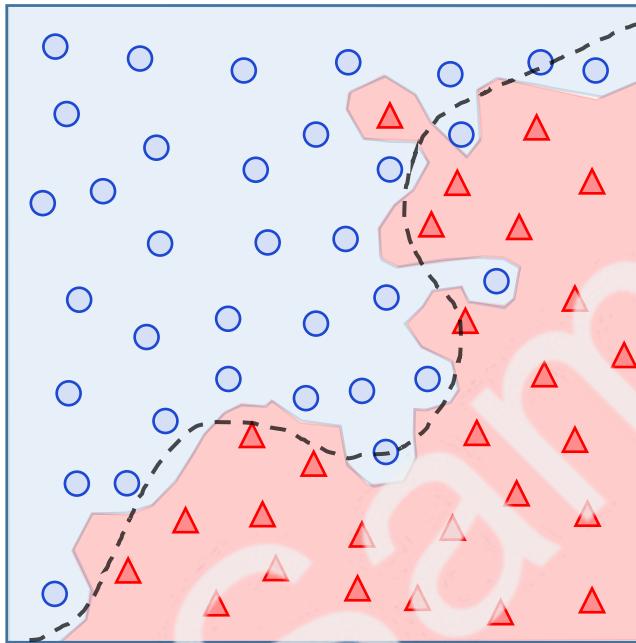
- ▶ Since there is no “model,” little insight can be extracted.
- ▶ No model parameters that store the learned pattern. The training dataset is required for prediction.
- ▶ Prediction is not efficient \Rightarrow “Lazy algorithm.”

| KNN classification algorithm



I KNN classification algorithm

- When k is too small, overfitting may happen.



Coding Exercise #0310



Follow practice steps on 'ex_0310.ipynb' file.

Unit 7.

SVM Algorithm

I 7.1. SVM Algorithm

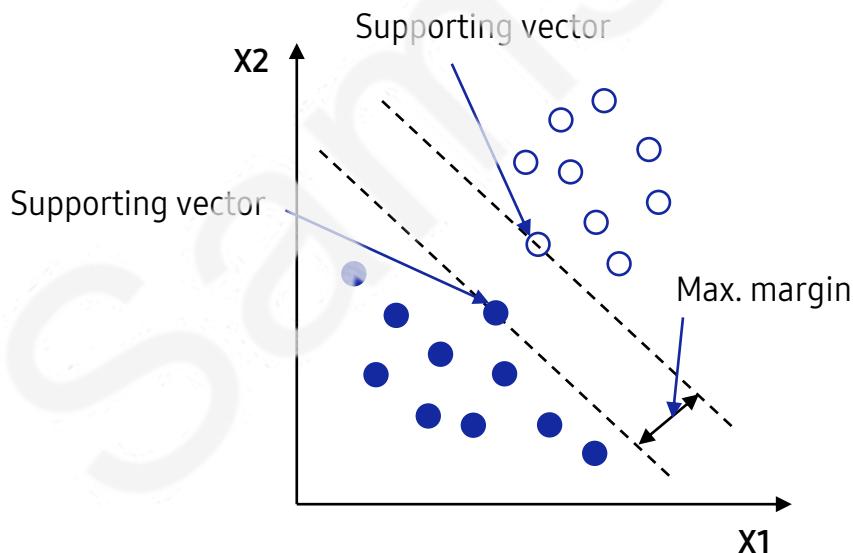
SVM Algorithm

| About SVM (Support Vector Machine)

- ▶ Enhanced classification accuracy by maximizing the margin
- ▶ Effective non-linear classification boundary by the “kernel” transformation

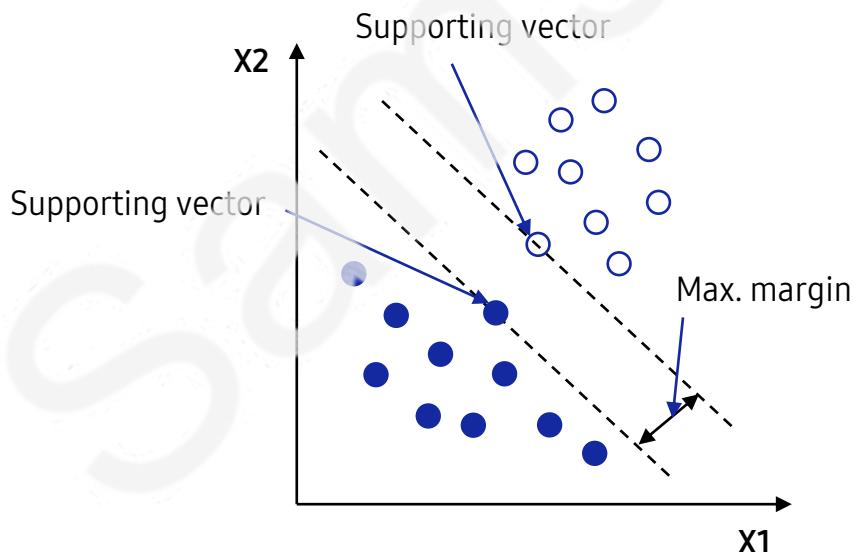
About SVM (Support Vector Machine)

- ▶ Support vector is a model that classifies data by finding the line (or hyperplane) where the distance (margin) between data in different categories becomes maximum.
- ▶ As shown in the figure below, the support vector machine model finds the hyperplane that splits the data into two different categories with maximum margin to classify data.
- ▶ There would be many lines or planes that split data into two different categories. However, points that go over the classification boundary can occur in evaluation data or unknown future data as the points near the boundary slightly change.



About SVM (Support Vector Machine)

- To minimize such a possibility, a line (or hyperplane) making the maximum margin between different data categories should be found. In other words, the aim is to find a hyperplane that can induce maximum differentiation for the best possible generalization to classify and predict the future data, not the current training data.
- The point in the category closest to the boundary line is called a support vector, and each classification should have at least one support vector.



| About SVM (Support Vector Machine)

- ▶ However, it is not always possible to classify all data linearly. Sometimes, data classification should be done in a curve or a more complex non-linear classification plane.
- ▶ If so, use the Kernel trick method for mapping the given data into a higher dimension and find a hyperplane that can classify the data in the converted dimension.
- ▶ Instead of converting the data to a higher dimension, the Kernel trick method uses the Kernel function that returns a similar value to when performing internal calculations between vectors in the higher dimension. This can result in a non-linear classification without moving data to a higher dimension.
- ▶ The major Kernel functions include polynomial, Gaussian, and sigmoid kernels.

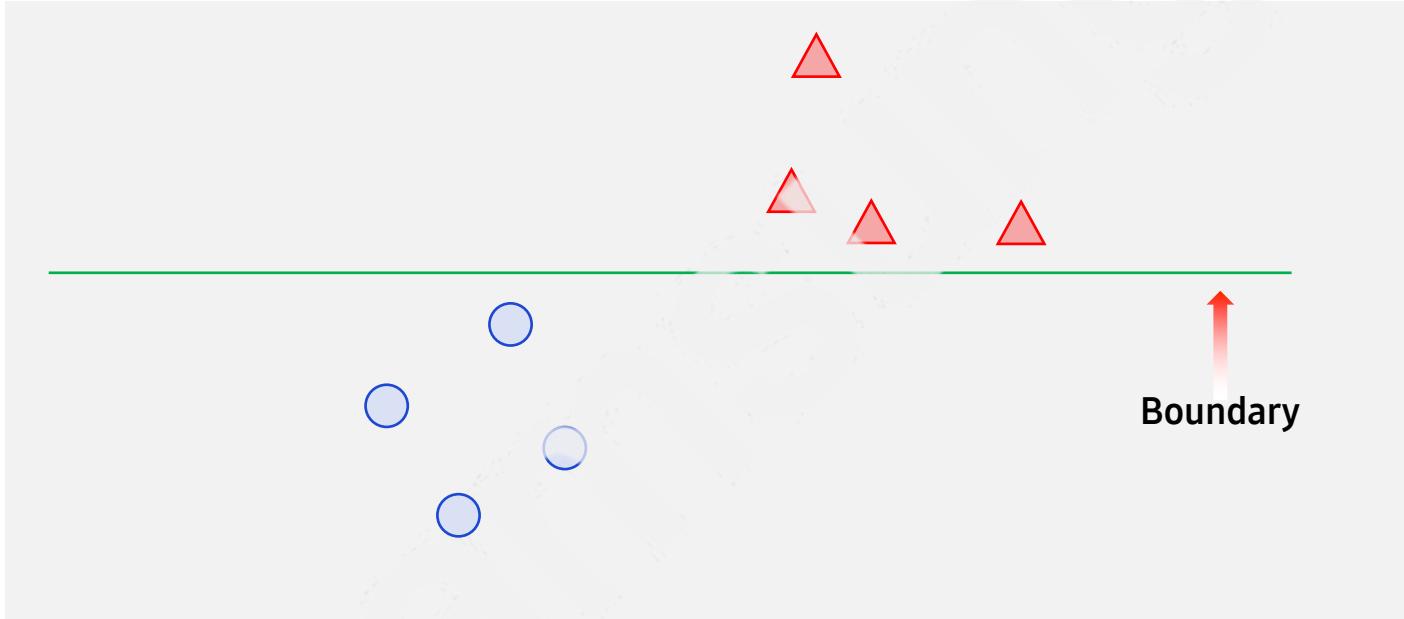
Pros

- ▶ Not very sensitive to the outliers.
- ▶ Performance is good.

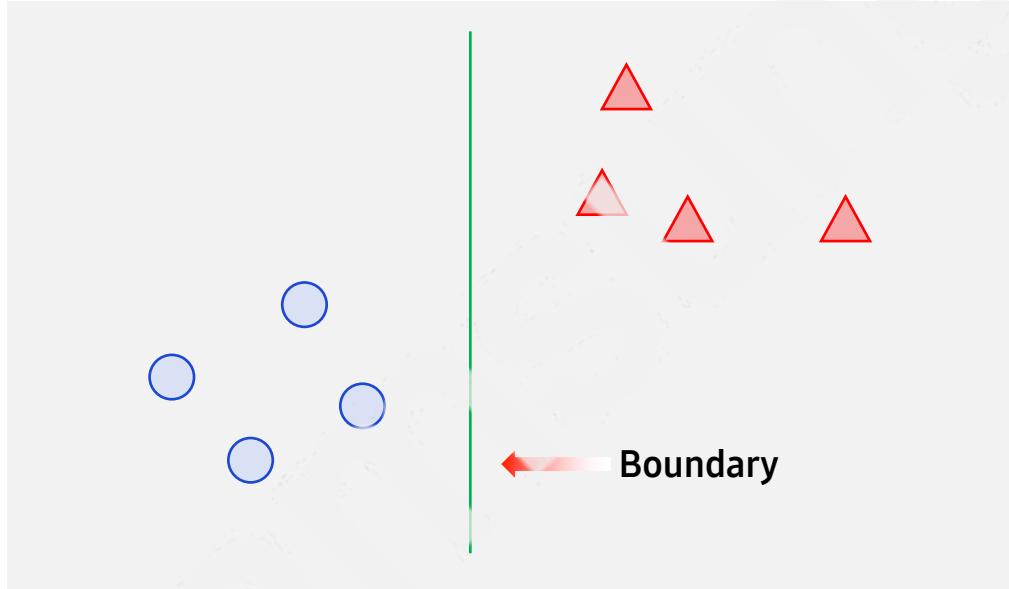
Cons

- ▶ Training is relatively slow. Performs poorly for large data.
- ▶ The kernel and the hyperparameter set should be carefully optimized.
- ▶ Not much insight can be gained.

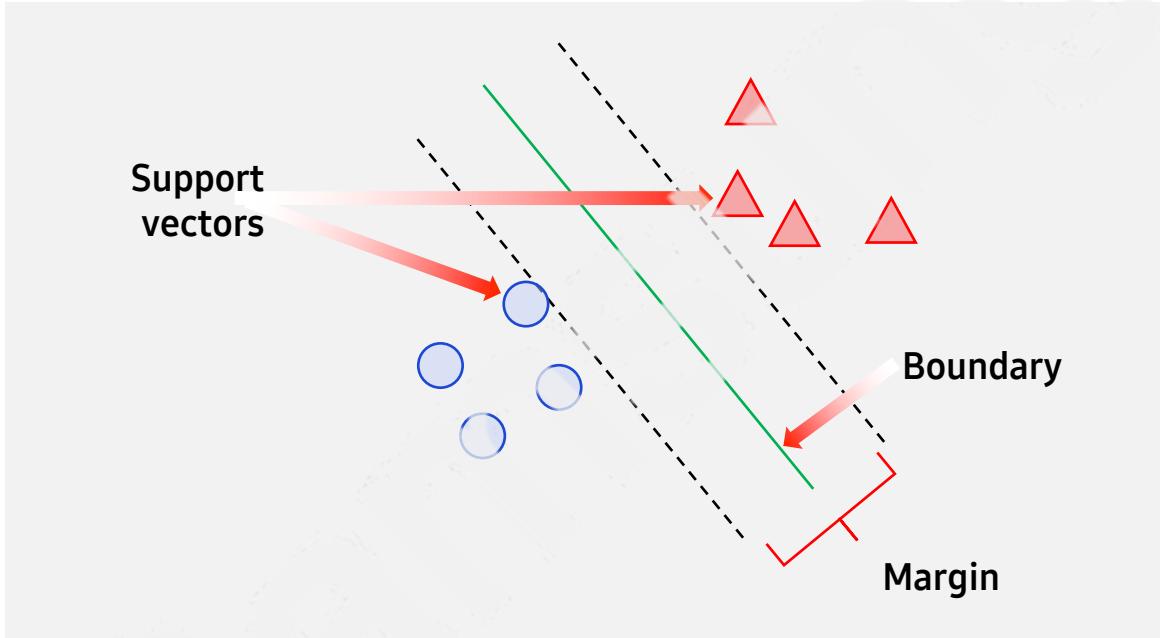
I SVM classification algorithm



I SVM classification algorithm



I SVM classification algorithm



| Hyperplane

- ▶ For a k dimensional configurational space, the hyperplane has the dimension $k-1$.

Ex For a two dimensional space, a hyperplane is a bisecting line that can be parametrized as:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

The two dimensional space is subdivided into two:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$$

An observation would belong to either one of them \Rightarrow binary classification!

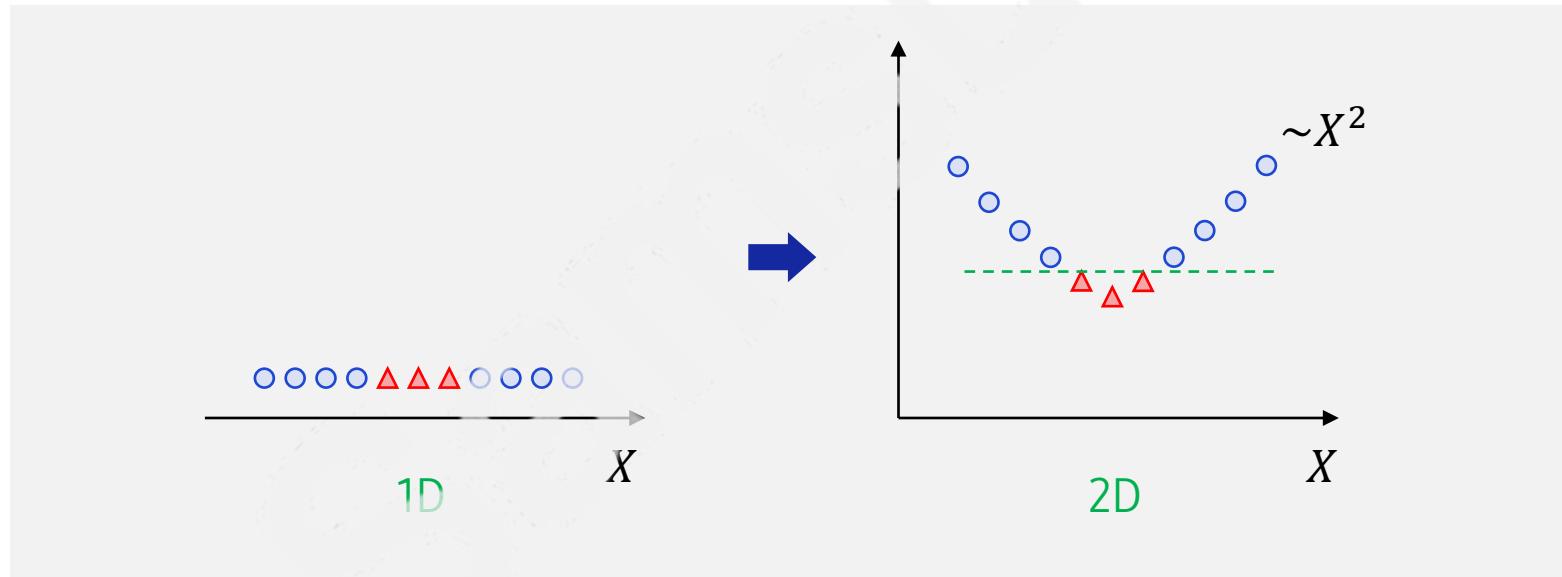
| Margin optimization (for the binary y)

- ▶ The boundary margin is maximized.
- ▶ If a clear margin is not possible to establish, an error is allowed within a limit.
- ▶ Target is to maximize M by optimizing the parameters $\beta_0, \beta_1, \dots, \beta_k$ subject to the constraints:
 - **Constraint 1:** $\sum_{j=1}^k \beta_j^2 = 1$
 - **Constraint 2:** $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik}) \geq M(1 - \varepsilon_i)$, $i = 1, 2, \dots, n$
 y_i is either -1 or +1.
 - **Constraint 3:** $\varepsilon_i \geq 0$ and $\sum_{i=1}^n \varepsilon_i \leq C$
↳ C is a hyperparameter related to the miss-classification errors.

| Kernel

- ▶ Mapping to a higher dimension using the “kernel” functions.
- ▶ Kernel functions introduce an effective non-linear classification boundary.

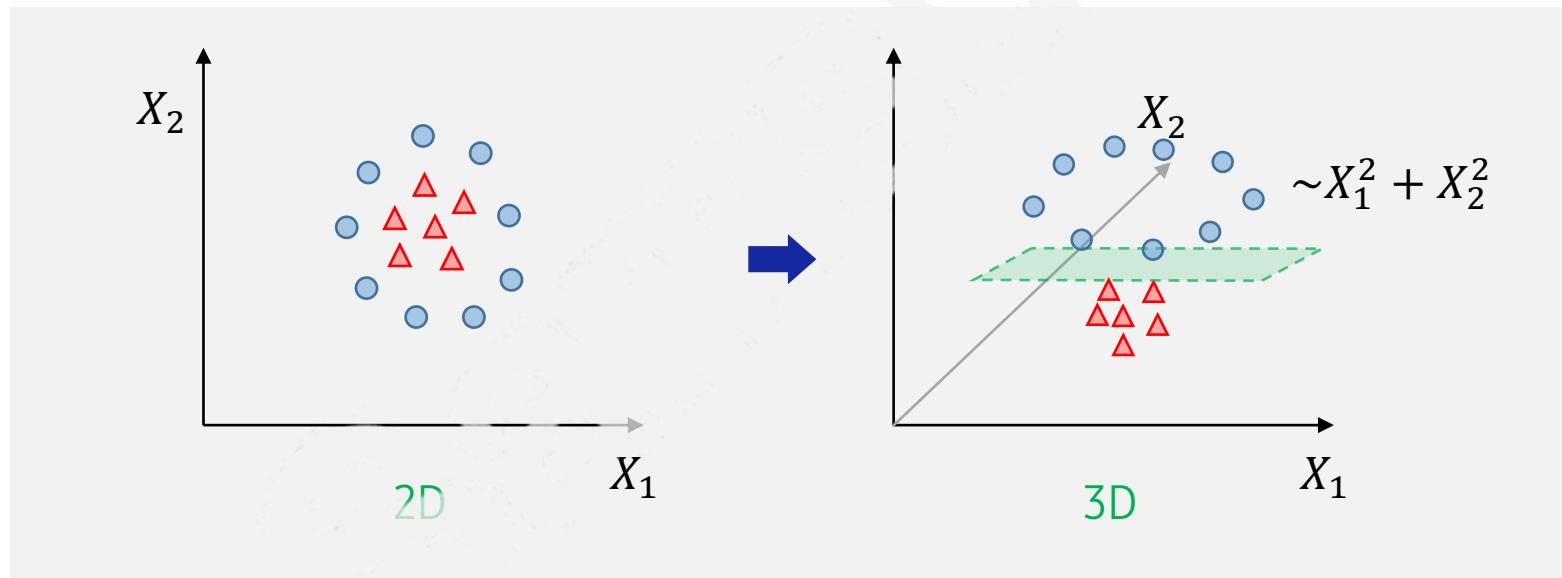
Ex Polynomial kernel



| Kernel

- ▶ Mapping to a higher dimension using the “kernel” functions.
- ▶ Kernel functions introduce an effective non-linear classification boundary.

Ex Polynomial kernel



| Kernel

- ▶ Effective mapping to a higher dimension by giving the inner product of two vectors x_1 and x_2 .
 - Linear: $K(x_1, x_2) = x_1^t x_2$
 - Polynomial: $K(x_1, x_2) = (x_1^t x_2 + c)^d$, where $c > 0$
 - Sigmoid: $K(x_1, x_2) = \tanh(a(x_1^t x_2) + b)$, where $a, b > 0$
 - Radial function basis (rbf): $K(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$, where $\gamma > 0$

Coding Exercise #0311



Follow practice steps on 'ex_0311.ipynb' file.

Unit 8.

Ensemble Algorithm

- | **8.1. The concept of Ensemble Algorithm and Voting**
- | 8.2. Bagging & Random Forest
- | 8.3. Boosting

Ensemble algorithms

| About ensemble algorithms

- ▶ Strong predictive model based on the weaker learners.

- ▶ **Voting type:**

- A collection of basic learners that “vote”
- An ensemble of different kinds of learners **Ex** `Decision Tree, KNN, SVM, etc.`

- ▶ **Bagging type:**

- A collection of independent weak learners that “vote”
- An ensemble of the same kind of weak learners **Ex** `Random Forest`

- ▶ **Boosting type:**

- A series of weak learners that adaptatively learn and predict **Ex** `AdaBoost, GBM, XGBoost, etc.`
- The series is grown by adding new learners multiplied by the “boosting weights.”

| About ensemble algorithms

- ▶ Making a more suitable decision-making by appropriately combining multiple opinions obtained by different experts
- ▶ Majority Voting – Analysis using many different classification models with one identical training set
- ▶ Bagging (different training set)
 - bootstrap + aggregating
 - Bootstrap + aggregating
 - Bagging (random sampling with replacement (random bootstrap))

Ex Random Forest

- ▶ Boosting
 - Retain 50% of incorrectly classified data or use weight for sample selection.

| bootstrap

- ▶ Estimation of unknown statistics
- ▶ Easy and effective estimation method with the unknown distribution of the model parameter sample
- ▶ Process of recalculating the statistics and model for each sample through additional sampling with replacement from the current sample
 - No assumption is required such that the parameters or sample statistics should be in a normal distribution.
- ▶ bootstrap sample – Aggregation of observed data (sampling with replacement obtained from the record value and dependent variable)

Contents to be learned in the future

- Resampling – Involves permutation under-sampling without replacement
- Bootstrap aggregation – Making a result from aggregating predicted values obtained from different bootstrap samples

| What is Ensemble Learning?

- ▶ The term 'ensemble' can be defined as follows:
 - In statistical mechanics, an ensemble of a system refers to the collection of equivalent systems.
- ▶ In other words, it is an assembly of similar groups.
- ▶ Instead of expecting performance results from a single model, ensemble learning draws a better result using the collective intelligence of different models, such as averaging out many different single models or making a decision based on the majority vote.
- ▶ There are many different ensemble methods using collective intelligence.
 - Voting – Drawing results through voting
 - Bagging – Bootstrap aggregating (duplicated creation of various samples)
 - Boosting – Weighting by supplementing previous errors
 - Stacking – A meta-model based on different models
- ▶ There can be other more different methods since ensemble learning applies a certain technique/methodology. Yet, the four methods listed above are the most representative ensemble techniques in the sklearn library.

Voting Ensemble

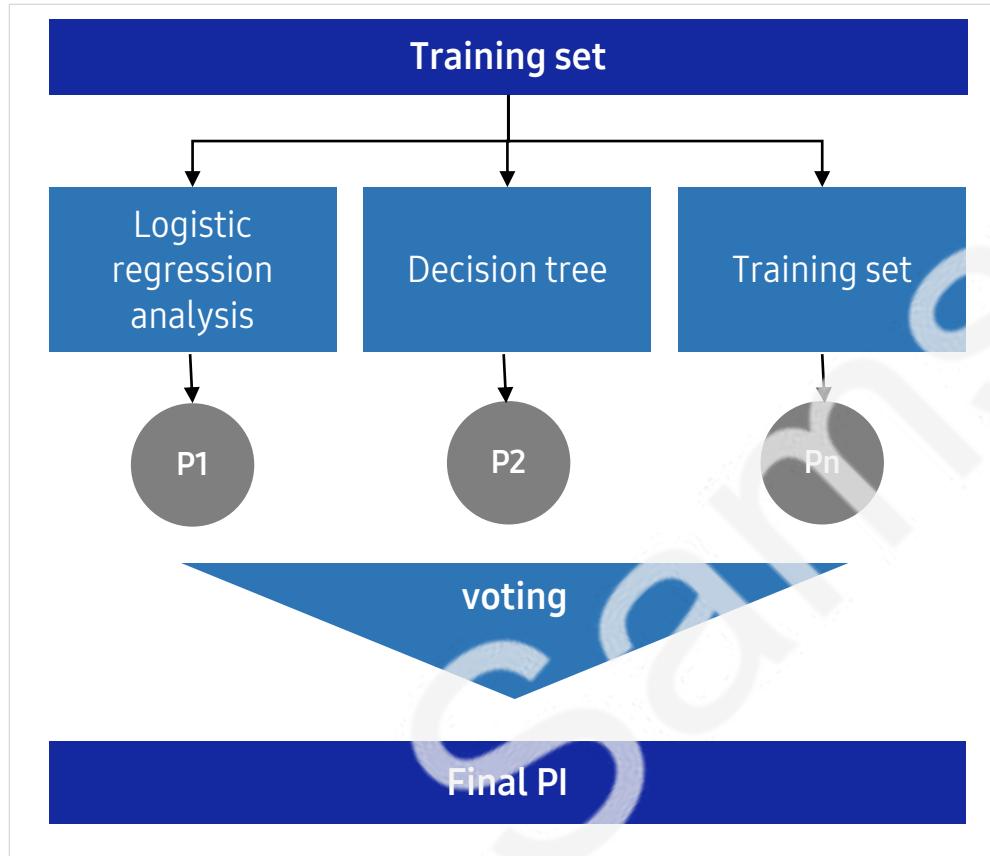
| Voting ensemble

- ▶ Can be applied to classification and regression.
- ▶ The learners that form an ensemble should be of different kinds for a good performance.
- ▶ Two voting methods for the classifier:
 - Hard: predicted class label is given by the majority rule voting.
 - Soft: predicted class label is given by the **argmax** of the sum of the predicted probabilities.

| Voting

- ▶ As the word itself, voting makes a decision through votes. Voting is similar to bagging as it uses a voting method, but they are highly differentiated from each other as follows:
 - Voting: Combines different algorithm models.
 - Bagging: Uses different sample combinations within the same algorithm.
- ▶ Voting selects final results by having final voting on results deduced by different algorithms.
- ▶ Voting is classified into hard voting and a soft voting.
 - Hard voting: Decides the final value of the result through voting.
 - Soft voting: Draws the final value by adding all the probability values of getting the final result and then calculating every probability of the final result.

Voting



- ▶ Use a single training set.
 - ▶ Use different classification models.
 - ▶ Predictive value
 - ▶ If the predictive values of each analysis model are different from voting, choose the result with the most values.
 - hard voting
 - soft voting
- Predict the highest class by averaging out the prediction of individual classifier.

| Voting

- ▶ Hard Voting
 - Taking classification as an example. Suppose the predictive values for classification are 1,0,0,1,1 since 1 has three votes and 0 has 2 votes. In that case, 1 becomes the final predictive value in the hard voting method.
- ▶ Soft Voting
 - Soft voting method calculates the average value of each probability and then determines the one with the highest probability.
 - Suppose the probability of getting class 0 is (0.4, 0.9, 0.9, 0.4, 0.4), and the probability of getting class 1 is (0.6, 0.1, 0.1, 0.6, 0.6). The final probability of getting class 0 is $(0.4+0.9+0.9+0.4+0.4) / 5 = 0.44$; the final probability of getting class 1 is $(0.6+0.1+0.1+0.6+0.6) / 5 = 0.4$. Therefore, the selected final value is different from the result of the hard voting above.
- ▶ In general, using the soft voting method is considered more reasonable than the hard voting method in competitions because the soft voting method provides a much better actual performance result.

| Voting ensemble in Scikit-Learn

- ▶ To import the voting ensemble as class:

```
from sklearn.ensemble import VotingClassifier      # For classification  
from sklearn.ensemble import VotingRegressor       # For regression
```

- ▶ To instantiate an object that implements voting ensemble:

Ex myKNN = KNeighborsClassifier(n_neighbors = 3)

myLR = LogisticRegression()

myVotingEnsemble=VotingClassifier(estimators=[('lr',myLR),('knn',myKNN)],voting='hard')

| Voting ensemble in Scikit-Learn

- ▶ We can train and predict just like any other estimator:

Ex `myVotingEnsemble.fit(X_train, Y_train)`

`myVotingEnsemble.predict(X_test)`

| Scikit-Learn VotingClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
estimators	The list of basic learner objects
voting	Either 'soft' or 'hard' (for classifier only)

- ▶ More information can be found at:

Classifier: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html>

Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html>

Unit 8.

Ensemble Algorithm

- | 8.1. The concept of Ensemble Algorithm and Voting
- | **8.2. Bagging & Random Forest**
- | 8.3. Boosting

Bagging Ensemble: Random Forest

| About Random Forest

- ▶ An ensemble algorithm based on trees
- ▶ Can be applied to classification and regression

| Pros

- ▶ Powerful
- ▶ Few assumptions
- ▶ Little or no concern about the overfitting problem

| Cons

- ▶ Training is time consuming.

| Random Forest algorithm



- ▶ Randomly chosen trees form a forest
- ▶ Prediction is decided by the majority vote.

| Random Forest algorithm

- 1) Make trees with randomly selected variables and observations.
- 2) Keep only those with the lowest Gini impurity (or entropy).
- 3) Repeat from step 1) a given number of times.
- 4) Using the trees gathered during the training step, we can make predictions by majority vote.

| Scikit-Learn RandomForestClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
n_estimators	The number of trees in the forest
max_depth	The maximum depth of a tree
min_samples_leaf	The minimum number of sample points required to be at a leaf node
min_samples_split	The minimum number of sample points required to split an internal node
max_features	The number of features to consider when looking for the best split

- ▶ Except for “n_estimators,” the rest are analogous to those of DecisionTreeClassifier/Regressor.
- ▶ More information can be found at:

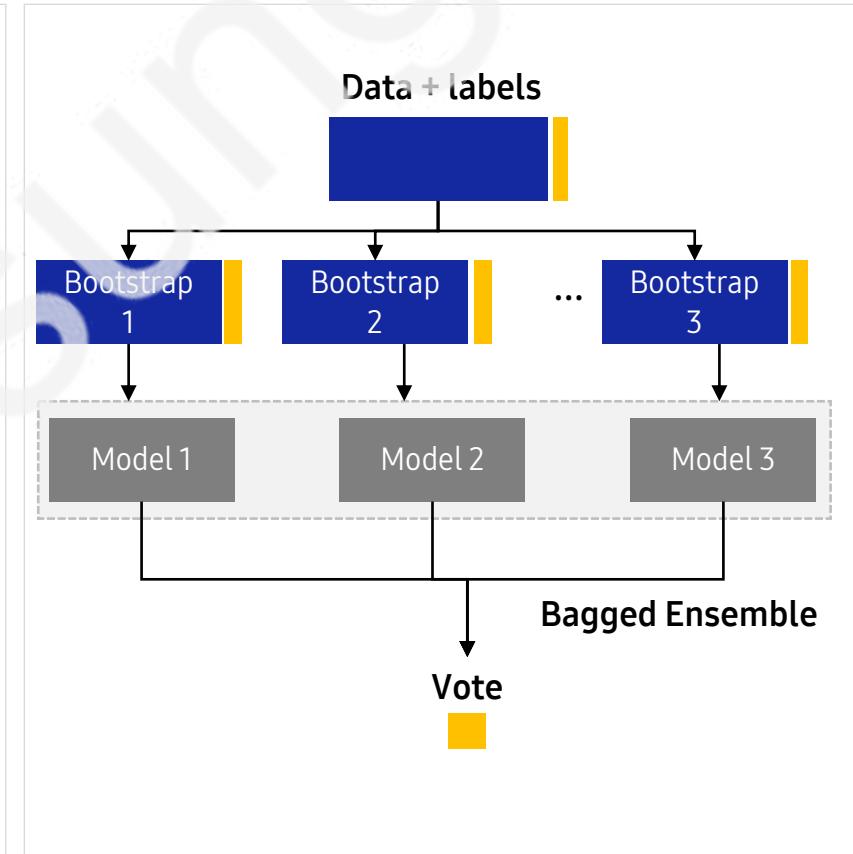
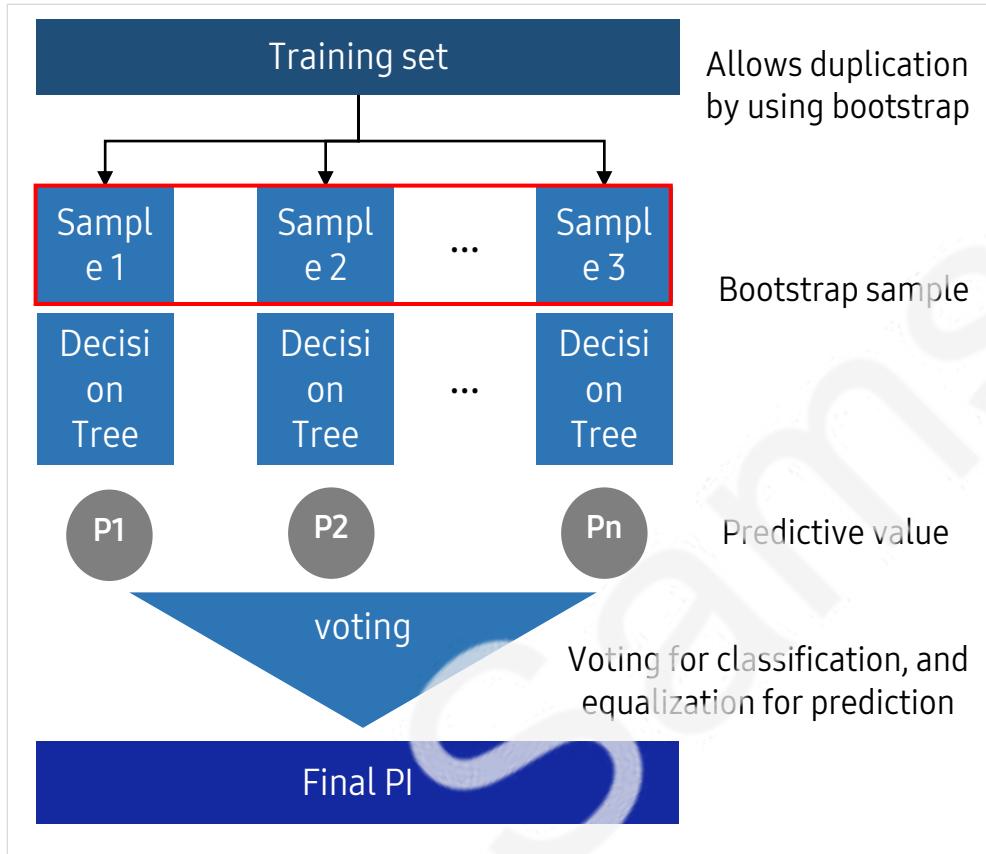
Classifier: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

| Bagging

- ▶ Bagging-based ensemble method
- Ex Random Forest algorithm
 - Easy to use since it is well constructed in the Sklearn library
 - Relatively quick performance speed
 - High performance
- ▶ The ensemble method has been widely used as it raises the performance level and is easy to use. The bagging-based ensemble method is commonly found in the high-ranked solutions in Kaggle.

| Bagging



| Bagging

- ▶ Bagging is an abbreviation of Bootstrap Aggregating.
- ▶ Bootstrap = Sample
- ▶ Aggregating = Adding up
- ▶ Bootstrap refers to a method that allows overlapping of different data sets for sampling and splitting.

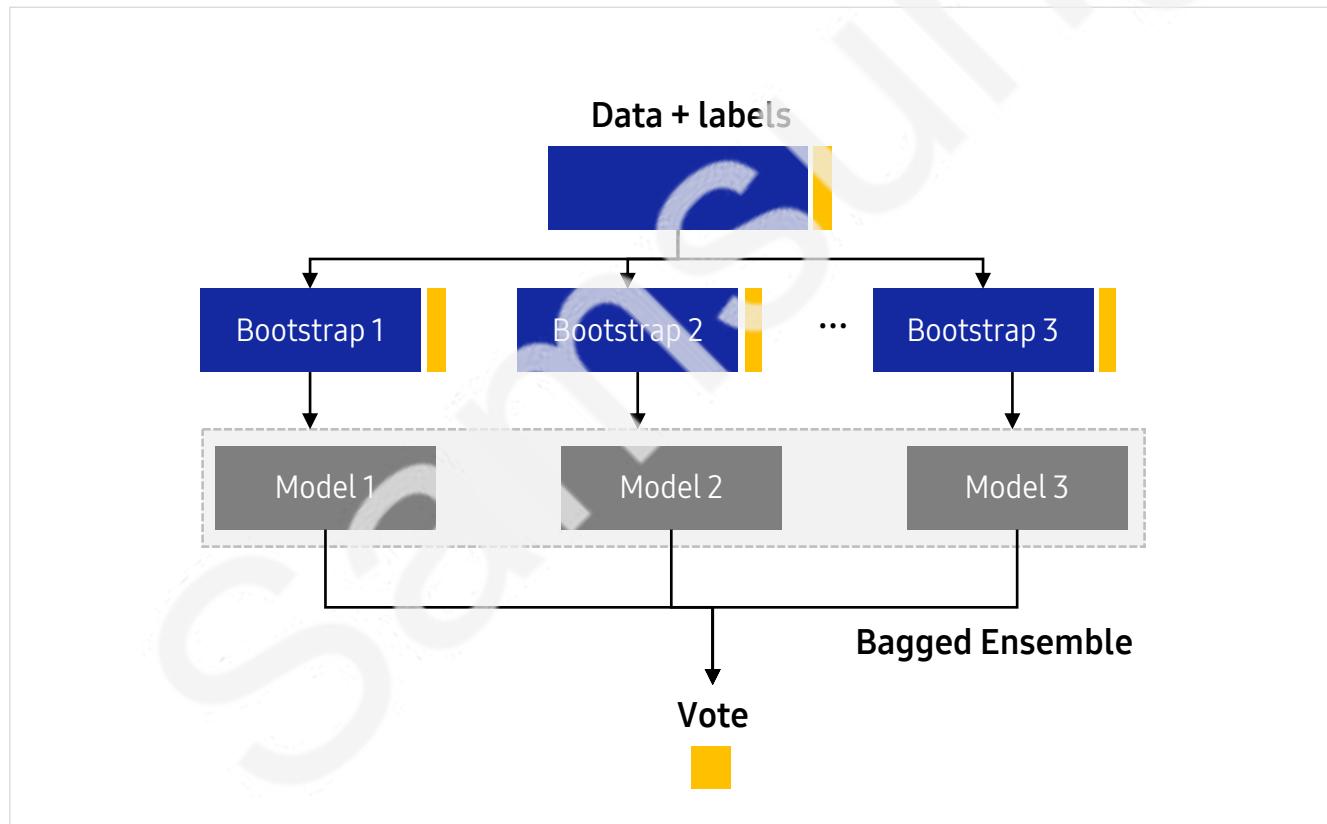
| Bagging

Ex Random Forest is a typical bagging method algorithm.

- ▶ It creates multiple decision trees and performs sampling of different data sets while allowing overlapped data sets.
- ▶ If the data set consists of [1, 2, 3, 4, 5]:
 - Group 1 = [1, 2, 3]
 - Group 2 = [1, 3, 4]
 - Group 3 = [2, 3, 5]
- ▶ This is the bootstrap method. In the classification problem, voting is done on each tree trained with different sampling for the final prediction result.
- ▶ In regression problems, the average of each obtained value is calculated.

| Bagging

- ▶ “Bagging”: Bootstrap AGGREGatING



I Differences Between Bagging and Voting

- ▶ The greatest difference between the bagging and voting methods is whether to use multiple single algorithms or apply various algorithms to the same sample data set.
- ▶ In general, the bragging method is to train a single algorithm with different sampling data sets and perform voting. It has relatively better usability than the voting method because it uses a single algorithm. Thus, what is important is the hyperparameter of the single algorithm.
- ▶ Taking the Random Forest algorithm as an example again, it is suitable to obtain the baseline score as it requires simple hyperparameter setting such as how many trees to use (`n_estimators`), maximum depth (`max_depth`), and the minimum number of samples for splitting (`min_samples_leaf`).

I Advantages of the Bagging Ensemble

- ▶ When using the bagging method, it can reduce variance compared to making a prediction with a single model. There are three major training errors in a model: variance, noise, and bias. (Of course, more significant factors include overfitting/underfitting and many different preprocessing issues, but assume they are already being set.)
- ▶ The ensemble method reduces variance, thus enhancing the performance of the final result.

| `sklearn.ensemble.BaggingClassifier/BaggingRegressor`

- ▶ The `sklearn` library package provides the wrapper class called `BaggingClassifier/BaggingRegressor`.
- ▶ When designating the base algorithm to the `base_estimator` parameter, the `BaggingClassifier/BaggingRegressor` performs bagging ensemble.

Unit 8.

Ensemble Algorithm

- | 8.1. The concept of Ensemble Algorithm and Voting
- | 8.2. Bagging & Random Forest
- | **8.3. Boosting**

Boosting Ensemble: AdaBoost

| About AdaBoost

- ▶ A sequence of weak learners such as trees
- ▶ A weighted ensemble of weak learners
 - One set of weights that increase the importance of the better-performing learners
 - One set of weights that increase the importance of the wrongly classified observations
- ▶ Similar pros and cons as the Random Forest

| AdaBoost classification algorithm

- ▶ Let's suppose n observations for the training step: \mathbf{x}_i and y_i .
We also suppose that $y_i \in \{-1, +1\}$. (binary y)
- ▶ We will make a series of weak learners $G_m(\mathbf{x})$ with $m = 1, \dots, M$.
- ▶ The ensemble classifier is made up of a linear combination of these weak learners:

$$G_{ensemble}(\mathbf{x}) = sign \left(\sum_{m=1}^M \alpha_m G_m(\mathbf{x}) \right)$$

where α_m are the “boosting weights” that need to be calculated.

- ▶ Heavier weight is given to a better performing learner.

| AdaBoost classification algorithm

1) For the first step ($m=1$), equal weight is assigned to the observations:

$$w_i^{(1)} = \frac{1}{n}$$

2) For the boost sequence $m=1, \dots, M$:

a) Train the learner $G_m(\mathbf{x})$ using observations weighted by $w_i^{(m)}$.

b) Calculate the error ratio:

$$\varepsilon_m = \frac{\sum_{i=1}^n w_i^{(m)} I(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^n w_i^{(m)}}$$

$\Rightarrow I(y_i \neq G_m(\mathbf{x}_i))$ gives 1 for an incorrect prediction, else 0.

$\Rightarrow 0 \leq \varepsilon_m \leq 1$

| AdaBoost classification algorithm

3) For the boost sequence $m=1, \dots, M$:

c) Calculate the boosting weight: α_m

$$\alpha_m = \frac{1}{2} \log \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$

⇒ As $\varepsilon_m \rightarrow 0$, α_m is a large positive number. The learner is given more importance!

⇒ As $\varepsilon_m \cong 0.5$, $\alpha_m \cong 0$.

⇒ As $\varepsilon_m \rightarrow 1$, α_m is a large negative number.

d) For the next step, the weights of the **wrongly** predicted observation are rescaled by a factor e^{α_m} .

This can be compactly expressed as: $w_i^{(m+1)} = w_i^{(m)} \times e^{\alpha_m \cdot I(y_i \neq g_m(x_i))}$, where $i = 1, \dots, n$

⇒ In the next sequence step, the **wrongly** predicted observations receive heavier weight.

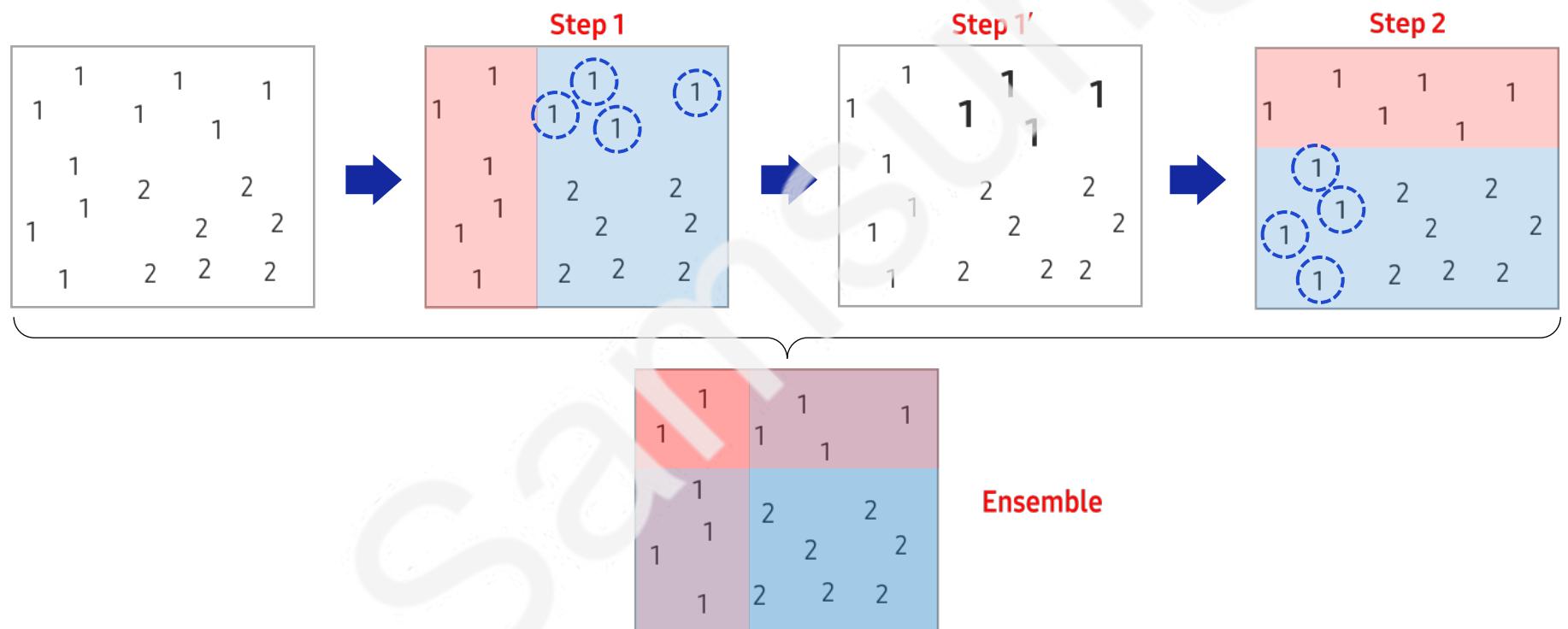
| AdaBoost classification algorithm

4) The ensemble classifier is made up of a linear combination of weak learners:

$$G_{ensemble}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(\mathbf{x})\right)$$

5) For a new testing condition \mathbf{x}' , we can predict y' by $G_{ensemble}(\mathbf{x}')$.

| AdaBoost classification algorithm



| Scikit-Learn AdaBoostClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
base_estimator	The base estimator with which the boosted ensemble is built
n_estimators	The maximum number of estimators at which boosting is terminated
learning_rate	The rate by which the contribution of each learner is shrunken
algorithm	Either 'SAMME' or 'SAMME.R'

- ▶ “base_estimator” is by default **None**, which means **DecisionTreeClassifier(max_depth=1)**.
- ▶ More information can be found at:

Classifier: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>

Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html>

Boosting Ensemble: GBM & XGBoost

| Gradient Boosting Machine (GBM)

- ▶ GBM is also made up of a sequence of weak learners similar to AdaBoost.
- ▶ The disagreement between the predicted \hat{y} and the true y can be represented by a “loss” function.
- ▶ In the sequence of weak learners, the learner at step m , $G_m(\mathbf{x})$ can be obtained by moving the previous step learner $G_{m-1}(\mathbf{x})$ towards the direction that decreases the loss as defined above.
- ▶ These updating movements are restricted to a set of base learners.

| Scikit-Learn GradientBoostingClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
loss	The loss function
n_estimators	The number of boosting steps (weak learners)
learning_rate	The contribution of each weak learner
subsample	The fraction of data that will be used by individual weak learner

- ▶ Need to be tuned for optimized performance.
- ▶ More information can be found at:

Classifier: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>

Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>

| Extreme Gradient Boosting (XGBoost)

- ▶ Improves upon GBM in the execution speed.
- ▶ More resistant to the overfitting than GBM.
- ▶ Not included in the Scikit-Learn library. Requires installation of the “xgboost” library.

XGBClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
booster	gbtree or gblinear
n_estimators	The number of boosting steps (weak learners)
learning_rate	The contribution of each weak learner
subsample	The fraction of data that will be used by individual weak learner

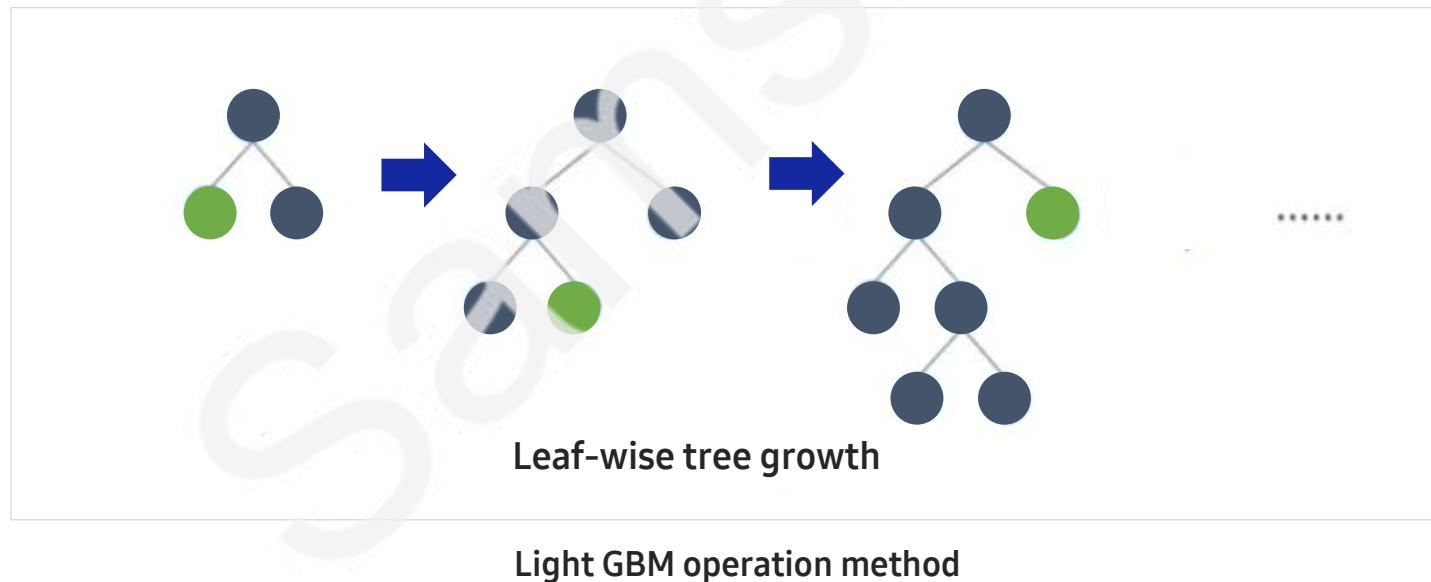
- ▶ Need to be tuned for optimized performance.
- ▶ More information can be found at: <https://xgboost.readthedocs.io/en/latest/python/index.htm>

XGBoost

- ▶ XGBoost is a library that implements the gradient boosting algorithm to be used in the distributed system.
- ▶ It supports both regression and classification problems. This popularly used algorithm features good performance and resource efficiency.
- ▶ Gradient boost is a representative algorithm using the boosting method. The XGBoost is the library using this algorithm to support parallel training.
- ▶ It has been recently used a lot due to its high performance and computer resource application rate. It has become more popular as it was frequently used by top rankers of Kaggle.

| Light GBM

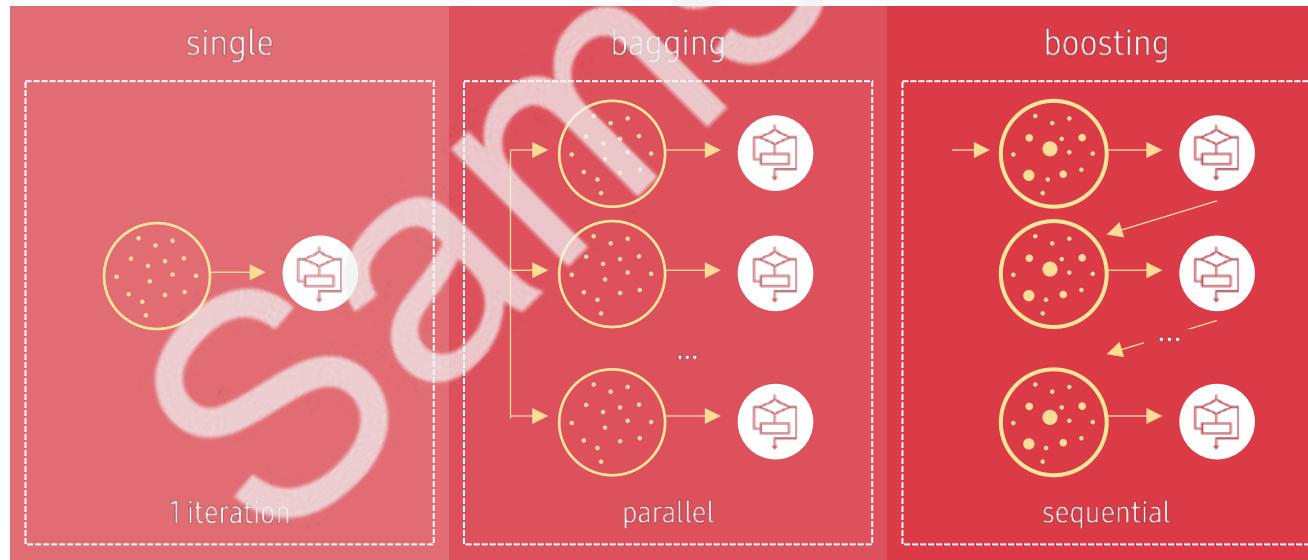
- ▶ While the tree is vertically expanded with Light GBM, other algorithms expand the tree horizontally. In other words, while the Light GBM is leaf-wise, other algorithms are level-wise.
- ▶ For expansion, the leaf with max delta loss is selected. When expanding the same leaf, the leaf-wise algorithm can reduce more loss than the level-wise algorithm.
- ▶ The following diagram shows how a boosting algorithm is implemented, which differs from the Light GBM.



| Boosting

- ▶ The boosting algorithm is also ensemble learning. After sequentially learning weak learning machines, it supplements errors by adding weight to inaccurately predicted data from the previous learning.
- ▶ The difference from other ensemble methods is that it performs sequential learning and supplements errors by adding weight. However, one of the disadvantages is that it is difficult to process parallel due to its sequential property, leading to a longer learning time compared to other ensembles.

Single estimator, Bagging, Boosting



| Learning a series of predictors by supplementing previous models

▶ AdaBoost

- Train the first distributor (e.g., decision tree) in the training set and make a prediction.
- Slightly increase the weight of the training sample with an inaccurately classified algorithm.
- The second distributor uses the updated weight to be trained at the training set and makes a prediction again.
- The weight is updated again, and the same process is repeated.

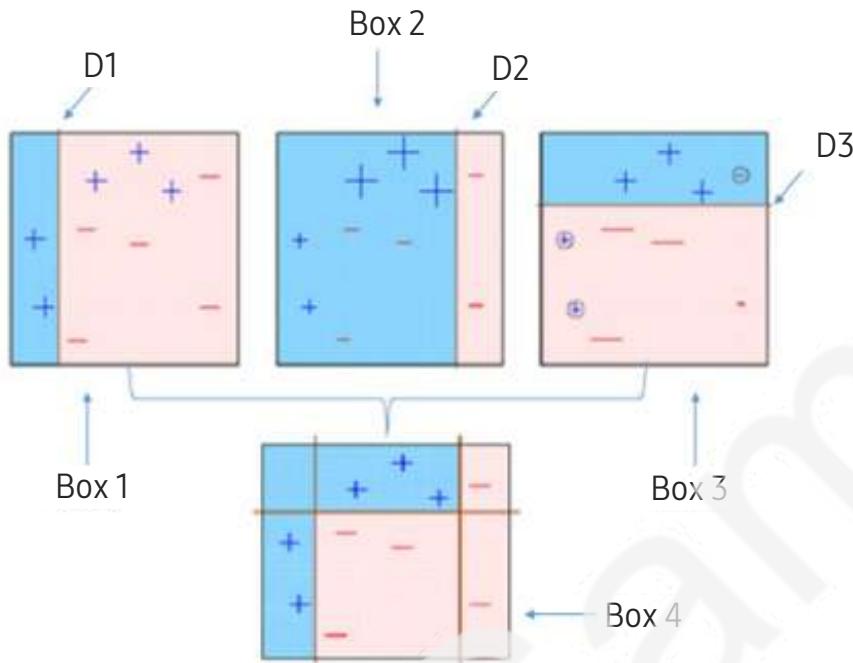
▶ Gradient boosting

- Similar to AdaBoost, gradient boosting sequentially adds predictors to the ensemble to correct the previous errors.
- However, instead of modifying the sample weight repeatedly as AdaBoost, it trains a new predictor to residual error created by the previous predictor.
- For regression problems – gradient boosted regression tree (GBRT)

| AdaBoost (Adaptive Boosting)

- ▶ AdaBoost (Adaptive Boosting) is an adaptive boosting method, one of the most representative boosting algorithms.
- ▶ As explained, the AdaBoost sequentially learns weak learning machines and supplements errors by applying weights to inaccurately predicted data.

| Principle of the AdaBoost



- ▶ Box 1 results from a weak learning machine classified as the D1 sector. However, the error rate is relatively high because some data sets are expressed in + distributed in the red sector.
- ▶ The D2 line in Box 2 moves to the right to supplement the error rate from Box 1. Here, the data sets expressed in – are distributed in the blue sector for better performance, but it is not satisfactory yet.
- ▶ The D3 line in Box 3 is horizontally drawn on the top. However, the data set expressed in – is incorrectly classified.
- ▶ By training the previous Box 1, 2, and 3, Box 4 finds the most ideal combination. It shows much better performance compared to the previous three individual learning machines.

| Principle of the AdaBoost

- ▶ How is the weight applied for combination?

Ex Suppose that the following weight will be applied to the performance of Box 1~3.

- Performance of Box 1: weight = 0.2
- Performance of Box 2: weight = 0.5
- Performance of Box 3: weight = 0.6

It can be expressed as the following formula:

$$0.2 * \text{Box 1} + 0.5 * \text{Box 2} + 0.6 * \text{Box 3} = \text{Box 4}$$

Gradient Descent

- ▶ The key to the boosting method is to supplement errors from the previous learning.
- ▶ The AdaBoosting and gradient descent methods are slightly different in how to supplement errors.
- ▶ Gradient descent uses differentiation to minimize the difference between the predicted value and actual data.
 - Weight
 - Input_data = feature data (input data)
 - Bias
 - Y_actual = actual data value
 - Y_predict = predicted value
 - Loss = error rate
- ▶ $Y_{\text{predict}} = \text{weight} * \text{input_data} + \text{bias}$
 - The predictive value can be obtained from the above formula.
Calculating the difference with actual data will result in the total error rate.
- ▶ $\text{Loss} = Y_{\text{predict}} - Y_{\text{actual}}$
 - (There are many different functional formulas to define error rate, including root means square error and mean absolute error, but the above definition is provided for convenience.)
 - The purpose of gradient descent is to find the weight that makes the loss closest to 0.

| Gradient Boosting Machine (GBM)

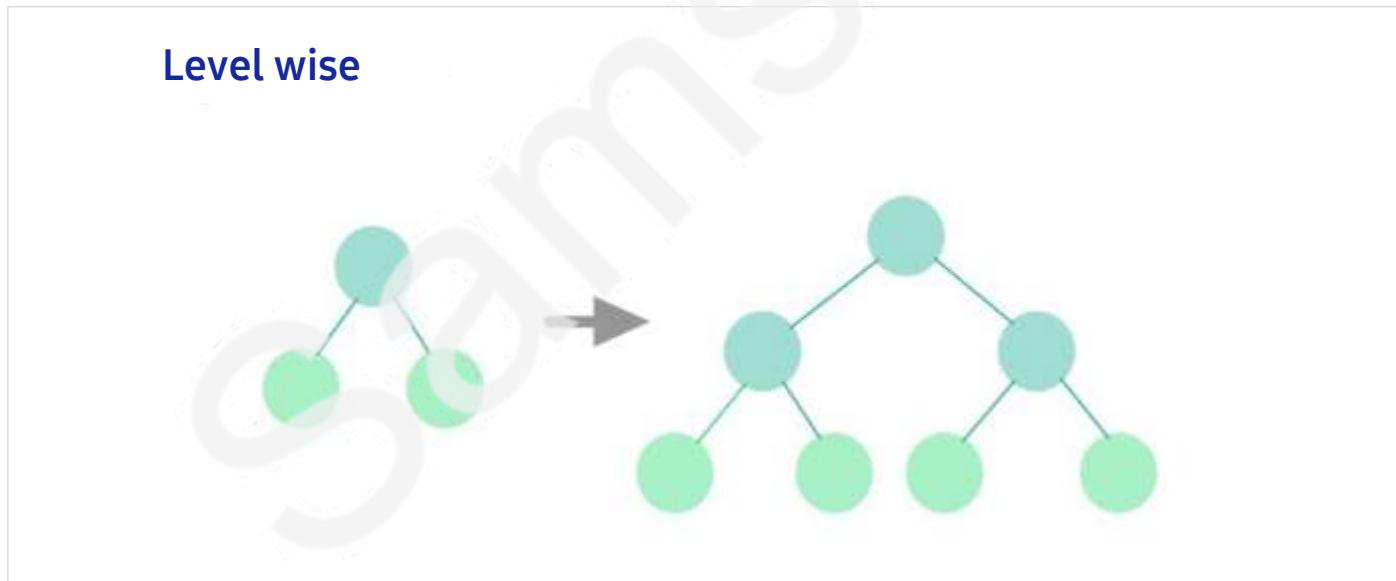
- ▶ The boosting algorithm in gradient descent is called gradient boosting machine, abbreviated as GBM. It is provided in the sklearn package and applicable to both classification and regression problems.
 - GradientBoostingClassifier
 - GradientBoostingRegressor
- ▶ It is extremely easy to apply in actual problems.
- ▶ XGBoost and LightGBM are two major ML packages mostly used in Kaggle.
- ▶ XGBoost and LightGBM are not provided in the existing sklearn package. However, the sklearn wrapper class allows an easy application for fit & predict similar to the ML class of the existing sklearn package.

| Gradient Boosting Machine (GBM)

- ▶ Together with the XGBoost, LightGBM is the most highlighted boosting algorithm. Although XGBoost has an excellent performance, its learning time is too long.
- ▶ Advantages of the LightGBM
 - Short learning time
 - Relatively small memory use
 - Automatic conversion and optimal splitting of categorical features

| Characteristics of the LightGBM

- ▶ The existing tree-based algorithm uses a level-wise method. It splits but maintains a balanced tree as much as possible, so the tree depth becomes minimum. A disadvantage of the level-wise method is that it takes time to make a balanced tree.
- ▶ On the other hand, LightGBM uses a leaf-wise method. It does not balance the tree but continuously splits the leaf node with the max data loss. The tree depth becomes greater, and an asymmetrical tree is created. The repetition of the leaf node with the max data loss minimizes predicted error loss than splitting a balanced tree.



| Hyperparameter tuning methods

- ▶ The increased Num_leaves value raises accuracy. However, it also increases the tree depth and makes the model complex, thus increasing the probability of overfitting.
- ▶ Min_data_in_leaf is a significant parameter for overfitting. It depends on the Num_leaves and the size of training data, but in general, it prevents a deeper tree when setting it as a higher value.
- ▶ Max_depth constraints the depth size and improves overfitting when combined with the two parameters above.

| XGBoost

Parameter	Default value	Description
num_iterations	100	Designates the number of trees for repetitive work. Overfitting occurs if the value is too high.
learning_rate	0.1	Updated when boosting steps are repeatedly conducted. The value is designated between 0~1.
max_depth	1	Identical to the max_depth of tree-based algorithms. No restriction is applied to tree depth when entering a value smaller than 0.
min_data_in_leaf	20	Identical to the min_samples_leaf of a decision tree. Used as a parameter to control overfitting.
num_leaves	3	Signifies the max. The number of leaves for one tree.
boosting	GBDT	
bagging_fraction	1.0	Designates the ratio for data sampling. Used to control overfitting.
feature_fraction	1.0	The ratio of the random feature for individual tree learning
Lambda_l1	0.0	The value for L1 regulation
Lambda_l2	0.0	The value for L2 regulation

Coding Exercise #0312



Follow practice steps on 'ex_0312.ipynb' file

Coding Exercise #0313



Follow practice steps on 'ex_0313.ipynb' file

Coding Exercise #0314



Follow practice steps on 'ex_0314.ipynb' file

A photograph of a person working at a desk. They are wearing an orange long-sleeved shirt and are holding a black coffee cup with a white lid in their right hand. Their left hand is on a black keyboard. In the background, there are two computer monitors on stands. The image has a dark, semi-transparent overlay.

End of
Document

Samsung

SAMSUNG

Together for Tomorrow!
Enabling People

Education for Future Generations

©2022 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.