SAMSUNG

Samsung Innovation Campus

Artificial Intelligence Course

Fernando Martínez Plumed - fmartinez@dsic.upv.es

Together for Tomorrow!
Enabling People
Education for Future Generations

Calendar

Oct 7	Oct 8	Oct 9	Oct 10	Oct 11
18:00 - 21:00 **ML Supervised** (Unit 1)	*18:00 - 21:00* **ML Supervised** (Unit 1)	Bank Holiday	*16:00 - 21:00* **ML Supervised** (Units 2 & 3)	*16:00 - 21:00* **ML Supervised** (Unit 4, 5,6 & 7)
Oct 14	**Oct 15**	**Oct 16**	**Oct 17**	**Oct 18**
16:00 - 21:00 **ML Unservised** (Unit 1)	*16:00 - 21:00* **ML Unservised** (Unit 2)	*16:00 - 21:00* **Test &** **Project ML**	16:00 - 21:00 **Project ML**	16:00 - 21:00 **Project ML**
Oct 21	**Oct 22**	**Oct 23**	**Oct 24**	**Oct 25**
No classes **Project ML**	No classes **Project ML**	No classes **Project ML**	No classes **Project ML**	No classes **Project ML**
Oct 28				
16:00 - 18:30 **Project ML** Evaluation				

Chapter 5.

Machine Learning 1
- Supervised Learning

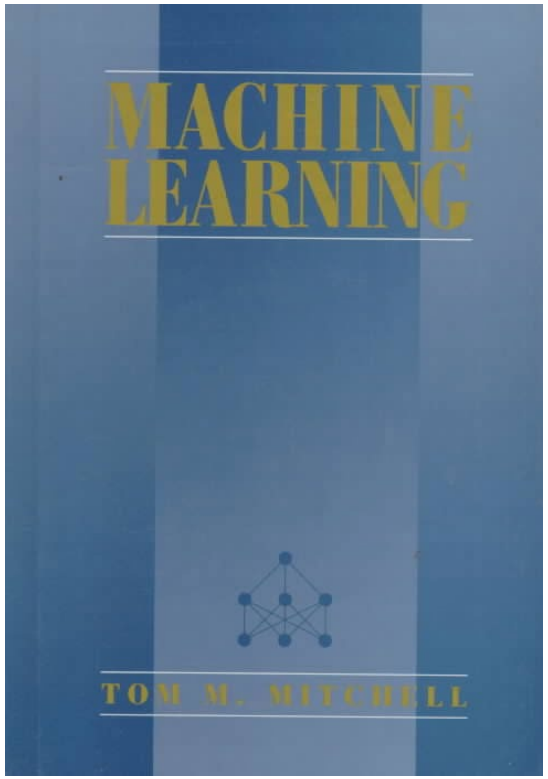
Artificial Intelligence Course

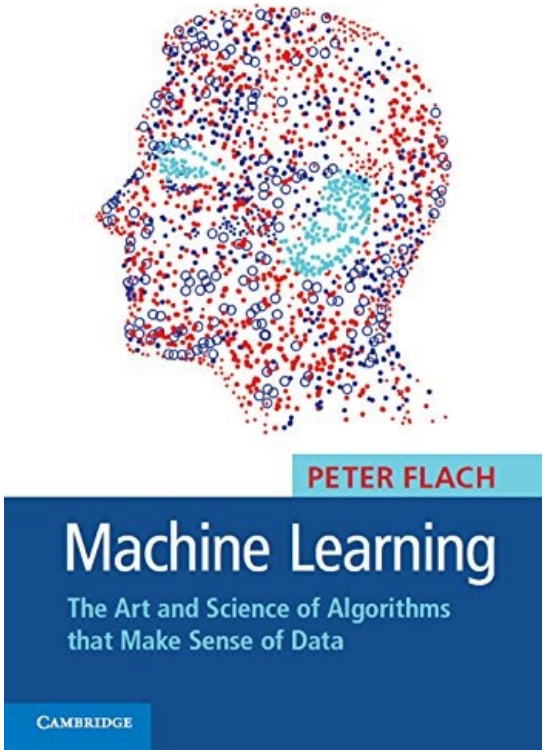
Chapter Description

Chapter objectives

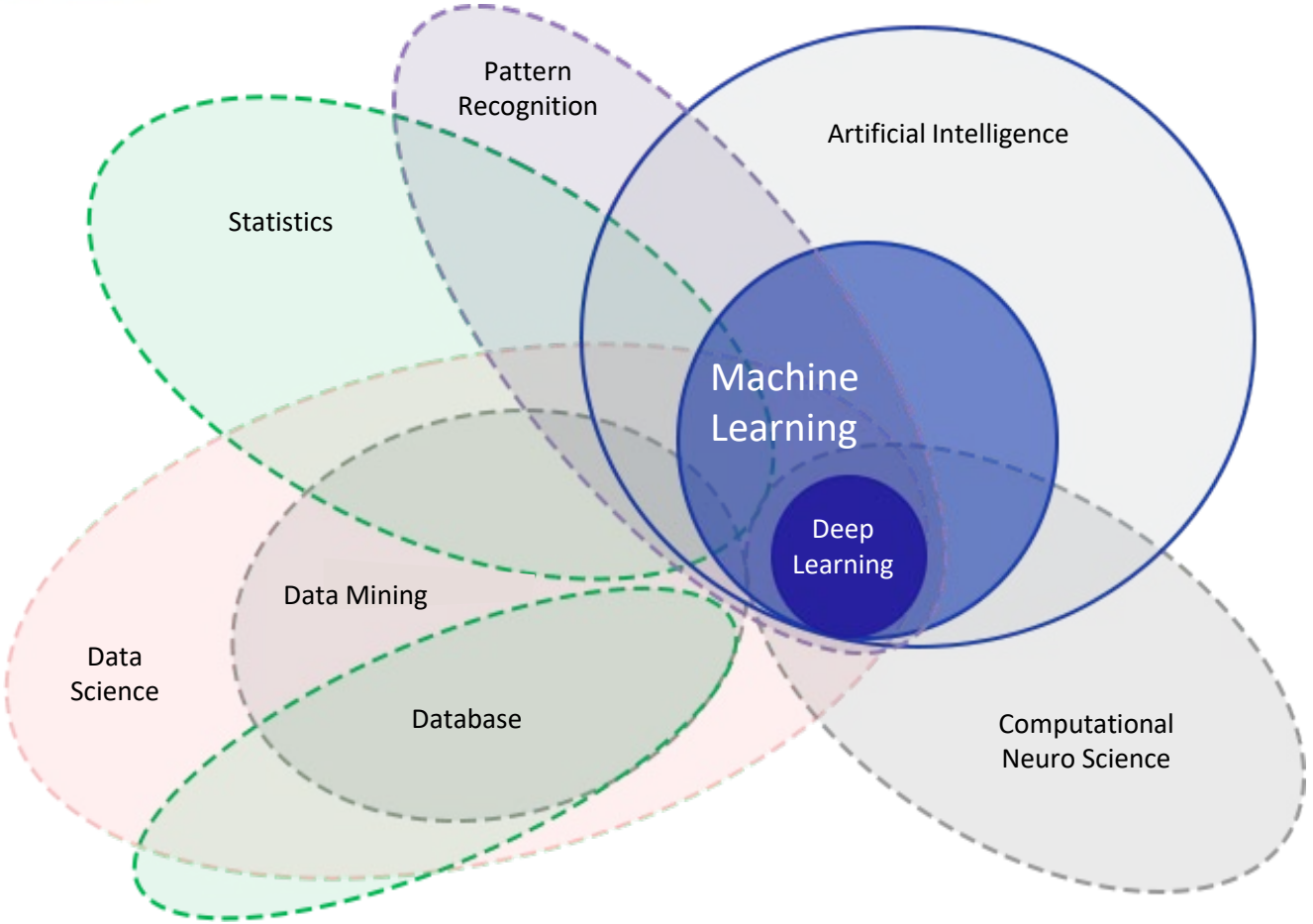
- ✓ Develop a solid grasp of the **fundamentals of machine learning-driven data analysis** and effectively oversee the entire process.
- ✓ Be able to **select and apply a machine learning algorithm that is the most suitable to the given problem** and perform hyperparameter tuning.
- ✓ Be able to **design, maintain, and optimize a machine learning workflow** for AI modeling by using structured data.

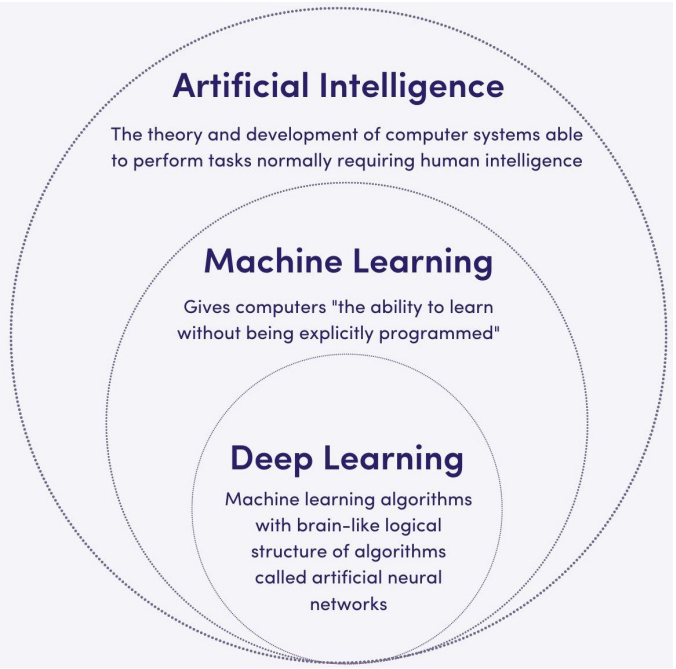
Chapter contents





http://www.cs.cmu.edu/~tom/mlbook.html





Connection among machine learning and different kinds of study

Unit 1.

Machine Learning Based Data Analysis

What is machine learning?

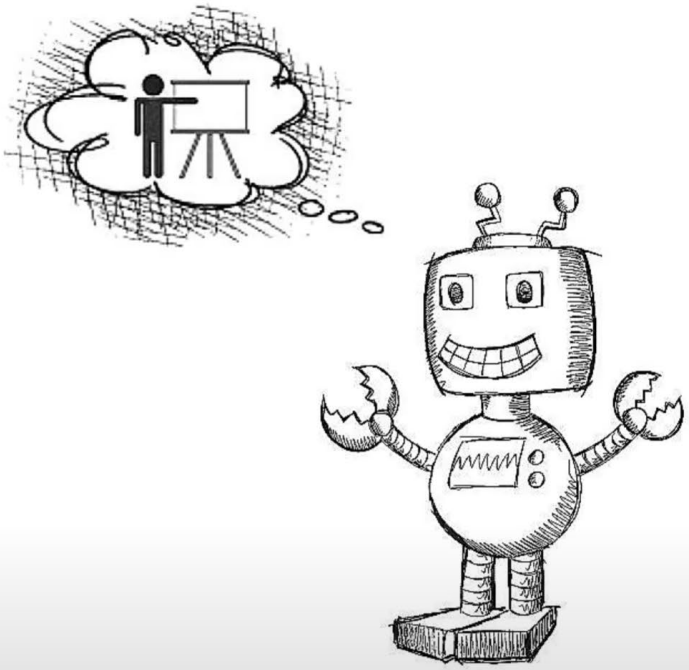
What is machine learning?

- Machine learning is a branch of artificial intelligence (AI) which involves the development of algorithms and statistical models that enable computer systems to learn from data and solve tasks without being explicitly programmed to do so.
- This is achieved through the use of various techniques such as **supervised learning**, **unsupervised learning** and **reinforcement learning**.

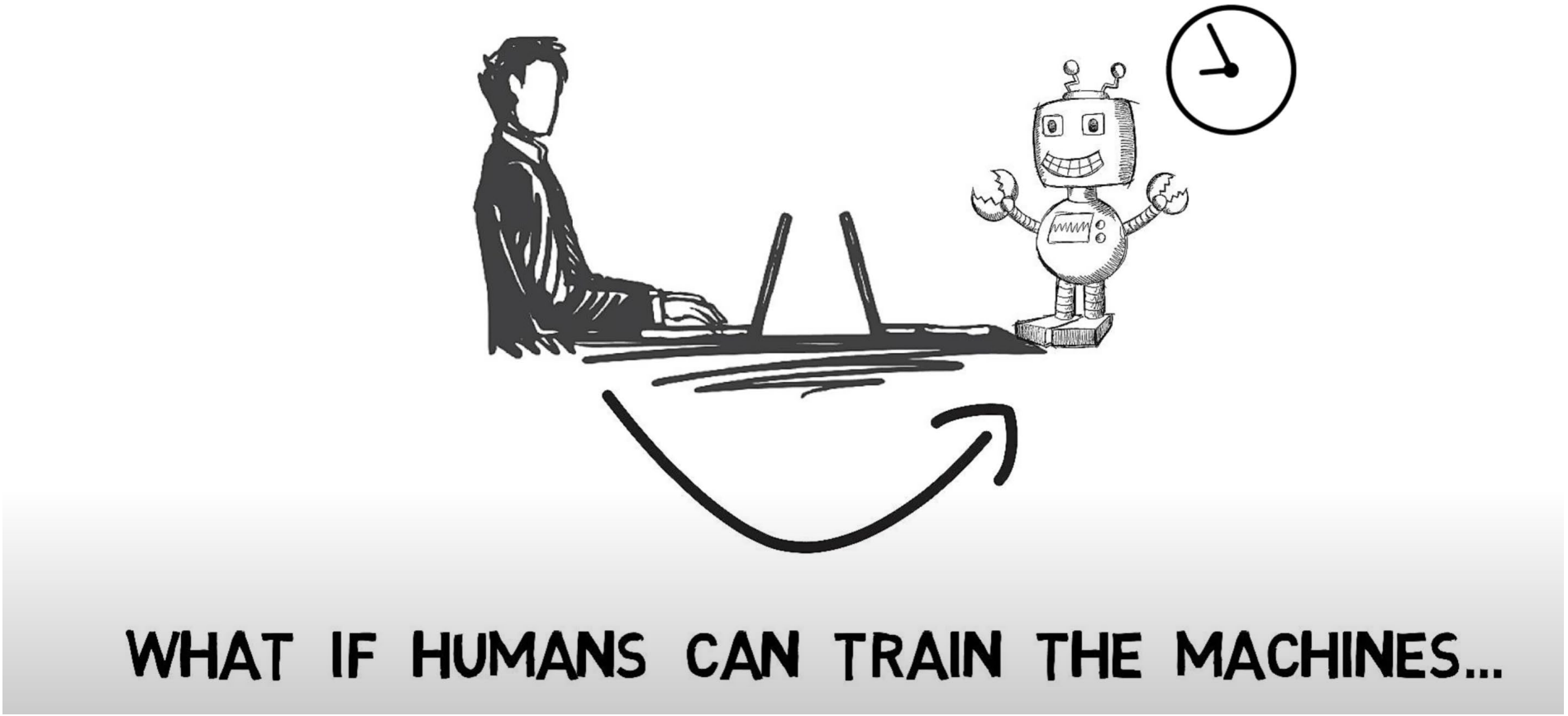
Why is machine learning important?

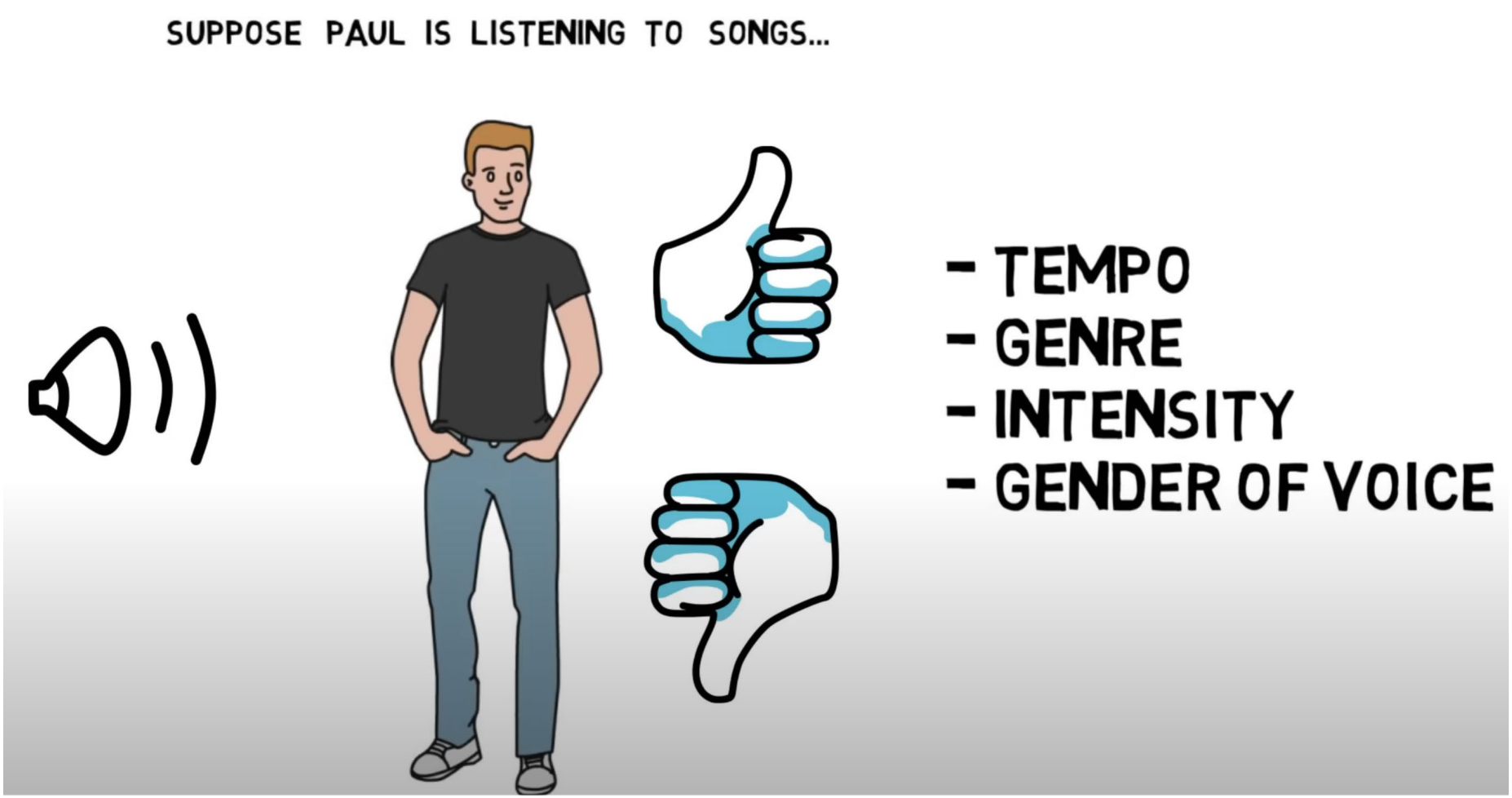
- ML enables computers to learn from data, **improving decision-making processes** and **creating intelligent systems** that enhance numerous aspects of **life** and **industry**.

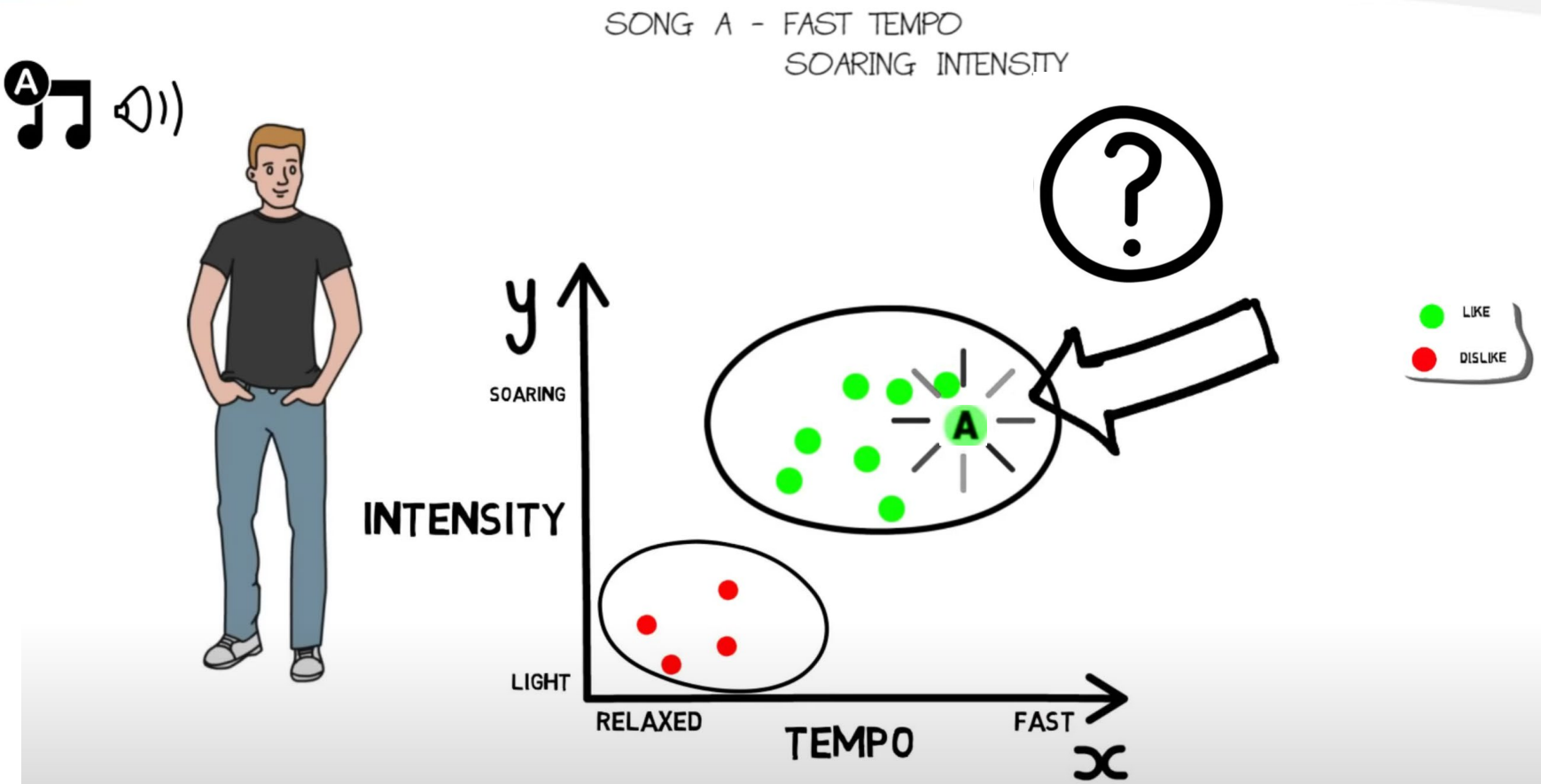
HUMANS LEARN FROM PAST EXPERIENCES

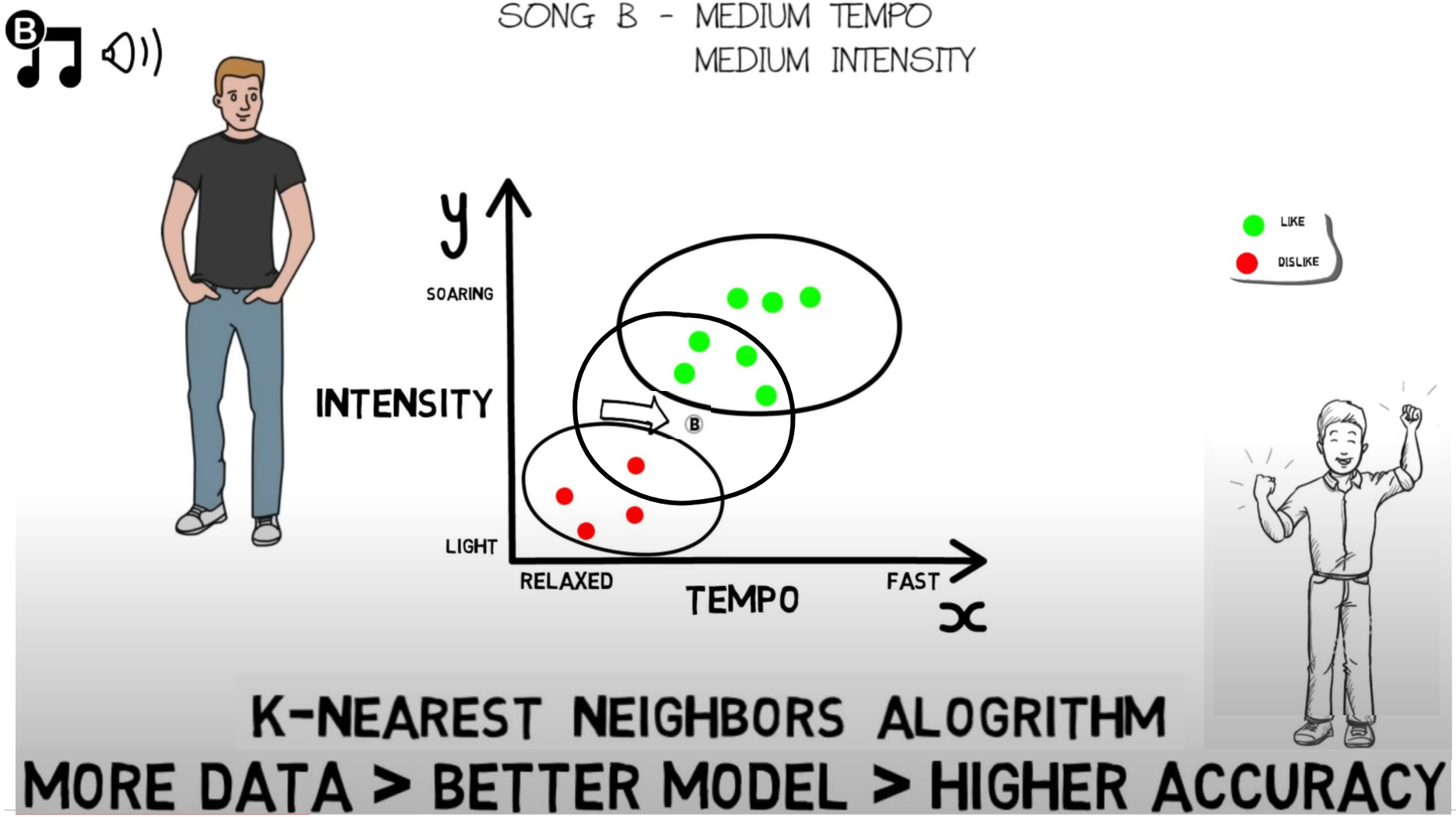


MACHINES FOLLOW INSTRUCTIONS GIVEN BY HUMANS


WHAT IF HUMANS CAN TRAIN THE MACHINES...

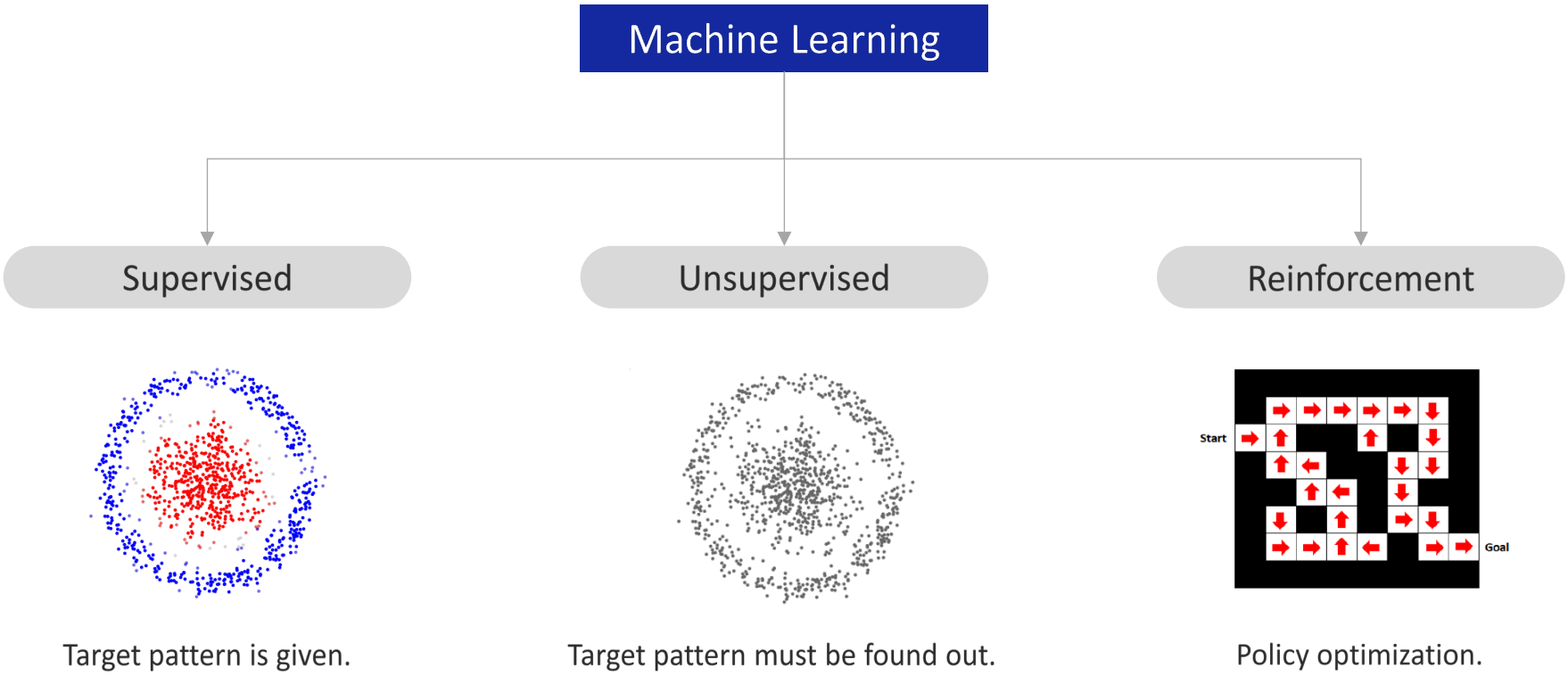

SUPPOSE PAUL IS LISTENING TO SONGS...
- TEMPO
- GENRE
- INTENSITY
- GENDER OF VOICE


SONG A - FAST TEMPO
SOARING INTENSITY
A
?
y
SOARING
INTENSITY
LIGHT
RELAXED
TEMPO
FAST
x
A
LIKE
DISLIKE


SONG B - MEDIUM TEMPO
MEDIUM INTENSITY
y
SOARING
INTENSITY
LIGHT
RELAXED
TEMPO
FAST
x
B
LIKE
DISLIKE
K-NEAREST NEIGHBORS ALOGRITHM
MORE DATA > BETTER MODEL > HIGHER ACCURACY

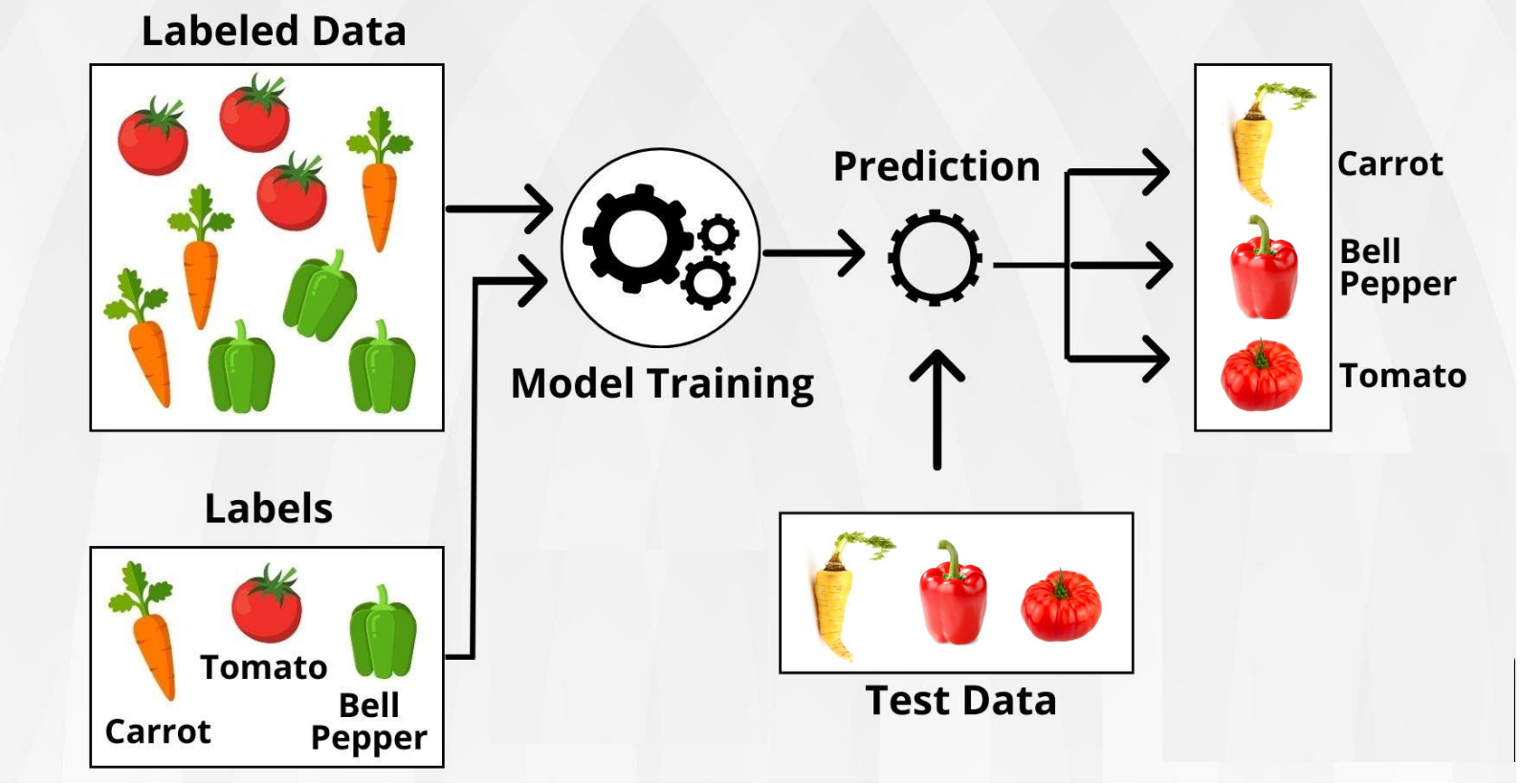
What are the different types of machine learning?

- There are three basic approaches: supervised learning, unsupervised learning and reinforcement learning. The type of algorithm data scientists choose to use depends on what type of data they want to predict.



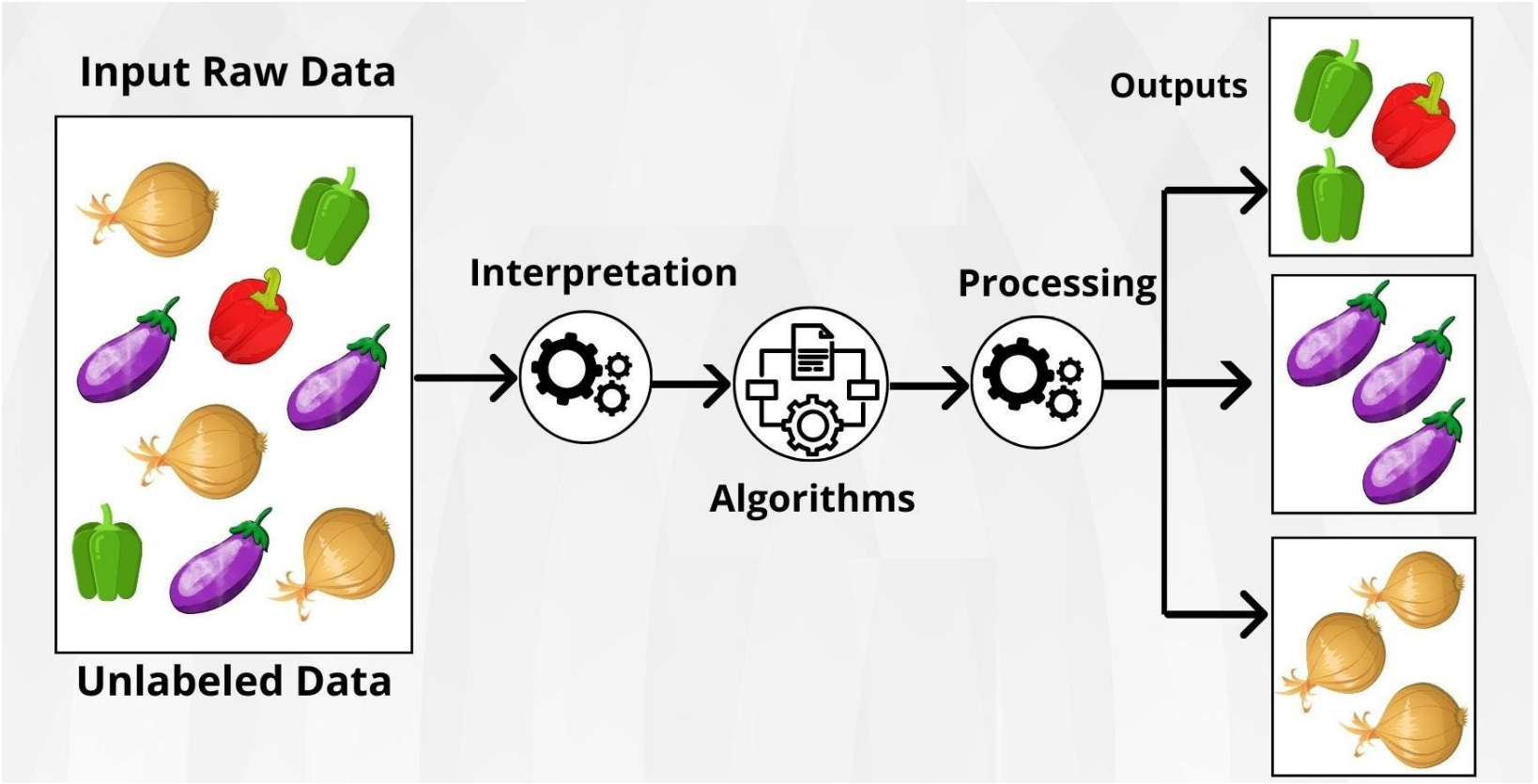
What are the different types of machine learning?

- There are three basic approaches: **supervised learning**, unsupervised learning and reinforcement learning. The type of algorithm data scientists choose to use depends on what type of data they want to predict.



What are the different types of machine learning?

- There are three basic approaches: supervised learning, **unsupervised learning** and reinforcement learning. The type of algorithm data scientists choose to use depends on what type of data they want to predict.



Examples

- **AGENT in a BANK:**
  - **Should I grant credit to this customer?**

Historical Data/Experience

Id	Credit-p (years)	Credit-a (euros)	Salary (euros)	Own House	Defaulter accounts	…	Returns-credit
101	15	60.000	2.200	yes	2	…	**no**
102	2	30.000	3.500	yes	0	…	**yes**
103	9	9.000	1.700	yes	1	…	**no**
104	15	18.000	1.900	no	0	…	**yes**
105	10	24.000	2.100	no	0	…	**no**
…	…	…	…	…	…	…	…

Machine Learning

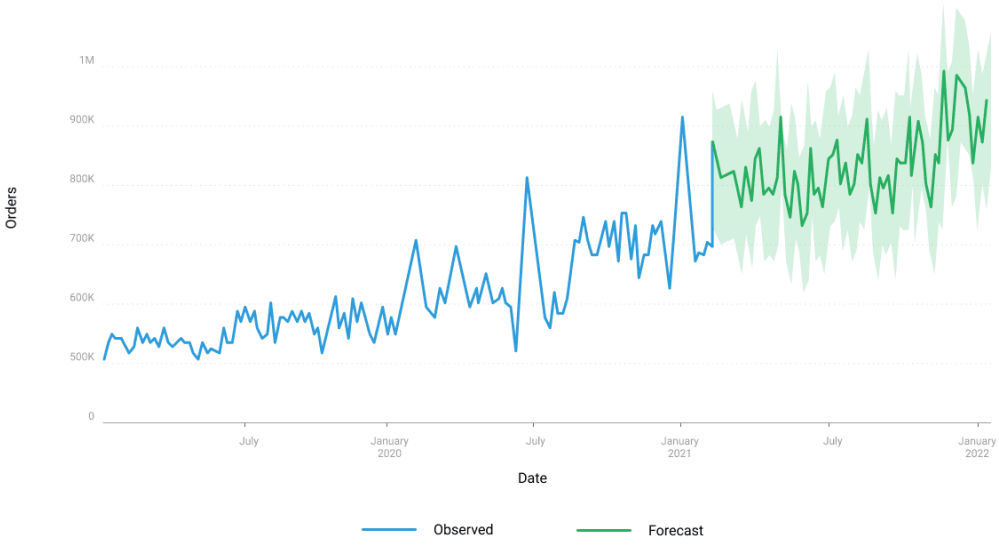
Model / Patterns

If Defaulter-accounts > 0 **then** Returns-credit = no

If Defaulter-accounts = 0 **and** [(Salary > 2.500) **or** (credit-p > 10)] **then** Returns-credit = yes

Examples

- **SALES MANAGER of a SALES COMPANY:**
  - **How many flat-screen TVs are expected to be sold next month?**



Historical Data/Experience

PRODUCT	Month–12	...	Month–4	Month–3	Month–2	Month–1	Month
Flat TV 30'	20	...	52	14	139	74	?
BlueRay	11	...	43	32	26	59	?
PlayStation	50	…	61	14	5	28	?
Five star fridge	3	…	21	27	1	49	?
Three star fridge	14	...	27	2	25	12	?
…	…	…	…	…	…	…	…

Machine Learning

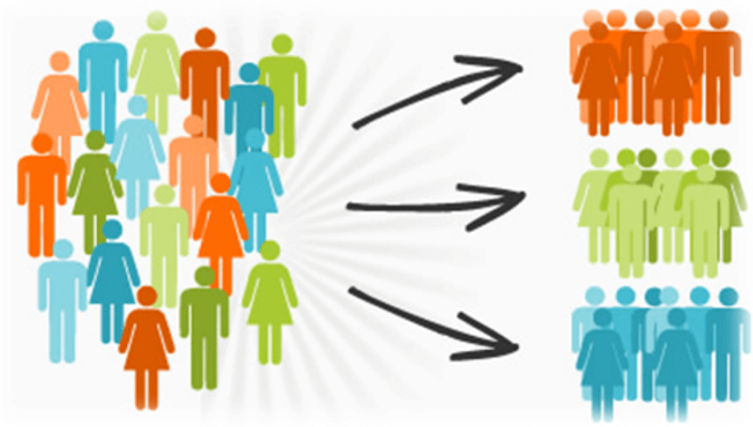
Model / Patterns

Linear Model: TV Sales for Next Month:

$$V(\text{Month})_{\text{flatTV}} = 0.62 \cdot V(\text{Month-1})_{\text{flatTV}} + 0.33 \cdot V(\text{Month-2})_{\text{flatTV}} + 0.12 \cdot V(\text{Month-1})_{\text{BlueRay}} - 0.05$$

Examples

- **HR MANAGER in a COMPANY:**
  - **What types of employees do I have?**



Historical Data/Experience

Id	Salary	Married	Car	Children	Rent/ Owner	Union	Off sick/year	Work years	Gender
1	10000	yes	no	0	Rent	no	7	15	M
2	20000	no	yes	1	Rent	yes	3	3	F
3	15000	yes	yes	2	Owner	yes	5	10	M
4	30000	yes	yes	1	Rent	no	15	7	F
5	10000	yes	yes	0	Owner	yes	1	6	M
6	40000	no	yes	0	Rent	yes	3	16	F
7	25000	no	no	0	Rent	yes	0	8	M
8	20000	no	yes	0	Owner	yes	2	6	F
15	8000	no	yes	0	Rent	no	3	2	M
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

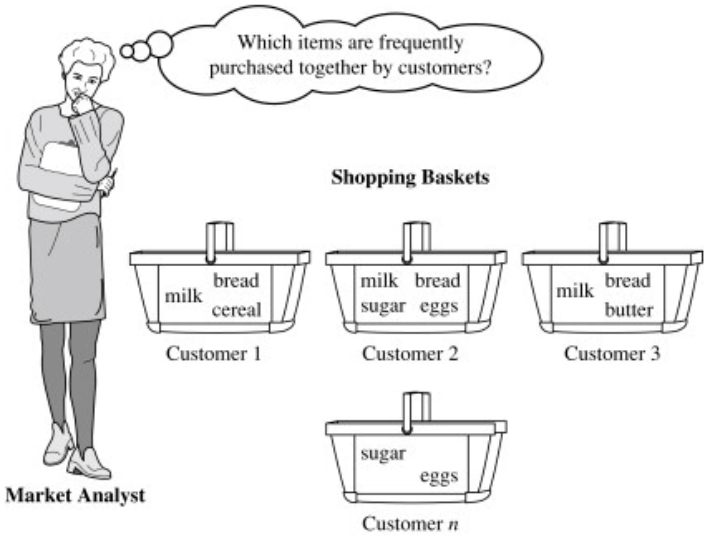
Machine Learning

Model / Patterns

- **Group 1:** Without children and in a rented house. Low participation in unions. Many days off sick.
- **Group 2:** Without children and with car. High participation in unions. Few days off sick. More women and in rented houses.
- **Group 3:** With children, married and with car. More men and usually house owners. Low participation in unions.

Examples

- **SUPERMARKET MANAGER:**
  - **When you buy eggs, do you usually buy oil as well?**



Historical Data/Experience

BasketId	Eggs	Oil	Nappies	Wine	Milk	Butter	Salmon	Endive	...
1	yes	yes	no	yes	no	yes	yes	yes	...
2	no	yes	no	no	yes	no	no	yes	...
3	no	no	yes	no	yes	no	no	no	...
4	no	yes	yes	no	yes	no	no	no	...
5	yes	yes	no	no	no	yes	no	yes	...
6	yes	no	no	yes	yes	yes	yes	no	...
7	no	no	no	no	no	no	no	no	...
8	yes	yes	yes	yes	yes	yes	yes	no	...
...	...	...	...	...	...	...	...	...	...

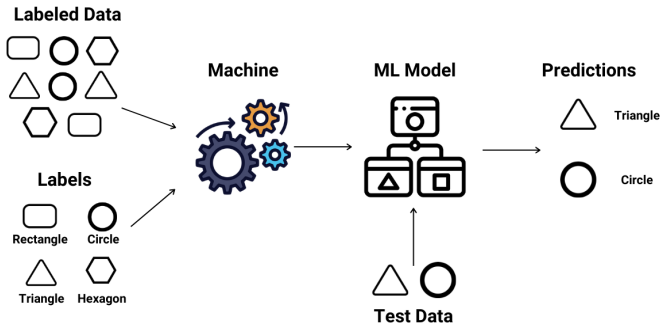
Machine Learning

Model / Patterns

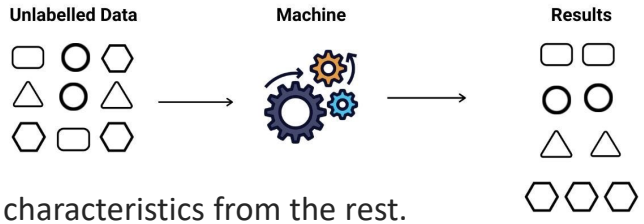
Eggs → Oil : Confidence = 75%, Support = 37%

What are the different types of machine learning?

- **Supervised/predictive task:** The problem is presented with example inputs and their desired outputs and the goal is to learn a **general rule that maps inputs to outputs**. It answers questions about data that we do not know (usually, but not always, future).
  - **Classification**: Output is categorical
  - **Regression**: Output is numerical
    - What will sales be next year?
    - Is this transaction fraudulent?
    - What type of insurance is customer X most likely to take out?

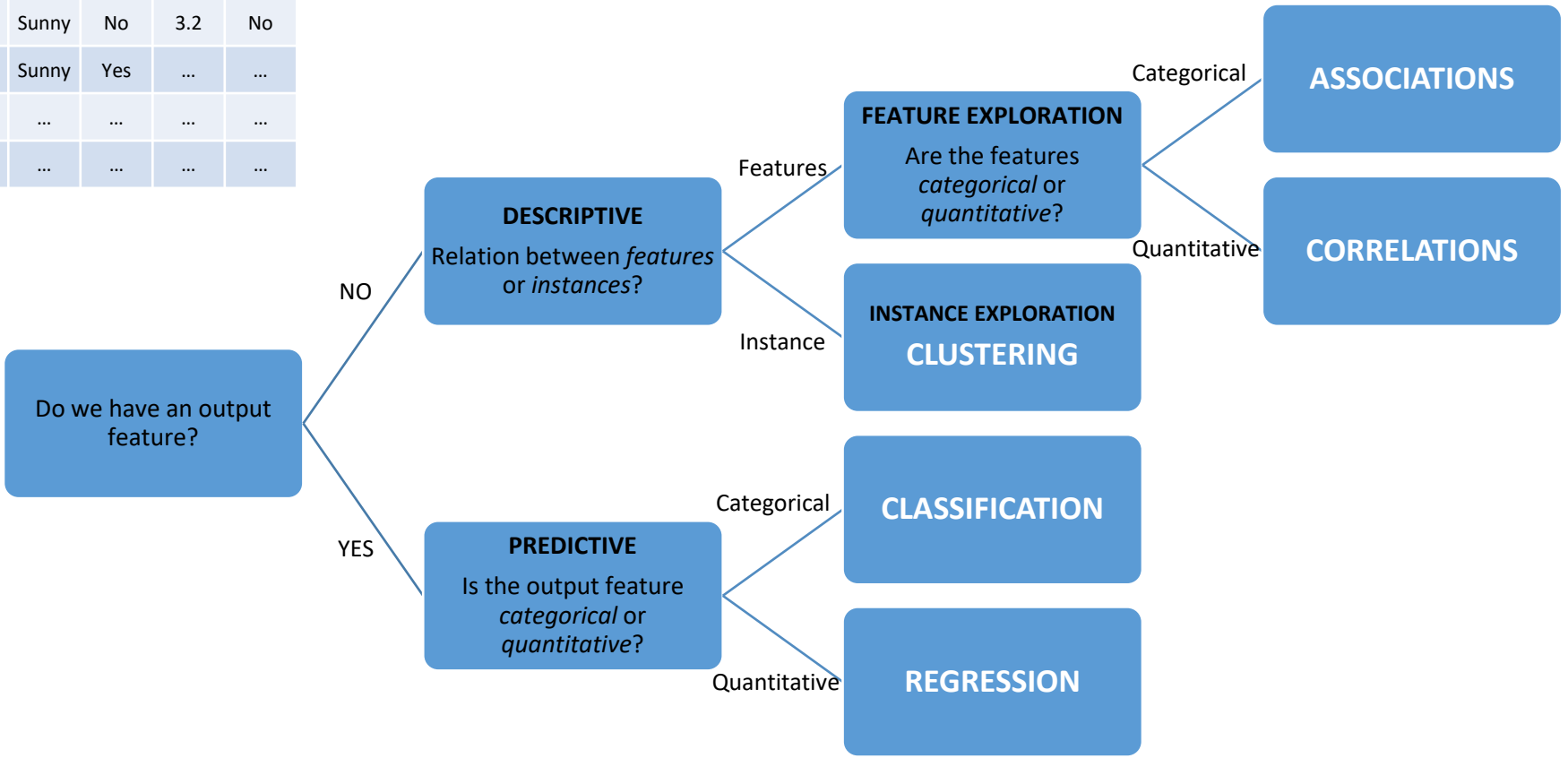


- **Unsupervised/descriptive task: No labels are given** to the learning algorithm, leaving it on its own to find **structure in its input**. It provides information about the relationships between data and their characteristics. It generates information of the type.
  - **Clustering**, **association rules**…
    - Customers who buy nappies tend to buy beer.
    - Smoking and alcohol are the most important factors in disease Y.
    - Customers without television and with bicycles have very different characteristics from the rest.

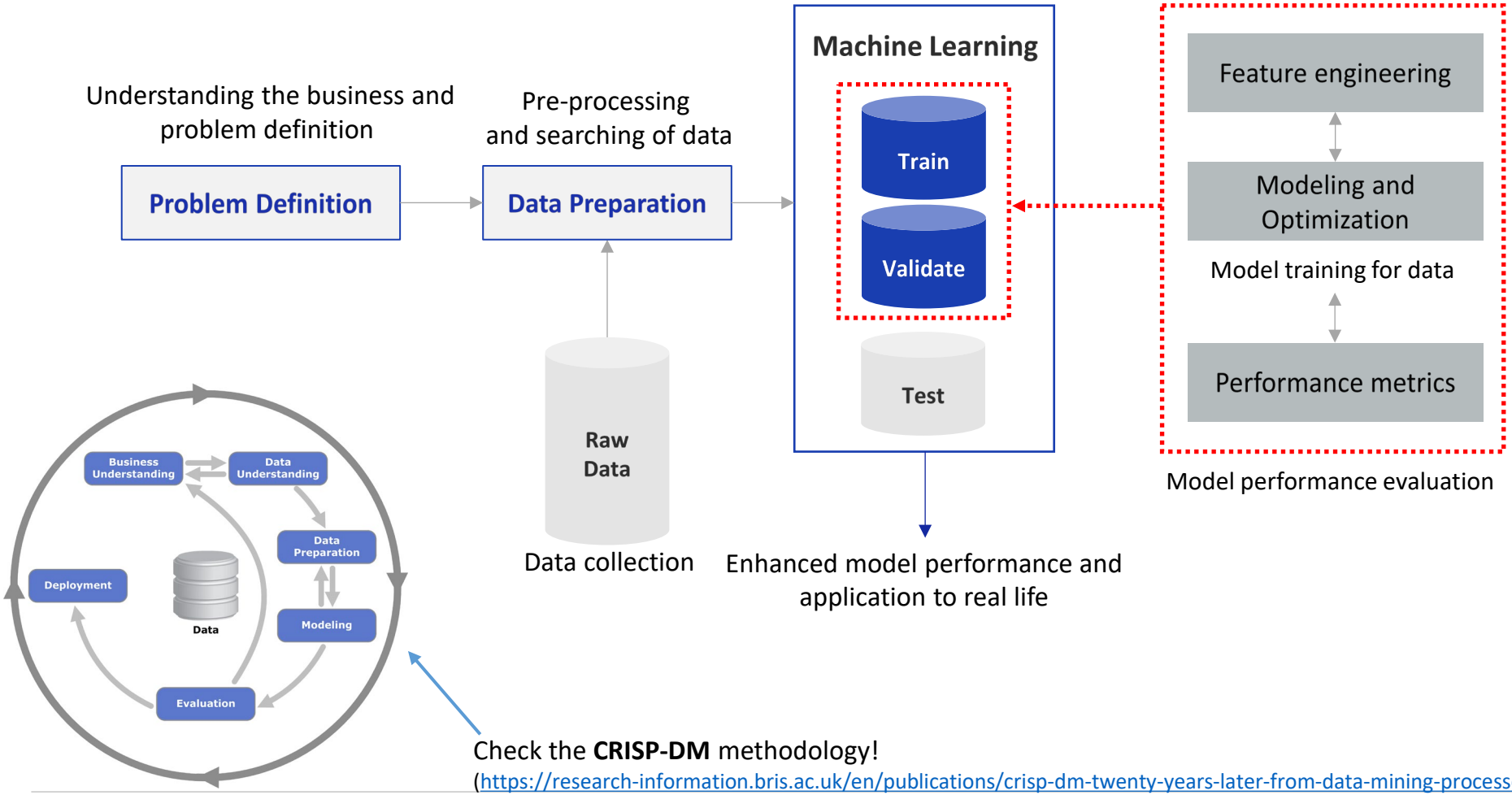


- **Reinforcement learning task** : A computer program interacts with a dynamic environment in which it must perform a certain goal: Driving a vehicle or videogames.

Att1	Att2	Att3	Att4	Label
3.2	Rainy	Yes	4.5	Yes
-4.8	Sunny	No	3.2	No
2.1	Sunny	Yes	...	...
...	...	...	...	...
...	...	...	...	...



Machine learning workflow



Check the **CRISP-DM** methodology!
(https://research-information.bris.ac.uk/en/publications/crisp-dm-twenty-years-later-from-data-mining-processes-to-data-sc)

Machine learning types:

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

Machine learning types:

TECHNIQUE	PREDICTIVE / SUPERVISED		DESCRIPTIVE / UNSUPERVISED		
	Classification	Regression	Clustering	Association rules	Other (factorial, correl, scatter)
Neural Networks	✓	✓	✓		
Decision Trees	✓	✓	✓		
Kohonen			✓		
Linear regression (local, global), exp..		✓			
Logistic Regression	✓				
Kmeans	✓		✓		
A Priori (associations)				✓	
factorial analysis, multivariate analysis					✓
CN2	✓				
K-NN	✓		✓		
RBF	✓				
Bayes Classifiers	✓	✓			

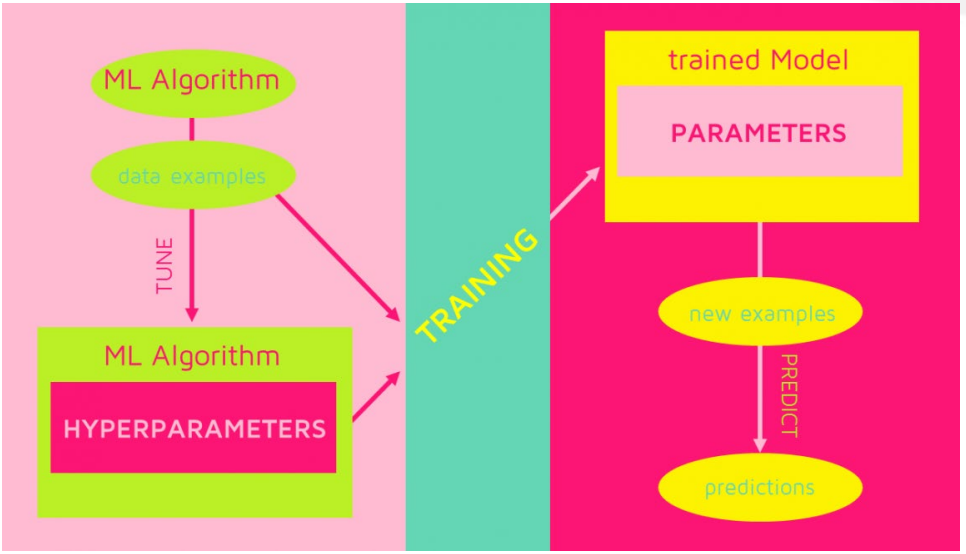
Parameters vs. Hyperparameters

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.



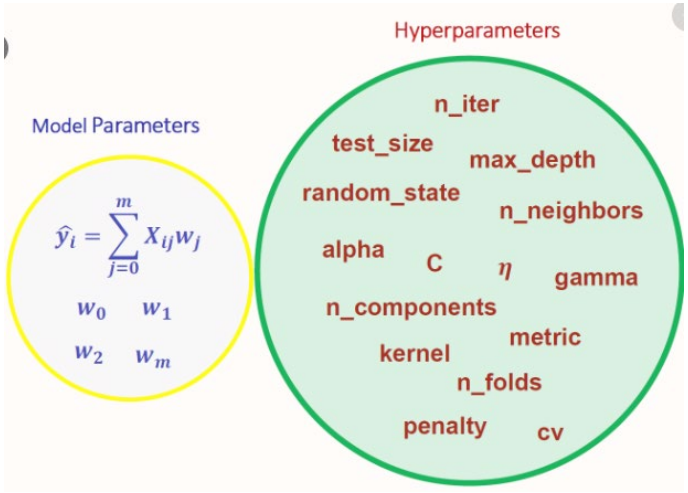
Hyperparameters

- **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

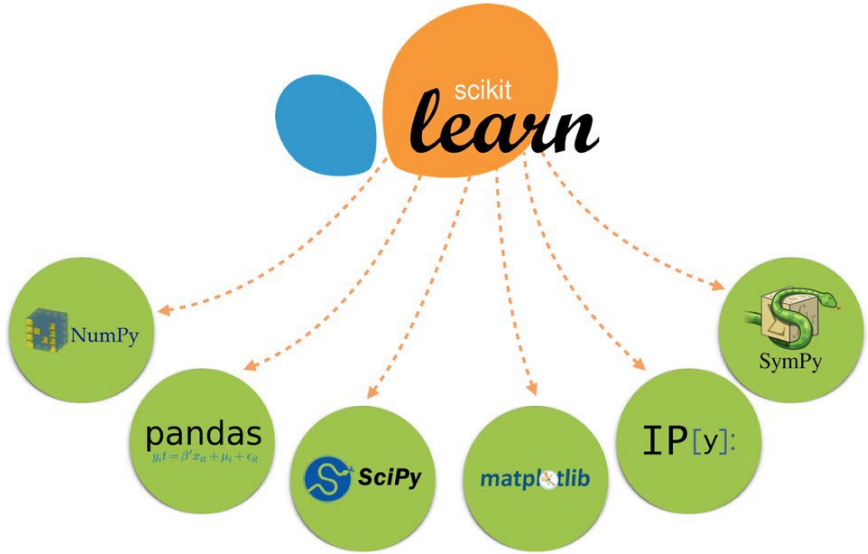
- 1.1. What is machine learning?
- 1.2. Phyton scikit-learn library for machine learning
- 1.3. Preparation and division of data set
- 1.4. Data pre-processing for making a good training data set
- 1.5. Practicing to find an optimal method to solve problems with scikit-learn

Features of the scikit-learn library

Facade is a structural design pattern that provides a simplified interface to a library.

Features

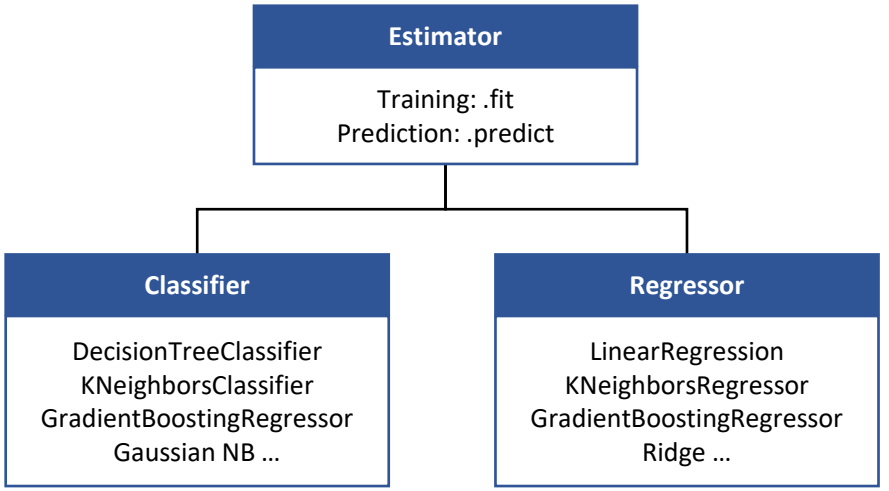
- Integrated library interface by applying the **facade** design pattern
- Installed with various kinds of **machine learning algorithms**, **model selection** and **data pre-processing** functions
- Simple and **efficient** tool to analyze predicted data
- Based on **NumPy**, **SciPy** and **matplotlib**
- 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.


```
from sklearn import tree
t = tree.DecisionTreeClassifier(criterion="entropy")

t = t.fit(train_attributes, train_labels)          Build the decision tree

t.predict(example_attributes)                      Predict a new example
```



Scikit-Learn Library

About the Scikit-Learn library

- To import a machine learning algorithm as class:

  from sklearn. <family> import <machine learning algorithm> **Ex** from sklearn.linear_model import LinearRegression
- Hyperparameters are specified when the ML object is instantiated **Ex** myModel = KNeighborsClassifier(n_neighbors=10)
- To train a **supervised** learning model: **Ex** myModel.fit(X_train, Y_train)
- To train a **unsupervised** learning model: **Ex** myModel.fit(X_train)
- To **predict** using an already trained model: **Ex** myModel.predict(X_test)
- To import a **preprocessor** as class: **Ex** from sklearn.preprocessing import <a preprocessor>
- To **split the dataset** into a training set and a testing set:

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

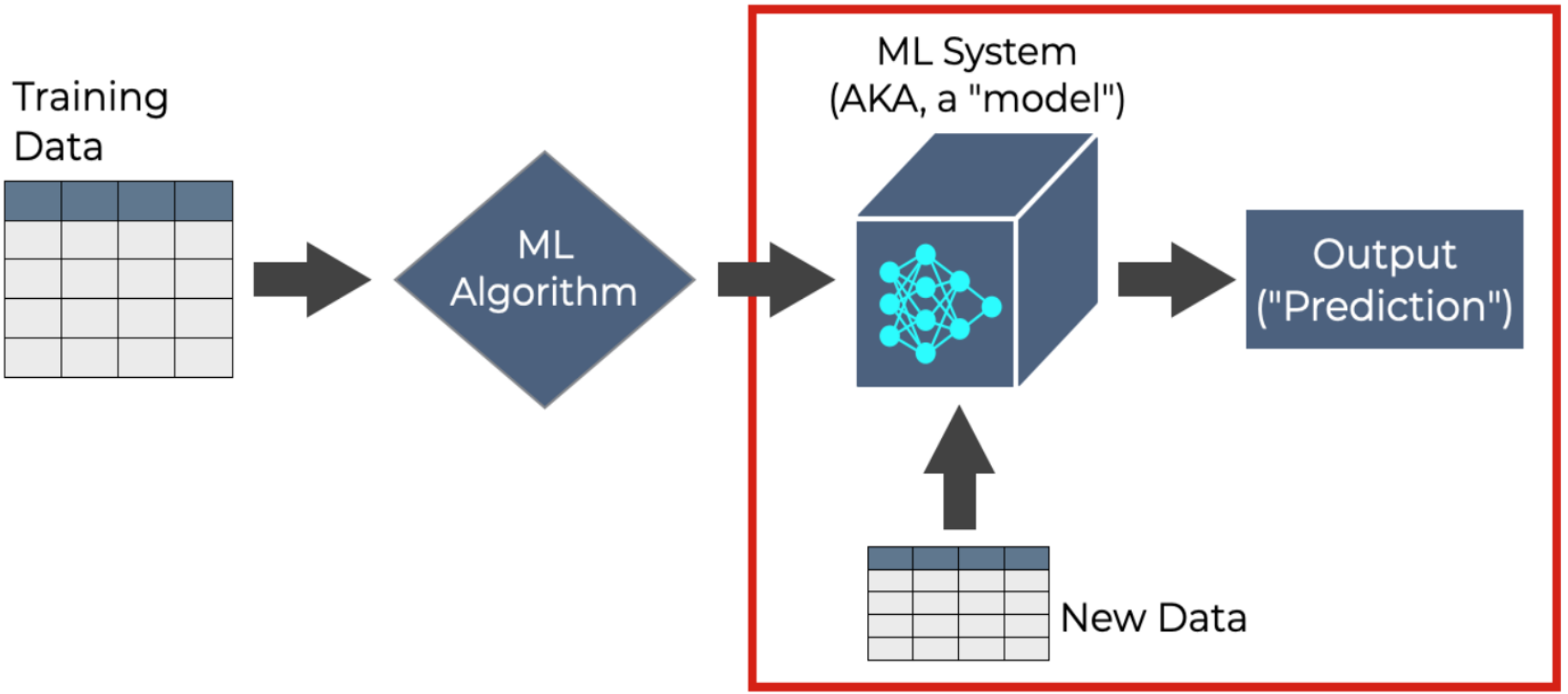
- To calculate a performance metric (accuracy): **Ex** metrics.accuracy_score(Y_test, Y_pred)
- To cross validate and do hyperparameter tuning at the same time:

Ex
```
myGridCV = GridSearchCV(estimator, parameter_grid, cv=k)
myGridCV.fit(X_train, Y_train)
```

With Scikit-Learn, we can easily train a model and make predictions.

Also, with Scikit-Learn calculating performance metrics, hyperparameter tuning, etc. are a breeze.

Practicing scikit-learn



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

```
import pandas as pd
```

```
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

- mean radius
- mean texture
- mean perimeter
- mean area
- mean smoothness
- mean compactness
- mean concavity
- mean concave points
- mean symmetry
- mean fractal dimension
- radius error
- texture error
- perimeter error
- area error
- smoothness error
- compactness error
- concavity error
- concave points error
- symmetry error
- fractal dimension error
- worst radius
- worst texture
- worst perimeter
- worst area
- worst smoothness
- worst compactness
- worst concavity
- worst concave points
- worst symmetry
- worst fractal dimension

Line 5

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

Practicing scikit-learn

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

Out[6]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
564	0
565	0
566	0
567	0
568	1

569 rows × 1 columns

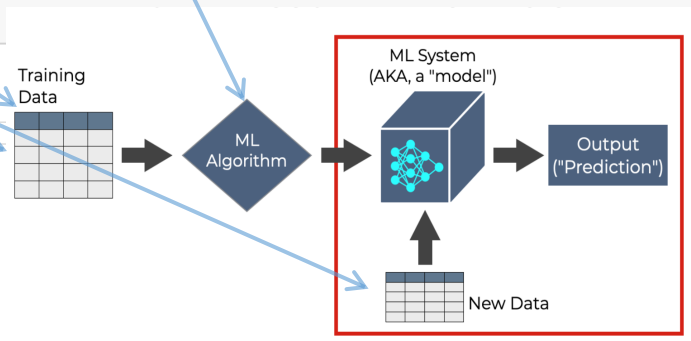
Line 6

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

Practicing scikit-learn

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

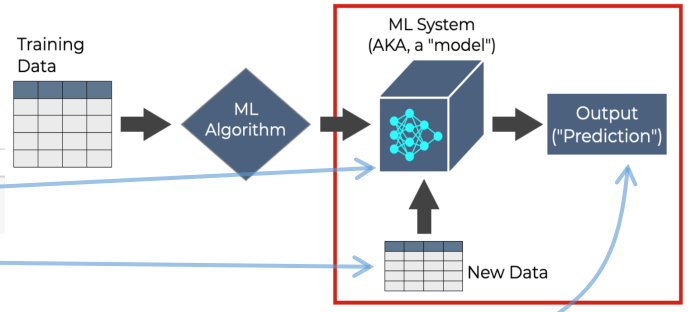
- Instancing the estimator and hyperparameter setting
- Model initialization by using the entropy for branching

fit

- Use the fit method with instance estimator for training. Send the training data and label data together as an argument to 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, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
               0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
               1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
               0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
               1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
               0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
               1, 1, 1, 0, 0, 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 [15]:
```
model.fit(X_train, y_train)
pred = model.predict(X_test)
print(pred)
```

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

In [16]:
```
print(y_test)
```

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

Practicing scikit-learn

```
In [27]: sum(pred == y_test)/len(y_test)

Out[27]: 0.958041958041958
```

```
In [28]: from sklearn.metrics import accuracy_score
         accuracy_score(pred,y_test)

Out[28]: 0.958041958041958
```

- It showed 95.8% accuracy.
- Further steps for increasing the 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 of standardization is z-transformation, and the standardized value is also referred to as z-score.
  - Standardization is widely used in data pre-processing in general and the following is the equation.

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

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

transform

- Feature processing is done with 'transform' to return the result.

In [3]:
```
1 from sklearn.preprocessing import StandardScaler
2
3 print(X_train)  # Result check before pre-processing
4
5 scaler = StandardScaler()
6 scaler.fit(X_train)
7 X_train = scaler.transform(X_train)    # Pre-processing – Apply scaling
8
9 X_train    # Result check after pre-processing
```

```
[[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]]
```

Result check before pre-processing

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]])
```

Result check after pre-processing

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

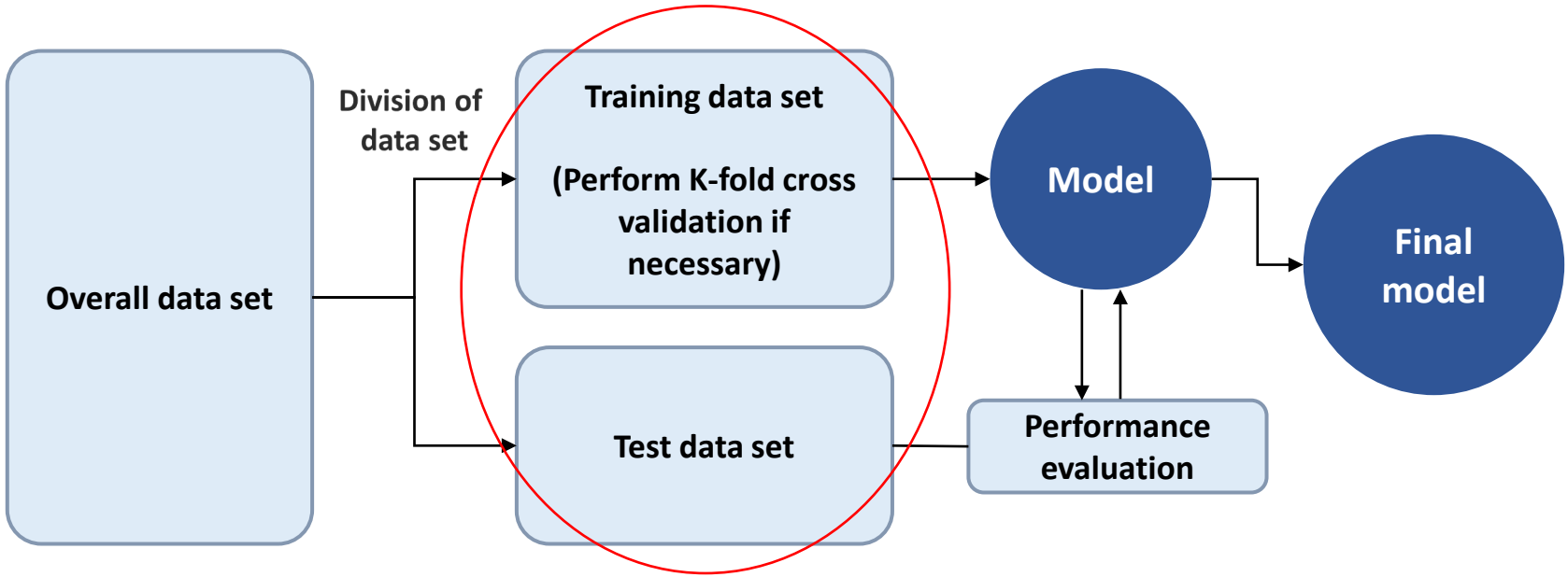
Preparation and division of data set

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 training data set and test data set for machine learning technique application** according to the analysis purpose and data set features.
- Be able to divide training data set and validation data set and decide 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 set and and perform sampling by considering the prediction results based on data features and classified variable distribution.
- Be able to analyze differences of 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.

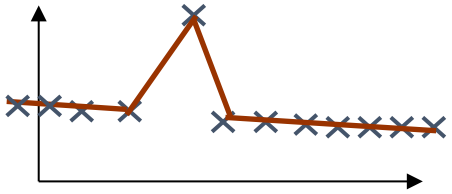


Machine learning modeling process through division of training and test data sets

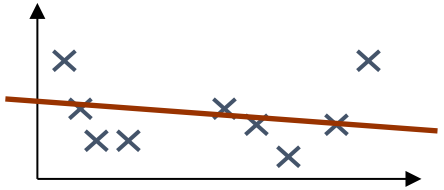
GOLDEN RULE: Never use the same example for training the model and evaluating it!!

Overfitting and generalization of modeling

- If we use all data to train the models and evaluate them, we get overoptimistic models:
  - **Over-fitting**:

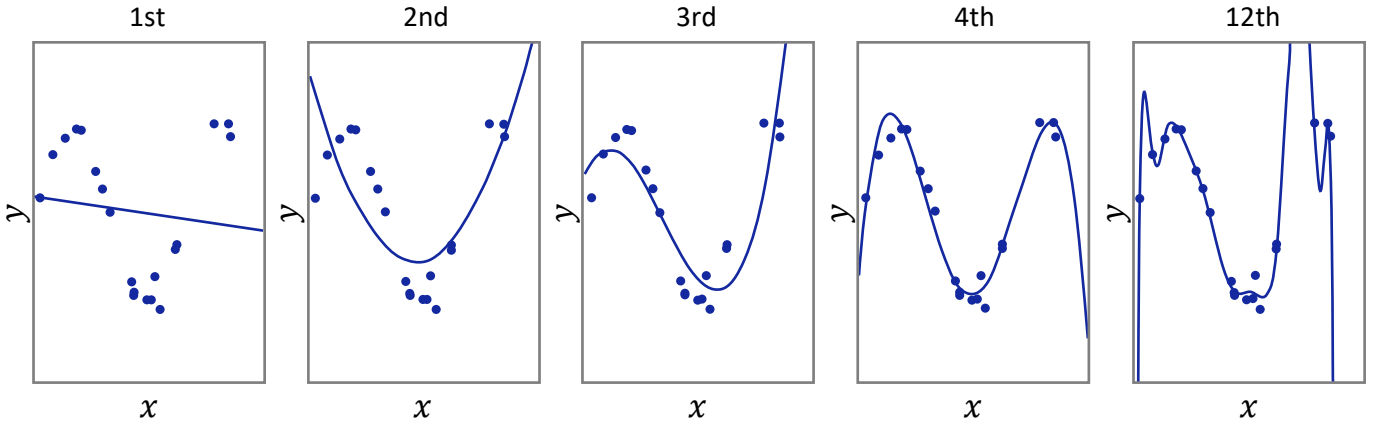


- If we try to compensate by generalising the model (e.g., pruning a tree), we may get:
  - **Under-fitting**:



- The chance that the patterns from the training data and new data to perfectly accord is extremely low.
- To prevent such issues, the data set is generally divided into **training data** set and **test data** set.

Overfitting and generalization of modeling



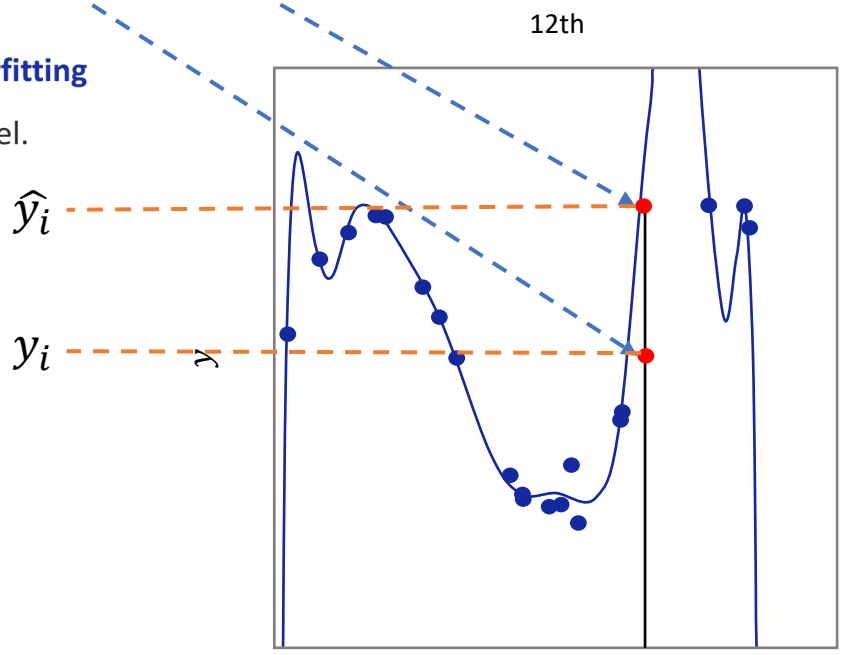
Overfitting and underfitting

- Even if machine learning finds the optimal solution in data distribution, a wide margin of error occurs, and this is because the model has a small capacity. Such a phenomenon is referred to as underfitting, and 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 12^{th} order polynomial.
- The model capacity got larger, and there are 13 parameters for estimation.

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

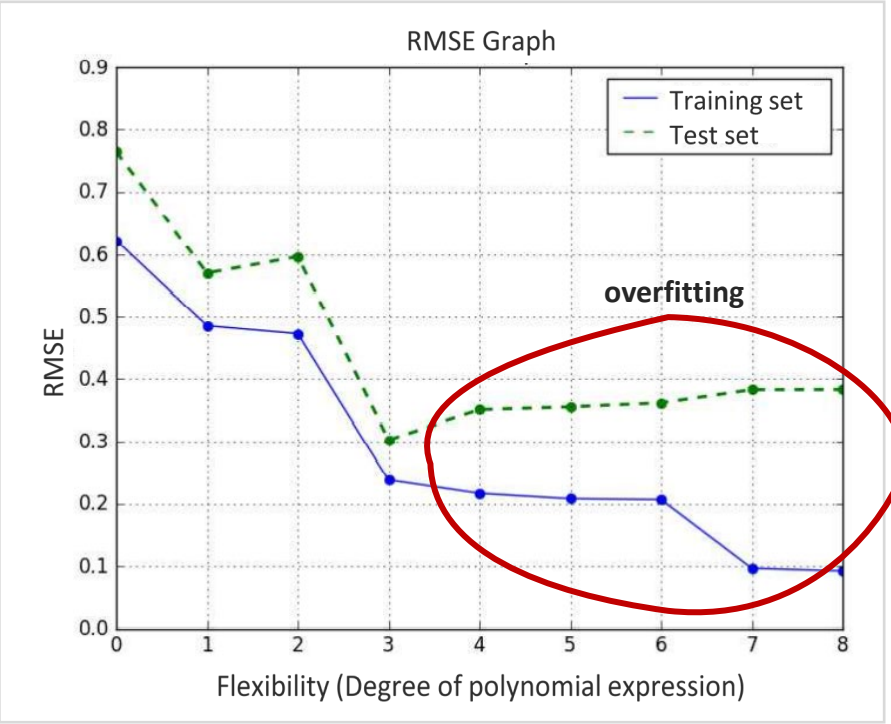
Overfitting and generalization of modeling

- 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



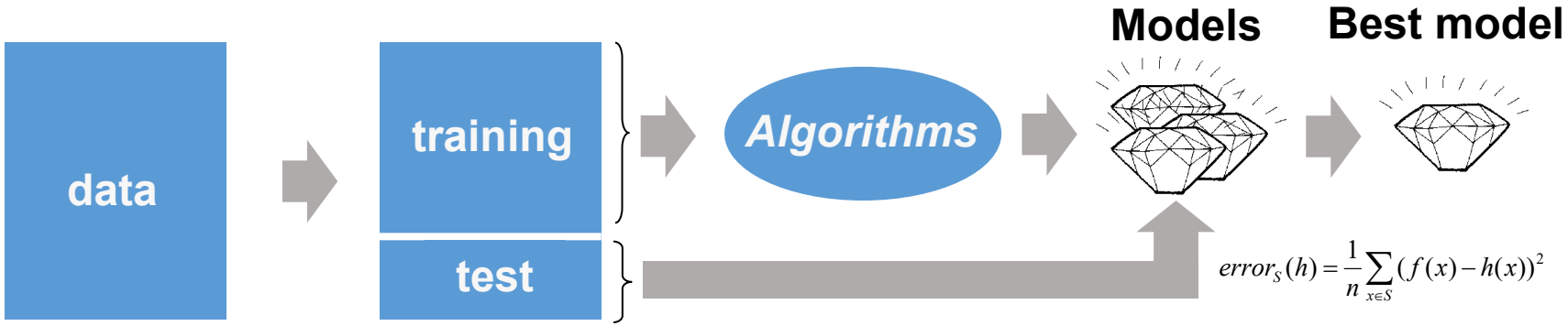
Comparison of the difference between the root mean squared errors of training and test data

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

- As the flexibility of machine learning technique increases (followed by increased order of polynomial), the root mean squared error of the **training data** set shows a monotone decreasing trend,
- The root mean 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 4^{th} order polynomial.
- If there was no root mean squared error index calculated with test data, the root mean squared error of the training data would decline as the order of polynomial rises to lead to selecting an overfitting model.
- Thus, test data is required.

Method and process of data set division

- Too much training data: **poor evaluation**
- Too much test data: **poor training**
- Can we have more training data and more test data without breaking the golden rule?
  - Repeat the experiment!
    - **Cross validation**: Data is split in n folds of equal size.
    - **Hold-out**: special case with as many folds as examples.



In [1]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data=load_breast_cancer()
X_train, X_test, y_train, y_test=train_test_split(data.data, data.target, random_state=42)
```

Cross-Validation

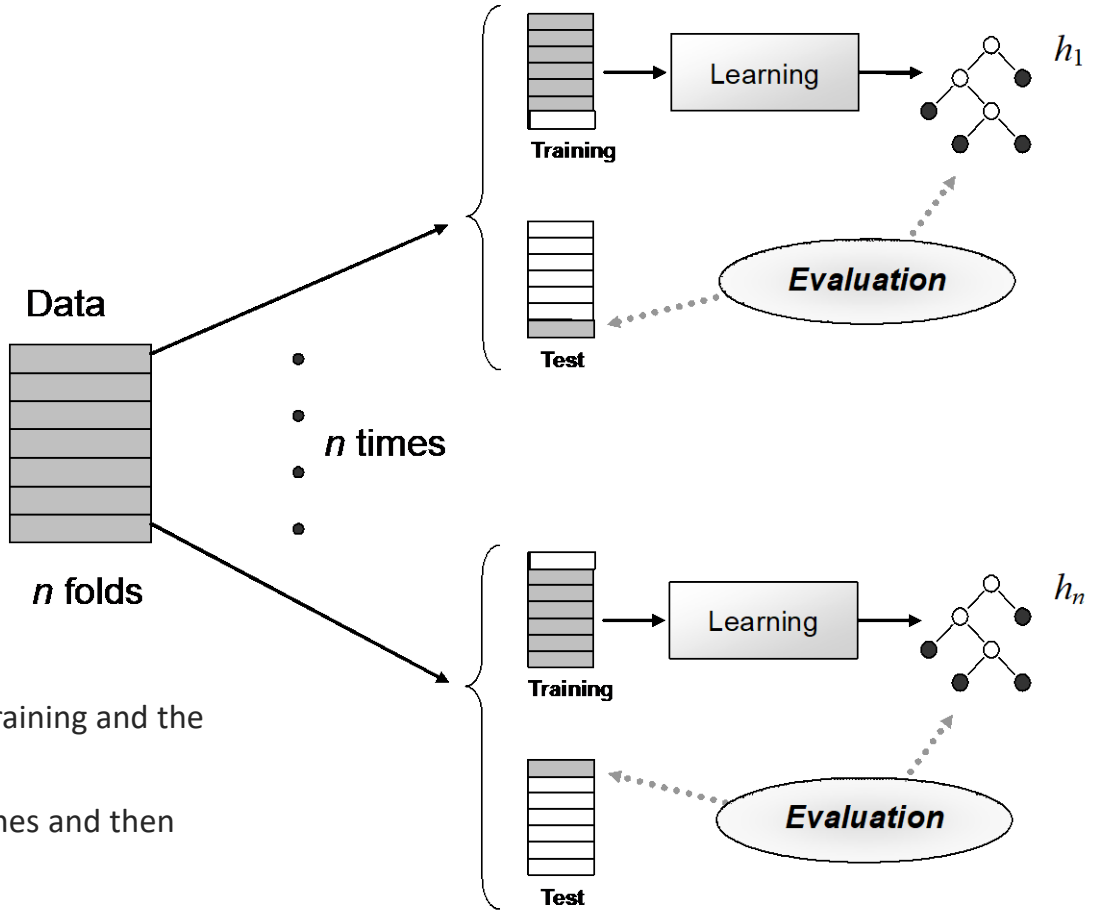
1) Randomly divide the data set into training and test sets.

2) Adjust the model on the training set.

3) Test model on test set.

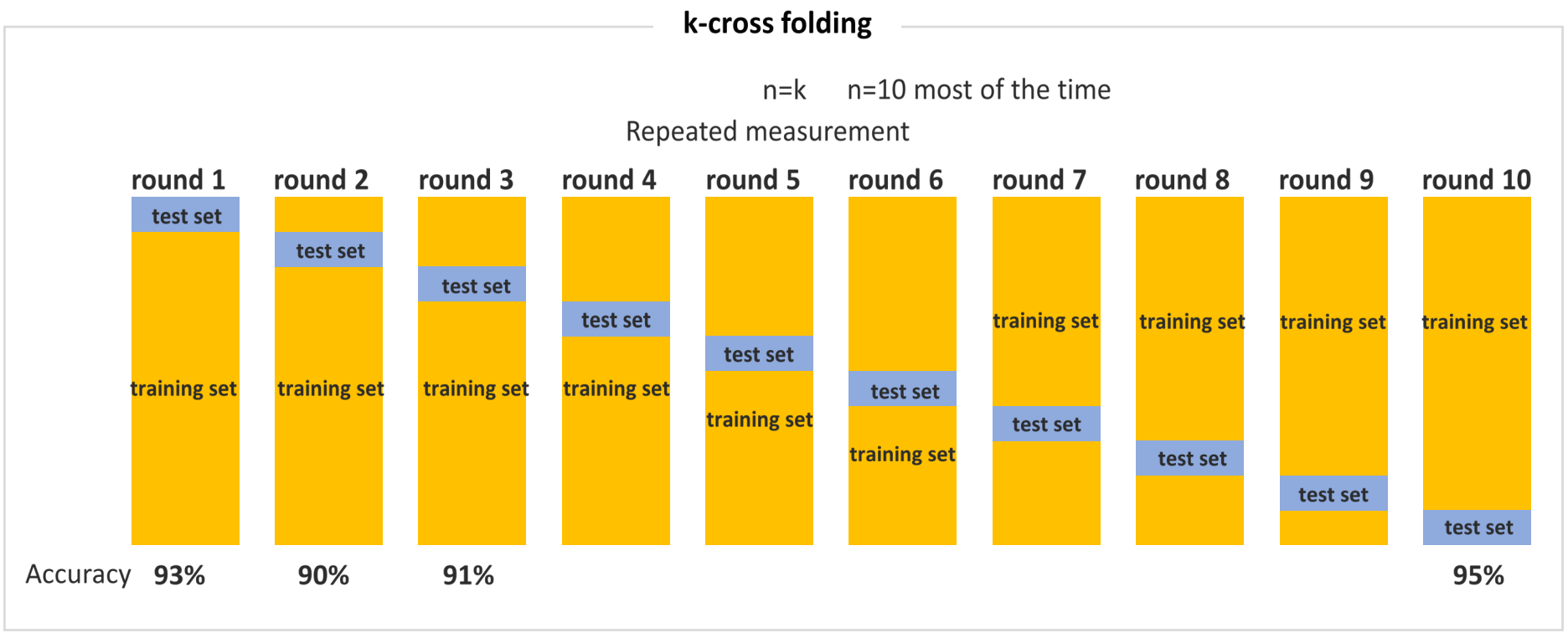
4) Calculate and save the fit statistic using test data (#3).

5) Repeat 1 through 4 multiple times, then average the results of step 4.



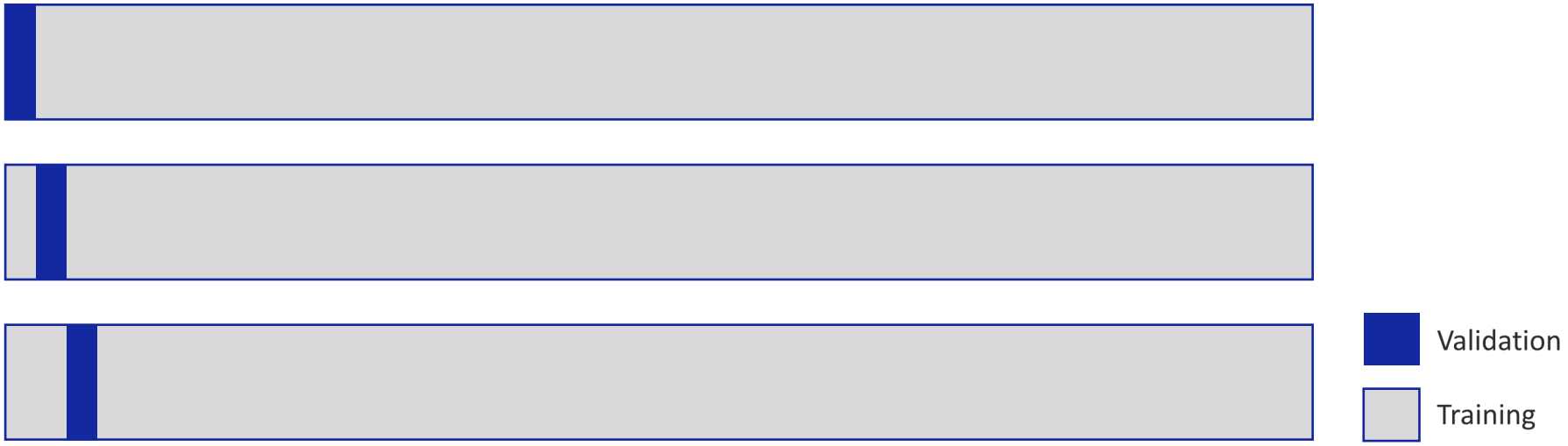
- We take all possible combinations with n–1 for training and the remaining fold for test.
- The error (or any other metric) is calculated n times and then averaged.
- **A final model is trained with all the data.**

Cross-Validation method: k-cross folding



Final average accuracy (Round1, Round2… Round10)

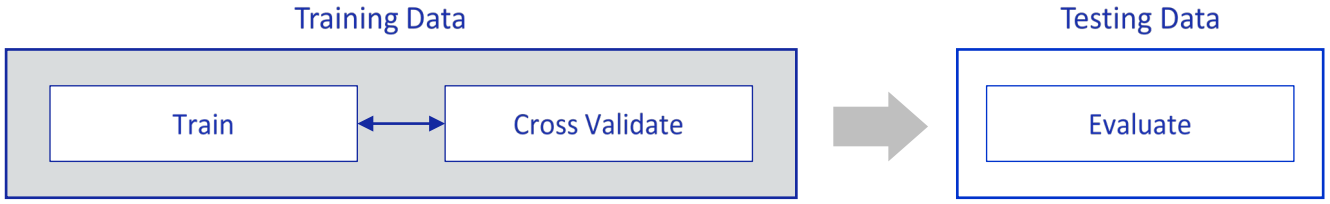
Cross-Validation method: **Leave One Out (LOO)**



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

Cross-Validation and Hyperparameter optimization

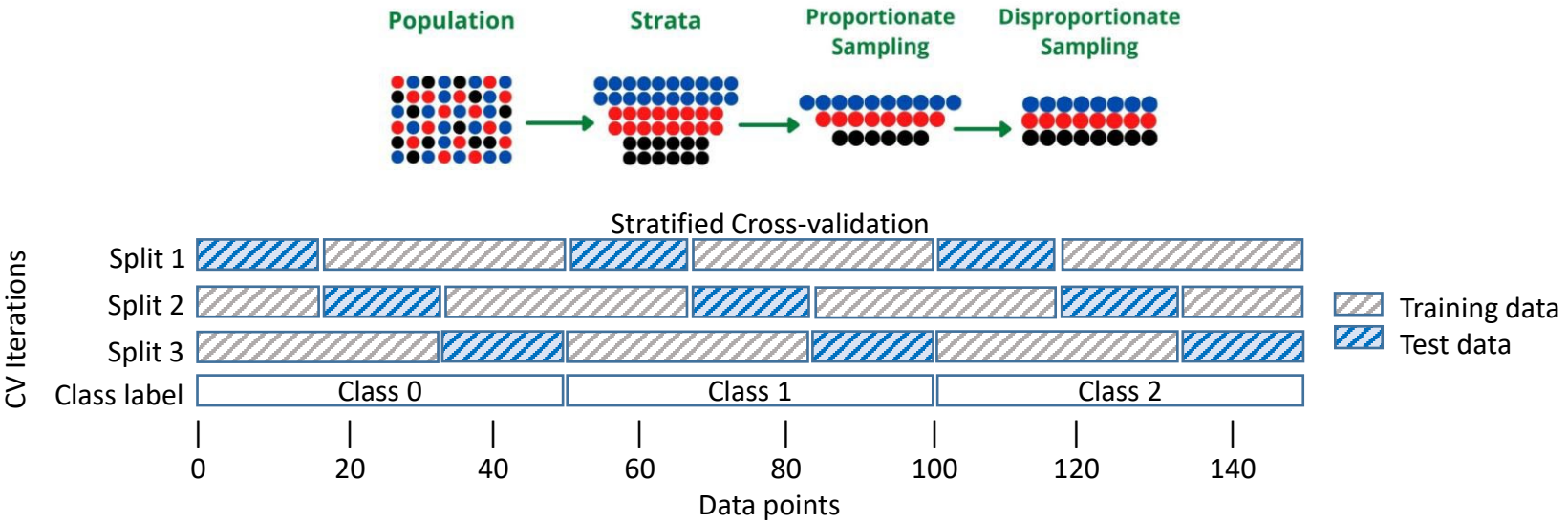
- The data should be split into a training set and a testing set.
- In principle, the testing set should be used only once! It should not be reused!
- We would like to evaluate realistic errors while training by splitting the training data into two.



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

stratified

- The random splitting of the train and test sets would result in inconstant target class when hold-out. If so, **the data distribution differs in the train set and validation set which affects learning**.
- For machine learning, a premise is present in which **the distribution of training data and real-life data distribution are the same**. If the premise is not followed, the performance of learning model falls.




```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=3)
train_index, test_index in skf.split(X, y)
```

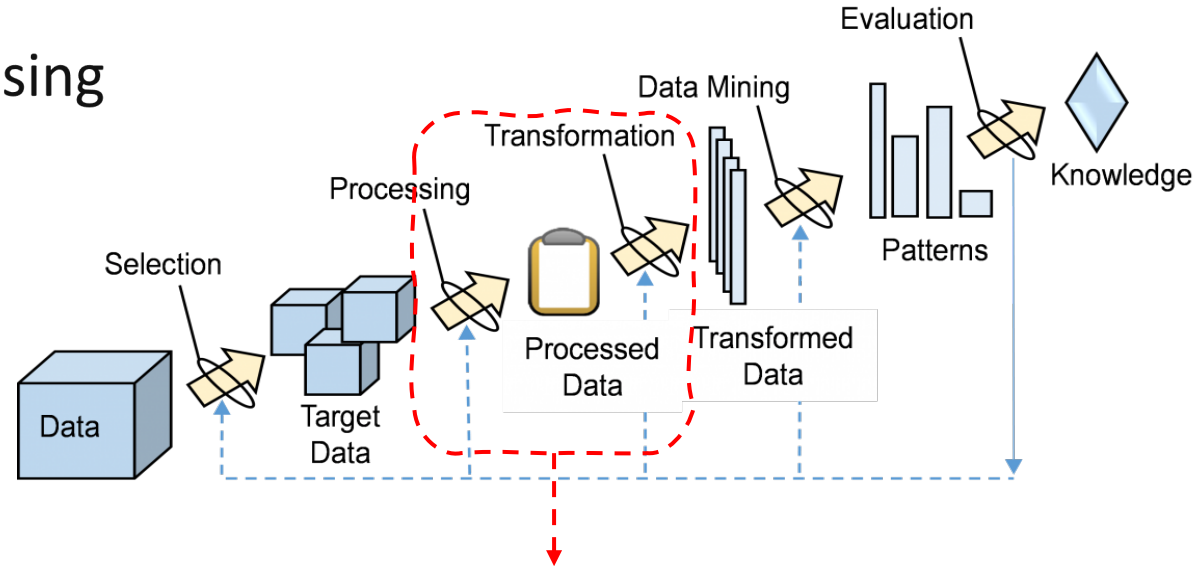
- This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

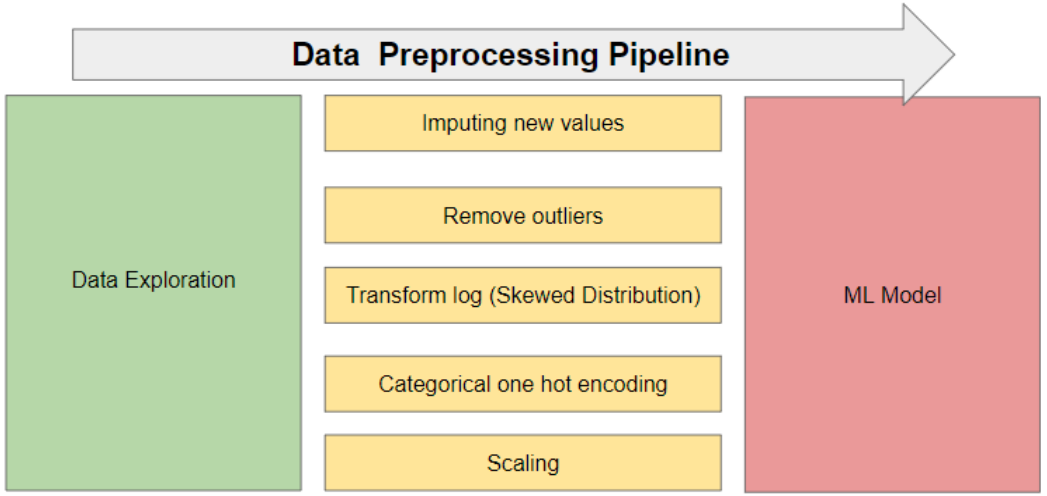
Unit 1.

Machine Learning Based Data Analysis

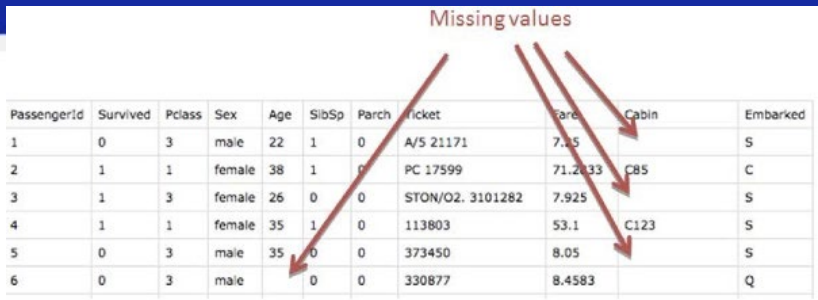
- 1.1. What is machine learning?
- 1.2. Phyton scikit-learn library for machine learning
- 1.3. Preparation and division of data set
- 1.4. Data pre-processing for making a good training data set
- 1.5. Practicing to find an optimal method to solve problems with scikit-learn

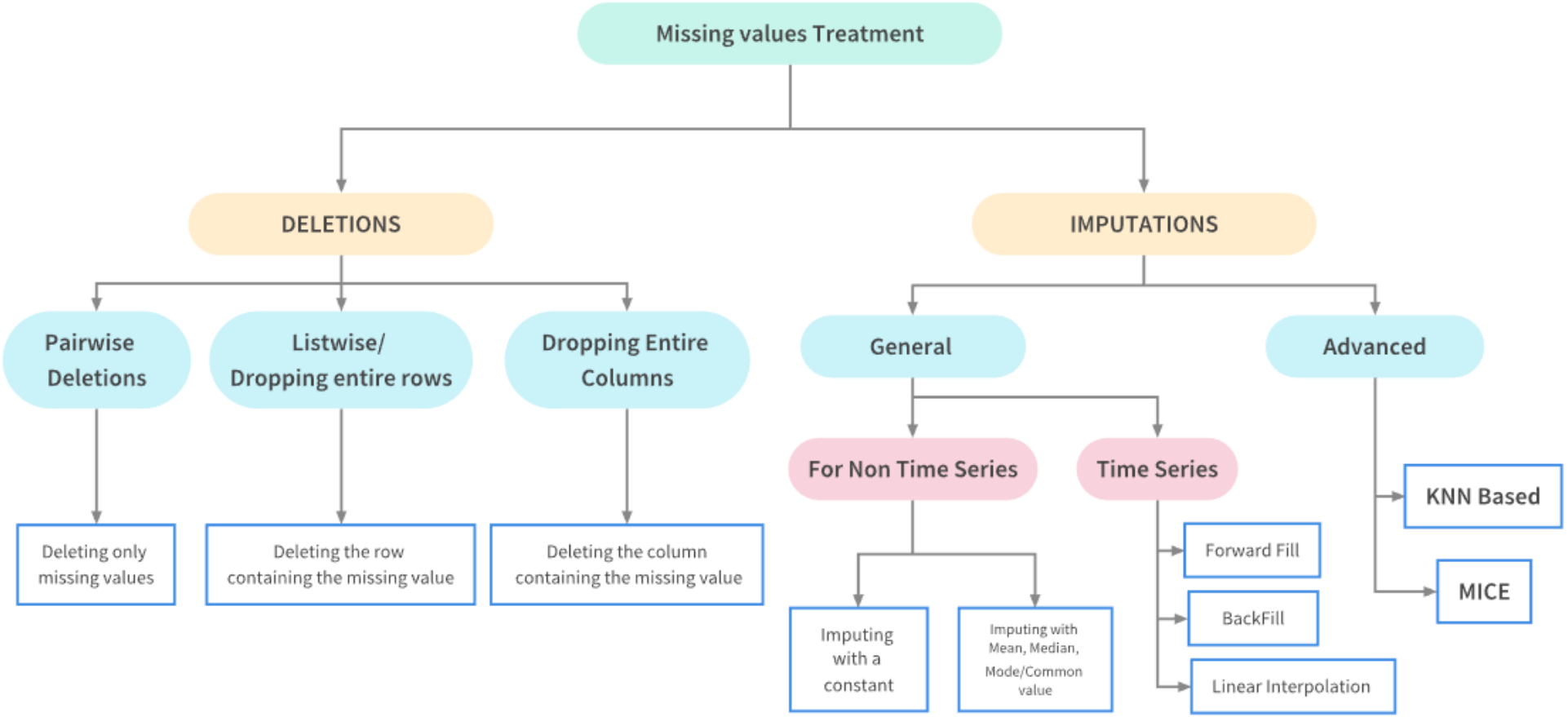
Data pre-processing





Missing value processing





Missing value processing

- Identify the missing value from the table type data below and make appropriate processing.
  - In python, the missing value is specified as np.nan or null value.
  - 'nan' is 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 data frame with boolean which show 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.

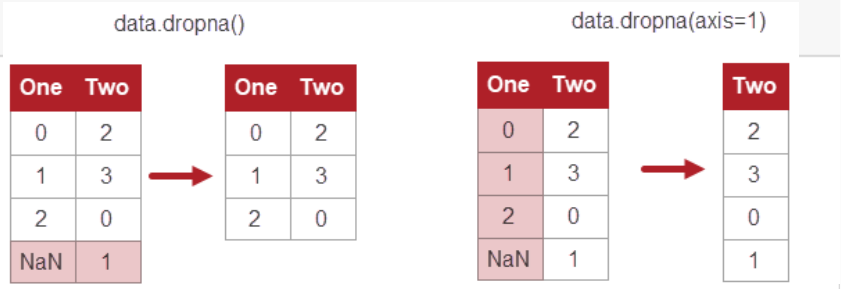
Removing the training sample or feature with missing value

- Completely delete a certain training sample (row) or feature (column). Use **dropna**(). Axis=0 is default

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

Out[18]:

	A	B	C	D
0	1.0	2.0	3.0	4.0



- 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
```

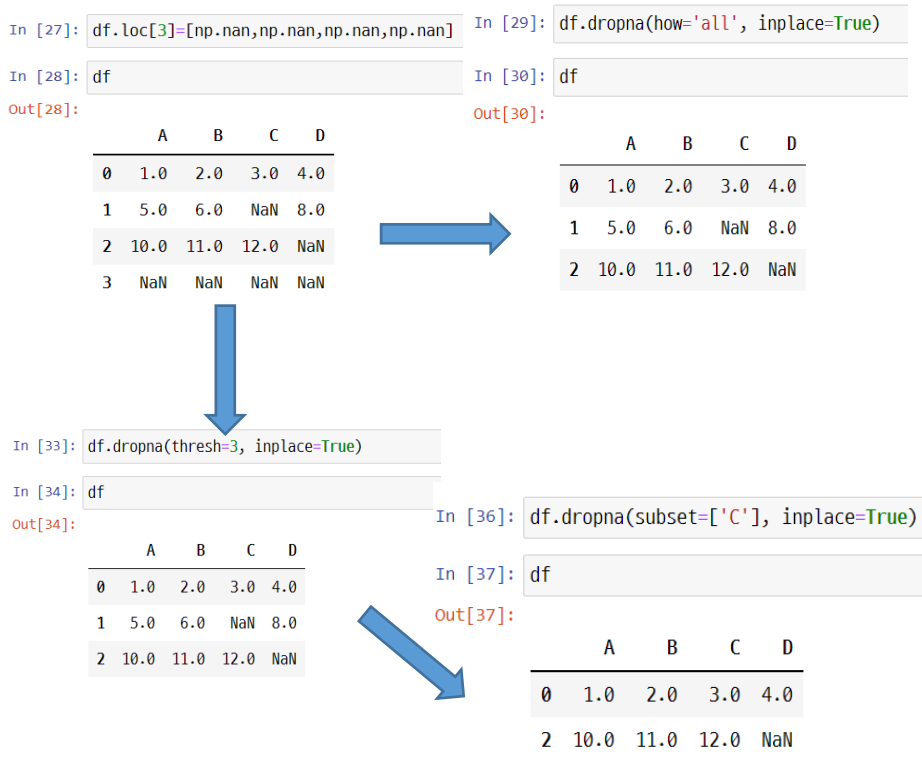
Out[20]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

Removing the training sample or feature with missing value

- Here is a brief overview of the parameters for **dropna**():

- **axis**: This parameter accepts 0 or 'index' (default) to drop rows with missing values, and 1 or 'columns' to drop columns with missing values.
- **how**: This parameter accepts two possible values - **'any'** (default) and **'all'**. If set to 'any', a row/column is dropped if it contains at least one missing value (NA). If set to 'all', a row/column is dropped only if all its values are missing.
- **thresh**: This parameter accepts an integer value. It requires a certain number of non-NA values in a row/column to avoid being dropped. If not provided, the default behavior is determined by the how parameter.
- **subset**: This parameter accepts an array-like object (list, tuple, etc.) containing labels of the columns to consider when dropping rows. Only rows with missing values in the specified columns will be dropped.



Imputation

- It is sometimes hard to delete a training sample or a certain column, and this is because it loses too much of useful data. If so, estimate missing values from other training samples in 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.

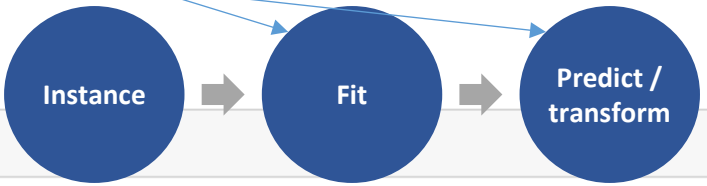
```
In [38]: from sklearn.impute import SimpleImputer

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 [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
```



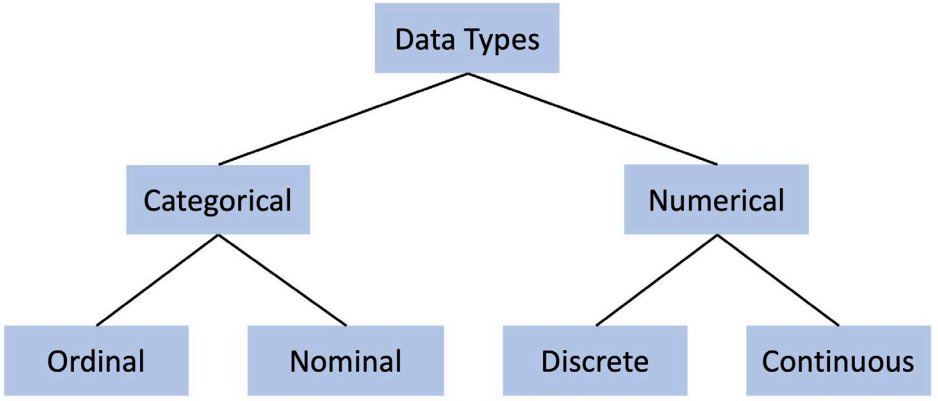
Line 45

- Check it is the average of the column.

- For strategy parameters, **median** and **most_frequent** can be also set.

Categorical data processing

- Data is generally classified into categorical scale and continuous scale 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 nominal scale and ordinal scale.
- **Continuous scale**
  - It is a scale that divides linked data into the purpose of survey and is classified into interval scale and ratio scale.
- 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 have features that have orders and do not have orders.
  Size is ordered, but color is not ordered. Thus, size is classified as ordinals scale while color is 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

- 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']
```

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

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 not possible to directly use categorial data to machine learning algorithm such as regression analysis, etc., and if so, conversion is required to be recognized by a computer.
- In such cases, use dummy variable which is 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 or not.**
- If a certain feature is present, it is expressed as 1 and if it's not found, it is classified as 0.

pd.get_dummies()

train_df:

	Call_Me?	Money	Target
0	Yes	5	10
1	No	3	4
2	Maybe	5	5
3	Yes	10	7
4	Yes	9	9

Data Preprocessing

train_features:

	Money	Call_Me?_Maybe	Call_Me?_No	Call_Me?_Yes
0	5	0	0	1
1	3	0	1	0
2	5	1	0	0
3	10	0	0	1
4	9	0	0	1

test_df:

	Call_Me?	Money
0	Yes	5
1	Yes	2
2	Yes	3
3	Yes	6
4	No	1

Data Preprocessing

test_features:

	Money	Call_Me?_No	Call_Me?_Yes
0	5	0	1
1	2	0	1
2	3	0	1
3	6	0	1
4	1	1	0

OneHotEncoder()

train_df:

	Call_Me?	Money	Target
0	Yes	5	10
1	No	3	4
2	Maybe	5	5
3	Yes	10	7
4	Yes	9	9

Data Preprocessing

train_features:

	Money	Call_Me?_Maybe	Call_Me?_No	Call_Me?_Yes
0	5	0.0	0.0	1.0
1	3	0.0	1.0	0.0
2	5	1.0	0.0	0.0
3	10	0.0	0.0	1.0
4	9	0.0	0.0	1.0

test_df:

	Call_Me?	Money
0	Yes	5
1	Yes	2
2	Yes	3
3	Yes	6
4	No	1

Data Preprocessing

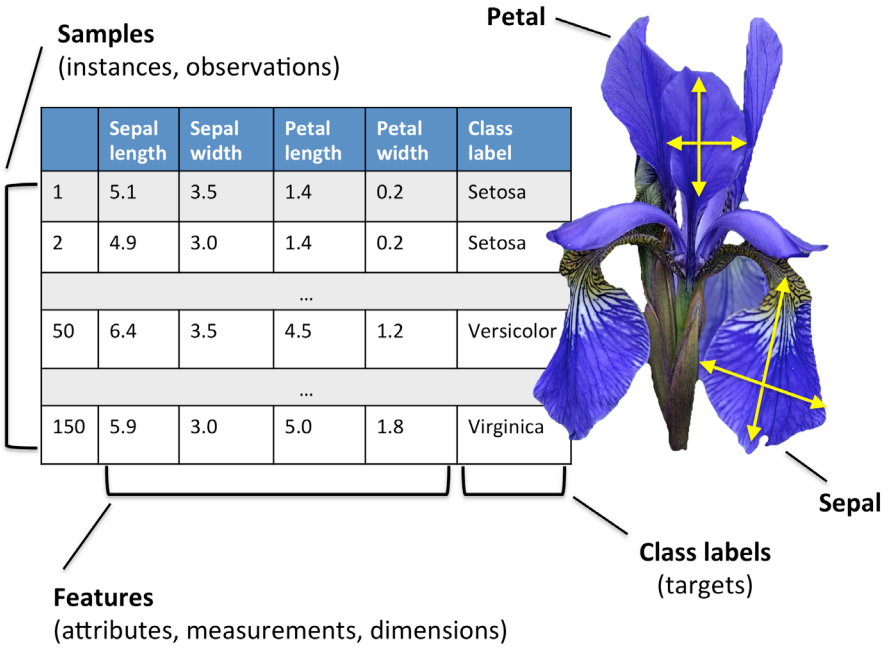
test_features:

	Money	Call_Me?_Maybe	Call_Me?_No	Call_Me?_Yes
0	5	0.0	0.0	1.0
1	2	0.0	0.0	1.0
2	3	0.0	0.0	1.0
3	6	0.0	0.0	1.0
4	1	0.0	1.0	0.0

- For quick data cleaning and EDA, it makes a lot of sense to use pandas **get_dummies()**. However, if I plan to transform a categorical column to multiple binary columns for machine learning, it's better to use **OneHotEncoder().**

Application of one-hot encoding to unordered feature

- Practice with the iris.target object. (iris = load_iris())



	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

```
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],

In [69]: iris.target

Out[69]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

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
...	...	...	...

OneHotEncoder

```
In [23]: from sklearn.preprocessing import OneHotEncoder
         from sklearn.datasets import load_iris
         iris = load_iris()
         iris_target = pd.DataFrame(iris.target, columns = ["species"])
         iris_target["species"].replace({0:"setosa", 1:"versicolor", 2:"virginica"}, inplace = True)
```

```
         from sklearn.preprocessing import OneHotEncoder
         onehot_encoder = OneHotEncoder()
         ohe_reshaped = onehot_encoder.fit_transform(iris_target)
```

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

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

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

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

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

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

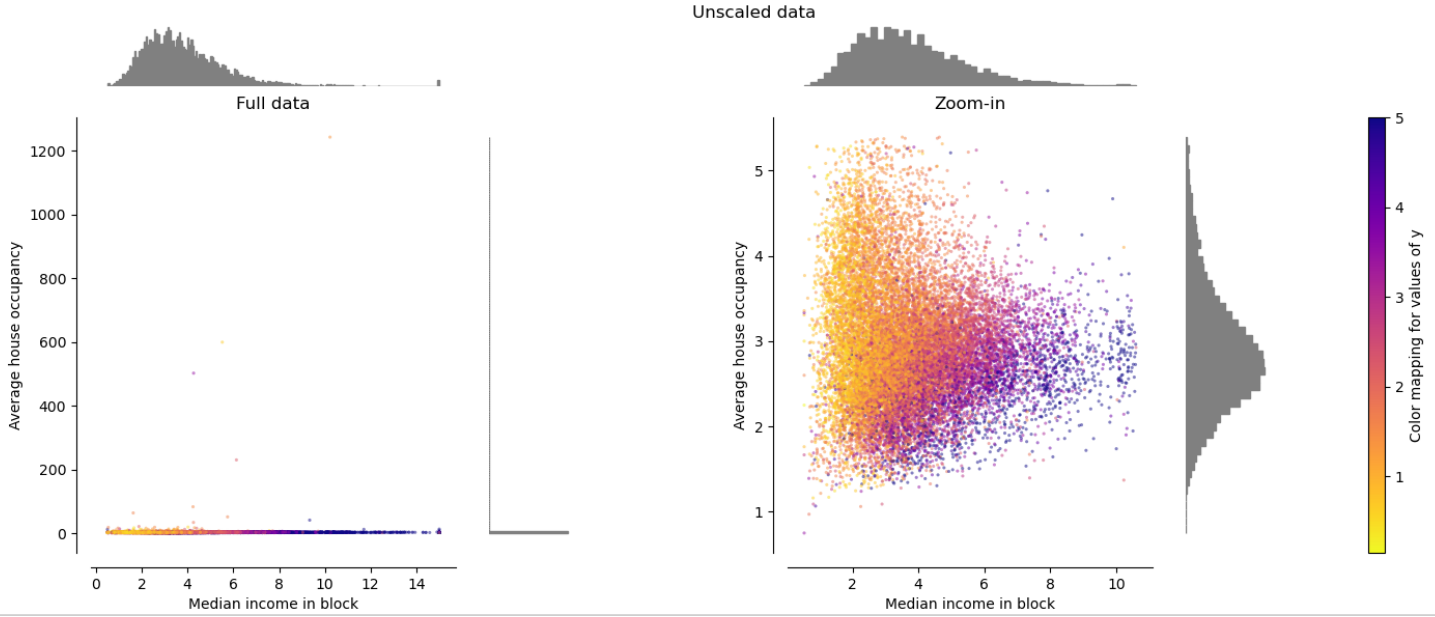
Refer to the figure below for easier understanding.

	Column index	0	1	2	
Row 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)

Arranging the scale between features (variables)

Many estimators are designed with the assumption that **each feature** takes **values close to zero** or more importantly that **all features vary on comparable scales**.

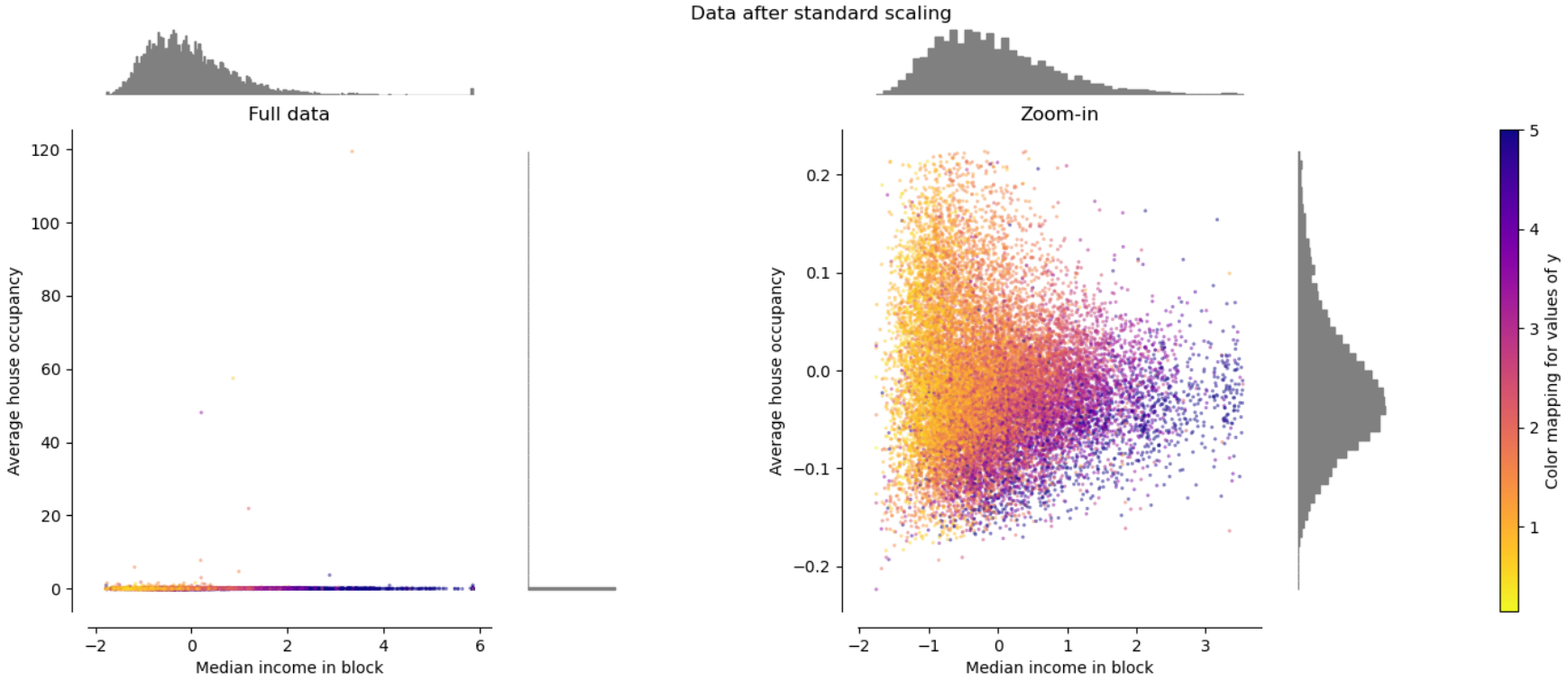
- In particular, metric-based and gradient-based estimators often **assume approximately standardized data** (centered features with unit variances). A notable exception are decision tree-based estimators that are robust to arbitrary scaling of the data.
- **Scalers** are linear transformers and differ from each other in the way they estimate the parameters used to shift and scale each feature.
- ScikitLearn provides different **scalers**, **transformers**, and **normalizers** to bring the data within a pre-defined range.



Arranging the scale between features (variables)

- **StandardScaler** removes the mean and scales the data to unit variance (z = (x - u) / s). The outliers have an influence when computing the empirical mean and standard deviation: the spread of the transformed data on each feature is very different. StandardScaler therefore cannot guarantee balanced feature scales in the presence of outliers.

```
In [17]: from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

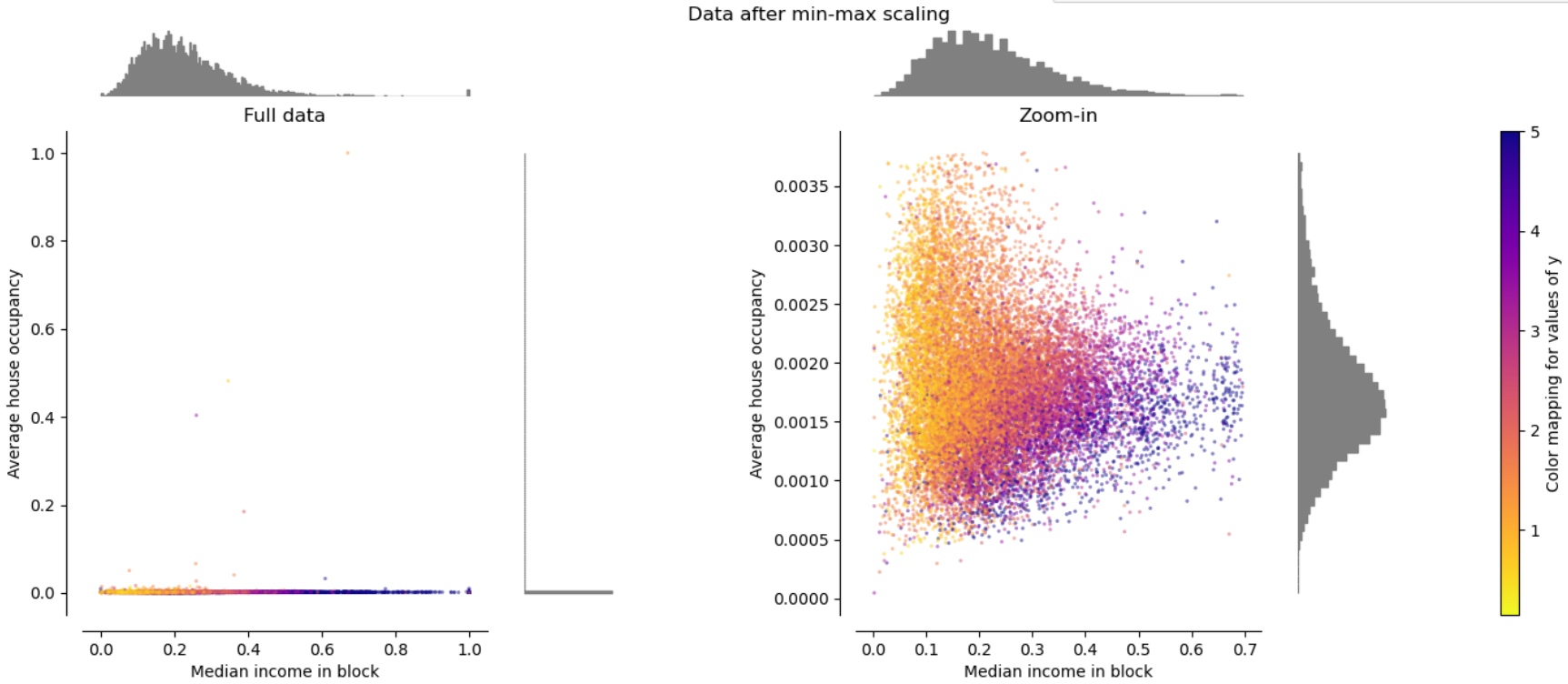


Arranging the scale between features (variables)

- **MinMaxScaler** rescales the data set such that all feature values are in the range [0, 1]. Both StandardScaler and MinMaxScaler are very sensitive to the presence of outliers.

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

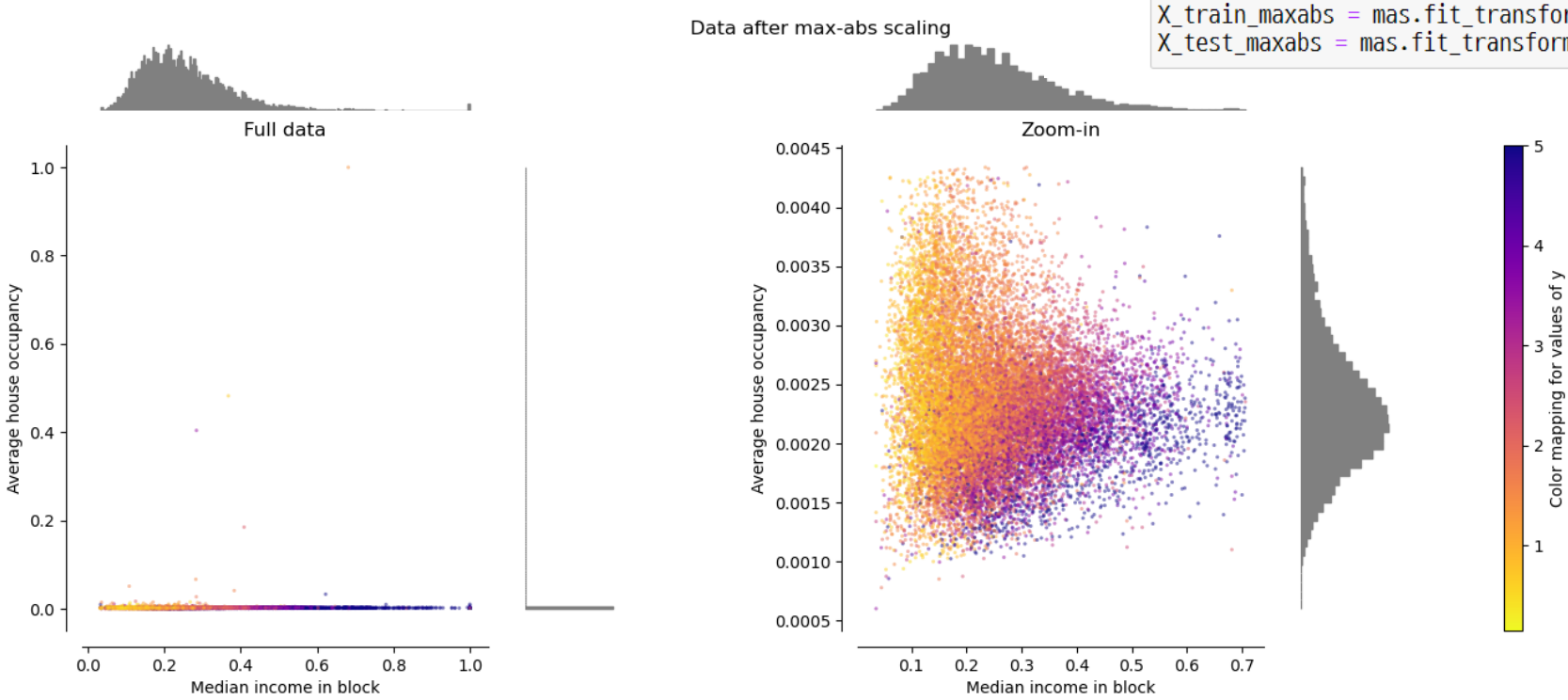
```
In [16]: from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```



Arranging the scale between features (variables)

- **MaxAbsScaler** divides the data into the maximum absolute value based on each feature. It is similar to MinMaxScaler except that the values are mapped across several ranges depending on whether negative OR positive values are present. If only positive values are present, the range is [0, 1]. If only negative values are present, the range is [-1, 0].

```
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)
```

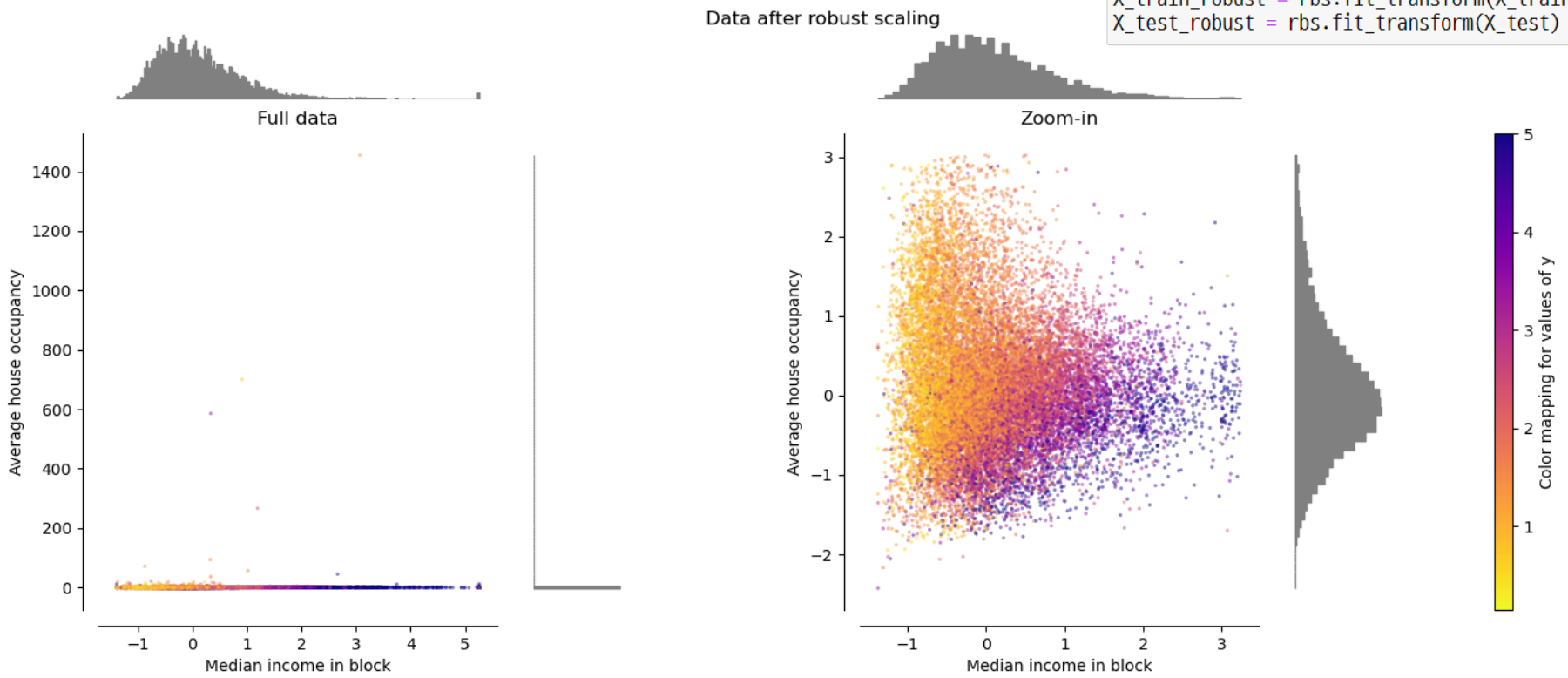


Arranging the scale between features (variables)

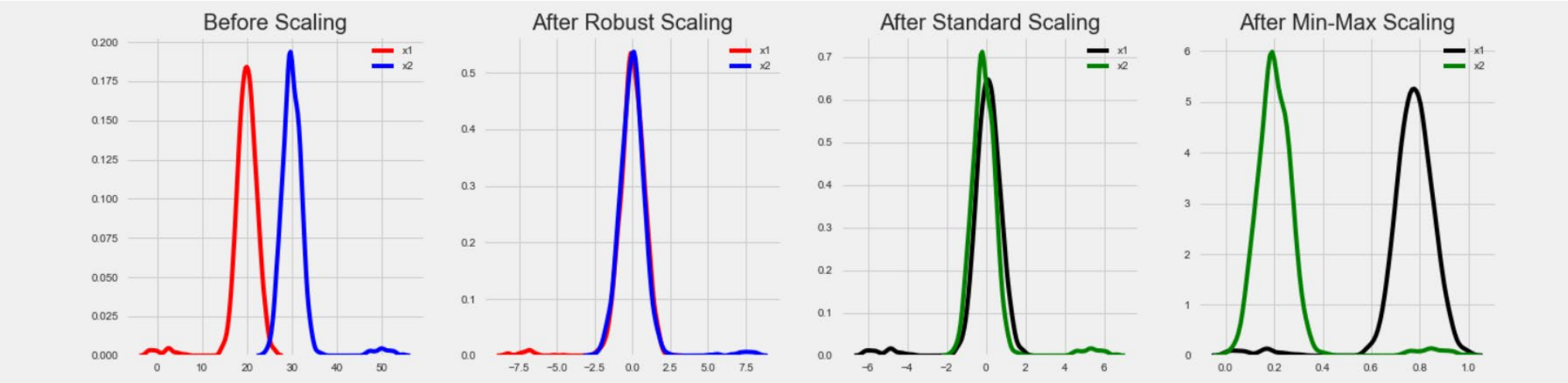
$$X_{new} = \frac{X - X_{median}}{IQR}$$

- Unlike the previous scalers, the centering and scaling statistics of **RobustScaler** are based on percentiles and are therefore not influenced by a small number of very large marginal outliers.

```
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)



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

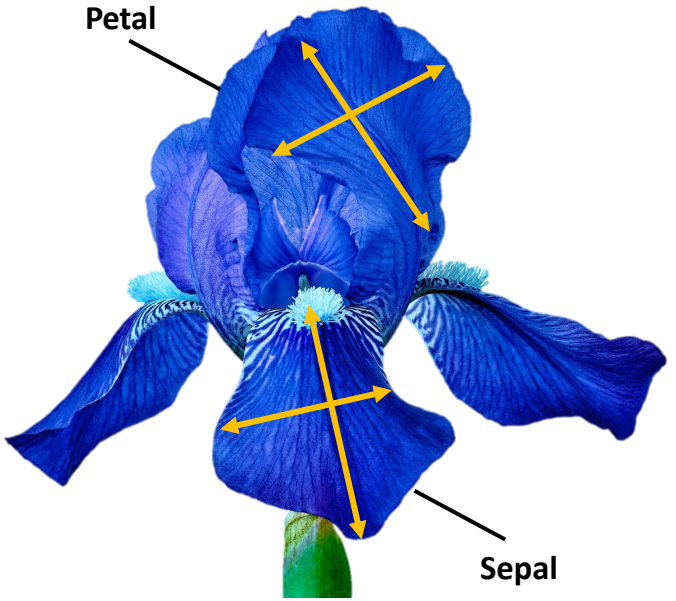
Classroom practicing (for students)

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 as **supervised or unsupervised problems**.

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 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

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.

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
```

Out[6]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

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

Iris data pre-processing and EDA

In [8]:
```
target
```

Out[8]:

	species
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 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 the 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

Excercise

- Change the target value (“Species”) to {0 -> setosa, 1 -> versicolor, 2 -> virginica} using the **map** function.

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

Iris data pre-processing and EDA

Excercise

- Check the number of missing values per attribute

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

Iris data pre-processing and EDA

- Basic statistical analysis
  - Perform basic statistical analysis for a better understanding of data. From analysis, it is possible to have better understand the data by understanding the data size (numbers of data), shape of data (matrix shape), data type, data distribution, and the relationship between features and 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
```

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 by 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 analysis as the ‘size’ counts the number of data including missing values while 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

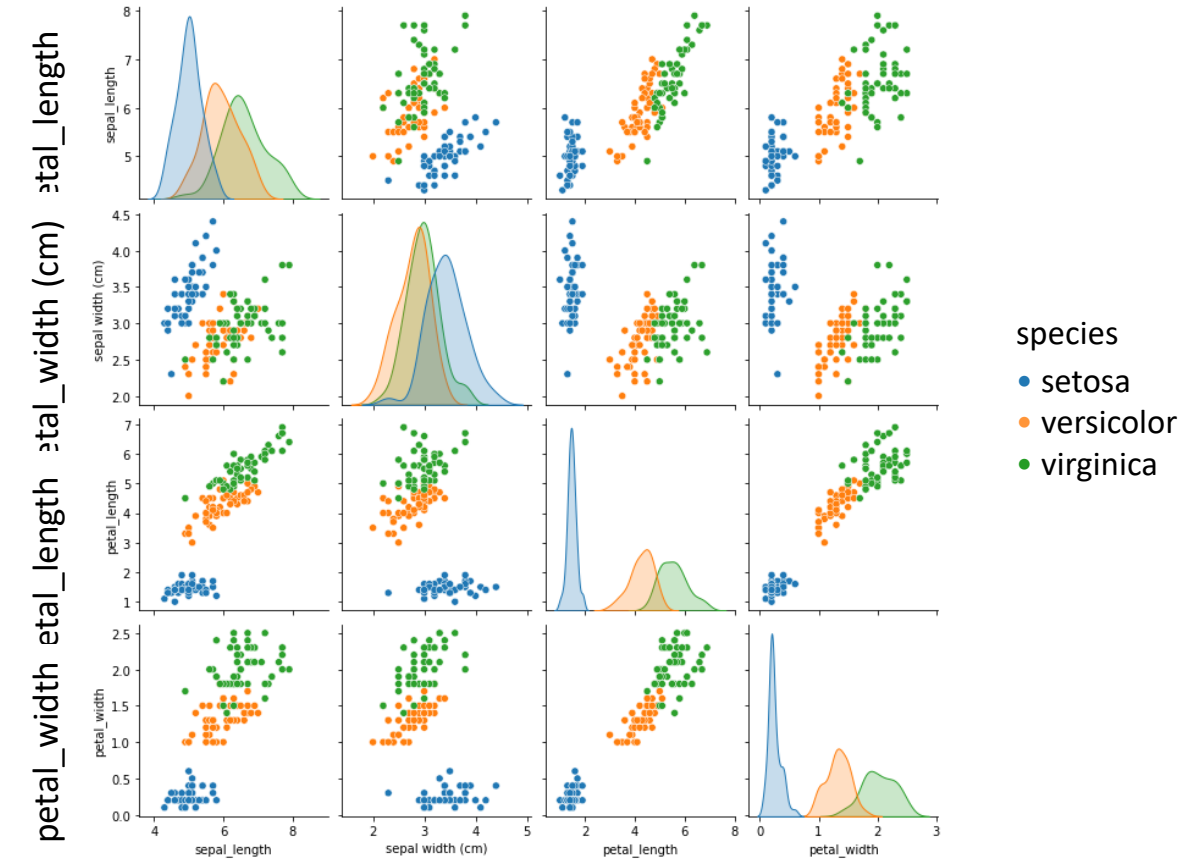
- Data visualization
  - Previously, basic statistics analysis was done on the data, but it is not easy to understand due to too many numbers, and incorrect reading of decimal points would result in significant error. If so, visualization of data as a graph provides an intuitive understanding of the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

Visualizing the correlation between features and data distribution by using pairplot

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

Visualizing the correlation between features and data distribution by using pairplot

Out [21] : []



Excercise

- What can you observe? Check the distribution for the different classes, their separability for the different attributes, etc.

Visualizing the class ratio of the target

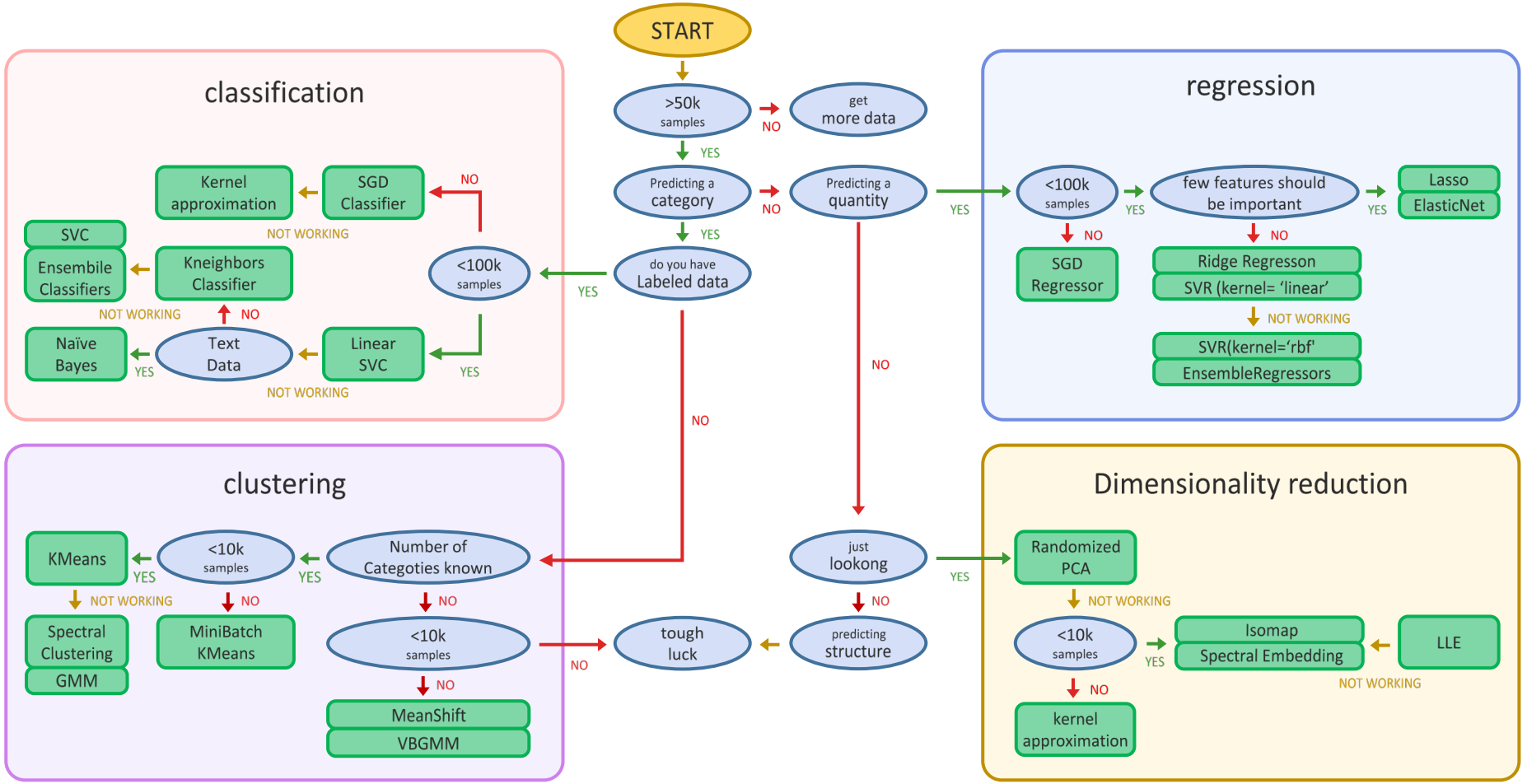
- Before starting with the machine learning procedures, split the data set into the training data and performance test data. The final objective of machine learning is to create a generalized model so that it can accurately predict new data. If evaluating the performance with data that was used in learning, the possibility of getting 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 hold out method.

Excercise

- Split the training data set and performance test data set 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. Use the **test_size=0.33** option separates 33% of the total data as test set. Set **random_state=42** to allow reproducible results for the practicing problem. If not designating random_state, the data set for conversion will differ every time.

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

Algorithm selection



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 decision tree through the hyperparameters.
  - Lowering **max_depth** would constraint 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**: Largest number of the feature that will be used for splitting by each node.
- Increasing the parameter that starts with min_ or lowering the parameter that starts with max_ would increase the model constraint.

Model learning

- Perform model learning with the train 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 by using the performance test data set. In the Scikit-learn, score refers to accuracy. Since the iris data set is well-structured data set for practice, it generally shows a high performance in any model.

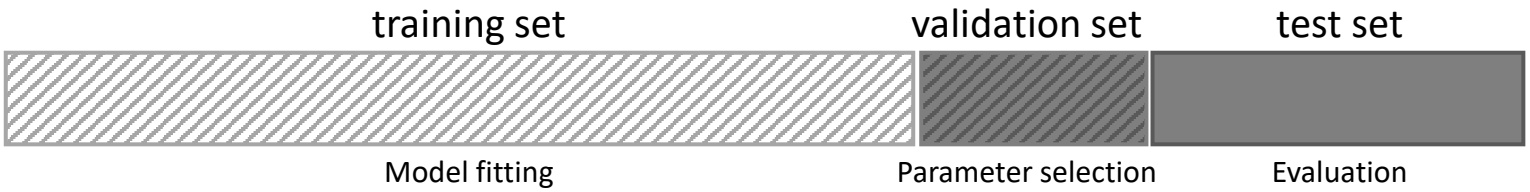
```
In [29]: model.score(X_test,y_test)
Out[29]: 0.98
```

Model generalization strategy (**RECAP**)

- As machine learning shows data-driven model performance, enough amount of data is required for good performance. 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 split with Train_test_split is for the final performance evaluation of the model. Because 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, and it is also used to find hyperparameters.

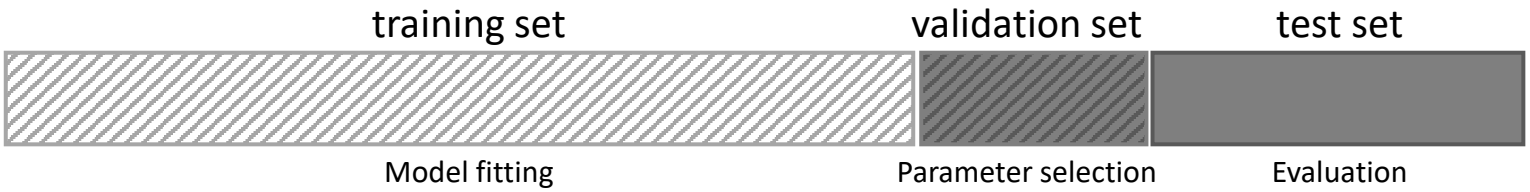


Model generalization strategy (**RECAP**)

- As machine learning shows data-driven model performance, enough amount of data is required for good performance. 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 split with Train_test_split is for the final performance evaluation of the model. Because 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, and it is also used to find hyperparameters.



Cross_val_score

- Cross validation can be easily performed by 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(str(i) + "th cross validation score: " + repr(_))
print("Final cross validation score: " + repr(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
```

Learning Curve

- What is the effect of the **number of training examples**, on training and CV errors? We draw a **learning curve**.
- The learning curve shows **how performance changes when slightly increasing the amount of training data** by setting x-axis as number of training data and y-axis as performance score. The test score is calculated by internal cross validation.

- What is the effect of the number of training examples on **training error**?
  - If the training set is **small** → Easier to fit every single training example perfectly → Your training error = 0 or small
  - If the training set grows **larger** → Harder to fit every single training example perfectly → Your training error increases
- What is the effect of the number of training examples on **cross validation error**?
  - The more data you have → Your cross validation error decreases

- The learning curve can be drawn by using the **scikitplot library** which supports the scikit-learn. Install the library separately from scikit-learn to use (!pip install scikit-plot).

In [34]:
```
!pip install scikit-plot
```

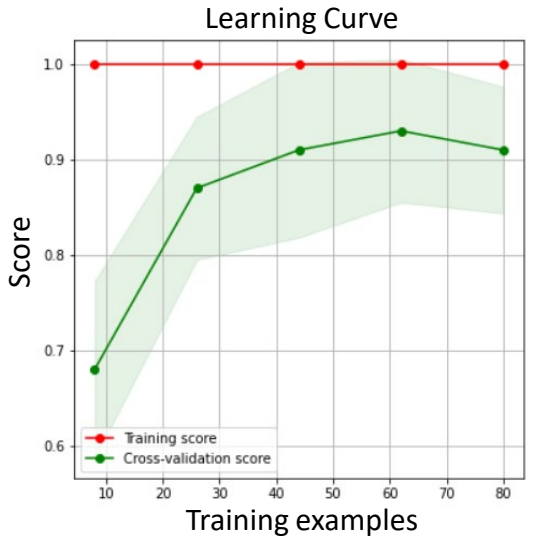
```
Requirement already satisfied: scikit-plot in c:\users\emcast\anaconda3\lib\site-packages (0.3.7)
Requirement already satisfied: scipy>=0.9 in c:\users\emcast\anaconda3\lib\site-packages (from scikit-plot) (1.6.2)
Requirement already satisfied: matplotlib>=1.4.0 in c:\users\emcast\anaconda3\lib\site-packages (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)
```

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()
```



Excercise

- What do you observe?

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 by using GridSearchCV**
  - In general, hyperparameters are found by the analyst's expertise.
  - The scikit-learn provides the **GridSearchCV** function that finds hyperparameters, and it is a function that lists all number of cases regarding hyperparameter combination on a grid and learns and performs performance measure to every combination.
  - It may seem like working without any plans, but the work is automatically performed by the machine as the range of hyperparameters is designated by the analyst. While it takes some time, it eases finding hyperparameters.

- **How it works?** Similar to algorithms, instance the **GridSearchCV** as well. When instancing, send the instanced algorithm model as the argument to the estimator option. For param_grid, send the dictionary containing hyperparameters for testing as the argument.

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)
```

Use only “parameters = {"max_depth" : [4,6,8,10,12]}” to avoid long waiting times

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

- The optimal parameters and optimized performance found with GridSearCV are recorded in best_params_and best_score_ attributes.
- If the refit option is set 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
randomight_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

```
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 final evaluation of the model. For 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', '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.

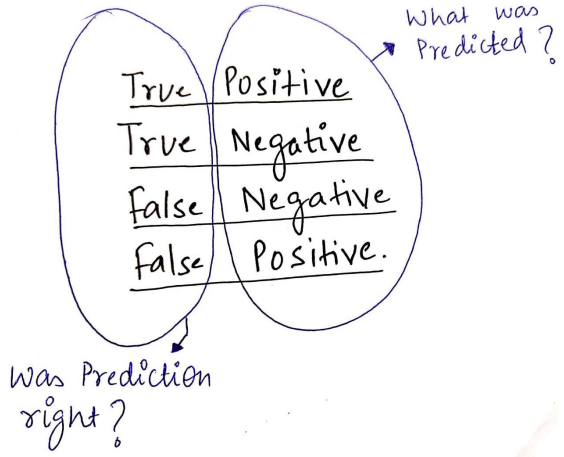
- However, let's assume that there are 48 setosas, 1 versicolor, and 1 virginica in the test set. When making an evaluation by using this test set, a problems is that it would have 96% accuracy. Nevertheless, it's not because the model performance is great. It would be necessary to check other evaluation criteria as well to accurately evaluate the model performance.

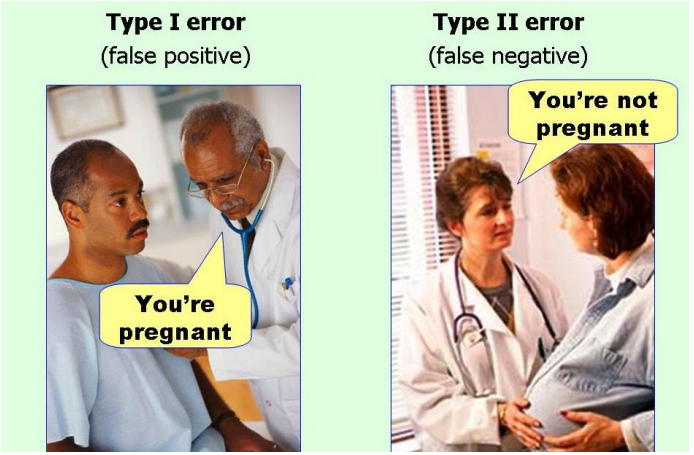
Confusion Matrix

- 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 four concepts provided above (TP, FP, TN, FN).
- Use the confusion matrix to analyze both right and wrong predicted results. The confusion matrix can validate the performance in different ways to see how well the predicted and actual targets got right.





Confusion Matrix

Sources: [21][22][23][24][25][26][27][28][29] view · talk · edit

		Predicted condition			
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) = TPR + TNR − 1	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)**	**True positive (TP),** **hit**	**False negative (FN),** **type II error, miss,** **underestimation**	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	Negative (N)	**False positive (FP),** **type I error, false alarm,** **overestimation**	**True negative (TN),** **correct rejection**	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P+N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) = $\frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) = $\frac{TN}{PN} = 1 - FOR$	Markedness (MK), deltaP (Δp) = PPV + NPV − 1	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$
	Balanced accuracy (BA) = $\frac{TPR + TNR}{2}$	F_1 score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes–Mallows index (FM) = $\sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index = $\frac{TP}{TP + FN + FP}$

https://en.wikipedia.org/wiki/Confusion_matrix

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**

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

Confusion matrix of the iris data set

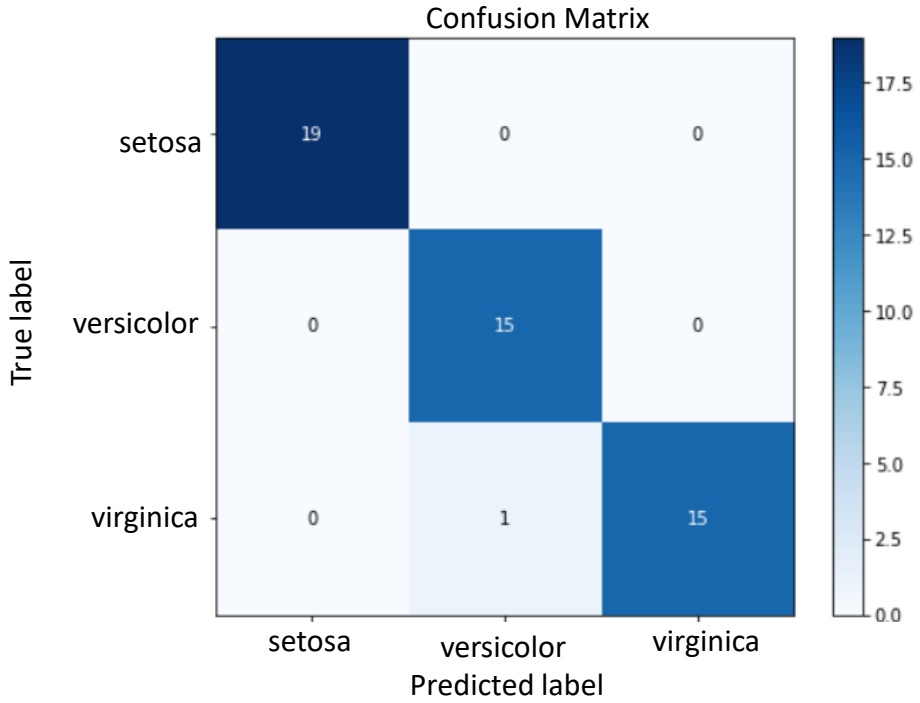
```
In [43]: from sklearn.metrics import confusion_matrix
         pred=model2.predict(X_test)
         confusion_matrix(y_test,pred)

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 by using confusion_matrix function. Send the arguments to actual class and then predicted class.

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 x-axis and y-axis, but scikit-plot has labels for the axis for easier result interpretation.

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



precision / recall / f-score / accuracy

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

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

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

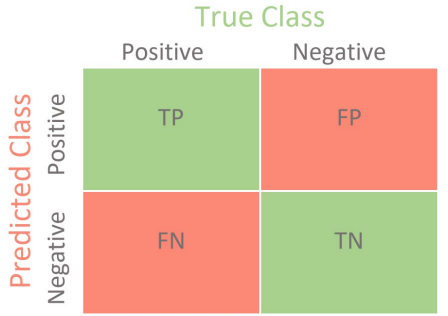
Ex Iris data

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

precision

- Precision — Also called Positive predictive value. The ratio of **correct positive predictions** to the **total predicted positives**.

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




```
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

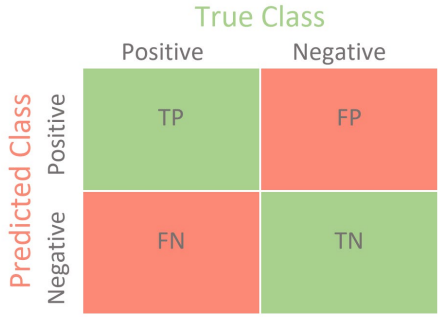
Line 45

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

recall

- Also called Sensitivity, Probability of Detection, True Positive Rate. The ratio **of correct positive predictions** to the **total positives examples**.

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



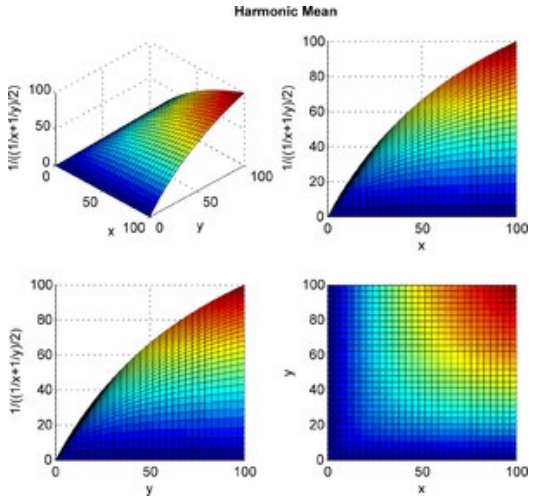

```
In [46]: from sklearn.metrics import recall_score
recalls=recall_score(y_test,model2.predict(X_test),average=None)
for target, score in zip(data.target_names, recalls):
    print(f"{target}sensitivity:{score}")
```

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

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, and 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, $\boldsymbol{\beta}$ is set 1 most of the time, which is specifically referred to as **f1-score**.

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

Applications

- Often used in the field of information retrieval for measuring *search, document classification, and query classification* performance. It is particularly relevant in applications which are **primarily concerned with the positive class and where the positive class is rare relative to the negative class**.
- The F-score is also used in machine learning. However, the F-measures **do not take true negatives into account**, hence measures such as the Matthews correlation coefficient, Informedness or Cohen's kappa may be preferred to assess the performance of a binary classifier.

f-score

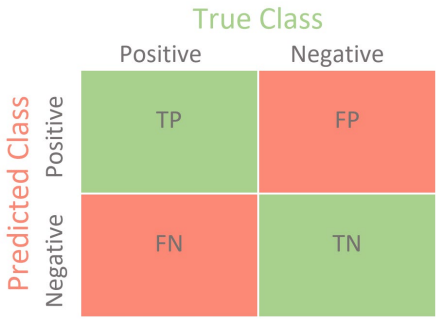
In [47]:
```
from sklearn.metrics import fbeta_score, f1_score

fbetas=fbeta_score(y_test,pred,beta=1,average=None)

for target, score in zip(data.target_names, fbetas):
    print(f"{target}fbetas 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 fbetas score : 1.0
versicolor fbetas score : 0.967741935483871
virginica fbetas score : 0.967741935483871
setosa f1 score : 1.0
versicolor f1 score : 0.967741935483871
virginica f1 score : 0.967741935483871
```

accuracy

- Accuracy is defined as the ratio of correctly predicted examples by the total examples.

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



- Remember, accuracy is a very useful metric when all the classes are equally important.
- But this might not be the case if we are predicting if a patient has cancer. In this example, we can probably tolerate FPs but not FNs.

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
```

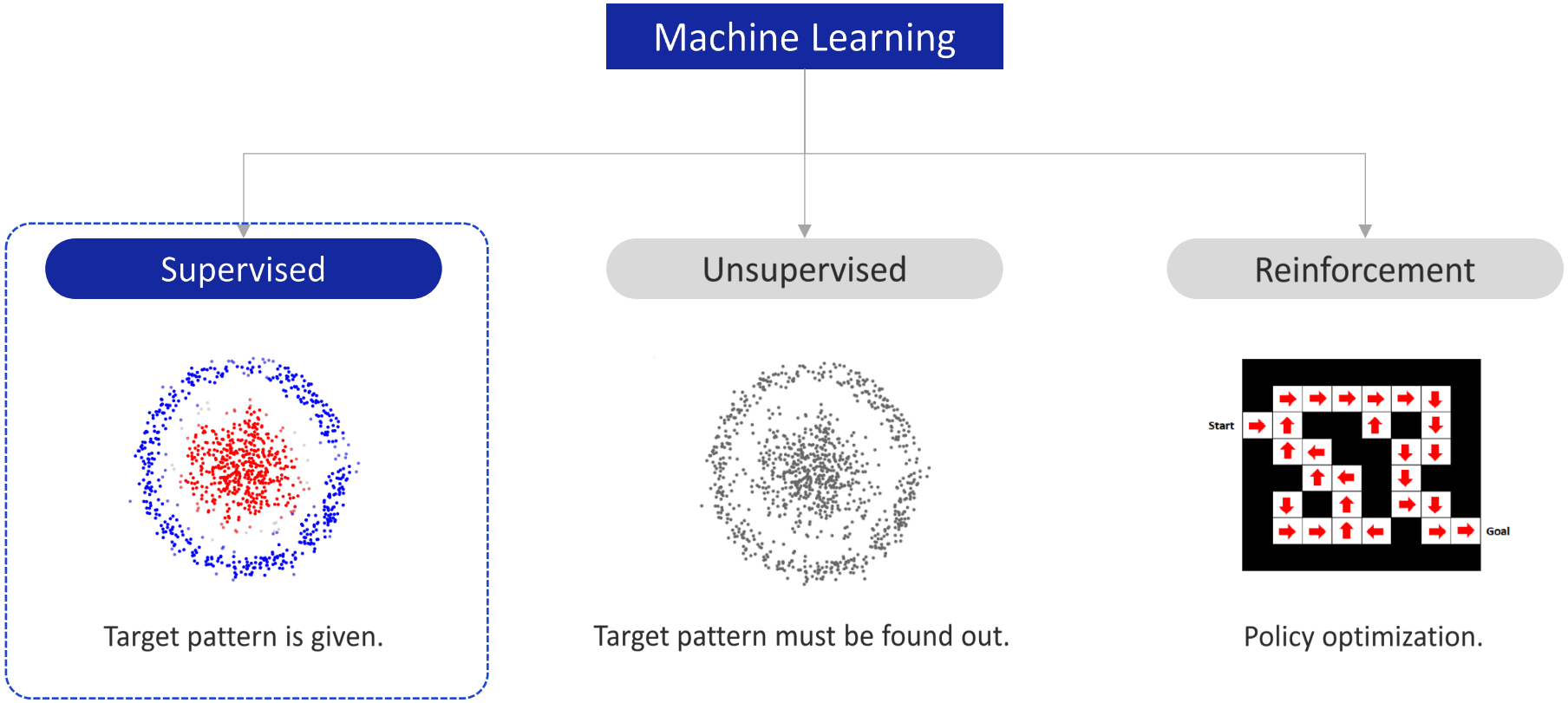
- **Support** is the number of actual target classes

Unit 2.

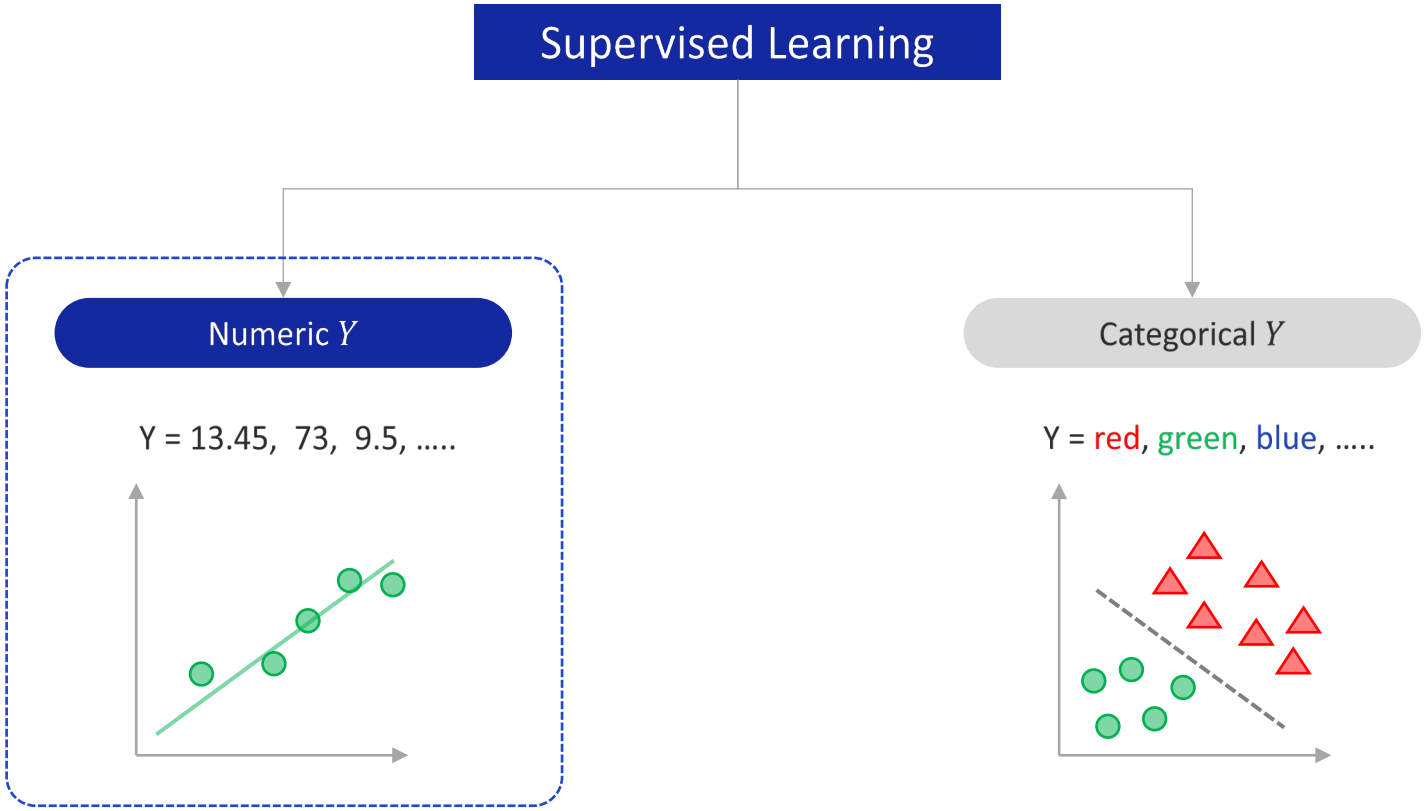
Application of the Supervised Learning Model for Numerical Prediction

- 2.1. Linear Regression Basics
- 2.2. Linear Regression Diagnostics
- 2.3. Other Regression Types
- 2.4. Practicing the Supervised Learning Model for Numerical Prediction

Machine Learning Types



Linear Regression Basics



About linear regression

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

Purpose of linear regression

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

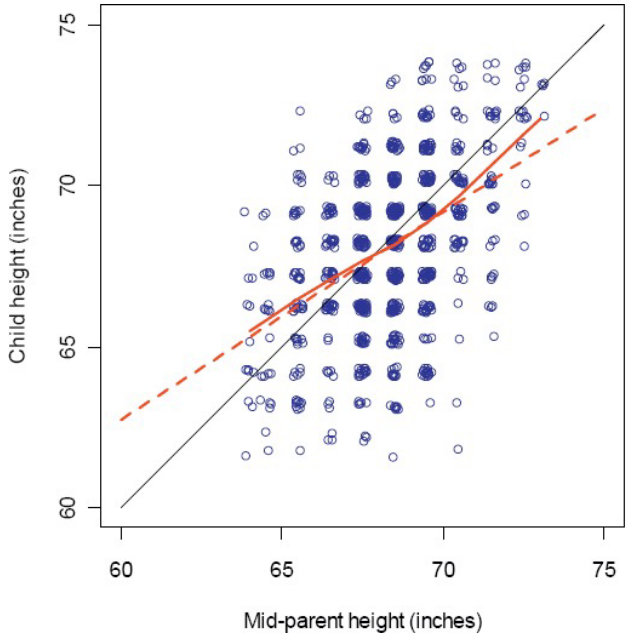
Ex If real **estate price** is the response variable Y, which are the most statistically meaningful explanatory variables? Area ($\mathbf{X_1}$), location ($\mathbf{X_2}$), age ($\mathbf{X_3}$), distance to business center ($\mathbf{X_4}$), etc.

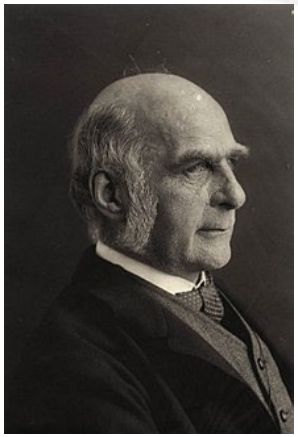
b) Predict the response given the conditions for the explanatory variables.

Ex Price of a 10-year-old apartment of area 100 $m2$ with location 3 km from the business center?
← "predict" the value that is not open to the public yet.

Historical background

- Term “regression” 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 (i.e., several independent variables in a model are correlated).

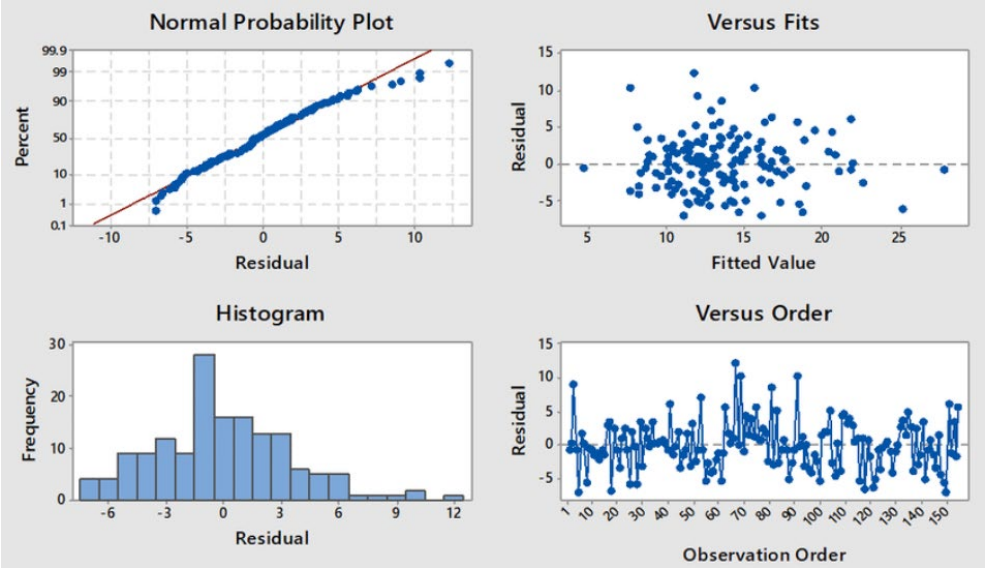
$$Y = a + bX$$

X is the explanatory variable and **Y** is the dependent variable.
The **slope** of the line is **b**, and **a** is the **intercept** (the value of y when x = 0).

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

- Variable values are given by data:

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

Response variable — Explanatory variables

Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

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

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

Regression coefficients

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

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

$$E[\varepsilon] = 0$$
$$Var(\varepsilon) = \sigma_\varepsilon^2$$

Interpreting the regression coefficients

$$\Delta Y = \beta_0 + \beta_1 \Delta X_1 + \cdots + \beta_i \Delta X_i + \cdots + \beta_k \Delta X_k$$

- If X_1, X_1, …, X_k change by ΔX_1, ΔX_2, …, ΔX_k then the change in Y is ΔY.

$$\Delta Y = \beta_0 + \beta_1 \Delta X_1 + \cdots + \beta_i \Delta X_i + \cdots + \beta_k \Delta X_k$$

ΔX_i = 1

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

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

β_0: Intercept

$\beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon$ = 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 and 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.

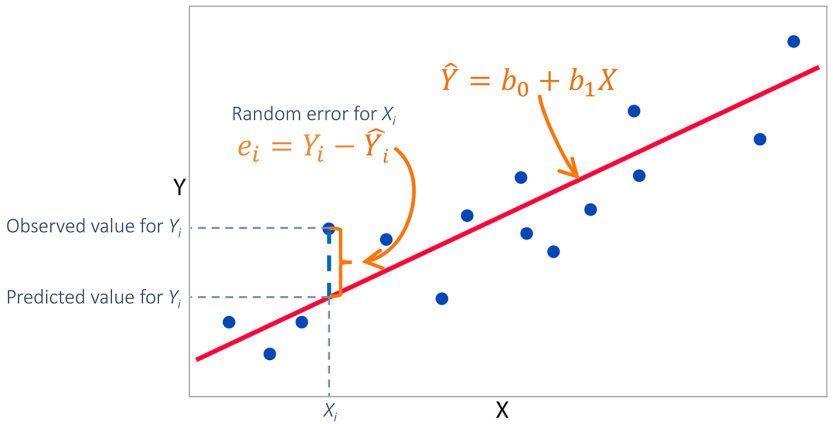
Ordinary Least Squares (OLS) solution

- We have an over determined system of linear equations; the exact solution does not exist.

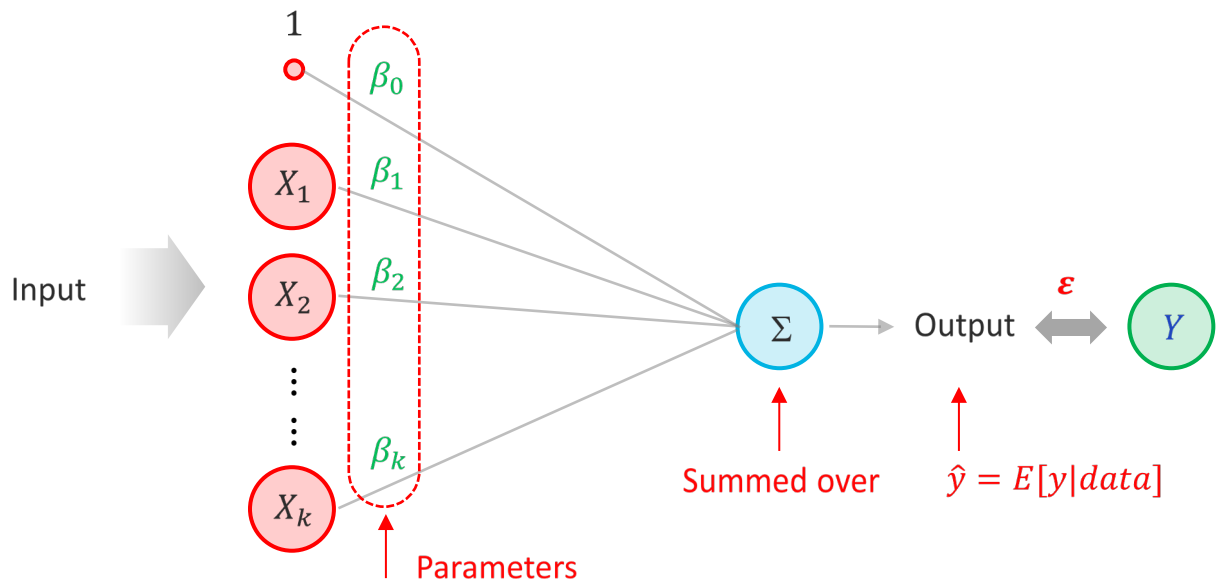
$$y_j = \beta_0 + \beta_1 x_{j,1} + \beta_2 x_{j,2} + \cdots + \beta_K x_{j,k} + \varepsilon_j$$

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

- **Ordinary least squares (OLS)** regression is a statistical method that produces the one straight line that minimizes the total squared error.
- OLS chooses the parameters of a linear function of a set of explanatory variables by the principle of least squares:
  - Minimizing the sum of the squares of the differences between the observed dependent variable (error of prediction) in the given dataset and those predicted by the linear function of the independent variable.



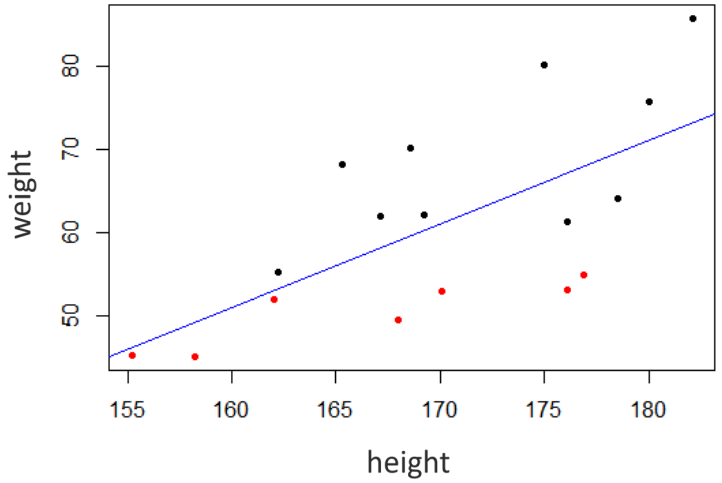
Regression training and prediction (testing)



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

Role of categorical variables

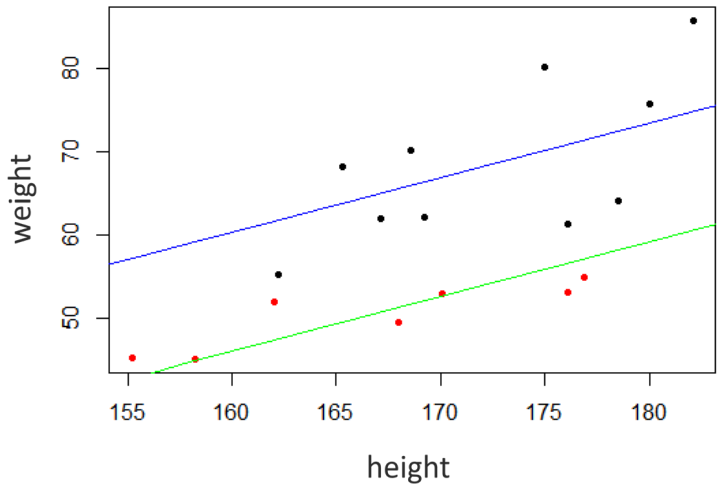
- Without categorical variable → weight ~ height



- $= male$
- $= female$

Role of categorical variables

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



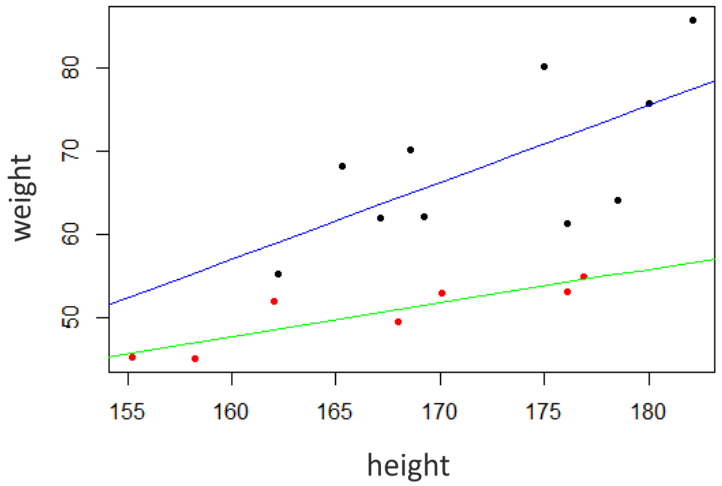
- $= male$
- $= female$

- The categorical variable must be turned into **dummy variable**(s) first.
- Here, we include a categorical variable as an independent variable.
- This makes the **intercept dependent on the category or type**.
- Effectively raises or lowers the intercept.

Role of categorical variables

- With a categorical variable **gender** that "interacts" → weight ~ height + **gender** + height × **gender**

Interacting term



- $= male$
- $= female$

- 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

Linear Regression Diagnostics

- Linear regression diagnostic methods

  1) Error metrics: MSE, RMSE, MAE, etc.

  2) Coefficient of determination or “r-squared” $R2$.

  3) F-test for overall significance of the linear model.

  4) t-test for significance of individual regression coefficients.

  5) Correlation between Y and $\hat{Y}$.

- Modelling: optimization of the information criteria: AIC or BIC.

- Residual and leverage analysis.

Error metrics

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

$$\mathbf{RMSE} = \sqrt{MSE}$$

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

- The smaller, the better!

Predicted Value($\hat{y}$)	Actual Value (y)	Error	Error^2
100 mill. €	102 mill. €	2	4
102 mill. €	110 mill. €	8	64
105 mill. €	95 mill. €	10	100
95 mill. €	75 mill. €	20	400
101 mill. €	103 mill. €	2	4
105 mill. €	110 mill. €	5	25
105 mill. €	98 mill. €	7	49
40 mill. €	32 mill. €	8	64
220 mill. €	215 mill. €	5	25
100 mill. €	103 mill. €	3	9

MAE= 60/10 =6

MSE= 744/10 = 74,4

RMSE= sqrt(744/10) = 8.63

R^2, F-statistic, and T-statistics

- **R^2 (R-squared):** measures the proportion of the variability in the dependent variable that can be predicted from the independent variable(s): 1 indicates a perfect fit
- **F-Statistic**: Tests the overall significance of the model.
  - Null hypothesis $\boldsymbol{H0}$: $\beta1=\beta2=\cdots=\beta K=0$ Alternate hypothesis $\boldsymbol{H1}$: At least a βi is non zero
- **T-Statistic**: The T-statistic is used to test the hypothesis that a certain variable's regression coefficient in the population equal zero and gives an idea of how significant the variable is.
  - Null hypothesis $\boldsymbol{H0}$: $\beta i=0$ Alternate hypothesis $\boldsymbol{H1}$: $\beta i \neq 0$

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

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

OLS Regression Results

Dep. Variable:	y	R-squared:	0.518
Model:	OLS	Adj. R-squared:	0.507
Method:	Least Squares	F-statistic:	46.27
Date:	Wed, 08 Mar 2017	Prob (F-statistic):	3.83e-62
Time:	10:08:24	Log-Likelihood:	-2386.0
No. Observations:	442	AIC:	4794.
Df Residuals:	431	BIC:	4839.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	152.1335	2.576	59.061	0.000	147.071	157.196
x1	-10.0122	59.749	-0.168	0.867	-127.448	107.424
x2	-239.8191	61.222	-3.917	0.000	-360.151	-119.488
x3	519.8398	66.534	7.813	0.000	389.069	650.610
x4	324.3904	65.422	4.958	0.000	195.805	452.976
x5	-792.1842	416.684	-1.901	0.058	-1611.169	26.801
x6	476.7458	339.035	1.406	0.160	-189.621	1143.113
x7	101.0446	212.533	0.475	0.635	-316.685	518.774
x8	177.0642	161.476	1.097	0.273	-140.313	494.442
x9	751.2793	171.902	4.370	0.000	413.409	1089.150
x10	67.6254	65.984	1.025	0.306	-62.065	197.316

Here the p-value is the probability that $\boldsymbol{H0}$ for the full model is true (i.e., that all of the regression coefficients are zero). Since the p-value is approximately zero, we reject the null hypothesis.

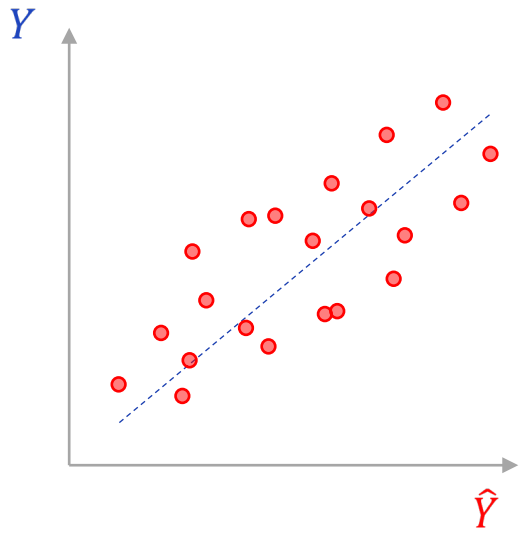
If the p-value is below a reference (say 0.05), then $\boldsymbol{H0}$ is rejected in favor of $\boldsymbol{H1}$. In this case, inclusion of the explanatory variable Xi is justified.

Further info:

https://medium.com/analytics-vidhya/f-statistic-understanding-model-significance-using-python-c1371980b796

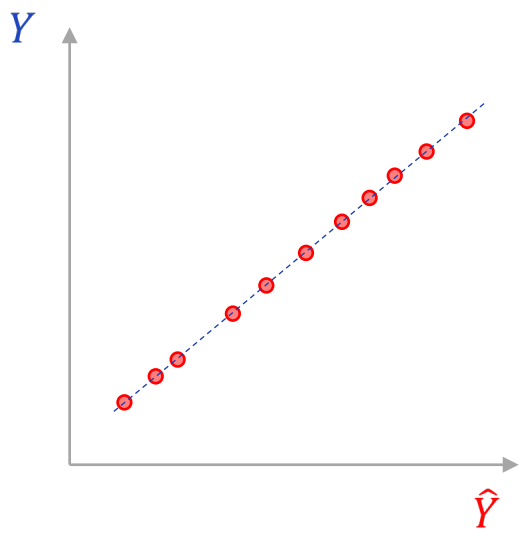
https://stackoverflow.com/questions/27928275/find-p-value-significance-in-scikit-learn-linearregression

Correlation between Y and $\hat{Y}$



Weak positive correlation

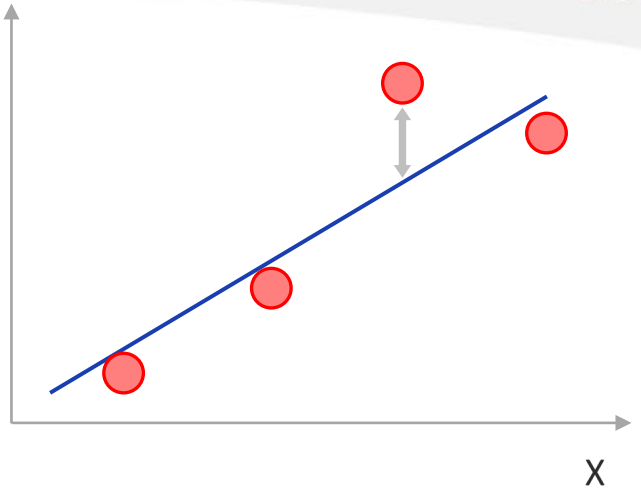
<



Strong positive correlation

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.



- Reasons for residual analysis:

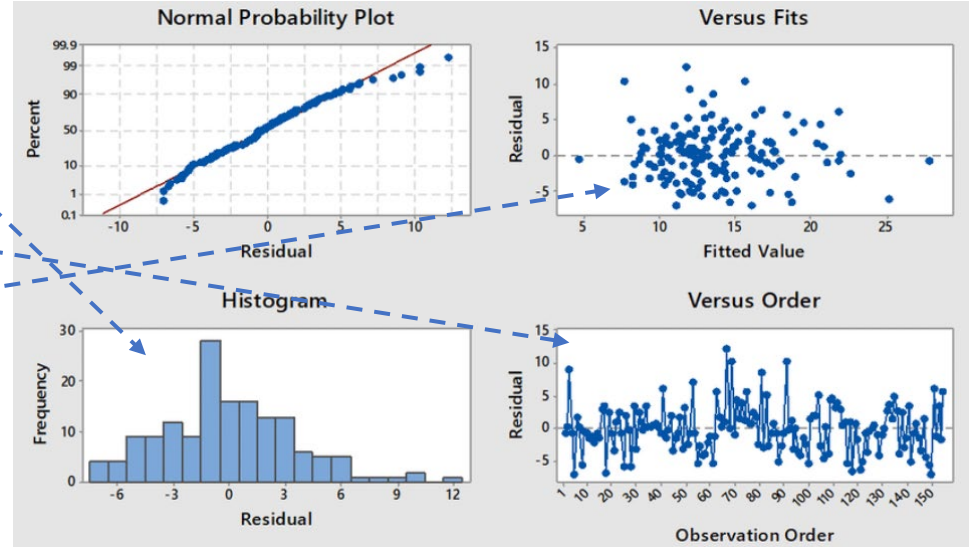
  1) To detect outliers in Y.

  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**.

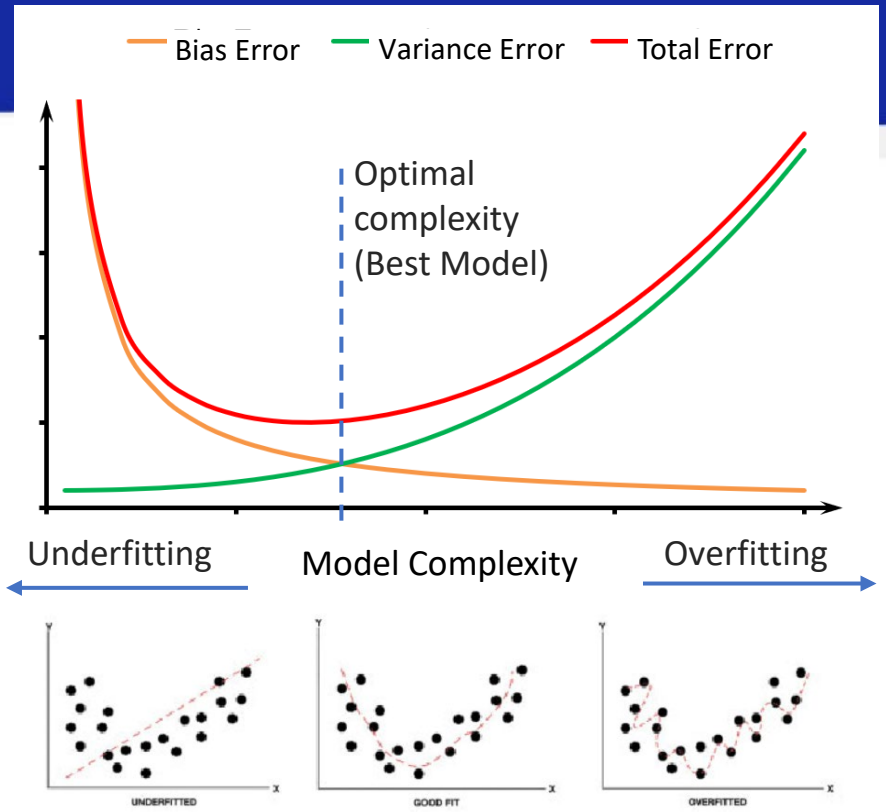


Unit 2.

Application of the Supervised Learning Model for Numerical Prediction

Regularized Regression

- Bias-Variance trade off



- **Bias error:** Difference between the average prediction of our model and the correct value. **Model with high bias pays very little attention to the training data and oversimplifies the model**. High error on training and test data.
- **Variance error:** Variability of model prediction. **Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before**. Low error on training data & high error rates on test data.
- The goal should be to minimize the **Total error** = **Bias error** + **Variance error**.
- When there is an excessive amount of **variance error**: **Ridge** and **Lasso** to bring the model towards the optimal point.
- When there is an excessive amount of **bias error**: **polynomial regression** to bring the model towards the optimal point.

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

Ridge regression

- Useful when the usual linear regression overfits (bias error << variance error).
- We remember that the OLS solution consists in minimizing $|\boldsymbol{\varepsilon}|^2$.

Squared differences between the observed dependent variable and the predictions

- Minimize the following "loss function":

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

This penalty is known as **l2** and has the effect of reducing the value of all the coefficients of the model proportionally, but not to zero (when λ is large).

- Positive and lager λ further constraints the coefficients β_i decreasing the variance (overfitting) error.
- However, too large λ can make the model too "biased".

When there is an excessive amount of **variance error**: **Ridge** and **Lasso** to bring the model towards the optimal point.

Lasso regression

- Also useful when the usual linear regression overfits (bias error << variance error).
- Minimize the following "loss function":

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

This penalty is known as **l1** and, when λ is large, the coefficients β_i can become exactly equal to zero (when λ is large)

- The main practical difference between lasso and ridge is that lasso makes some coefficients exactly zero and thus performs predictor selection, while ridge does not exclude any. This is a notable advantage of lasso in scenarios where not all predictors are important for the model and it is desired that the less influential ones are excluded.

Polynomial Regression

Polynomial regression

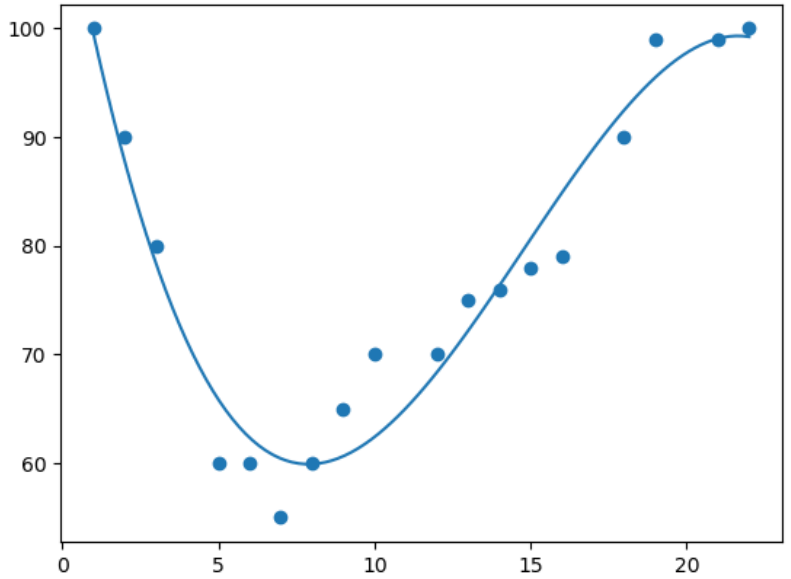
- Useful when the usual linear regression underfits (**bias error >> variance error**).
- We can model the relationship between X and Y using the polynomials:

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

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

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

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



Unit 2.

Application of 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_0302Ext.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. Classification Performance Metrics

Machine learning for classification

- **Classification** is used if the objective variables (or response variables) can be classified into a certain type of **categories** such as discrete or nominal type.
- It is the most commonly and frequently found in the 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) Churn prediction

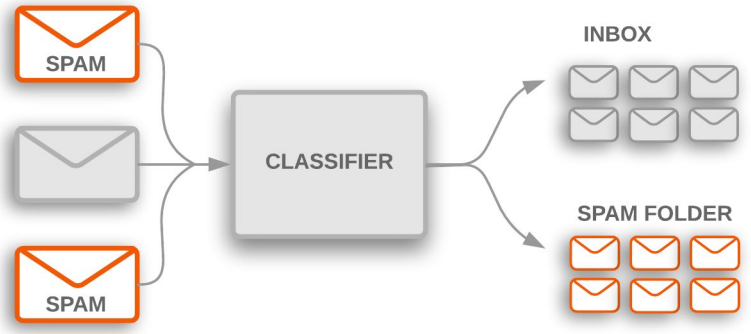
(4) Classifying customers' credit rating

(5) Prediction of occurrence of a certain disease (e.g. cancer)

(6) Prediction of customer reaction to a specific marketing event

(7) Prediction of customer's purchase

(8) …



Types of machine learning algorithm for classification

- There are many different kinds of classification type machine learning algorithms, and some of them 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 the Bayes' theorem that makes classification towards the higher probability by expressing the conditional probability of objective variables (or response variables) as multiplication of the prior probability and likelihood function. All of the observed values are assumed to statistically independently occur 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 that are most associated with objective variables.	Divide & Conquer

Types of machine learning algorithm for classification

- There are many different kinds of classification type machine learning algorithms, and some of them 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
Xboost	An ensemble method that builds a predictive model by combining the predictions of multiple individual models, often decision trees, in an iterative manner.	Ensemble model

Unit 3.

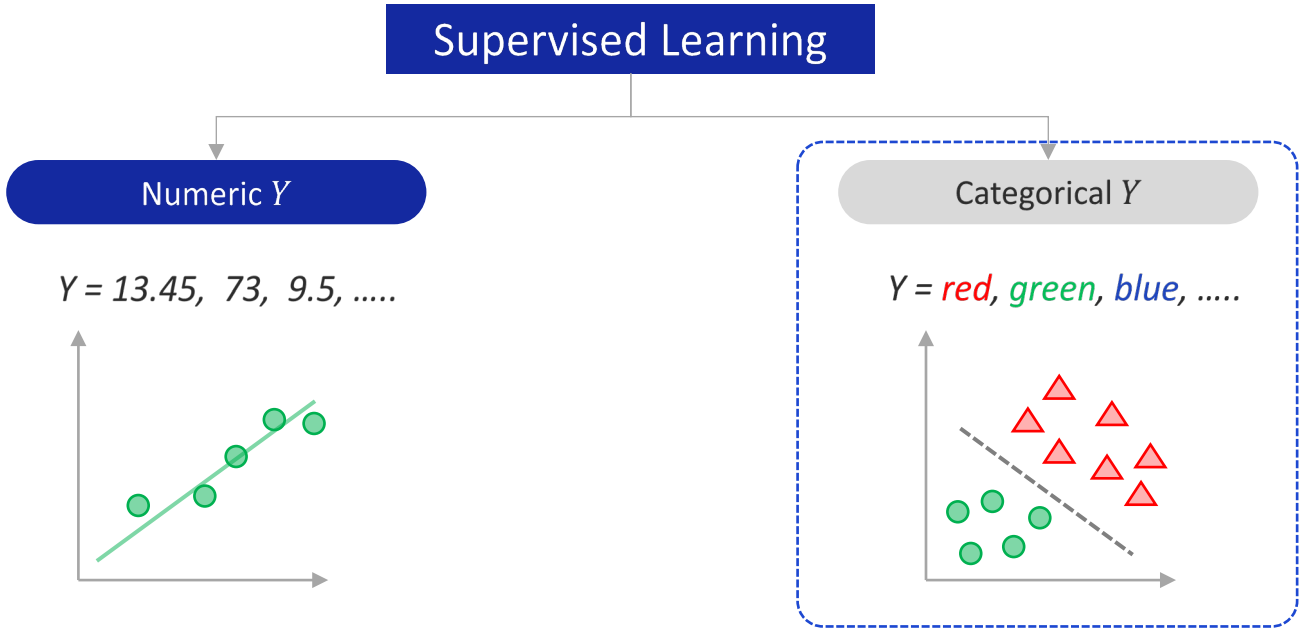
Application of Supervised Learning Model for Classification

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

Logistic Regression Basics

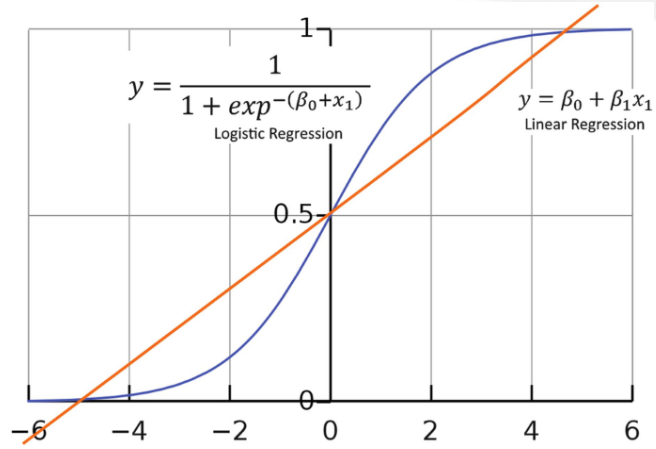
What is logistic regression analysis?

- In contrast to general linear regression analysis is 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 y value which is an objective variable, but it is P(Y=i), which is the probability of objective variable y to become a certain category (i).



About logistic regression

- There is one ore more explanatory variables: $X_1, X_2, \ldots, X_k$
- 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.

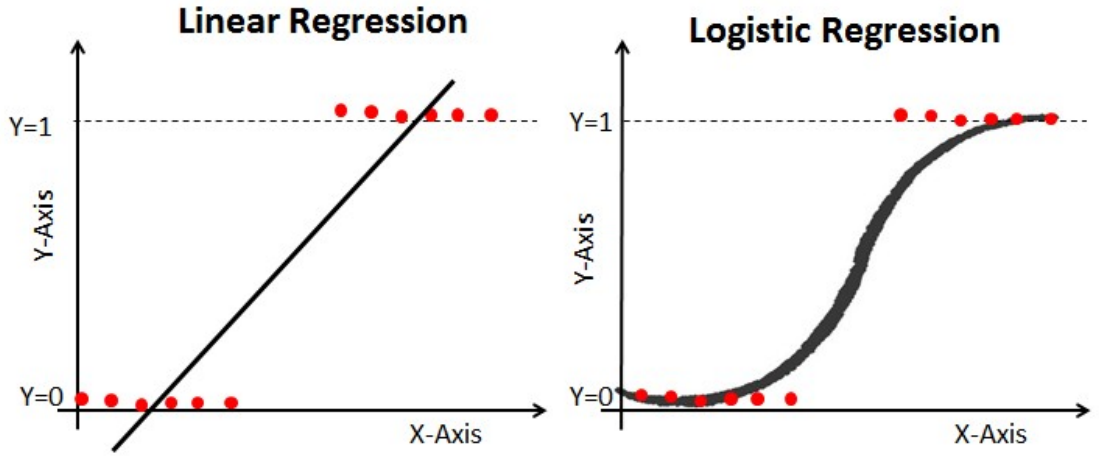


Pros

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

Cons

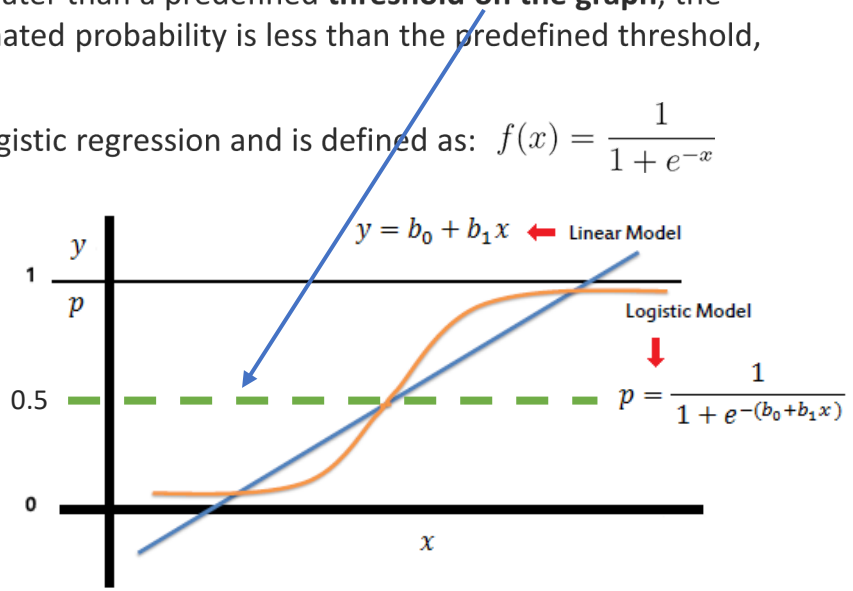
- Not among the most accurate classification algorithms.
- Assumes that the explanatory variables are independent without multi-collinearity.



About logistic regression

- The linear combinations of variables X_i is the so-called “Logit” denoted here as $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$
- Logistic regression uses a **logistic function** called a sigmoid function to **map predictions and their probabilities**. The sigmoid function refers to an S-shaped curve that converts any real value to a range between 0 and 1.
- If the output of the sigmoid function (estimated probability) is greater than a predefined **threshold on the graph**, the model predicts that the instance belongs to that class. If the estimated probability is less than the predefined threshold, the model predicts that the instance does not belong to the class.
- The sigmoid function is referred to as an activation function for logistic regression and is defined as: $f(x) = \frac{1}{1+e^{-x}}$
- The following equation represents logistic regression:

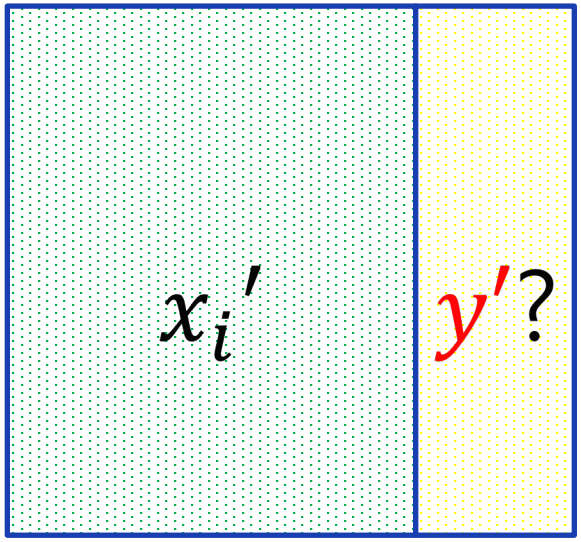
$$f(Y) = \frac{1}{1+e^{-Y}} = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$$



- This equation is similar to linear regression, where the input values are combined linearly to predict an output value using weights or coefficient values. However, unlike linear regression, the output value modelled here is a binary value (0 or 1) rather than a numeric value.

Logistic regression training and prediction (testing)

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

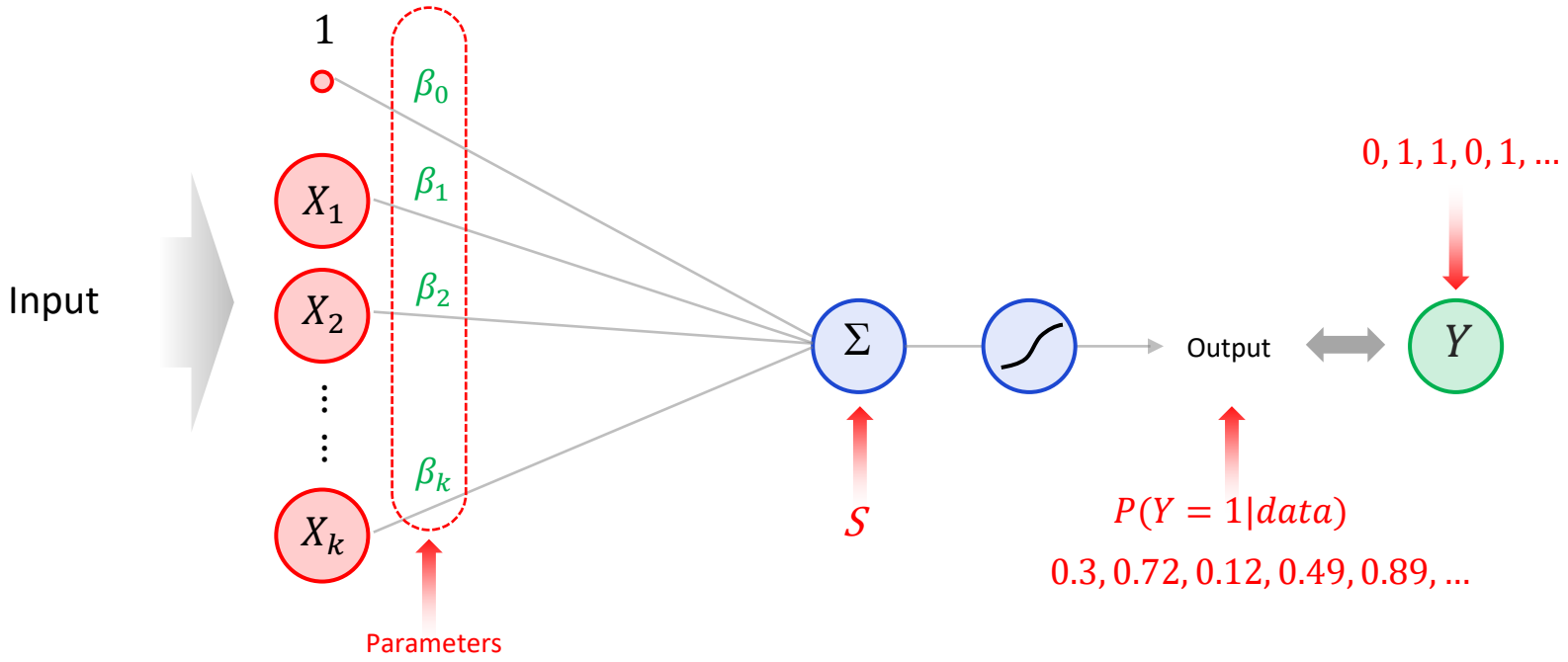


Compare the conditional probability with a cutoff.

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

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

Logistic regression training and prediction (testing)



Unit 3.

Application of Supervised Learning Model for Classification

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

Note

- Accuracy alone is not sufficient for testing.

  Ex If frauds constitute only 1% of all the transactions, a fraud detection system (FDS) that predicts as non-fraud all of the transactions, the accuracy would be quite high at 99%.

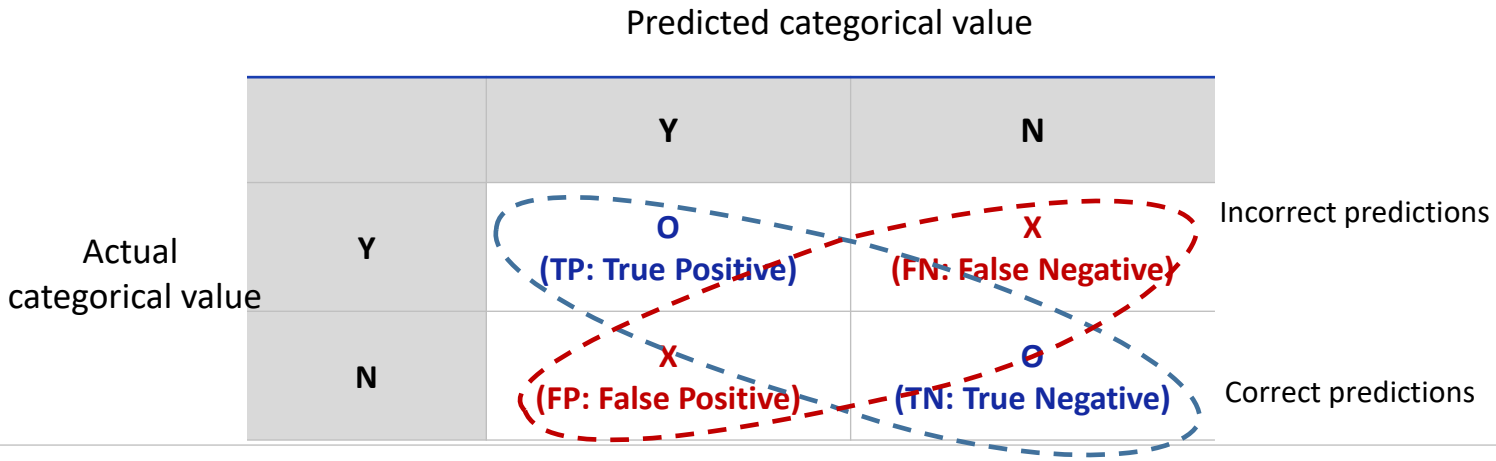
  However, such FDS would be useless because it misses out that 1% that really matters.

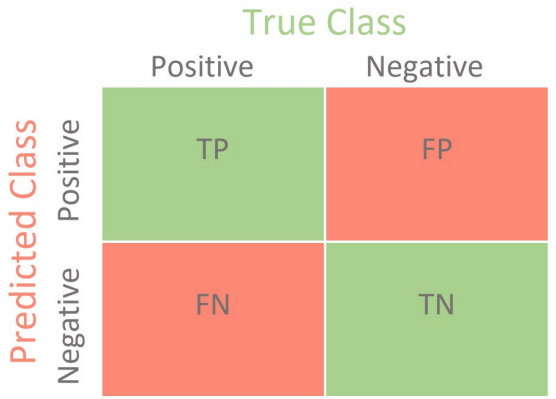
- We should also consider metrics other than just the accuracy.

Logistic Regression Performance Metrics

Confusion matrix

- In the classification machine learning method, the most commonly used method for results evaluation of analysis model is the metric calculation including classification accuracy by using confusion matrix.
- The confusion matrix refers to the matrix which makes a crosstable of predicted classification category from the analysis model and actual classified category of data.
- Inaccurate classification of uninterested categories into interested categories is called FP (False Positive), and inaccurate classification of interested categories into uninterested categories is called FN (False Negative).
- There are various kinds of metrics based on different combinations of TP, TN, FP, FN of the confusion matrix for analysis result evaluation of classification machine learning methods.





Metrics

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)
recall = 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, 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)	$\{2 * (precision) * (recall)\} \div (precision + recall) = \frac{(2 * TP)}{2 * TP + FP + FN}$	Ranged between 0~1, it is the harmonic mean between precision and sensitivity (recall). If both of the precision and sensitivity are high, f-Measure also tends to have a larger value.

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 (**160 test examples**).

Confusion matrix: Accuracy

Ex

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

$$\text{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ predictions} = \frac{140}{160} = 0.875$$

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

Confusion matrix: Recall/Sensitivity

Ex

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

True Positive Rate = Sensitivity = **Recall** = $\frac{Number\ of\ correctly\ predicted\ 1s}{Total\ number\ of\ 1s} = \frac{20}{25} = 0.8$

Confusion matrix: Precision

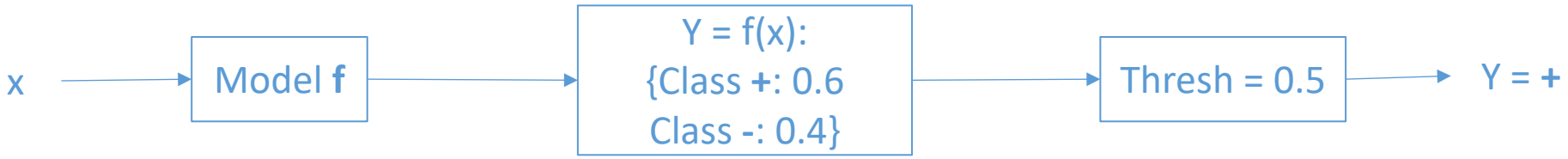
Ex

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

Positive Predicted Value = **Precision** = $\frac{Number\ of\ correctly\ predicted\ 1s}{Total\ number\ of\ 1s} = \frac{20}{35} = 0.57$

Crisp and Soft Classifiers & ROC curve

- A "**hard**" or "**crisp**" classifier predicts a class between a set of possible classes.
- A "**soft**" or "**scoring**" classifier (probabilistically) predicts a class, but accompanies each prediction with an estimation of the reliability (confidence or class probability) of each prediction.
- Most learning methods can be adapted to generate this confidence value.
- **A soft classifier can be converted into a crisp classifier using a threshold.**
  - Example: "if score > 0.7 then class A, otherwise class B".
  - With different thresholds, we have different classifiers, giving more or less relevance to each of the classes



Coding Exercise #0307

Follow practice steps on 'ex_0307b.ipynb' file.

Unit 4.

Decision Tree

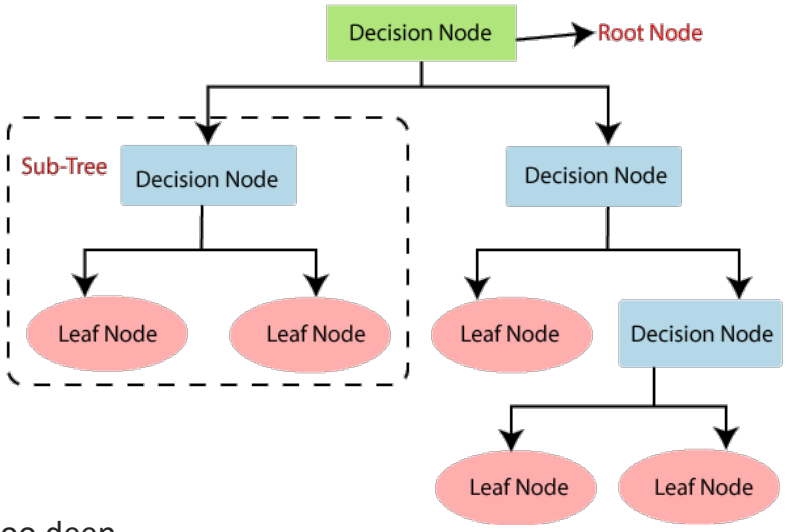
4.1. Tree Algorithm

Overview of Decision Tree

Definition

- A decision tree is a popular machine-learning algorithm used for both **classification** and **regression** tasks.
- It is a **tree-like** structure that consists of **nodes**, **branches**, and **leaves**.
  - The nodes represent **features or attributes**,
  - branches represent **decisions** based on these attributes,
  - and leaves represent the final outcome or **class label**.

RULE INDUCTION BASED ON PARTITION



Pros

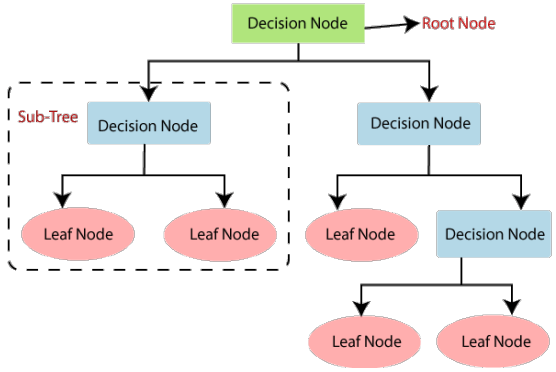
- Easy to understand and interpret.
- Capable of handling both numerical and categorical data.
- Robust to outliers and missing values.

Cons

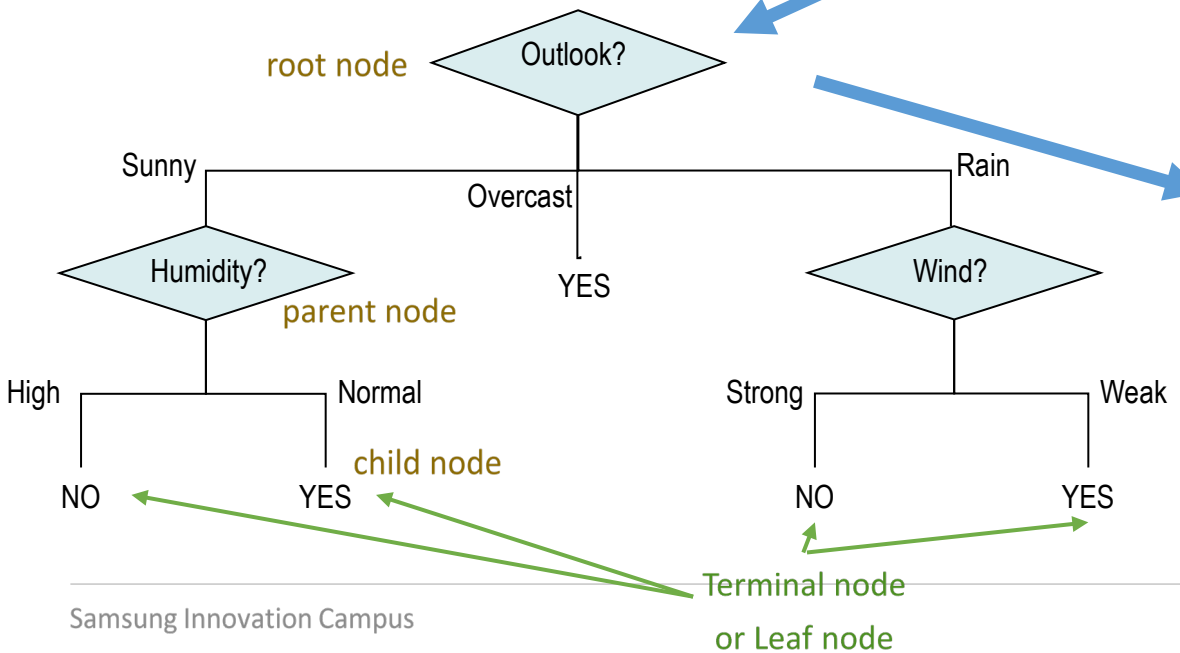
- They can be prone to overfitting, especially when the tree becomes too deep.
- They can be sensitive to small variations in the data, leading to different tree structures.
- They may not be the most accurate model, but they are often used as a base learner in ensemble methods like Random Forests and Gradient Boosting Machines to improve overall performance.

Composition

- Node, Branch, Depth



Day	Outlook	Temp.	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



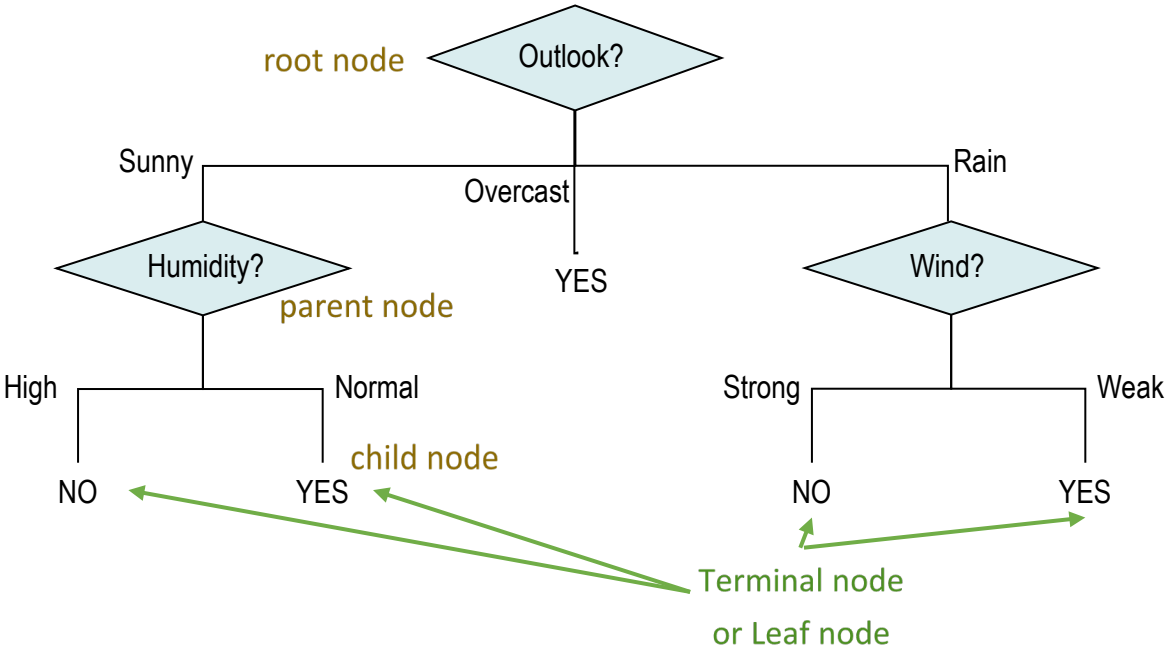
Each path from the root is a rule:

IF Sky=Overcast
  THEN PlayTennis=yes
  ELSE IF Sky=Sunny
    THEN IF Humidity=High THEN PlayTennis=no
      ELSE PlayTennis= yes
    ELSE IF Wind=Strong THEN PlayTennis=no
      ELSE PlayTennis= yes

Composition

- Node, Branch, Depth

Day	Outlook	Temp.	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



Construction & use of a decision tree

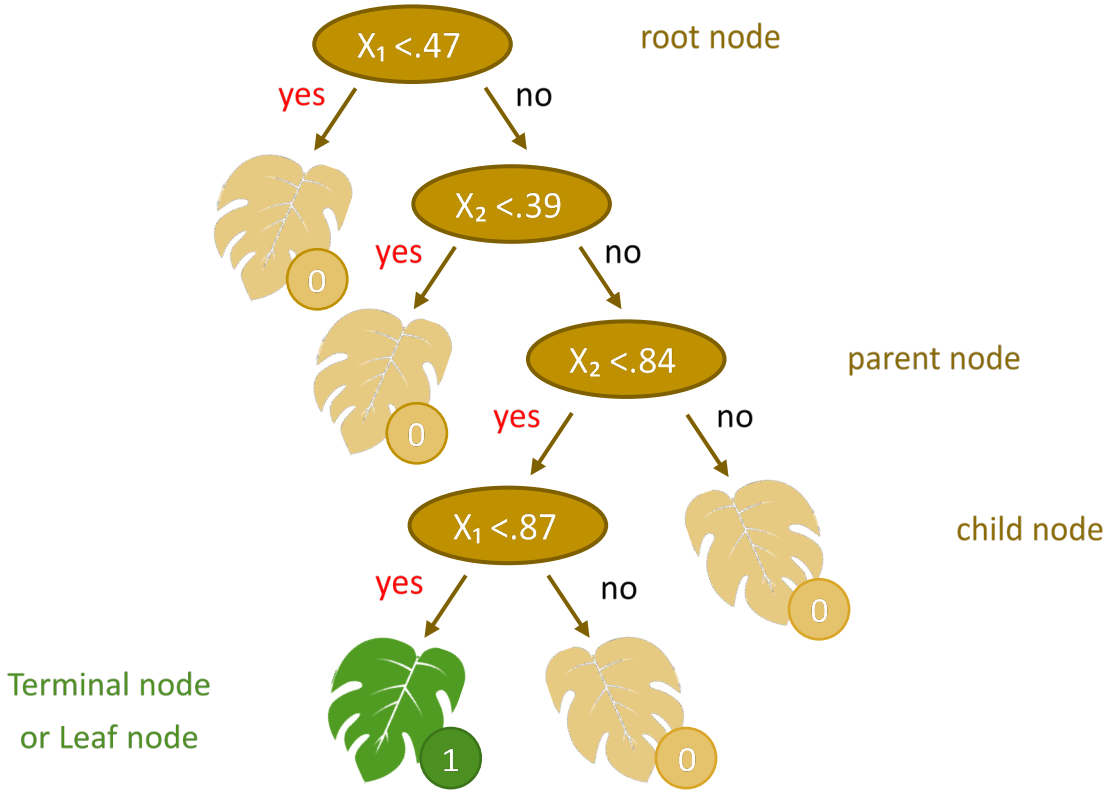
- The main steps in building a decision tree are:

1. **Select the best attribute** to split the data based on a chosen criterion (e.g., Gini impurity, information gain).
2. **Partition the data** into subsets according to the selected attribute.
3. **Recursively apply the above steps** on each subset until stopping criteria are met.

- To make **predictions** using a decision tree, an input data point is passed through the tree, following the branches based on the values of its features.
- The final outcome is determined by the majority class or the mean target value in the corresponding leaf node.

Composition

- Node, Branch, Depth

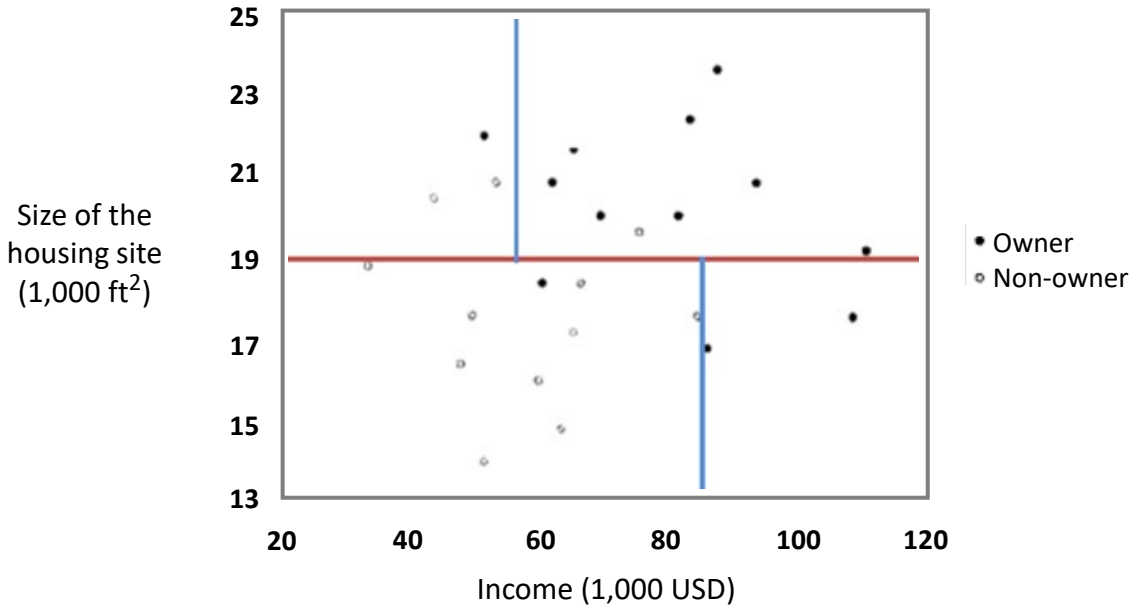


Repetitive splitting process

Purpose

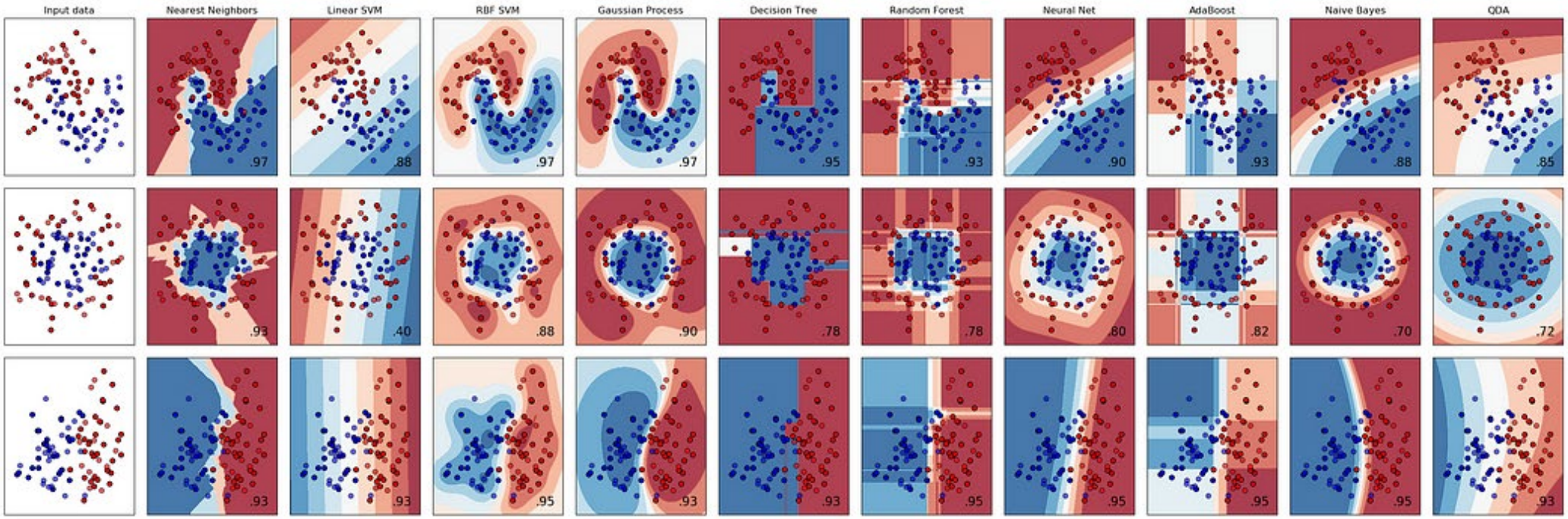
- Divide the entire space into rectangles and make each of it as pure or homogeneous as possible.
- Definition of 'pure':
  - Divide the area into pure, or homogenous, rectangular spaces as much as possible
  - All variables in the final rectangle

    belong to the same group.



Repetitive splitting process

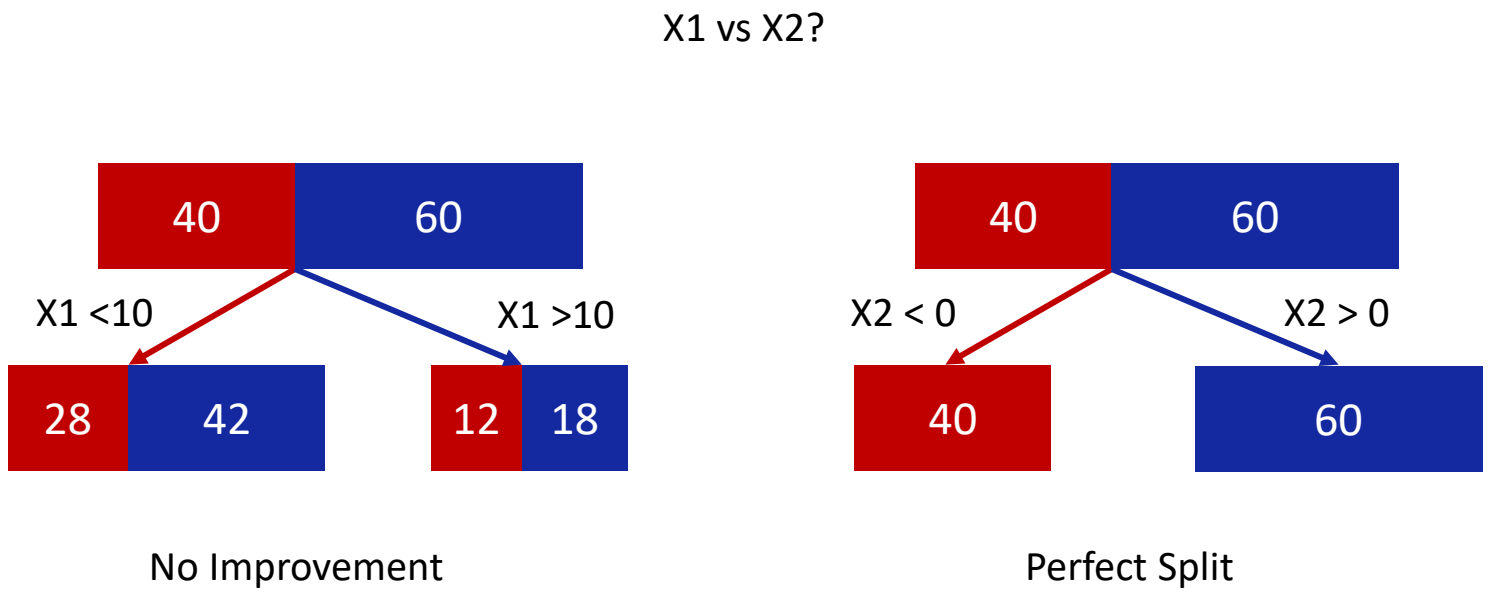
Decision boundaries by technique



Split criterion of the decision tree

Split criterion

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



Split criterion

- Which attribute do we choose at each level?
  - Depends on the splitting criterion different algorithms exist: (ID3, C4.5, CART)
    - **Entropy:** measures homogeneity of examples.
    - **Information gain:** measures the expected reduction in entropy when choosing attribute, A to split D
    - **Chi squared statistic – p value**: Creates child nodes with predictor variable with the lowest P value and the optimal partitioning
    - **F-statistic in ANOVA**: creates child nodes with predictor variable with the lowest P value
    - **Variance reduction**: Creates child nodes with the optimal partitioning that maximizes variance reduction

	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	

- In order to evaluate how well the data is separated, specific criterion is required. In general, impurity is the evaluation criterion, and the impurity becomes higher as various classifications are mixed in the node, and it is the lowest when there's only one classification in the node.

Impurity measure

Gini index

- Selects child nodes with **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 , …, pk, it is expressed as the following equation.

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

GI = 1 –(1/6)²–(1/6)²–(2/6)²–(2/6)²= .73

GI = 1 –(6/7)²–(1/7)²= .24

Entropy measure

- In thermodynamics, entropy measures degree of disorder.
- Creates child nodes with 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 , …, pk, 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 terminal node without further splitting

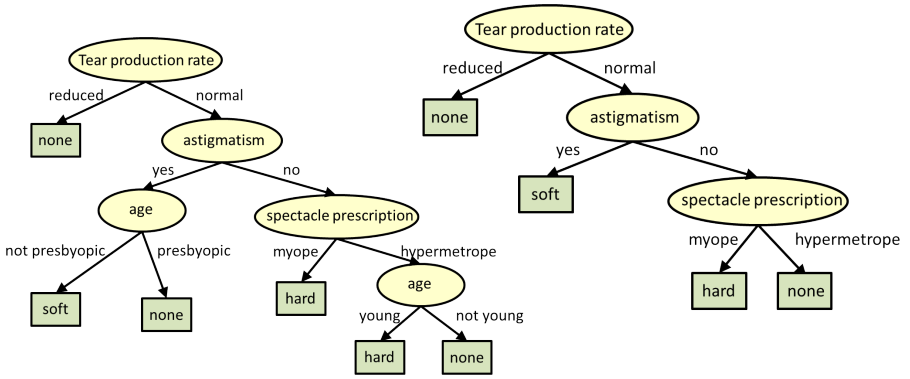
- Designates the depth of decision tree
- Designates the minimum number of records in the terminal node

Pruning

- To remove some of the branches or subtrees that do not significantly contribute to the predictive power of the model, resulting in a simpler, more general and more interpretable tree.

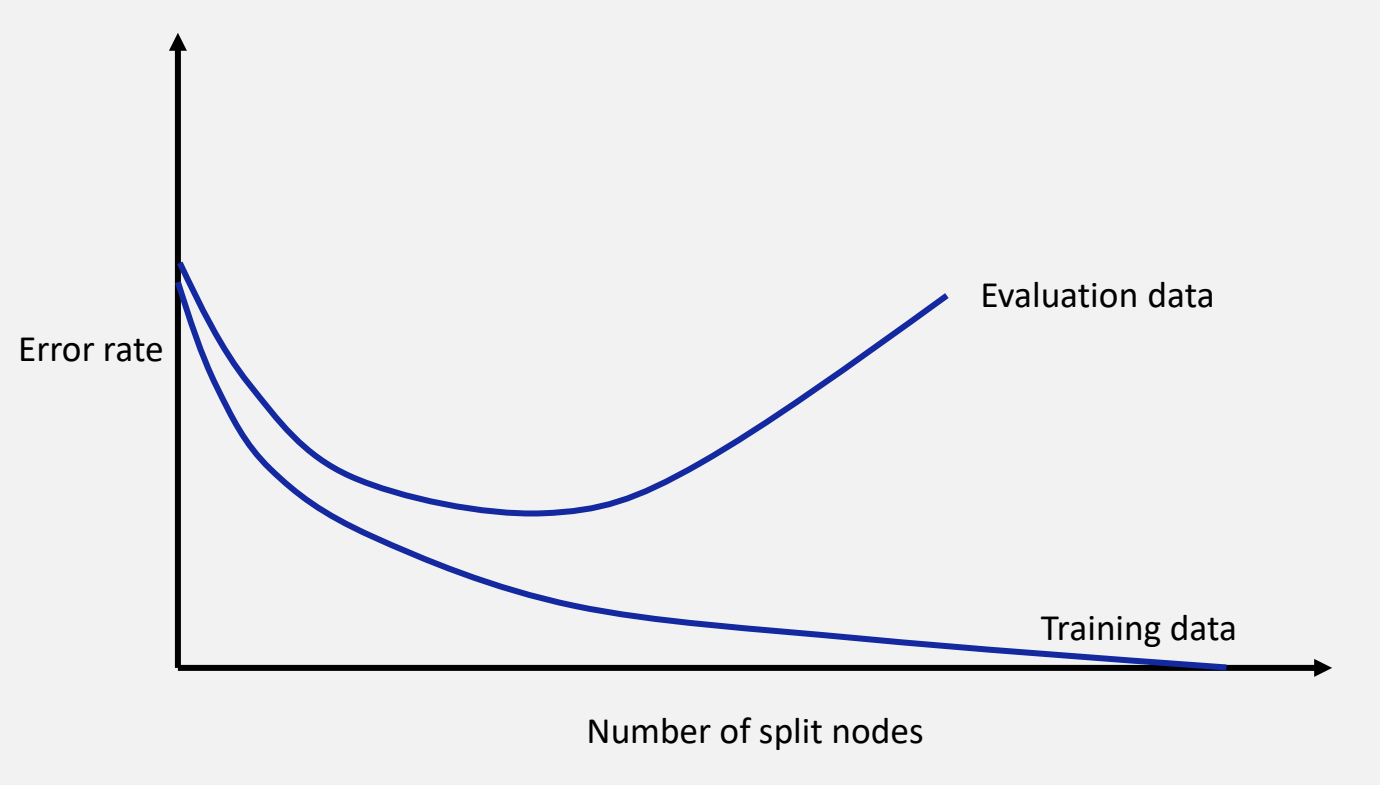
- **Pre-pruning.** This method stops the growth of the tree during the construction process. Pre-pruning involves setting criteria such as **maximum depth**, **minimum number of samples per node**, or **minimum improvement in impurity** to decide when to stop splitting a node further.

- **Post-pruning.** The decision tree is first grown to its full size without any constraints. Then, the tree is pruned by iteratively removing branches or subtrees in a bottom-up manner that contributes the least to the model's performance (validation error or cost-complexity measure).



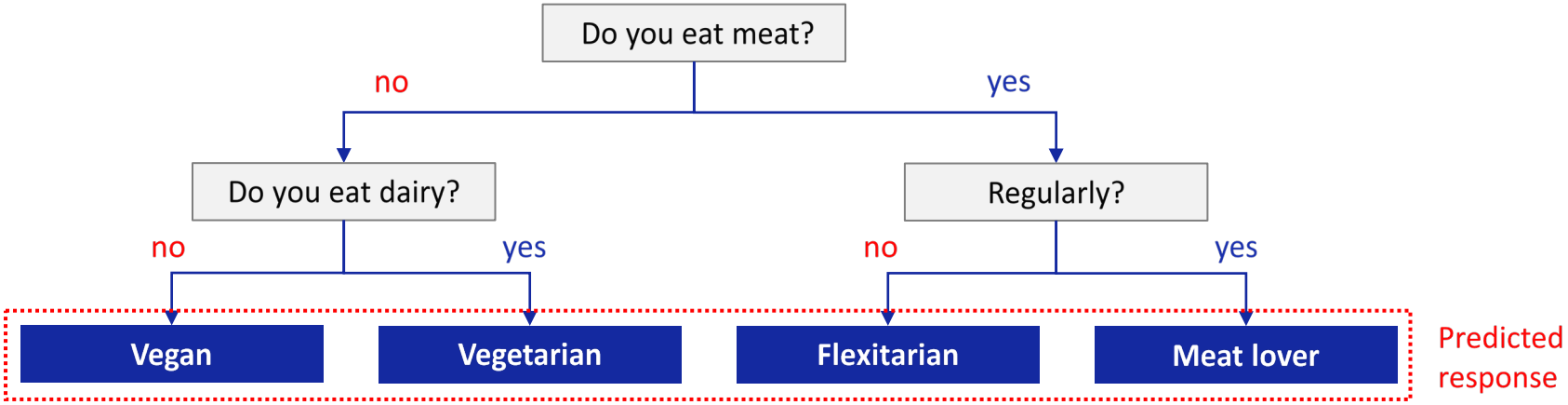
Overfitting problem

Overfitting problem graph



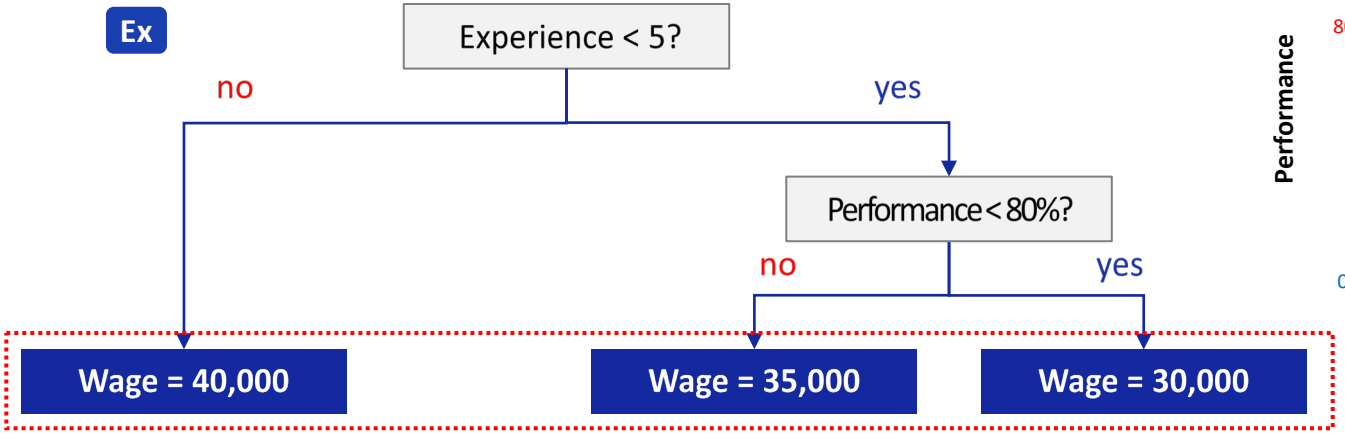
Classification Tree

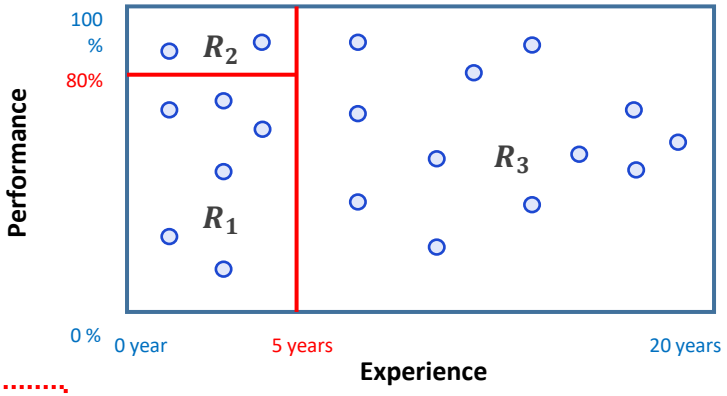
Ex



Regression Tree

Ex





The predicted value in the region R_i is given by the average in that region.

Scikit-Learn DecisionTreeClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
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.
max_leaf_nodes	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

Coding Exercise #0308

Follow practice steps on 'ex_0308.ipynb’ file.

Unit 6.

KNN Algorithm

6.1. KNN Algorithm

KNN Algorithm

About KNN (K Nearest Neighbors)

- KNN works by finding the 'k' nearest neighbors (data points) in the training dataset to a given test data point, based on a distance metric like **Euclidean**, **Manhattan**, or **Minkowski distance** (a generalization of the previous ones).
  - It is considered a lazy learning algorithm because it **does not build an explicit model during the training phase**.
  - Instead, it stores the entire training dataset and performs calculations during the prediction phase.
- There are classification and regression variants.
  - The output for the test data point is then determined by the majority class label (in **classification**)
  - or the average value (in **regression**) of these 'k' nearest neighbors.

Pros

- Simple and easy to understand.
- Works well with small datasets and low-dimensional data.
- Automatically adapts to non-linear decision boundaries.
- Requires minimal training time since there is no explicit model building.

Cons

- Since there is no “model”, little insight can be extracted.
- The training dataset is required for prediction.
- Prediction is not efficient ⇨ “Lazy algorithm”.

About KNN (K Nearest Neighbors)

- For KNN, specific criteria are required for how to measure the '**analogy**' between the certain data set and surrounding data set and **how many data points** will be used for 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 inversing the squared **Euclidean distance** between two points. If the points are discrete variables, use **Jaccard coefficient**.

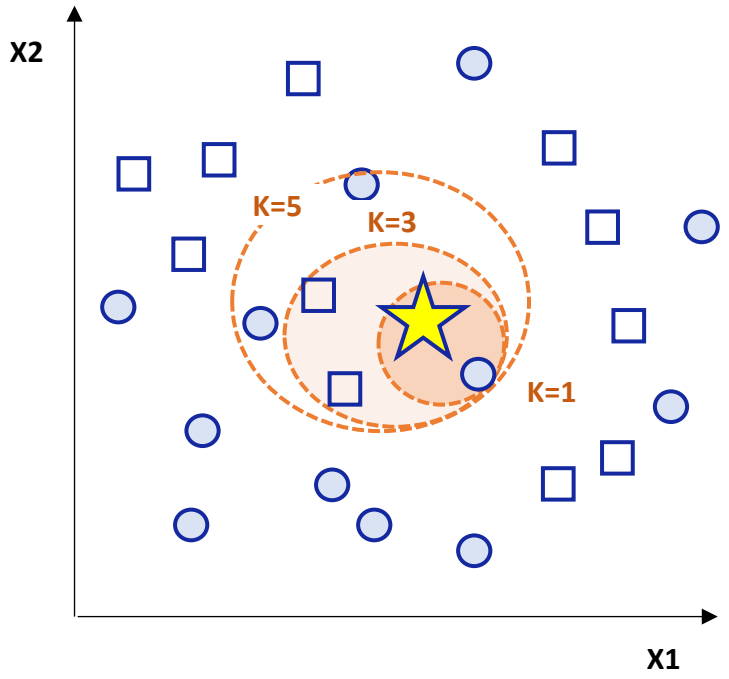
2) Objective variable classification criteria

- **K** in the KNN refers to **number of surrounding data points** that will be used for classifying objective variables of the specific data point after measuring the analogy between the specific data point and other surrounding data sets.

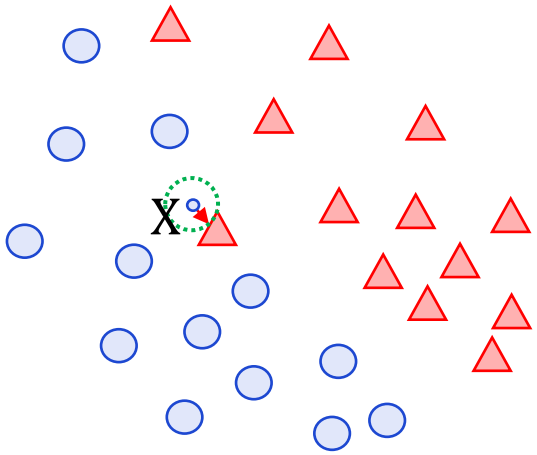
Ex Take movie recommendation as an example, where it is the recommendation of a movie that is similar to the customer's favorite movie and the customer did not watch yet. If so, when considering only one movie that is the most similar to the customer's favorite, it is '1-nearest neighbor,' and when considering three similar movies, it would be '3-nearest neighbor.'

About KNN (K Nearest Neighbors)

- The figure is a conceptual expression of the change in objective variables depending on K value setting in the K-nearest neighbor method.
  - The '☆' point is the data value that needs to be classified, and the points shaped in a square and circle are other data points present around.
    - If **K=1**, since the closest data point to the star is a circle, the objective variable of '☆' will be classified as 'ㅇ.'
    - If **K=3**, three points that are closest to '☆' will be considered, which include two squares and one circle. So, in this case, '☆' will be classified as '□'.
    - If **K=5**, because there are more circles around '☆,' it will be again classified as 'ㅇ'.
- Different K values significantly change the predicted result of the objective variable category. Therefore, setting an appropriate K value is important in the K-nearest neighbor method.
- There are no clear theoretical or statistical standards for an appropriate K value.
  - In general, a random initial K value is designated between K=3 and K=9, which is then used to test the classification performance by using training and evaluation data for selecting the optimal K value.

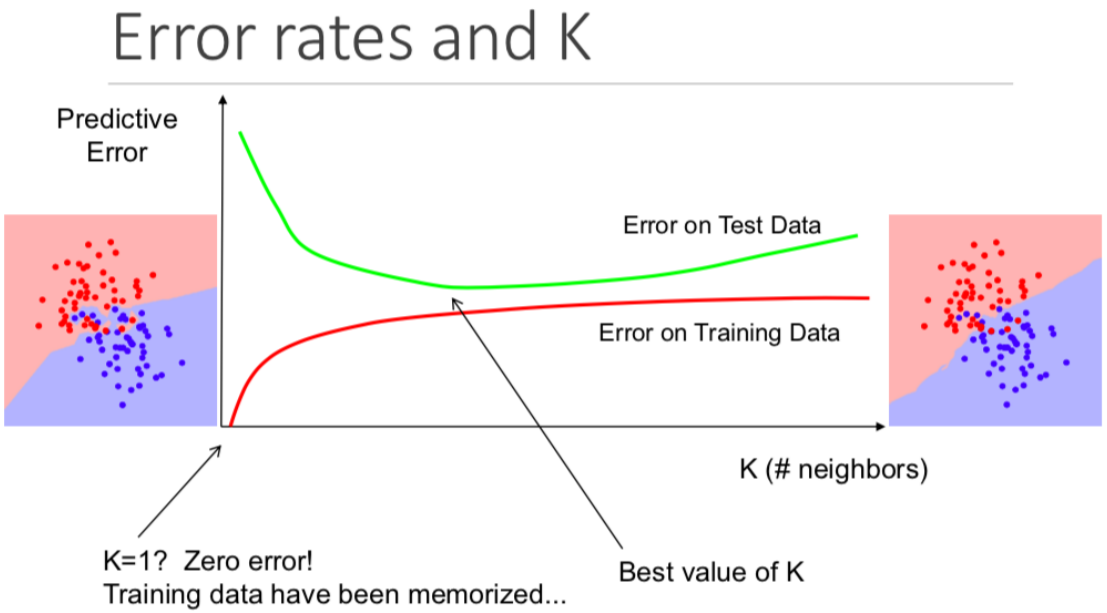


KNN classification algorithm



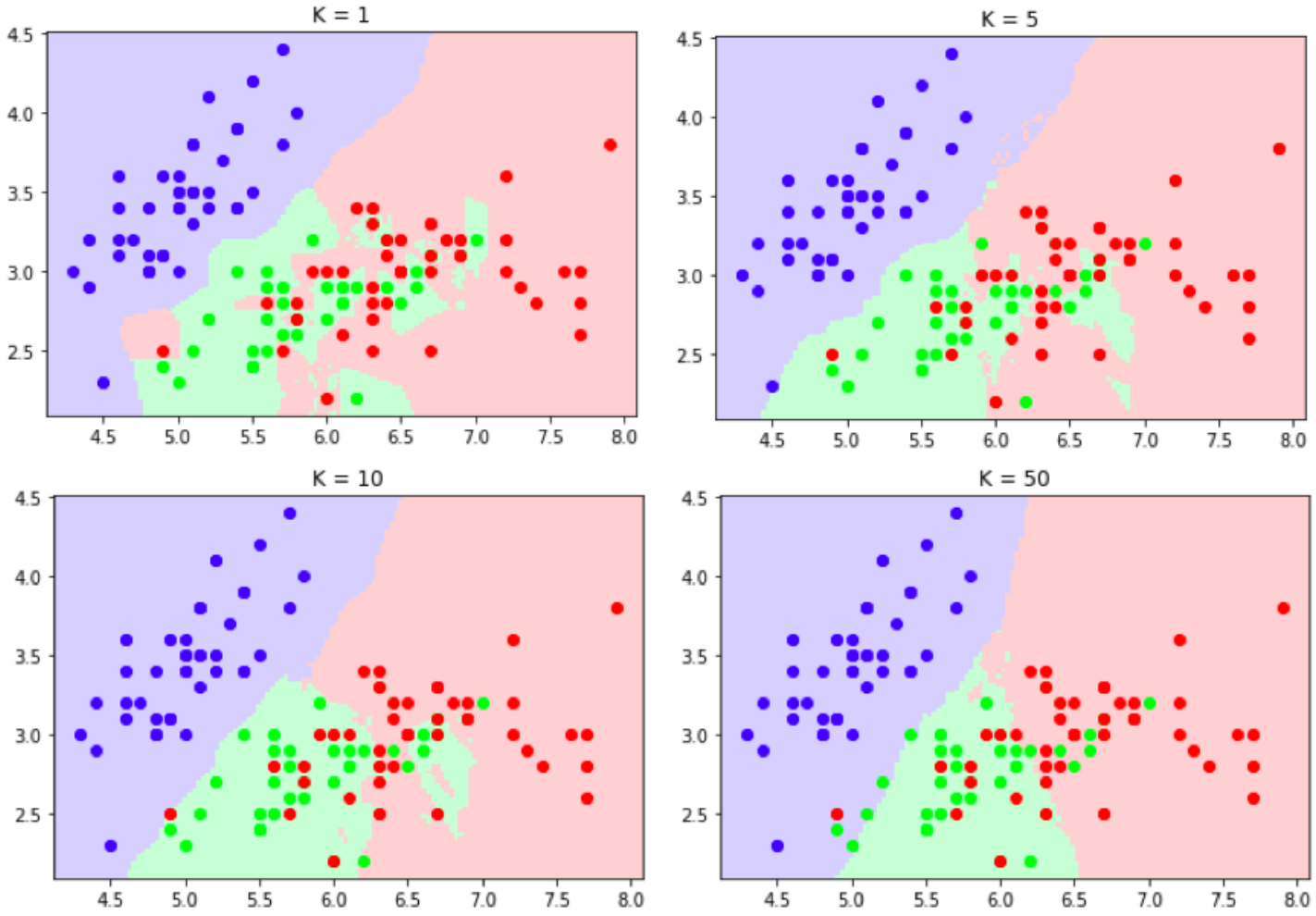
When $k = 1$,

X is classified as △



KNN classification algorithm

- We can see that when **K is small**, there are some outliers of green label are still green, and outliers of red label are still red.
- When **K becomes larger**, the boundary is more consistent and reasonable.



Coding Exercise #0310

Follow practice steps on 'ex_0310.ipynb’ file.

Unit 6.

SVM Algorithm

6.1. SVM Algorithm

SVM Algorithm

About SVM (Support Vector Machine):

- The main idea behind SVM is to find an optimal hyperplane that separates the classes with the maximum margin.
- For linearly separable data, the SVM algorithm finds a linear hyperplane. However, real-world data is not linearly separable.
- To handle such cases, SVM uses a technique called the kernel trick. The kernel trick implicitly maps the input data to a higher-dimensional space where the data becomes linearly separable.

Pros

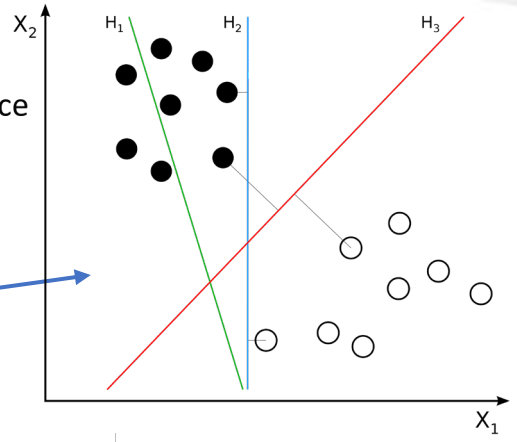
- Effective in high-dimensional spaces.
- Can handle non-linear decision boundaries using the kernel trick.
- Robust to outliers and overfitting due to the maximization of the margin.
- Only support vectors are relevant for making predictions, reducing memory requirements.

Cons

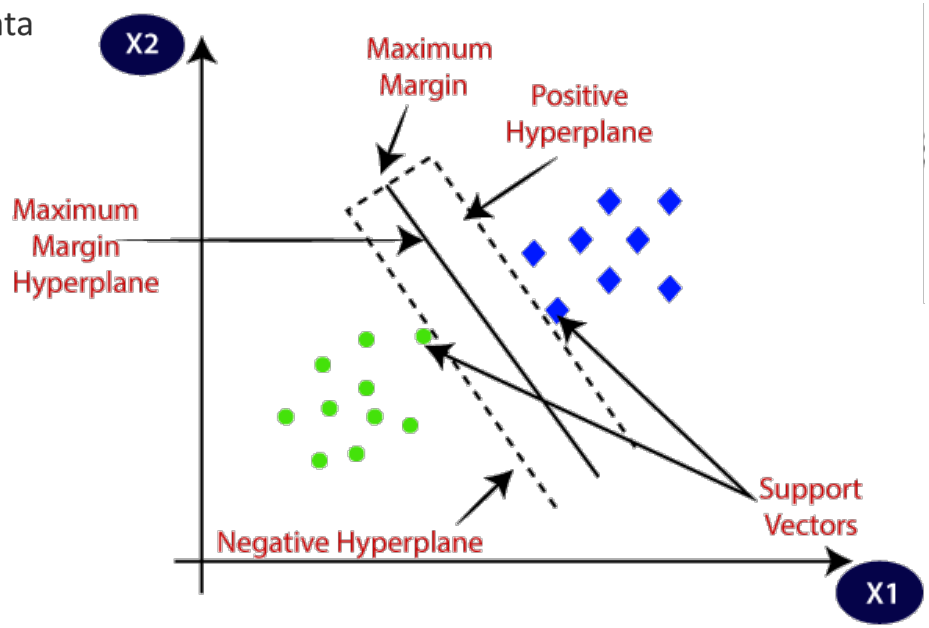
- Computationally expensive, especially for large datasets and non-linear kernels.
- Choosing the right kernel and hyperparameters can be challenging.
- Limited interpretability compared to other algorithms like decision trees.
- Performs poorly with imbalanced datasets, although this can be mitigated using techniques like class weighting and oversampling.

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.
- SVM model **finds the hyperplane that split the data in two different categories** with maximum margin to classify data.
- There would be **many lines or planes that split data in two different categories.**

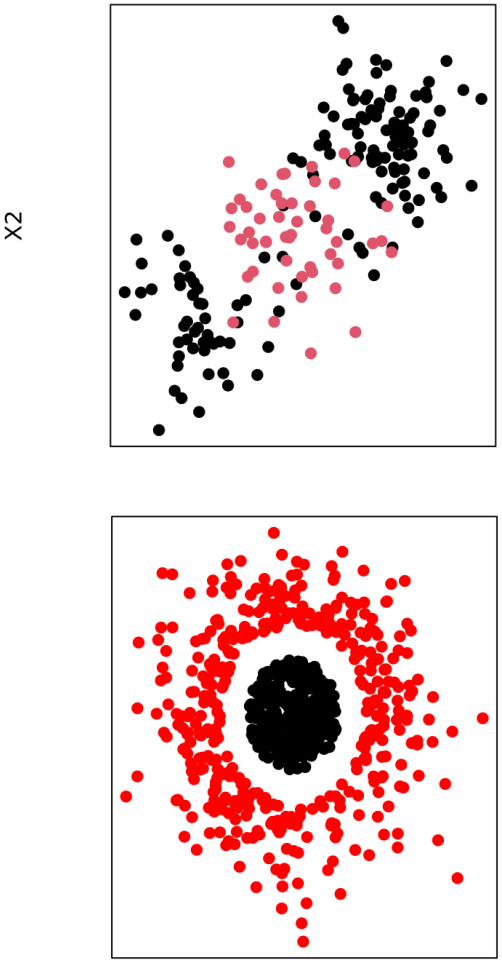


- The hyperplane should be positioned with the maximum distance to the data points.
- The data points with the minimum distance to the hyperplane are called **Support Vectors**.



About SVM (Support Vector Machine)

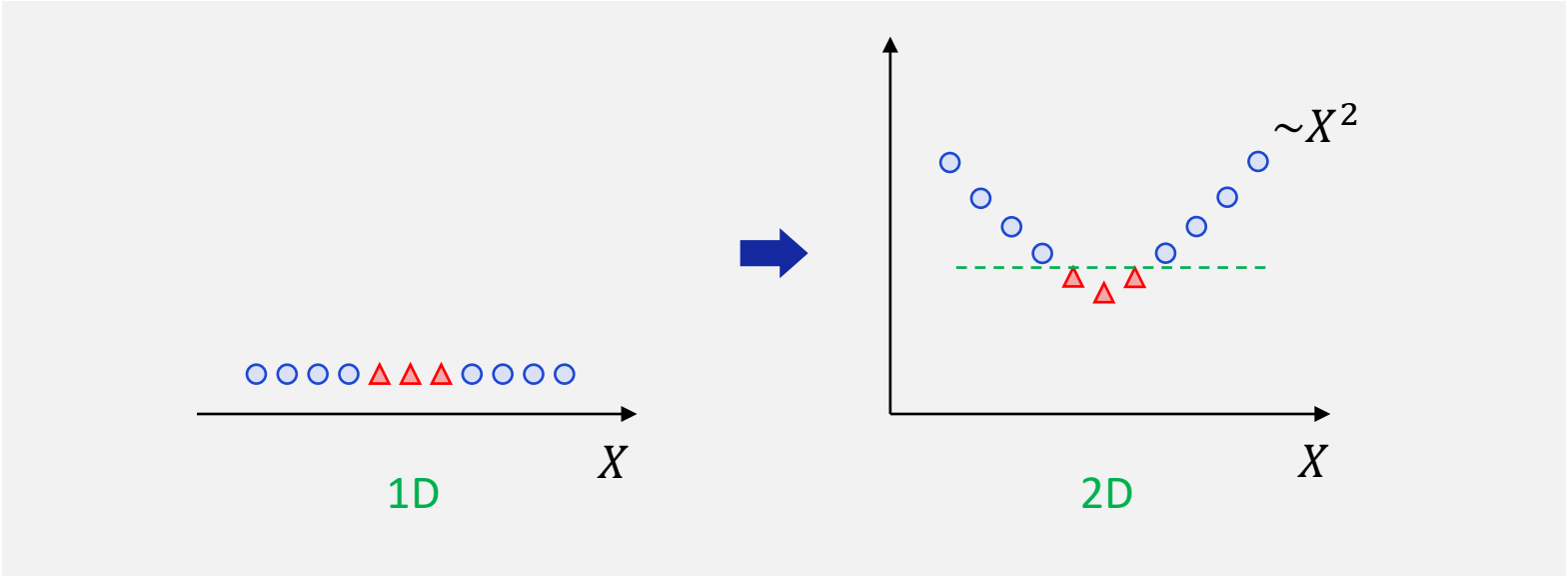
- However, it is not always possible to classify all data linearly. Sometimes, data classification should be done in a curve or more complex non-linear classification plane.
- If so, use the **Kernel trick** method for mapping the given data into a higher dimension and find a hyperplane that can classify the data in the converted dimension.
- The major Kernel functions include **polynomial** Kernel, **Gaussian** Kernel, **sigmoid** kernel, etc.



Kernel

- Mapping to a higher dimension using the “kernel” functions.
- Kernel functions introduce an effective non-linear classification boundary.

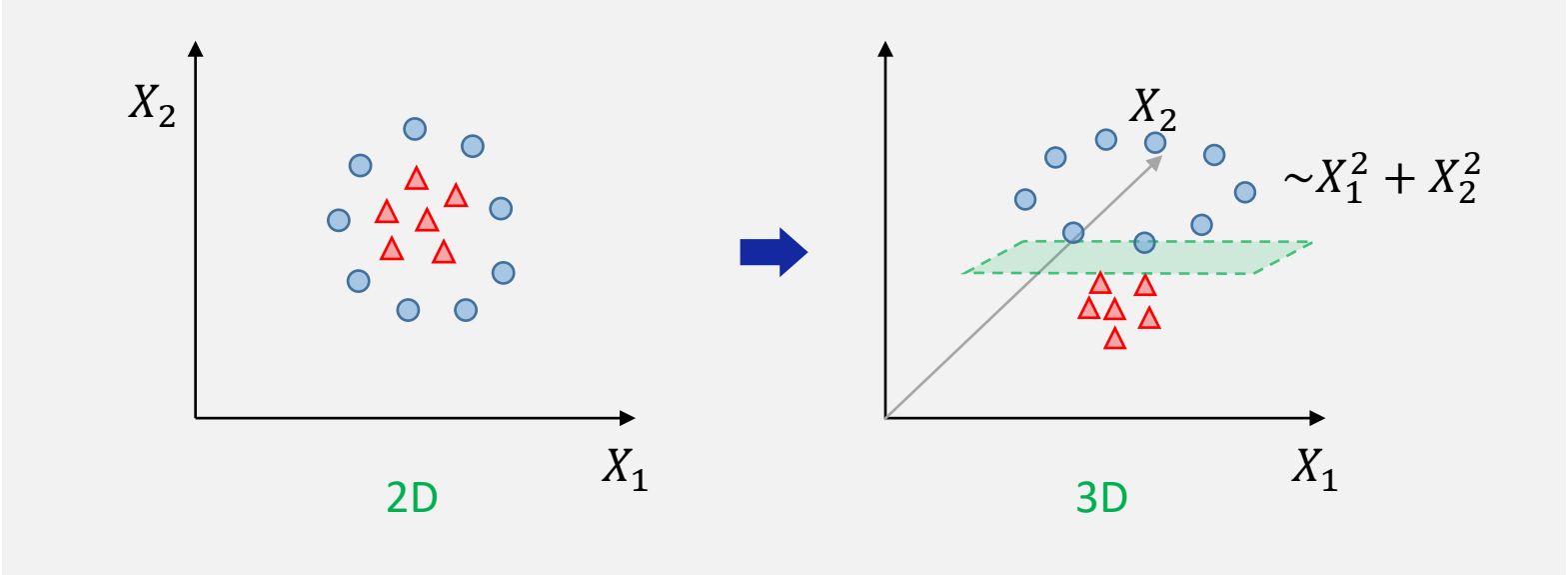
Ex Polynomial kernel.



Kernel

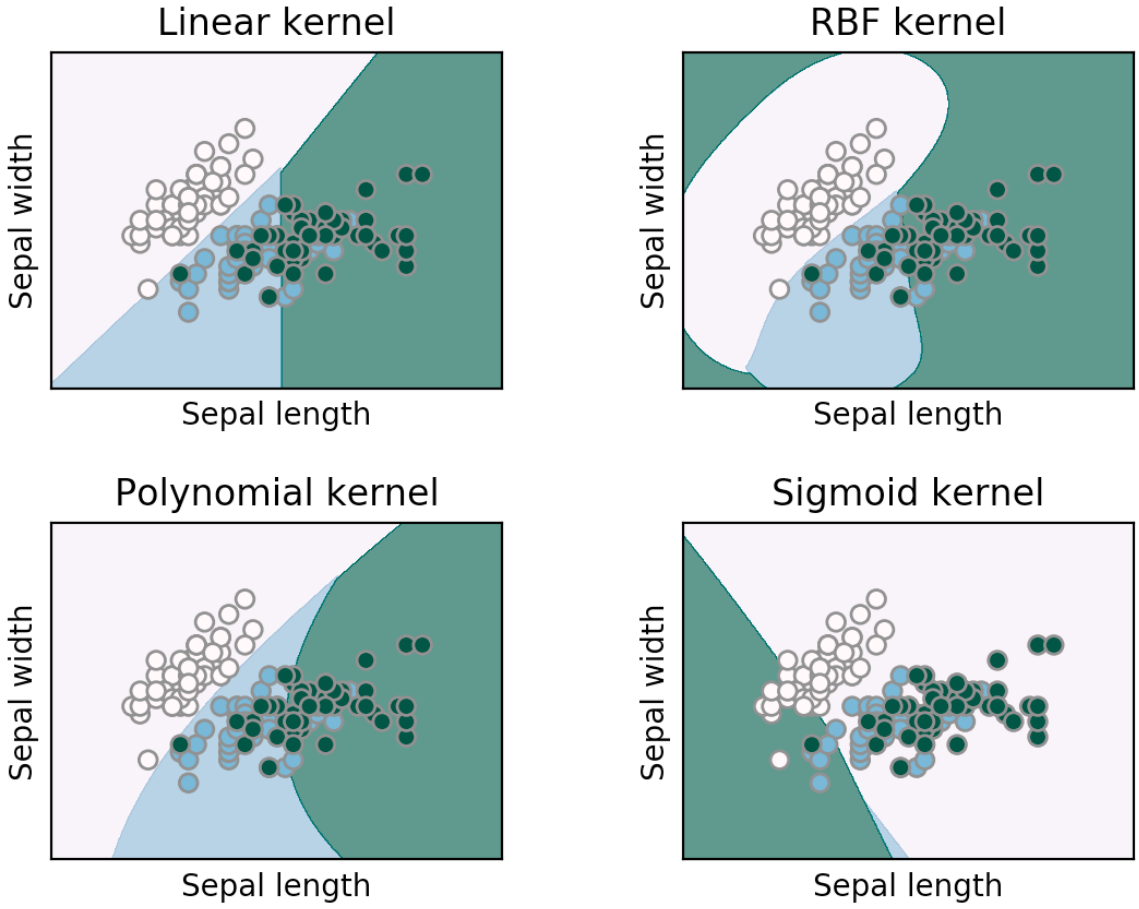
- Mapping to a higher dimension using the “kernel” functions.
- Kernel functions introduce an effective non-linear classification boundary.

Ex Polynomial kernel.



Kernel

- Effective mapping to a higher dimension by giving the inner product of two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.
  - Linear: $K(\mathrm{x}_1, \mathrm{x}_2) = \mathrm{x}_1^{\mathrm{t}}\mathrm{x}_2$
  - Polynomial: $K(\mathrm{x}_1, \mathrm{x}_2) = (\mathrm{x}_1^{\mathrm{t}}\mathrm{x}_2 + \mathrm{c})^{\mathrm{d}}$ where $\mathrm{c} > 0$
  - Sigmoid: $K(\mathrm{x}_1, \mathrm{x}_2) = tanh(\mathrm{a}(\mathrm{x}_1^{\mathrm{t}}\mathrm{x}_2) + \mathrm{b})$ where $\mathrm{a}, \mathrm{b} > 0$
  - Radial function basis (rbf): $K(\mathrm{x}_1, \mathrm{x}_2) = exp(-\gamma\|\mathrm{x}_1 - \mathrm{x}_2\|^2)$ where $\gamma > 0$



Coding Exercise #0311

Follow practice steps on 'ex_0311.ipynb’ file.

Unit 7.

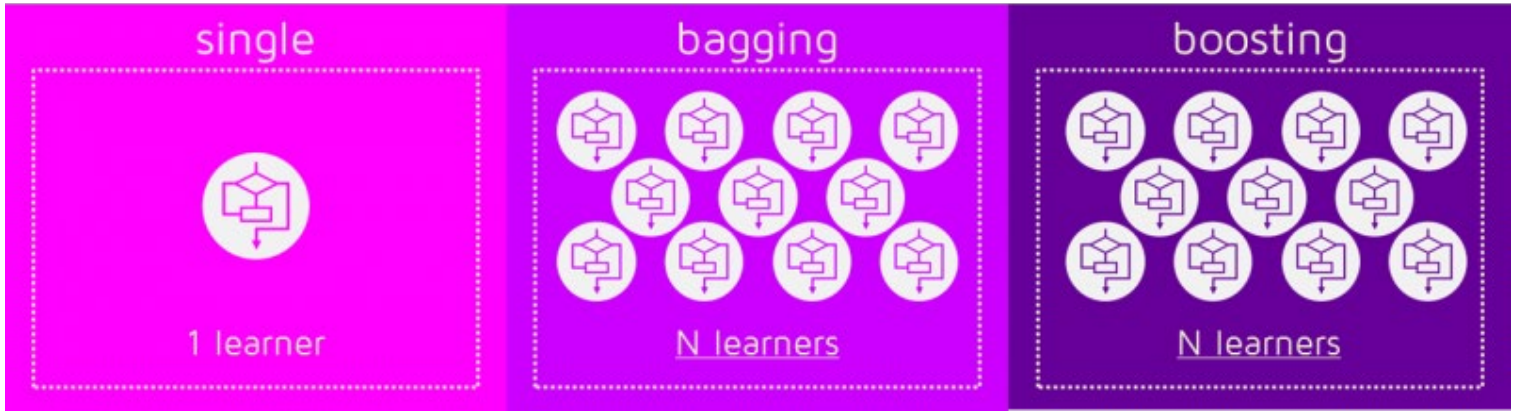
Ensemble Algorithms

*The **Wisdom of Crowds**: human groups are capable of reaching smarter decisions than the smartest of their members would have reached.*

What is Ensemble Learning?

- The idea is to train multiple models using the same learning algorithm.
- Instead of expecting performance results from a single model, the purpose of ensemble learning is to draw a better result by using **collective intelligence of different models** such as **averaging** out or making a decision based on **majority vote**, etc. many different single models
- The main causes of error in learning are due to noise, bias and variance. **Ensemble** helps to minimize these factors. These methods are designed to improve the stability and the accuracy of Machine Learning algorithms.
  - Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.

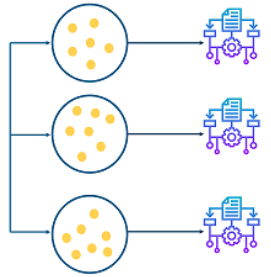
1906 country fair in Plymouth, 800 people participated in a contest to estimate the weight of a slaughtered and dressed ox. Statistician Francis Galton observed that the median guess, 1207 pounds, was accurate within 1% of the true weight of 1198 pounds

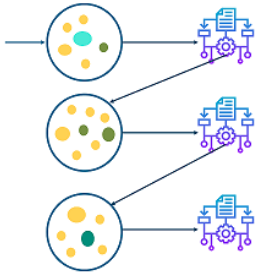


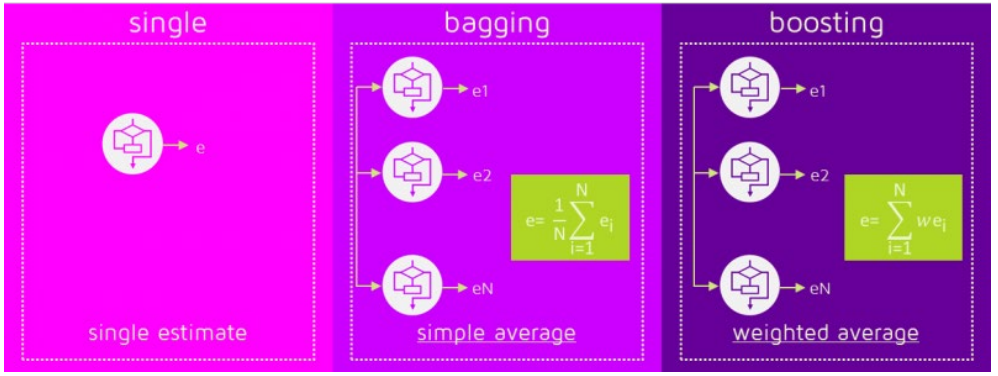
Ensemble algorithms

About ensemble algorithms:

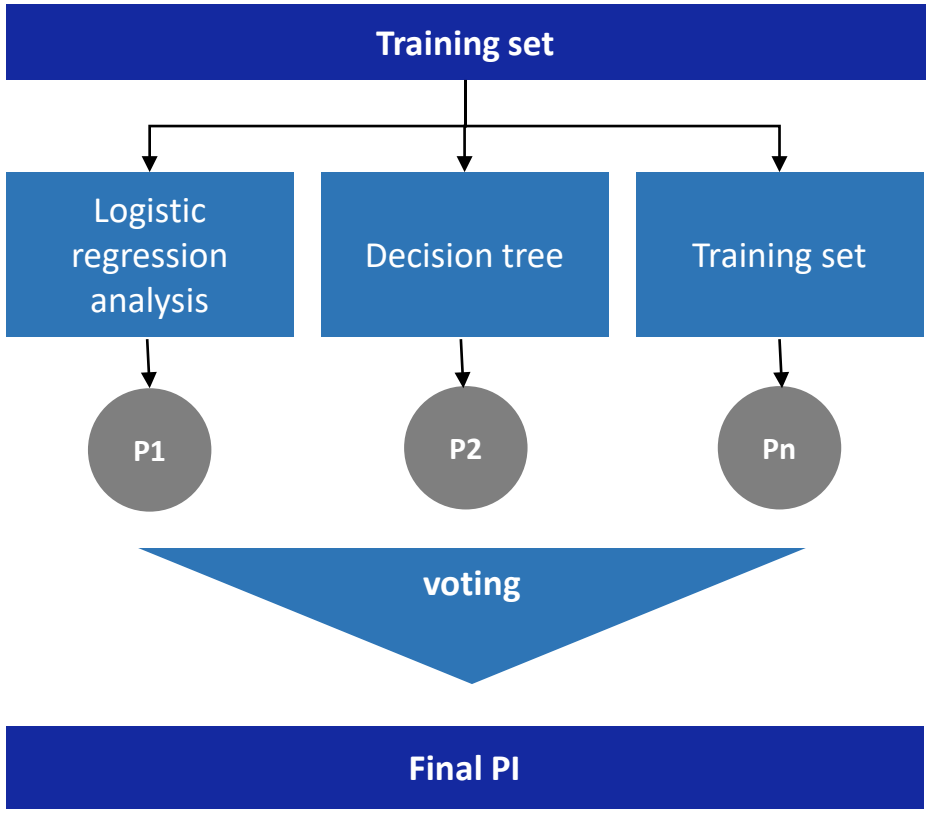
- Strong predictive model based on the weaker learners.
- There are many different ensemble methods to use collective intelligence:
- **Voting type**:
  - A collection of basic learners (different kind) that "vote". **Ex** Combine Tree, KNN, SVM…
- **Bagging type**:
  - A collection of **independent** weak learners (same kind) that "vote". **Ex** Random Forest
- **Boosting type**:
  - A series of weak learners that **adaptatively learn and predict**. **Ex** AdaBoost, GBM, XGBoost....







Voting



- Use a single training set.
- Use different classification models.
- 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 Vote**
  - Taking classification as an example, if the predictive values for classification are 1,0,0,1,1, since 1 has three votes and 0 has 2 votes, 1 becomes the final predictive value in hard voting method.
- **Soft Vote**
  - Soft vote method calculates the average value of each probability and then determines the one with the highest probability.
  - If 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**, and the final probability of getting class 1 is (0.6+0.1+0.1+0.6+0.6) / 5 = **0.4**. So, the selected final value is different from the result of the hard vote above.
- In general, using the **soft vote method is considered more reasonable** than the hard vote method in competitions, because the soft vote 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)
myLL = LogisticRegression()
myVotingEnsemble=VotingClassifier(estimators=[('lr',myLL),('knn',myKNN)],voting='hard')
```

- We can train and predict just like any other estimator:

Ex

```
myVotingEnsemble.fit(X_train, Y_train)
myVotingEnsemble.predict(X_test)
```

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 Algorithms

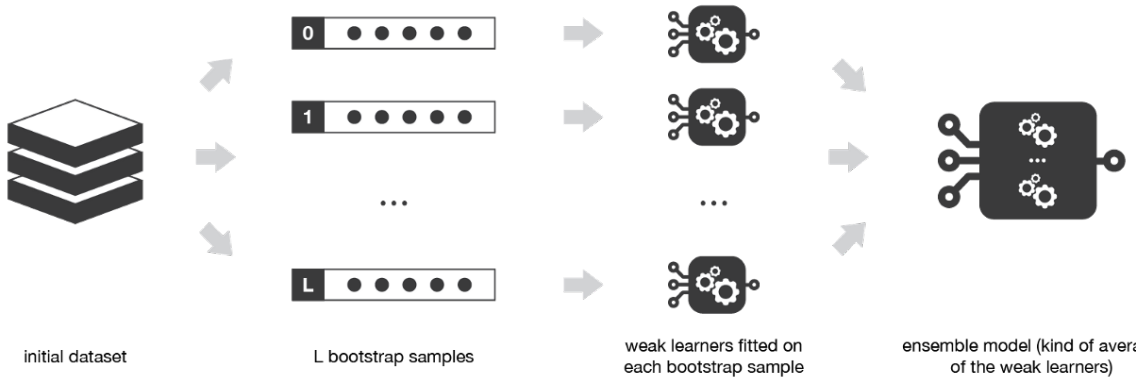
Bagging

- Bagging-based ensemble method ("**Bagging**" : **B**ootstrap **AGG**regat**ING**)
- Bagging considers homogeneous weak learners, learning them independently from each other in parallel and combining them following some kind of deterministic averaging process

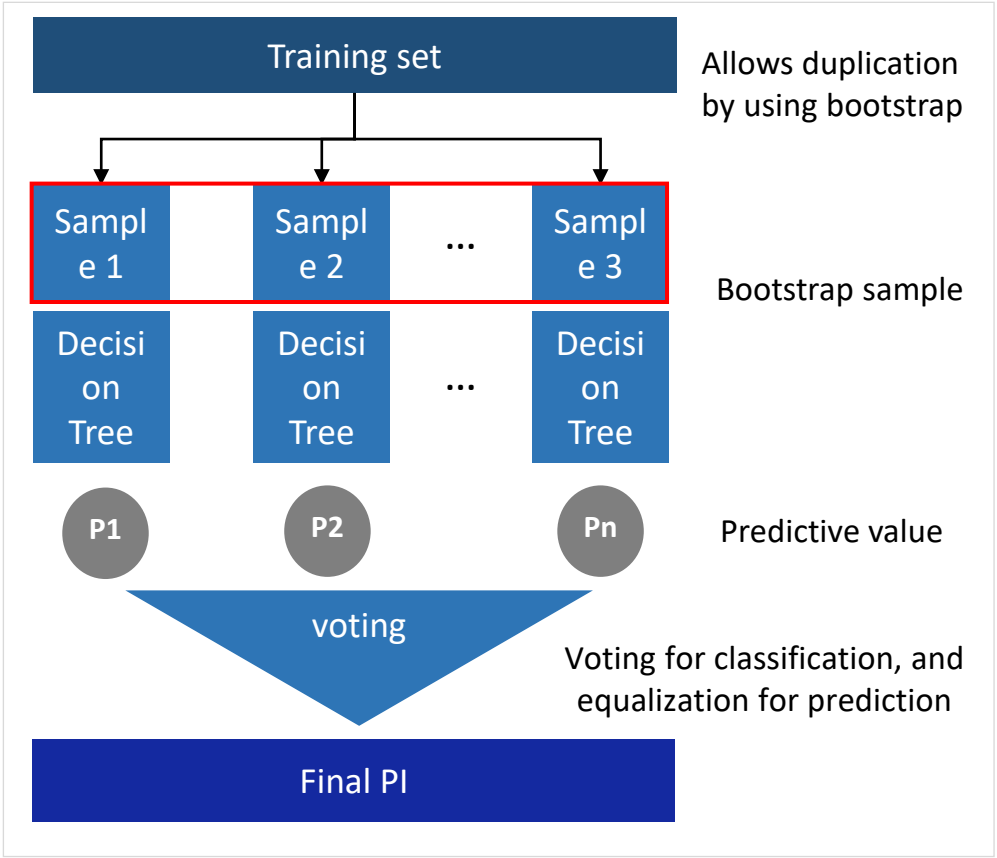
  Ex Random Forest algorithm

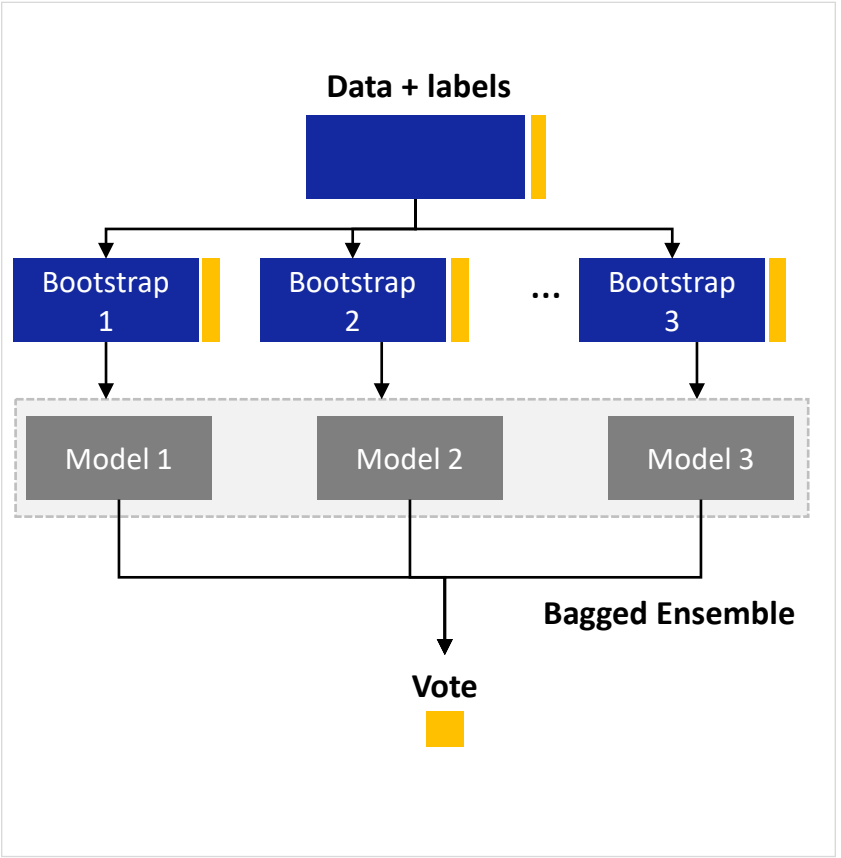
  - Easy to use since it is well constructed in the Sklearn library. Relatively quick performance speed. High performance

- The idea of bagging is simple: we want **to fit several independent models and "average" their predictions** in order to obtain a model with a lower variance. We rely on the good "approximate properties" of bootstrap samples (**representativity** and **independence**) to fit models that are almost independent.
- Basically, ensemble method raises the performance level and because it is easy to use, this method has been widely used. The bagging-based ensemble method is commonly found in the high ranked solutions in Kaggle.



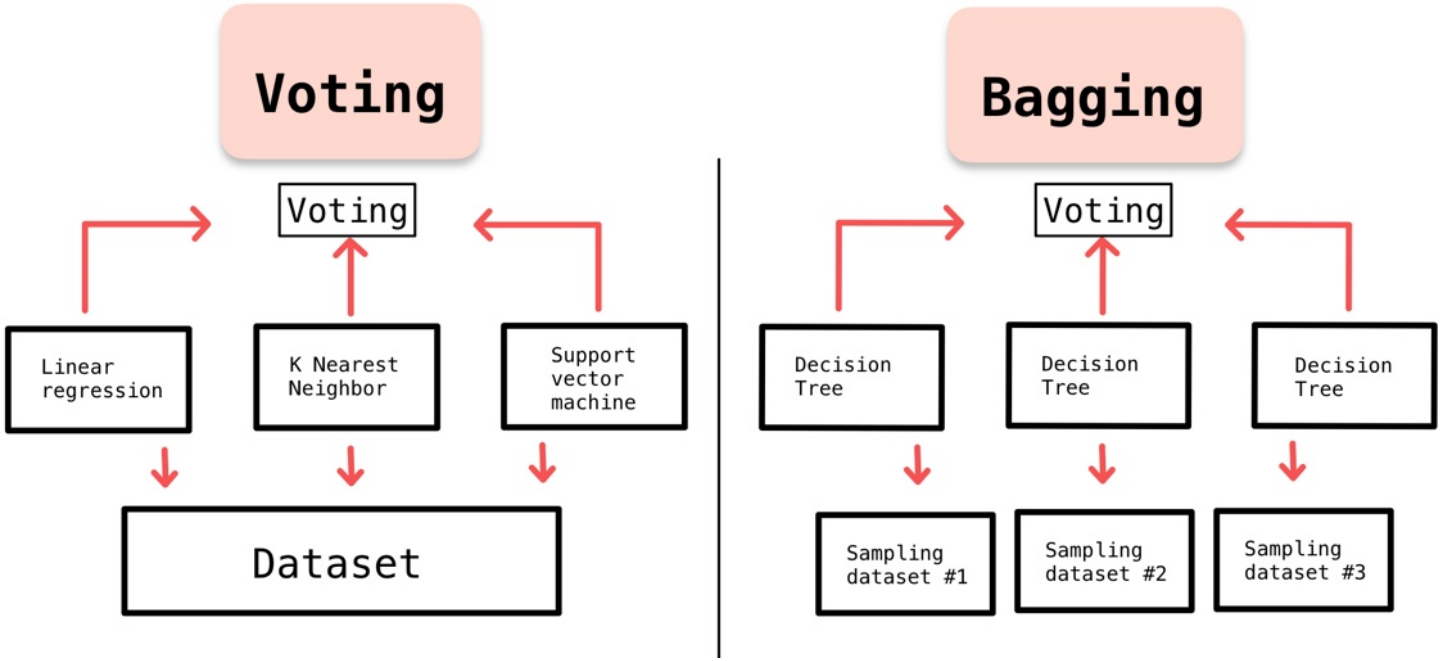
Bagging

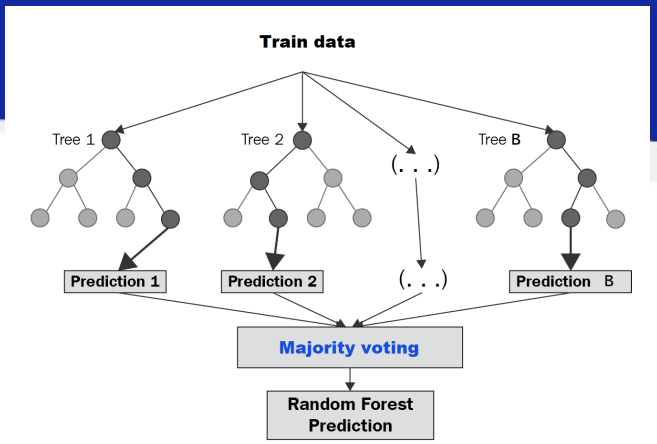

Training set
Allows duplication by using bootstrap
Sampl e 1
Sampl e 2
...
Sampl e 3
Bootstrap sample
Decisi on Tree
Decisi on Tree
...
Decisi on Tree
P1
P2
Pn
Predictive value
voting
Voting for classification, and equalization for prediction
Final PI


Data + labels
Bootstrap 1
Bootstrap 2
...
Bootstrap 3
Model 1
Model 2
Model 3
Bagged Ensemble
Vote

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 bagging 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.





Bagging Ensemble: Random Forest

About Random Forest

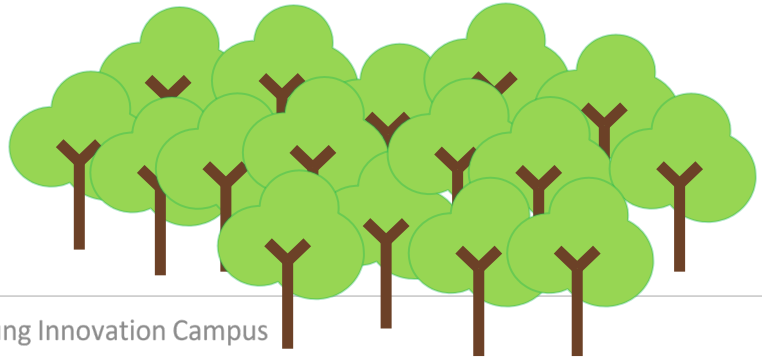
- An ensemble algorithm based on **uncorrelated** models (trees) operating as a committee.
- Can be applied to classification and regression.

Pros

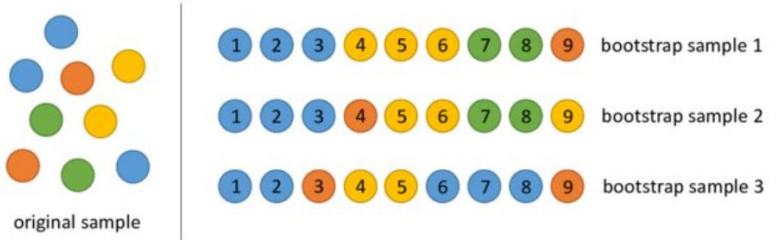
- Powerful.
- Few assumptions.
- Little or no concern about the overfitting problem.

Cons

- Training is time consuming.



Bagging (Bootstrap Aggregation) — DTs are very sensitive to the data they are trained on (small changes to the training set → different tree structures). Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees.



Feature Randomness — In a normal DTs, when spliting a node, we consider every feature and pick the one that produces the purest separation. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and lower correlation across trees

Random Forest algorithm:

1) Make trees with the randomly selected variables and observations.

2) Keep only those that have the lowest Gini impurity (or entropy).

3) Repeat from the step 1) for a given number of times.

4) Using the trees gathered during the training step, we can make predictions by majority vote.

What do we need in order for our random forest to make accurate class predictions?

- We need features that have at least some predictive power. After all, if we put garbage in then we will get garbage out.
- The trees of the forest and more importantly their predictions **need to be uncorrelated** (or at least have low correlations with each other). While the algorithm itself via feature randomness tries to engineer these low correlations for us, the features we select and the hyper-parameters we choose will impact the ultimate correlations as well.

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

Unit 8.

Ensemble Algorithms

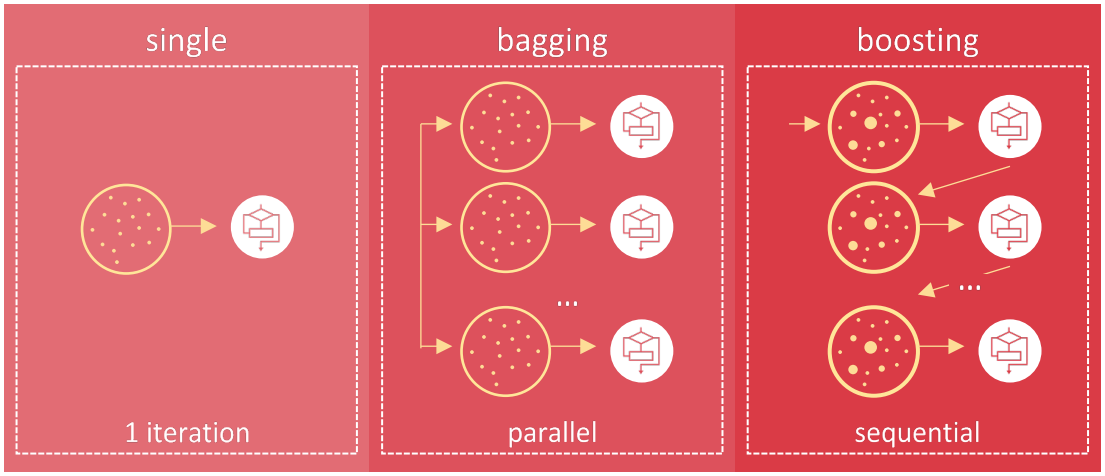
- 8.1. The concept of Ensemble Algorithm and Voting
- 8.2. Bagging & Random Forest
- 8.3. Boosting

Boosting

- The boosting algorithm is also ensemble learning. After learning weak learning machines in order, 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 that it **supplements errors by adding weight**.
- In sequential methods the different combined **weak models are no longer fitted independently** from each others. The idea is to fit models iteratively such that the training of model at a given step depends on the models fitted at the previous steps.

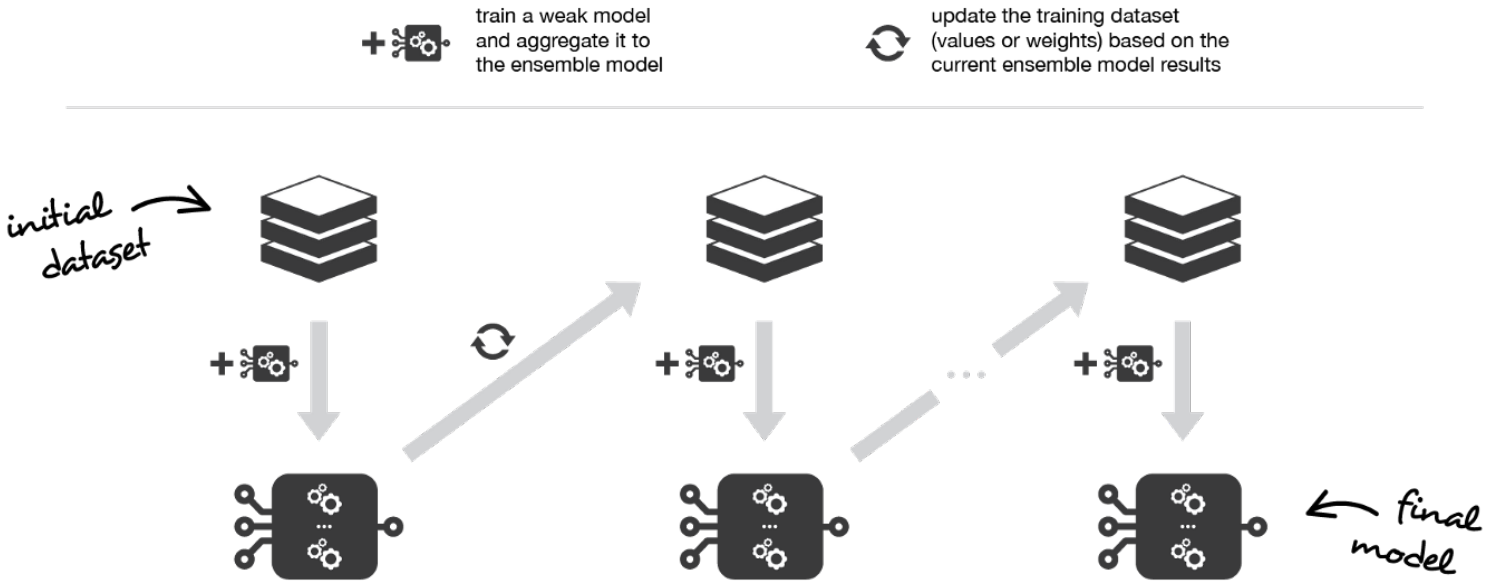
  - "Boosting" ensemble models are in general less biased than the weak learners that compose it (each new model focus its efforts on the most difficult observations to fit up to now).
  - Disadvantages include parallel processing is difficult since its sequential property, and because of it takes longer learning time compared to other ensembles.

Single estimator, Bagging, Boosting



Boosting

- Once the weak learners have been chosen, we still need to define how they will be sequentially fitted.
  - What **information** from previous models do we take into account when fitting the current model?
- And how they will be aggregated.
  - How do we **aggregate** the current model to the previous ones?



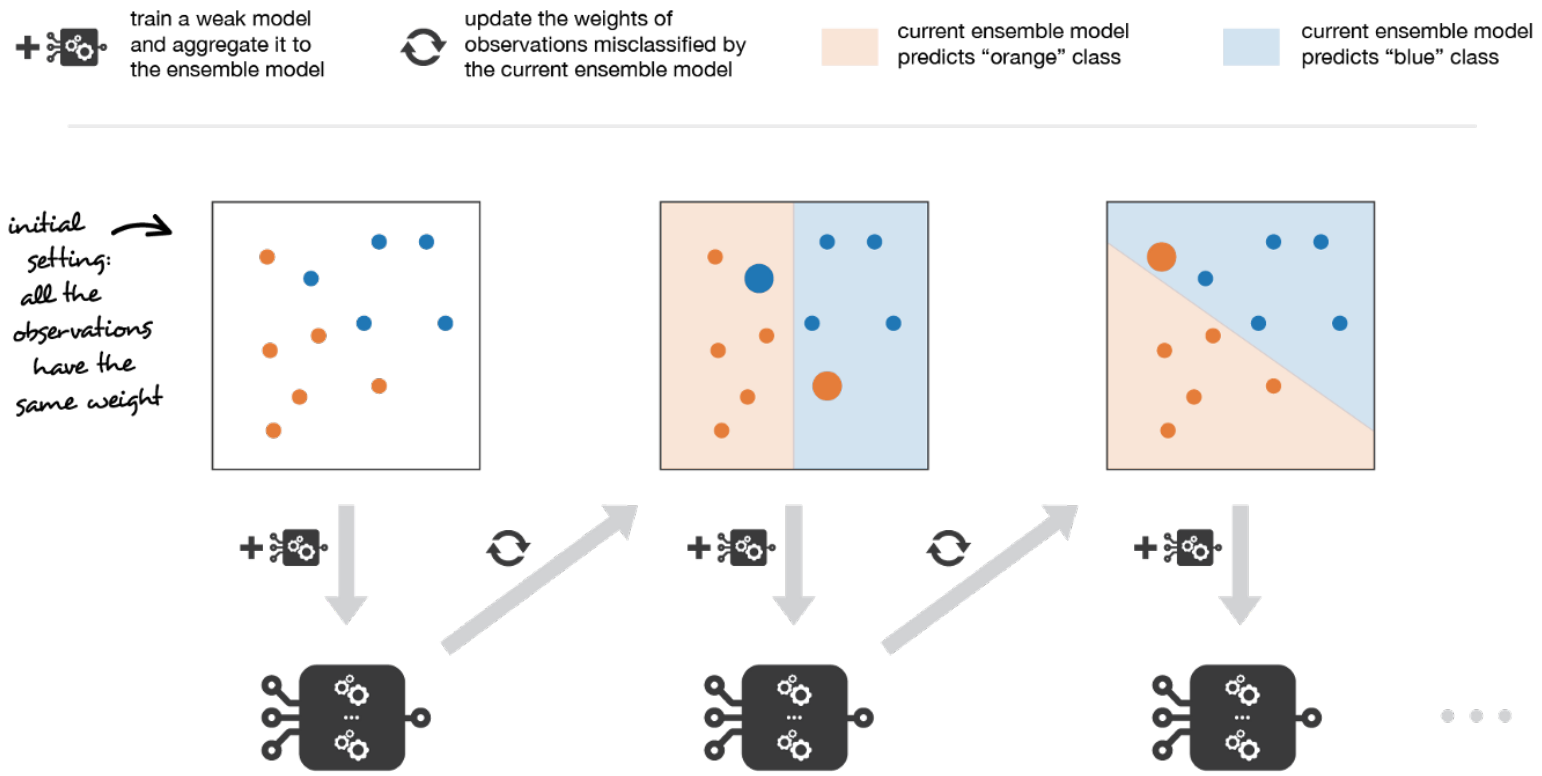
Learning a series of predictors by supplementing previous models

- Two important boosting algorithms: **AdaBoost** and **Gradient Boosting**
  - They differ on how they create and aggregate the weak learners during the sequential process.
- **AdaBoost**
  - Train the first distributor (e.g. decision tree) in the training set and make prediction.
  - **The weight of the train sample with inaccurately classified algorithm is relatively increased**.
  - The second distributor uses updated weight to be trained at the training set and makes prediction again.
  - The weight is updated again, and the same process is repeated.
- **Gradient Boosting**
  - Like AdaBoost, gradient boosting sequentially adds predictors to 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 can be briefly explained with making sequential learning of weak learning machines and supplementing errors while applying weight to inaccurately predicted data.
- **Procedure**:
  - Assuming we are facing a binary classification problem, with **N observations** in our dataset and we want to use **AdaBoost algorithm** with a given family of weak models.
  - At the very beginning of the algorithm (first model of the sequence**), all the observations have the same weights 1/N**. Then, we repeat **L times (for the L learners in the sequence)** the following steps:
    1. Fit the best possible weak model with the current observations weights
    2. Compute the evaluation metric of the weak learner that indicates how much this weak learner should be taken into account into the ensemble model
    3. Update the strong learner by adding the new weak learner multiplied by its update coefficient
    4. Compute new observations weights that express which observations we would like to focus on at the next iteration (weights of observations wrongly predicted by the aggregated model increase and weights of the correctly predicted observations decrease)
  - Repeating these steps, we have then built sequentially our L models and aggregate them into a simple linear combination weighted by coefficients expressing the performance of each learner.

AdaBoost (Adaptive Boosting)



AdaBoost updates weights of the observations at each iteration. Weights of well classified observations decrease relatively to weights of misclassified observations. Models that perform better have higher weights in the final ensemble model.

AdaBoost classification algorithm

- Let's suppose n observations for the training step: $\boldsymbol{x}_i$ and y_i.
- We also suppose that $y_i \in \{-1,+1\}$. (binary y)
- We will make a series of weak learners $G_m(\boldsymbol{x})$ with $m = 1,\ldots,M$.
- The ensemble classifier is made up by a linear combination of these weak learners:

$$G_{ensemble}(\boldsymbol{x}) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$$

where α_m are the "boosting weights" that need to be calculated. $sign()$ function returns +1 or −1

- Heavier weight is given to a better performing learner.

$$\alpha_m = \frac{1}{2} log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

⇨ As $\varepsilon_m \to 0$, α_m is a large positive number.

The learner is given more importance!

⇨ As $\varepsilon_m \cong 0.5$, $\alpha_m \cong 0$.

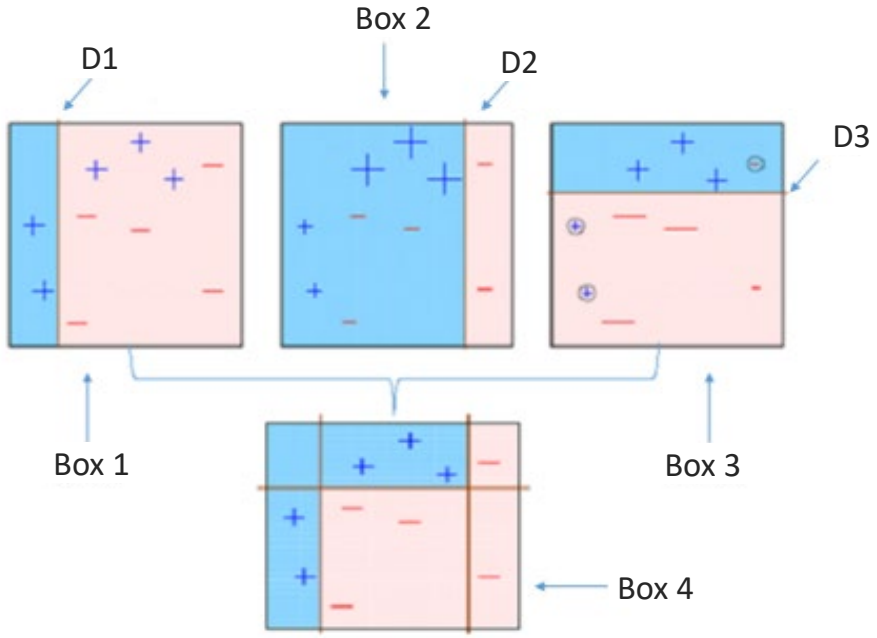
⇨ As $\varepsilon_m \to 1$, α_m is a large negative number.

$$\varepsilon_m = \frac{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G_m(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i^{(m)}}$$

⇨ $I(y_i \neq G_m(\boldsymbol{x}_i))$ gives 1 for an incorrect prediction, else 0.

⇨ $0 \leq \varepsilon_m \leq 1$

Principle of the AdaBoost



- Box 1 is the result of weak learning machine classified as D1 sector. However, because there are data sets expressed in + distributed in the red sector, the error rate is quite high.
- In Box 2, the D2 line is moved to the right to supplement 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.
- In Box 3, the D3 line is horizontally drown 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. Compared to the previous three individual learning machines, it shows much better performance.

- 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

0.2 * Box 1 + 0.5 + Box 2 + 0.6 * Box 3 = Box 4

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

Gradient Boosting

- In gradient boosting, the ensemble model we try to build is also a weighted sum of weak learners.
- The main difference with adaptative boosting is in the definition of the sequential optimisation process. Indeed, gradient boosting casts the problem into a gradient descent one: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model.
- Procedure:
  - A first weak learner **f1** is fitted to predict the response variable **y**, and the residuals **y - f1(x)** are calculated.
  - Then, a new model **f2** is fitted, which tries to predict the residuals of the previous model, in other words, it tries to correct the errors made by model **f1**

$$f1(x) \approx y$$

$$f2(x) \approx y - f1(x)$$

GB strongly relies on the prediction that the next model will reduce prediction errors when blended with previous ones.

  - In the next iteration, we calculate the residuals of the two models **y - f1(x) - f2(x)**, as well as the errors made by **f1** and that **f2** has not been able to correct, and a third model **f3** is fitted to try to correct for them

$$f3(x) \approx y - f1(x) - f2(x)$$

  - This process is repeated **M** times, so that each new model minimises the residuals (errors) of the previous one.
- Since the goal of Gradient Boosting is to minimise the residuals iteration by iteration, it is susceptible to overfitting. One way to avoid this problem is to use a regularisation value, also known as learning rate (λ), which limits the influence of each model on the ensemble.

Gradient Boosting Machine (GBM)

- The boosting algorithm in gradient descent is called gradient boosting machine, which is abbreviated as GBM. It is provided in the sklearn package and is applicable to both classification and regression problems.
  - **GradientBoostingClassifier**
  - **GradientBoostingRegressor**

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.
- It has been recently used a lot due to its high performance and computer resource application rate, and it has become more popular as it was frequently used by **top rankers of Kaggle.**

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.

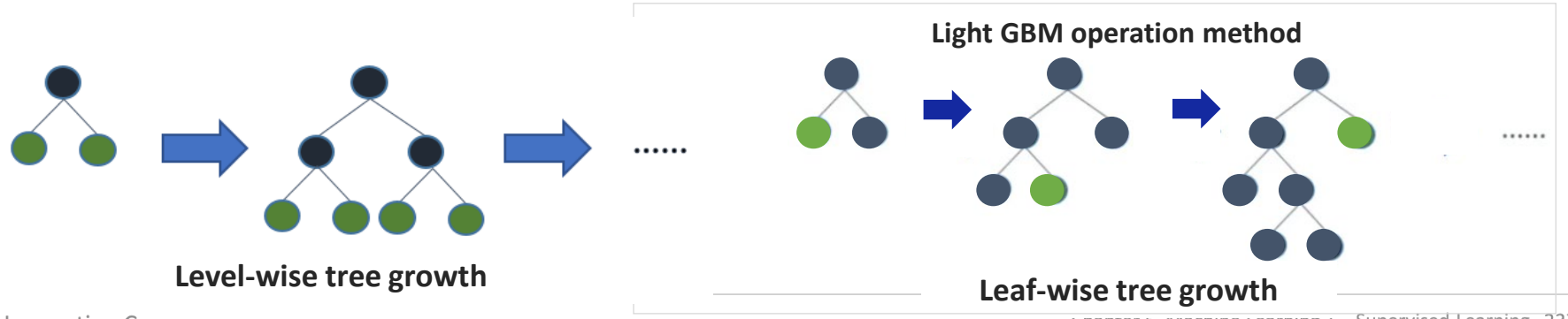
- Need to be tuned for optimized performance.
- More information can be found at: https://xgboost.readthedocs.io/en/latest/python/index.htm

Hyperparameter tuning methods

- The increased **Num_leaves** value raises accuracy, but it 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 high value of tree depth when setting a higher value.
- **Max_depth** constraints the size of depth. Improves overfitting when combined with two parameters above.

Light GBM

- Together with the XGBoost, **LightGBM** is the most highlighted boosting algorithm. Although XGBoost has an excellent performance, it takes too long learning period.
- Advantages of the **LightGBM**
  - Short learning time
  - Relatively small memory use
  - Automatic conversion and optimal splitting of categorical features
- While the tree is vertically expanded with LightGBM (leaf-wise), other algorithms expand the tree horizontally (level-wise) characterized by splitting while maintaining a balanced tree as much as possible, thus the tree depth would be minimum). A disadvantage of level wise method is that it takes some time to make a balanced tree.
- However, the leaf wise method of LightGBM does not balance the tree, but it continuously splits the leaf node that has the max. data loss. This way, the tree depth becomes greater, and an asymmetrical tree is created. The repetition of leaf node with the max. data loss minimizes predicted error loss than the splitting of a balanced tree



Coding Exercise #0313

Follow practice steps on 'ex_0313.ipynb' file

SAMSUNG