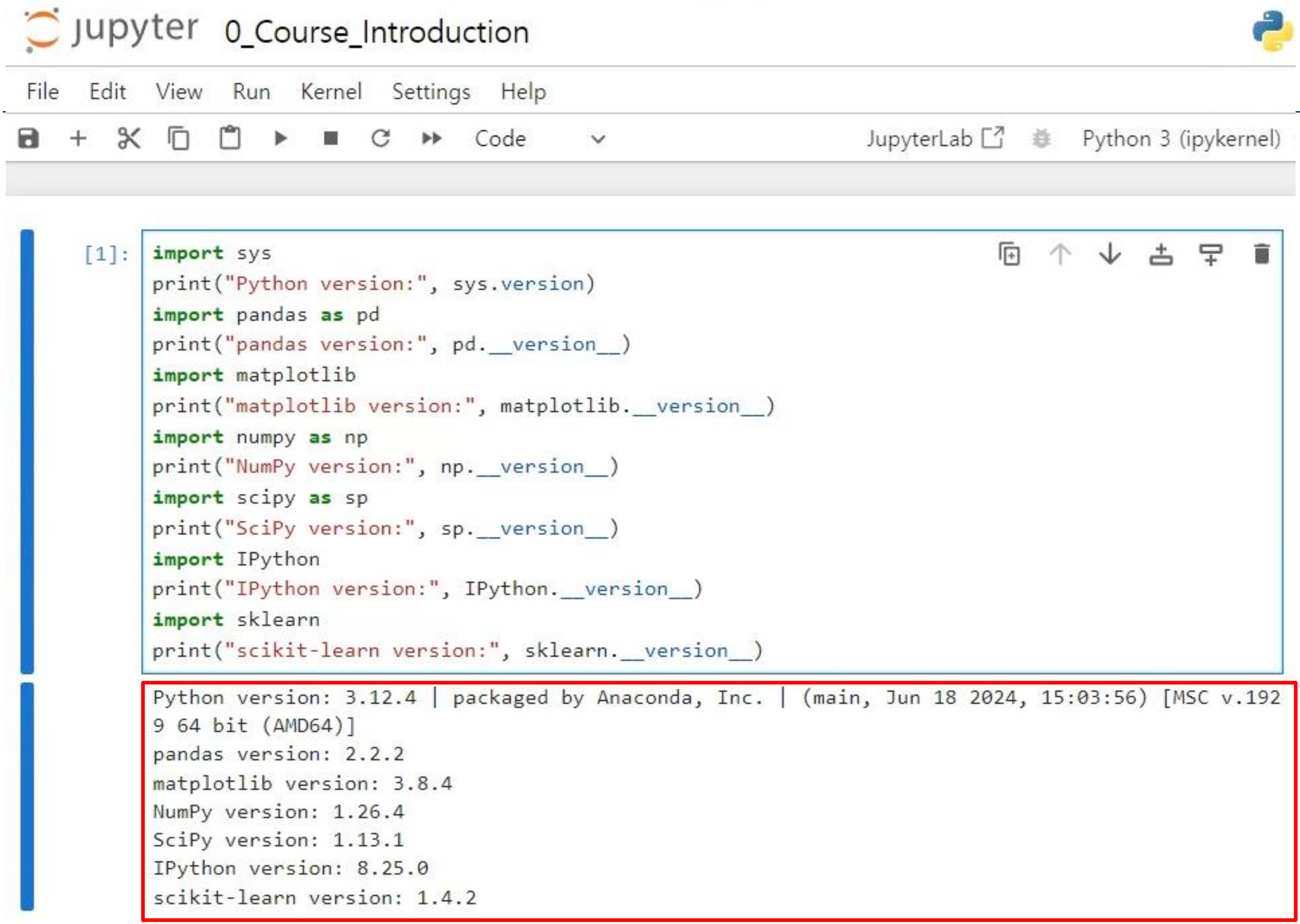
Supervised Learning – Part 1

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)


jupyter 0_Course_Introduction
File Edit View Run Kernel Settings Help
Code
JupyterLab
Python 3 (ipykernel)
[1]:

```
import sys
print("Python version:", sys.version)
import pandas as pd
print("pandas version:", pd.__version__)
import matplotlib
print("matplotlib version:", matplotlib.__version__)
import numpy as np
print("NumPy version:", np.__version__)
import scipy as sp
print("SciPy version:", sp.__version__)
import IPython
print("IPython version:", IPython.__version__)
import sklearn
print("scikit-learn version:", sklearn.__version__)
```

```
Python version: 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.192
9 64 bit (AMD64)]
pandas version: 2.2.2
matplotlib version: 3.8.4
NumPy version: 1.26.4
SciPy version: 1.13.1
IPython version: 8.25.0
scikit-learn version: 1.4.2
```

Supervised Learning

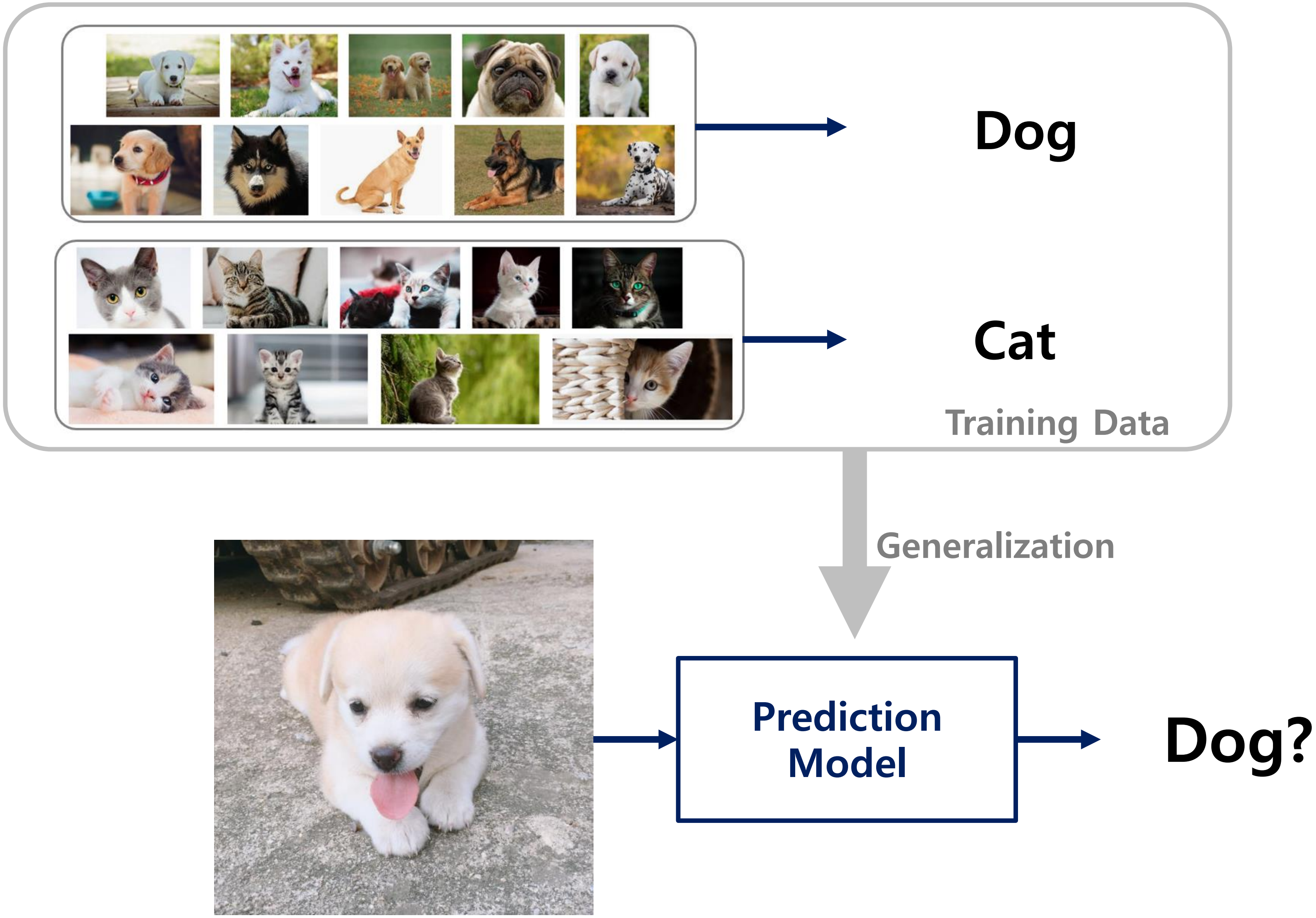
Supervised Learning

Dog

Cat

?

Supervised Learning



Dog

Cat

Training Data

Generalization

Prediction Model

Dog?

Supervised Learning

- **Supervised Learning**
  - *(in general)* **Labeled training dataset** $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, where each data point $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id})$ contains *d* feature values and is associated with a label y_i
  - To find the functional relationship between $\boldsymbol{x}$ and y, in the form of $\hat{y} = f(\boldsymbol{x})$, from the dataset
  - To predict the unknown label of a new data point

Supervised Learning

- ***Labeled Dataset***

Column: variable, attribute, **feature/target**, predictor/response, …

Input

Label

Row:
data point,
instance,
example,
record,
pattern,
object,
…

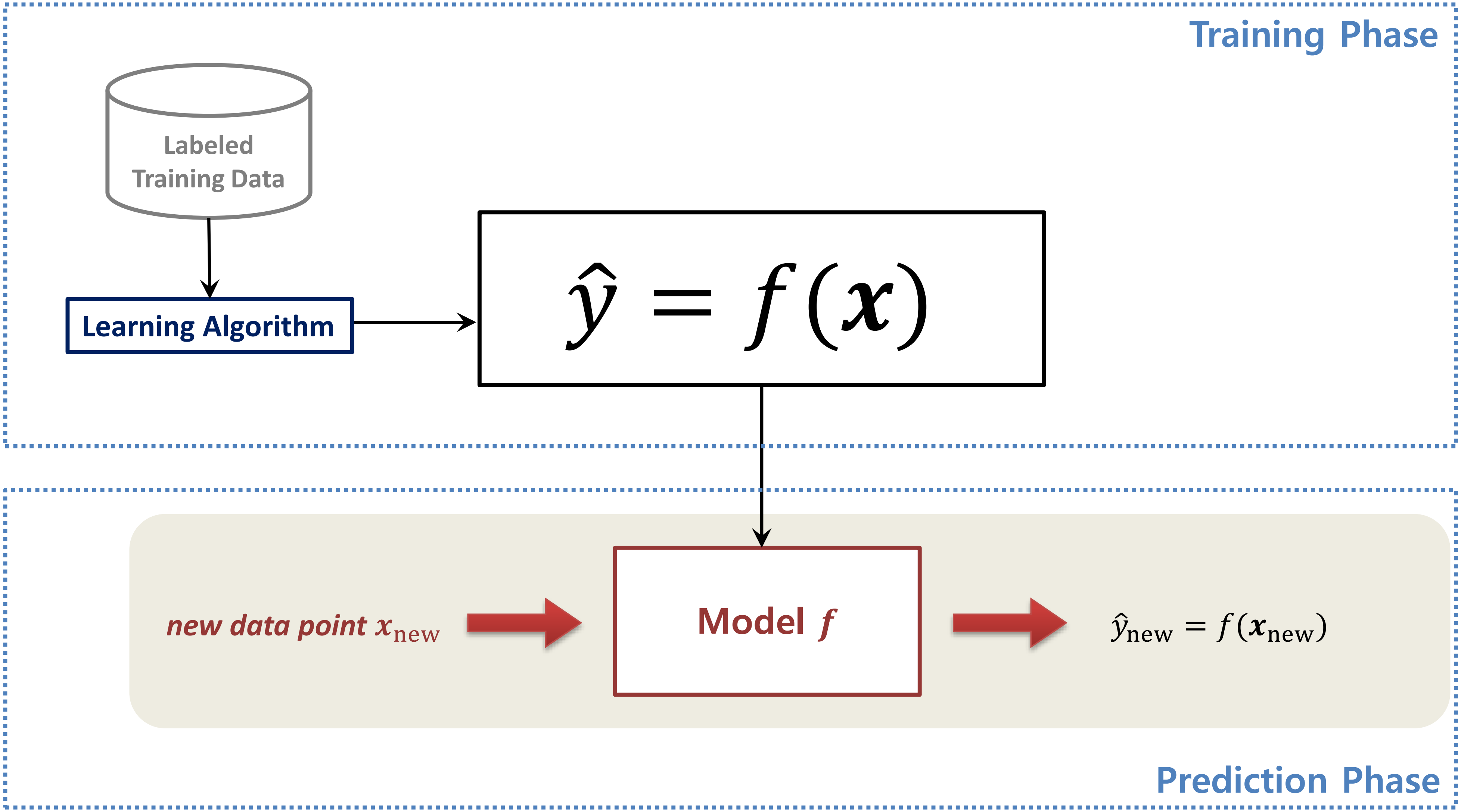
id	X_1	X_2	X_3	…	X_d
1	x_{11}	x_{12}	x_{13}	…	x_{1d}
2	x_{21}	x_{22}	x_{23}	…	x_{2d}
3	x_{31}	x_{32}	x_{33}	…	x_{3d}
4	x_{41}	x_{42}	x_{43}	…	x_{4d}
5	x_{51}	x_{52}	x_{53}	…	x_{5d}
6	x_{61}	x_{62}	x_{63}	…	x_{6d}
7	x_{71}	x_{72}	x_{73}	…	x_{7d}
…	…	…	…	…	…

Y
y_1
y_2
y_3
y_4
y_5
y_6
y_7
…

Supervised Learning

- Automation of decision-making processes by generalizing from known examples ***(training data)***
- The user provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired output given an input.
- The algorithm is able to create an output for an input it has never seen before without any help from a human. **(*generalization*)**
- Supervised learning algorithms are well understood and their performance is easy to measure.

Supervised Learning



Training Phase

Labeled Training Data

Learning Algorithm

$$\hat{y} = f(\boldsymbol{x})$$

new data point x_{new}

Model f

$\hat{y}_{\text{new}} = f(\boldsymbol{x}_{\text{new}})$

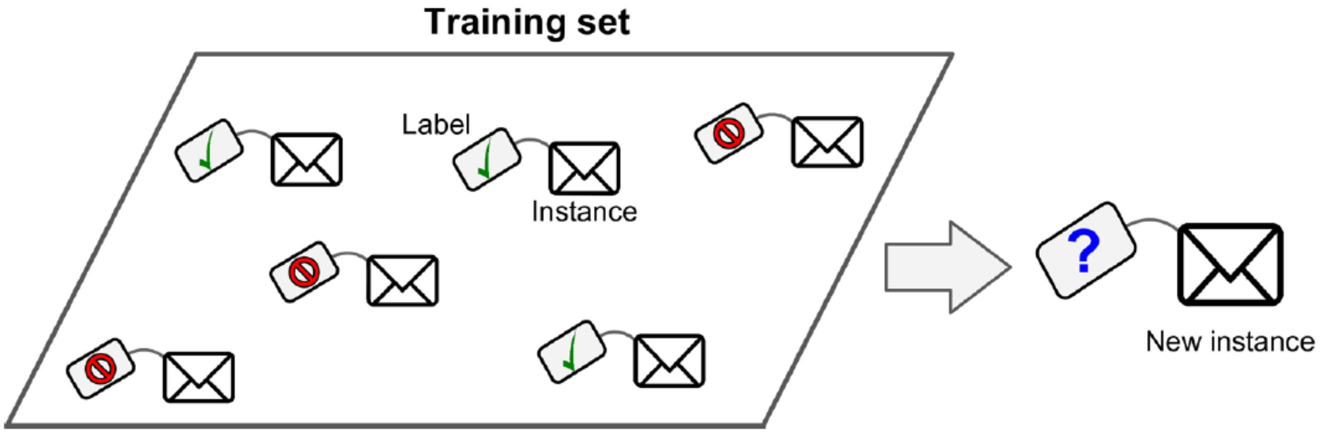
Prediction Phase

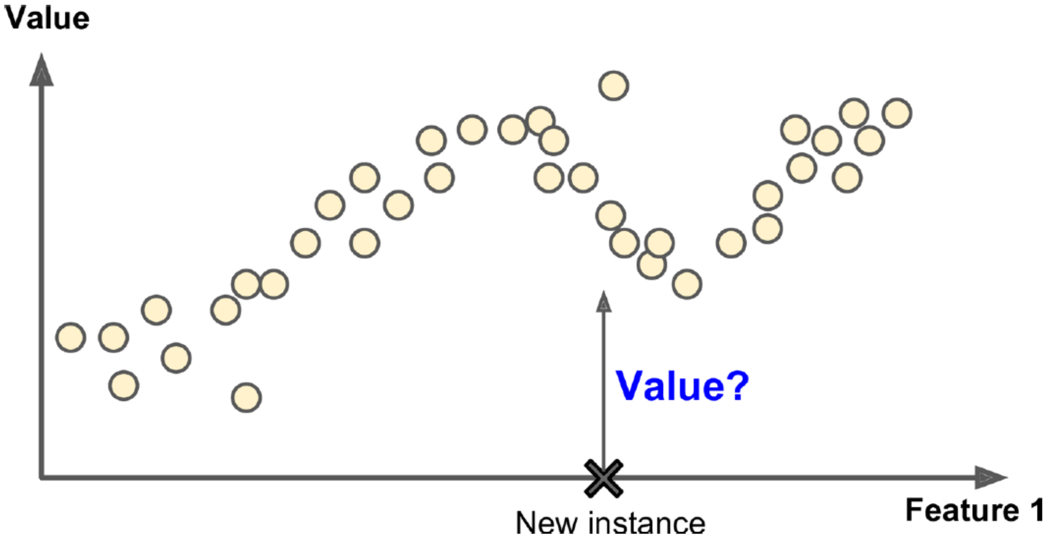
Types of Supervised Learning

- **Classification:** to predict a class label (a choice from a predefined list of possibilities)
  - Binary Classification: distinguishing between exactly two classes
  - Multi-Class Classification: classification between more than two classes
  - Multi-Label Classification

- **Regression:** to predict a continuous number (continuity in the label)

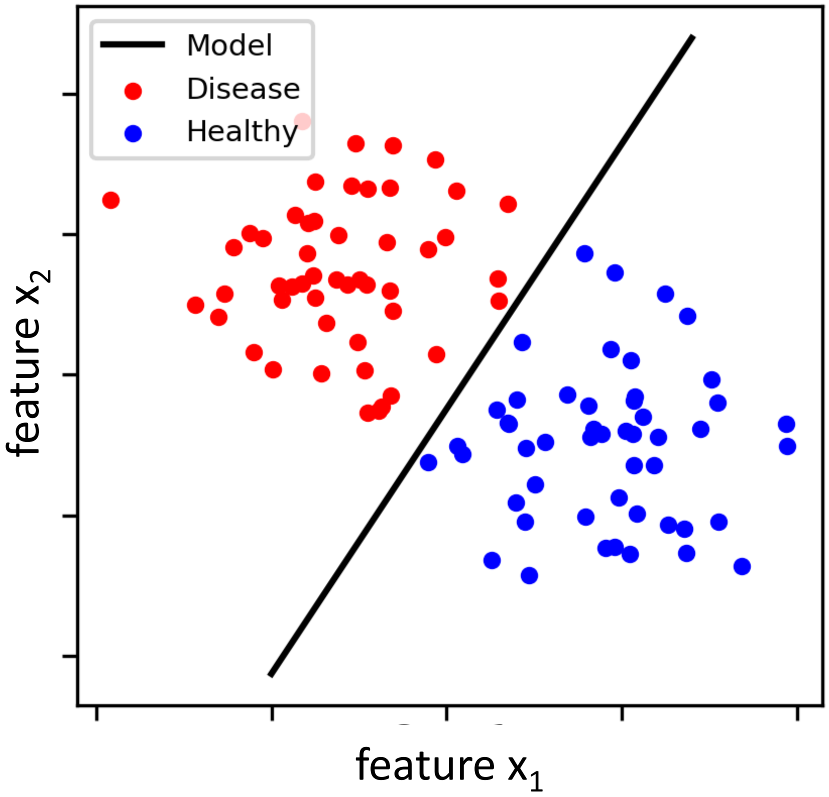
Types of Supervised Learning

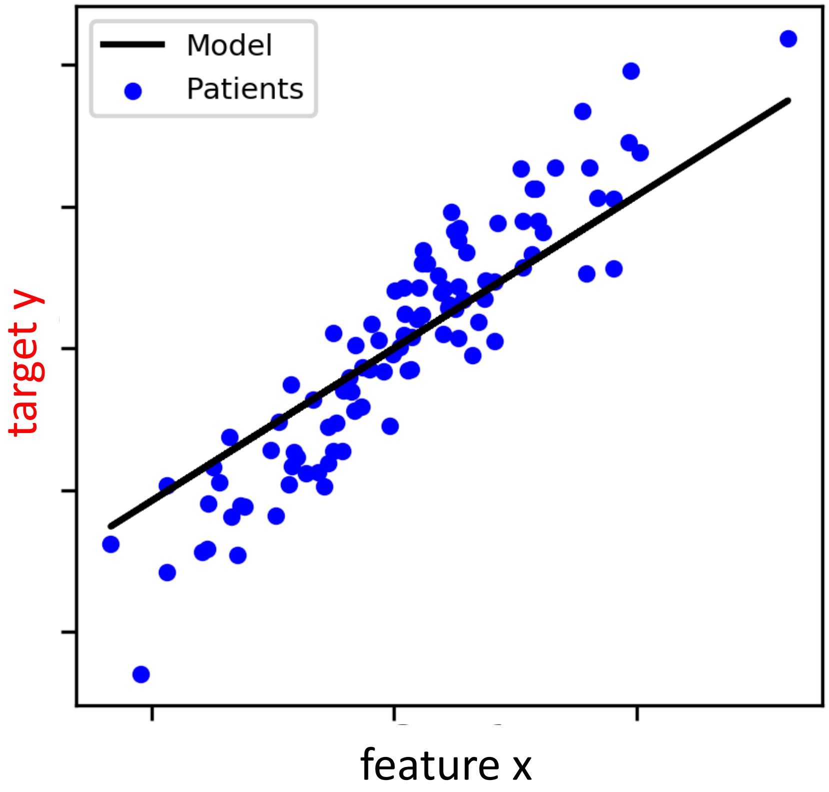




Types of Supervised Learning

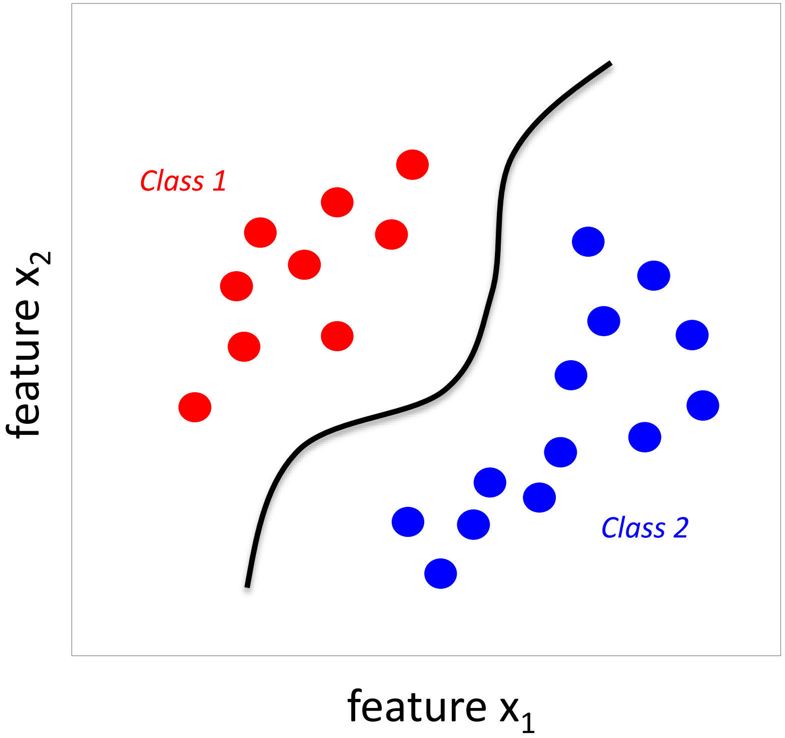
- **Classification vs Regression**

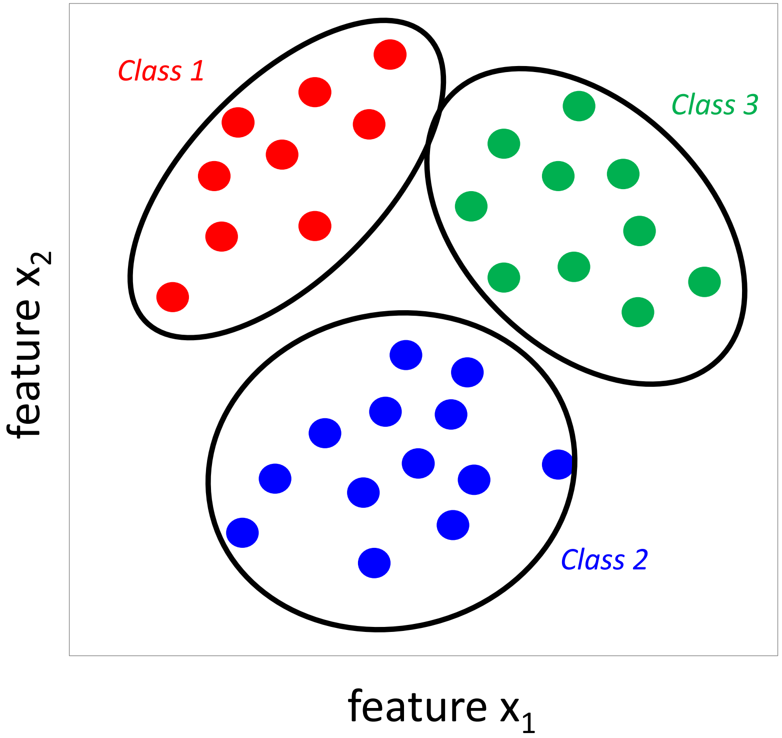




Types of Supervised Learning

- **Binary vs Multi-Class Classification**

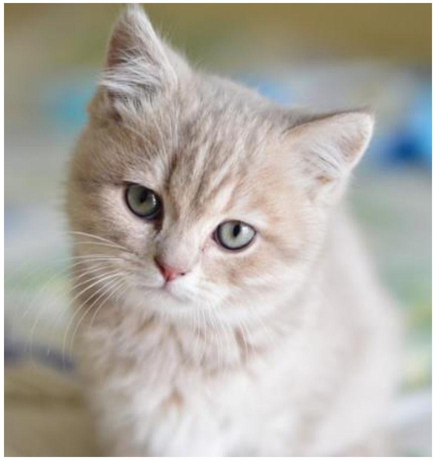




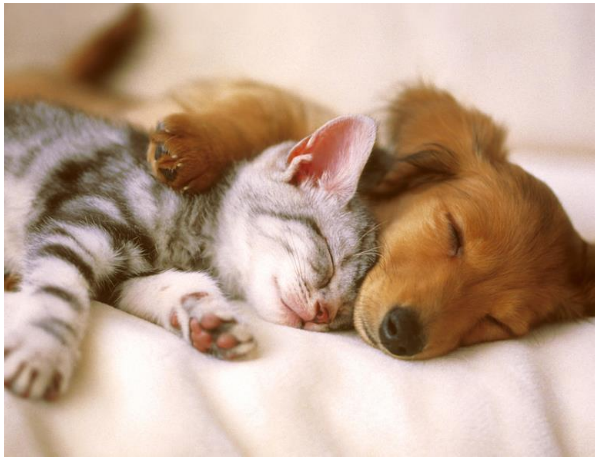
Types of Supervised Learning

- **Multi-Label Classification**

“Dog”



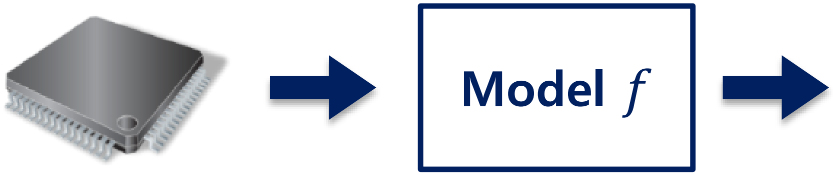
“Cat”



?

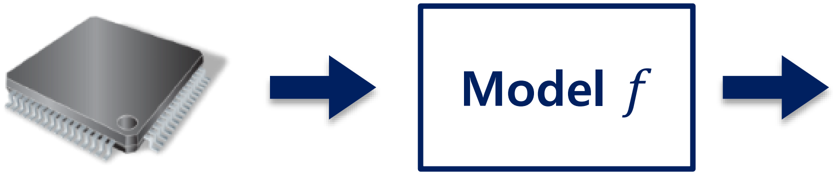
Types of Supervised Learning

- **Binary Classification**



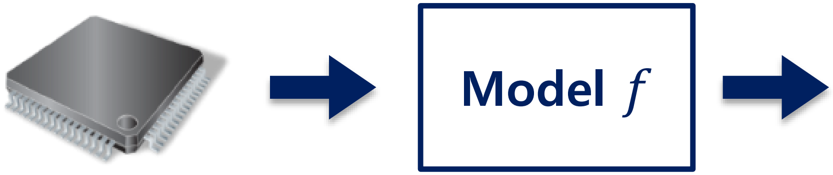
Pass or Fail

- **Multi-Class Classification**



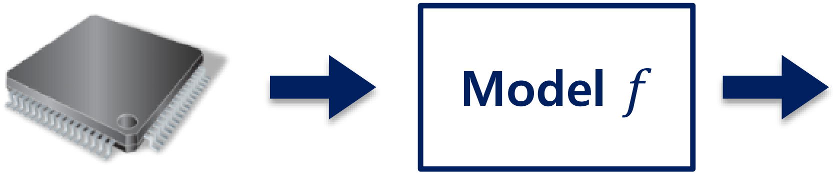
Each data point is assigned to only one label
Pass or Fail_A or Fail_B or …

- **Multi-Label Classification**



Each data point can be assigned to multiple labels
Pass or Fail_A or Fail_B or …

- **Regression**



Continuous-valued indicator
e.g.) 1.4, 3.2, -1.1, 0.4, …

Learning algorithms covered in this course

- **Supervised Learning** (Classification/Regression)
  - K-Nearest Neighbors
  - Linear Models (Logistic/Linear Regression)
  - Decision Trees
  - Random Forests
  - Support Vector Machines
  - Neural Networks

** Many algorithms have a classification and a regression variant, and we will describe both.*

** We will review the most popular machine learning algorithms, explain how they learn from data and how they make predictions, and examine the strengths and weaknesses of each algorithm.*

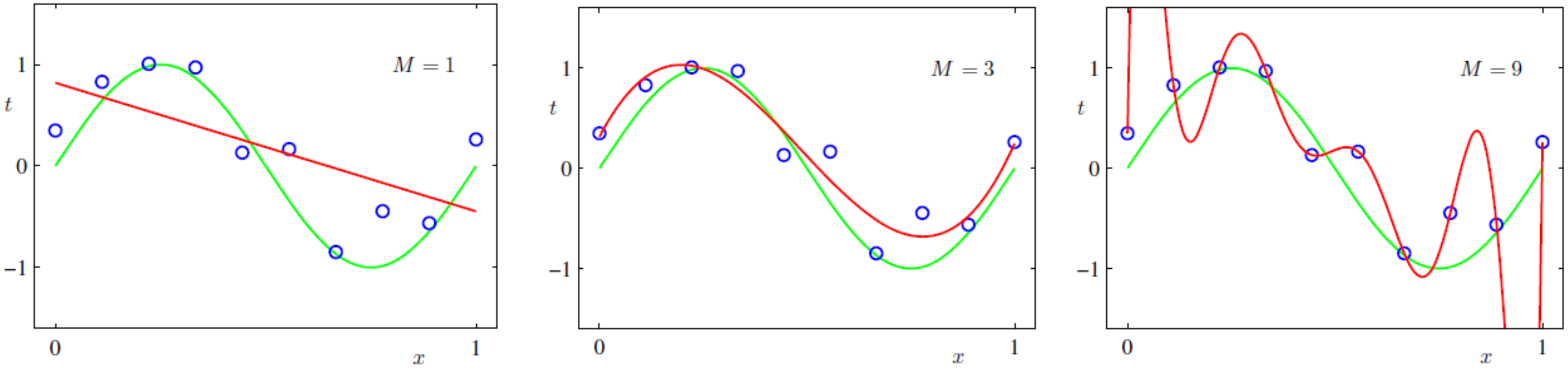
Generalization

Generalization

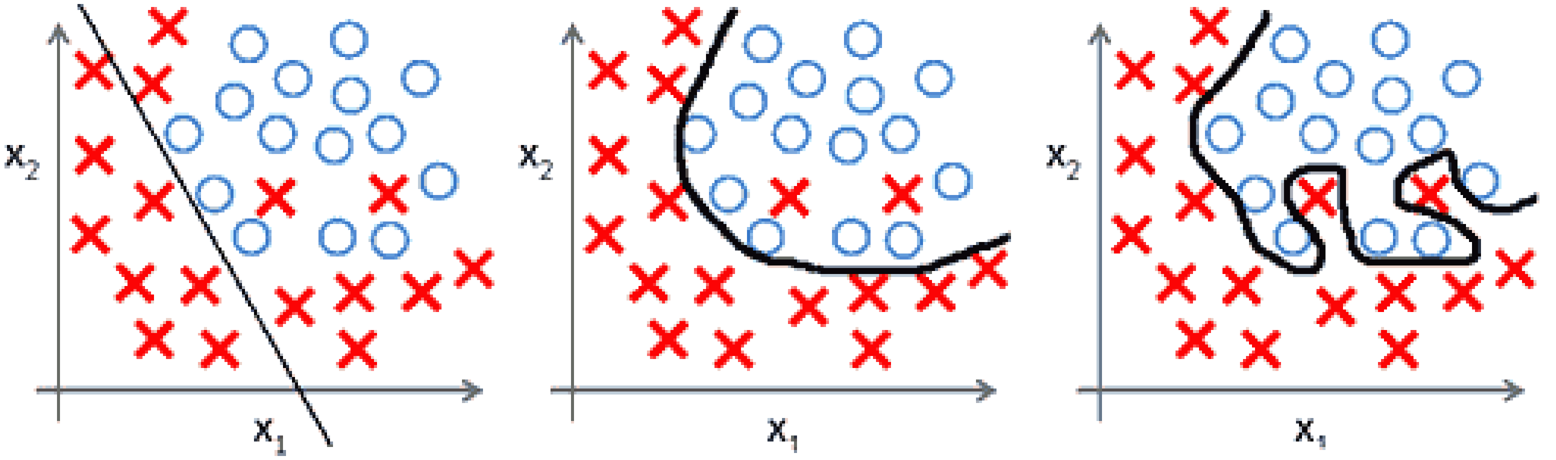
- In supervised learning, we want to build a model that will make accurate predictions on new, unseen data, which is called **generalization**.
- To evaluate generalization performance, we use a **test set** that were collected separately from the **training set**.
- If a model is able to make accurate predictions on the test set, we say it is able to **generalize** from the training set to the test set.
- We want to build a model that is able to generalize as accurately as possible.

Generalization

- **Generalization Failure**
  - In Regression…



  - In Classification…



Generalization

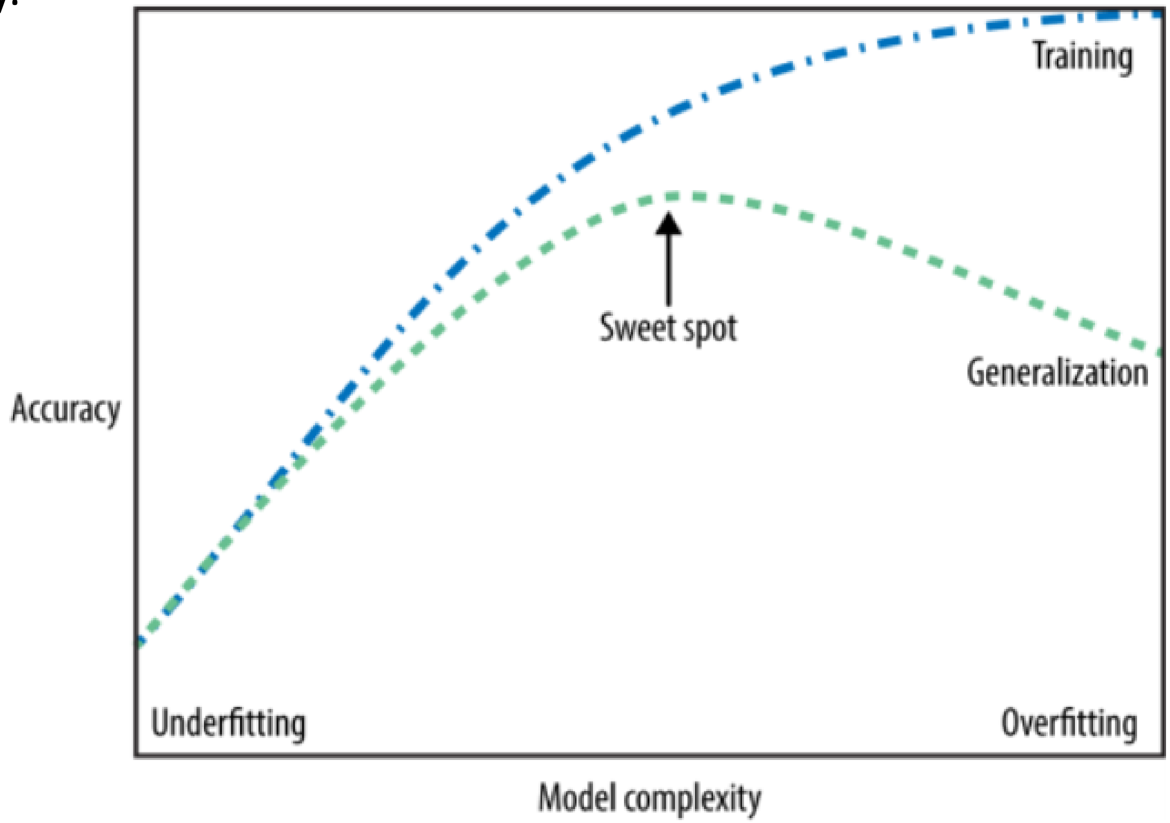
- **Generalization Failure**
  - The objective of supervised learning is to maximize the generalization performance
  - **Overfitting**
    - The model is too complex for the amount of information we have.
    - The model is fit too closely to the particularities of the training set.
    - The model will work well on the training set
      but will not be able to generalize to new data (the test set).
  - **Underfitting**
    - The model is too simple for the amount of information we have.
    - The model fails to capture all the aspects of and variability in the training set.
    - The model will do badly even on the training set.

Generalization

- **Model Complexity**
  - It’s important to note that **model complexity** is intimately tied to the **variation of inputs contained in your training dataset**.
  - The **larger variety of data points** your dataset contains, the more complex a model you can use **without overfitting**.
  - Usually, collecting more data points will yield more variety, so larger datasets allow building more complex models.
  - In the real world, you often have the ability to decide **how much data to collect**, which might be more beneficial than tweaking and tuning your model.

Generalization

- **The trade-off**
  - **Overfitting:** the gap between the training error and test error is too large.
  - **Underfitting**: the model is not able to obtain a sufficiently low error value on the training set.
  - We can control whether a model is more likely to overfit or underfit by altering its complexity.



Model Parameters/Hyperparameters

- **Parameters: Configuration internal to the model**
  - The values are **derived via model training.**
  - They are used to make predictions for new data.

- **Hyperparameters: Configuration for training of the model**
  - The values are **set before model training**.
  - They control the model complexity and learning algorithm's behavior

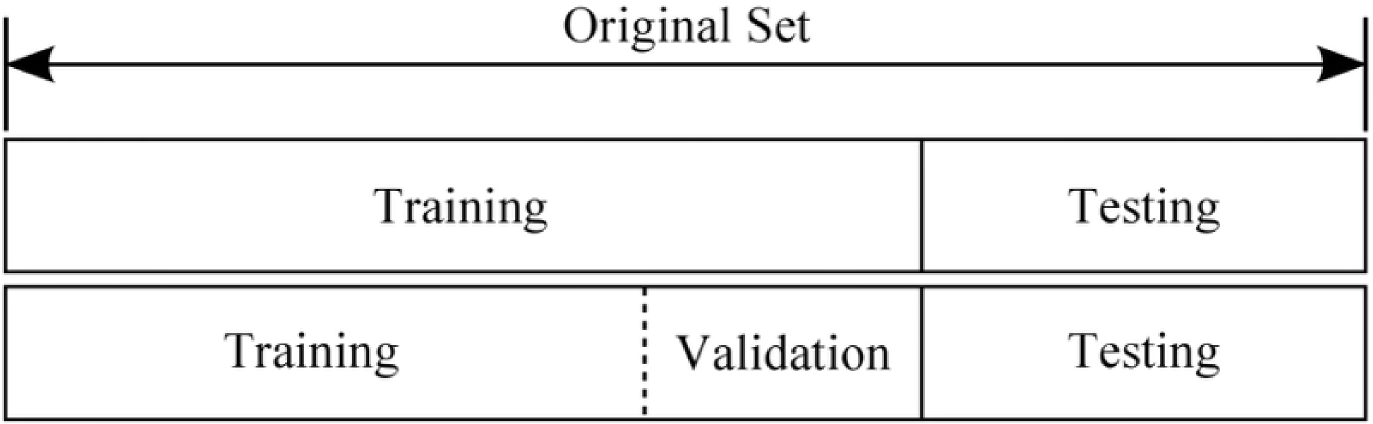
** Given the hyperparameters, the learning algorithm learns the parameters from the training data.*

** If you have to specify a model parameter manually then it is probably a model hyperparameter.*

Training, Validation, and Test

- **Training set:** to learn the **parameters** of the model
- **Validation set:** to choose the **hyperparameters** of the model
- **Test set:** for final evaluation of the **generalization ability** of the model
  (How well will our model perform with new data that were not observed during training and validation?)

▪**흔한 실수 1**: training set에 대해서 최종 성능평가
▪**흔한 실수 2**: validation set과 test set을 구분하지 않음



Original Set

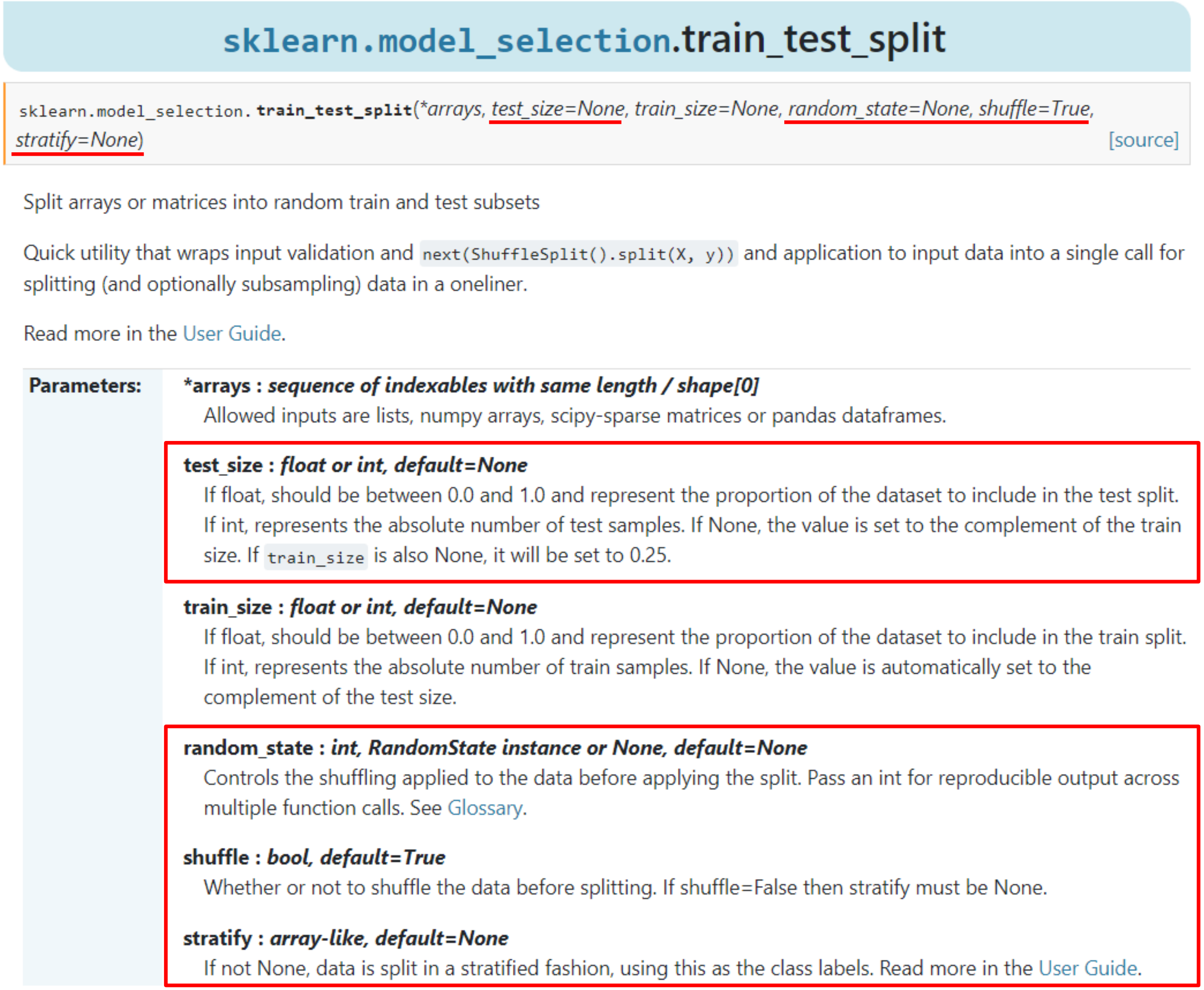
Training | Testing

Training | Validation | Testing

"must be disjoint!"

scikit-learn Practice: *train_test_split*

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html



sklearn.model_selection.train_test_split

`sklearn.model_selection.`**`train_test_split`**(**arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None*) [source]

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

Read more in the User Guide.

Parameters:

arrays : *sequence of indexables with same length / shape[0]

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : *int, RandomState instance or None, default=None*

Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : *bool, default=True*

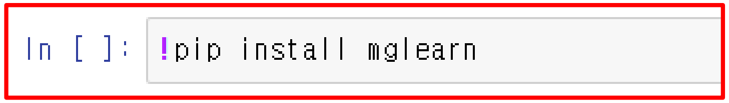
Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratify : *array-like, default=None*

If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

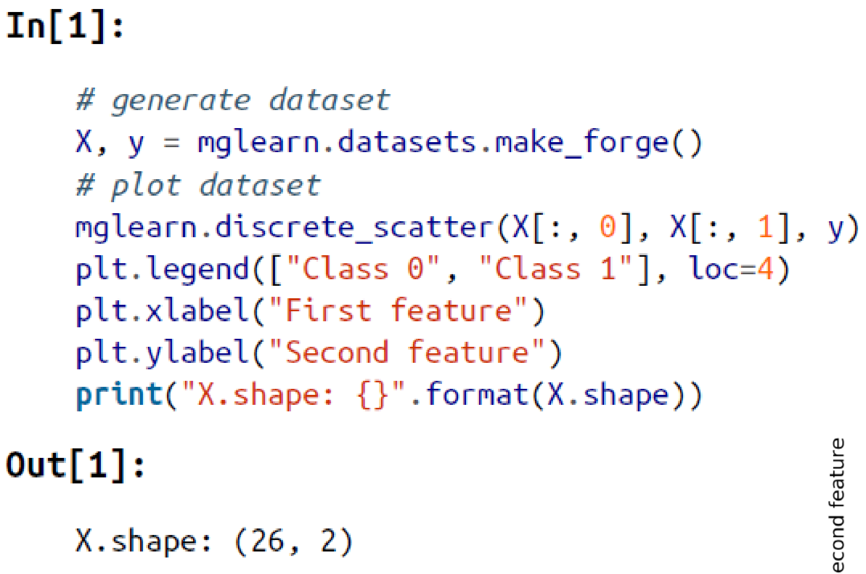
scikit-learn Practice: *train_test_split*

- **Example with the *forge* dataset**
  - The dataset consists of 26 data points with two classes (binary classification).



```
In [ ]: !pip install mglearn
```

import mglearn
import matplotlib.pyplot as plt

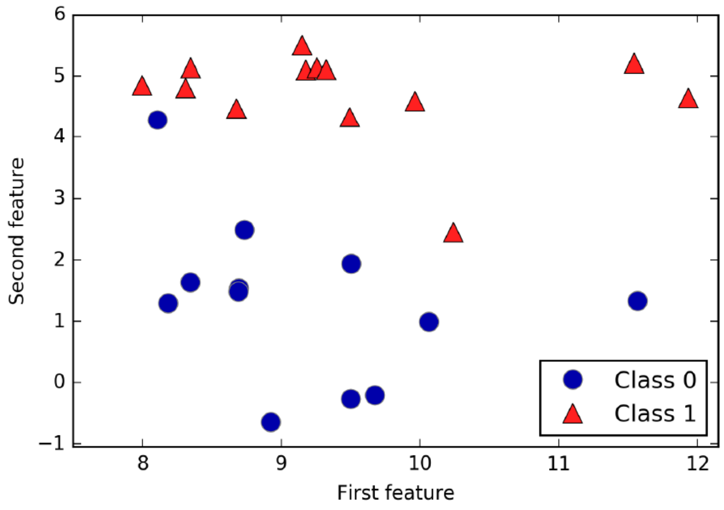


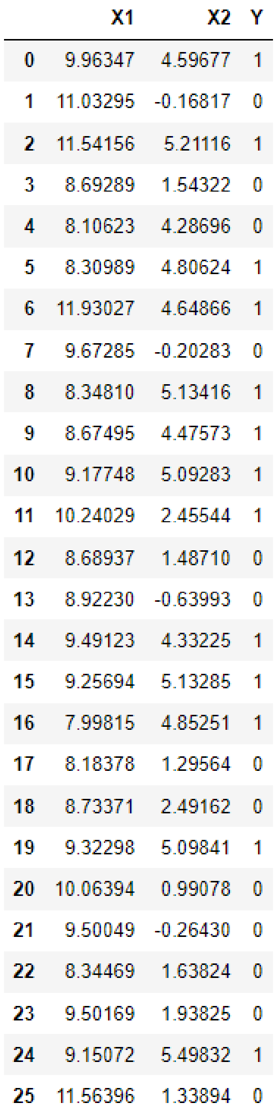
In[1]:

```
# generate dataset
X, y = mglearn.datasets.make_forge()
# plot dataset
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1"], loc=4)
plt.xlabel("First feature")
plt.ylabel("Second feature")
print("X.shape: {}".format(X.shape))
```

Out[1]:

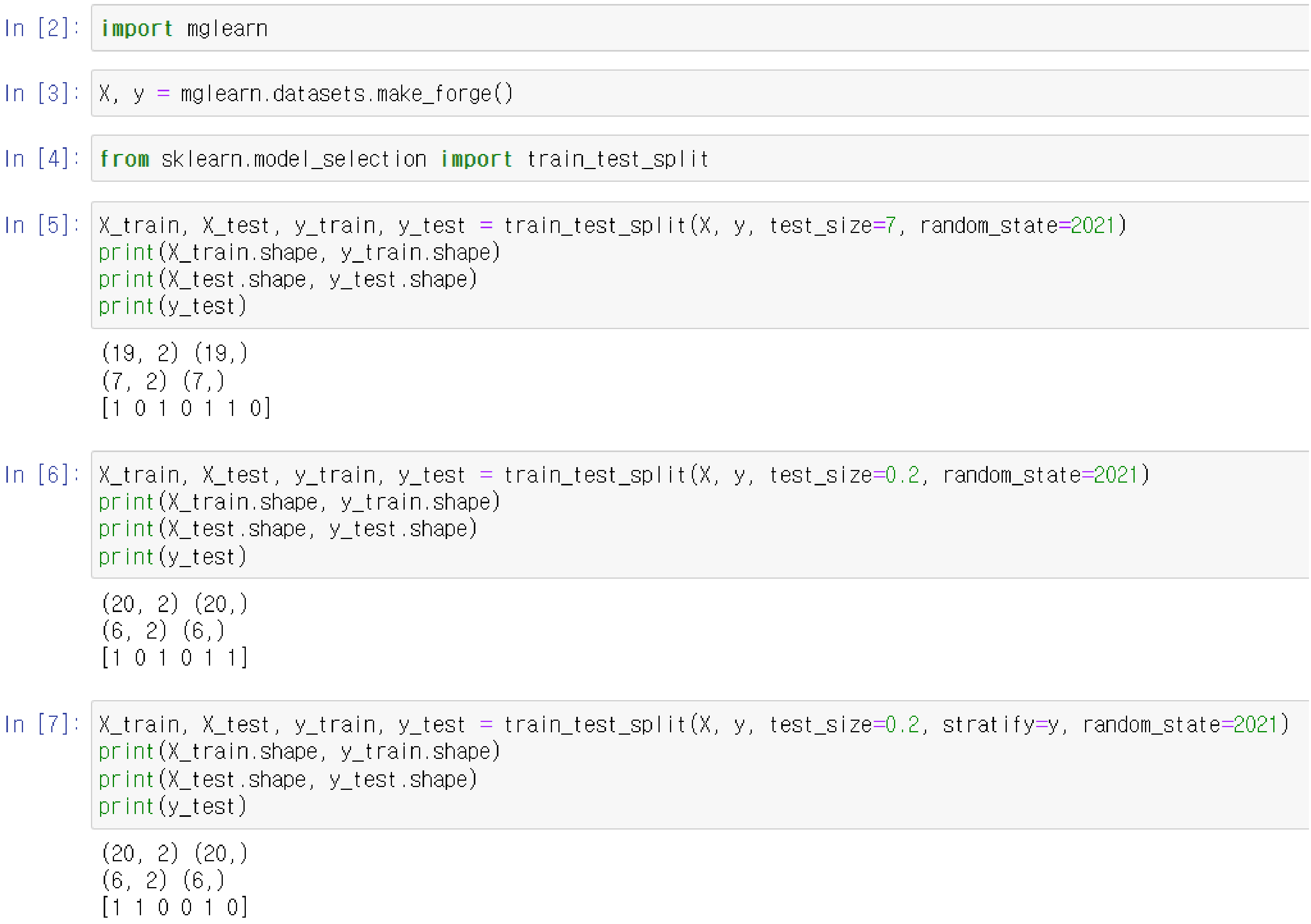
```
X.shape: (26, 2)
```





	X1	X2	Y
0	9.96347	4.59677	1
1	11.03295	-0.16817	0
2	11.54156	5.21116	1
3	8.69289	1.54322	0
4	8.10623	4.28696	0
5	8.30989	4.80624	1
6	11.93027	4.64866	1
7	9.67285	-0.20283	0
8	8.34810	5.13416	1
9	8.67495	4.47573	1
10	9.17748	5.09283	1
11	10.24029	2.45544	1
12	8.68937	1.48710	0
13	8.92230	-0.63993	0
14	9.49123	4.33225	1
15	9.25694	5.13285	1
16	7.99815	4.85251	1
17	8.18378	1.29564	0
18	8.73371	2.49162	0
19	9.32298	5.09841	1
20	10.06394	0.99078	0
21	9.50049	-0.26430	0
22	8.34469	1.63824	0
23	9.50169	1.93825	0
24	9.15072	5.49832	1
25	11.56396	1.33894	0

scikit-learn Practice: *train_test_split*



```
In [2]: import mglearn
```

```
In [3]: X, y = mglearn.datasets.make_forge()
```

```
In [4]: from sklearn.model_selection import train_test_split
```

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=7, random_state=2021)
        print(X_train.shape, y_train.shape)
        print(X_test.shape, y_test.shape)
        print(y_test)

        (19, 2) (19,)
        (7, 2) (7,)
        [1 0 1 0 1 1 0]
```

```
In [6]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2021)
        print(X_train.shape, y_train.shape)
        print(X_test.shape, y_test.shape)
        print(y_test)

        (20, 2) (20,)
        (6, 2) (6,)
        [1 0 1 0 1 1]
```

```
In [7]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=2021)
        print(X_train.shape, y_train.shape)
        print(X_test.shape, y_test.shape)
        print(y_test)

        (20, 2) (20,)
        (6, 2) (6,)
        [1 1 0 0 1 0]
```

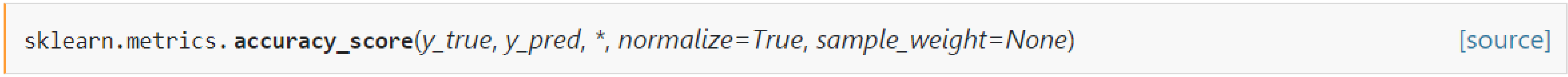
Performance Evaluation

- **Classification**

  Given a test set $D' = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$

  - **Accuracy**: the fraction of correctly classified data points

$$\text{Accuracy} = \frac{1}{n}\sum_i \mathrm{I}(y_i = \hat{y}_i) \times 100\%$$



```
sklearn.metrics.accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None)    [source]
```

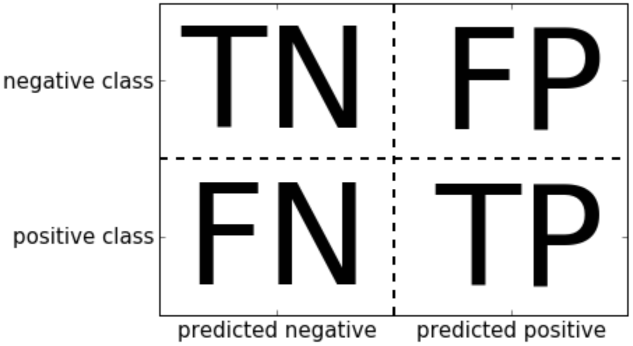
  - **Error Rate**: the fraction of mis-classified data points

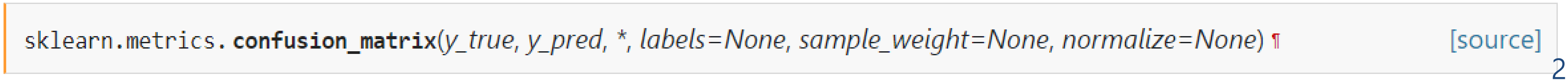
$$\text{Error Rate} = 1 - \text{Accuracy}$$

  - **Confusion Matrix**

    Positive Class (Our Main Interest), Negative Class

    FP: Type 1 Error, FN: Type 2 Error





```
sklearn.metrics.confusion_matrix(y_true, y_pred, *, labels=None, sample_weight=None, normalize=None) ¶    [source]
```

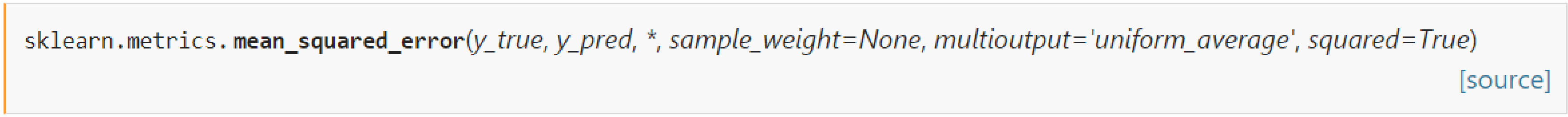
Performance Evaluation

- **Regression**

  Given a test set $D' = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$

  - **Root Mean Squared Error (RMSE)**

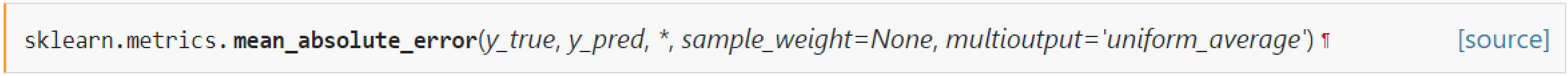
$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2}$$



```
sklearn.metrics.mean_squared_error(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average', squared=True)
[source]
```

  - **Mean Absolute Error (MAE)**

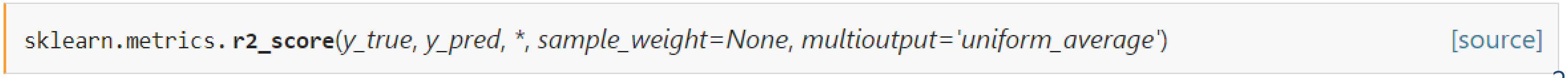
$$\text{MAE} = \frac{1}{n}\sum_i |y_i - \hat{y}_i|$$



```
sklearn.metrics.mean_absolute_error(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average') ¶   [source]
```

  - **Coefficient of Determination (R²)**

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \qquad \bar{y} = \frac{1}{n}\sum_i y_i$$



```
sklearn.metrics.r2_score(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average')   [source]
```