



# 3장 평가(Evaluation)

파이썬 머신러닝 완벽 가이드

# 분류(Classification) 성능 평가 지표

- 정확도(Accuracy)
- 오차행렬(Confusion Matrix)
- 정밀도(Precision)
- 재현율(Recall)
- F1 스코어
- ROC AUC



# 정확도(Accuracy)

$$\text{정확도(Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

- 정확도는 직관적으로 모델 예측 성능을 나타내는 평가 지표입니다. 하지만 이진 분류의 경우 데이터의 구성에 따라 ML 모델의 성능을 왜곡할 수 있기 때문에 정확도 수치 하나만 가지고 성능을 평가하지 않습니다.
- 특히 정확도는 불균형한(imbalanced) 레이블 값 분포에서 ML 모델의 성능을 판단할 경우, 적합한 평가 지표가 아닙니다.

# 정확도의 문제점

타이타닉 생존자 예측에서 여성은 모두 생존으로 판별

If Sex = '여성'

생존

MNIST 데이터셋을 multi classification에서 binary classification 으로 변경

0	1	2	3	4	False	False	False	False	False
0	1	2	3	4	0	1	2	3	4
5	6	7	8	9	False	False	True	False	False
5	6	7	8	9	5	6	7	8	9

# 오차 행렬(Confusion Matrix)

오차 행렬은 이진 분류의 예측 오류가 얼마인지와 더불어 어떠한 유형의 예측 오류가 발생하고 있는지를 함께 나타내는 지표입니다

		예측 클래스(Predicted Class)	
		Negative(0)	Positive (1)
실제 클래스 (Actual Class)	Negative(0)	TN (True Negative)	FP (False Positive)
	Positive(1)	FN (False Negative)	TP (True Positive)

# 오차 행렬을 통한 정확도 지표 문제점 인지

		예측 클래스	
		Negative	Positive
		TN	FP
Negative	예측 : Negative (7 이 아닌 Digit)	405 개	0
	실제 : Negative (7 이 아닌 Digit)		실제: Negative (7 이 아닌 Digit)
Positive	FN	예측 : Negative (7 이 아닌 Digit)	TP
	45 개	실제 : Positive ( Digit 7 )	예측: Positive ( Digit 7 )
			0
			실제 : Positive ( Digit 7 )

• TP는 0임. Positive로 예측이 한 건도 성공하지 않음.

• 이와 더불어 FP가 0 이므로 Positive로 예측 자체를 수행하지 않음을 알 수 있음.

정확도 = 예측 결과와 실제 값이 동일한 건수/전체 데이터 수 =  $(TN + TP) / (TN + FP + FN + TP)$

# 정밀도(Precision)과 재현율(Recall)

- 정밀도 =  $TP / (FP + TP)$
- 재현율 =  $TP / (FN + TP)$

		예측 클래스	
		Negative	Positive
실제 클래스	Negative	TN 405 개	FP 0
	Positive	FN 45 개	TP 0

- 정밀도는 예측을 Positive로 한 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율을 뜻합니다.
- 재현율은 실제 값이 Positive인 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율을 뜻합니다.

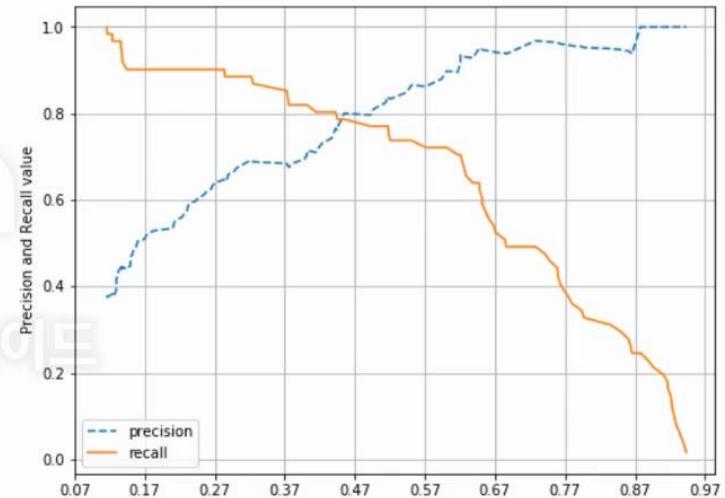
# 업무에 따른 재현율과 정밀도의 상대적 중요도

- 재현율이 상대적으로 더 중요한 지표인 경우는 실제 Positive 양성인 데이터 예측을 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 : 암 진단, 금융사기 판별
- 정밀도가 상대적으로 더 중요한 지표인 경우는 실제 Negative 음성인 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우: 스팸 메일

★ 불균형한 레이블 클래스를 가지는 이진 분류 모델에서는 많은 데이터 중에서 중점적으로 찾아야 하는 매우 적은 수의 결괏값에 Positive를 설정해 1값을 부여하고, 그렇지 않은 경우는 Negative로 0 값을 일반적으로 부여합니다.

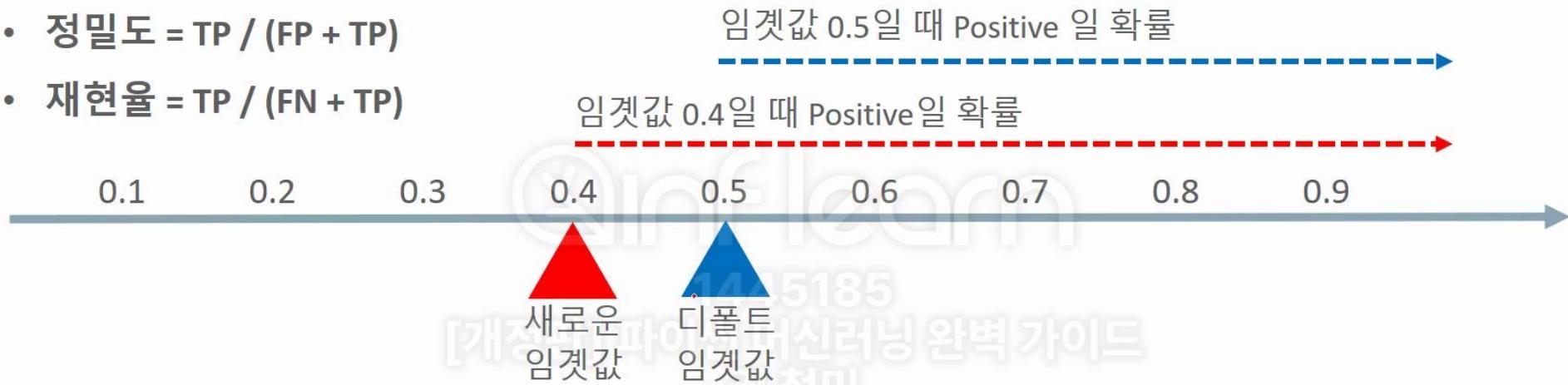
# 정밀도/재현율 트레이드오프

- 분류하려는 업무의 특성상 정밀도 또는 재현율이 특별히 강조돼야 할 경우 분류의 결정 임곗값(Threshold)을 조정해 정밀도 또는 재현율의 수치를 높일 수 있습니다.
- 하지만 정밀도와 재현율은 상호 보완적인 평가 지표이기 때문에 어느 한쪽을 강제로 높이면 다른 하나의 수치는 떨어지기 쉽습니다. 이를 정밀도/재현율의 트레이드오프(Trade-off)라고 부릅니다.



# 분류 결정 임곗값에 따른 Positive 예측 확률 변화

- 정밀도 =  $TP / (FP + TP)$
- 재현율 =  $TP / (FN + TP)$



분류 결정 임곗값이 낮아질 수록 Positive로 예측할 확률이 높아짐. 재현율 증가

- 사이킷런 Estimator 객체의 `predict_proba()` 메소드는 분류 결정 예측 확률을 반환합니다.
- 이를 이용하면 임의로 분류 결정 임곗값을 조정하면서 예측 확률을 변경할 수 있습니다.

## 분류 결정 임곗값에 따른 정밀도, 재현율 곡선

- 사이킷런은 `precision_recall_curve()` 함수를 통해 임곗값에 따른 정밀도, 재현율의 변화값을 제공합니다.

# 정밀도와 재현율의 맹점

## 정밀도를 100%로 만드는 법

- 확실한 기준이 되는 경우만 Positive로 예측하고 나머지는 모두 Negative로 예측합니다. 정밀도 =  $TP / (TP + FP)$ 입니다. 전체 환자 1000명 중 확실한 Positive 징후만 가진 환자는 단 1명이라고 하면 이 한 명만 Positive로 예측하고 나머지는 모두 Negative로 예측하더라도 FP는 0, TP는 1이 되므로 정밀도는  $1/(1+0)$ 으로 100%가 됩니다

## 재현율을 100%로 만드는 법

- 모든 환자를 Positive로 예측하면 됩니다. 재현율 =  $TP / (TP + FN)$ 이므로 전체 환자 1000명을 다 Positive로 예측하는 겁니다. 이 중 실제 양성인 사람이 30명 정도라도 TN이 수치에 포함되지 않고 FN은 아예 0이므로  $30/(30 + 0)$ 으로 100%가 됩니다.

# F1 Score

F1 스코어(Score)는 정밀도와 재현율을 결합한 지표입니다. F1 스코어는 정밀도와 재현율이 어느 한쪽으로 치우치지 않는 수치를 나타낼 때 상대적으로 높은 값을 가집니다. F1 스코어의 공식은 다음과 같습니다

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall}$$

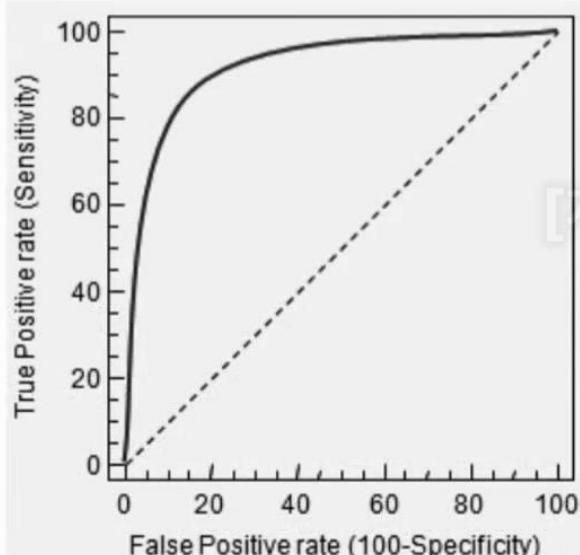
만일 A 예측 모델의 경우 정밀도가 0.9, 재현율이 0.1로 극단적인 차이가 있고, B 예측 모델은 정밀도가 0.5, 재현율이 0.5로 정밀도와 재현율이 큰 차이가 없다면 A 예측 모델의 F1 스코어는 0.18이고, B 예측 모델의 F1 스코어는 0.5로 B 모델이 A모델에 비해 매우 우수한 F1 스코어를 가지게 됩니다.

사이킷런은 f1 score를 위해 f1\_score( ) 함수를 제공합니다.



# ROC 곡선과 AUC

- ROC 곡선(Receiver Operation Characteristic Curve) 과 이에 기반한 AUC 스코어는 이진 분류의 예측 성능 측정에서 중요하게 사용되는 지표입니다. 일반적으로 의학 분야에서 많이 사용되지만, 머신러닝의 이진 분류 모델의 예측 성능을 판단하는 중요한 평가 지표이기도 합니다.



ROC 곡선은 FPR(False Positive Rate)이 변할 때 TPR(True Positive Rate)이 어떻게 변하는지를 나타내는 곡선입니다. FPR을 X 축으로, TPR을 Y 축으로 잡으면 FPR의 변화에 따른 TPR의 변화가 곡선 형태로 나타납니다.

분류의 성능 지표로 사용되는 것은 ROC 곡선 면적에 기반한 AUC 값으로 결정합니다. AUC(Area Under Curve) 값은 ROC 곡선 밑의 면적을 구한 것으로서 **일반적으로 1에 가까울수록 좋은 수치입니다**

# 사이킷런 ROC 곡선 및 AUC 스코어

- 사이킷런은 임곗값에 따른 ROC 곡선 데이터를 `roc_curve()`로, AUC 스코어를 `roc_auc_score()` 함수로 제공

API 명	입력 파라미터	반환 값
<code>roc_curve( y_true, y_score )</code>	<ul style="list-style-type: none"><li><code>y_true</code>: 실제 클래스 값 array ( array shape = [데이터 건수] )</li><li><code>y_score</code>: <code>predict_prob()</code>의 반환 값 array에서 Positive 컬럼의 예측 확률이 보통 사용됨. Binary 분류 시 shape = [n_samples]</li></ul>	<code>fpr</code> : <code>fpr</code> 값을 array로 반환 <code>tpr</code> : <code>tpr</code> 값을 array로 반환 <code>thresholds</code> : threshold 값을 array
<code>roc_auc_score(y_true, y_score )</code>	<ul style="list-style-type: none"><li><code>y_true</code>: 실제 클래스 값 array ( array shape = [데이터 건수] )</li><li><code>y_score</code>: <code>predict_prob()</code>의 반환 값 array에서 Positive 컬럼의 예측 확률이 보통 사용됨. Binary 분류 시 shape = [n_samples]</li></ul>	Auc 스코어 값



# 4장 분류(Classification)

파이썬 머신러닝 완벽 가이드

# 분류 알고리즘

분류(Classification)는 학습 데이터로 주어진 데이터의 피처와 레이블값(결정 값, 클래스 값)을 머신러닝 알고리즘으로 학습해 모델을 생성하고, 이렇게 생성된 모델에 새로운 데이터 값이 주어졌을 때 미지의 레이블 값을 예측하는 것입니다.

## 대표적인 분류 알고리즘들

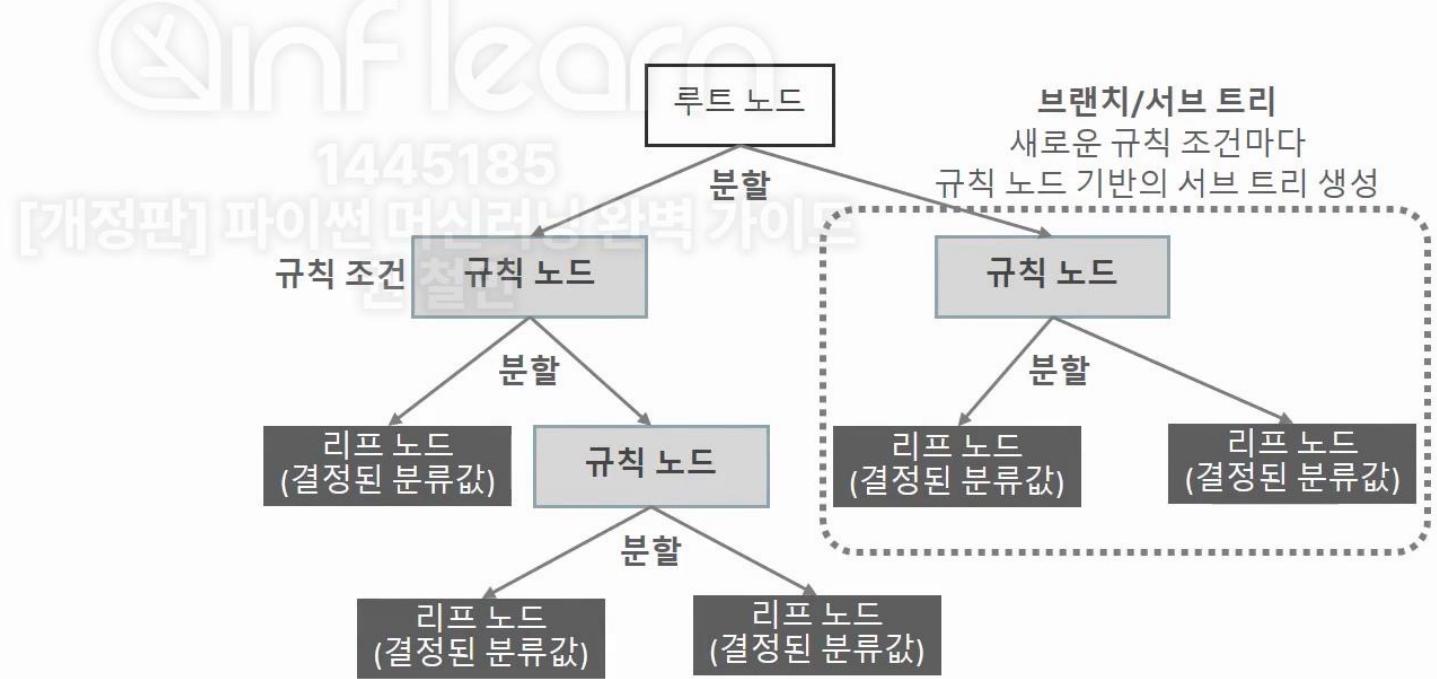
- 베이즈(Bayes) 통계와 생성 모델에 기반한 나이브 베이즈(Naïve Bayes)
- 독립변수와 종속변수의 선형 관계성에 기반한 로지스틱 회귀(Logistic Regression)
- 데이터 균일도에 따른 규칙 기반의 결정 트리(Decision Tree)
- 개별 클래스 간의 최대 분류 마진을 효과적으로 찾아주는 서포트 벡터 머신(Support Vector Machine)
- 근접 거리를 기준으로 하는 최소 근접(Nearest Neighbor) 알고리즘
- 심층 연결 기반의 신경망(Neural Network)
- 서로 다른(또는 같은) 머신러닝 알고리즘을 결합한 앙상블(Ensemble)

# 결정 트리와 앙상블

- 결정 트리는 매우 쉽고 유연하게 적용될 수 있는 알고리즘입니다. 또한 데이터의 스케일링이나 정규화 등의 사전 가공의 영향이 매우 적습니다. 하지만 예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 하며, 이로 인한 과적합(overfitting)이 발생해 반대로 예측 성능이 저하될 수도 있다는 단점이 있습니다.
- 하지만 이러한 단점이 앙상블 기법에서는 오히려 장점으로 작용합니다. 앙상블은 매우 많은 여러 개의 약한 학습기(즉, 예측 성능이 상대적으로 떨어지는 학습 알고리즘)를 결합해 확률적 보완과 오류가 발생한 부분에 대한 가중치를 계속 업데이트하면서 예측 성능을 향상시키는데, 결정 트리가 좋은 약한 학습기가 되기 때문입니다(GBM, XGBoost, LightGBM 등)

# 결정 트리

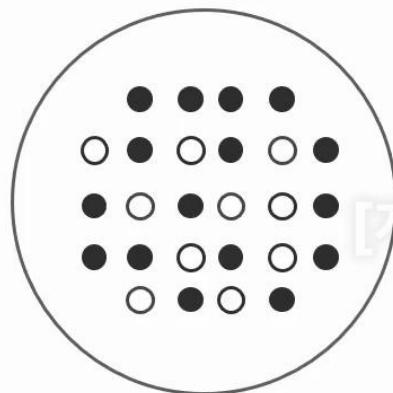
- 결정 트리 알고리즘은 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리(Tree) 기반의 분류 규칙을 만듭니다(If-Else 기반 규칙)
- 따라서 데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인가가 알고리즘의 성능을 크게 좌우합니다



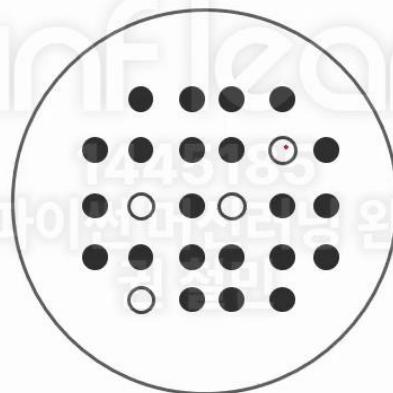
# 트리 분할을 위한 데이터의 균일도

다음중 가장 균일한 데이터 셋은?

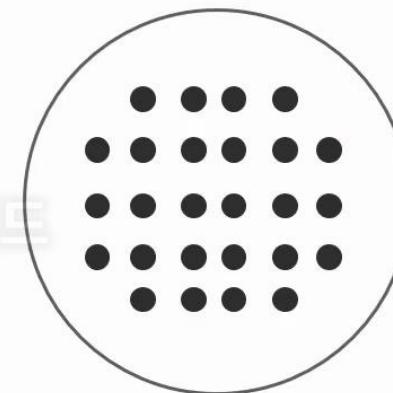
데이터 셋 A



데이터 셋 B

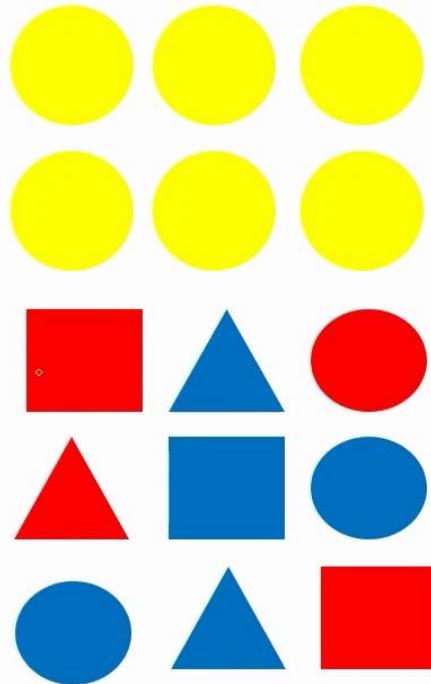


데이터 셋 C



답은 C

# 균일도 기반 규칙 조건



노랑색 블록의 경우 모두 동그라미로 구성되고, 빨강과 파랑 블록의 경우는 동그라미, 네모, 세모가 골고루 섞여 있다고 한다면 각 레고 블록을 분류하고자 할 때 가장 첫 번째로 만들어져야 하는 규칙 조건은?

1445185

[개정판] 파이썬 머신러닝 완벽 가이드

권철민

if 색깔 == '노란색':

# 정보 균일도 측정 방법

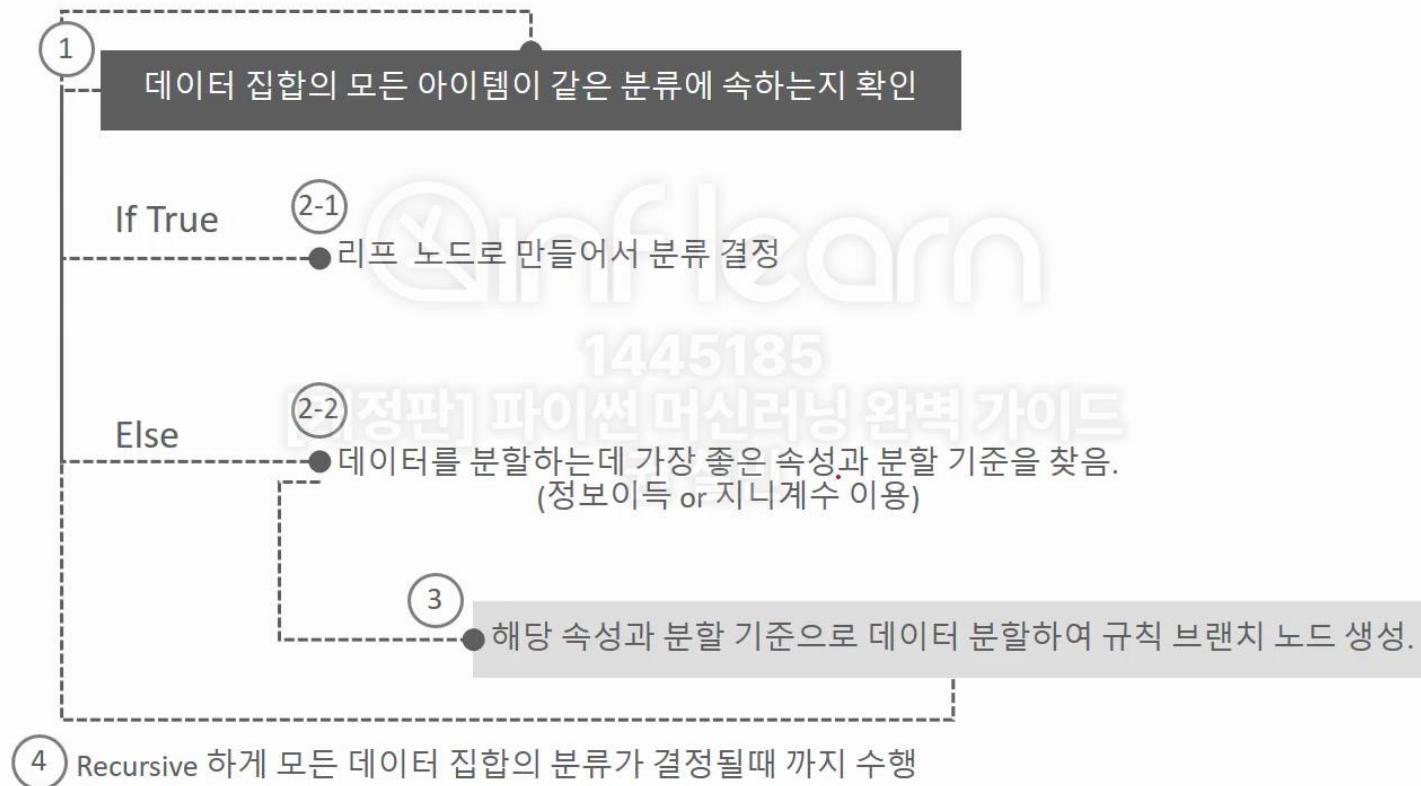
## 정보 이득(Information Gain)

정보 이득은 엔트로피라는 개념을 기반으로 합니다. 엔트로피는 주어진 데이터 집합의 혼잡도를 의미하는데, 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮습니다. 정보 이득 지수는 1에서 엔트로피 지수를 뺀 값입니다. 즉, 1-엔트로피 지수입니다. 결정 트리는 이 정보 이득 지수로 분할 기준을 정합니다. 즉, 정보 이득이 높은 속성을 기준으로 분할합니다.

## 지니 계수

지니 계수는 원래 경제학에서 불평등 지수를 나타낼 때 사용하는 계수입니다. 경제학자인 코라도 지니(Corrado Gini)의 이름에서 딴 계수로서 0이 가장 평등하고 1로 갈수록 불평등합니다. 머신러닝에 적용될 때는 지니 계수가 낮을 수록 데이터 균일도가 높은 것으로 해석되어 계수가 낮은 속성을 기준으로 분할 합니다.

# 결정 트리의 규칙 노드 생성 프로세스



# 결정 트리의 특징

## 결정 트리 장점

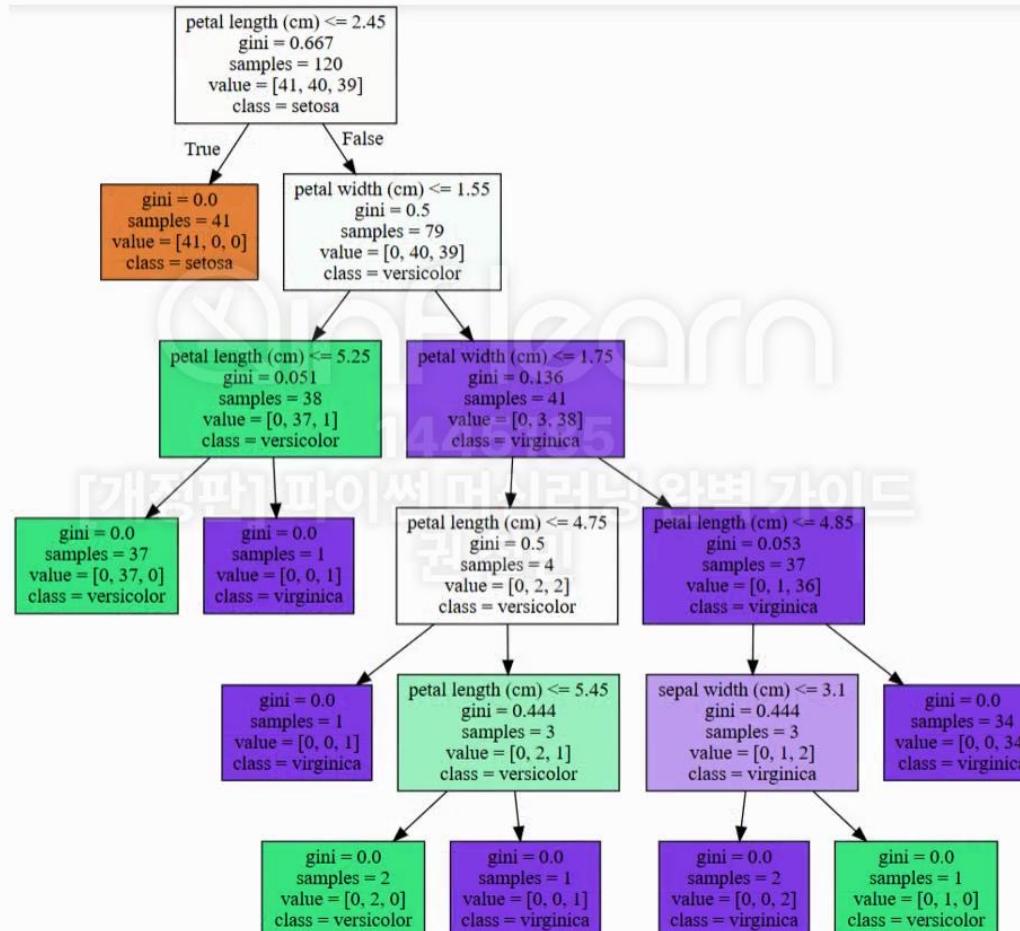
- 쉽다. 직관적이다
- 피처의 스케일링이나 정규화 등의 사전 가공 영향도가 크지 않음.
- 과적합으로 알고리즘 성능이 떨어진다. 이를 극복하기 위해 트리의 크기를 사전에 제한하는 튜닝 필요.

## 결정 트리 단점

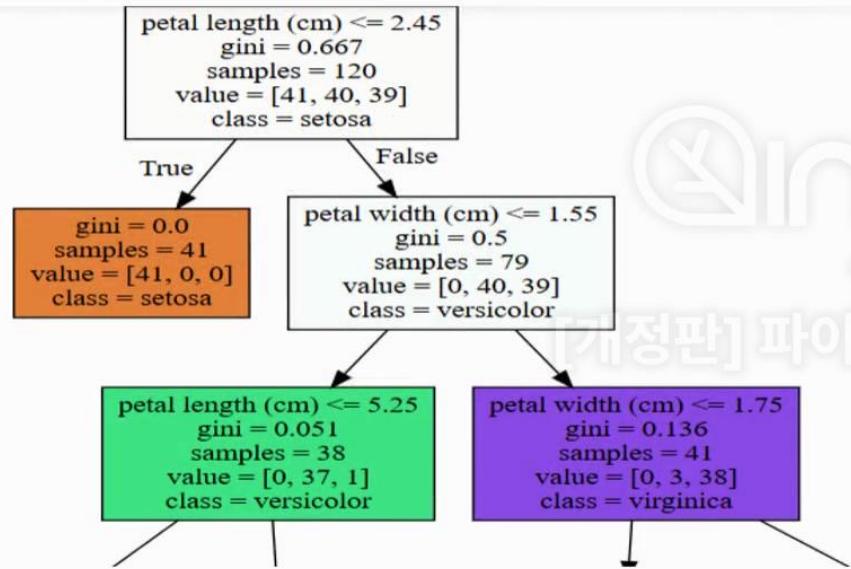
# 결정 트리 주요 하이퍼 파라미터

파라미터 명	설명
<b>max_depth</b>	<ul style="list-style-type: none"><li>트리의 최대 깊이를 규정.</li><li>디폴트는 <code>None</code>. <code>None</code>으로 설정하면 완벽하게 클래스 결정 값이 될 때까지 깊이를 계속 키우며 분할하거나 노드가 가지는 데이터 개수가 <code>min_samples_split</code>보다 작아질 때까지 계속 깊이를 증가시킴.</li><li>깊이가 깊어지면 <code>min_samples_split</code> 설정대로 최대 분할하여 과적합할 수 있으므로 적절한 값으로 제어 필요.</li><li>최적의 분할을 위해 고려할 최대 피처 개수. 디폴트는 <code>None</code>으로 데이터 세트의 모든 피처를 사용해 분할 수행.</li><li><code>int</code> 형으로 지정하면 대상 피처의 개수, <code>float</code> 형으로 지정하면 전체 피처 중 대상 피처의 퍼센트임</li></ul>
<b>max_features</b>	<ul style="list-style-type: none"><li>'sqrt'는 전체 피처 중 <math>\sqrt{\text{전체 피처 개수}}</math> 만큼 선정</li><li>'auto'로 지정하면 <code>sqrt</code>와 동일</li><li>'log'는 전체 피처 중 <math>\log_2(\text{전체 피처 개수})</math> 선정</li><li>'None'은 전체 피처 선정</li></ul>
<b>min_samples_split</b>	<ul style="list-style-type: none"><li>노드를 분할하기 위한 최소한의 샘플 데이터 수로 과적합을 제어하는 데 사용됨.</li><li>디폴트는 2이고 작게 설정할수록 분할되는 노드가 많아져서 과적합 가능성 증가</li><li>과적합을 제어. 1로 설정할 경우 분할되는 노드가 많아져서 과적합 가능성 증가</li></ul>
<b>min_samples_leaf</b>	<ul style="list-style-type: none"><li>말단 노드(Leaf)가 되기 위한 최소한의 샘플 데이터 수</li><li><code>Min_samples_split</code>과 유사하게 과적합 제어 용도. 그러나 비대칭적(imbalanced) 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 이 경우는 작게 설정 필요.</li></ul>
<b>max_leaf_nodes</b>	<ul style="list-style-type: none"><li>말단 노드(Leaf)의 최대 개수</li></ul>

# Graphviz를 이용한 결정 트리 모델의 시각화



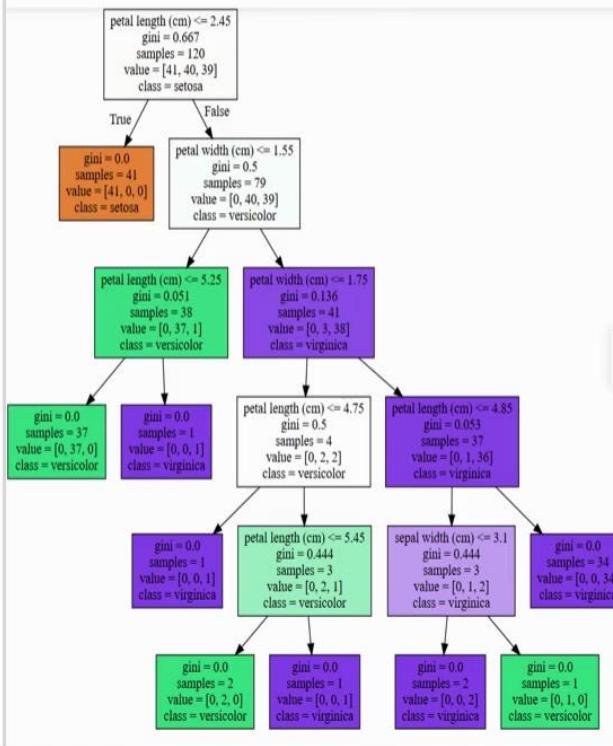
# Graphviz의 시각화 노드



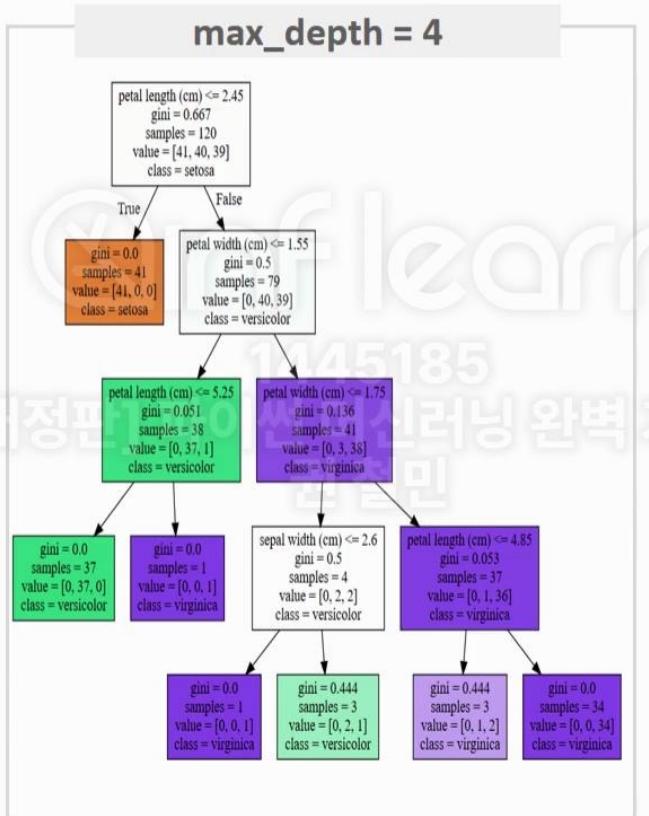
- petal length(cm) <= 2.45와 같이 피처의 조건이 있는 것은 자식 노드를 만들기 위한 규칙 조건입니다. 이 조건이 없으면 리프 노드입니다.
- gini는 다음의 value=[ ]로 주어진 데이터 분포에서의 지니 계수입니다.
- samples는 현 규칙에 해당하는 데이터 건수입니다.
- value = [ ]는 클래스 값 기반의 데이터 건수입니다. 붓꽃 데이터 세트는 클래스 값으로 0, 1, 2를 가지고 있으며, 0 : Setosa, 1: Versicolor, 2: Virginica 품종을 가리킵니다. 만일 Value = [41, 40, 39]라면 클래스 값의 순서로 Setosa 41개, Vesicolor 40개, Virginica 39개로 데이터가 구성돼 있다는 의미입니다.
- class는 value 리스트내에 가장 많은 건수를 가진 결정값입니다.

## max\_depth에 따른 결정 트리 구조

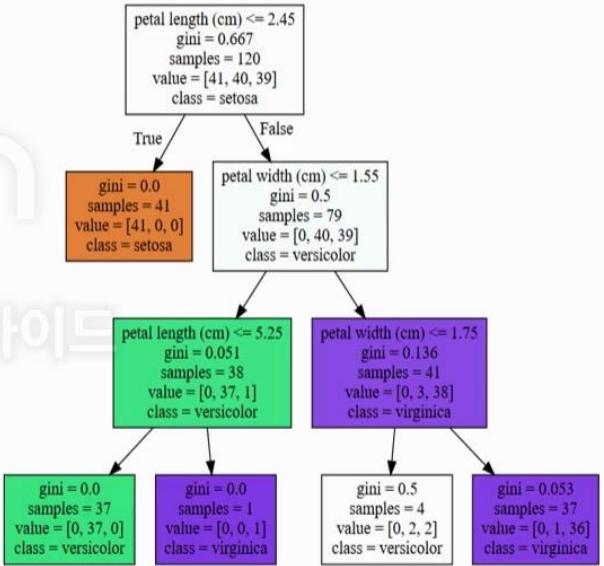
`max_depth` 제약 없음



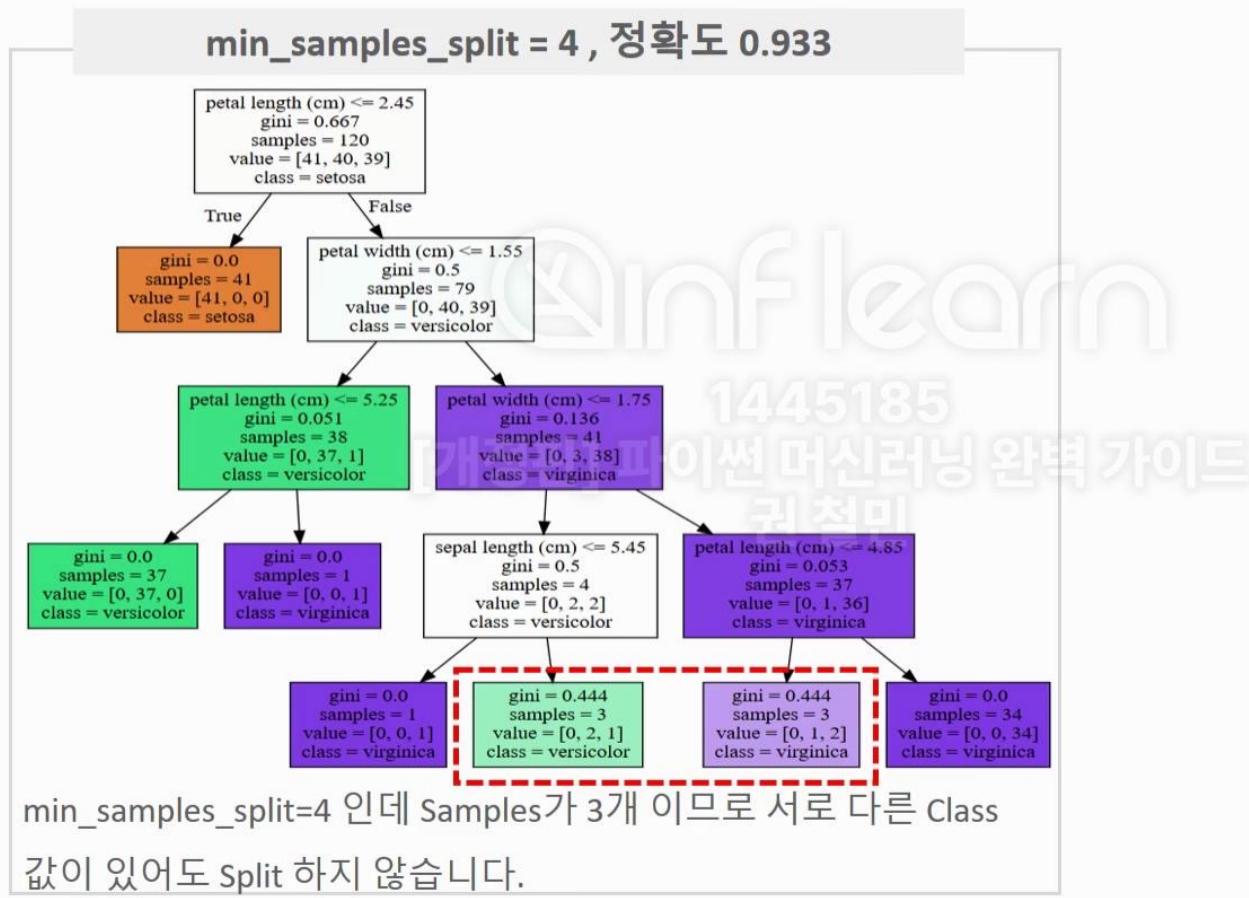
**max\_depth = 4**



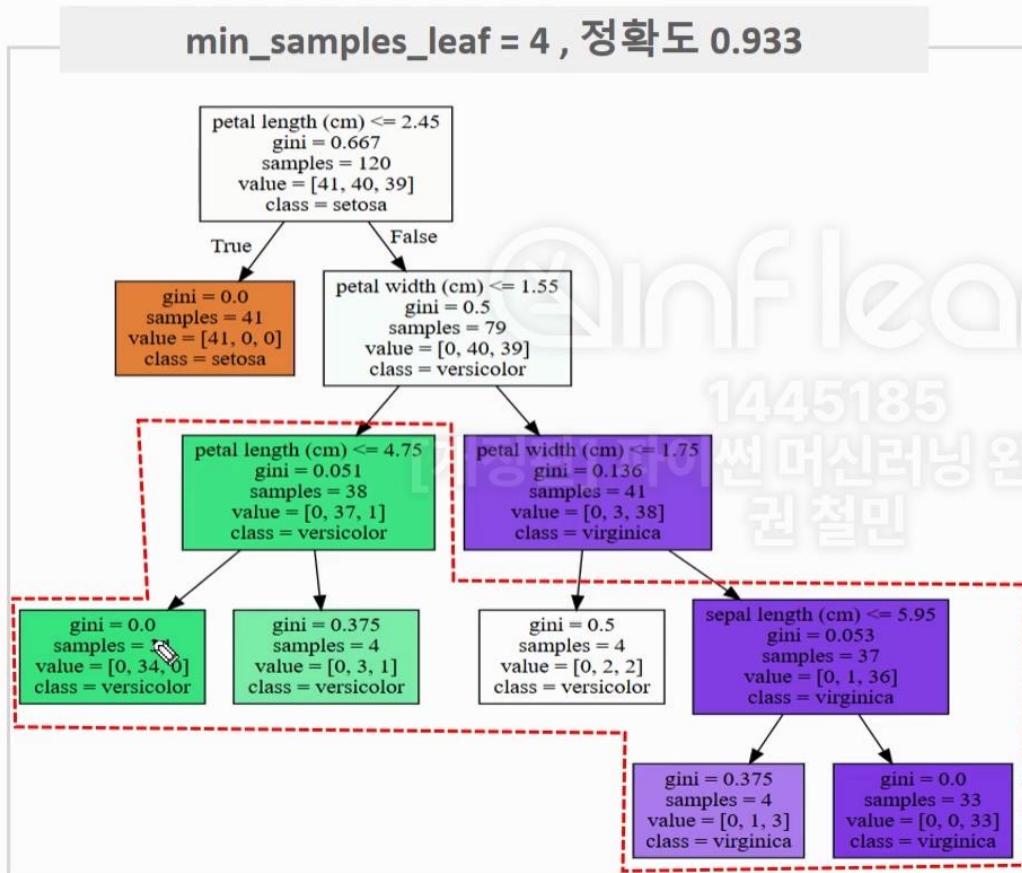
**max\_depth = 3**



# min\_samples\_split 에 따른 결정 트리 구조



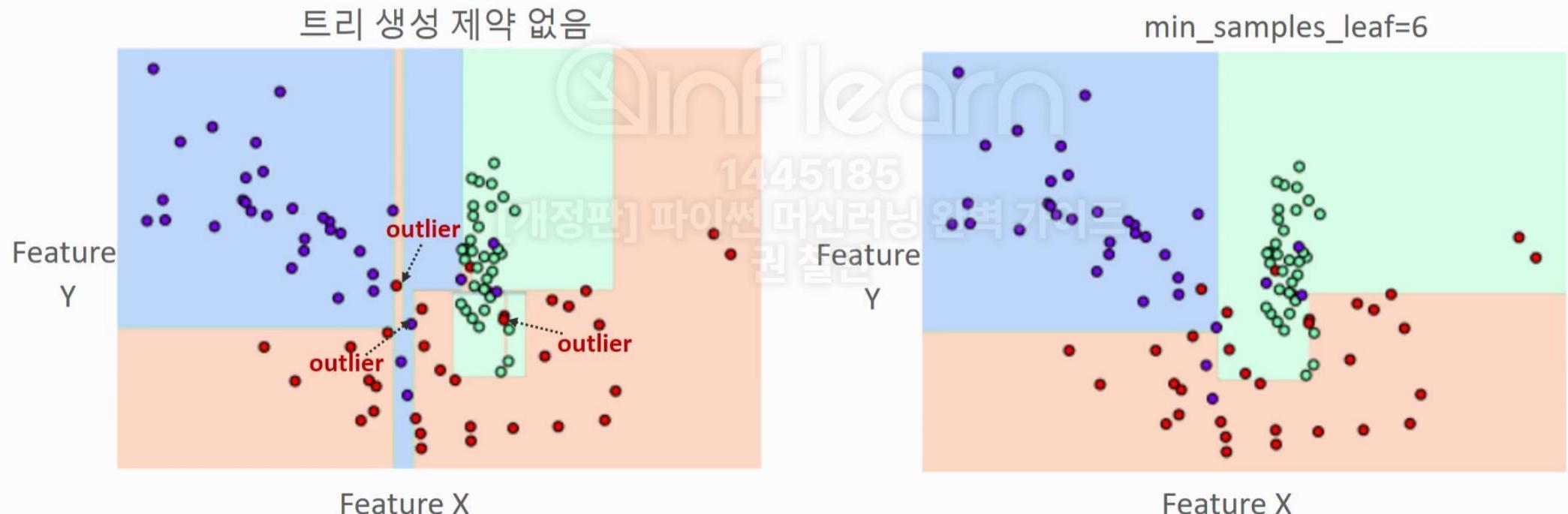
# min\_samples\_leaf에 따른 결정 트리 구조



노드가 분할될 경우 자식 노드들이 모두 샘플 데이터 건수가 4 이상을 만족할 수 있는지를 확인한 후에 분할을 수행하게 됩니다.  
결정 트리는 해당 조건을 반영하여 트리 구조를 변경합니다

# 결정 트리 과적합

2개의 Feature로 된 3개의 결정 클래스를 가지도록 `make_classification()` 함수를 이용하여 임의 데이터를 생성한 후 트리 생성 제약이 없는 경우와 `min_samples_leaf=6`으로 제약을 주었을 때 분류 기준선의 변화



# 앙상블 학습 – 앙상블 학습 개요

앙상블 학습 (Ensemble Learning) 을 통한 분류는 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법을 말합니다.

어려운 문제의 결론을 내기 위해 여러 명의 전문가로 위원회를 구성해 다양한 의견을 수렴하고 결정하듯이 앙상블 학습의 목표는 다양한 분류기의 예측 결과를 결합함으로써 단일 분류기보다 신뢰성이 높은 예측값을 얻는 것입니다.



# 앙상블의 유형

- 앙상블의 유형은 일반적으로는 보팅(Voting), 배깅(Bagging), 부스팅(Boosting)으로 구분할 수 있으며, 이외에 스태킹(Stacking)등의 기법이 있습니다.
- 대표적인 배깅은 랜덤 포레스트(Random Forest) 알고리즘이 있으며, 부스팅은 에이다 부스팅, 그래디언트 부스팅, XGBoost, LightGBM 등이 있습니다. 정형 데이터의 분류나 회귀에서는 GBM 부스팅 계열의 앙상블이 전반적으로 높은 예측 성능을 나타냅니다.
- 넓은 의미로는 서로 다른 모델을 결합한 것들을 앙상블로 지칭하기도 합니다.

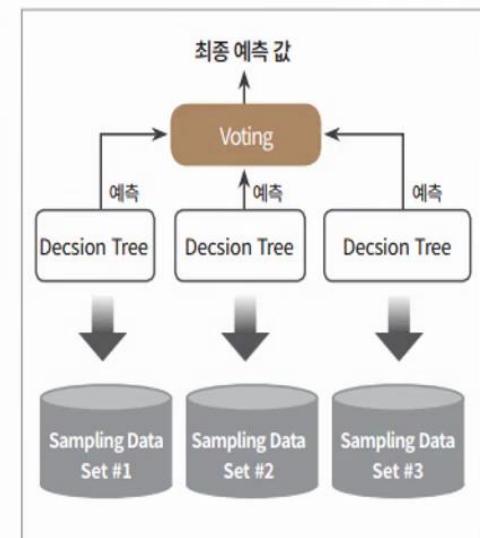
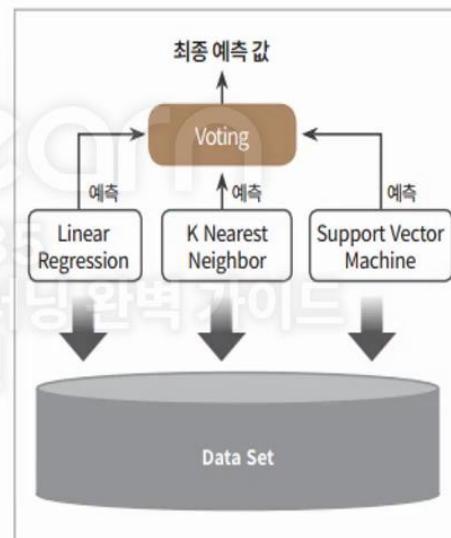
# 앙상블의 특징

- 단일 모델의 약점을 다수의 모델들을 결합하여 보완
- 뛰어난 성능을 가진 모델들로만 구성하는 것보다 성능이 떨어지더라도 서로 다른 유형의 모델을 섞는 것이 오히려 전체 성능이 도움이 될 수 있음.
- 랜덤 포레스트 및 뛰어난 부스팅 알고리즘들은 모두 결정 트리 알고리즘을 기반 알고리즘으로 적용함.
- 결정 트리의 단점인 과적합(오버 피팅)을 수십~수천개의 많은 분류기를 결합해 보완하고 장점인 직관적인 분류 기준은 강화됨.



# 보팅(Voting)과 배깅(Bagging) 개요

- 보팅과 배깅은 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식입니다.
- 보팅과 배깅의 다른 점은 보팅의 경우 일반적으로 서로 다른 알고리즘을 가진 분류기를 결합하는 것이고, 배깅의 경우 각각의 분류기가 모두 같은 유형의 알고리즘 기반이지만, 데이터 샘플링을 서로 다르게 가져가면서 학습을 수행해 보팅을 수행 하는 것입니다.

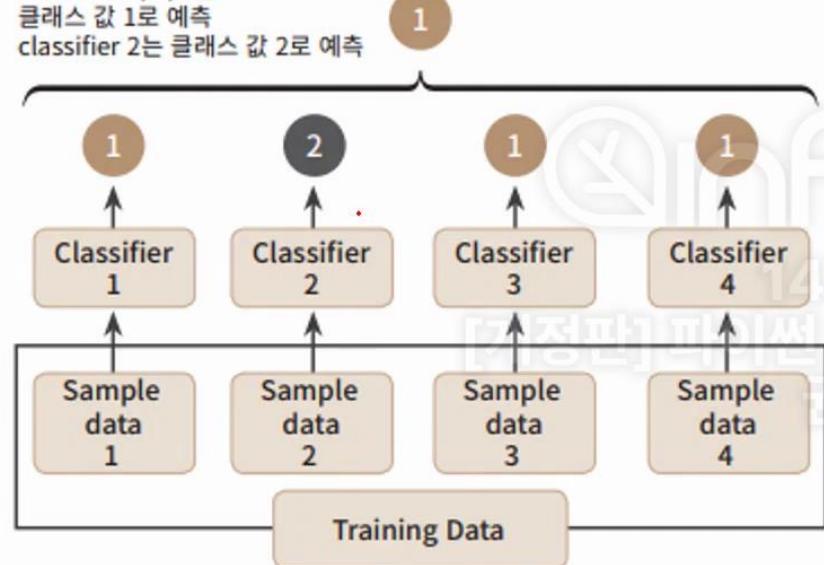


# 보팅 유형 – 하드 보팅(Hard Voting)과 소프트 보팅(Soft Voting)

Hard Voting은 다수의 classifier 간 다수결로 최종 class 결정

클래스 값 1로 예측  
classifier 1, 3, 4는  
클래스 값 1로 예측  
classifier 2는 클래스 값 2로 예측

1



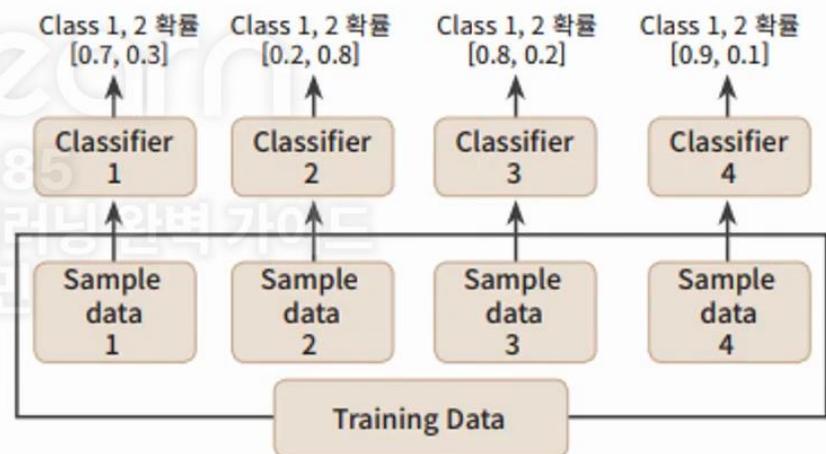
<하드 보팅>

Soft Voting은 다수의 classifier들의 class 확률을 평균하여 결정

클래스 값 1로 예측  
클래스 값 1일 확률: 0.65  
클래스 값 2일 확률: 0.35

1

predict\_proba() 메소드를  
이용하여 class 별 확률 결정

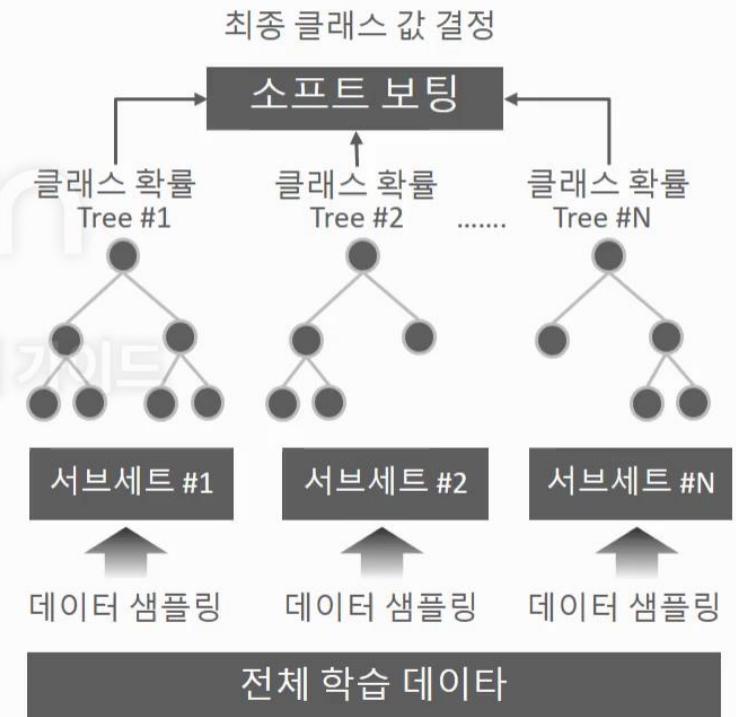


<소프트 보팅>

- 일반적으로 하드 보팅보다는 소프트 보팅이 예측 성능이 상대적으로 우수하여 주로 사용됨.
- 사이킷런은 **VotingClassifier** 클래스를 통해 보팅(Voting)을 지원

# 배깅(Bagging) – 랜덤 포레스트(Random Forest)

- 배깅의 대표적인 알고리즘은 랜덤 포레스트입니다.
- 랜덤 포레스트는 다재 다능한 알고리즘입니다. 앙상블 알고리즘 중 비교적 빠른 수행 속도를 가지고 있으며, 다양한 영역에서 높은 예측 성능을 보이고 있습니다.
- 랜덤 포레스트는 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정을하게 됩니다.



# 랜덤 포레스트의 부트스트래핑 분할

- 랜덤 포레스트는 개별적인 분류기의 기반 알고리즘은 결정 트리이지만 개별 트리가 학습하는 데이터 세트는 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트입니다. 이렇게 여러 개의 데이터 세트를 중첩되게 분리하는 것을 부트스트래핑(bootstrapping) 분할 방식이라고 합니다(그래서 배깅(Bagging)이 bootstrap aggregating의 줄임말입니다).
- 원본 데이터의 건수가 10개인 학습 데이터 세트에 랜덤 포레스트를 3개의 결정 트리 기반으로 학습하려고 n\_estimators=3으로 하이퍼 파라미터를 부여하면 다음과 같이 데이터 서브세트가 만들어 집니다.



# 사이킷런 랜덤 포레스트 하이퍼 파라미터

사이킷런은 랜덤 포레스트 분류를 위해 `RandomForestClassifier` 클래스를 제공합니다.

## RandomForestClassifier 하이퍼 파라미터

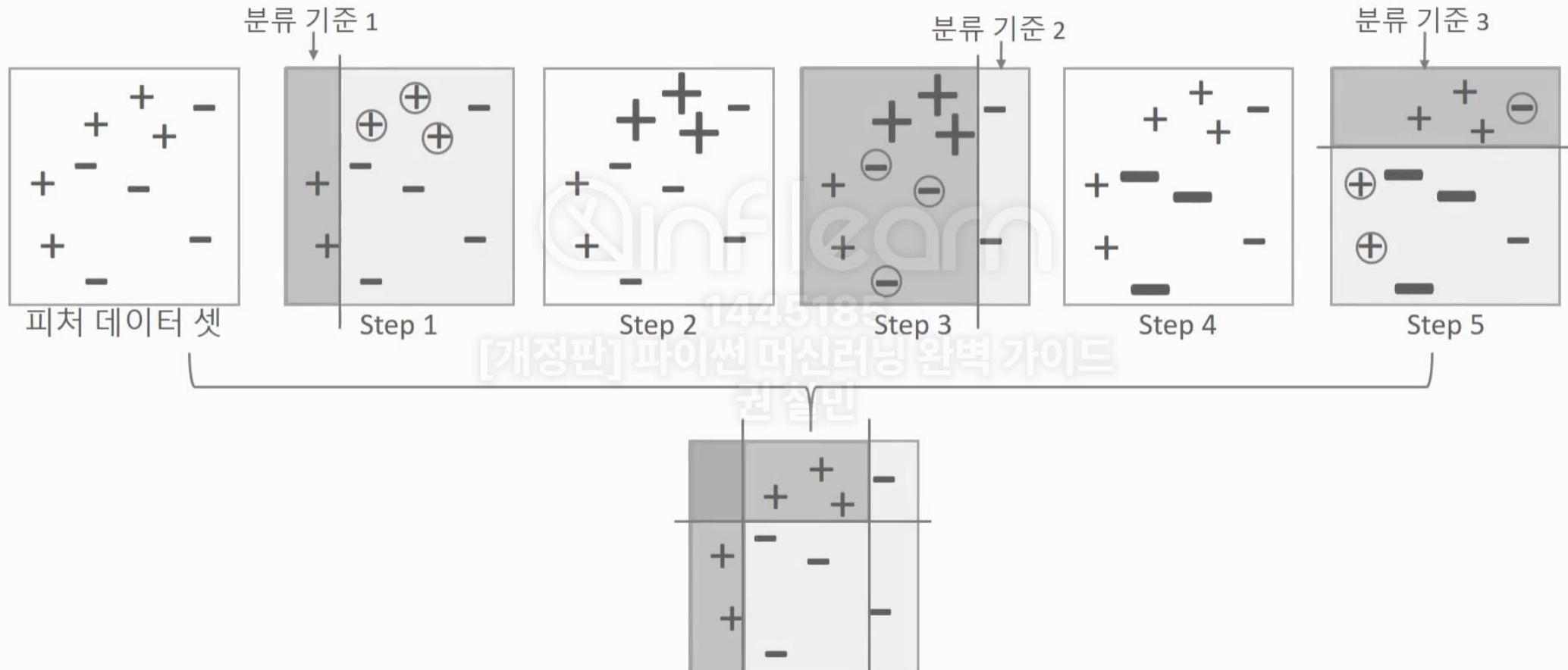
- `n_estimators`: 랜덤 포레스트에서 결정 트리의 개수를 지정합니다. 디폴트는 100개입니다. 많이 설정할수록 좋은 성능을 기대할 수 있지만 계속 증가시킨다고 성능이 무조건 향상되는 것은 아닙니다. 또한 늘릴수록 학습 수행 시간이 오래 걸리는 것도 감안해야 합니다.
- `max_features`는 결정 트리에 사용된 `max_features` 파라미터와 같습니다. 하지만 `RandomForestClassifier`의 기본 `max_features`는 'None'이 아니라 'auto', 즉 'sqrt'와 같습니다. 따라서 랜덤 포레스트의 트리를 분할하는 피처를 참조할 때 전체 피처가 아니라  $\text{sqrt}(\text{전체 피처 개수})$ 만큼 참조합니다(전체 피처가 16개라면 분할을 위해 4개 참조).
- `max_depth`나 `min_samples_leaf`와 같이 결정 트리에서 과적합을 개선하기 위해 사용되는 파라미터가 랜덤 포레스트에도 똑같이 적용될 수 있습니다.



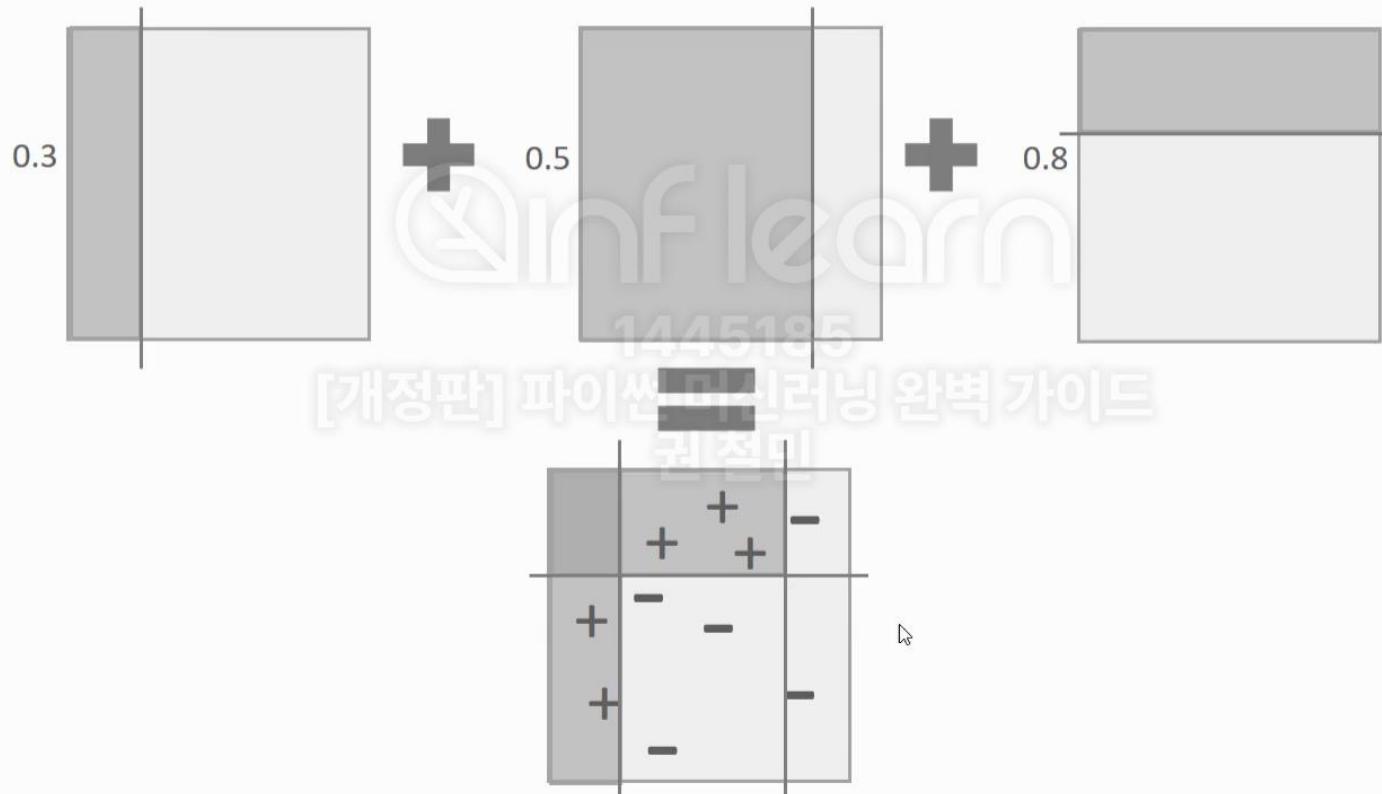
# 부스팅(Boosting)

- 부스팅 알고리즘은 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측한 데이터나 학습 트리에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식입니다.
- 부스팅의 대표적인 구현은 AdaBoost(Adaptive boosting)와 그래디언트 부스트가 있습니다

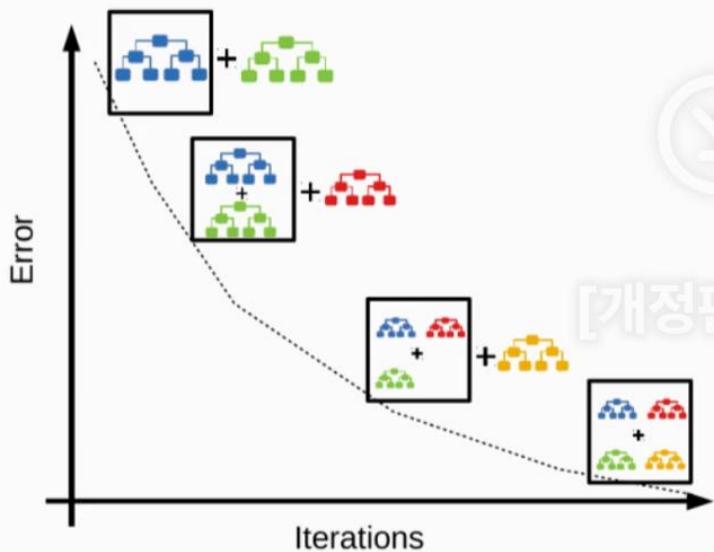
# 에이다 부스팅의 학습/예측 프로세스 - 1



# 에이다 부스팅의 학습/예측 프로세스 - 1



# GBM(Gradient Boost Machine) 개요



GBM(Gradient Boost Machine)도 에이다부스트와 유사하나, 가중치 업데이트를 경사 하강법 (Gradient Descent)을 이용하는 것이 큰 차이입니다.

오류 값은 실제 값 – 예측값입니다. 분류의 실제 결과값을  $y$ , 피처를  $x_1, x_2, \dots, x_n$ , 그리고 이 피처에 기반한 예측 함수를  $F(x)$  함수라고 하면 오류식  $h(x) = y - F(x)$ 이 됩니다. 이 오류식  $h(x) = y - F(x)$ 을 최소화하는 방향성을 가지고 반복적으로 가중치 값을 업데이트하는 것이 경사 하강법(Gradient Descent)입니다.

경사 하강법은 반복 수행을 통해 오류를 최소화할 수 있도록 가중치의 업데이트 값을 도출하는 기법으로서 머신러닝에서 중요한 기법 중 하나입니다.

# 사이킷런 GBM 주요 하이퍼 파라미터 및 튜닝

사이킷런은 GBM 분류를 위해 GradientBoostingClassifier 클래스를 제공합니다.

- **loss:** 경사 하강법에서 사용할 비용 함수를 지정합니다. 특별한 이유가 없으면 기본값인 'deviance'를 그대로 적용합니다.
- **learning\_rate:** GBM이 학습을 진행할 때마다 적용하는 학습률입니다. Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수입니다. 0~1 사이의 값을 지정할 수 있으며 기본값은 0.1입니다. 너무 작은 값을 적용하면 업데이트 되는 값이 작아져서 최소 오류 값을 찾아 예측 성능이 높아질 가능성이 높습니다. 하지만 많은 weak learner는 순차적인 반복이 필요해서 수행 시간이 오래 걸리고, 또 너무 작게 설정하면 모든 weak learner의 반복이 완료돼도 최소 오류 값을 찾지 못할 수 있습니다. 반대로 큰 값을 적용하면 최소 오류 값을 찾지 못하고 그냥 지나쳐 버려 예측 성능이 떨어질 가능성이 높아지지만, 빠른 수행이 가능합니다.
- **n\_estimators:** weak learner의 개수입니다. weak learner가 순차적으로 오류를 보정하므로 개수가 많을수록 예측 성능이 일정 수준까지는 좋아질 수 있습니다. 하지만 개수가 많을수록 수행 시간이 오래 걸립니다. 기본값은 100입니다.
- **subsample:** weak learner가 학습에 사용하는 데이터의 샘플링 비율입니다. 기본값은 1이며, 이는 전체 학습 데이터를 기반으로 학습한다는 의미입니다(0.5이면 학습 데이터의 50%). 과적합이 염려되는 경우 subsample을 1보다 작은 값으로 설정합니다.



# XGBoost 개요

XGBoost  
eXtra Gradient Boost

## 주요 장점

- 뛰어난 예측 성능
- GBM 대비 빠른 수행 시간
  - CPU 병렬 처리, GPU 지원
- 다양한 성능 향상 기능
  - 규제(Regularization) 기능 탑재
    - Tree Pruning
- 다양한 편의 기능
  - 조기 중단(Early Stopping)
  - 자체 내장된 교차 검증
  - 결손값 자체 처리

# XGBoost 파이썬 구현



C/C++ Native Module



파이썬 Wrapper

- XGBoost는 최초 C/C++로 작성됨.

- C/C++ 모듈을 호출하는  
파이썬 Wrapper



사이킷런 Wrapper

- 사이킷런 프레임과 통합 될 수 있는  
파이썬 Wrapper Class 지원
  - ✓ XGBClassifier
  - ✓ XGBRegressor
- 학습과 예측을 다른 사이킷런 API와  
동일하게 fit()과 predict()로 수행.
- GridSearchCV 와 같은 다른 사이킷런  
모듈과 같이 사용 가능.

# XGBoost 파이썬 래퍼와 사이킷런 래퍼 API 비교

항목	파이썬 Wrapper	사이킷런 Wrapper
사용 모듈	<code>from xgboost as xgb</code>	<code>from xgboost import XGBClassifier</code> *
학습용과 테스트용 데이터 세트	<code>DMatrix</code> 객체를 별도 생성 <code>train = xgb.DMatrix(data=X_train, label=y_train)</code> <code>DMatrix</code> 생성자로 피처 데이터 세트와 레이블 데이터 세트를 입력	넘파이나 판다스를 이용
학습 API	<code>Xgb_model=xgb.train()</code> <code>Xgb_model</code> 은 학습된 객체를 반환 받음	<code>XGBClassifier.fit()</code>
예측 API	<code>xgb.train()</code> 으로 학습된 객체에서 <code>predict()</code> 호출. 즉 <code>Xgb_model.predict()</code> 이때 반환 결과는 예측 결과가 아니라 예측 결과를 추정하는 확률값 반환	<code>XGBClassifier.predict()</code> 예측 결과값 반환
피처 중요도 시각화	<code>plot_importance()</code> 함수 이용	<code>plot_importance()</code> 함수 이용

# XGBoost 파이썬 래퍼 와 사이킷런 래퍼 하이퍼 파라미터 비교

파이썬 Wrapper	사이킷런 Wrapper	하이퍼 파라미터 설명
<code>eta</code>	<code>learning_rate</code>	GBM의 학습률(learning rate)과 같은 파라미터입니다. 0에서 1 사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값. 파이썬 래퍼 기반의 xgboost를 이용할 경우 디폴트는 0.3. 사이킷런 래퍼 클래스를 이용할 경우 eta는 learning_rate 파라미터로 대체되며, 디폴트는 0.1입니다.
<code>num_boost_rounds</code>	<code>n_estimators</code>	사이킷런 앙상블의 n_estimators와 동일. 약한 학습기의 개수(반복 수행 회수)
<code>min_child_weight</code>	<code>min_child_weight</code>	결정트리의 min_child_leaf와 유사. 과적합 조절용
<code>max_depth</code>	<code>max_depth</code>	결정트리의 max_depth와 동일. 트리의 최대 깊이
<code>sub_sample</code>	<code>subsample</code>	GBM의 subsample과 동일. 트리가 커져서 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정합니다. sub_sample=0.5로 지정하면 전체 데이터의 절반을 트리를 생성하는 데 사용합니다. 0에서 1사이의 값이 가능하나 일반적으로 0.5 ~ 1 사이의 값을 사용합니다.

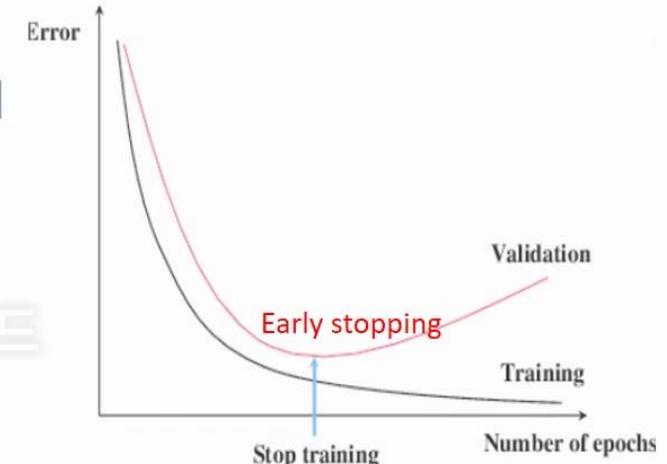
사이킷런 Wrapper의 경우 GBM에 동일한 하이퍼 파라미터가 있다면 이를 사용하고 그렇지 않다면 파이썬 Wrapper의 하이퍼 파라미터를 사용

# XGBoost 파이썬 래퍼 와 사이킷런 래퍼 하이퍼 파라미터 비교

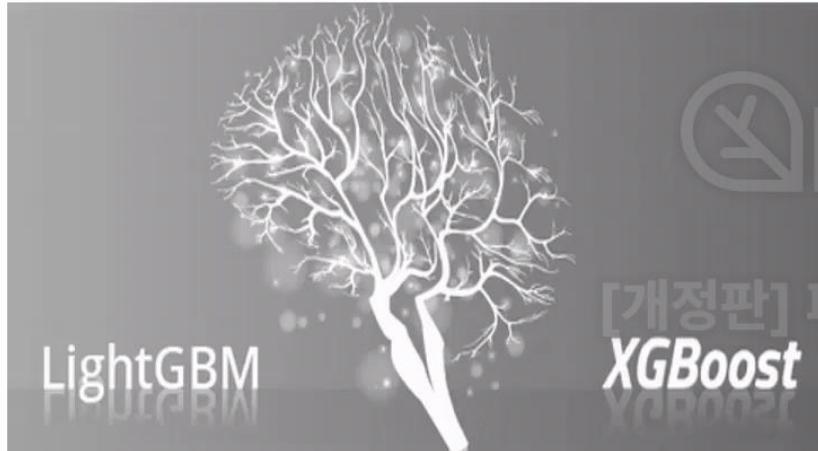
파이썬 Wrapper	사이킷런 Wrapper	하이퍼 파라미터 설명
lambda	reg_lambda	L2 규제(Regularization) 적용 값. 기본값은 1임. 값이 클 수록 규제 값이 커짐. 과적합 제어
alpha	reg_alpha	L1 규제(Regularization) 적용 값. 기본값은 0임. 값이 클 수록 규제 값이 커짐. 과적합 제어
colsample_bytree	colsample_bytree	GBM의 max_features와 유사합니다. 트리 생성에 필요한 피처(컬럼)을 임의로 샘플링 하는 데 사용됩니다. 매우 많은 피처가 있는 경우 과적합을 조정하는 데 적용합니다
scale_pos_weight	scale_pos_weight	특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형을 유지하기 위한 파라미터입니다. 기본값은 1
gamma	gamma	트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값입니다. 해당 값보다 큰 손실(loss)이 감소된 경우에 리프 노드를 분리합니다. 값이 클수록 과적합 감소 효과가 있습니다

# XGBoost 조기 중단 기능(Early Stopping)

- XGBoost 는 특정 반복 횟수 만큼 더 이상 비용함수가 감소하지 않으면 지정된 반복횟수를 다 완료하지 않고 수행을 종료할 수 있음.
- 학습을 위한 시간을 단축 시킬 수 있음. 특히 최적화 튜닝 단계에서 적절하게 사용 가능.
- 너무 반복 횟수를 단축할 경우 예측 성능 최적화가 안된 상태에서 학습이 종료 될 수 있으므로 유의 필요.
- 조기 중단 설정을 위한 주요 파라미터
  - ✓ `early_stopping_rounds` : 더 이상 비용 평가 지표가 감소하지 않는 최대 반복횟수.
  - ✓ `eval_metric` : 반복 수행 시 사용하는 비용 평가 지표.
  - ✓ `eval_set` : 평가를 수행하는 별도의 검증 데이터 세트. 일반적으로 검증 데이터 세트에서 반복적으로 비용 감소 성능 평가.



# LightGBM 개요

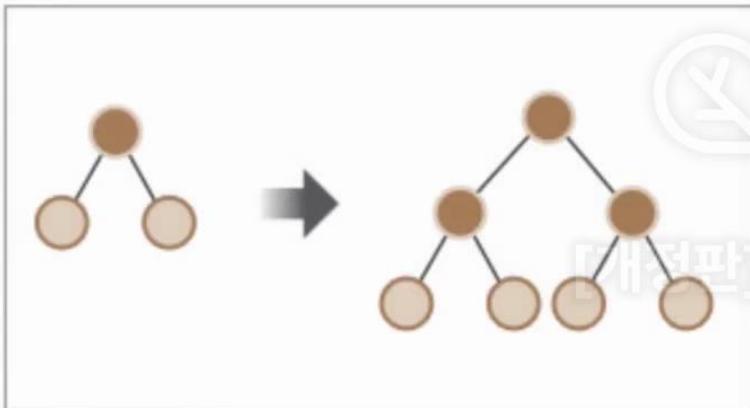


## XGBoost 대비 장점

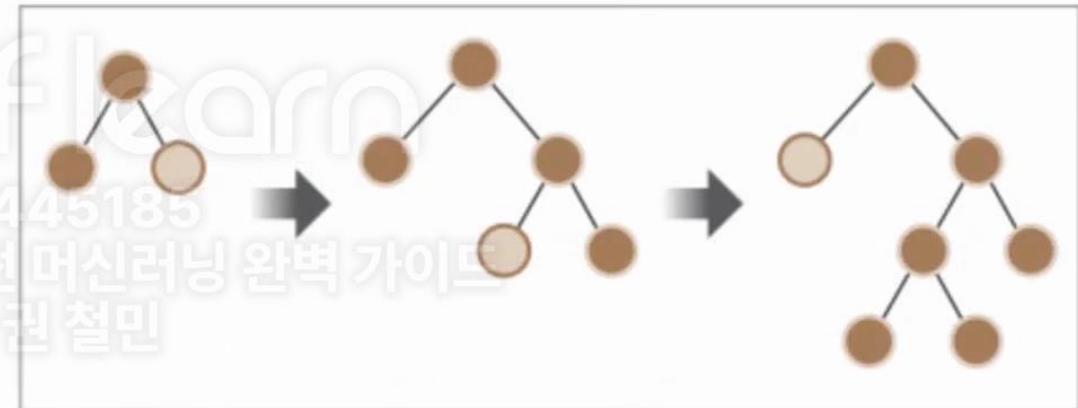
- 더 빠른 학습과 예측 수행 시간.
- 더 작은 메모리 사용량.
- 카테고리형 피처의 자동 변환과 최적 분할(원-핫 인코딩 등을 사용하지 않고도 카테고리형 피처를 최적으로 변환하고 이에 따른 노드 분할 수행).

# LightGBM 트리 분할 방식 – 리프 중심

균형 트리 분할(Level Wise)



리프 중심 트리 분할(Leaf Wise)



# XGBoost 파이썬 구현



C/C++ Native Module



파이썬 Wrapper

- XGBoost는 최초 C/C++로 작성됨.

- C/C++ 모듈을 호출하는  
파이썬 Wrapper



사이킷런 Wrapper

- 사이킷런 프레임과 통합 될 수 있는  
파이썬 Wrapper Class 지원
  - ✓ XGBClassifier
  - ✓ XGBRegressor
- 학습과 예측을 다른 사이킷런 API와  
동일하게 fit()과 predict()로 수행.
- GridSearchCV 와 같은 다른 사이킷런  
모듈과 같이 사용 가능.

# LightGBM 하이퍼 파라미터

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	
파라미터명	num_iterations	n_estimators	약한 학습기의 개수(반복 수행 회수)
	learning_rate	learning_rate	학습률(learning rate). 0에서 1 사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값
	max_depth	max_depth	결정트리의 max_depth와 동일. 트리의 최대 깊이
	min_data_in_leaf	min_child_samples	리프 노드가 될 수 있는 최소 데이터 건수(Sample 수)
	bagging_fraction	subsample	트리가 커져서 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정합니다. sub_sample=0.5로 지정하면 전체 데이터의 절반을 트리를 생성하는 데 사용합니다.
	feature_fraction	colsample_bytree	GBM의 max_features와 유사합니다. 트리 생성에 필요한 피처(컬럼)을 임의로 샘플링 하는 데 사용됩니다. 매우 많은 피처가 있는 경우 과적합을 조정하는 데 적용합니다

LightGBM 사이킷런 래퍼는 XGBoost 사이킷런 래퍼에 해당 하이퍼 파라미터가 있으면 이를 그대로 사용하고 그렇지 않으면 파이썬 래퍼 LightGBM 하이퍼 파라미터를 사용.

# LightGBM 하이퍼 파라미터 설명

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	
파라미터명	lambda_l2	reg_lambda	L2 규제(Regularization) 적용 값. 기본값은 1임. 값이 클 수록 규제 값이 커짐. 과적합 제어
	lambda_l1	reg_alpha	L1 규제(Regularization) 적용 값. 기본값은 0임. 값이 클 수록 규제 값이 커짐. 과적합 제어
	early_stopping_round	early_stopping_rounds	학습 조기 종료를 위한 early stopping interval 값
	num_leaves	num_leaves	최대 리프노드 갯수
	min_sum_hessian_in_leaf	min_child_weight	결정트리의 min_child_leaf와 유사. 과적합 조절용

num\_leaves의 개수를 중심으로 min\_child\_samples(min\_data\_in\_leaf), max\_depth를 함께 조정하면서 모델의 복잡도를 줄이는 것이 기본 튜닝 방안

# 파이썬 래퍼와 사이킷런 래퍼 하이퍼 파라미터 비교

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
파라미터명	num_iterations	n_estimators	n_estimators
	learning_rate	learning_rate	learning_rate
	max_depth	max_depth	max_depth
	min_data_in_leaf	min_child_samples	N/A
	bagging_fraction	subsample	subsample
	feature_fraction	colsample_bytree	colsample_bytree
	lambda_l2	reg_lambda	reg_lambda
	lambda_l1	reg_alpha	reg_alpha
	early_stopping_round	early_stopping_rounds	early_stopping_rounds
	num_leaves	num_leaves	N/A
	min_sum_hessian_in_leaf	min_child_weight	min_child_weight

# 하이퍼 파라미터 튜닝 수행 방법

Grid Search

Random Search

Bayesian  
Optimization

수동 튜닝

# 하이퍼 파라미터 튜닝의 주요 이슈

- Gradient Boosting 기반 알고리즘은 튜닝 해야 할 하이퍼 파라미터 개수가 많고 범위가 넓어서 가능한 개별 경우의 수가 너무 많음.
- 이러한 경우의 수가 많을 경우 데이터가 크면 하이퍼 파라미터 튜닝에 굉장히 오랜 시간이 투입되어야 함.

# Grid Search 와 Random Search의 주요 이슈

GridSearchCV(classifier, params, cv=3)

RandomizedSearch(classifier, params, cv=3, n\_iter=10)

```
params = {  
    'max_depth' = [10, 20, 30, 40, 50],  
    'num_leaves'= [ 35, 45, 55, 65],  
    'colsample_bytree'=[0.5, 0.6, 0.7, 0.8, 0.9, 1.0],  
    'subsample'= [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],  
    'max_bin'= [100, 200, 300, 400]  
    'min_child_weight'= [10, 20, 30, 40]  
}
```

- GridSearchCV는 수행 시간이 너무 오래 걸림. 개별 하이퍼 파라미터들을 Grid 형태로 지정하는 것은 한계가 존재(데이터 세트가 작을 때 유리)
- RandomizedSearch는 수행 시간은 줄여 주지만, Random한 선택으로 최적 하이퍼 파라미터 검출에 태생적 제약(데이터 세트가 클 때 유리)
- 두가지 방법 모두 Iteration 중에 어느정도 최적화된 하이퍼 파라미터들을 활용하면서 최적화를 수행할 수 없음.

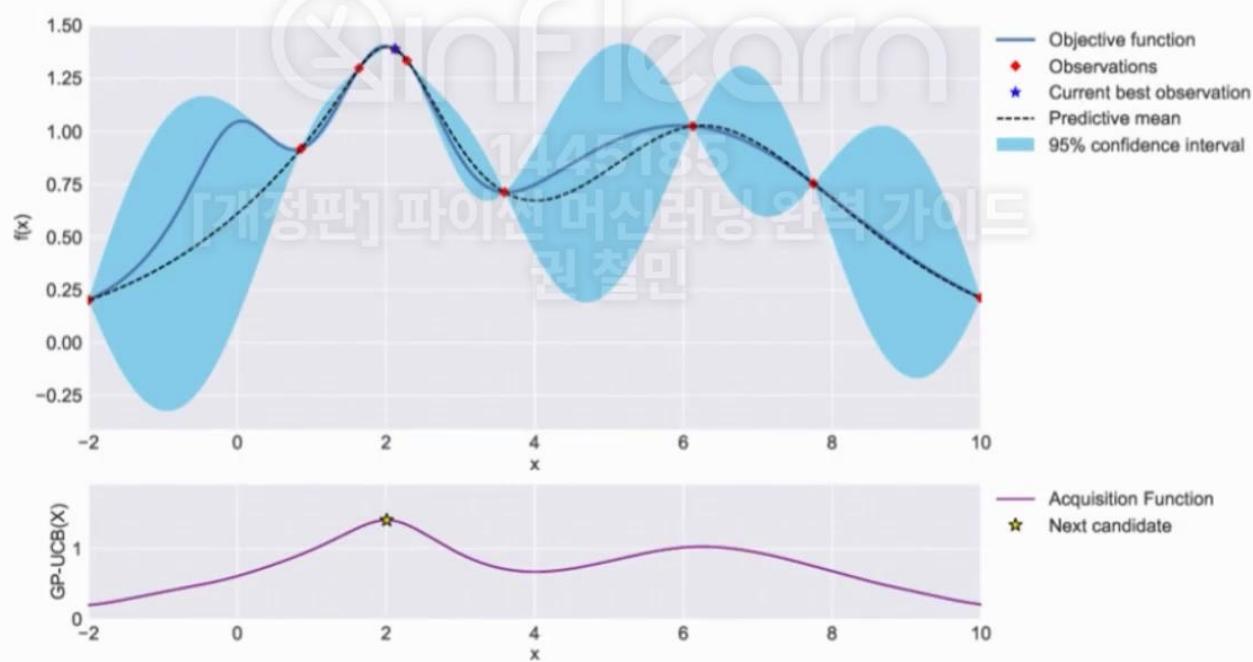
# Bayesian 최적화가 필요한 순간



- 가능한 최소의 시도로 최적의 답을 찾아야 할 경우
- 개별 시도가 너무 많은 시간/자원이 필요할 때

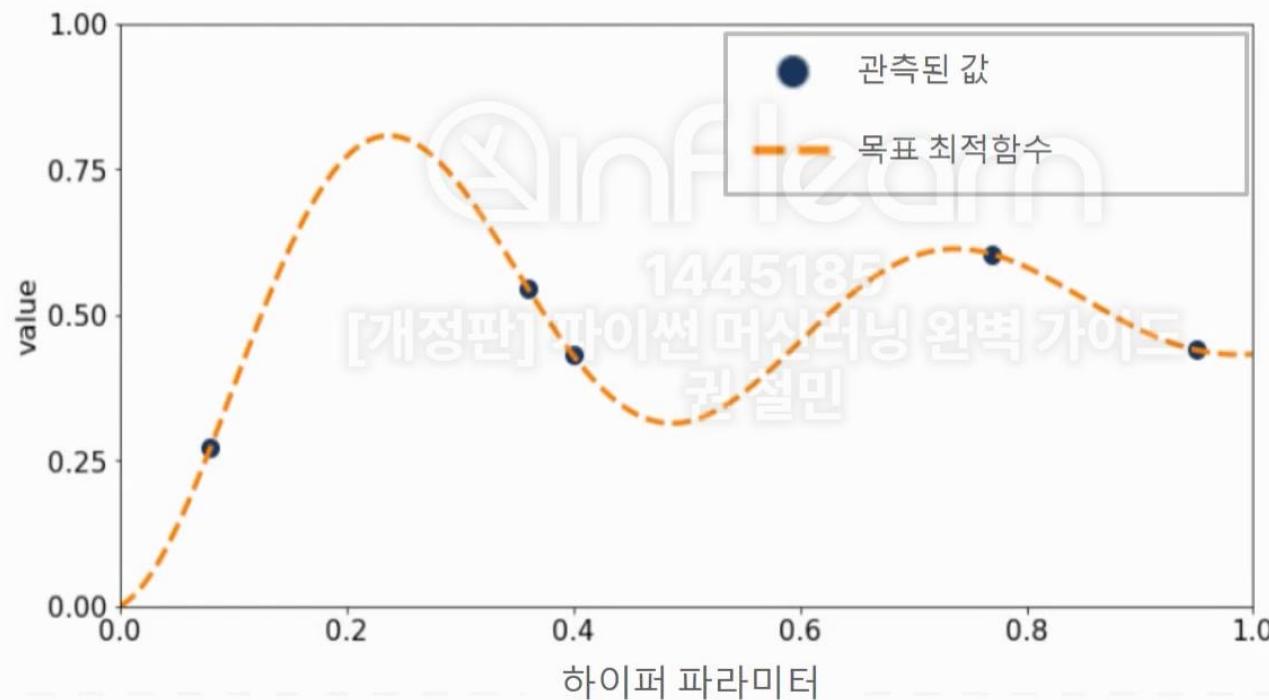
# 베이지안 최적화(Bayesian Optimization) 개요

- 베이지안 최적화는 미지의 함수가 반환하는 값의 최소 또는 최대값 만드는 최적해를 짧은 반복을 통해 찾아내는 최적화 방식.
- 베이지안 최적화는 새로운 데이터를 입력 받았을 때 최적 함수를 예측하는 사후 모델을 개선해 나가면서 최적 함수를 도출
- 대체 모델(Surrogate Model)과 획득 함수로 구성되며, 대체 모델은 획득 함수로부터 최적 입력 값을 추천 받은 뒤 이를 기반으로 최적 함수 모델을 개선
- 획득 함수는 개선된 대체 모델을 기반으로 다시 최적 입력 값을 계산



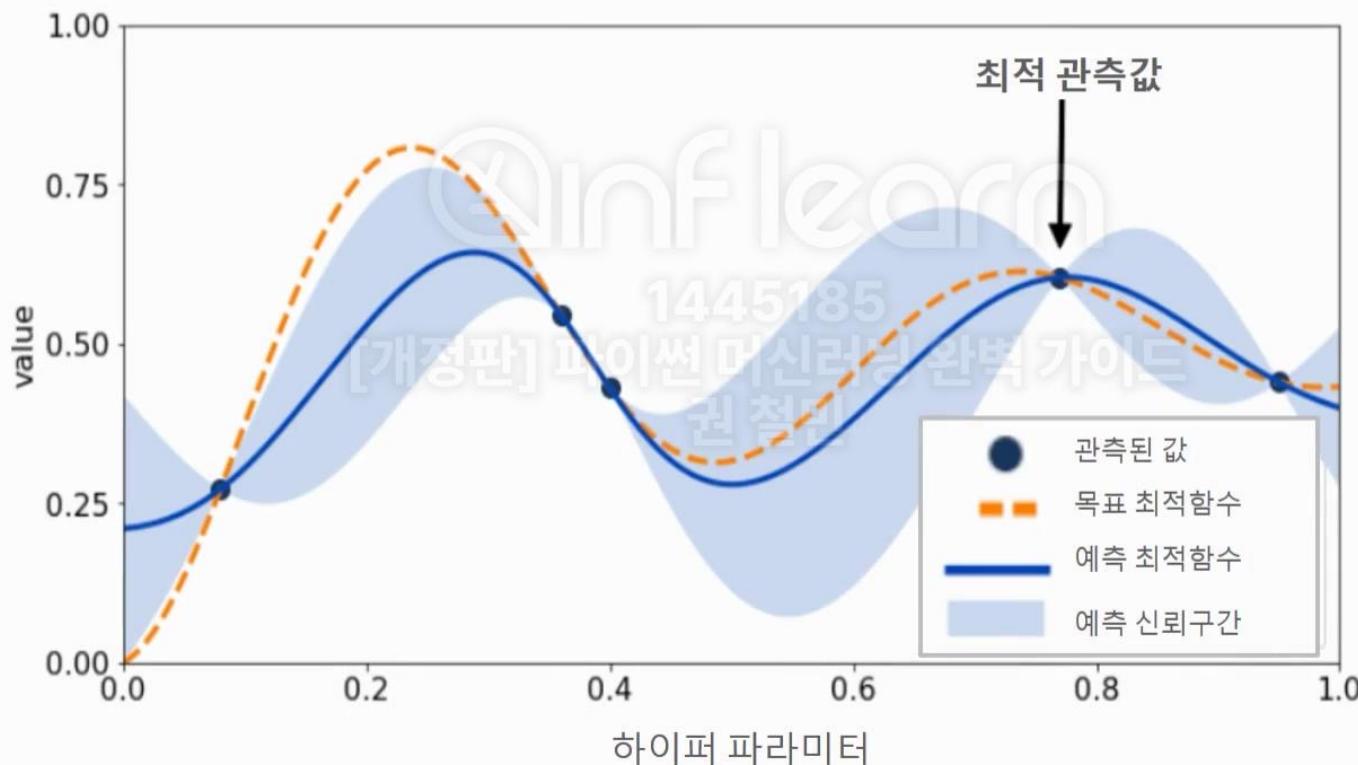
# 베이지안 최적화 수행 단계 – Step 1

Step 1: 최초에는 랜덤하게 하이퍼 파라미터들을 샘플링하여 성능 결과를 관측



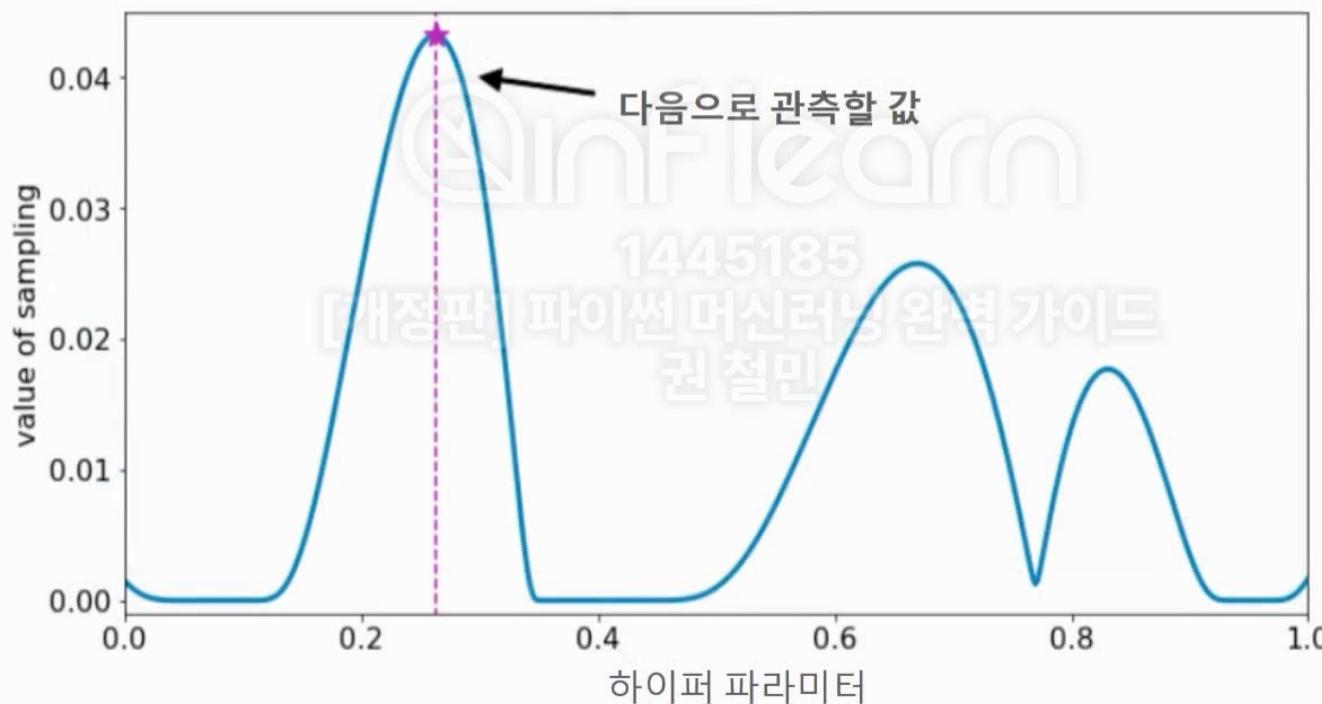
## 베이지안 최적화 수행 단계 – Step 2

Step 2: 관측된 값을 기반으로 대체 모델은 최적 함수를 예측 추정



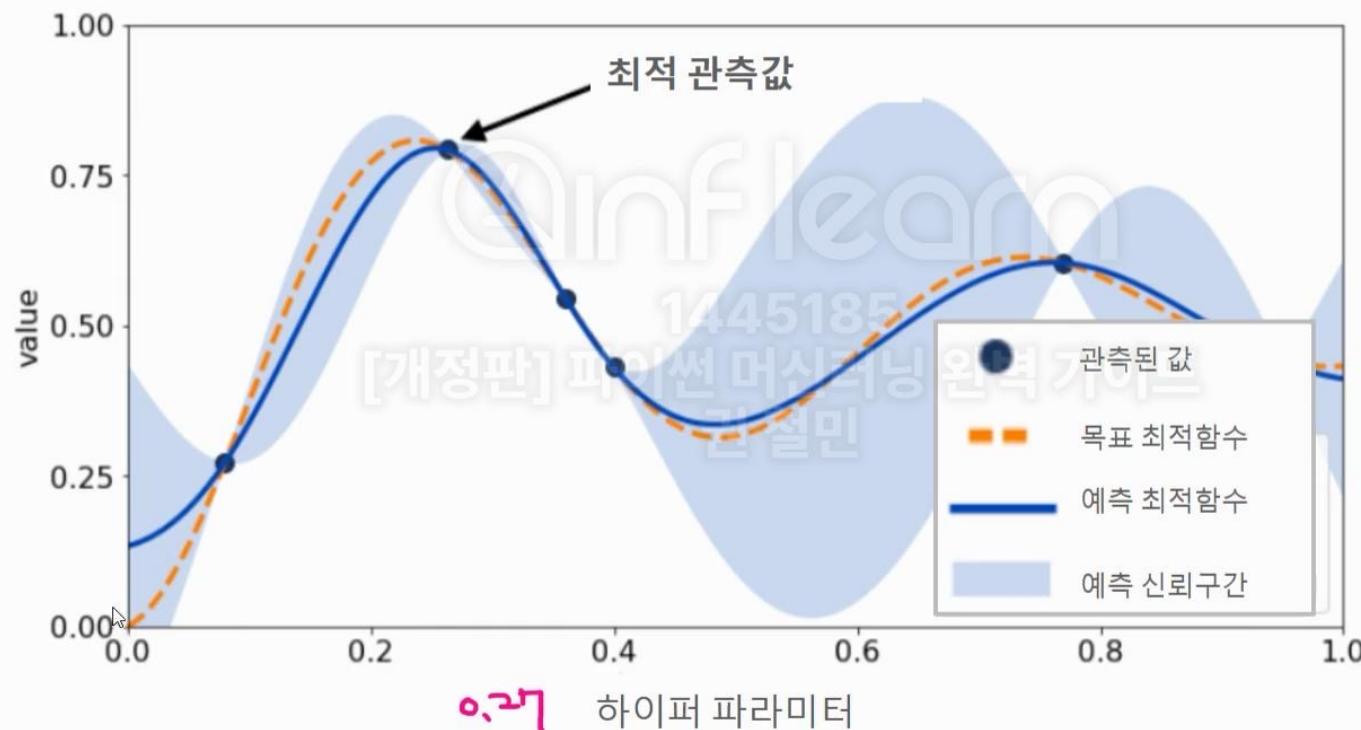
# 베이지안 최적화 수행 단계 – Step 3

Step 3: 획득 함수에서 다음으로 관측할 하이퍼 파라미터 추출



# 베이지안 최적화 수행 단계 – Step 4

Step 4: 해당 하이퍼 파라미터로 관측된 값을 기반으로 대체 모델은 다시 최적 함수 예측 추정



# 베이지안 최적화 구현 요소

## 입력 값 범위

```
Search_space = {'x': (-10, 10), 'y': (-15, 15)}
```

많은 X 입력 값



많은 Y 입력 값



## 함수

```
def black_box_function(x, y):  
    return -x ** 2 - 20*y
```

함수 반환 최소값 유추

```
fmin(fn=black_box_function, space=search_space,  
algo=tpe.suggest, max_evals=20, trials=trial_val)
```

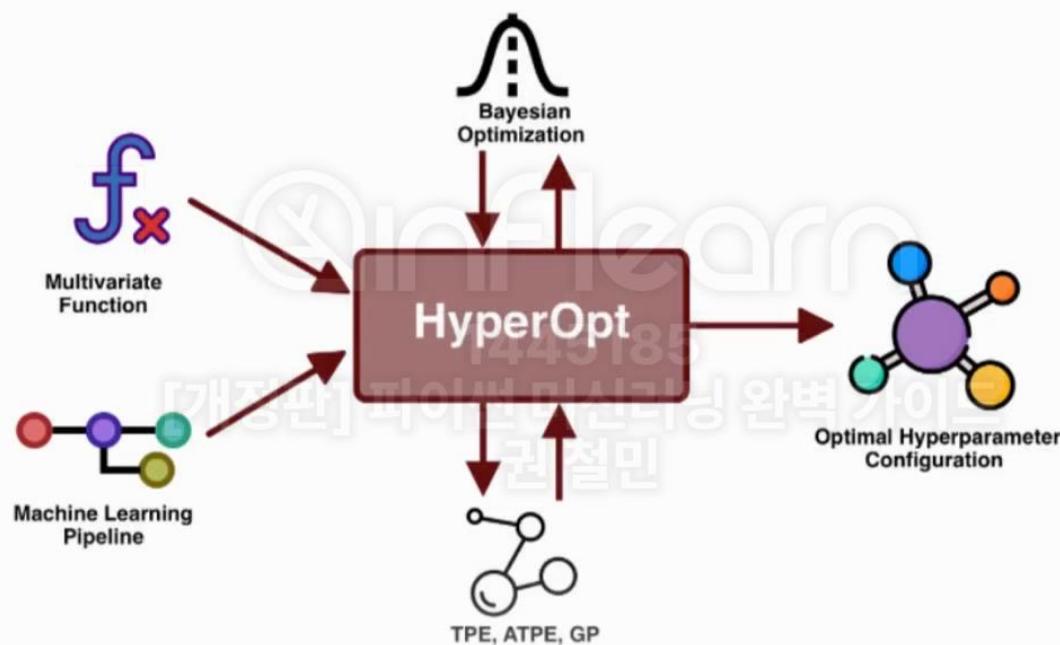
# 베이지안 최적화를 구현한 주요 패키지

HyperOpt

Bayesian  
optimization

Optuna

# 베이지안 최적화를 위한 HyperOpt



<https://towardsdatascience.com/hyperopt-hyperparameter-tuning-based-on-bayesian-optimization-7fa32dffaf29>

# HyperOpt를 통한 최적화 예시

## Search Space(입력 값 범위)

```
search_space = {'x': hp.quniform('x', 5, 15, 1),  
                'y': hp.uniform('y', 0.01, 0.1)}
```

많은 x 입력 값



많은 y 입력 값



## 목적 함수

```
def objective_func(search_space):  
    x = search_space['x']  
    y = search_space['y']  
    print('x:', x, 'y:', y)  
    return {'loss': x ** 2 + y * 20, 'status': STATUS_OK }
```

목적 함수 반환 최소값 유추

```
best = fmin(fn=objective_func, space=search_space, algo=algo,  
            max_evals=5, trials=trials)
```

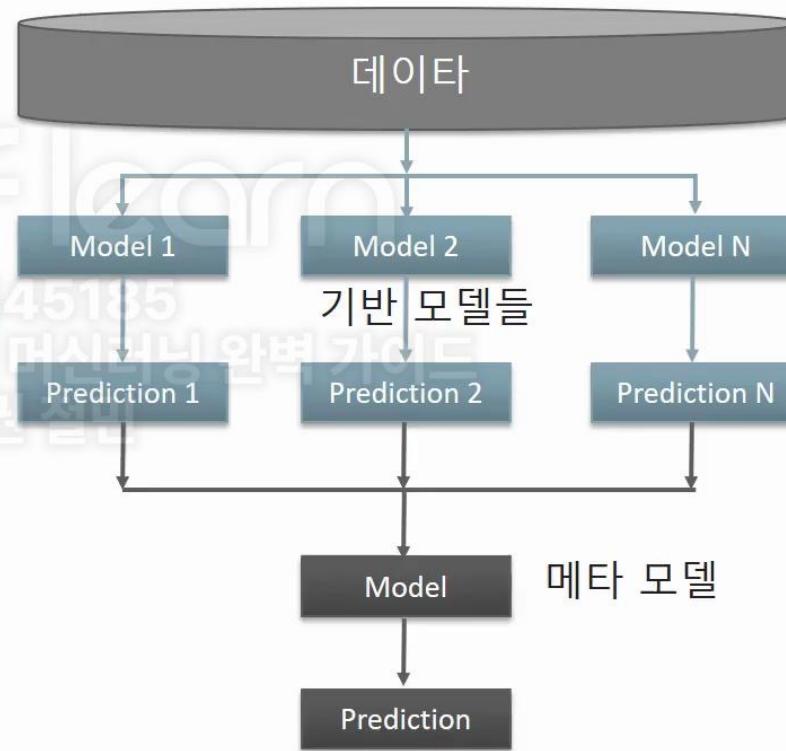
```
[{'loss': 81.20833131199375},  
 {'loss': 169.20757538485393},  
 {'loss': 121.10536542037384},  
 {'loss': 64.08021188657003},  
 {'loss': 81.42067134007004}]
```

# HyperOpt의 주요 구성 요소

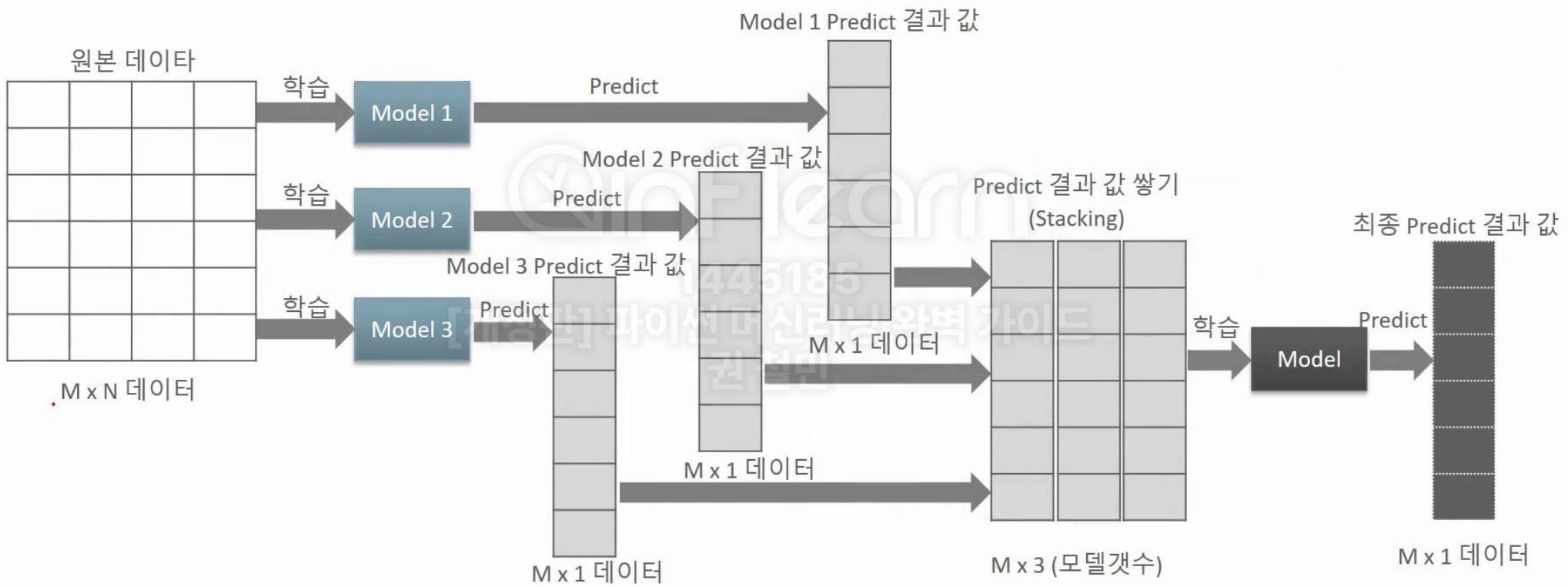
구성 요소	설명
search space (입력값 범위)	<ul style="list-style-type: none"><li>여러개의 입력 변수들과 이들 값의 범위를 지정</li><li>hp.quniform(label, low, high, q) : label로 지정된 입력 값 변수 검색 공간을 최소값 low에서 최대값 high까지 q의 간격을 가지 고 설정.</li><li>hp.uniform(label, low, high): 최소값 low에서 최대값 high 까지 정규 분포 형태의 검색 공간 설정</li><li>hp.randint(label, upper): 0부터 최대값 upper까지 random한 정수 값으로 검색 공간 설정.</li><li>hp.loguniform(label, low, high): <math>\exp(\text{uniform}(\text{low}, \text{high}))</math> 값을 반환하며, 반환 값의 log 변환 된 값은 정규 분포 형태를 가지는 검 색 공간 설정.</li></ul>
목적 함수	search space를 입력 받아 로직에 따라 loss값을 계산하고 이를 반환하는 함수. 반드시 dictionary 형태의 값을 반환하고 여기에 'loss':loss값이 기재되어야 함.
목적 함수의 최소값을 찾는 함수	<ul style="list-style-type: none"><li>목적 함수를 실행하여 최소 반환값(loss)를 최적으로 찾아 내는 함수</li><li>Bayesian 최적화 기법으로 입력 변수들의 search space상에서 정해진 횟수만큼 입력 변수들을 입력 하여 목적 함수의 반환 값(loss)을 최적으로 찾아냄</li><li>hyperopt는 이를 위해 fmin() 함수를 제공.</li><li>fmin() 함수의 인자로 목적함수, search space, 베이지안 최적화 기법유형, 최적화 시도횟수, 최적화 로그 기록 객체를 인자 로 넣어줌. <code>best = fmin(objective, space=hp.uniform('x', -10, 10), algo=tpe.suggest, max_evals=100, trials=trials)</code></li></ul>

# Basic Stacking Model – Diagram

기반 모델들이 예측한 값들을 Stacking 형태로  
만들어서 메타 모델이 이를 학습하고 예측하는 모델



# Basic Stacking Model – Example



# 교차 검증 세트 기반의 스태킹

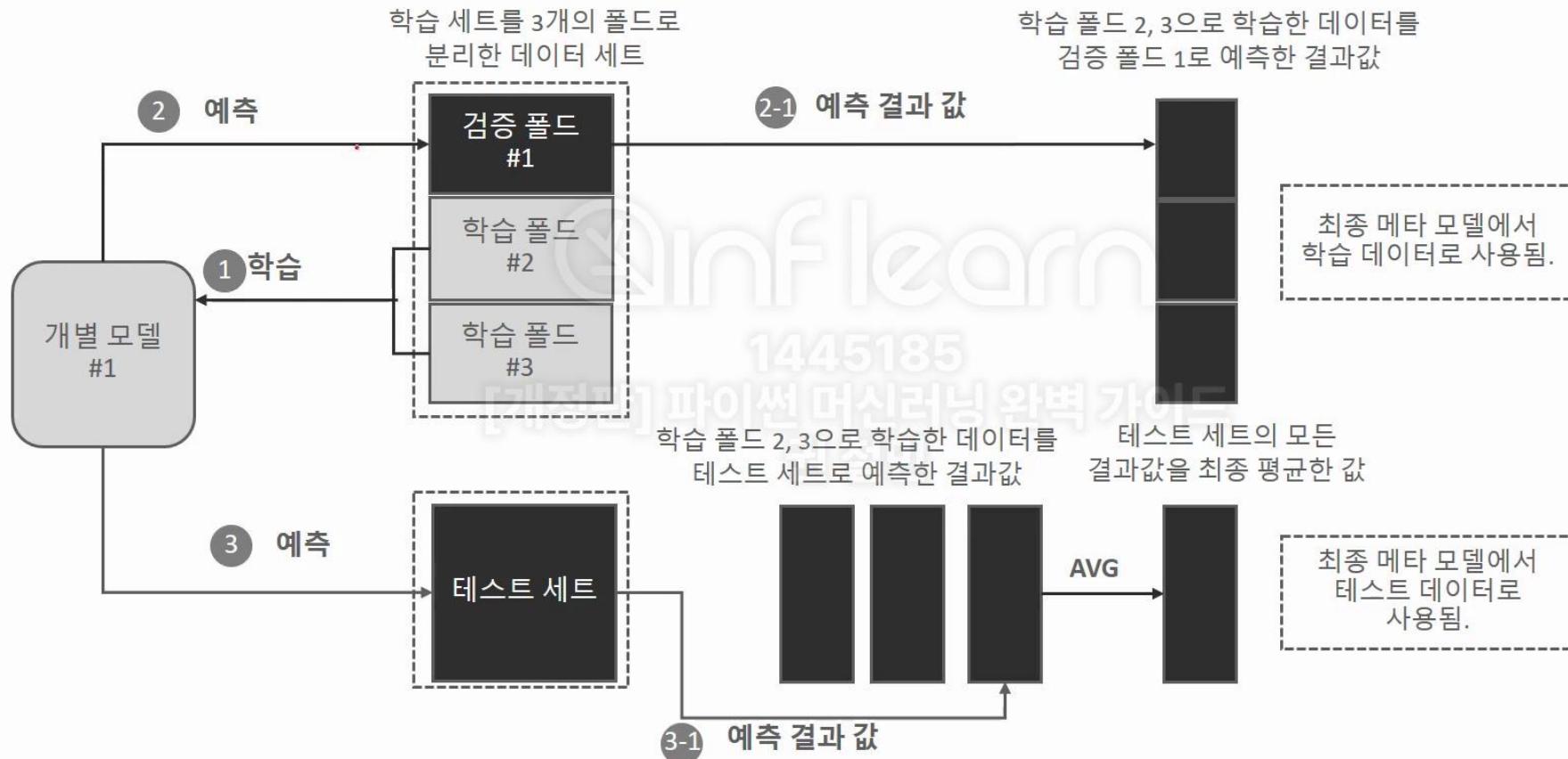
교차 검증 세트 기반의 스태킹은 이에 대한 개선을 위해 개별 모델들이 각각 교차 검증으로 메타 모델을 위한 학습용 스태킹 데이터 생성과 예측을 위한 테스트용 스태킹 데이터를 생성한 뒤 이를 기반으로 메타 모델이 학습과 예측을 수행합니다. 이는 다음과 같이 2단계의 스텝으로 구분될 수 있습니다.

- 스텝 1: 각 모델별로 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터를 생성합니다.
- 스텝 2: 스텝 1에서 개별 모델들이 생성한 학습용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 최종 학습용 데이터 세트를 생성합니다. 마찬가지로 각 모델들이 생성한 테스트용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 예측할 최종 테스트 데이터 세트를 생성합니다. 메타 모델은 최종적으로 생성된 학습 데이터 세트와 원본 학습 데이터의 레이블 데이터를 기반으로 학습한 뒤, 최종적으로 생성된 테스트 데이터 세트를 예측하고, 원본 테스트 데이터의 레이블 데이터를 기반으로 평가합니다

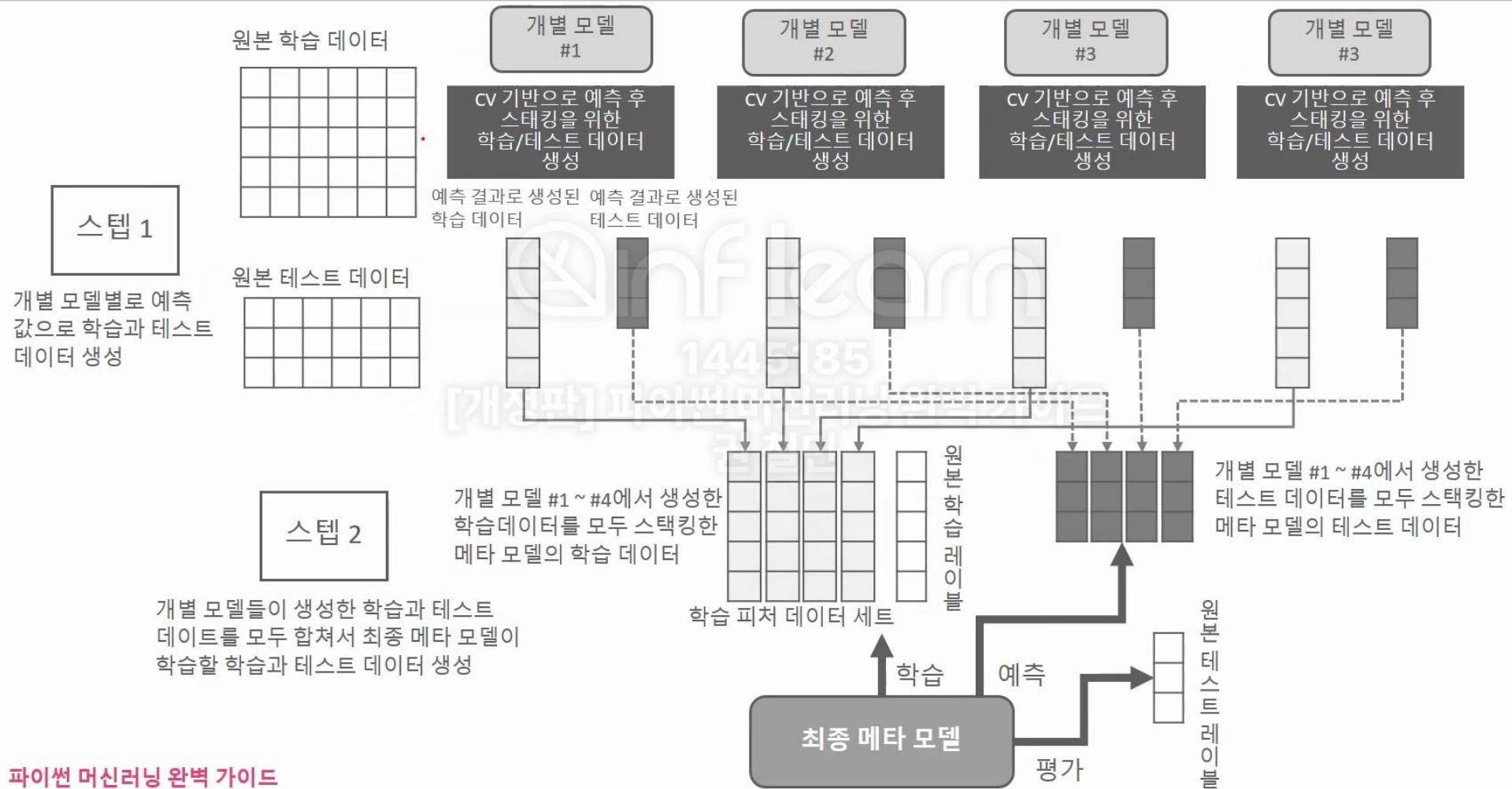
# 교차 검증 세트 기반 스태킹 모델 만들기 – K Fold 첫번째



# 교차 검증 세트 기반 스태킹 모델 만들기 – K Fold 세번째



# 교차 검증 세트 기반 스태킹 모델 만들기



## Feature Selection

---

- 모델을 구성하는 주요 피처들을 선택
  - 불 필요한 다수의 피처들로 인해 모델 성능을 떨어뜨릴 가능성 제거
  - 설명 가능한 모델이 될 수 있도록 피처들을 선별

1445185  
[개정판] 파이썬 머신러닝 완벽 가이드  
권철민

# Feature Selection 유형

- 피처값의 분포, Null, 피처간 높은 상관도, 결정값과의 독립성 등을 고려
- 모델의 피처 중요도(Feature importance) 기반



# Feature Selection 유형

- 피처값의 분포, Null, 피처간 높은 상관도, 결정값과의 독립성 등을 고려
- 모델의 피처 중요도(Feature importance) 기반



# 사이킷런 Feature Selection 지원

- **RFE(Recursive Feature Elimination)**

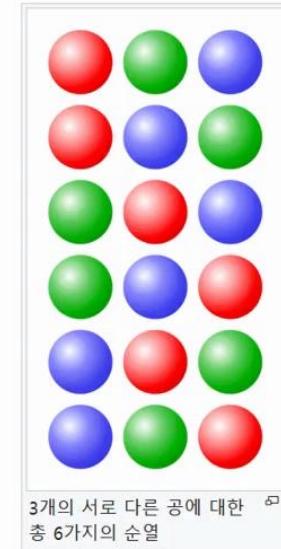
- 모델 최초 학습 후 Feature 중요도 선정
  - feature 중요도가 낮은 속성들을 차례로 제거해 가면서 반복적으로 학습/평가를 수행하여 최적 feature 추출
  - 수행시간이 오래 걸리고, 낮은 속성들을 제거해 나가는 메커니즘이 정확한 Feature Selection을 찾는 목표에 정확히 부합하지 않을 수 있음

- **SelectFromModel**

- 모델 최초 학습 후 선정된 Feature 중요도에 따라 평균/중앙값의 특정 비율 이상인 Feature들을 선택

# Permutation(순열) importance 개요

- 특정 피처들의 값을 완전히 변조했을 때 모델 성능이 얼마나 저하되는지를 기준으로 해당 피처의 중요도를 산정
- 학습 데이터를 제거하거나/변조하면 다시 재 학습을 수행해야 하므로 수행 시간이 오래 걸림.
- 일반적으로 테스트 데이터(검증 데이터)에 특정 피처들을 반복적으로 변조한 뒤 해당 피처의 중요도를 평균적으로 산정



# Permutation importance 프로세스

검증(테스트) 데이터 세트에서 ftr1을 K 번 random shuffling

ftr1	ftr2	ftr3
A	가	
B	나	
C	다	

ftr1	ftr2	ftr3
B	가	
A	나	
C	다	

.....

ftr1	ftr2	ftr3
C	가	
A	나	
B	다	

- Ftr1이 중요 feature이면 evaluation 성능이 감소
- 원본 evaluation score에서 평균적으로 얼마나 성능이 감소했는지 계산해서 ftr1의 평균 evaluation score 계산

검증(테스트) 데이터 세트에서 ftr2을 K 번 random shuffling

ftr1	ftr2	ftr3
A	가	
B	나	
C	다	

ftr1	ftr2	ftr3
A	나	
B	가	
C	다	

.....

ftr1	ftr2	ftr3
A	다	
B	나	
C	가	

## Permutation importance 프로세스

- 원본 모델의 기준 평가 성능을 설정.
- 개별 feature 별로 아래 수행
  1. 설정된 iteration값 별로 아래 수행
    - a. 해당 feature로 shuffle
    - b. 모델 성능 평가
  2. 기준 평가 성능에서 모델 성능이 얼마나 저하되었는지 평가

## 왜 feature importance 는 절대적인 feature selection 기준이 될 수 없는가?

- Feature importance는 최적 tree 구조를 만들기 위한 피처들의 impurity가 중요 기준임. 결정 값과 관련이 없어도 feature importance가 높아 질 수 있음.
- Feature importance는 학습 데이터를 기반으로 생성됨. 테스트 데이터에서는 달라질 수 있음
- Feature importance는 number형의 높은 cardinality feature에 biased 되어 있음

# 분류(Classification) Summary

- 결정 트리와 결정 트리 기반의 앙상블
- 배깅과 부스팅
  - 랜덤 포레스트, GBM
- GBM의 기능을 더욱 향상 시킨 XGBoost, LightGBM
- 스태킹 모델



1445185

[개정판] 파이썬 머신러닝 완벽 가이드

권철민