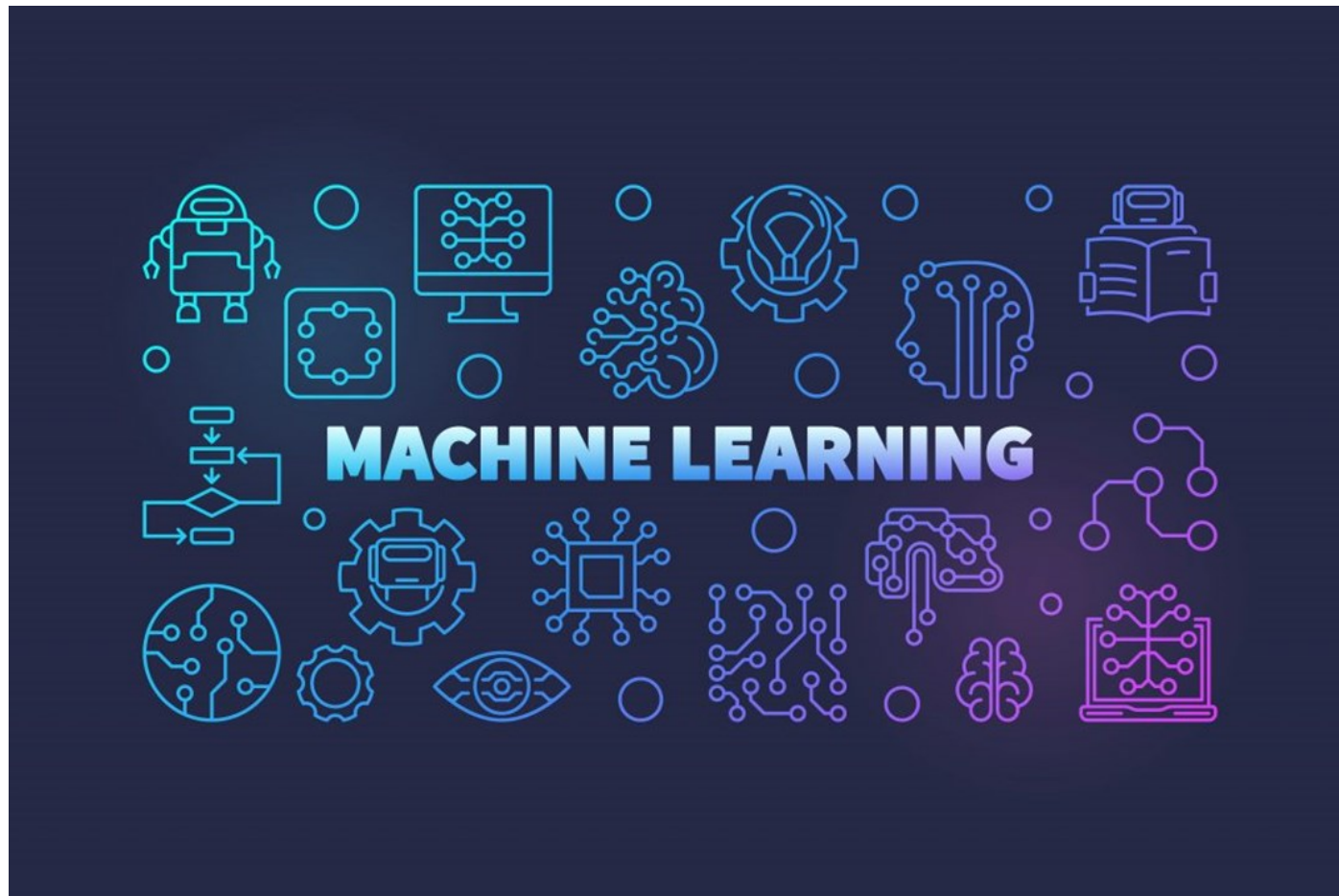


# 머신러닝 핵심 알고리즘



	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4



# 머신러닝

머신러닝은 컴퓨터 알고리즘이 데이터를 학습하여 입력과 출력간의 관계를 찾는 과정입니다.  
학습 데이터에 정답 레이블이 있는지 여부에 따라 지도학습과 비지도학습으로 구분을 합니다.

## ■ 지도학습(supervised learning)

- 학습시 정답을 알려 주면서 진행하는 학습으로, 학습시 데이터와 레이블(정답)이 함께 제공됩니다.
- **레이블(Label)** = 정답, 실제값, 타겟, 클래스,  $y$
- 예측된 값 = 예측값, 분류값,  $\hat{y}$  (y hat)
- 데이터마다 레이블을 달기 위해 많은 시간을 투자해야 합니다.
- 지도학습 **모델**에는 **분류모델**(이진분류, 다중분류)과 **회귀모델**(주가예측 등)이 있습니다.

## ■ 비지도학습(unsupervised learning)

- 레이블(정답) 없이 진행되는 학습으로, 데이터 자체에서 패턴을 찾아내야 할 때 사용합니다.
- 비지도학습의 대표적인 예는 군집화(clustering)와 차원축소가 있습니다.

# 지도학습

지도 학습은 정답이 있는 데이터를 활용해 데이터를 학습시키는 것입니다.  
입력 값(X data)과 입력값에 대한 Label(Y data)를 같이 제공하여 모델을 학습시킵니다.

## ■ 모델

- 데이터들의 패턴을 대표할 수 있는 함수, 예)  $f(x) = ax + b$
- 함수의 입력은 독립변수이고 출력은 종속변수로, 독립변수들에 의해 출력값이 정해집니다.

## ■ 분류 모델(Classification)

- 레이블의 값들이 이산적으로 나뉘질 수 있는 문제에 사용합니다.
- 예) 스팸 메일 분류, 품종 분류

Classification



## ■ 회귀 모델(Regression)

- 레이블의 값들이 연속적인 문제에 사용합니다.
- 예) 날씨 예측, 주가 예측

Regression



# 지도학습 데이터셋 구조

각 열(column)을 특징/속성(feature) 이라고 합니다.  
데이터 컬럼(column)중에 하나를 선택해서 레이블로 사용합니다.

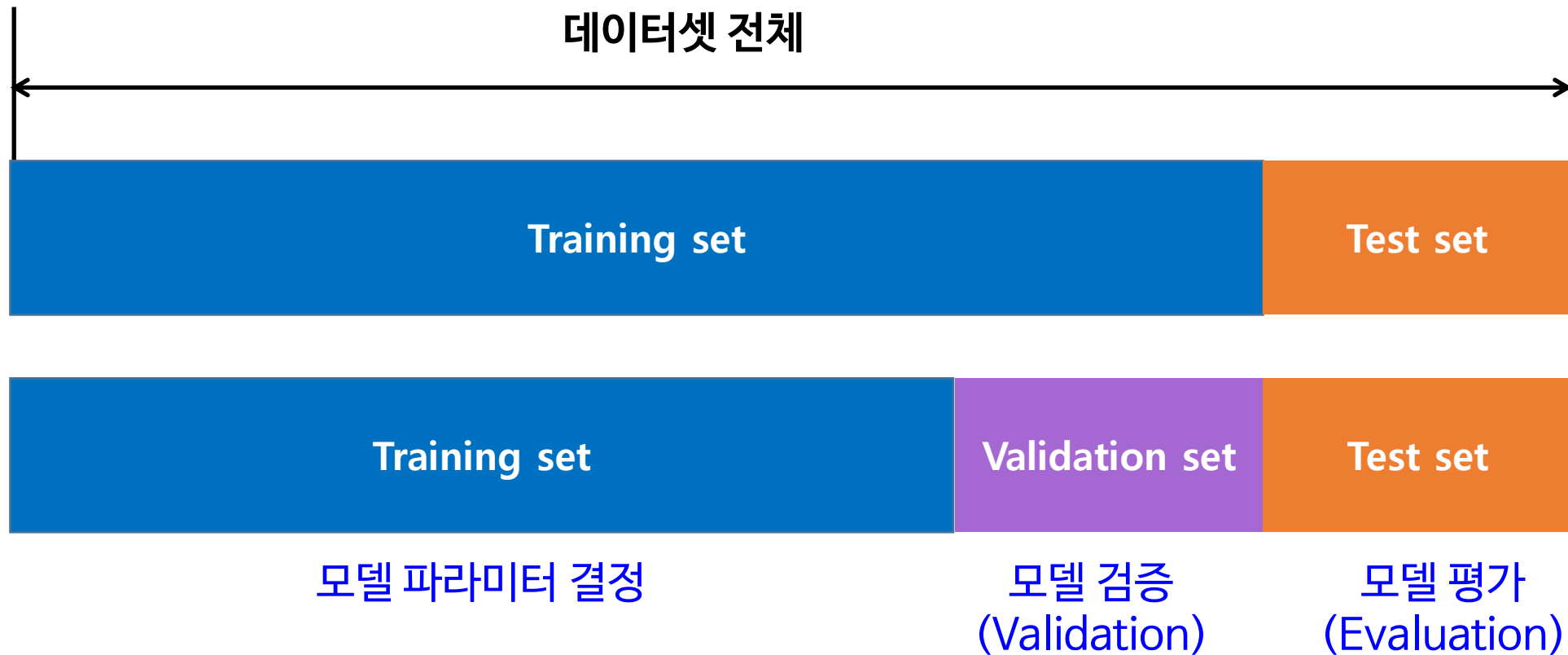
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

각 행(row)을  
예제 (Example)  
데이터라고  
합니다.

예측 모델 (회귀 모델)  
팁의 크기를 예측

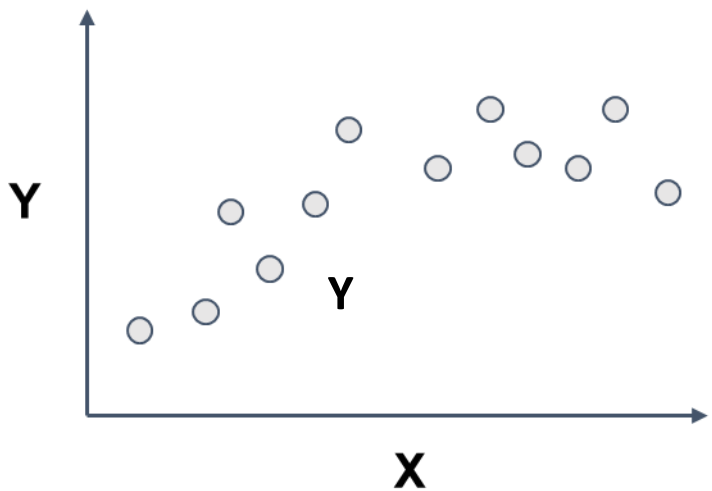
분류 모델  
손님의 성별을 예측

# 데이터셋 분리

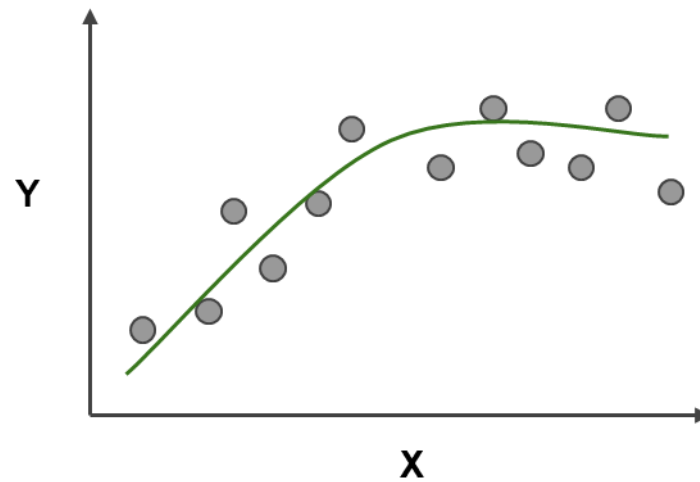


# 모델 선택

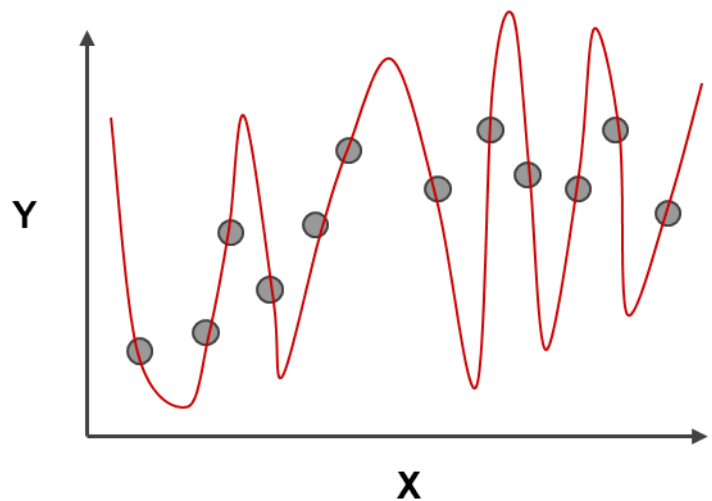
데이터



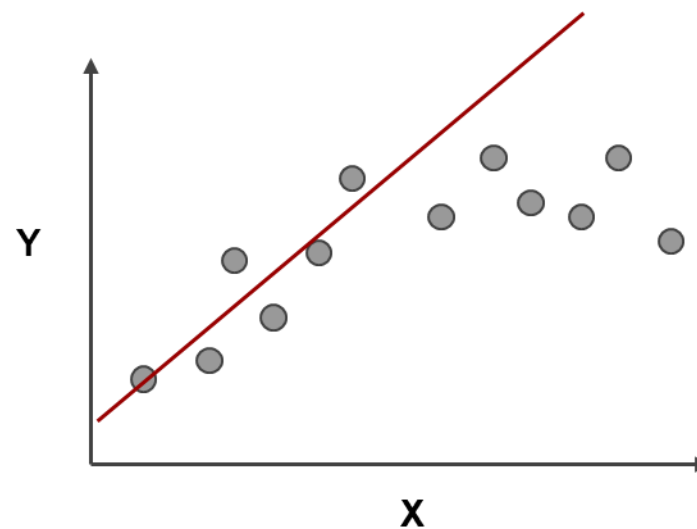
좋은 모델(Model)



과적합(Overfitting)

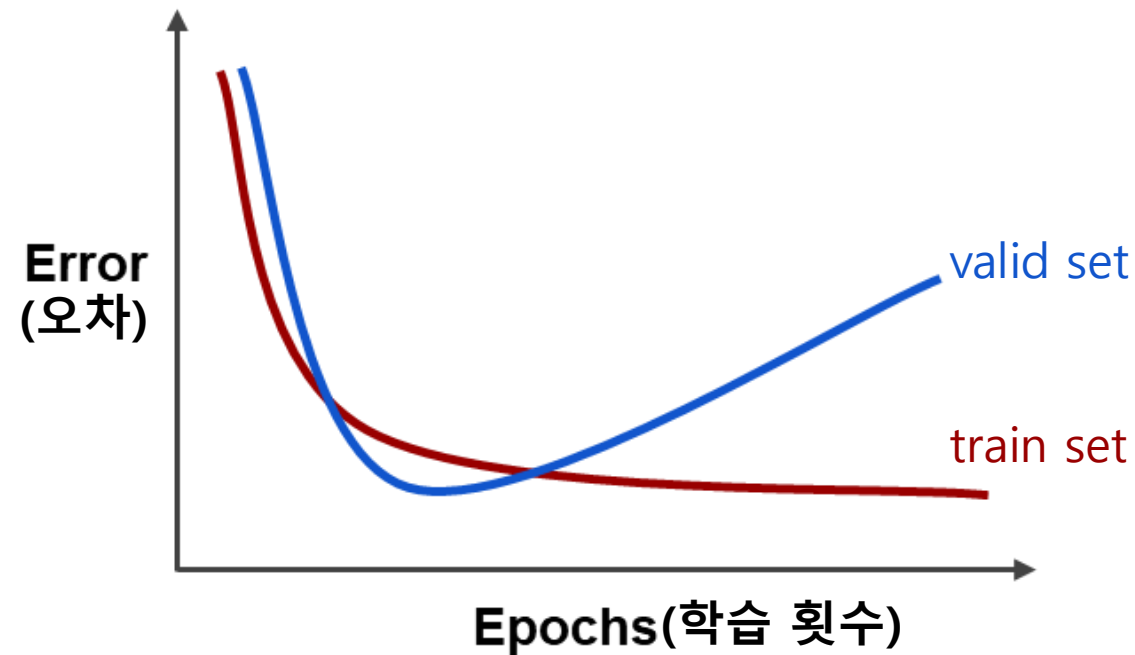
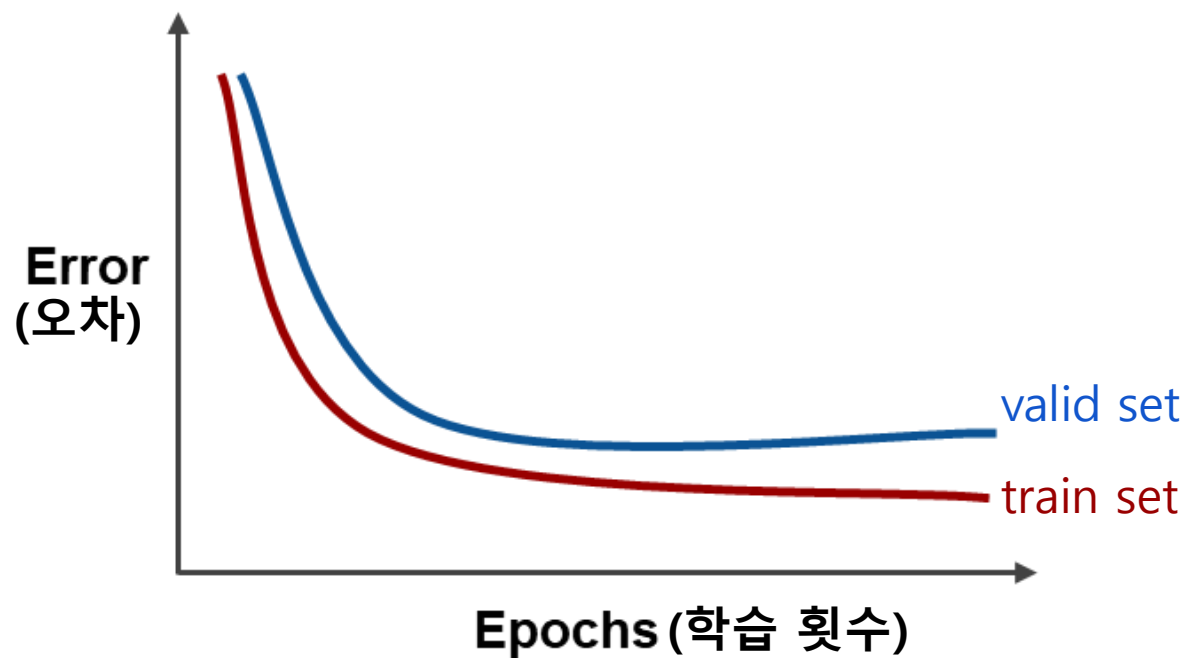


과소적합(Underfitting)



# 모델 성능 측정

모델학습을 진행하면서 손실(Loss, Error, 오차)을 지속적으로 측정합니다.

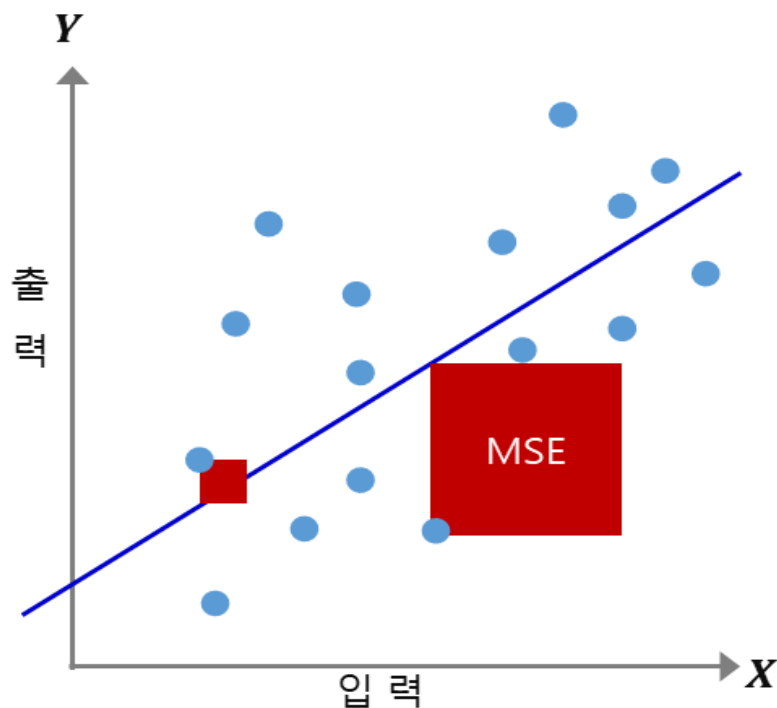




# 회귀모델 손실함수(Loss Function)

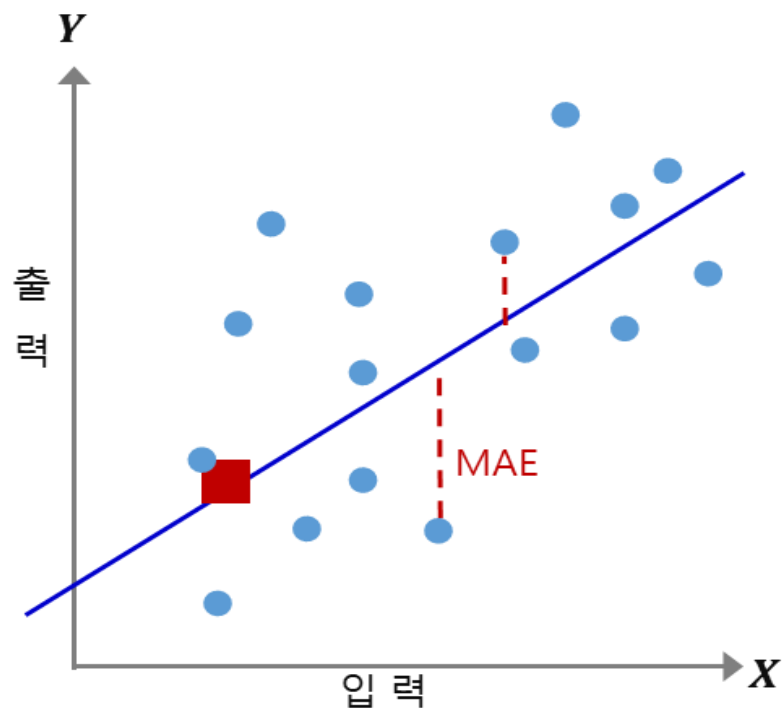
회귀모델(Regression)에서는 주로 평균제곱오차(MSE)를 손실함수로 사용합니다.

## ■ MSE(Mean Squared Error)



$$\frac{1}{n} \sum_{i=0}^n (y_i - \hat{y}_i)^2$$

## ■ MAE(Mean Absolute Error)



$$\frac{1}{n} \sum_{i=0}^n (|y_i - \hat{y}_i|)$$

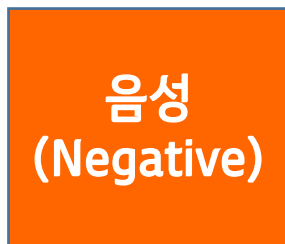
$y_i$  : 실제값  
 $\hat{y}_i$  : 모델 출력값(예측값)

# 분류모델 손실함수(Loss Function)

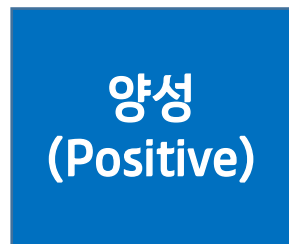
이진분류는 Binary Cross Entropy 를 다중분류는 Categorical Cross Entropy 손실함수를 사용합니다.

## ■ 이진분류(Binary Classification)







0



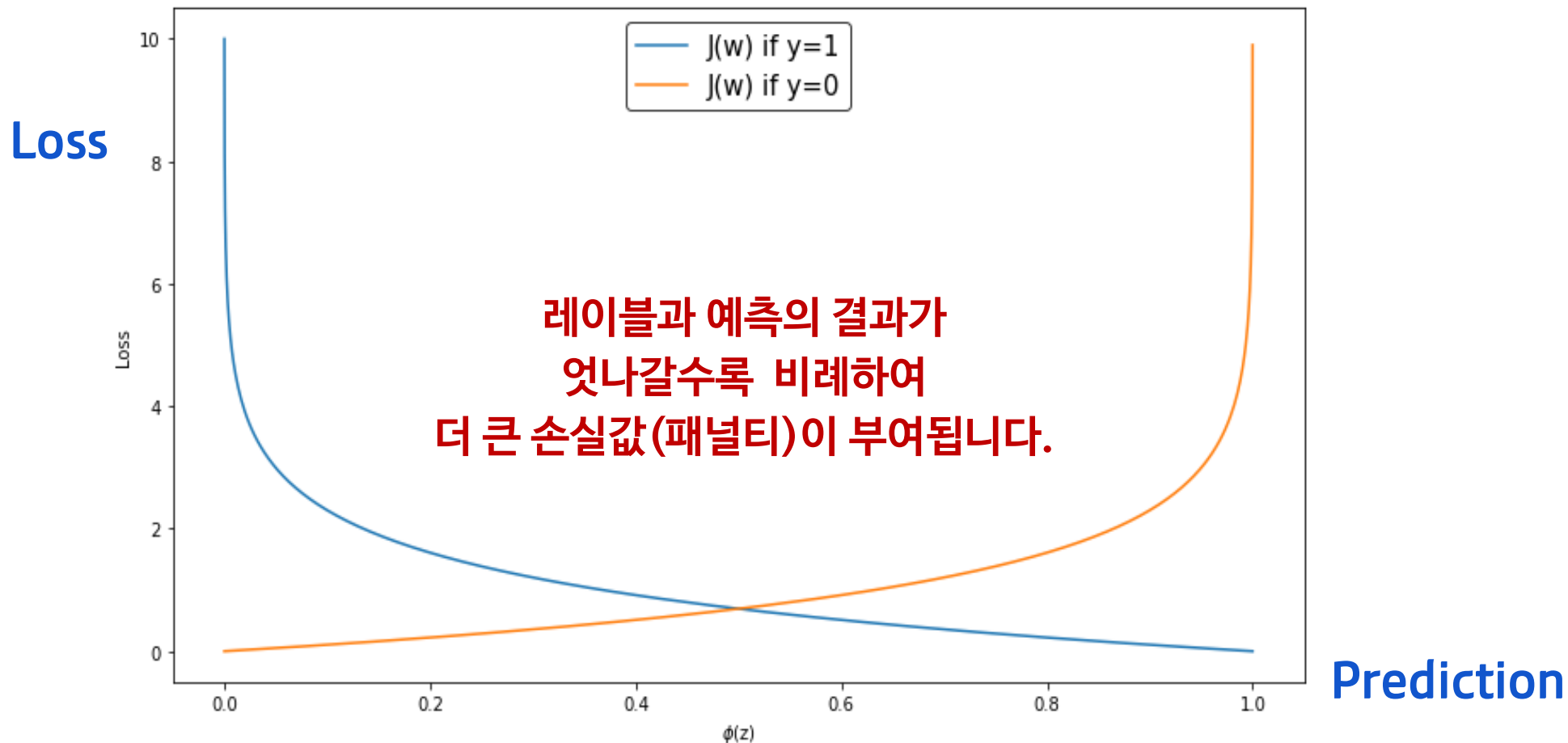
1



## ■ 다중분류(Multi-Class Classification)

C = 3	Samples		
  			
	Labels (t)		
	[0 0 1]	[1 0 0]	[0 1 0]

# 분류모델 손실함수(Loss Function)



$$\text{BCELoss}(\hat{y}_i, y_i) = -(y_i \times \log \hat{y}_i + (1 - y_i) \times \log(1 - \hat{y}_i))$$

$$\text{CCEE} = - \sum_{i=0}^n \text{CrossEntropy}(\hat{y}_i, y_i) \quad \text{CrossEntropy}(\hat{y}_i, y_i) = y_i \times \log \hat{y}_i$$

# 분류모델 성능측정 - 오차행렬 (Confusion Matrix)

분류모델 성능평가에 사용하며 대략적인 성능확인과 모델의 성능을 오차행렬을 기반으로 수치로 표현할 수 있습니다.

		예측값	
		Negative (0)	Positive (1)
실제값	Negative (0)	True Negative	False Positive
	Positive (1)	False Negative	True Positive

Positive(양성) = 1 = true

Negative(음성) = 0 = false

TP(True Positive) : 실제값이 Positive(양성, 1)이고, 예측값도 Positive(양성, 1)로 맞게 예측함

TN(True Negative) : 실제값이 Negative(음성, 0)이고, 예측값도 Negative(음성, 0)로 맞게 예측함

FP(False Positive) : 실제값은 Negative(음성, 0)인데, 예측값은 Positive(양성, 1)로 틀리게 예측함

FN(False Negative) : 실제값은 Positive(양성, 1), 예측값은 Negative(음성, 0)로 예측함,

# 분류모델 성능측정 - 평가지표

## ■ 정밀도(Precision)

모델이 True(Positive) 라고 분류한 것 중에서 실제 True(Positive) 인 것의 비율  
날씨 예측 모델이 맑다로 예측했는데, 실제 날씨가 맑았는지는 나타낸 지표입니다.

$$(Precision) = \frac{TP}{TP + FP} \quad \text{예측값 기준}$$

## ■ 재현율/회수율(Recall)

실제 True인 것 중에서 모델이 True라고 예측한 것의 비율  
실제 날씨가 맑은 날 중에서 모델이 맑다고 예측한 비율을 나타낸 지표입니다.

$$(Recall) = \frac{TP}{TP + FN} \quad \text{실제값 기준}$$

## ■ 정확도(Accuracy)

가장 직관적으로 모델의 성능을 나타낼 수 있는 평가 지표  
혼동행렬상에서는 대각선(TP)을 전체 셀로 나눈 값에 해당합니다.  
한달 동안에 맑은 날이 28일이고 비가 오는 날이 이틀인 경우, 비가 오는 것을 예측하는 성능은 매우 낮을 수 밖에 없으므로 이를 보완할 지표가 필요합니다.

$$(Accuracy) = \frac{TP + TN}{TP + FN + FP + TN}$$

## ■ F1 점수(F1 score)

정밀도와 재현율의 조화평균입니다.

$$(F1-score) = 2 \times \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

# 분류모델 성능측정 - 평가지표

실제값( $y_{\text{true}}$ ) = [1, 0, 1, 1, 0, 1]

예측값( $y_{\text{pred}}$ ) = [0, 0, 1, 1, 0, 0]

		예측값	
		0	1
실제값	0	2	0
	1	2	2

Precision =  $2/2 = 1$

Recall =  $2/4 = 0.5$

# 분류모델 성능측정

confusion.ipynb

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
```

```
y_true = [1, 0, 1, 1, 0, 1]
y_pred = [0, 0, 1, 1, 0, 0]
cm = confusion_matrix(y_true, y_pred)
```

```
print(cm)
```

```
[Out] array([[2, 0],
            [2, 2]], dtype=int64)
```

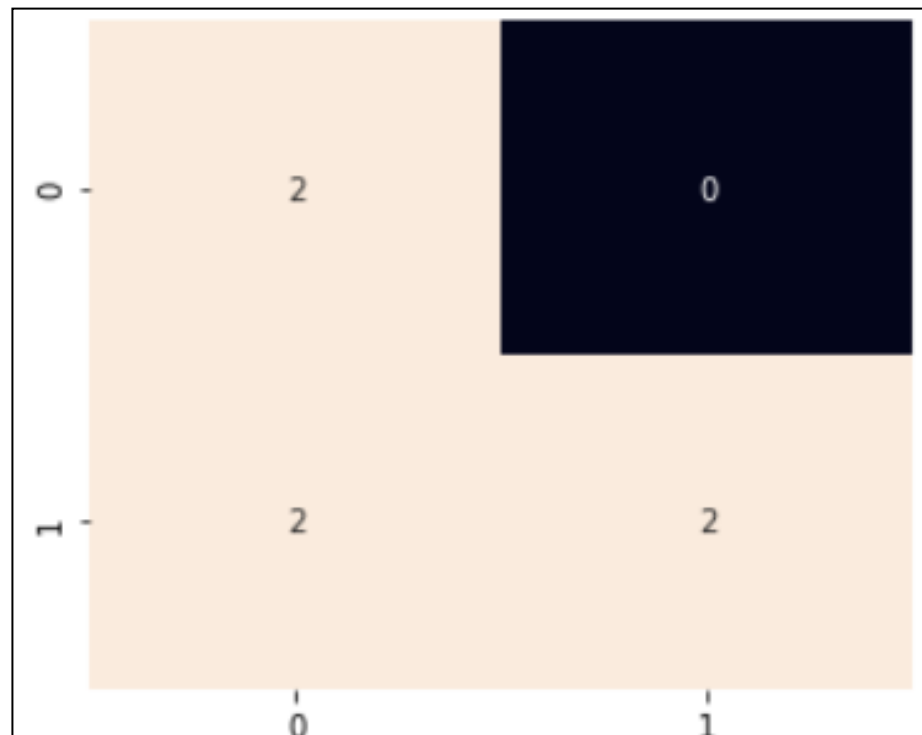
```
sns.heatmap(cm, annot=True)
```

```
precision_score(y_true, y_pred)
```

```
[Out] 1.0
```

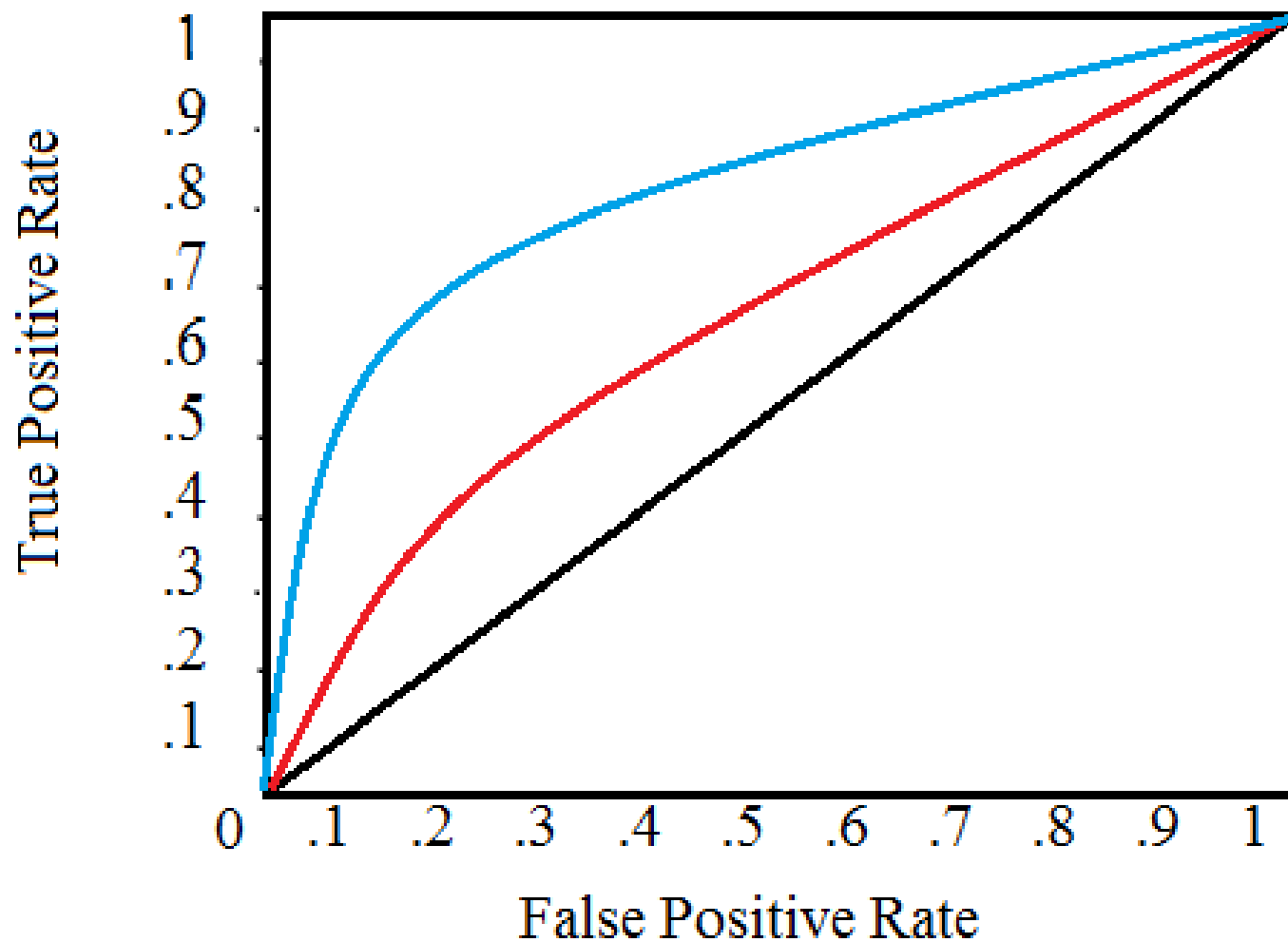
```
recall_score(y_true, y_pred)
```

```
[Out] 0.5
```



# 분류모델 성능측정 - ROC 곡선

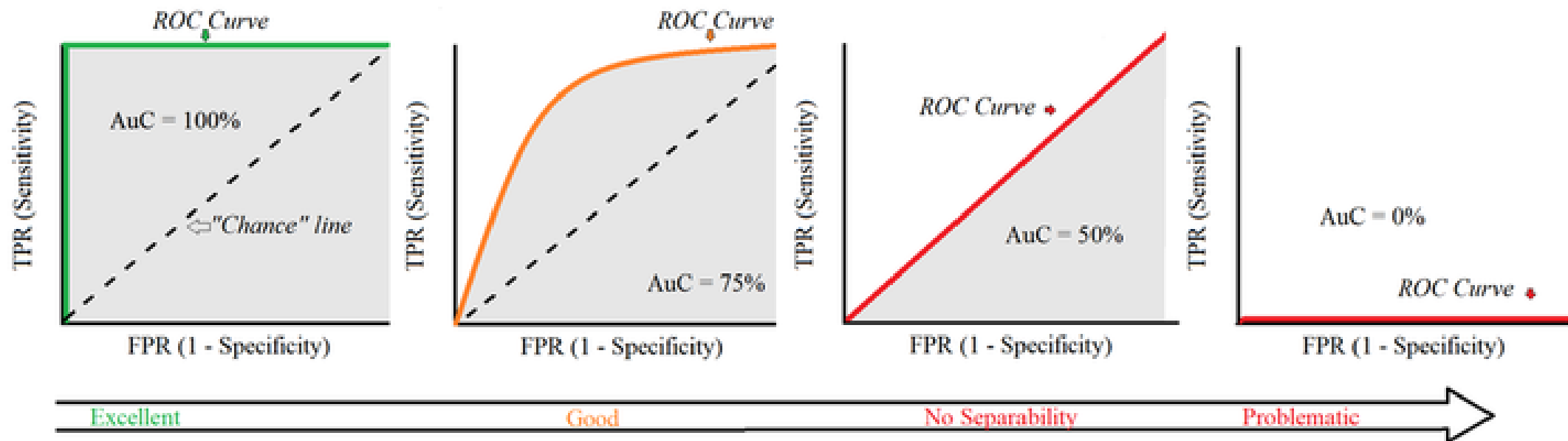
ROC(Receiver Operating Characteristic)는 거짓 양성 비율(False Positive Rate)에 대한 진짜 양성 비율(True Positive Rate, 재현율)의 곡선입니다. ROC 곡선으로 모델 성능을 평가 및 최적의 분류기준(threshold)을 찾을 수 있습니다.



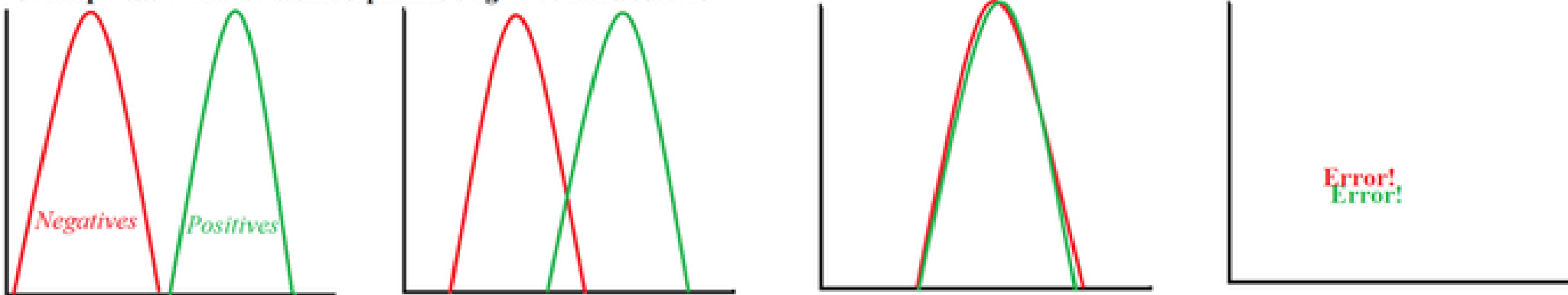


# 분류모델 성능측정 - AUC(Area under the Curve)

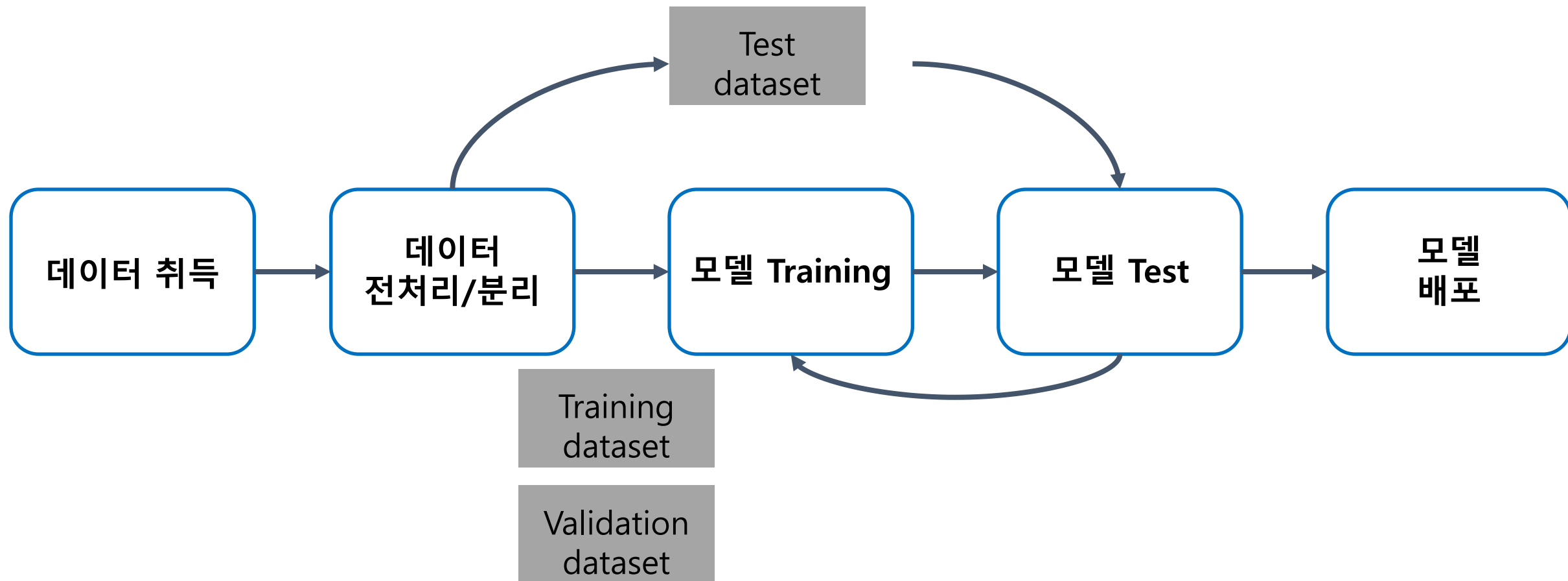
ROC 곡선 아래의 면적(AUC)을 측정하면 모델의 성능을 평가하거나 최적의 분류기준(threshold)을 찾을 수 있습니다.



Overlap = How well the model separates Negatives and Positives



# 머신러닝 프로세스



# 사이킷런 (Scikit-learn)

가장 인기있는 머신러닝 패키지이며, 많은 머신러닝 알고리즘이 내장되어 있습니다.

 [Install](#) [User Guide](#) [API](#)

<https://scikit-learn.org>

## scikit-learn

*Machine Learning in Python*

[Getting Started](#) [Release Highlights for 0.23](#) [GitHub](#)

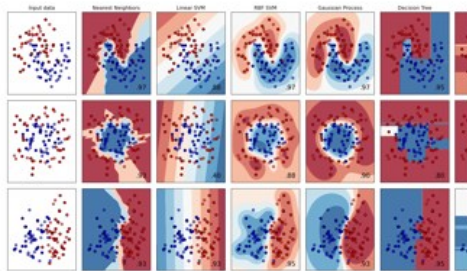
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...

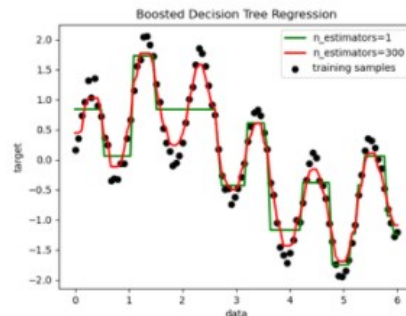


## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...



## Clustering

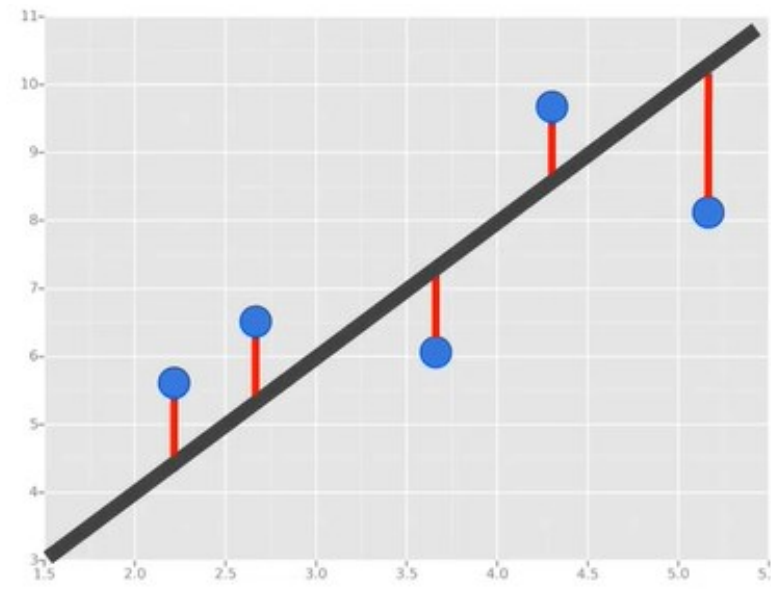
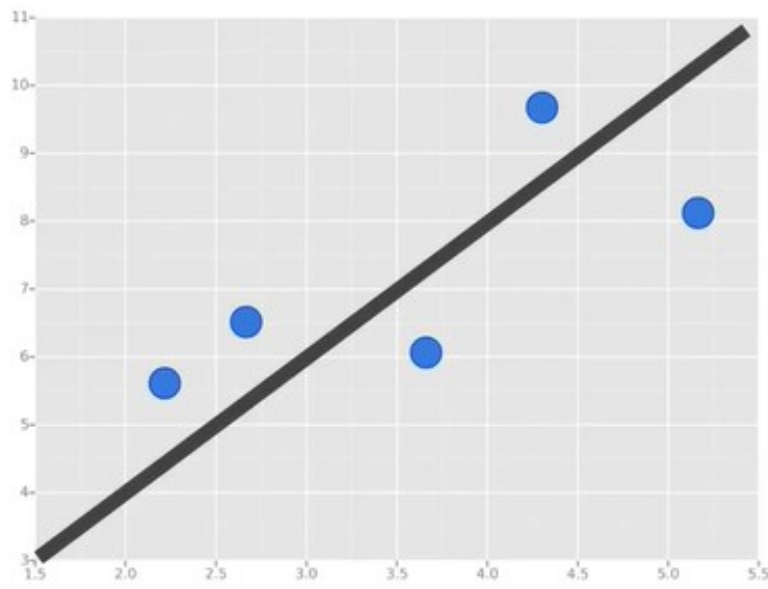
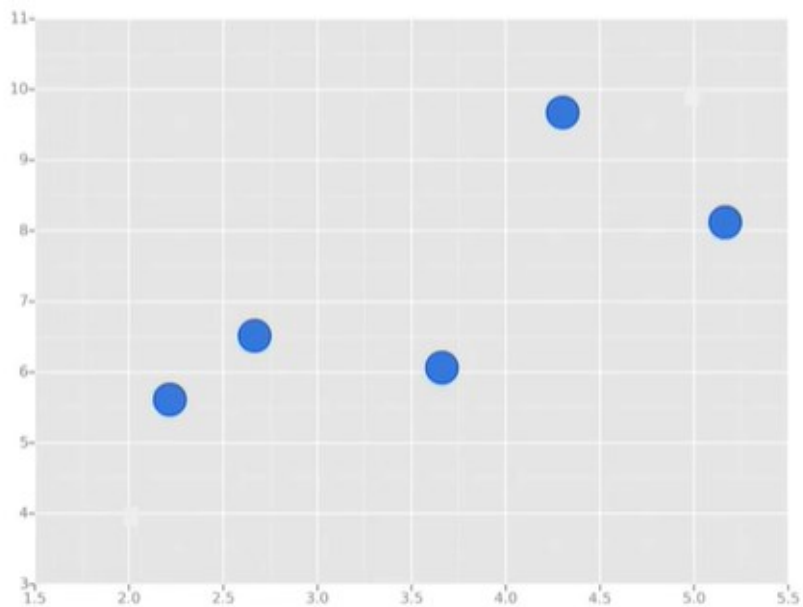
Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...



# 선형 회귀 (Linear Regression)



# 선형 회귀 (Linear Regression)



LinearRegression.ipynb

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1,1)
y = np.array([13, 25, 34, 47, 59, 62, 79, 88, 90, 100])
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

# 분류 (Classification)



Classification.ipynb

■ setosa



■ versicolor



■ virginica



```
from sklearn import datasets
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
```

# Train Test 데이터셋 분할

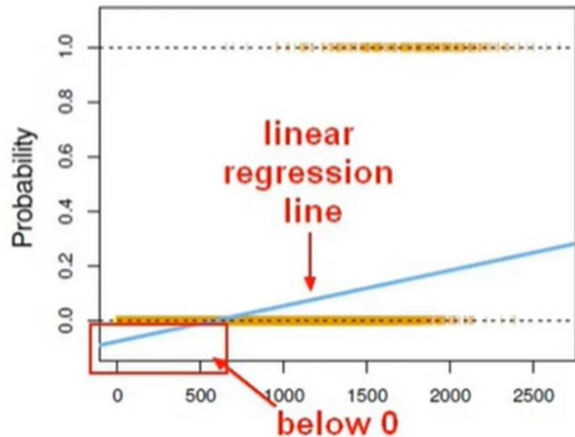
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    iris['data'],  
    iris['target'],  
    test_size=0.3,  
    shuffle=True,  
    stratify=iris.target,  
    random_state=42)
```

# 로지스틱 회귀(Logistic Regression)

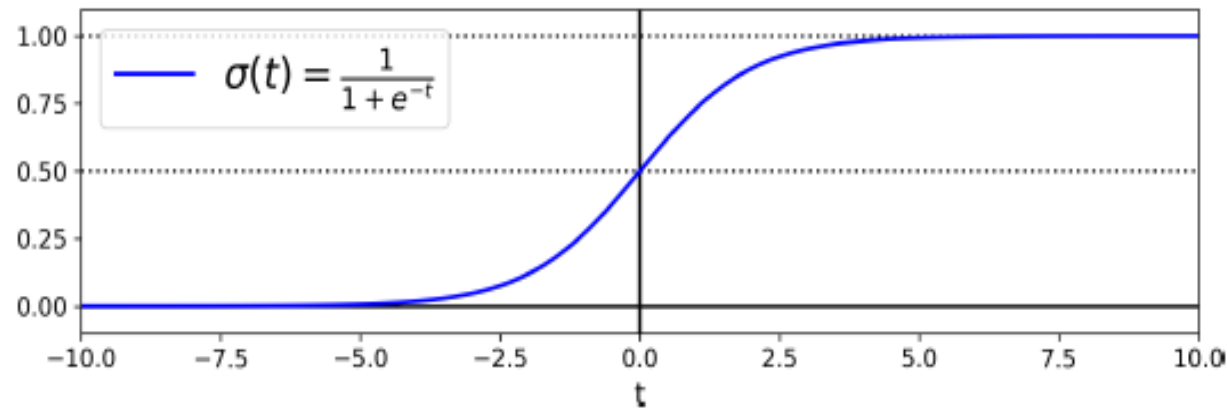
- 이진 분류 규칙은 0과 1의 두 클래스를 갖는 것으로, 일반 선형 회귀 모델을 이진분류에 사용할 수 없습니다.
- 대신 선형 회귀를 로지스틱 회귀 곡선으로 변환 할 수 있으며, 로지스틱 회귀 곡선은 0과 1 사이에서만 이동할 수 있으므로 분류에 사용할 수 있습니다.
- 로지스틱 회귀는 선형 회귀처럼 바로 결과를 출력하지 않고 로지스틱(logistic)을 출력합니다.
- 로지스틱 회귀는 샘플이 특정 데이터에 속할 확률을 추정(이진분류)하는 데 사용됩니다.
- 추정 확률이 50%가 넘으면 모델은 그 샘플이 해당 클래스에 속한다고 예측합니다.

일반 선형 모델



로지스틱 함수

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



로지스틱 확률모델

$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma(\mathbf{x}^T \boldsymbol{\theta})$$

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$



# 로지스틱 회귀(Logistic Regression)

```
from sklearn.linear_model import LogisticRegression
```

```
# 모델 학습
```

```
lr = LogisticRegression()  
lr.fit(X_train, y_train)
```

```
# 예측
```

```
pred = lr.predict(X_test)  
print(f'예측값: {pred[:10]}')
```

```
# 모델 성능 평가
```

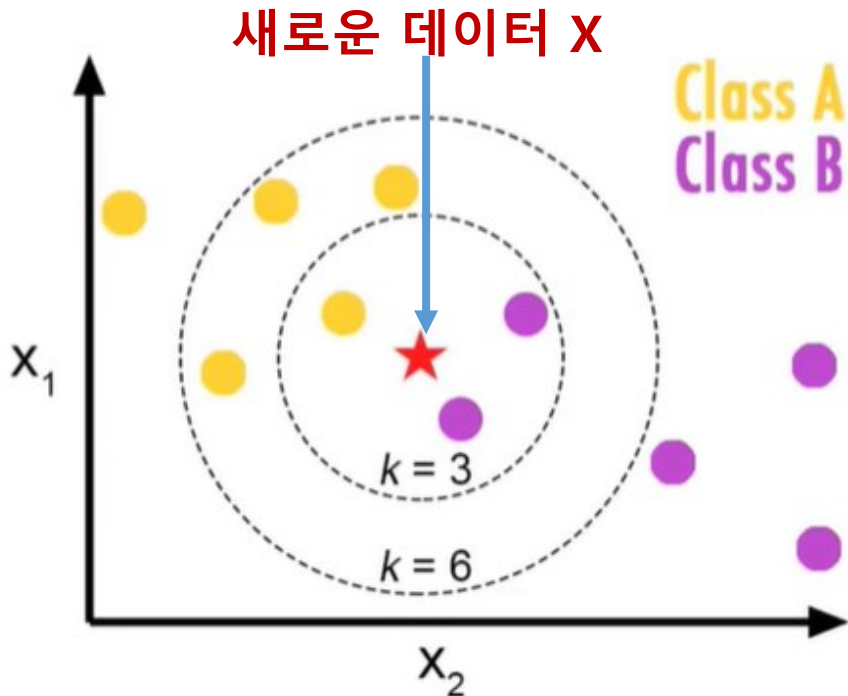
```
accuracy = accuracy_score(y_test, pred)  
print(f'Mean accuracy score: {accuracy:.4}')
```

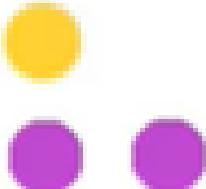
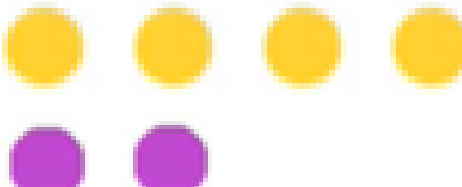
```
# 확률값 출력
```

```
prob = lr.predict_proba(X_test)  
print(f'Probability: {prob[0]}')
```

# KNN (K-Nearest Neighbor)

- KNN은 새로운 데이터가 주어졌을 때 기존 데이터 가운데 가장 가까운 k개 이웃의 정보로 새로운 데이터를 예측하는 방법론입니다. 아래 그림처럼 검은색 점의 범주 정보는 주변 이웃들을 가지고 추론해낼 수 있습니다.
- 만약 k값이 3이면 **Class B**, k가 6이면 **Class A**로 분류(classification)하는 것입니다.
- 만약, 회귀(regression) 문제라면 이웃들 종속변수(y)의 평균이 예측값이 됩니다.
- 알고리즘이 간단하며 큰 데이터셋과 고차원 데이터에 적합하지 않은 단점이 있습니다.



K	이웃(Neighbor)	예측값
3		Class B
7		Class A

# KNN (K-Nearest Neighbor)

```
from sklearn.linear_model import KNeighborsClassifier
```

```
# 모델 학습
```

```
knn = KNeighborsClassifier(n_neighbors=7)
```

```
knn.fit(X_train, y_train)
```

```
# 예측
```

```
pred = knn.predict(X_test)
```

```
print(f'예측값: {pred[:10]}')
```

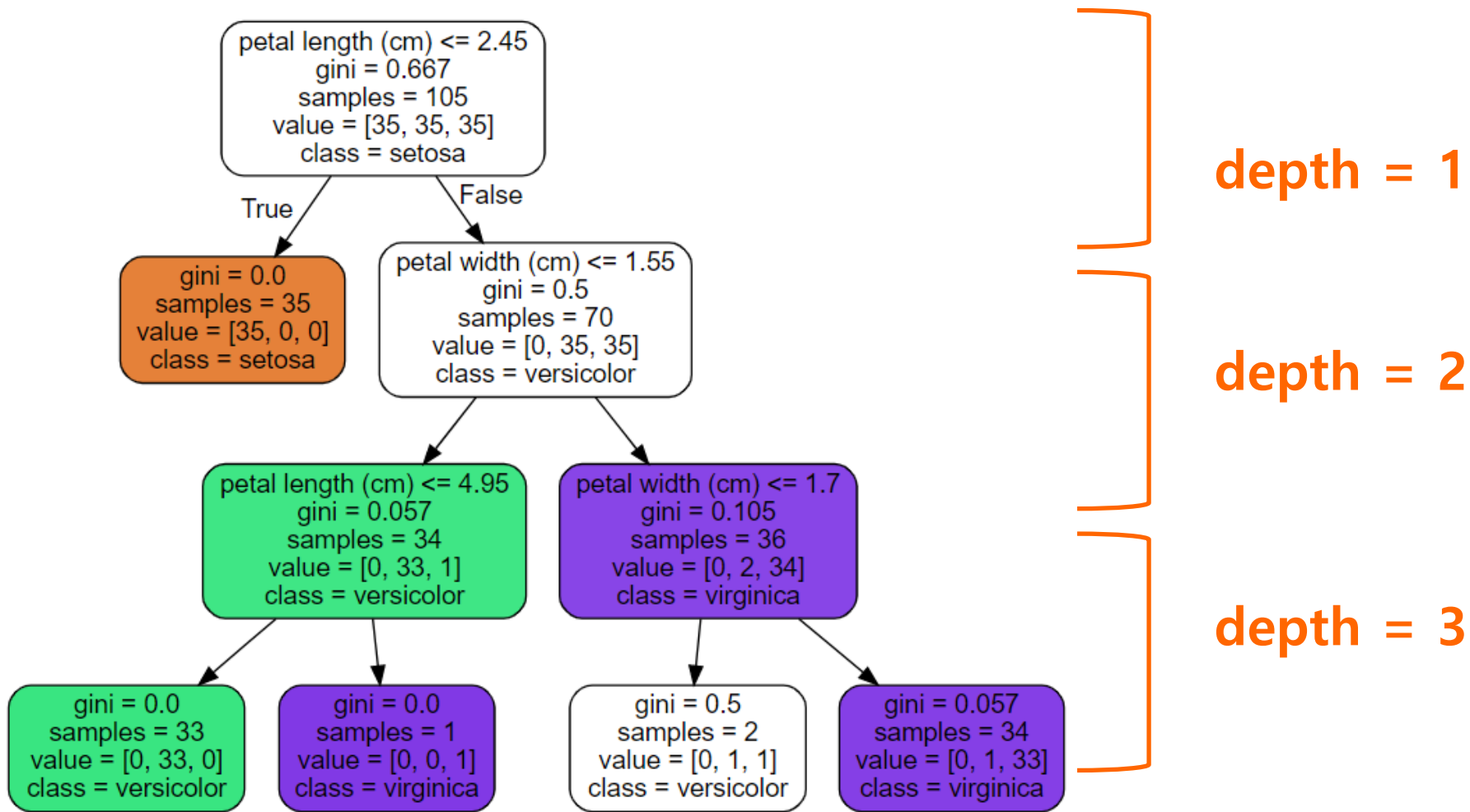
```
# 모델 성능 평가
```

```
accuracy = accuracy_score(y_test, pred)
```

```
print(f'Mean accuracy score: {accuracy:.4}')
```

# 의사결정트리 (Decision Tree)

의사결정트리 모델은 트리(Tree) 알고리즘을 사용합니다. 트리의 각 분기점(node)에 데이터셋의 Feature를 하나씩 위치시키고, 각 분기점(node)에서 임의의 조건식으로 가지를 나누면서 데이터를 구분합니다.



# 의사결정트리 (Decision Tree)

```
from sklearn.linear_model import DecisionTreeClassifier

# 모델 학습
dtc = DecisionTreeClassifier(max_depth=3, random_state=42)
dtc.fit(X_train, y_train)

# 예측
pred = dtc.predict(X_test)
print(f'예측값: {pred[:10]}')

# 모델 성능 평가
accuracy = accuracy_score(y_test, pred)
print(f'Mean accuracy score: {accuracy:.4}')

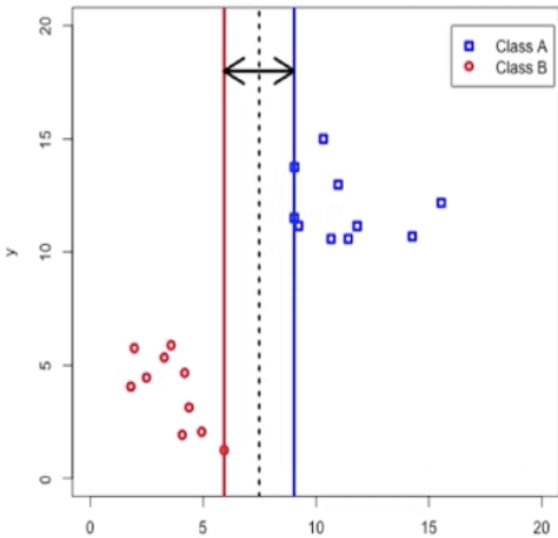
# 확률값 출력
prob = dtc.predict_proba(X_test)
print(f'Probability: {prob[0]}')
```

# 서포트 벡터 머신(SVM)

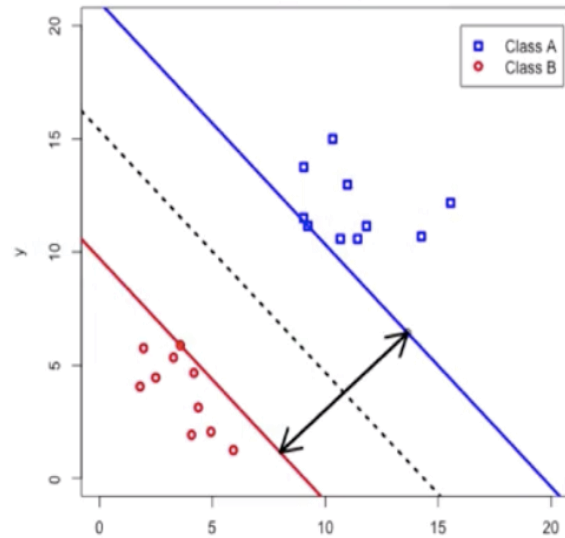
- 서포트 벡터 머신은 선형/비선형 분류, 회귀, 이상치 탐색에도 사용할 수 있는 다목적 머신러닝 모델입니다.
- SVM은 복잡한 분류 모델에 잘 들어 맞으며 작거나 중간 크기의 데이터셋에 적합합니다.

SVMs maximize the margin between two classes

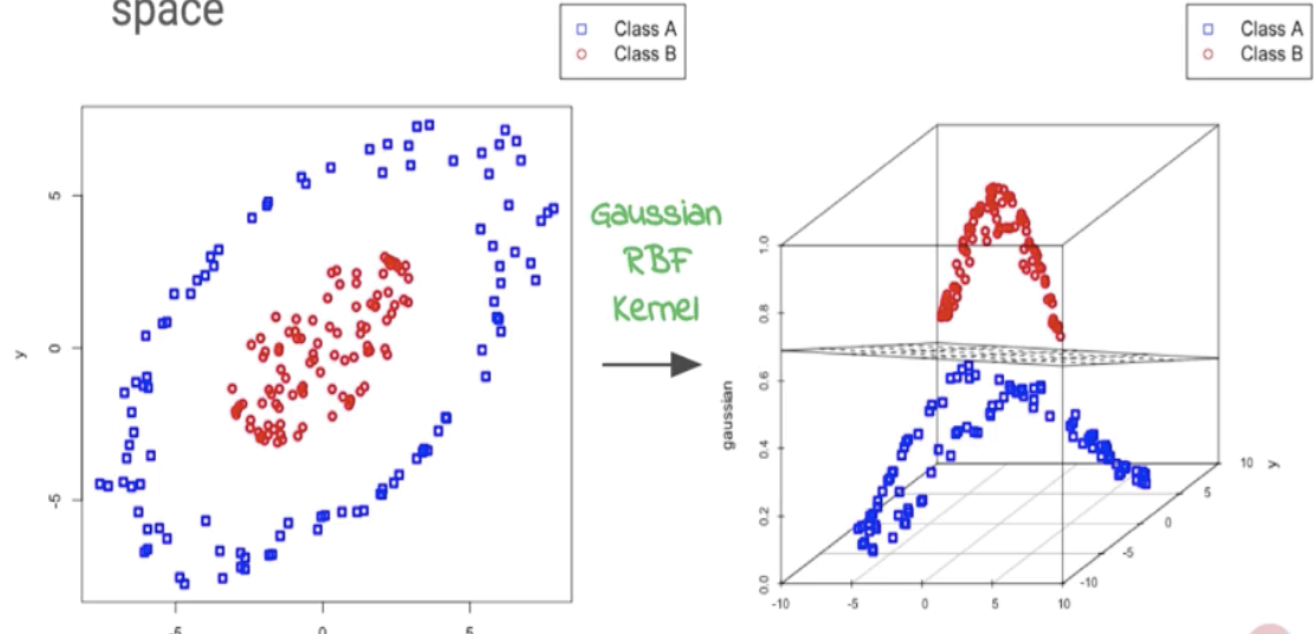
Small margin



Large margin



Kernels transform the input space into a more usable feature space



# 서포트 벡터 머신(SVM)

```
from sklearn.svm import SVC
```

```
# 모델 학습
```

```
svc = SVC(kernel='rbf')
```

```
svc.fit(X_train, y_train)
```

```
# 예측
```

```
pred = svc.predict(X_test)
```

```
print(f'예측값: {pred[:10]}')
```

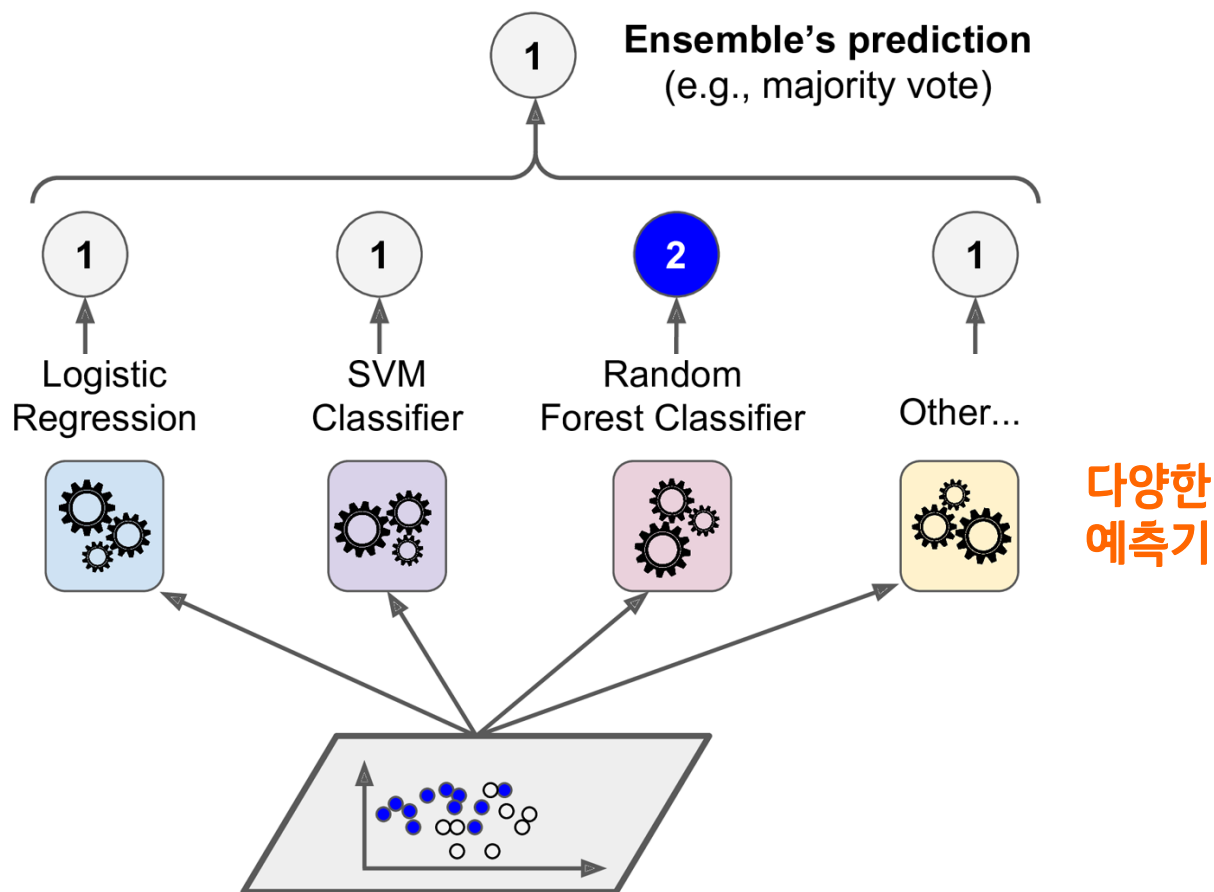
```
# 모델 성능
```

```
acc = accuracy_score(y_test, pred)
```

```
print(f'Mean accuracy score: {accuracy:.4}')
```

# 앙상블 학습(Ensemble Learning)

- 일련의 예측기(분류, 회귀)로부터 예측을 수집하면, 가장 좋은 모델 1개보다 더 좋은 예측을 얻을 수 있을 것입니다.
- 일련의 예측기를 앙상블이라 부르고 이를 앙상블 학습(Ensemble Learning)이라고 합니다.
- 가장 인기 있는 앙상블 방법에는 배깅, 부스팅이 있습니다.

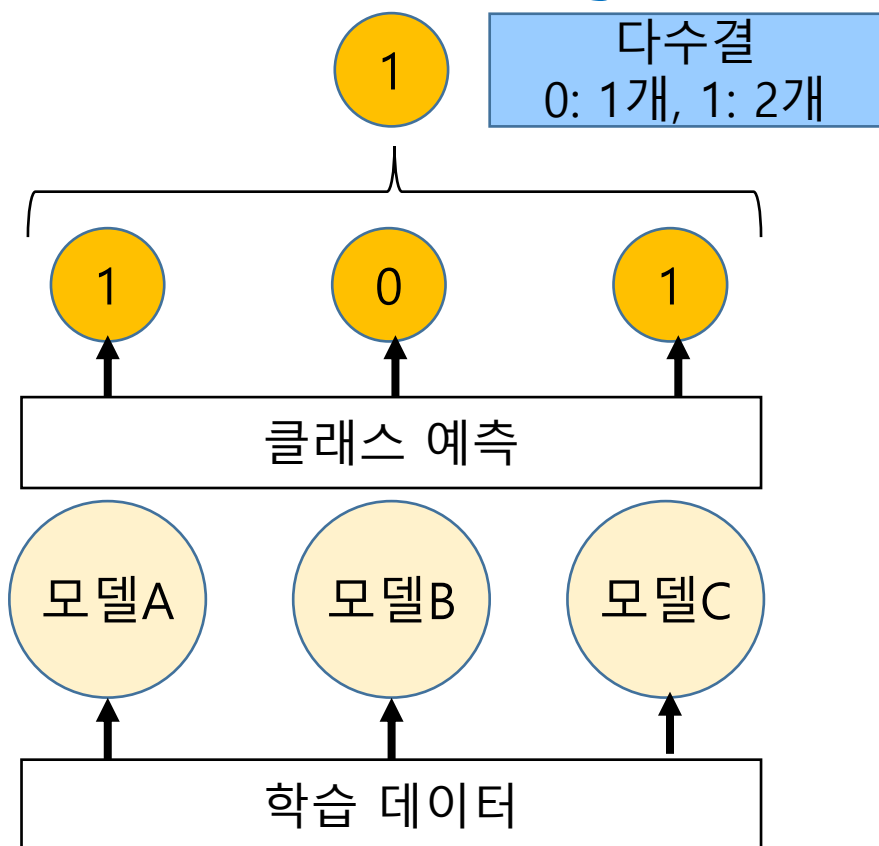




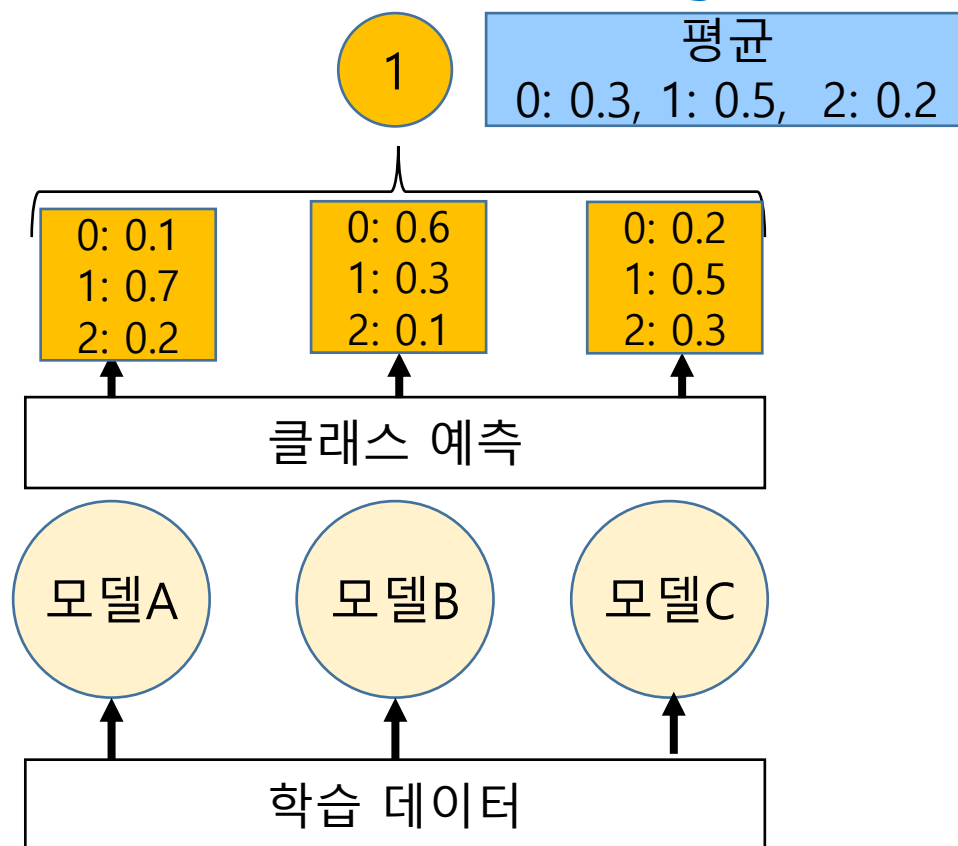
# 앙상블 모델 - 보팅(Voting)

- 보팅(Voting)은 여러 개의 모델이 예측한 값을 결합하여 최종 예측값을 결정하는 앙상블 방법입니다.
- 하드 보팅(hard voting)은 모델이 예측한 값 중에서 다수결로 최종 분류 클래스를 정합니다.
- 소프트 보팅(soft voting)은 각 분류 클래스별 예측 확률을 평균하여 최종 분류 클래스를 정합니다.

## ■ 하드 보팅(hard voting)



## ■ 소프트 보팅(soft voting)



# 앙상블 모델 - 보팅(Voting)

```
from sklearn.ensemble import VotingClassifier

# 모델 학습
hvc = VotingClassifier(estimators=[('KNN', knn), ('DT', dtc),
                                   ('SVM', svc)], voting='hard')
hvc.fit(X_train, y_train)

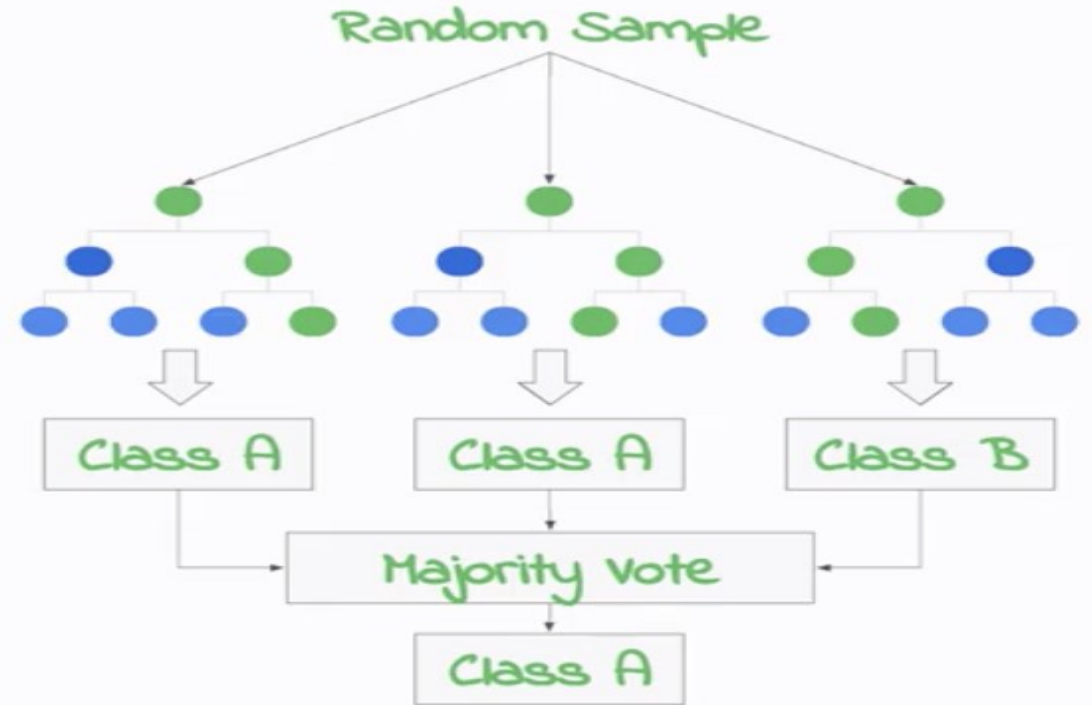
# 예측
pred = hvc.predict(X_test)
print(f'예측값: {pred[:10]}')

# 모델 성능 평가
accuracy = accuracy_score(y_test, pred)
print(f'Mean accuracy score: {accuracy:.4}')
```

# 앙상블 모델 - 배깅(Bagging)

- 다양한 분류기를 만드는 각기 다른 훈련 알고리즘을 사용하는 것과, 같은 알고리즘을 사용하고, 훈련 세트의 서브셋을 무작위로 구성하여 각기 다르게 학습시키는 방법이 있습니다.
- 훈련세트에서 중복을 허용하여 샘플링 하는 방식을 **bootstrap aggregating**, 배깅(**bagging**)라고 합니다.
- 통계학에서 중복을 허용한 리샘플링을 부트스트래핑(bootstrapping)이라고 합니다.
- 중복을 허용하지 않고 샘플링 하는 방식은 페이스팅(pasting)이라고 합니다.
- 랜덤 포레스트(Random Forest)는 일반적으로 배깅(또는 페이스팅)을 적용한 의사결정트리의 앙상블입니다.

Random forest: Strong learner from many weak learners



# 앙상블 모델 - 랜덤 포레스트(Random Forest, 배경)

```
from sklearn.ensemble import RandomForestClassifier
```

# 모델 학습

```
rfc = RandomForestClassifier(n_estimators=50, max_depth=3,  
                             random_state=20)
```

```
rfc.fit(X_train, y_train)
```

# 예측

```
pred = rfc.predict(X_test)  
print(f'예측값: {pred[:10]}')
```

# 모델 성능 평가

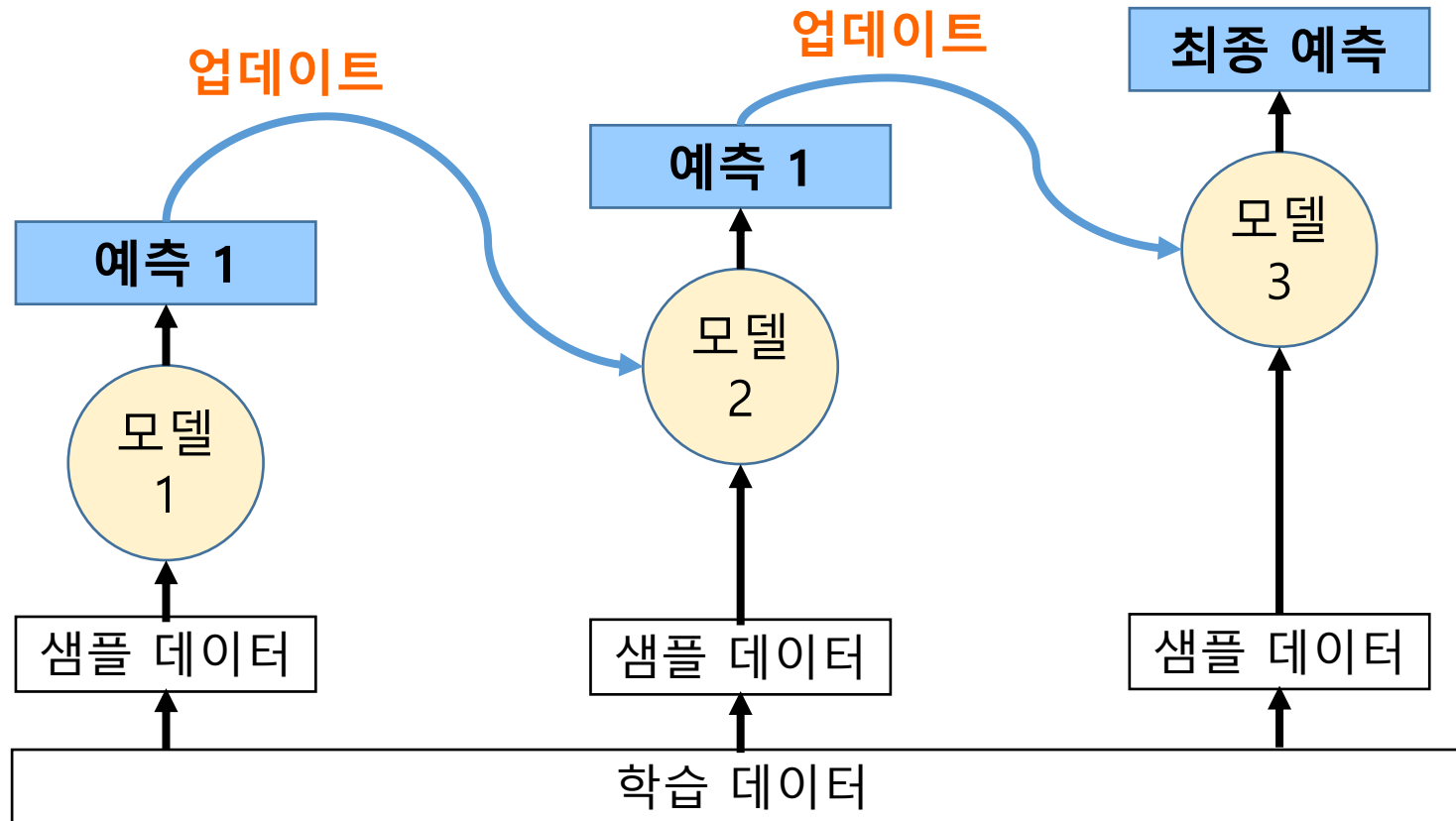
```
accuracy = accuracy_score(y_test, pred)  
print(f'Mean accuracy score: {accuracy:.4}')
```

shift+tab키 : 함수 설명 보기

```
Init signature:  
RandomForestClassifier(  
    n_estimators=100,  
    *,  
    criterion='gini',  
    max_depth=None,  
    min_samples_split=2,  
    min_samples_leaf=1,  
    min_weight_fraction_leaf=0.0,  
    max_features='auto',
```

# 앙상블 모델 - 부스팅(Boosting)

- 부스팅(Boosting)은 여러 개의 모델을 순차적으로 학습합니다.
- 잘못 예측한 데이터에 대한 예측 오차를 줄일 수 있는 방향으로 모델을 계속 업데이트 합니다.
- XGBoost, LGBM모델은 Kaggle(<https://www.kaggle.com/>) 경진대회에서 많이 사용되고 있는 알고리즘입니다.



# 앙상블 모델 - XGBoost (Extreme Gradient Boosting, 부스팅)

```
!pip install xgboost
```

```
from xgboost import XGBClassifier
```

```
# 모델 학습
```

```
xgbc = XGBClassifier(n_estimators=50, max_depth=3, random_state=42)
```

```
xgbc.fit(X_train, y_train)
```

```
# 예측
```

```
pred = xgbc.predict(X_test)
```

```
print(f'예측값: {pred[:10]}')
```

```
# 모델 성능 평가
```

```
acc = accuracy_score(y_test, pred)
```

```
print(f'Mean accuracy score: {accuracy:.4}')
```

# 앙상블 모델 - Light GBM (Gradient Boosting Machine, 부스팅)

```
from xgboost import LGBMClassifier
from sklearn.metrics import confusion_matrix

# 모델 학습
lgbc = LGBMClassifier(n_estimators=50, max_depth=3, random_state=42)
lgbc.fit(X_train, y_train)

# 예측
pred = lgbc.predict(X_test)
print(f'예측값: {pred[:10]}')

# 모델 성능 평가
acc = accuracy_score(y_test, pred)
print(f'Mean accuracy score: {accuracy:.4}')
print(confusion_matrix(y_test, y_pred))
```

# Thank you