PART II

다양한 MODEL

ML 모델과 데이터와 학습

ML 모델과 데이터와 학습

◆ 편향과 분산

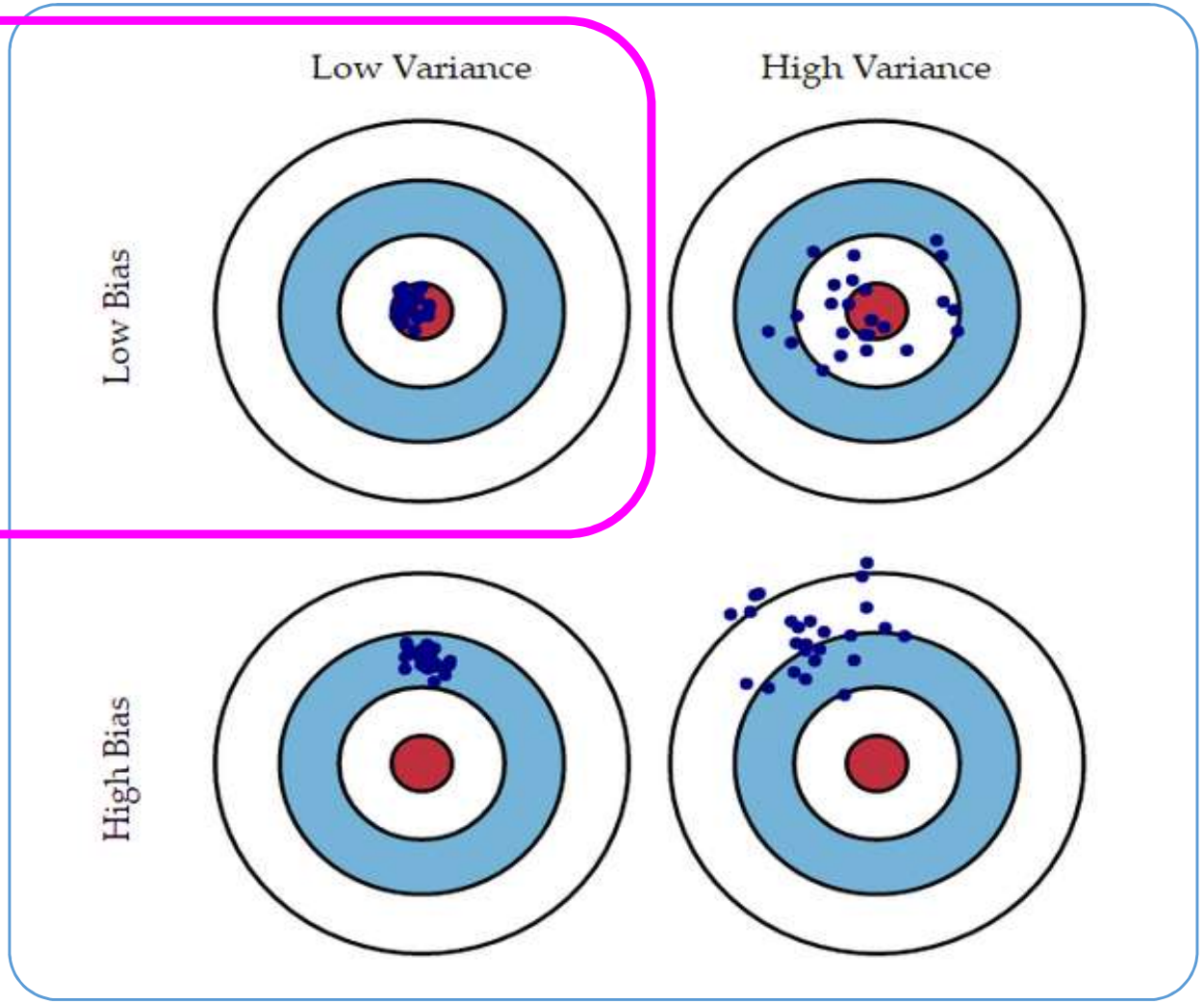
- **모델 예측값에 대한 표현**
  - **편향 (Bias)** ➔ **예측값과 정답** 간의 관계
    ➔ 예측값과 실제값 사이의 제곱 에러
  - **분산(Variance)** ➔ **예측값 끼리**의 관계

ML 모델과 데이터와 학습

◆ 편향과 분산

편향 ▼ 분산 ▼

좋은 모델(Model)



ML 모델과 데이터와 학습

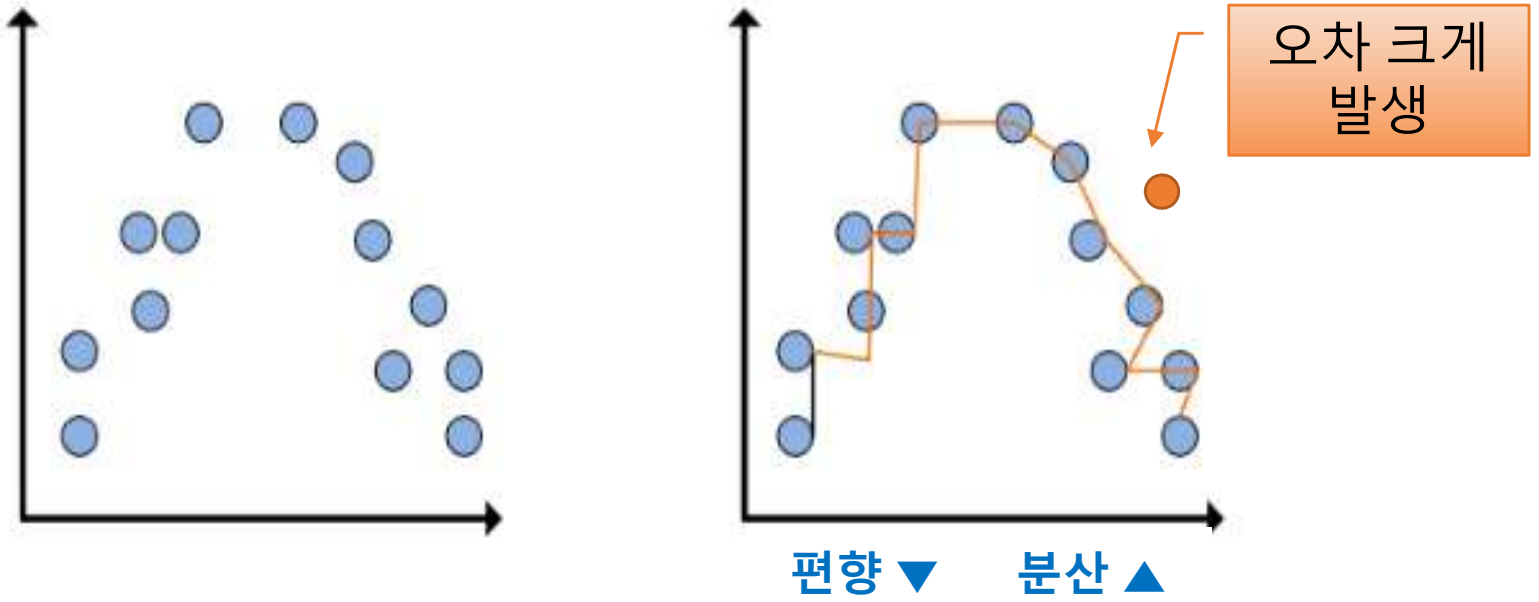
◆ 과대적합(Overfitting)

- **훈련 데이터 셋 특화된 즉 최적화된 모델**
- 새로운 데이터에 대한 **오차가 매우 커짐**
- **원인**
  - 데이터(특성)이 많아 **모델** 지나치게 **복잡**
  - 너무 **많은 학습**

ML 모델과 데이터와 학습

◆ 과대적합(Overfitting)

- 훈　련 데이터 ➔ 오차 없음
- 새로운 데이터 ➔ 오차 크게 발생 확률 높음!



ML 모델과 데이터와 학습

◆ 과소적합(Unterfitting)

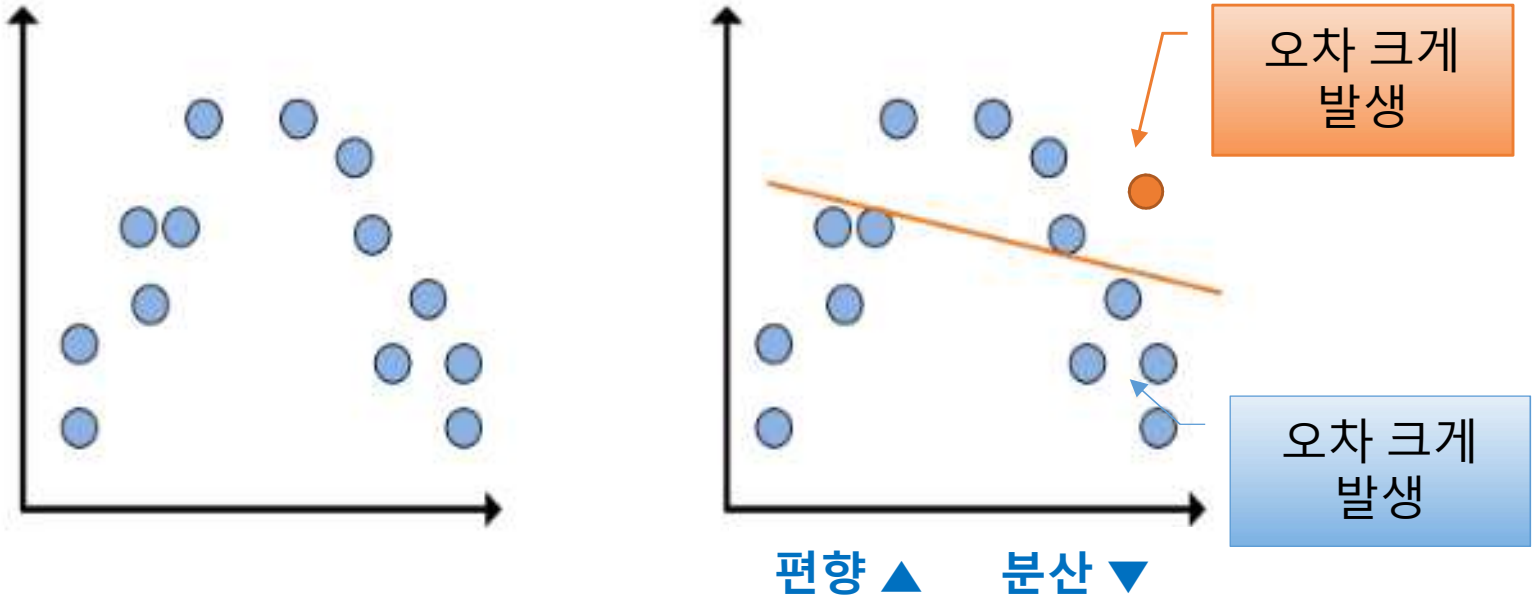
- **훈련 데이터 셋의 규칙/패턴 반영되지 않는 모델**
- 훈련 데이터와 새로운 데이터 대한 **오차가 매우 커짐**
- **훈련 데이터 오류가 줄어들지 않음**

- 원인
  - ➢ 데이터(특성)이 부족하여 **모델** 지나치게 **단순**
  - ➢ **학습** 횟수 **부족**

ML 모델과 데이터와 학습

◆ 과소적합(Unterfitting)

- 훈　련 데이터 ➔ 오차 크게 발생
- 새로운 데이터 ➔ 오차 크게 발생 확률 높음!



ML 모델과 데이터와 학습

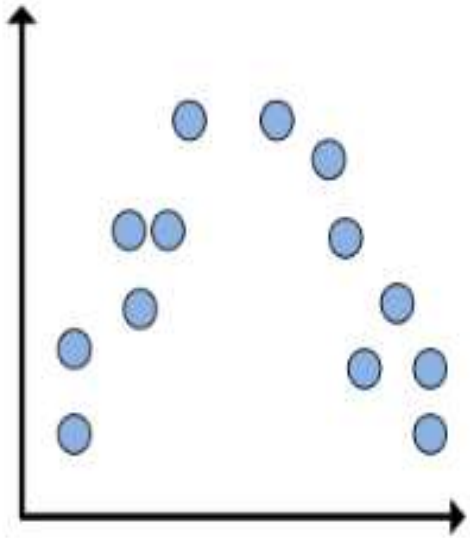
◆ 최적적합(Optimalfitting)

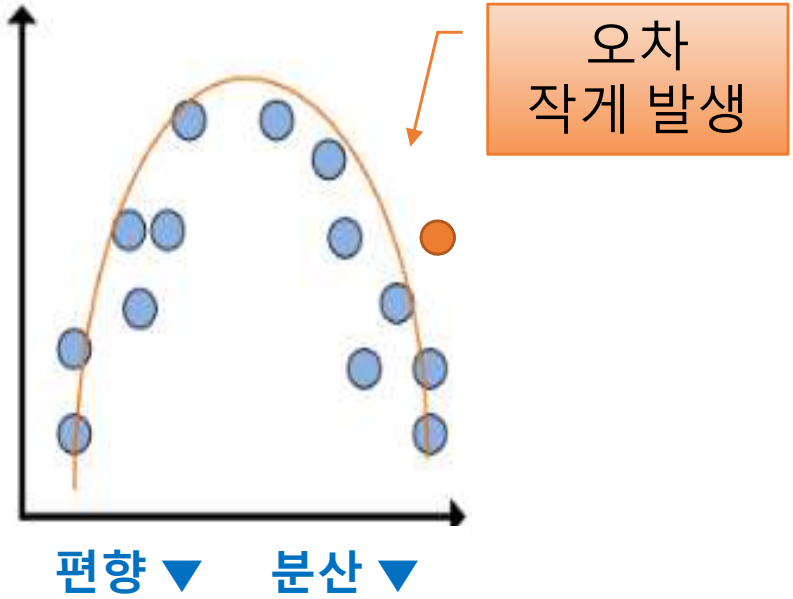
- **훈련 데이터 셋의 규칙/패턴이 일반화(Generalization) 된 모델**
- 훈련 데이터셋과 새로운 데이터 대한 오차 및 정확도 비슷
- **새로운 데이터에 대한 정확도 높음**

ML 모델과 데이터와 학습

◆ 최적적합(Optimalfitting)

- 훈　련 데이터 ➔ 오차 약간 발생
- 새로운 데이터 ➔ 오차 약간 발생 확률 높음!





ML 모델과 데이터와 학습

◆ 최적적합(Optimalfitting)

- **조건**
  - 편중되지 않은 **다양성 갖춘 데이터**로 학습 진행
  - **양질의 많은 데이터**
  - **규제(Regularization)** 통한 **모델 복잡도 적정수준** 설정

ML 모델과 데이터와 학습

◆ 모델 복잡도(Model Complexity)

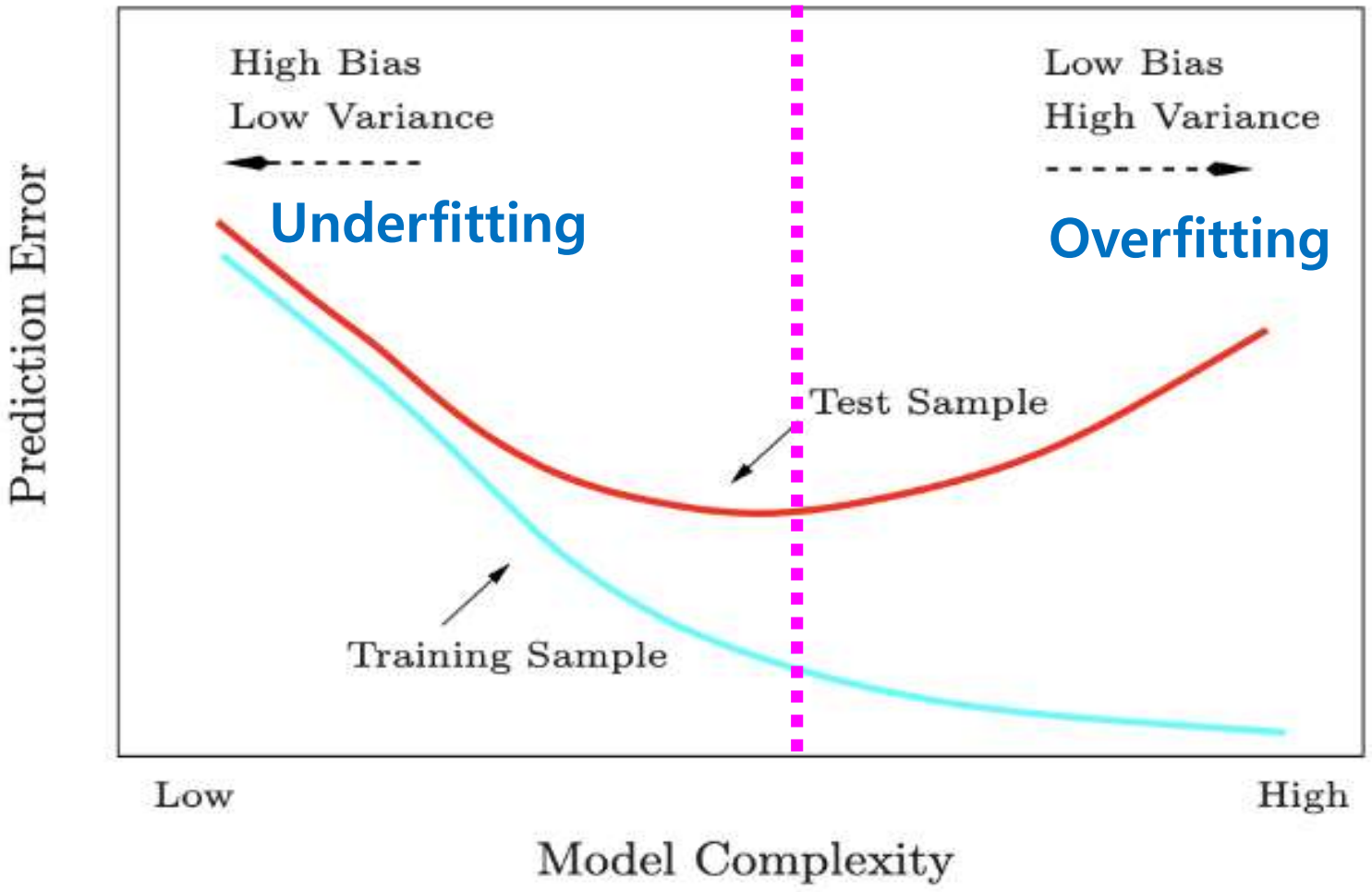
- 내부 구조가 이해하기 어려운 모델
- 모델 마다 복잡성 기준 및 결정 다름

관점에 따라 다양하게 정의

- 복잡도 고려 사항
  - 특성 파라미터 개수
  - 하이퍼파라미터 개수
  - 입력 데이터셋 개수
  - 학습 반복 횟수
  - 앙상블 모델 개수
  - 모델 차수

ML 모델과 데이터와 학습

◆ 모델 복잡도(Model Complexity)



ML 모델과 데이터와 학습

◆ 모델 복잡도(Model Complexity)

Occam's Rozor (오캄의 면도날)

같은 현상을 설명하는 두 개의 주장이 있다면, 간단한 쪽을 선택하라.

Given two equally accurate theories, choose the one that is less complex

모델 자체가 복잡	➔	**덜 복잡한 모델** 사용
데이터 부족	➔	**다양성 보장 데이터 제공**
변수가 너무 많아서 복잡	➔	**변수 개수 줄**임
학습이 너무 과해서 복잡	➔	**적당 선**에서 중단하
학습 데이터 불량	➔	**데이터 깨끗**이 정제

ML 모델과 데이터와 학습

◆ 모델 복잡도(Model Complexity)

- 복잡도 제어 방법
  - 입력 데이터 셋 늘리기 ▲ ← 시간, 비용, 어려움 | 교차검증
  - 학습 반복 횟수 조절 ← Early Stopping
  - 모델 차수 ▼
  - 특성 파라미터 개수 줄이기 ▼
  - 특성 파라미터마다 패널티(규제) 부여 즉 정칙화(Requlrization)
  - 앙상블 모델 개수

ML 모델과 데이터와 학습

◆ 모델 복잡도(Model Complexity)

- **가중치 감소(Weight Decay)**
  - 모델 학습 과정 **과도한 가중치 부여 제어**하는 방법
  - 가중치가 클 수록 더 큰 패널티 부여
    - ➔ L(Loss Function) 1 : cost function에 **가중치 절대값** 더함
    - ➔ L(Loss Function) 2 : cost function에 **가중치 제곱값** 더함
    - ➔ ElasticNet : L1+L2 합친것

ML 모델과 데이터와 학습

◆ 데이터와 학습

- 목적에 맞는 **다양성이 보장**된 **데이터** 많아야 함
- 데이터 양이 많을 수록 일반화 좋음 **단, 다양한 데이터!!**
- **편증된 데이터는 오히려 역효과**

- 학 습 데이터
- **검 증 데이터 ➔ 학습 진행 시 사용되어 학습 중단 시점 체크**
- 테스트 데이터

ML 모델과 데이터와 학습

◆ 특성공학(Feature Engineering)

- **차원 축소**

  → **특징 추출**이라고도 함

  → 과적합을 피하며 모델의 복잡도를 낮출 수 있음

- **특징 생성**

  → **특징 구축**이라고도 하며 흔히 특성공학이라고 함

  → 초기 주어진 데이터에서 모델 성능 높일 수 있는 새로운 특성 생성하는 과정

RIDGE , LASSO REGRESSION

REGRESSION 회귀

◆ 특성공학(Feature Engineering)

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.preprocessing import PolynomialFeatures
(
 degree=2,                    # 다항식의 차수
 interaction_only=False,      # 제곱값 제외 여부
 include_bias=True,           # 절편 포함
 order='C'                    # 행 우선 저장
)
```

REGRESSION 회귀

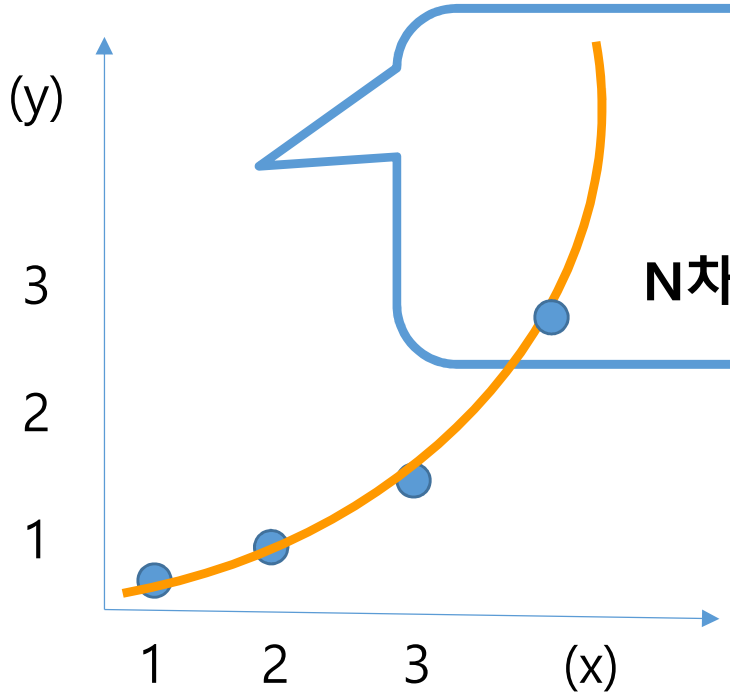
◆ POLYNOMIAL REGRESSION

- **1차 함수식으로 표현 할 수 없는** 복잡한 데이터 분포 적용
- **비선형 데이터 적용** 선형 모델
- 극단적 높은 차수 모델 구현할 경우 과적합 현상 발생

REGRESSION 회귀

◆ POLYNOMIAL REGRESSION

➢ **곡선 형태** 데이터 분포



많은 데이터 만족하는 곡선

2차 함수 $y=a_2x^2+a_1x+b$

N차 함수 $y=a_dx^d+..+a_2x^2+a_1x^1+b$

REGRESSION 회귀

◆ Ridge REGRESSION

- 기존 선형 모델에 규제항 추가한 회귀 모델
- **MSE + 패널티항 => 최소가 되는 가중치와 편향 찾는 모델**
- 훈련시킬 수록 비용함수가 작아지는 방향으로 진행
- 비용함수 : **기존 MSE + 가중치 L2 norm 추가**
  - 가중치 제곱 모두 합한 후, 규제 강도 하이퍼파라미터 alpha 추가

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

가중치 합 0에 가까워짐

REGRESSION 회귀

◆ Ridge REGRESSION

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import Ridge
( alpha=1.0,              # 규제 강도 설정, 클수록 회귀계수 작아짐, 양수
 fit_intercept=True     ,  #  절편 상수 사용 여부 설정
 normalize               # 매개변수 무시 여부
 copy_X                  # X의 복사 여부
 max_iter                # 계산 작업 수
 tol                     # 정밀도
 solver                  # 계산 알고리즘 (auto, svd, cholesky, lsqr, sparse_cg, sag, saga)
random_state             # 난수 seed 설정  )
```

REGRESSION 회귀

◆ Lasso REGRESSION

- 기존 선형 모델에 규제항 추가한 회귀 모델
- **MSE + 패널티항 => 최소가 되는 가중치와 편향 찾는 모델**
- 훈련시킬 수록 비용함수가 작아지는 방향으로 진행
- 중요도 낮은 가중치 제거함 ➔ **중요한 특성/가중치 선택 훈련**
- 비용함수 : **기존 MSE + 가중치 L1 norm 추가 (가중치 절대값 합)**

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

가중치 합 0

REGRESSION 회귀

◆ Ridge REGRESSION

➢ Scikit-Learn Lib 사용 - 학습객체

```
from sklearn.linear_model import Lasso
( alpha=1.0,              # 규제 강도 설정, 클수록 회귀계수 작아짐, 양수
 fit_intercept=True     ,  #  절편 상수 사용 여부 설정
 normalize               # 매개변수 무시 여부
 copy_X                    # X의 복사 여부
 max_iter                  # 계산 작업 수
 tol                       # 정밀도
warm_start                 # 이전 모델을 초기화로 적합하게 사용할 것인지 여부
positive                   # 계수가 양수로 사용할 것인지 여부
 solver                    # 계산 알고리즘 (auto, svd, cholesky, lsqr, sparse_cg, sag, saga)
random_state               # 난수 seed 설정
selection                  # 계수의 업데이트 방법 설정 )
```

REGRESSION 회귀

◆ Elastic Net REGRESSION

- Ridge와 Lasso를 합친 선형 모델
- Ridge와 Lasso의 최적화 지점이 다르므로 두 개 값 합쳐서 규제
- 비용함수 : 기존 MSE + L2 norm + L1 norm

$$J(\theta) = \mathrm{MSE}(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \theta_i^2$$

SGD OPTIMIZER

SGD OPTIMIZER

◆ SGD 최적화

- **확률적 경사하강법(SGD, Stochastic Gradient Descent)을 이용하여 구현하는 모델들**
- 분류와 회귀 모두에 적용

SGD OPTIMIZER

◆ SGD 최적화

- **SGDClassifer**
  - 이진분류 : 로지스틱회귀 , 이진 크로스엔트로피 손실함수
  - 다중분류 : 크로스엔트로피 손실함수
- **SGDRegressor**
  - 평균제곱근오차 손실함수 사용

SGD OPTIMIZER

◆ SGDClassifier

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import SGDClassifier
(loss='hinge'          : 손실함수 설정
 penalty= ' l2'        : 규제 방법 설정
alpha=0.0001           : 규제 값 설정 ( 클수록 강력한 규제)
l1_ratio=0.15          : L1규제 비율
fit_intercept=True     : 절편 사용 여부 설정
max_iter=1000          : 학습 횟수 설정
tol=0.001              : 정밀도
```

SGD OPTIMIZER

◆ SGDClassifier

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import SGDClassifier
( shuffle=True                  : 에포크 후 데이터 섞는 유무
 verbose=0                      : 설명 출력 여부
 epsilon=0.1                    : 손실 함수에서 사용되는 값
 n_jobs=None                    : 병렬처리 설정 (CPU 수)
 random_state=None              : 난수 설정
 learning_rate= ' optimal'      : 학습 속도
 eta0=0.0                       : 초기 학습속도
```


SGD OPTIMIZER

◆ SGDClassifier

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import SGDClassifier
( power_t=0.5                    : 역 스케일링 학습률
  early_stopping=False           : 조기 중지 여부 설정
  validation_fraction=0.1        : 조기 중지 위한 검증 셋 비율
  n_iter_no_change=5             : 조기 중비 전 반복횟수
  class_weight=None              : 클래스별 가중치
  warm_start=False               : 초기화 유무
  average=False                  : True -> 모든 업데이트에 대한 평균 SGD 가중치
                                   계산 후 결과를 coef_속성에 저장
)
```

SGD OPTIMIZER

◆ SGDClassifier

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import SGDClassifier
( power_t=0.5                    : 역 스케일링 학습률
  early_stopping=False           : 조기 중지 여부 설정
  validation_fraction=0.1        : 조기 중지 위한 검증 셋 비율
  n_iter_no_change=5             : 조기 중비 전 반복횟수
  class_weight=None              : 클래스별 가중치
  warm_start=False               : 초기화 유무
  average=False                  : True -> 모든 업데이트에 대한 평균 SGD 가중치
                                   계산 후 결과를 coef_속성에 저장
)
```

SUPPORT VECTOR MACHINE

SVM MODEL

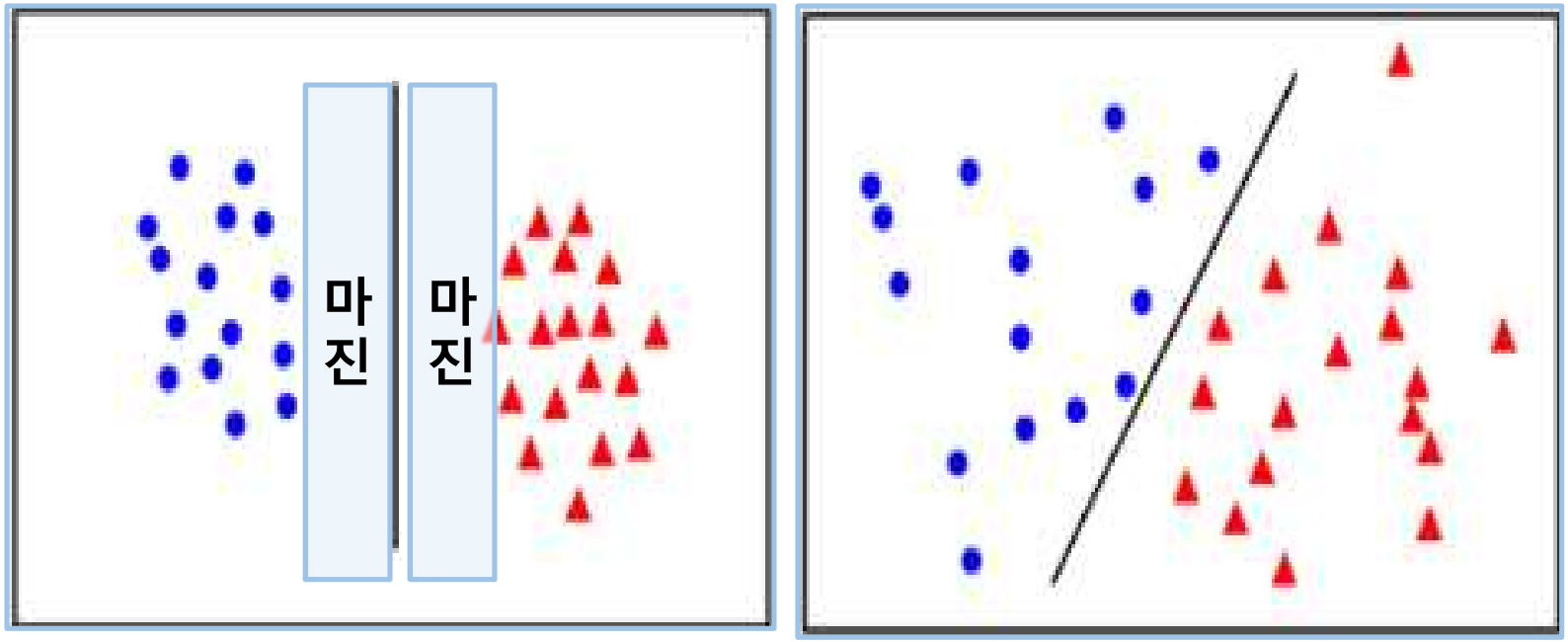
◆ SVM(Support Vector Machine)

- 데이터를 분류해주는 기술
- 활발한 연구로 **높은 인식 성능**
- **구분선**을 기준으로 분류
- 분류시 **최고의 마진을 가지는 구분선 구하는** 분류
- 분류와 회귀에 사용이 되는 기술로 **SVC, SVR**이라고 부름
- Tree기반과 ML에서 가장 인기 있는 모델

SVM MODEL

◆ SVM(Support Vector Machine)

➢ 선형적 분류

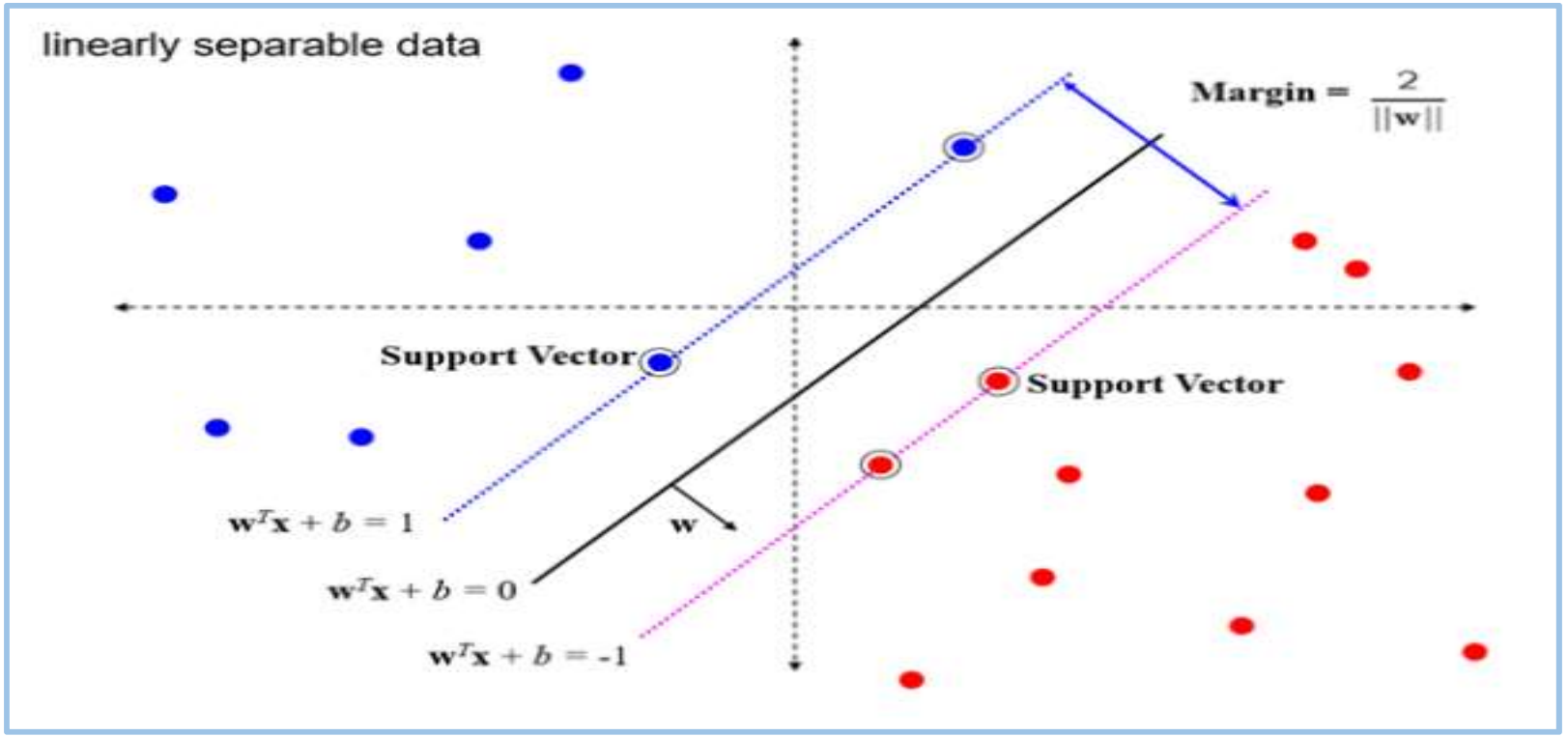


마진이 클수록 학습되지 않은 **새로운 데이터에 대한 분류/판단이 좋음**

SVM MODEL

◆ SVM(Support Vector Machine)

➢ 선형적 분류

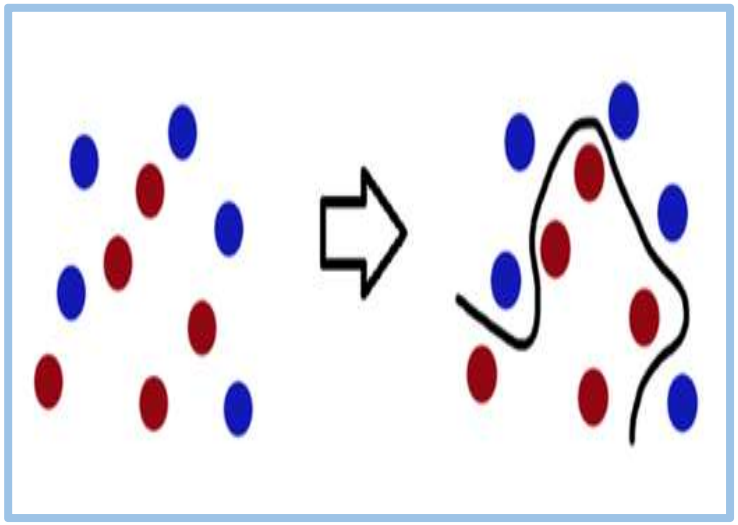


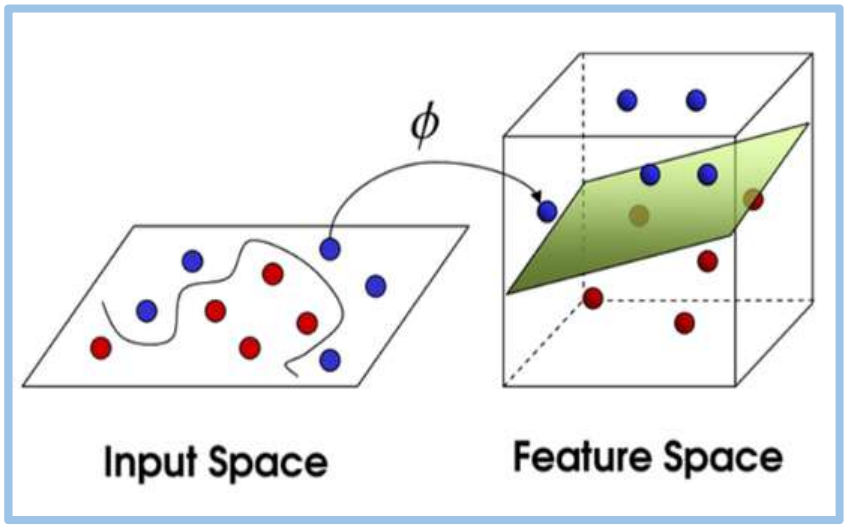
Black선을 3차원으로 볼 경우 평면 ➔ **Hyper-Plane(초평면)**

SVM MODEL

◆ SVM(Support Vector Machine)

➢ 비선형적 분류





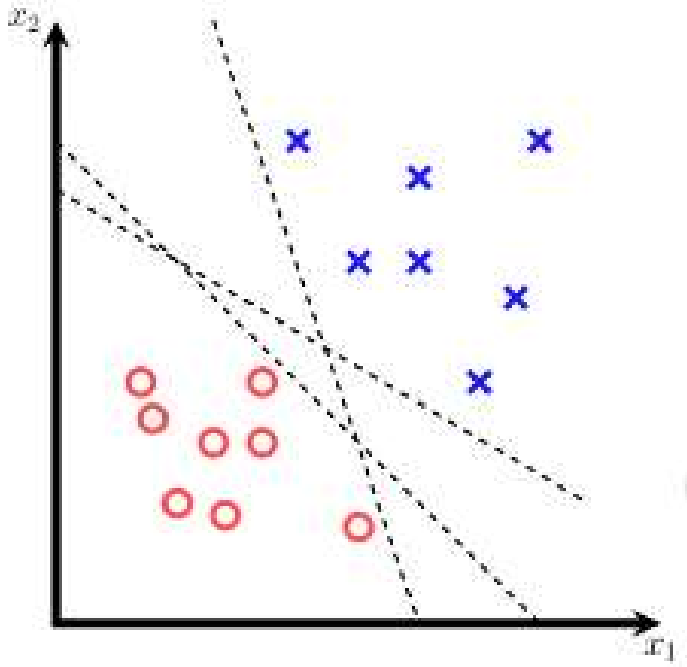
공간 상에 구별 가능한 방향으로 매핑시켜 구분

Kernel Tick(Kernel function)

SVM MODEL

◆ SVM(Support Vector Machine)

➢ 분류 규칙(Decision Rule)



$$(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_N, y_N)$$

$$y = \begin{cases} +1 \\ -1 \end{cases}$$

$$y = w^T x - w_0 = \sum_{n=1}^{N} a_n y_n x_n^T x - w_0$$

SVM MODEL

◆ SVM(Support Vector Machine)

➢ 일반화(Generalization)

- 과적합 문제 해결 위한 방법
- 데이터 분류 후 마진(Margin)이 최대가 되는 구분선 설정

➢ 소프트 마진(Soft margin)

- 하나의 선으로 완벽히 분류 => 과적합 발생 & 마진 최소화
- **분류 오류 허용하는 방식**

SVM MODEL

◆ SVM(Support Vector Machine)

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.svm import SVC
( C=1.0                규제 강도 ( Cost Funciton ) , 큰값(약) - 작은값(강), squred L2
  kernel='rbf'         커널 지정 ('linear', 'poly', 'rbf', 'sigmoid', 'precomputed'
  degree               다항식 커벌함수 차수 지정
  gamma=scale          'rbf', 'poly' 및 'sigmoid'에 대한 커널 계수
  coef0=0.0                    절편 존재 여부 설정
  intercept_scaling=1          정규화 효과 정도
  class_weight =1              클래스 가중치
  random_state                 난수 seed 설정
```

SVM MODEL

◆ SVM CLASSIFIER

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.svm import SVC
(
solver='lbfgs'      # 최적화 문제 사용 알고리즘
max_iter            # 학습 횟수
multi_class         # 다중 분류 시 (ovr, multinomial, auto)로 설정
verbose             # 동작 과정에 대한 출력 메시지
warm_start          # 이전 모델을 초기화로 적합하게 사용할 것인지 여부
n_jobs              # 병렬 처리 할 때 사용되는 CPU 코어 수
l1_ratio            # L1 규제의 비율(Elastic-Net에만 사용)  )
```

DECISION TREE

DECISION TREE

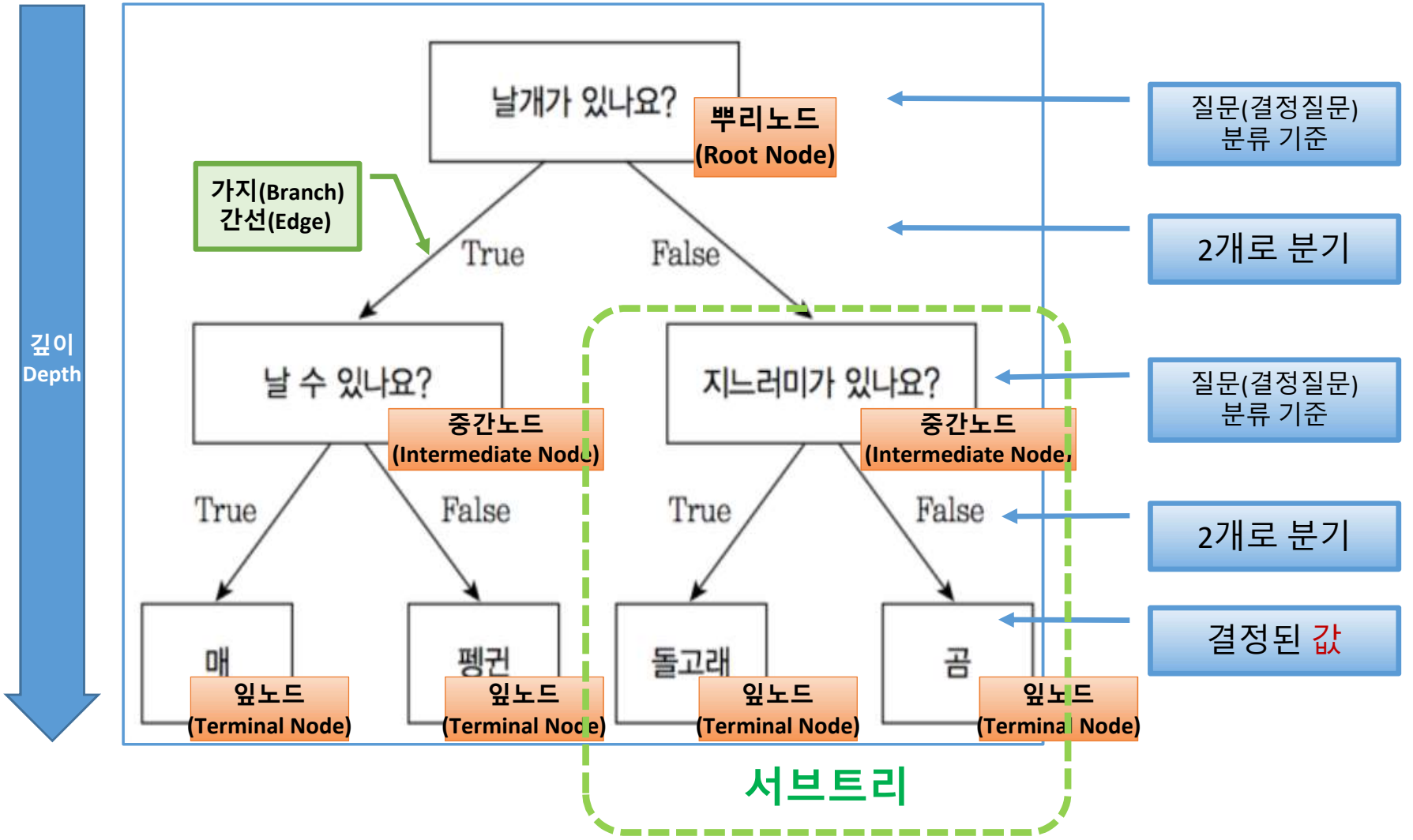
◆ 결정 트리

- **예/아니오 질문(특정 기준)**으로 **학습** 진행 ➔ 스무고개 퀴즈
- **질문(특정 기준)**을 무엇으로 하느냐가 **성능 크게 좌우**
- **질문(특정 기준)**에 따라 **데이터 구분하는 모델** ➔ 결정 트리 모델
- **직관적, 범용성, 해석력이 좋**지만 **데이터에 민감**함
- **데이터 사전 가공**에 대한 **영향이 매우 적음**
- **분류**와 **회귀 모두 가능**한 **지도 학습 모델** 중 하나

  ➔ CART(Classification And Regression Tree)라고도 함

DECISION TREE

◆ 결정 트리



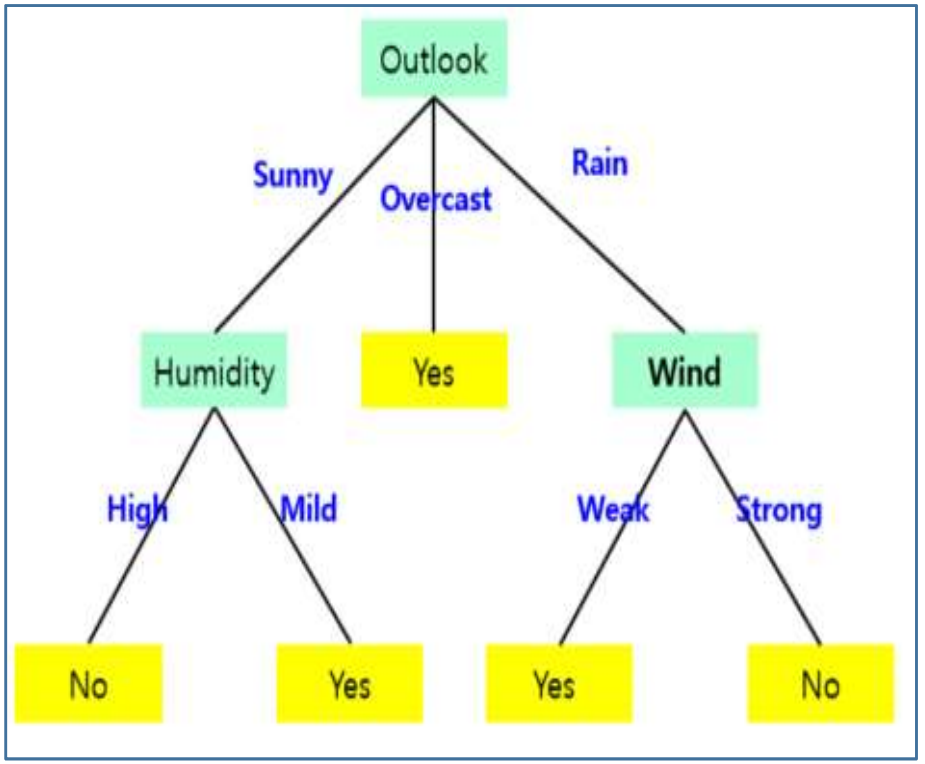
BY ANECE SO

DECISION TREE

◆ 결정 트리

- ➢ 범주형 타입 입력 & 출력

속성/특성/피쳐					타겟
Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

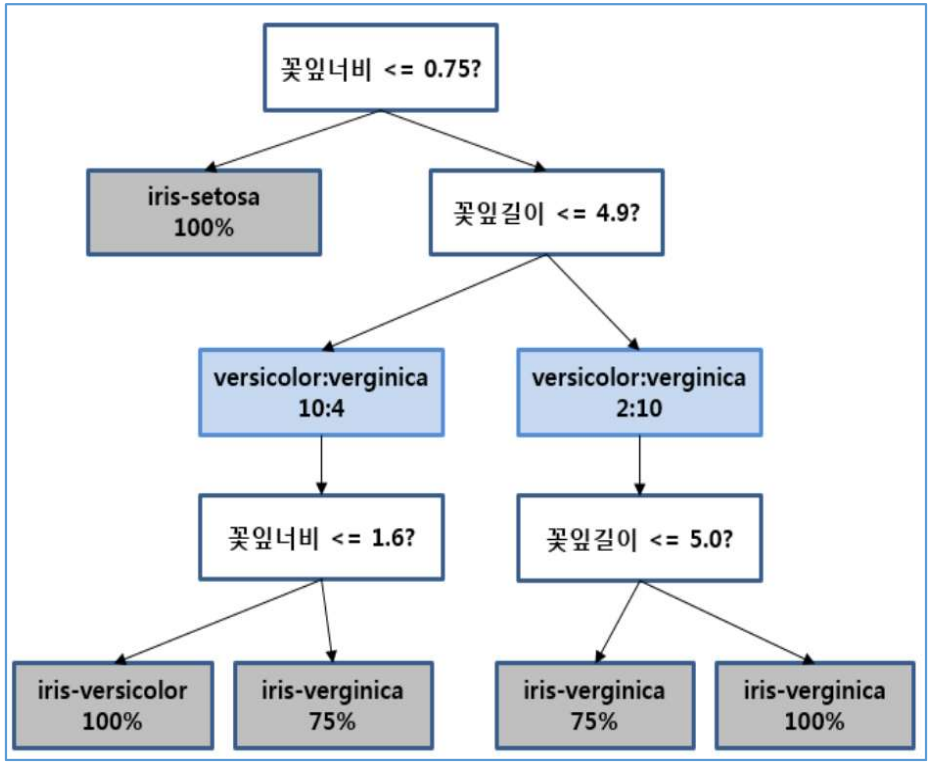


DECISION TREE

◆ 결정 트리

➢ 수치형 타입 입력&출력

속성/특성/피쳐				타겟
SepalLength	SepalWidth	PetalLength	PetalWidth	Name
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3.0	1.4	0.1	Iris-setosa



DECISION TREE

◆ 결정 트리

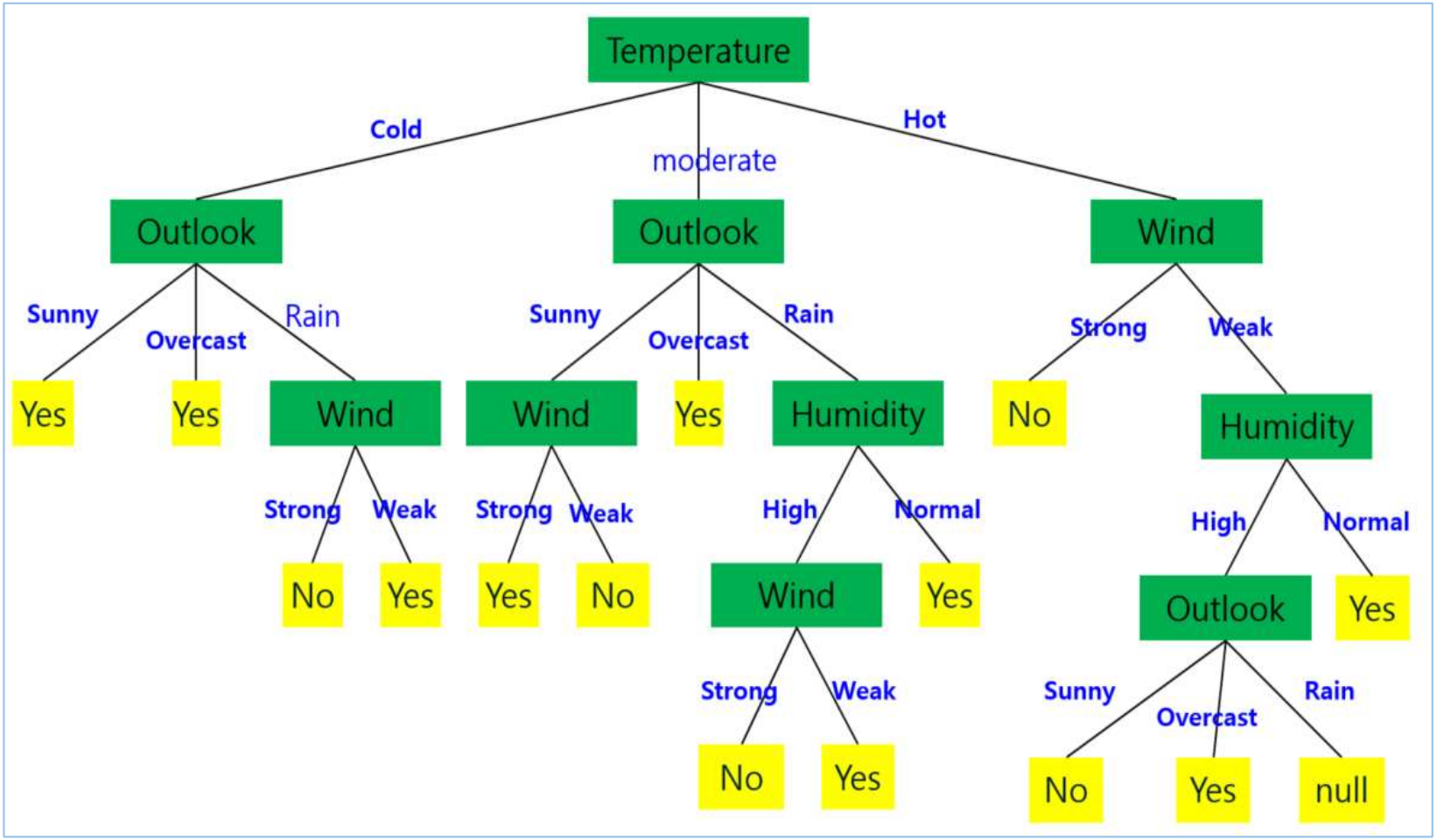
➢ 동작 알고리즘

- 모든 데이터 포함한 **하나의 노드(Root Node)로 구성된 트리에서 시작**
- **반복적 노드 분할 과정**
  - **분할 속성(splitting attribute)**을 선택
  - 속성값에 따라 **서브 트리(subtree)를 생성**
  - 데이터를 속성값에 따라 **분배**

DECISION TREE

◆ 결정 트리

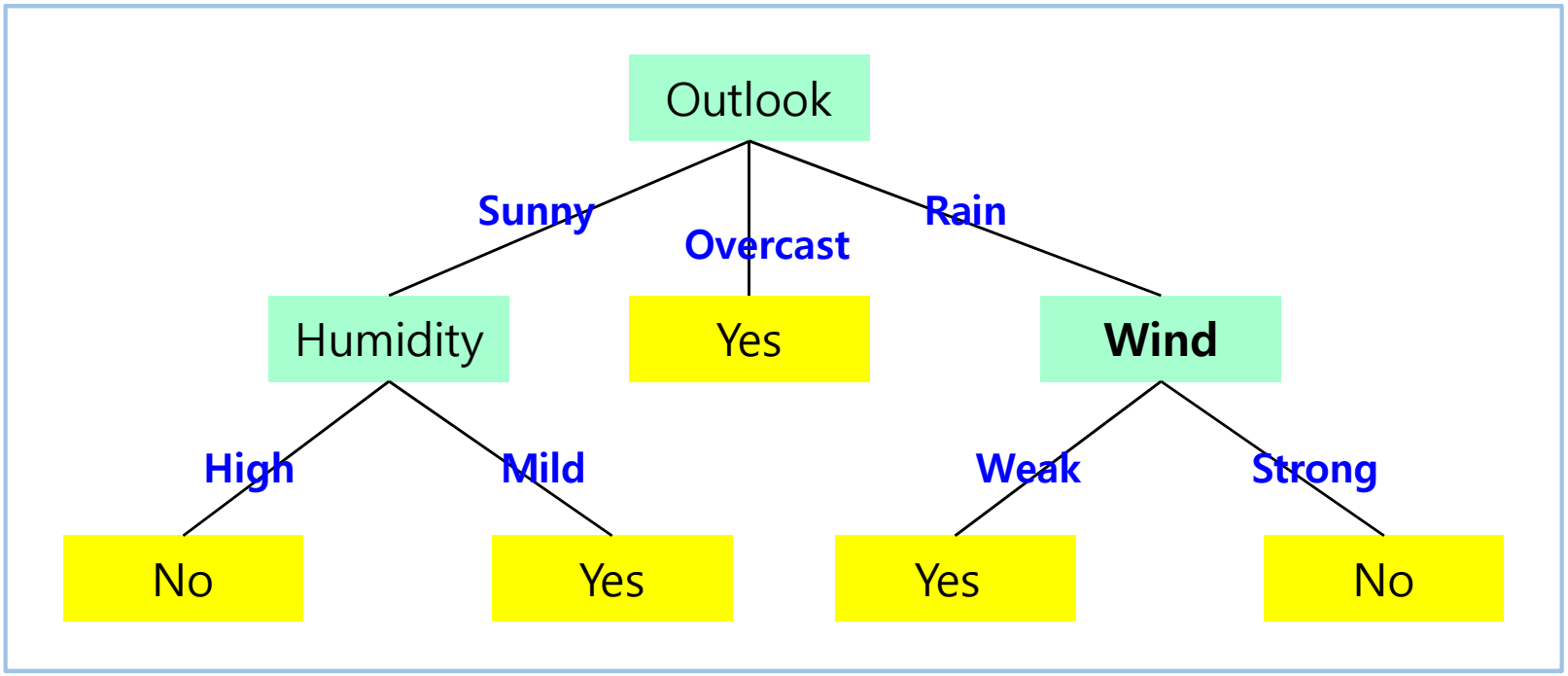
➢ 동일 문제 - 분할 속성에 따른 복잡한 트리



DECISION TREE

◆ 결정 트리

➢ 동일 문제 - 분할 속성에 따른 간단한 트리



DECISION TREE

◆ 결정 트리

➢ 분할 속성(Spliting Attribute) 결정

- 결정 트리 구성 및 성능에 가장 큰 영향
- 속성 결정 기준

  ➔ 분할 후 **가능한 많은 동일 분류의 데이터가 모이는 속성**

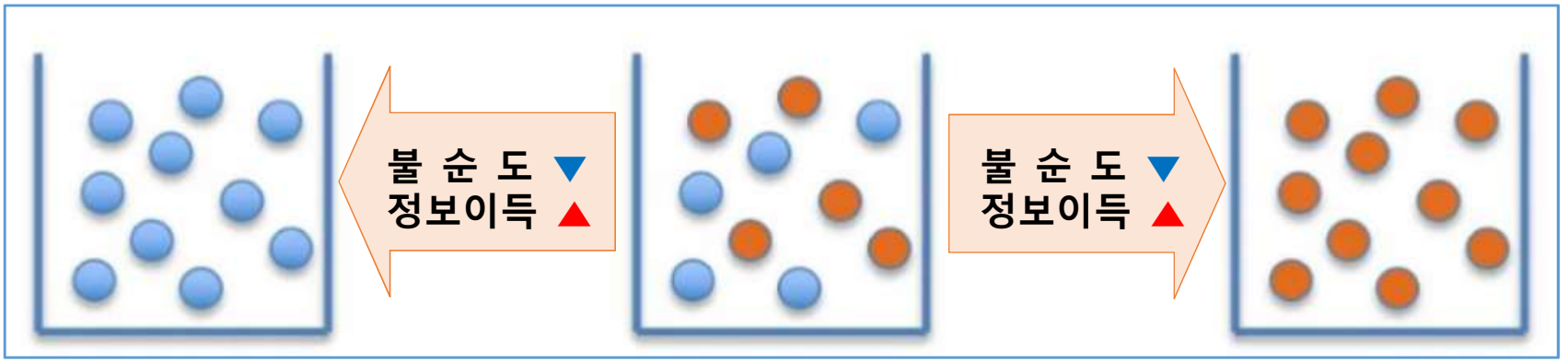
  ➔ 분할 후 동일 분류 데이터 모이는 정도 측정 필요

DECISION TREE

◆ 결정 트리

➢ 분할 속성(Spliting Attribute) 결정

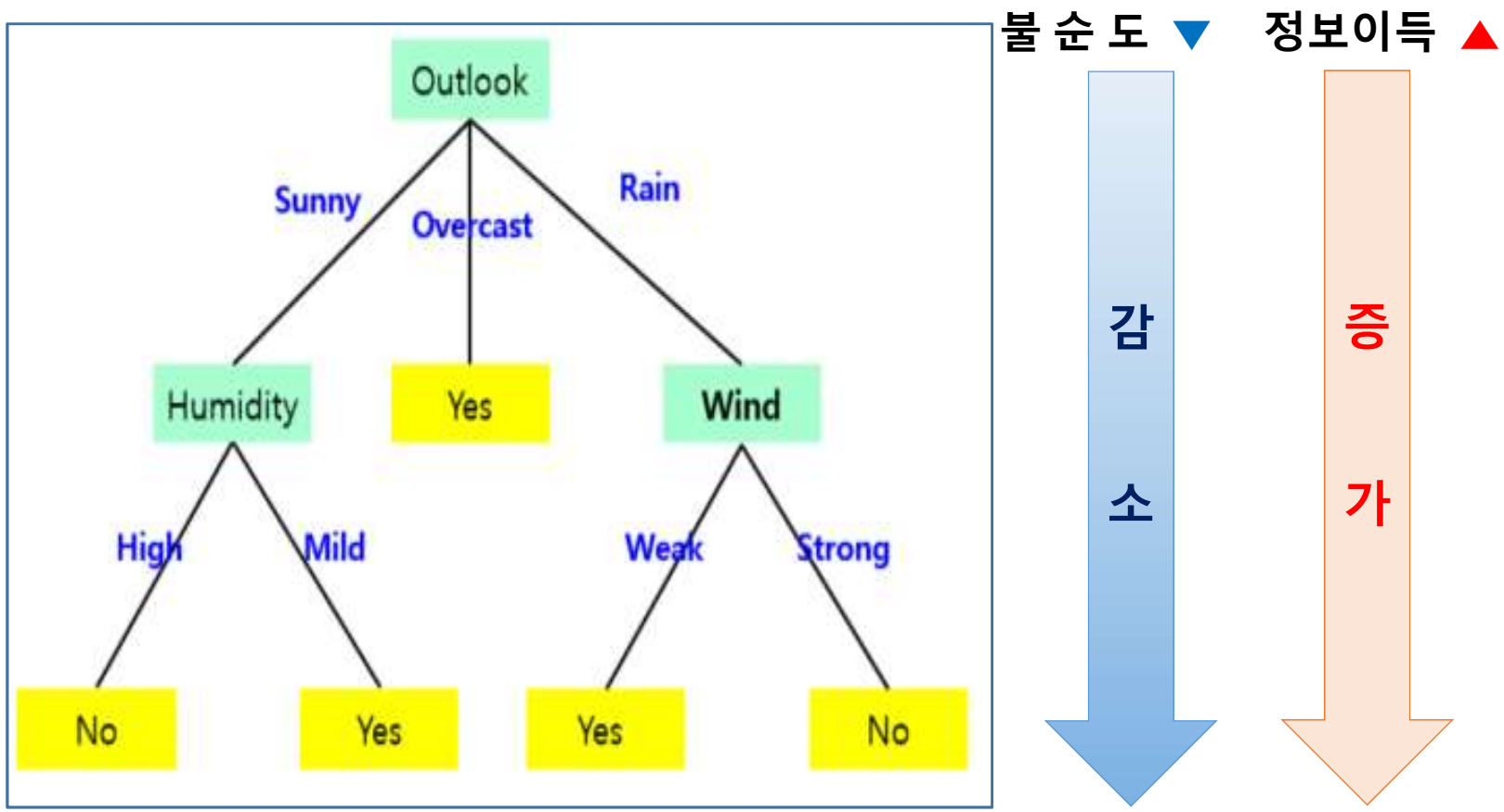
- **고려 사항**
  - **불순도(Impurity)** : 서로 다른 데이터가 얼마나 섞여 있는지 의미
  - **정보 이득(IG)** : 분할 후 불순도의 차이



DECISION TREE

◆ 결정 트리

➢ 분할 속성(Spliting Attribute) 결정



DECISION TREE

◆ 결정 트리

➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)

정보이득(IG) 측정값으로 불순도를 수치화

값의 범위 : 0 ~ log(nc) **0에 가까울수록 좋음**

- 9 □ (사각형)
- 5 Δ (삼각형)
- 분류별 확률(Class Probability)

$$p(\square) = \frac{9}{14} \quad p(\Delta) = \frac{5}{14}$$

- 엔트로피 $I = -\sum_{c} p(c) \log_2 p(c)$

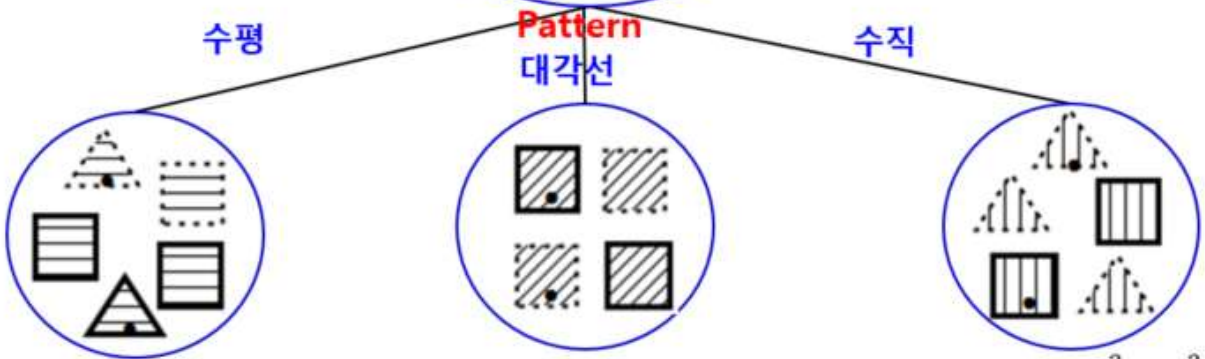
$$I = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940 \text{ bits}$$

DECISION TREE

◆ 결정 트리

➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)

$$I = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$



$$I_{horizontal} = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971 \quad I_{diagonal} = -\frac{0}{4}\log_2\frac{0}{4} - \frac{4}{4}\log_2\frac{4}{4} = 0.0 \quad I_{vertical} = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971$$

$$I_{res}(Pattern) = \sum p(v)I(v) = \frac{5}{14}\cdot 0.971 + \frac{4}{14}\cdot 0 + \frac{5}{14}\cdot 0.971 = 0.694$$

$$IG(Pattern) = I - I_{res}(Pattern) = 0.940 - 0.694 = 0.246$$

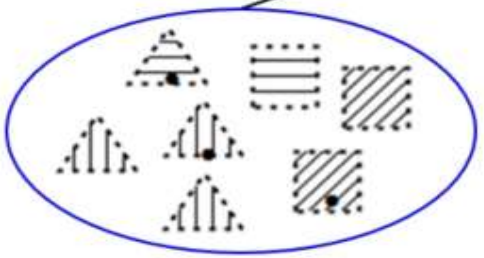
DECISION TREE

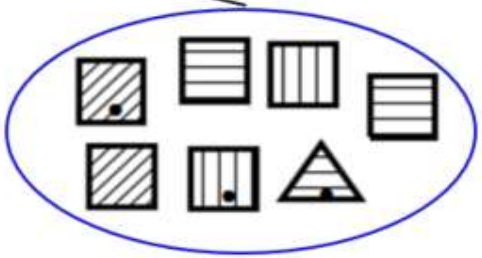
◆ 결정 트리

➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)

$$I = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

점선 Outline 실선





$$I_{dashed} = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985$$

$$I_{solid} = -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} = 0.592$$

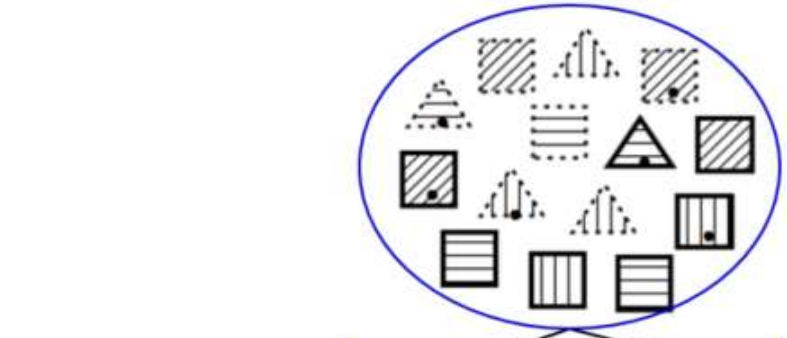
$$I_{res}(Outline) = \sum p(v)I(v) = \frac{7}{14}\cdot 0.985 + \frac{7}{14}\cdot 0.592 = 0.789$$

$$IG(Outline) = I - I_{res}(Outline) = 0.940 - 0.789 = 0.151$$

DECISION TREE

◆ 결정 트리

➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)



$$I = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

무 Dot 유

$$I_{no} = -\frac{2}{8}\log_2\frac{2}{8} - \frac{6}{8}\log_2\frac{6}{8} = 0.811$$

$$I_{yes} = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1.000$$

$$I_{res}(Dot) = \sum p(v)I(v) = \frac{8}{14}\cdot 0.811 + \frac{6}{14}\cdot 1.000 = 0.892$$

$$IG(Dot) = I - I_{res}(Dot) = 0.940 - 0.892 = 0.048$$

DECISION TREE

◆ 결정 트리

➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)

- 속성별 정보 이득

  IG(Pattern) = **0.246**

  IG(Outline) = 0.151

  IG(Dot) = 0.048

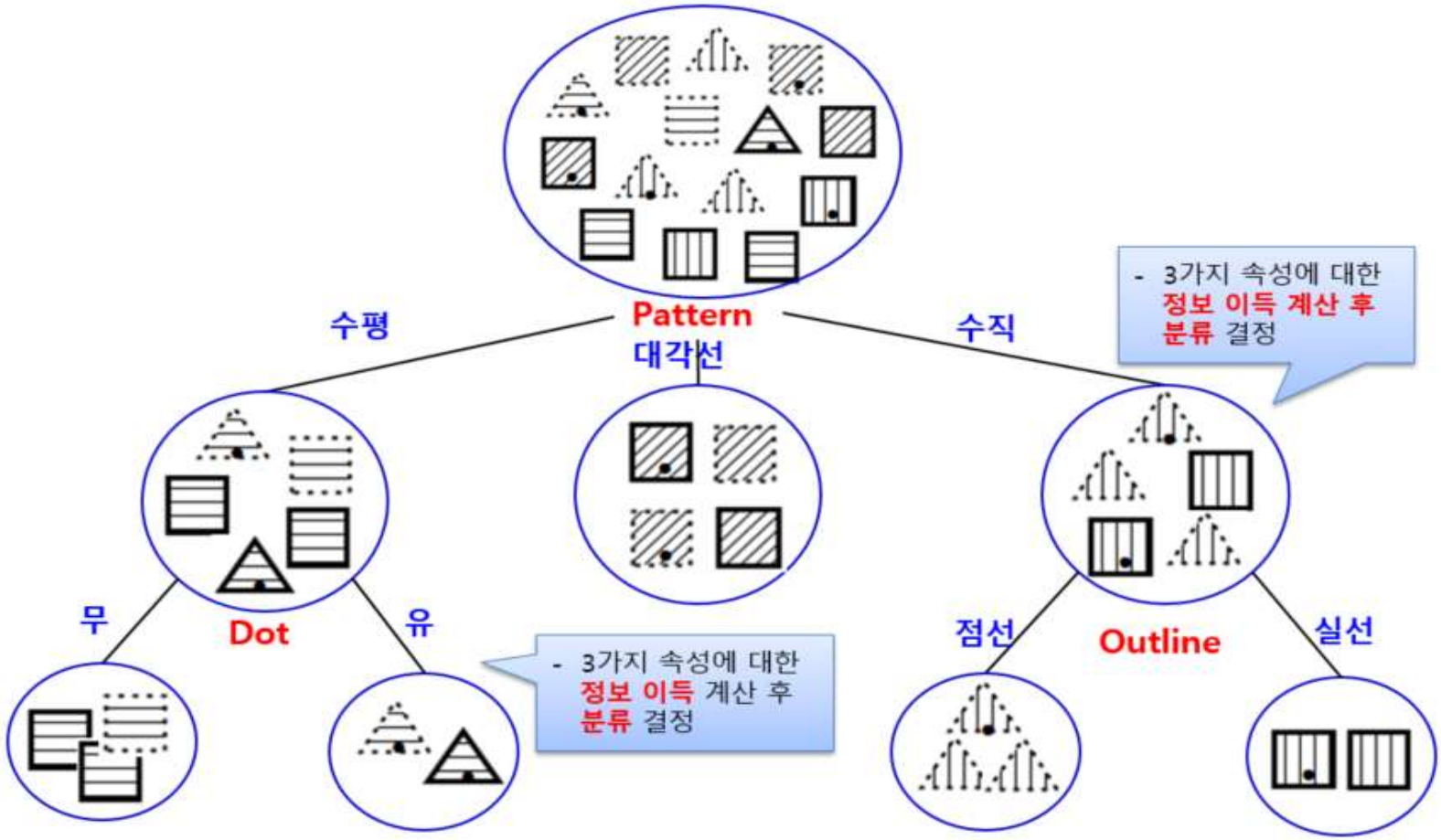
분할속성 선택

정보이득이 큰 것 선택 **"Pattern" 선택**

DECISION TREE

◆ 결정 트리

- ➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)



DECISION TREE

◆ 결정 트리

- ➢ 불순도(Impurity) 수치화 - 엔트로피(Entropy)

Entropy = 1

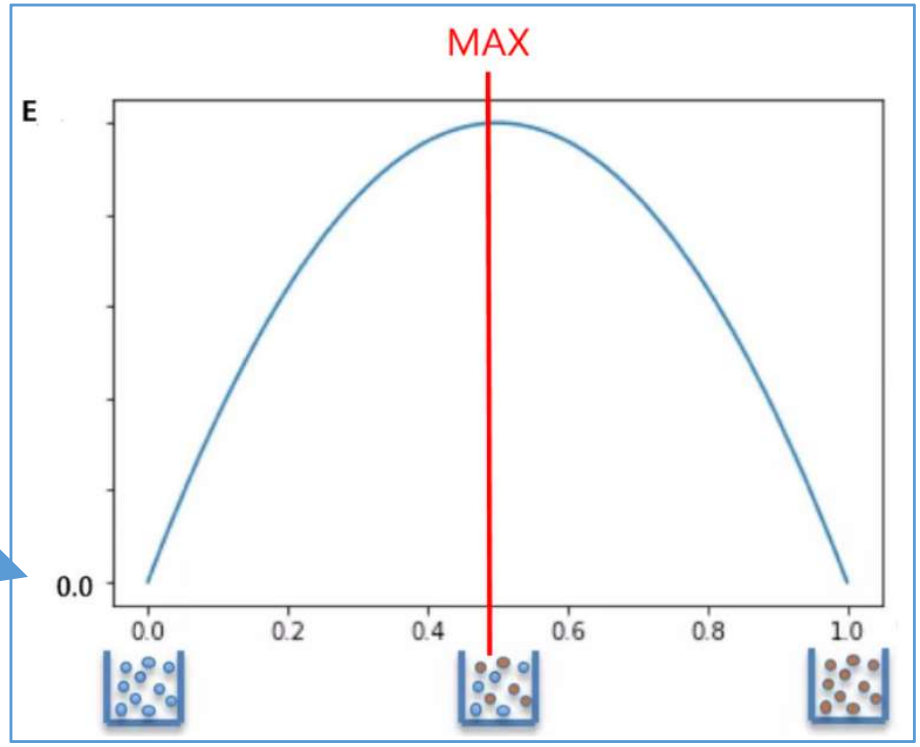
반반 있는 상태

분류 어려움

Entropy = 0

한 종류만 존재

분류하기 쉬움



DECISION TREE

◆ 결정 트리

➢ 불순도(Impurity) 수치화 - 지니(Gini) 계수

데이터의 통계적 분산정도 즉 균일도로 불순도를 수치화 함

$$Gini = \sum_{i \neq j} p(i)p(j)$$

$$p(\square) = \frac{9}{14}$$

$$p(\triangle) = \frac{5}{14}$$

$$Gini = \frac{9}{14} \times \frac{5}{14} = 0.230$$

DECISION TREE

◆ 결정 트리

➢ 가지치기(Pruning)

- **과대적합(Overfitting)을 방지**하기 위한 방법
- 트리에 **가지가 지나치게 많을 때** 나타남
- 최대 깊이나 Terminal Node의 최대 개수 제한
  - ➢ **min_sample_split :** 한 노드에 최소 데이터 수
    최소 데이터 수 아래로 분할하지 않음
  - ➢ **max_depth :** 최대 깊이 조정

DECISION TREE

◆ 결정 트리

➢ Scikit-Learn LIB

sklearn.tree.**DecisionTreeClassifier**

(*,

criterion='gini'	**: 분할 품질을 측정하는 기능**
splitter='best'	: 각 노드 분할 선택 방법 설정
max_depth=None	: 트리 최대 깊이 (값이 클수록 모델 복잡도 ▲)
min_samples_split=2	: 자식 노드 분할 위한 최소 샘플 수
min_samples_leaf=1	: 리프 노드에 있어야 할 최소 샘플 수
min_weight_fraction_leaf=0.0	: 가중치가 부여된 샘플 수에서의 비율
max_features=None	: 각 노드 분할에 사용할 특징 최대 수

DECISION TREE

◆ 결정 트리

- Scikit-Learn LIB

```
sklearn.tree.DecisionTreeClassifier
(
        random_state=None                 : 난수 seed 설정
        max_leaf_nodes=None               : 리프 노드의 최대수
        min_impurity_decrease=0.0         : 최소 불순도
        class_weight=None                 : 클래스 가중치
        ccp_alpha=0.0
)
```


ENSEMBLE

앙상블(ENSEMBLE)

◆ 앙상블 학습 / 방법

- **여러 모델 또는 동일한 모델 여러 개 결합하여 정확도 높은 모델을 만드는 학습 방법**
  - Estermotor : 학습 완료된 모델, 추정기

- **핵심원리 ➔ 모델(Estermotor 추정기)의 편향과 분산 줄이기**
  - 편향(Bias)
    - 예측값과 정답이 떨어져 있는 정도(오차 Error)
    - 편향이 크면 '과소적합'
  - 분산(Variance)
    - 예측값들의 흩어져있는 분포 정도

앙상블(ENSEMBLE)

◆ 앙상블 학습 / 방법

- **모델 및 샘플 결합 방법에 따른 분류**
  - ➢ 보팅(Voting 투표) 기반 앙상블
  - ➢ 배깅(Bagging, Bootstrap Aggregating) 기반 앙상블
  - ➢ 부스팅(Boosting)기반 앙상블

- **기법에 따른 분류**
  - ➢ 평균화 기법 : 배깅, 랜덤포레드스
  - ➢ 순차적 기법 : 에이다부스트, 그레디언트 부스팅

앙상블(ENSEMBLE)

◆ 앙상블 학습 / 방법

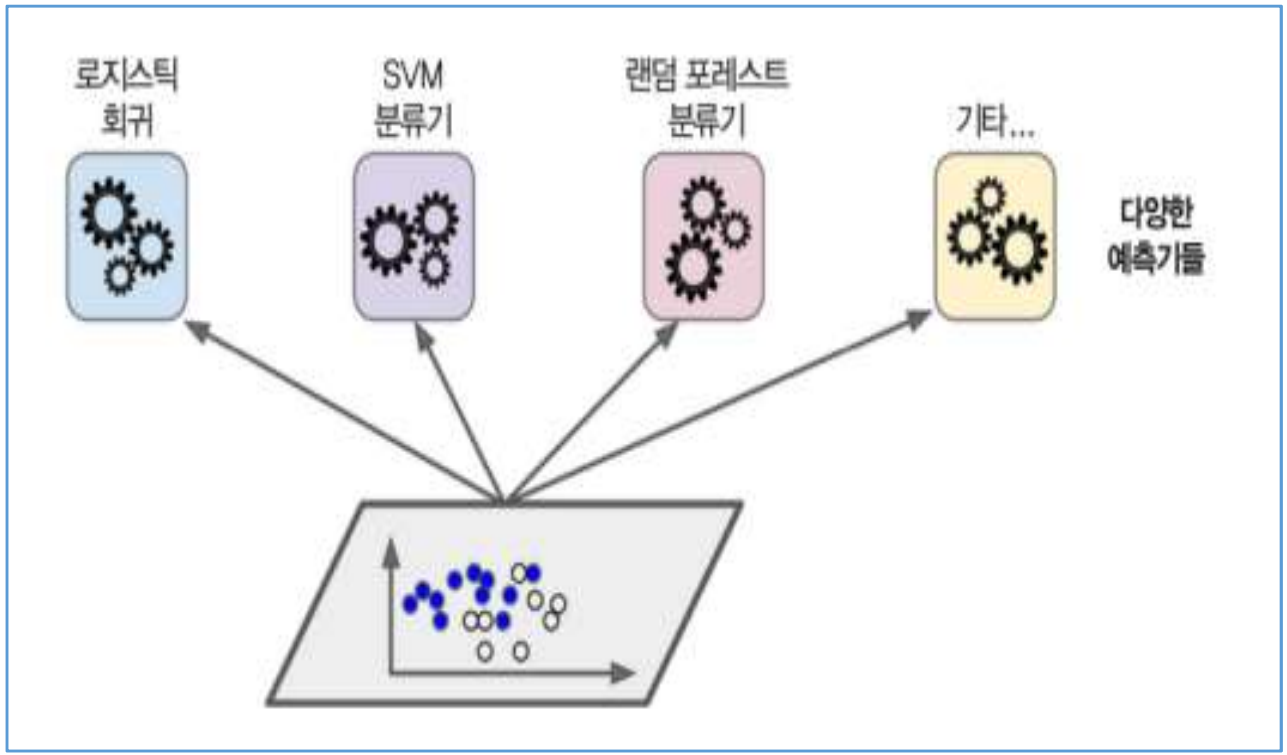
- **DecisionTree를 많이 사용하는 이유**
  - ➢ 가만히 두면 성능이 최대가 됨 즉 과대적합
  - ➢ 여러 개의 Tree를 사용해서 과대적합을 줄이는 방식으로 사용

앙상블(ENSEMBLE)

◆ 보팅(Voting)

매우 단순한 개념 및 형태이지만 강력한 성능을 가짐

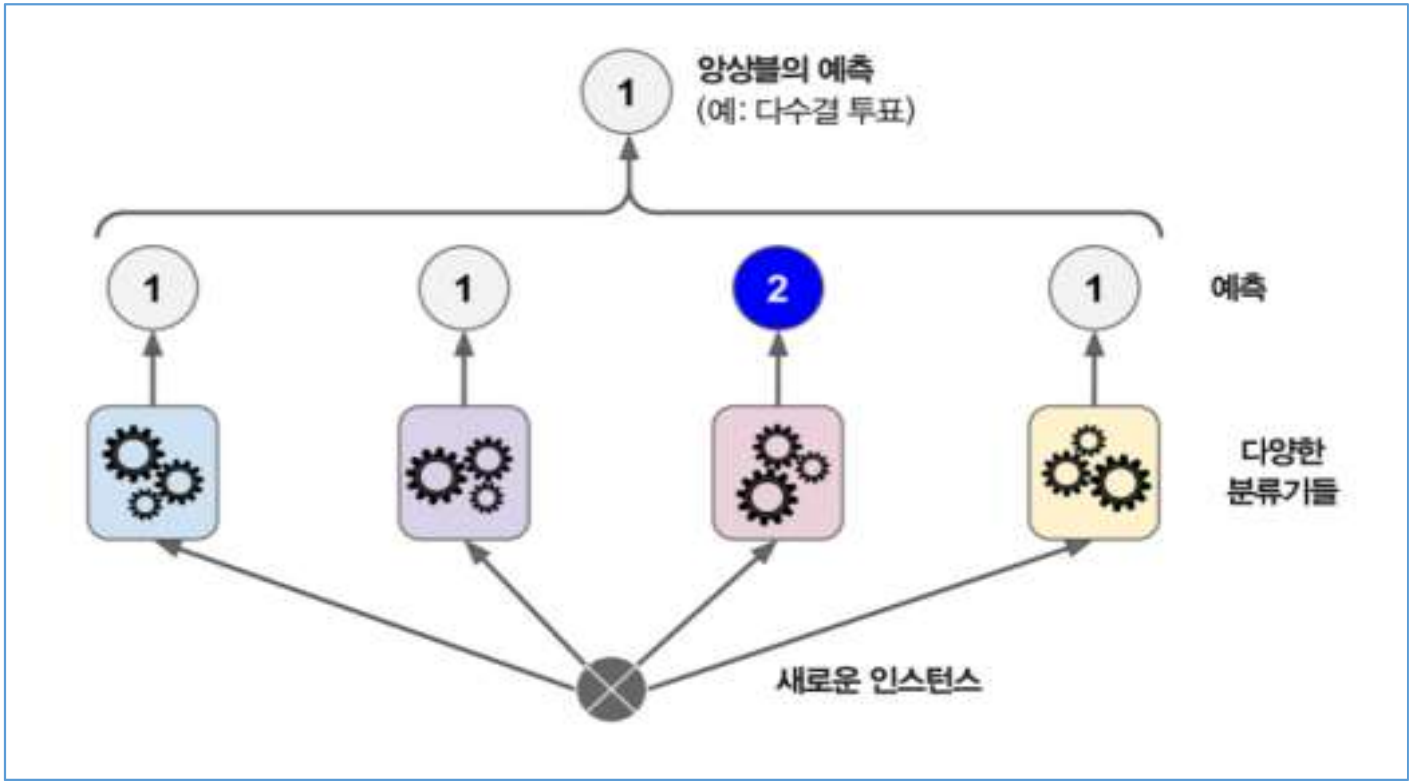
동일 훈련데이터로 여러 알고리즘의 여러 모델(추정기) 병렬 학습 진행



앙상블(ENSEMBLE)

◆ 보팅(Voting)

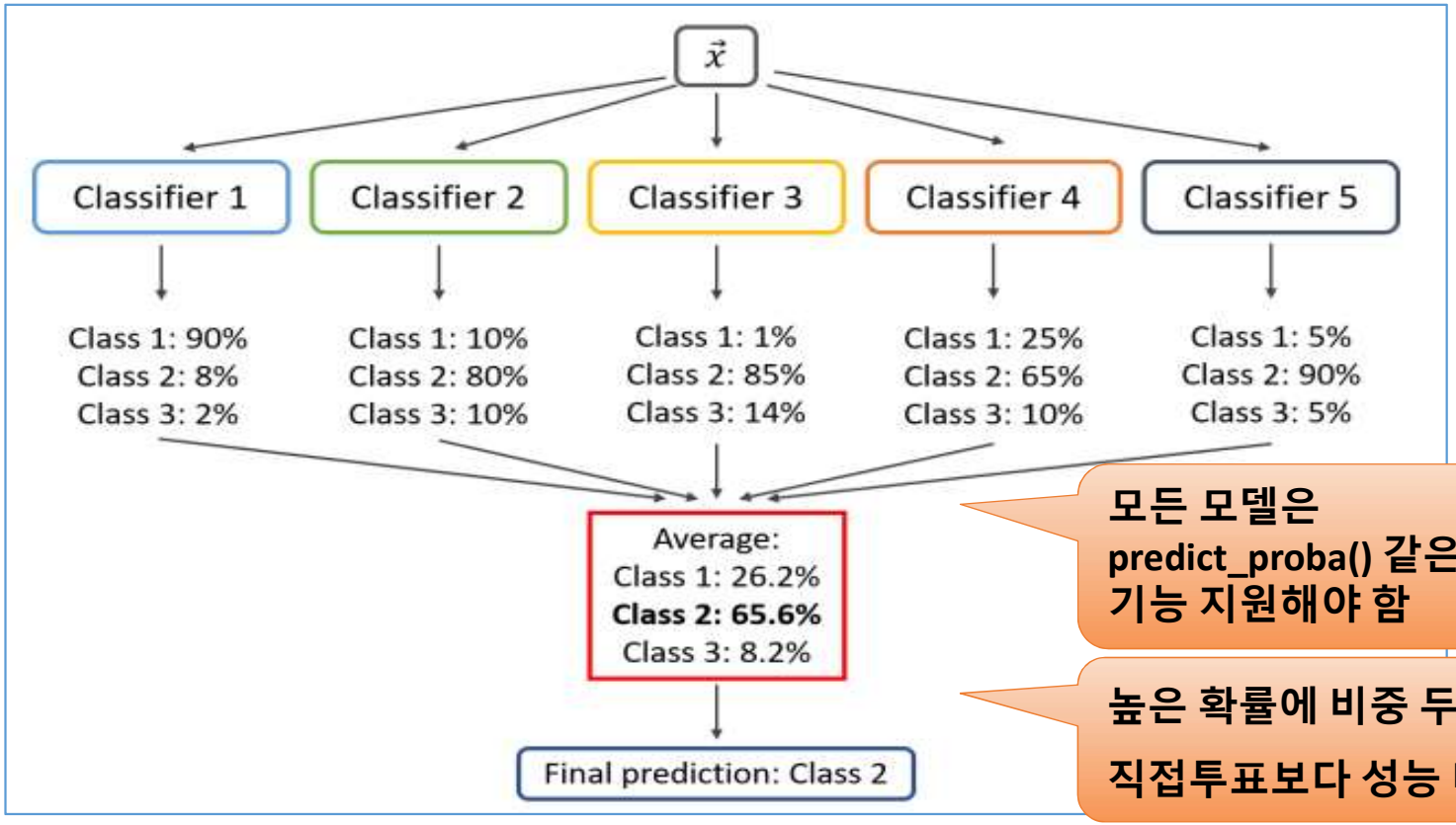
직접 투표 방식 ➔ 여러 모델의 예측값들의 다수로 결정



앙상블(ENSEMBLE)

◆ 보팅(Voting)

간접 투표 방식 ➔ 여러 모델 예측한 확률값들의 평균값으로 예측값 결정



모든 모델은 predict_proba() 같은 확률 예측 기능 지원해야 함

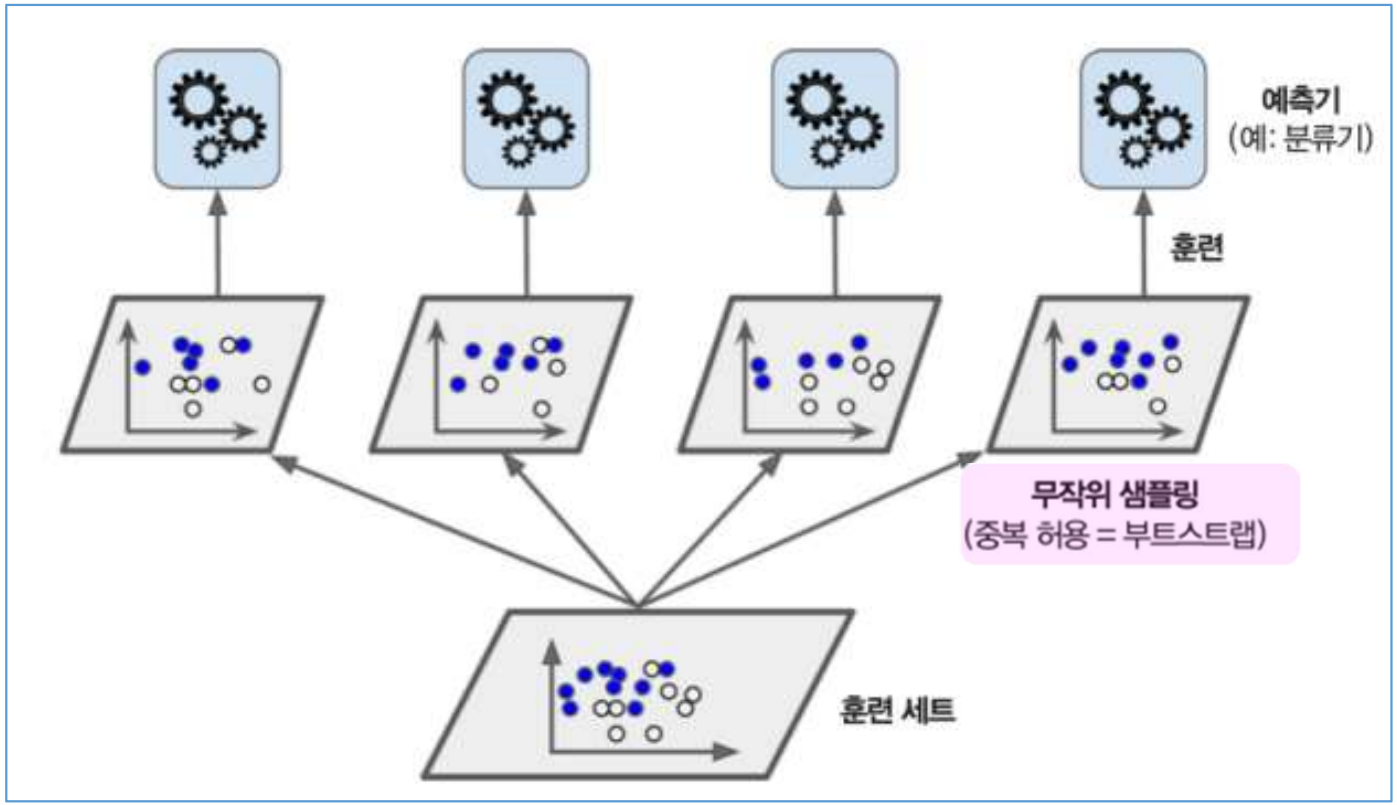
높은 확률에 비중 두어 직접투표보다 성능 더 좋음!

앙상블(ENSEMBLE)

◆ 배깅(Bagging, Bootstrap Aggregation)

중복 허용 샘플링하는 방식 의미하며 학습 데이터 부족 해결

중복 샘플링 다른 데이터와 동일 알고리즘의 여러 모델 병렬 학습 진행



앙상블(ENSEMBLE)

◆ 배깅(Bagging, Bootstrap Aggregation)

동작원리

- **중복 허용 랜덤** 샘플링 통한 **학습 샘플 데이터 생성**
- **동일 모델 여러 개 병렬 학습** 진행
- 학습 결과 결합
  - 분류 : 다중 투표
  - 회귀 : 평균

대표 알고리즘

- RandomForest => RandomForestClassifier
  RandomForestRegressor

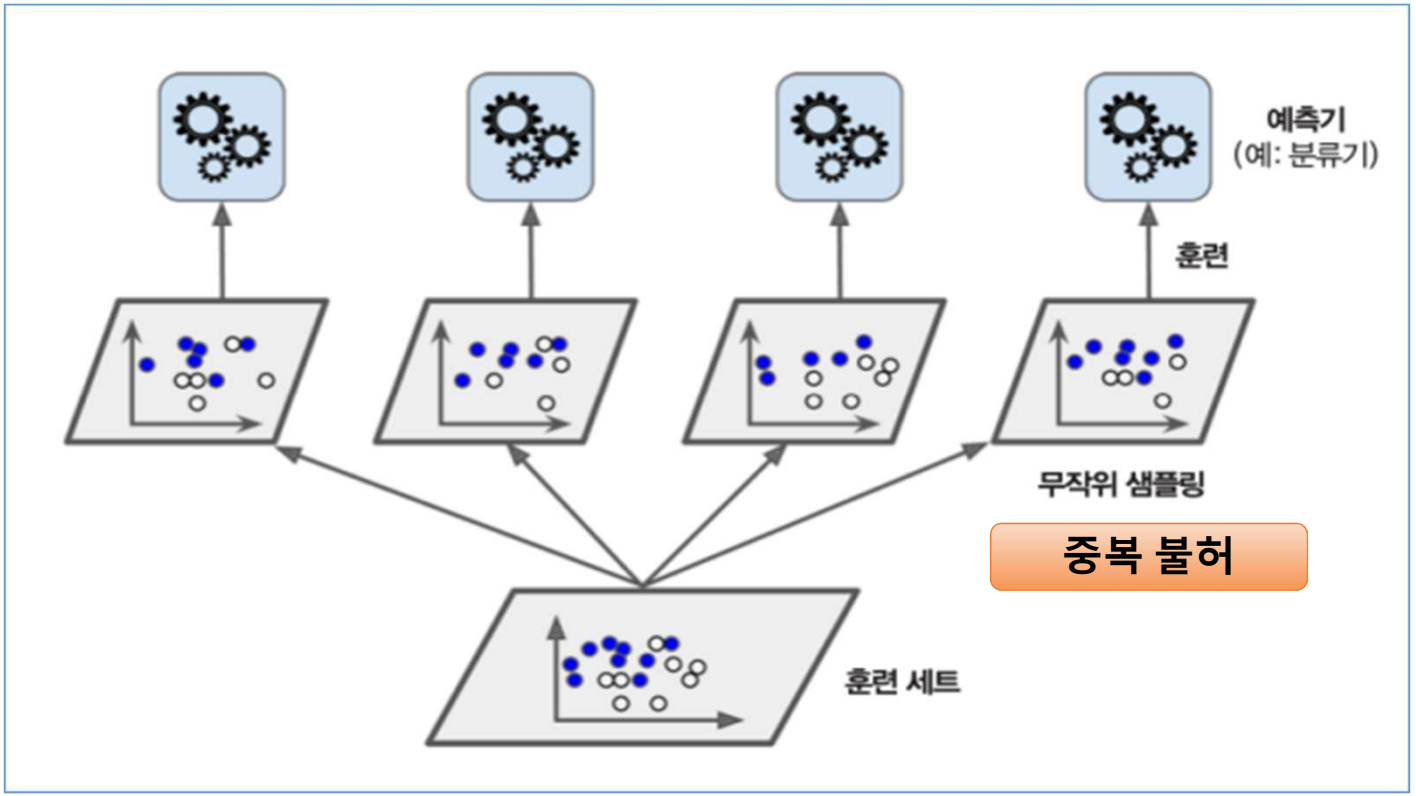
앙상블(ENSEMBLE)

◆ 페이스팅(Pasting)

중복 불허 샘플링하는 방식 의미하며 병렬 학습 진행

중복 불허 샘플링 다른 데이터와 동일 알고리즘의 여러 모델 병렬 학습

대표 알고리즘 : ExtraTrees



앙상블(ENSEMBLE)

◆ 랜덤포레스트(RandomForest)

- **앙상블 학습의 대표적인 방법 중 하나**
- 안정적인 성능으로 널리 사용
- 결정트리를 여러 개 랜덤하게 만들어 모델들의 예측값들을 사용
- 앙상블 알고리즘 중 비교적 빠른 수행 속도
- 다양한 영역에서 높은 예측 성능

- 분류, 회귀 모두 사용
  - ➢ 분류 : RandomForestClassifier
  - ➢ 회귀 : RandomForestRegressor

앙상블(ENSEMBLE)

◆ 랜덤포레스트(RandomForest)

『**동작 방식**』

① **중복 허용**한 훈련 데이터 **랜덤**하게 추출 ➔ 랜덤 샘플링

➔ **중복 허용** 샘플링, **부트스트랩 샘플**

➔ **매번 훈련 데이터 셋 다름**

➔ **중복 허용되어 성능 나쁨 : 결정트리 과적합 회피**

② 전체 특성에서 **무작위 특성 추출**

➔ **불순도 최소가 되는 특성 선택 ➔ 최적 노드 분할 적용**

➔ 분류 : 선택 특성 수 = 전체 특성 개수의 **제곱근 (특성 줄임)**

➔ 회귀 : 선택 특성 수 = 전체 특성

앙상블(ENSEMBLE)

◆ 랜덤포레스트(RandomForest)

『동작 방식』

③ **예측결과 결정**

➔ 분류 : 각 트리의 클래스별 확률 평균 **후 가장 높은 확률 가진 클래스** 선정

➔ 회귀 : 예측값들의 **평균값을 예측값**

랜덤한 샘플과 특성 추출 후 모델들 생성

훈련 세트에 **과대적합 방지**

검증 세트, 테스트 세트의 **안정적인 성능 보장**

앙상블(ENSEMBLE)

◆ 랜덤포레스트(RandomForest)

OOB(Out Of Bag)

➔ 중복 샘플링으로 남겨진 데이터 발생

➔ 해당 데이터를 **검증 세트로 사용** ➔ **OOB Sample**

특성중요도

➔ 랜덤포레스의 장점

➔ **특성의 상대적 중요도 측정** 가능

앙상블(ENSEMBLE)

◆ 랜덤포레스트(RandomForest)

- Scikit-Learn LIB

```
sklearn.ensemble.RandomForestClassifier
( n_estimators=100          # DecisionTree 100개
  criterion='gini'
  max_depth=None
  min_samples_split=2
  min_samples_leaf=1
  min_weight_fraction_leaf=0.0
  max_features='sqrt'
  max_leaf_nodes=None
  min_impurity_decrease=0.0
```

앙상블(ENSEMBLE)

◆ 랜덤포레스트(RandomForest)

➢ Scikit-Learn LIB

```
sklearn.ensemble.RandomForestClassifier
(  bootstrap=True              # 중복 허용 랜덤 샘플링
   oob_score=False,            # OOB 샘플로 성능 평가, 검증 데이터로 사용
   n_jobs=None
   random_state=None,
   verbose=0
   warm_start=False
   class_weight=None
   ccp_alpha=0.0
   max_samples=None
 )
```

앙상블(ENSEMBLE)

◆ 엑스트라 트리(ExtraTrees)

- RandomForest와 매우 비슷
- **가장 극단적인 형태의 랜덤 포레스트**
- 분산 감소 - 편향 증가

- 차이점
  - **샘플데이터 ➔** 부트스트랩 샘플 사용하지 않음. 즉 **중복불허!**
    ➔ 전체 입력 샘플 사용
  - **노드 분할 ➔ 무작위 분할** (가장 좋은 분할 찾지 X)
    ➔ 무작위 분할 후 그 중 좋은 것 선택 후 분할
    ➔ **RandomForest에 비해 속도 빨라짐**
    ➔ **RandomForest 보다 Tree 수 늘려야 함**

앙상블(ENSEMBLE)

◆ 엑스트라 트리(ExtraTrees)

- Scikit-Learn LIB

```
sklearn.ensemble.ExtraTreesClassifier
( n_estimators=100        # DecisionTree 100개
  criterion='gini'
  max_depth=None
  min_samples_split=2
  min_samples_leaf=1
  min_weight_fraction_leaf=0.0
  max_features='sqrt'
  max_leaf_nodes=None
  min_impurity_decrease=0.0
```

앙상블(ENSEMBLE)

◆ 엑스트라 트리(ExtraTrees)

➢ Scikit-Learn LIB

```
sklearn.ensemble.ExtraTreesClassifier
(  bootstrap=False          # 중복 불허 랜덤 샘플링
   oob_score=False,         # OOB 샘플로 성능 평가, 검증 데이터로 사용
   n_jobs=None
   random_state=None,
   verbose=0
   warm_start=False
   class_weight=None
   ccp_alpha=0.0
   max_samples=None
 )
```

앙상블(ENSEMBLE)

◆ 부스팅(Boosting)

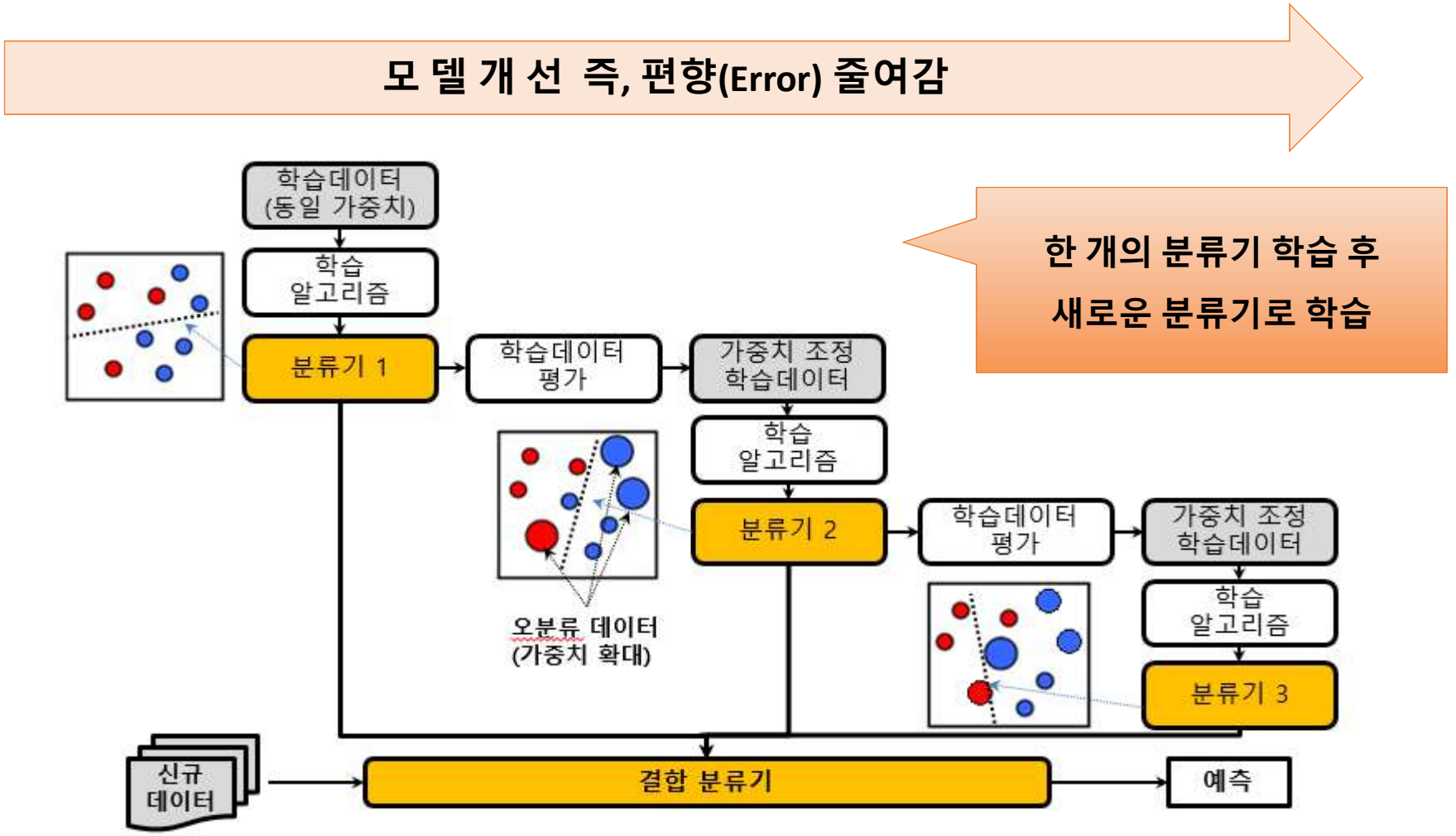
성능 약한 학습기 여러 개 순차적으로 모델(추정기) 추가 및 학습 진행

이전 모델(추정기)의 학습 결과 토대로 다음 모델(추정기)의 학습 데이터

샘플 가중치 조정해서 학습 진행

앙상블(ENSEMBLE)

◆ 부스팅(Boosting)



앙상블(ENSEMBLE)

◆ 부스팅(Boosting)

동작원리

- 이전 학습 결과의 오답에 대한 가중치 부여
- 다음 학습에서 오답에 대한 학습 진행
- 모델 성능 개선

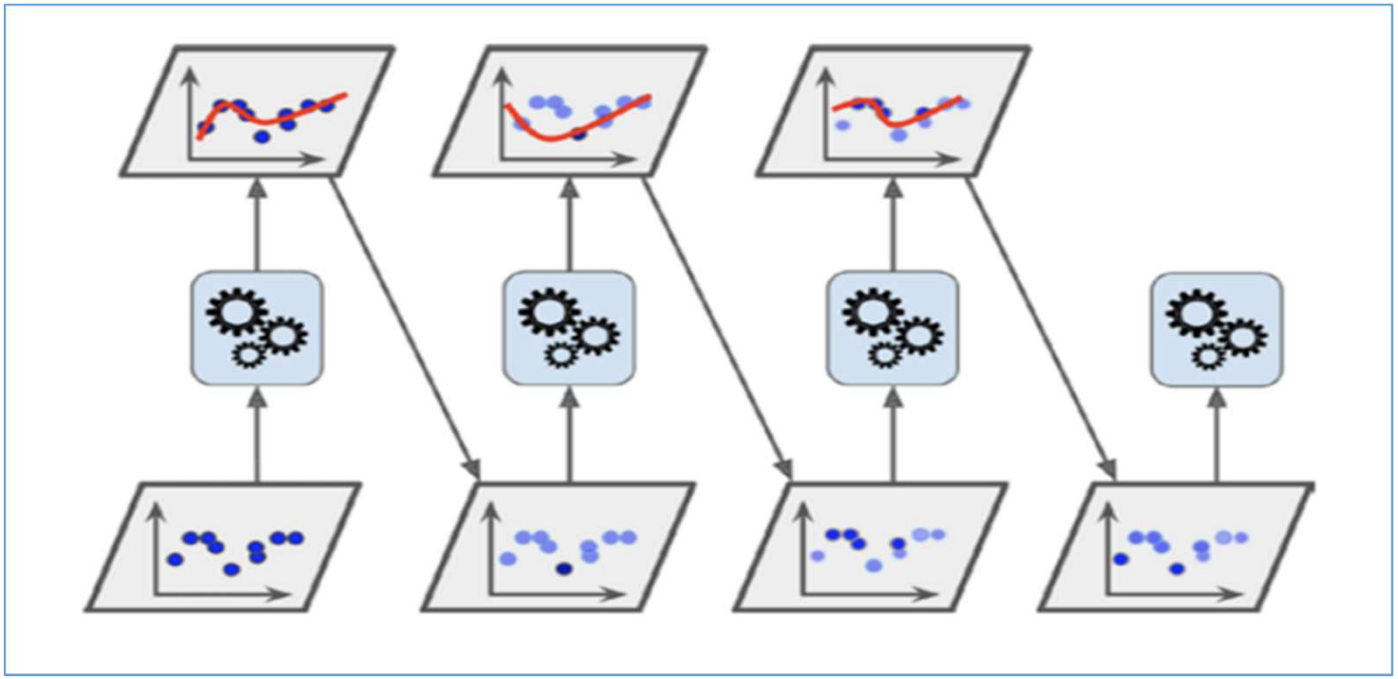
대표 알고리즘

- AdaBoost
- GradiantBoost
- XGBoost, LightGBM

앙상블(ENSEMBLE)

◆ 에이다부스트(AdaBoost)

- **좀 더 나은 모델 생성을 위해** 잘못 적용된 가중치 조정하여 새로운 모델을 추가하는 기법
- 과소적합했던 샘플들에 대한 가중치 더 높이는 방식



앙상블(ENSEMBLE)

◆ 그래디언트 부스팅(Gradient Boosting)

- AdaBoost와 동일한 방식
- 샘플의 가중치를 수정하는 **대신 이전 모델이 만든 오차(Error)에 대해 새로운 모델을 학습** 시키는 방식
- 모델 적용
  - 분류 : GradientBoostingClassifier
  - 회귀 : GradientBoostoingRegressor

앙상블(ENSEMBLE)

◆ 히스토그램기반부스팅(Histogram-based GB)

- **그레이디언트 부스팅의 속도를 개선**한 버전
- 안정적인 결과와 **높은 성능**으로 인기가 좋음
- 정형 데이터를 다루는 **머신러닝 알고리즘 중 가장 인기가 높음**

앙상블(ENSEMBLE)

◆ 히스토그램기반부스팅(Histogram-based GB)

- **동작원리**
  - ✓ **입력 특성 256개 구간**으로 나눔
  - ✓ 그중 하나 누락된 값을 위해 사용 ➔ 누락 특성 전처리 필요 x

- 대표 알고리즘
  - HistGradientBoostingClassifier
  - HistGradientBoostingRegressor
  - **XGBoost , LightGBM**

앙상블(ENSEMBLE)

◆ XGBoost(Extreme Gradient Boosting)

- Gradient Boosting을 분산환경에서도 실행할 수 있도록 구현
- **성능과 자원 효율 좋아 인기 있는 알고리즘**
- 분류, 회귀 모두 지원
- **여러개의 Decision Tree를 조합**해서 사용

- 설치 : !pip install xgboost

앙상블(ENSEMBLE)

◆ XGBoost(Extreme Gradient Boosting)

『 장 점 』

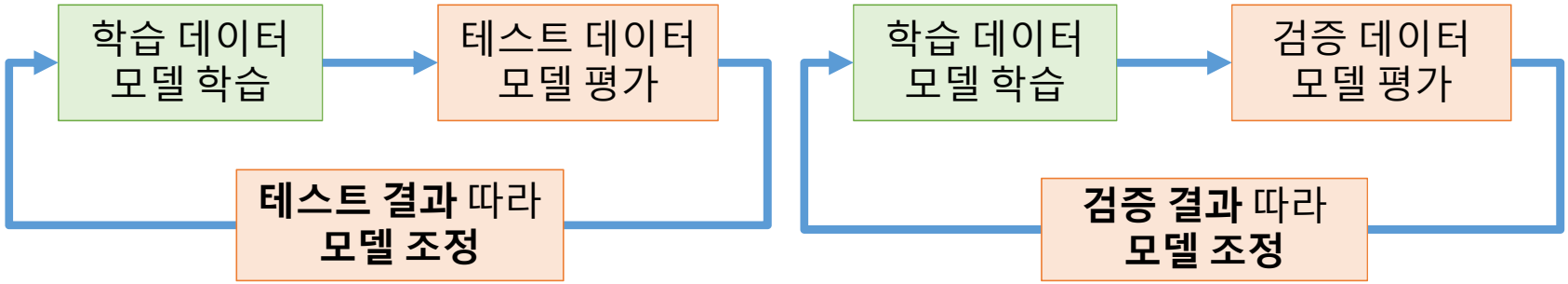
- GBM 대비 **빠른 수행시간**
  - 병렬 처리로 학습, 분류 속도 빠름
- 과적합 규제(Regularization)
  - 자체에 **과적합 규제 기능**으로 강한 내구성
- 분류와 회귀영역에서 **뛰어난 예측 성능** 발휘
- **Early Stopping(조기 종료)** 기능이 있음
- 다양한 옵션을 제공하며 **Customizing 용이**

Verification and Tuning

VERIFICATION & TUNING

◆ 검증 세트

- 학습 중 모델을 평가하기 위한 DataSet
- 가장 좋은 모델을 선택할 수 있음
- 학습 DataSet의 20~30% 정도



VERIFICATION & TUNING

◆ 교차 검증(CV : Cross Validation)

- 훈련 데이터가 줄어드는 문제 및 **데이터가 충분하지 않은 문제 해결**
- **테스트 데이터에 과대적합(Overfitting)** 문제 해결

- 훈련 데이터를 **동일 크기로 여러 조각 나눈** 후 데이터를 **교차시켜 훈련/검증 데이터**로 활용

[장점] 모든 데이터셋을 훈련과 평가에 활용 가능

=> 정확도 ▲ , 데이터부족 과소적합 방지

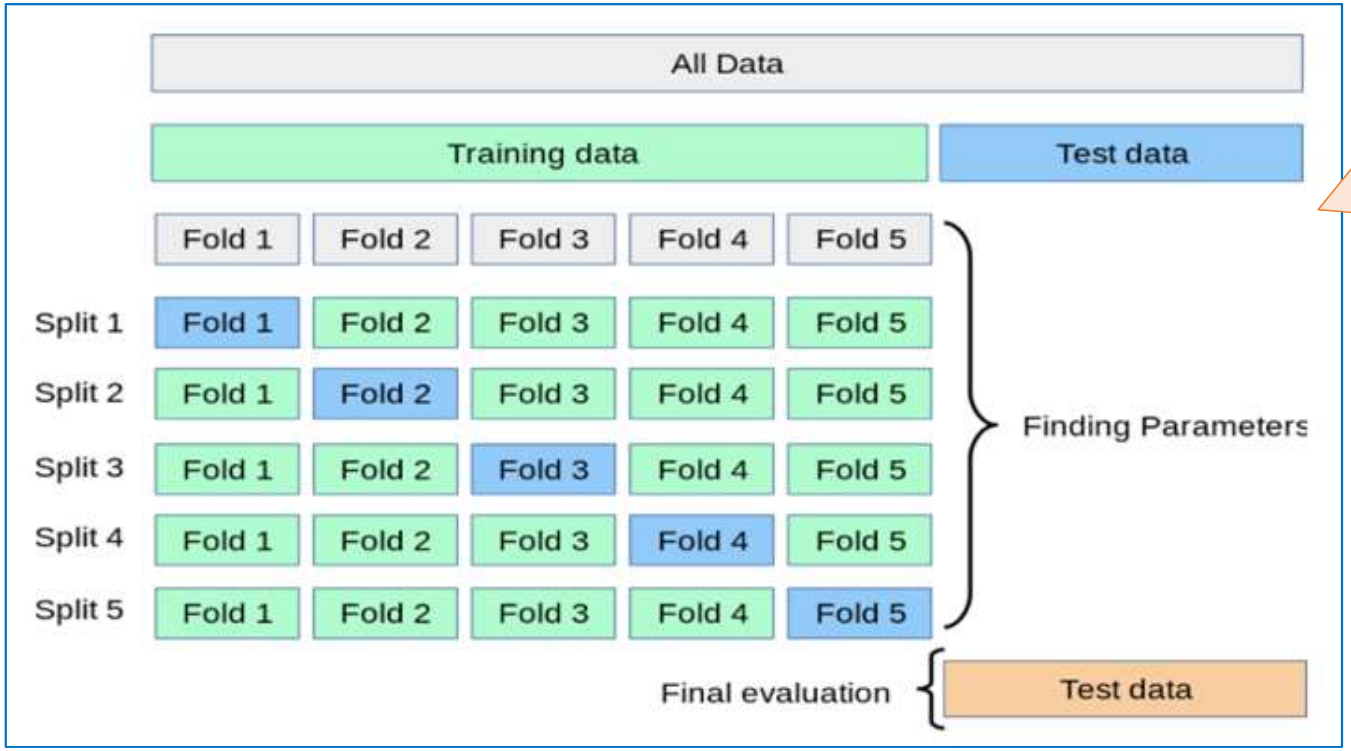
=> 평가용 데이터 편중 ▼, 평가 결과에 일반화된 모델 생성

[단점] 훈련과 평가에 많은 시간 소요

VERIFICATION & TUNING

◆ 교차 검증(CV : Cross Validation)

- **K-Fold Cross Validation**
  - 가장 일반적으로 사용되는 교차 검증 방법.
  - 보통 회귀 모델에 사용, 데이터가 독립적이고 동일한 분포 가진 경우 사용



총 K개의
성능 평가결과
평균

VERIFICATION & TUNING

◆ 교차 검증(CV : Cross Validation)

➢ Scikit-Learn LIB

```
sklearn.model_selection. KFold
(*,
  n_splits=5                    : 분할 개수, 최소 2개 이상
  shuffle=False                 : 데이터 섞기 설정
  random_state=None             : 난수 seed 설정
)
```

[단점] Target 데이터가 편중 되는 경우 발생

VERIFICATION & TUNING

◆ 교차 검증(CV : Cross Validation)

➢ Scikit-Learn LIB

```
# 불균형한 분포도를 가진 레이블 데이터 집합을 위한 KFold 방식
sklearn.model_selection. StratifiedFold
(*,
  n_splits=5                    : 분할 개수, 최소 2개 이상
  shuffle=False                 : 데이터 섞기 설정
  random_state=None             : 난수 seed 설정
)
```