

BITAmin

11주차 정규 Session

비타민 6기 3조

강혜연 이민준 정세영 정인영

CONTENTS

- 01. 배깅(Bagging)
- 02. 부스팅(Boosting)
- 03. 에이다부스트(AdaBoost)
- 04. GBM
- 05. XGBoost
- 06. LightGBM

01. 배깅(Bagging)

01. 배깅(Bagging)

※ 10주차 내용 복습

앙상블이란?

- 여러 개의 분류기를 생성하고 그 예측을 결합하는 학습 방법
⇒ 보다 정확한 최종 예측 도출이 가능
- 다양한 분류기의 예측 결과를 결합함으로써 단일 분류기보다 신뢰성이 높은 예측값을 얻는 것을 목적으로 함
⇒ 집단 지성과 같은 효과를 얻을 수 있음 (wisdom of crowd)
⇒ 하나의 강한 머신러닝 알고리즘보다 여러 개의 약한 머신러닝 알고리즘이 낫다

01. 배깅(Bagging)

※ 10주차 내용 복습

배깅과 부스팅은 모두 앙상블을 이용한 머신러닝 방법임.

앙상블이란?

따라서, 둘 다 여러 개의 모델을 학습시킴으로써 하나의 단일

- 여러 개의 분류기 모델에서는 얻을 수 없는 성능과 안정성을
⇒ 보다 정확한 최종 예측이 가능
이끌어낼 수 있다는 공통점이 있음.

- 다양한 분류기의 예측 결과를 결합함으로써 단일 분류기보다 신뢰성이 높은

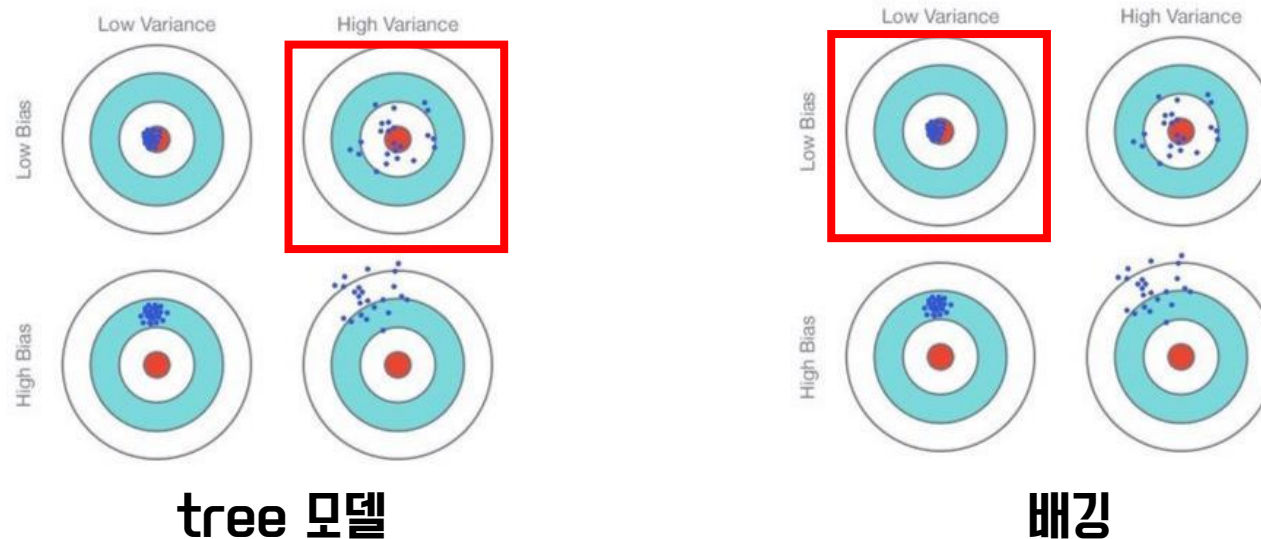
하지만, 여러 개의 모델을 어떻게 학습시키고 학습된 모델을
예측에 어떻게 활용하는지에 대한 접근 방법에서 차이가 있음.

⇒ 하나의 강한 머신러닝 알고리즘보다 여러 개의 약한 머신러닝 알고리즘이 있다

01. 배깅(Bagging)

배깅(Bagging)

- Bootstrap Aggregation의 약자
- 샘플을 여러 번 뽑아(Bootstrap) 각 모델을 학습시켜 결과물을 집계(Aggregation)하는 방법
⇒ 중복 추출 가능
- 장점
 - 1) tree 모델은 깊이 성장할수록 오버피팅이 심해져서 편향(bias)이 감소되고 분산(variance)은 증가함.
이에 비해 배깅은 tree 모델을 결합하여 편향은 유지하지만 전체적인 분산은 감소시켜 오버피팅 ↓



01. 배깅(Bagging)

배깅(Bagging)

- 2) 여러 개의 weak learners의 결과를 합쳐 하나의 strong learner가 만들어지도록 함
- 3) 대규모의 데이터를 사용할 때 용이
- 4) 데이터셋의 결측치가 알고리즘의 성과에 영향을 주지 않음

- 단점

- 1) 중복 추출로 인해 어떤 샘플은 사용되지 않고 어떤 샘플은 여러 번 사용되어 편향될 가능성이 있음
 - ** OOB (Out-Of-Bag) 샘플 : 샘플링 되지 않은 나머지 샘플**
- 2) Tree 모델의 장점이었던 모형 해석 능력이 없어짐. Tree모델은 시각화를 통해 직관적으로 분리 과정을 볼 수 있었지만 배깅은 그런 과정을 볼 수 없음.

- 대표적인 모델 : 랜덤포레스트(디시전트리에 배깅을 적용)

편향(Bias)과 분산(Variance)

사람에 비유를 하면 고집이 센 사람은 bias가, 귀가 얇은 사람은 variance가 크다는 것과 같음.

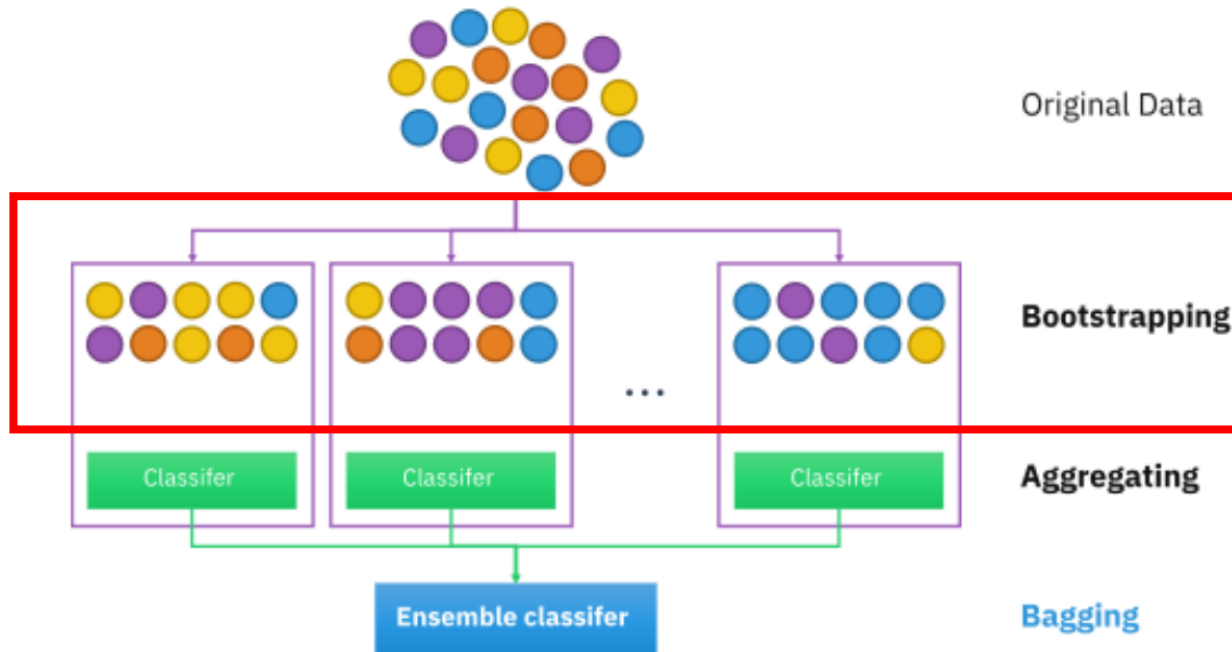
Bias가 크고 variance는 작음 => underfit
Variance가 크고 bias는 작음 => overfit

모델을 학습 시킬 때 우리의 목표는 bias와 variance가 모두 최소화되도록 하는 것.
그러나 일반적으로 이 둘은 동시에 최소화될 수 없는데, 이러한 현상을 bias-variance tradeoff (편향-분산 트레이드오프)라고 한다.

01. 배깅(Bagging)

배깅(Bagging) Steps

1. 먼저, 데이터로부터 복원 랜덤 샘플링(부트스트랩)을 한다. 이렇게 추출된 데이터가 표본 집단이 된다.



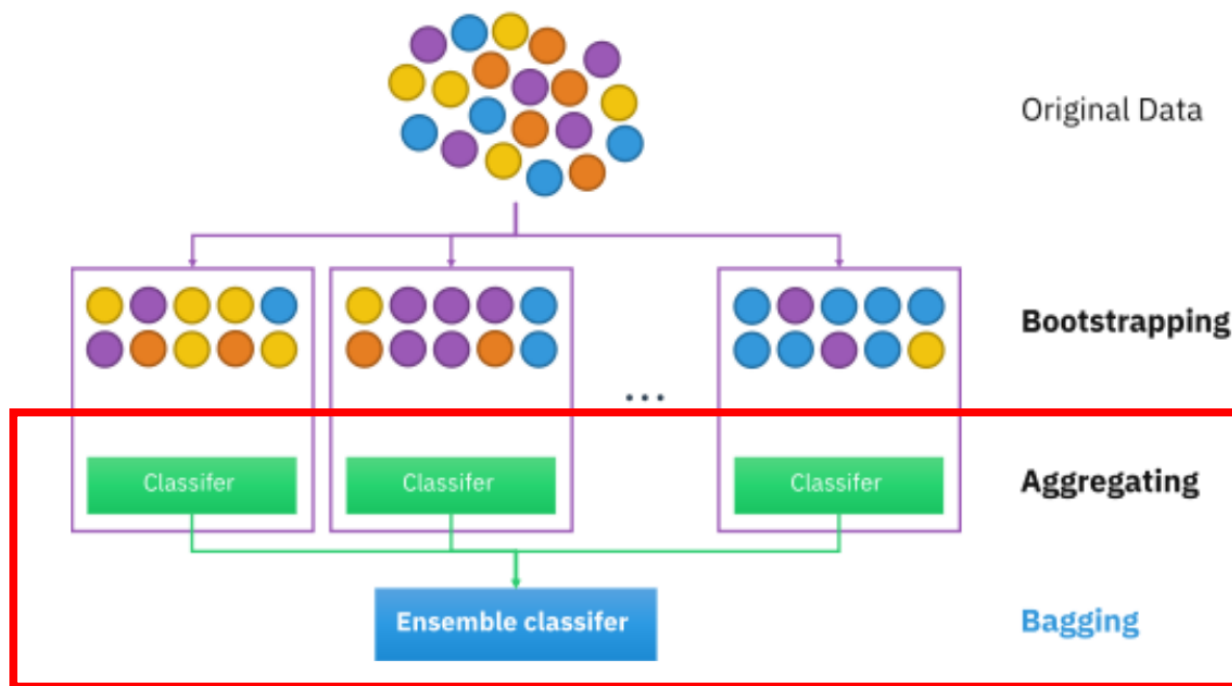
◀ 학습을 시킬 때 사용할 데이터들을 무작위로 복원 추출하여 각각의 샘플을 만들.

하나의 샘플에 같은 데이터가 들어갈 수도 있고, 각각 다른 데이터들이 들어갈 수 있음. (복원추출이기 때문에)

01. 배깅(Bagging)

배깅(Bagging) Steps

2. 이렇게 추출한 데이터로 모델을 학습시킨다. 그리고 학습된 모델의 결과를 집계하여 최종 결과 값을 구한다.



◀ 이렇게 만들어진 샘플들을 학습시켜 각각의 모델을 만들. 이때 각 모델은 독립적임.

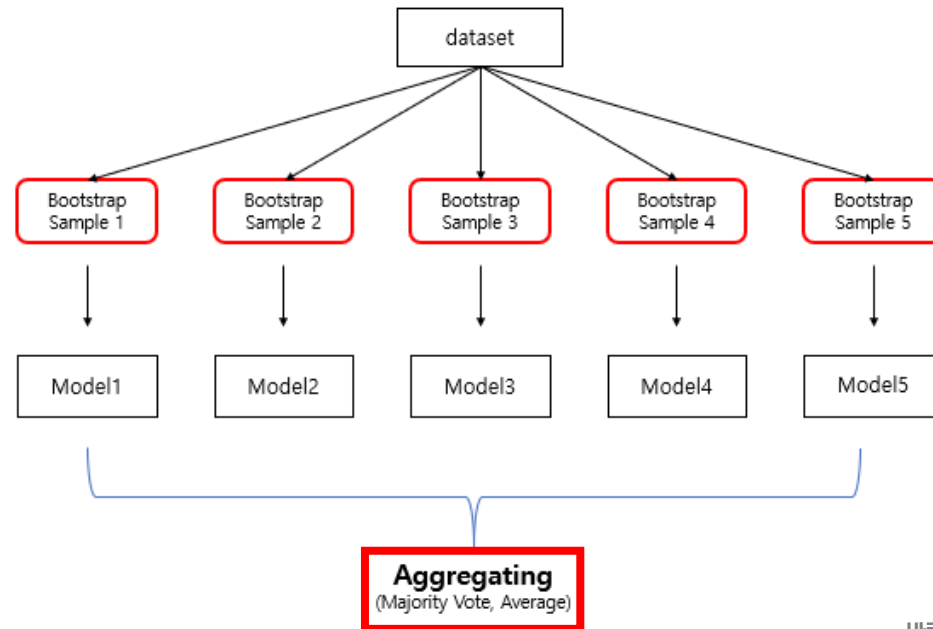
그리고 이 각각의 모델들의 결과를 집계하여 최종 결과 값을 만들어 냄.

01. 배깅(Bagging)

배깅(Bagging) Steps

이때,

- 범주형 데이터의 경우 보팅(voting) 방식으로 결과를 집계
: 하드 보팅(hard voting) or 소프트 보팅(soft voting)
- 연속형 데이터의 경우 평균으로 집계
: 각각의 결정 트리 모델이 예측한 값에 평균을 취해 최종 결과를 예측



02. 부스팅(Boosting)

02. 부스팅(Boosting)

부스팅(Boosting)

- 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식
- 처음 모델이 예측을 하면 그 예측 결과에 따라 데이터에 가중치가 부여되고, 부여된 가중치가 다음 모델에 영향을 줌.
- 잘못 분류된 데이터에 집중하여 새로운 분류 규칙을 만드는 단계를 반복함.
ex) 수학 문제를 푸는데 9번 문제를 계속 틀렸다고 가정했을 때, boosting 방식은 9번 문제에 가중치를 부여하여 9번 문제를 잘 맞춘 모델을 최종 모델로 설정함
- 배깅이 일반적인 모델을 만드는 데에 초점이 맞춰져 있다면, 부스팅은 맞추기 어려운 문제를 맞추는 데에 초점이 맞춰져 있음.
- 대표적인 알고리즘 : 에이다 부스트(AdaBoost), 그래디언트 부스트(GBM)

02. 부스팅(Boosting)

부스팅(Boosting)

- 장점

- 1) 배경에 비해 error가 적음 = 성능이 좋음
- 2) 알고리즘을 읽고 해석하기 쉬움-예측에 대한 해석을 다루기 쉬움

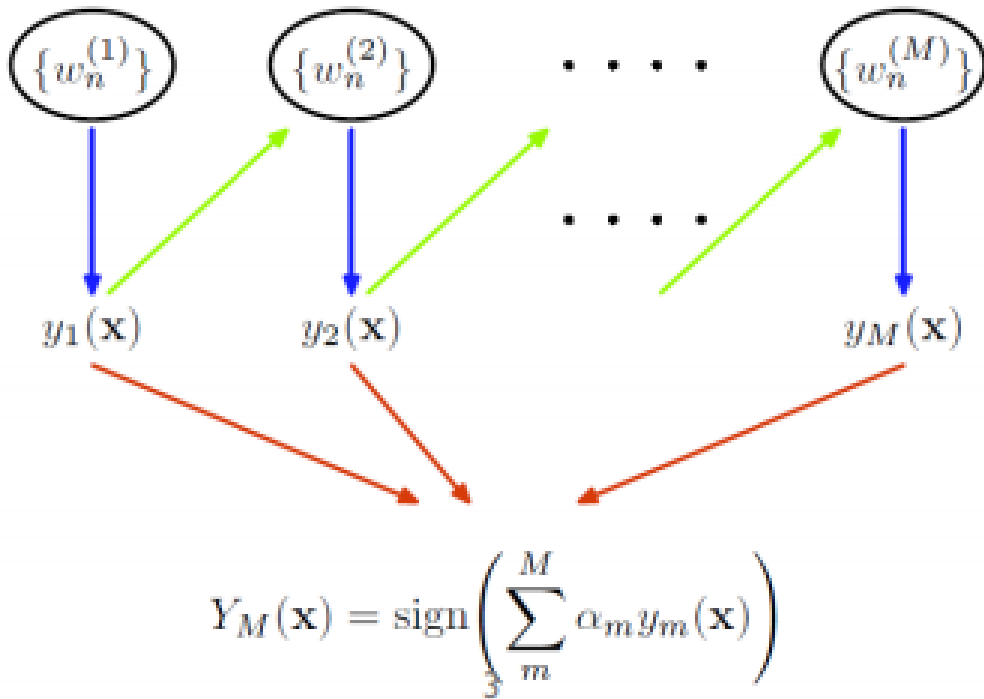
- 단점

- 1) 속도가 느리고 오버피팅(과적합) 될 가능성이 있음
 - 잘못 분류된 값들에 가중치를 부여하며 모델을 만들어가다 보면 train set에 최적화 되어 실제 test set에 적용했을 때 예측률을 감소시키는 과적합 문제가 발생할 수 있음
- 2) 모든 estimator가 이전 모델의 정확성에 기반하기 때문에 확장시키기 어려운 방법임

⇒ 따라서 두 방법 중 하나를 선택할 땐 개별 의사결정 나무의 성능이 낮은 것이 문제라면 부스팅 방법을, 개별 의사결정 나무의 오버 피팅이 문제라면 배깅 방법이 적합하다고 할 수 있음

02. 부스팅(Boosting)

부스팅(Boosting)



- 1) 한 round 당 하나의 모델을 순차적으로 학습시킴
- 2) 각 round의 마지막에는 오분류된 객체들이 새로운 train set에서 더 많이 나올 수 있게 가중치를 조절
- 3) 앞선 모델에서 나타난 에러는 다음 턴에 나오는 모델이 그 에러를 줄이기 위한 데이터 셋으로 학습될 수 있게 함

02. 부스팅(Boosting)

배깅(Bagging) vs 부스팅(Boosting) 정리

A sample of a single classifier on an imaginary set of data.	
(Original) Training Set	
Training-set-1:	1, 2, 3, 4, 5, 6, 7, 8

A sample of Bagging on the same data.	
(Resampled) Training Set	
Training-set-1:	2, 7, 8, 3, 7, 6, 3, 1
Training-set-2:	7, 8, 5, 6, 4, 2, 7, 1
Training-set-3:	3, 6, 2, 7, 5, 6, 2, 2
Training-set-4:	4, 5, 1, 4, 6, 4, 3, 8

A sample of Boosting on the same data.	
(Resampled) Training Set	
Training-set-1:	2, 7, 8, 3, 7, 6, 3, 1
Training-set-2:	1, 4, 5, 4, 1, 5, 6, 4
Training-set-3:	7, 1, 5, 8, 1, 8, 1, 4
Training-set-4:	1, 1, 6, 1, 1, 3, 1, 5

[Bagging]

데이터를 샘플링 할 때 training set의 순서, 크기와 상관없이 완벽하게 랜덤하게 추출

[Boosting]

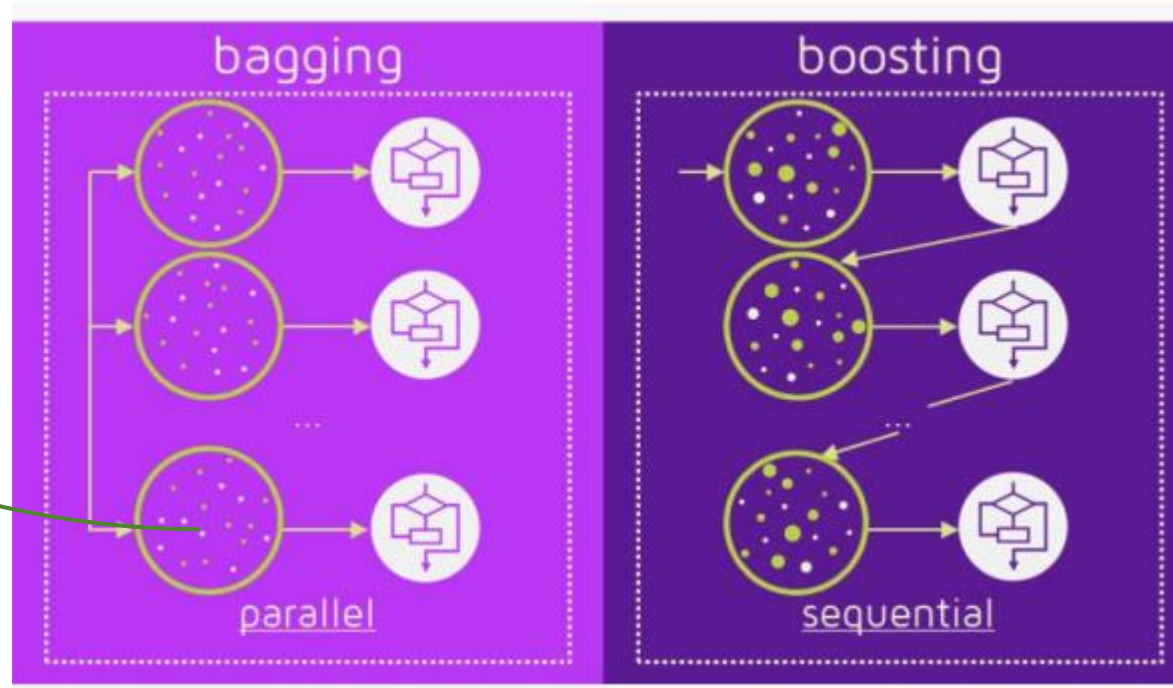
'1'의 비중치가 점점 높아지고 있음

⇒ h_1, h_2, h_3 는 모두 1번 객체를 정분류하지 못했을 가능성이 높음

02. 부스팅(Boosting)

배깅(Bagging) vs 부스팅(Boosting) 정리

동그라미 크기가 다 같음
=데이터가 선택될 확률이 똑같음
(uniform distribution)



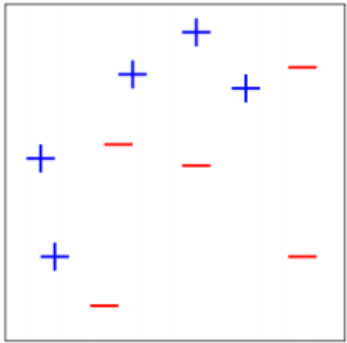
▲ 병렬적으로(parallel) 실행 가능

▲ 순차적으로(sequential) 실행해야 함
⇒ 앞선 모델이 잘 수행했는지 못했는지 평가한 후 그 다음 모델이 학습되어야 하기 때문

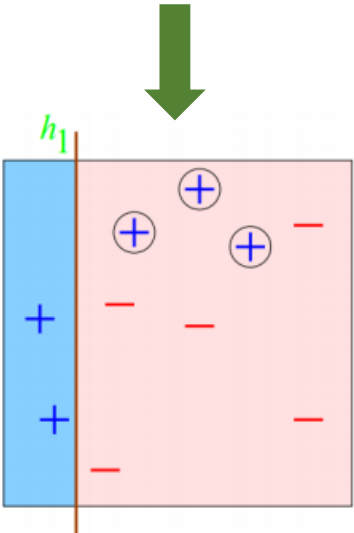
03. 에이다 부스트(AdaBoost)

03. 에이다 부스트(AdaBoost)

부스팅-에이다 부스트(AdaBoost) Steps



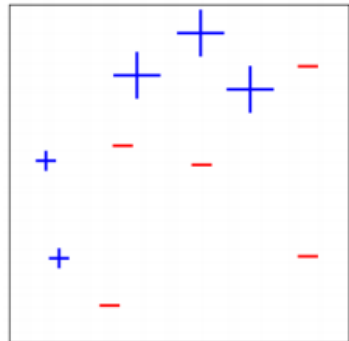
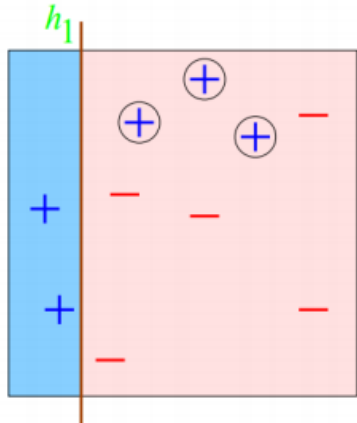
[Step 1] Sampling된 train set을 이용해서 하나의 weak learner(h_1) 생성



- Boosting에서 weak learner는 random guessing(무작위 추측)보다 조금 더 나은 수준의 model을 의미
- 일반적으로 split을 한 번만 하는 tree 모형(stump tree)

03. 에이다 부스트(AdaBoost)

부스팅-에이다 부스트(AdaBoost) Steps

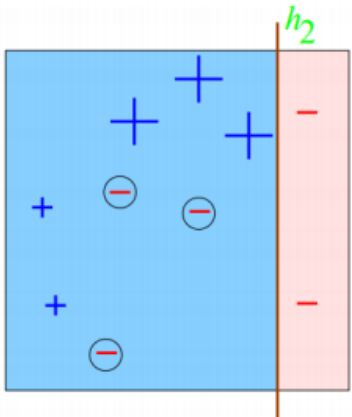
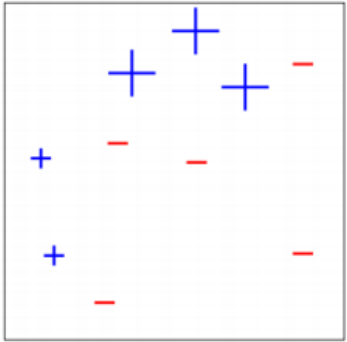


[Step 2] 오분류된 개체에 대해 가중치를 부여하여 다음 데이터 sampling 때 더 많이 뽑힐 수 있도록 함.

즉, 이전 모델보다 더 잘 고려될 수 있도록 데이터를 업데이트 함.

03. 에이다 부스트(AdaBoost)

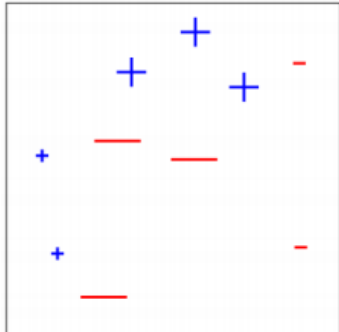
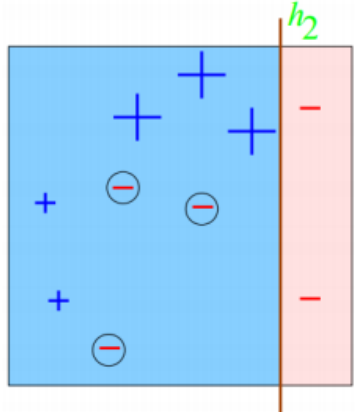
부스팅-에이다 부스트(AdaBoost) Steps



[Step 3] 두 번째 weak learner(h_2) 생성

03. 에이다 부스트(AdaBoost)

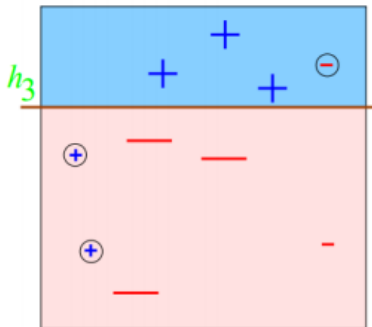
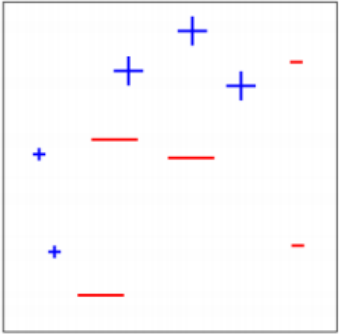
부스팅-에이다 부스트(AdaBoost) Steps



[Step 4] 앞의 과정과 마찬가지로 오분류된 개체에 가중치를 부여하여 더 잘 분류될 수 있도록 함

03. 에이다 부스트(AdaBoost)

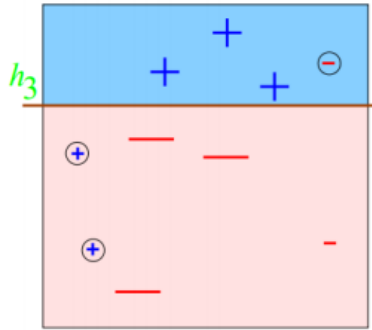
부스팅-에이다 부스트(AdaBoost) Steps



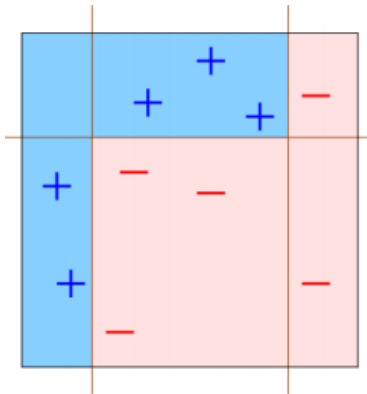
[Step 5] 세 번째 weak learner(h_3) 생성

03. 에이다 부스트(AdaBoost)

부스팅-에이다 부스트(AdaBoost) Steps



[Step 6] 세 개의 weak learner를 결합한 strong learner 생성



⇒ 3번의 과정만으로 데이터를 완벽히 분류

⇒ 개별의 약한 학습기보다 정확도가 월등히 높아짐

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

①
Input: Required ensemble size T

②
Input: Training set $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$

③
Define a uniform distribution $D_1(i)$ over elements of S .

for $t = 1$ to T do

④
Train a model h_t using distribution D_t .

⑤
Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$
If $\epsilon_t \geq 0.5$ break
Set $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

⑥
Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

end for

For a new testing point (x', y') ,

⑦
 $H(x') = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x') \right)$

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

①

Required ensemble size T

몇 개의 개별적인 learner를 쓸 것인지
(보통 50개~100개 사용)

②

Training set $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$

이해를 쉽게 하기 위해 y_i 를 -1 과 $+1$ 로 이범주 분류함
뒤에서 설명할 메커니즘에 잘 적용되기 위해 이렇게 정의
⇒ 크게 의미는 없음

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

③

Define a uniform distribution $D_1(i)$ over elements of S .

- D_t 는 t 번째 데이터 set
- $D_t(i)$ 는 t 번째 데이터 set에 i 번째 객체가 선택될 확률
- 즉, $D_1(i)$ 는 첫번째 데이터 set에 i 번째 관측치가 선택될 확률
- 첫번째 모델(weak learner)에 쓰일 데이터이므로 모든 관측치가 선택될 확률은 동일(uniform distribution)

④

Train a model h_t using distribution D_t .

- h_t 는 D_t 로 적합시키는 weak model
- t 번째 데이터로 t 번째 weak learner 생성
(앞서 말했듯이 weak learner는 random guessing보다 조금 더 나은 수준의 stump tree)

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

⑤

```
Calculate  $\epsilon_t = P_{D_t}(h_t(x) \neq y)$   
If  $\epsilon_t \geq 0.5$  break  
Set  $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
```

- 해당 모형의 오분류율 계산
- ϵ_t (t번째 객체의 오분류율) = t번째 객체의 예측값($h_t(x)$)과 실제값(y) 차이의 비율
- cf.) 오분류율은 0.5보다 작아야한다! 오분류율이 0.5보다 크다는 것은 random guessing보다도 못하다는 의미, 만일 오분류율이 0.5 이상이면 해당 모델은 사용하지 않음(break)

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

⑤

```
Calculate  $\epsilon_t = P_{D_t}(h_t(x) \neq y)$   
If  $\epsilon_t \geq 0.5$  break  
Set  $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
```

- α_t 는 가중치를 의미함
- ϵ_t 가 0.5로 근사하는 경우 $\rightarrow \alpha_t = \frac{1}{2} \ln \frac{0.5}{0.5} = 0 \rightarrow$ 가중치를 굉장히 작게 줌
- ϵ_t 가 0으로 근사하는 경우 $\rightarrow \alpha_t = \frac{1}{2} \ln \frac{1}{0} = \infty \rightarrow$ 가중치 굉장히 커짐
- 즉, 오분류율이 좋을수록(낮을수록), 학습 데이터의 정확도가 높을수록, 최종 모델을 결합하는 시점에서 가중치가 높아진다.

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

⑥

Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

- boosting 모델의 기본은 weak learner를 순차적으로 만들어가며 오류를 개선하는 것
- $y_i = h_t(x_i)$ 일 때 \rightarrow 둘의 곱은 항상 1 (y_i 는 실제값, $h_t(x_i)$ 는 예측값)
- $y_i \neq h_t(x_i)$ 일 때 \rightarrow 둘의 곱은 항상 -1
- h_t 가 정답을 잘 예측했다면 t+1번째 데이터 set에서 i번째 객체가 선택될 확률 낮아짐
- h_t 가 정답을 예측하지 못했다면 t+1번째 데이터 set에서 i번째 객체가 선택될 확률 높아짐
- 확률 증가감소의 폭은 가중치(α_t)의 영향을 받음

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

⑦

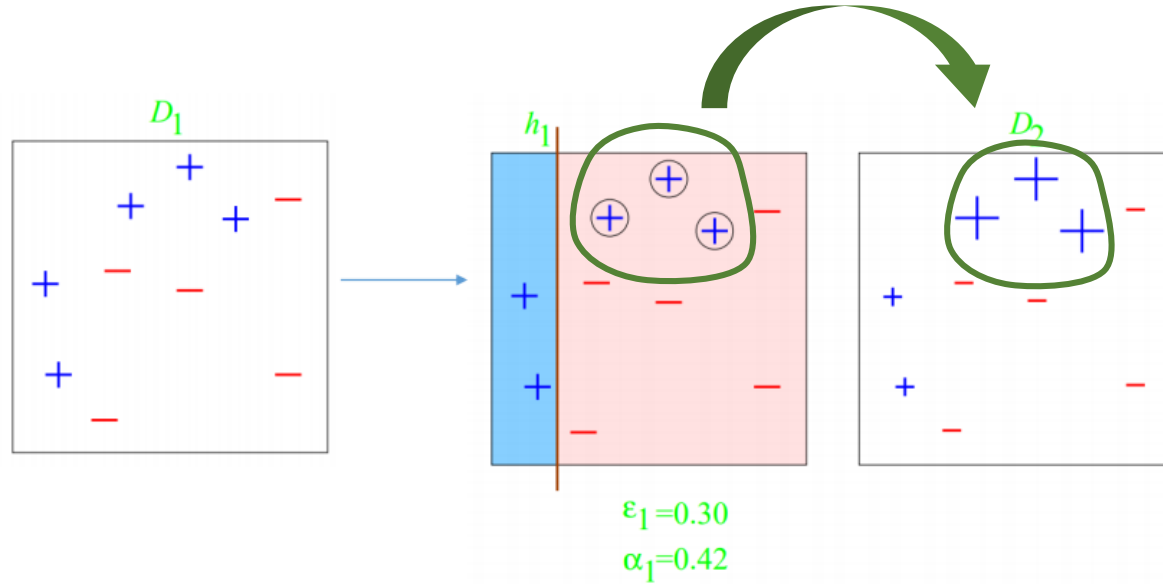
$$H(x') = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x')\right)$$

- 개별 weak learner에 가중치 적용하여 최종 모델 생성

앞서 본 예시를 통해 에이다 부스트 알고리즘에 대해 다시 한번 살펴보자!

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘



- 임의의 weak learner h_1 생성

⇒ 동그라미 쳐진 + 들이 잘못 분류됨

ϵ_1 (첫 번째 객체의 오분류율) = 0.3, α_1 (첫 번째 객체의 가중치) = 0.42

$$\text{Set } \alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

앞서 본 이 공식에
오분류율 값 0.3 대입
⇒ 0.42

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘

✓ The selection probability of x_i for the next training dataset

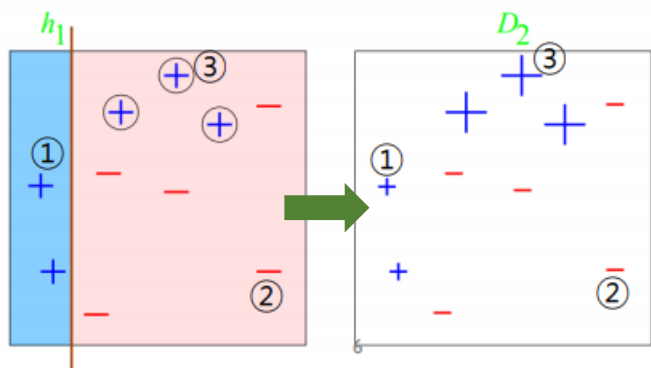
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

✓ Case 1: $y_i = 1, h_t(x_i) = 1 \rightarrow y_i h_t(x_i) = 1 \rightarrow -\alpha_t y_i h_t(x_i) < 0 \rightarrow \text{decrease } p$

✓ Case 2: $y_i = -1, h_t(x_i) = -1 \rightarrow y_i h_t(x_i) = 1 \rightarrow -\alpha_t y_i h_t(x_i) < 0 \rightarrow \text{decrease } p$

✓ Case 3: $y_i = 1, h_t(x_i) = -1 \rightarrow y_i h_t(x_i) = -1 \rightarrow -\alpha_t y_i h_t(x_i) > 0 \rightarrow \text{increase } p$

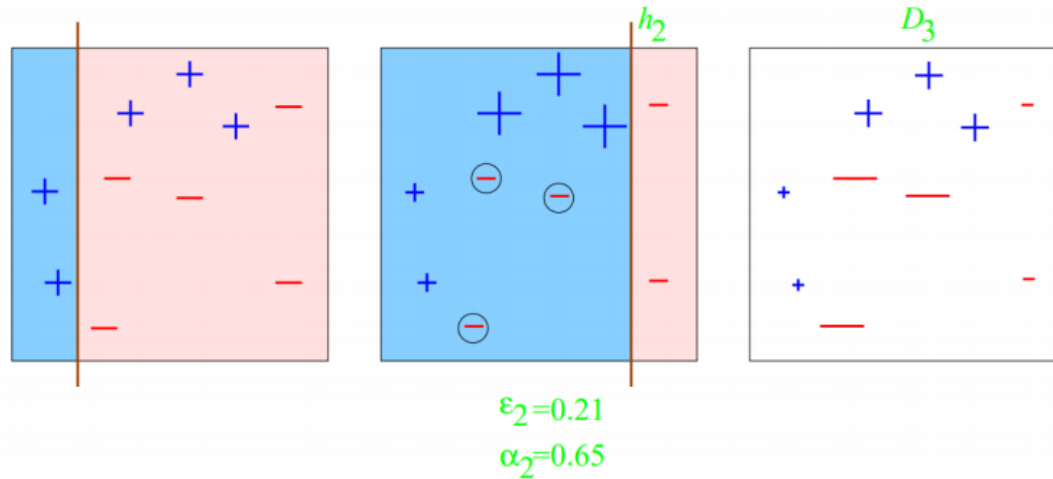
✓ α_t is the confidence of the current model that controls the magnitude of change



- Case1과 case2는 잘 분류된 경우로 현재보다 다음 train set에 선택될 확률이 감소한다.
- Case3는 잘못 분류된 경우로 선택될 확률 증가
- 조정된 확률에 따라 두번째 train set D2 생성

03. 에이다 부스트(AdaBoost)

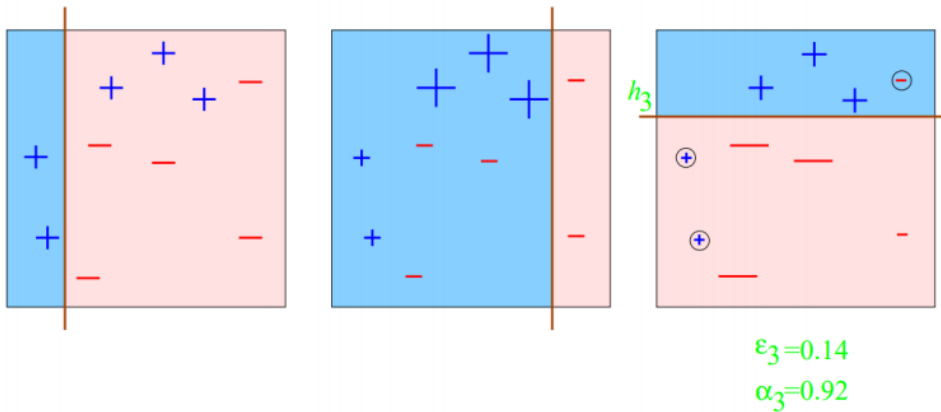
에이다 부스트(AdaBoost) 알고리즘



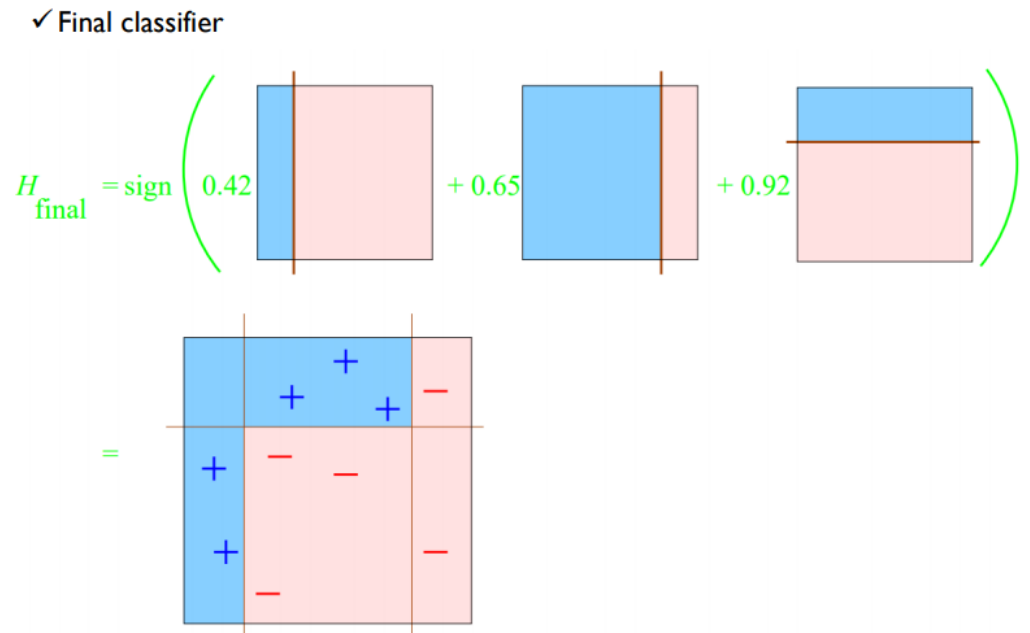
- D2에 weak learner h_2 생성
- 앞과 같은 과정을 거쳐 정분류된 관측치의 비중은 작아지고 오분류된 관측치의 비중은 커짐

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 알고리즘



- D3에 weak learner h_3 생성



- 3개의 weak learner에 가중치를 적용하여 계산한 최종 모델 생성

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 실습

```
# 라이브러리 import 및 데이터준비
import pandas as pd
import numpy as np
import warnings
import time
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer

warnings.filterwarnings('ignore')

dataset = load_breast_cancer()
X_features = dataset.data
y_label = dataset.target

cancer_df = pd.DataFrame(data=X_features, columns=dataset.feature_names)
cancer_df['target'] = y_label
cancer_df.head(3)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness		worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.8	1001.0	0.11840	0.27760		0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	■ ■ ■	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.0	1203.0	0.10960	0.15990		0.4504	0.2430	0.3613	0.08758	0

- 종양의 크기와 모양에 관련된 속성들이 숫자형으로 존재
- 악성인 malignant는 0, 양성은 benign는 1값

라이브러리 import 및 위스콘신 유방암 데이터 확인 후
train, test 데이터로 분리

```
print(dataset.target_names)
print(cancer_df['target'].value_counts())
```

```
['malignant' 'benign']
1    357
0    212
Name: target, dtype: int64
```

양성은 357개, 악성은 212개로 구성

```
# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test = train_test_split(X_features, y_label,
                                                    test_size=0.2, random_state=156)
```

```
print(X_train.shape, X_test.shape)
```

(455, 30) (114, 30)

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 실습

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
ada_clf = AdaBoostClassifier(
    base_estimator = DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

```
AdaBoostClassifier(algorithm='SAMME',
                    base_estimator=DecisionTreeClassifier(max_depth=1),
                    learning_rate=0.5, n_estimators=200, random_state=42)
```

**200개의 의사 결정 트리 모델을 개별 모델로 사용하는
에이다부스트 분석기 정의, 학습**

```
# Ada 부스트 수행 시간 측정을 위한, 시작 시간 설정.
start_time = time.time()

ada_pred = ada_clf.predict(X_test)
ada_accuracy = accuracy_score(y_test, ada_pred)

print('ADA 정확도: {0:.4f}'.format(ada_accuracy))
print("ADA 수행 시간: {0:.1f} 초 ".format(time.time() - start_time))
```

ADA 정확도: 0.9649
ADA 수행 시간: 0.1 초

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100, 300, 500],
    'learning_rate': [0.05, 0.1, 0.3]
}
grid_cv = GridSearchCV(ada_clf, param_grid=params, cv=3, verbose=1)
grid_cv.fit(X_train, y_train)
print('최적 하이퍼 파라미터: \n', grid_cv.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

Fitting 3 folds for each of 9 candidates, totalling 27 fits

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 27 out of 27 | elapsed: 26.7s finished
```

최적 하이퍼 파라미터:
{'learning_rate': 0.1, 'n_estimators': 500}
최고 예측 정확도: 0.9693

* Adaboost의 learning rate

- Adaboost에서 learning rate는 가중치에 존재

$$\alpha_m = L \frac{1}{2} \ln \frac{1-e_m}{e_m}$$

- L이 learning rate 의미

```
import warnings
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

* SAMME 알고리즘

```
ada_clf = AdaBoostClassifier(
    base_estimator = DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm='SAMME.R', random_state=42)
ada_clf.fit(X_train, y_train)
```

- 앞서 살펴본 알고리즘과 거의 유사하지만 가중치 계산에 차이가 있음

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                    algorithm='SAMME', n_estimators=200, random_state=42)
```

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1).$$

- K는 target 변수의 범주 개수
- 이진분류 즉, K = 2인 경우 위 알고리즘과 완전히 같다

에이다부스트 분석기 정의, 학습

```
# Ada 부스트 수행 시간 측정을 위한, 시작 시간 설정.
start_time = time.time()

ada_pred = ada_clf.predict(X_test)
ada_accuracy = accuracy_score(y_test, ada_pred)

print('ADA 정확도: {0:.4f}'.format(ada_accuracy))
print("ADA 수행 시간: {0:.1f} 초 ".format(time.time() - start_time))
```

ADA 정확도: 0.9649
ADA 수행 시간: 0.1 초

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100, 300, 500],
    'learning_rate': [0.05, 0.1, 0.3]
}

grid_cv = GridSearchCV(ada_clf, param_grid=params, cv=3, verbose=1)
grid_cv.fit(X_train, y_train)
print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

Fitting 3 folds for each of 9 candidates, totalling 27 fits

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 27 out of 27 | elapsed: 26.7s finished
```

최적 하이퍼 파라미터:
{'learning_rate': 0.1, 'n_estimators': 500}
최고 예측 정확도: 0.9693

03. 에이다 부스트(AdaBoost)

에이다 부스트(AdaBoost) 실습

```
# GridSearchCV를 이용하여 최적으로 학습된 estimator로 predict 수행.  
ada_pred = grid_cv.best_estimator_.predict(X_test)  
ada_accuracy = accuracy_score(y_test, ada_pred)  
print('ADA 정확도: {:.4f}'.format(ada_accuracy))
```

ADA 정확도: 0.9737

**최적 하이퍼 파라미터를 찾고
분석기의 정확도 확인!**

04. GBM

04. GBM

부스팅-GBM

Gradient boosting = gradient descent + boosting

Original Dataset		Modified Dataset 1		Modified Dataset 2	
x^1	y^1	x^1	$y^1 - f_1(x^1)$	x^1	$y^1 - f_1(x^1) - f_2(x^1)$
x^2	y^2	x^2	$y^2 - f_1(x^2)$	x^2	$y^2 - f_1(x^2) - f_2(x^2)$
x^3	y^3	x^3	$y^3 - f_1(x^3)$	x^3	$y^3 - f_1(x^3) - f_2(x^3)$
x^4	y^4	x^4	$y^4 - f_1(x^4)$	x^4	$y^4 - f_1(x^4) - f_2(x^4)$
x^5	y^5	x^5	$y^5 - f_1(x^5)$	x^5	$y^5 - f_1(x^5) - f_2(x^5)$
x^6	y^6	x^6	$y^6 - f_1(x^6)$	x^6	$y^6 - f_1(x^6) - f_2(x^6)$
x^7	y^7	x^7	$y^7 - f_1(x^7)$	x^7	$y^7 - f_1(x^7) - f_2(x^7)$
x^8	y^8	x^8	$y^8 - f_1(x^8)$	x^8	$y^8 - f_1(x^8) - f_2(x^8)$
x^9	y^9	x^9	$y^9 - f_1(x^9)$	x^9	$y^9 - f_1(x^9) - f_2(x^9)$
x^{10}	y^{10}	x^{10}	$y^{10} - f_1(x^{10})$	x^{10}	$y^{10} - f_1(x^{10}) - f_2(x^{10})$

...

- Adaboost와 유사하나, 가중치 업데이트를 경사 하강법(Gradient Descent)을 이용하는 것이 큰 차이
- Adaboost와 달리, weak learner 강화에 **전체 데이터를 사용**하고 오차를 보정해나가는 방식으로 진행되는 부스팅
- $Y - f_1(x)$, 즉 잔차를 새로운 목표값으로 설정
- 새로운 목표값(잔차)에 대한 모델 $f_2(x)$
- 앞에서 못맞췄던 부분만 집중해서 예측하는 것이 gradient boosting의 기본 아이디어


$$y = f_1(x) \quad y - f_1(x) = f_2(x) \quad y - f_1(x) - f_2(x) = f_3(x)$$

04. GBM

부스팅-GBM

Gradient boosting = gradient descent + boosting

Original Dataset

x^1	y^1
x^2	y^2
x^3	y^3
x^4	y^4
x^5	y^5
x^6	y^6
x^7	y^7
x^8	y^8
x^9	y^9
x^{10}	y^{10}

Modified Dataset 1

x^1	$y^1 - f_1(x^1)$
x^2	$y^2 - f_1(x^2)$
x^3	$y^3 - f_1(x^3)$
x^4	$y^4 - f_1(x^4)$
x^5	$y^5 - f_1(x^5)$
x^6	$y^6 - f_1(x^6)$
x^7	$y^7 - f_1(x^7)$
x^8	$y^8 - f_1(x^8)$
x^9	$y^9 - f_1(x^9)$
x^{10}	$y^{10} - f_1(x^{10})$

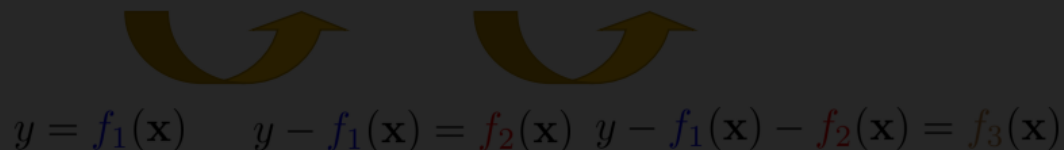
Modified Dataset 2

x^1	$y^1 - f_1(x^1) - f_2(x^1)$
x^2	$y^2 - f_1(x^2) - f_2(x^2)$
x^3	$y^3 - f_1(x^3) - f_2(x^3)$
x^4	$y^4 - f_1(x^4) - f_2(x^4)$
x^5	$y^5 - f_1(x^5) - f_2(x^5)$
x^6	$y^6 - f_1(x^6) - f_2(x^6)$
x^7	$y^7 - f_1(x^7) - f_2(x^7)$
x^8	$y^8 - f_1(x^8) - f_2(x^8)$
x^9	$y^9 - f_1(x^9) - f_2(x^9)$
x^{10}	$y^{10} - f_1(x^{10}) - f_2(x^{10})$

...

- Adaboost과 유사하나, 가중치 업데이트를 경사 하강법(Gradient Descent)으로 사용하는 것이 큰 차이
- Adaboost과 달리, weak learner 강화에 전체 데이터를 사용하고 오차를 보정해나가는 방식으로 진행되는 부스팅
- $Y - f_1(x)$, 즉 잔차를 새로운 목표값으로 설정
- 새로운 목표값(잔차)에 대한 모델 $f_2(x)$
- 앞에서 못맞췄던 부분만 집중해서 예측하는 것이 gradient boosting의 기본 아이디어

분류 섹션을 다루고 있지만 설명의 편의와 보다 더 쉬운 이해를 위해 회귀문제에 대입하여 설명 진행!!

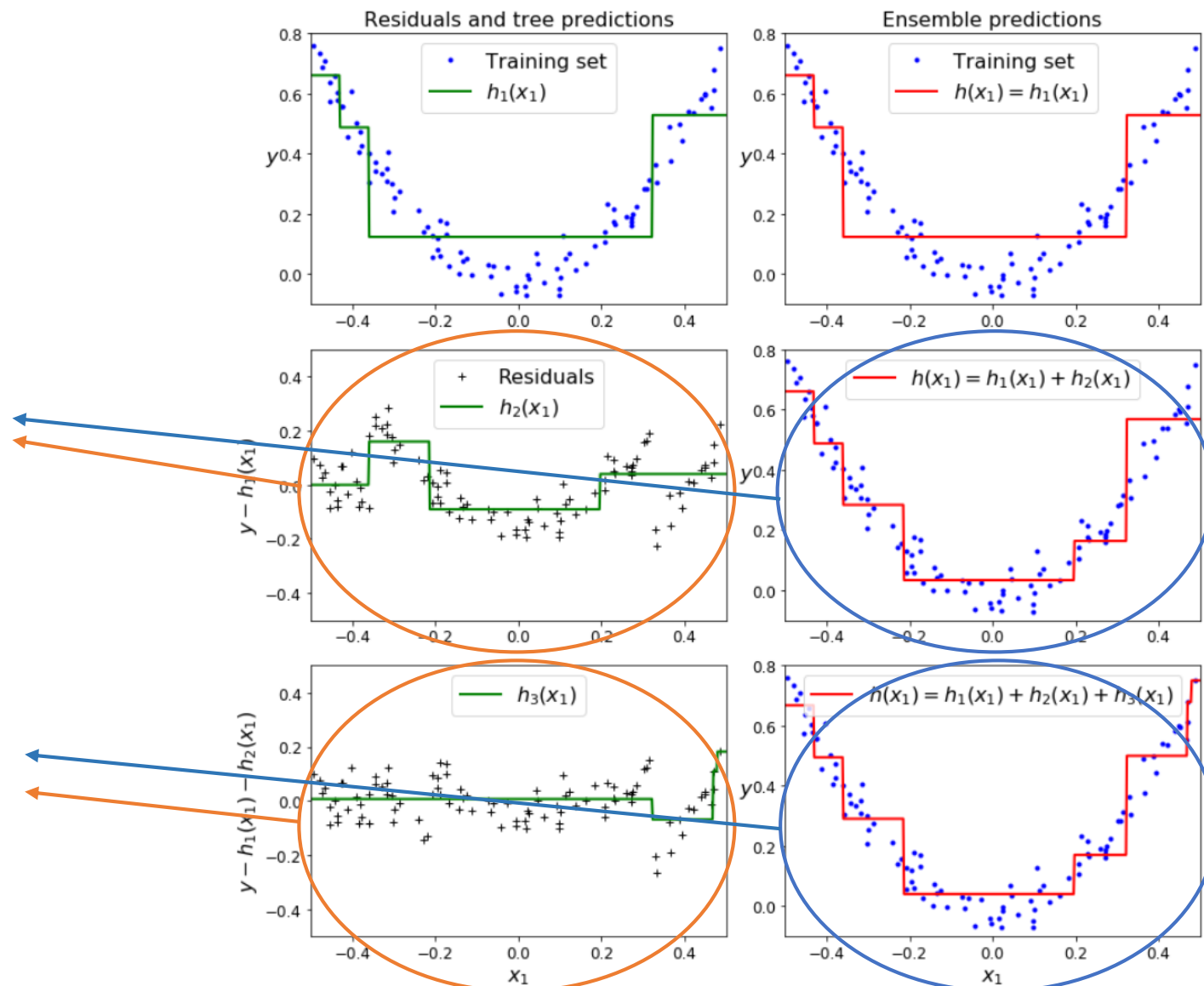

$$y = f_1(x) \quad y - f_1(x) = f_2(x) \quad y - f_1(x) - f_2(x) = f_3(x)$$

04. GBM

부스팅-GBM

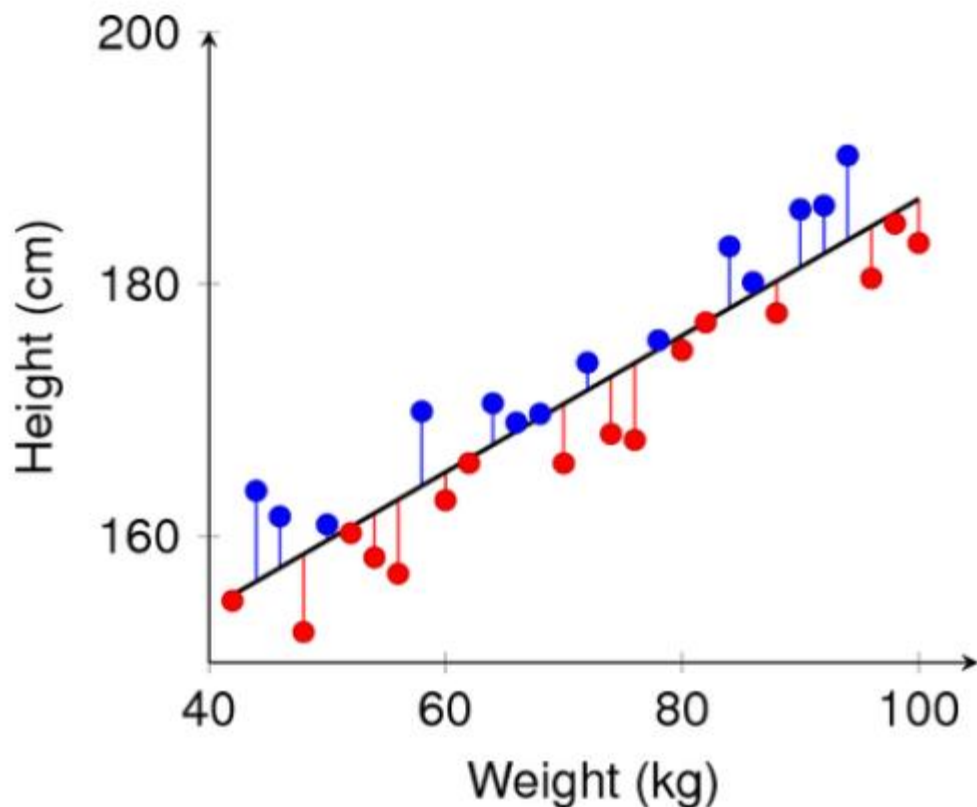
- 첫번째 모델이 예측하고 난 후의 잔차를 예측
- 첫번째 모델과 두번째 모델을 결합한 모델로 y 예측

- 앞선 모델이 예측한 잔차를 다시 예측
- 앞선 모델들과 결합하여 다시 y 예측



04. GBM

부스팅-GBM



- 회귀식이 $\hat{y} = f_1(x)$ 일 때 $y - f_1(x)$ 는 잔차(residual)임
- 이때 두 번째 모형($f_2(x)$)을 만들어서 $y - f_1(x) = f_2(x)$ 라고 한다면, 첫 번째 모형이 찾지 못했던 오차만큼을 추정할 수 있음
- Adaboosting에서는 이전 모델이 못 맞추고 있는 만큼을 가중치를 뒤서 데이터가 더 많이 샘플링 되게 하지만, GBM은 데이터를 샘플링 하는 것이 아니라 우리가 추정해야 하는 목표값(y)을 계속 바꾸는 것임

04. GBM

경사하강법(Gradient Descent)

비용함수(cost function)란?

=> 실제값(y)과 예측 함수($f(x)$)의 차이

$$\frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2$$

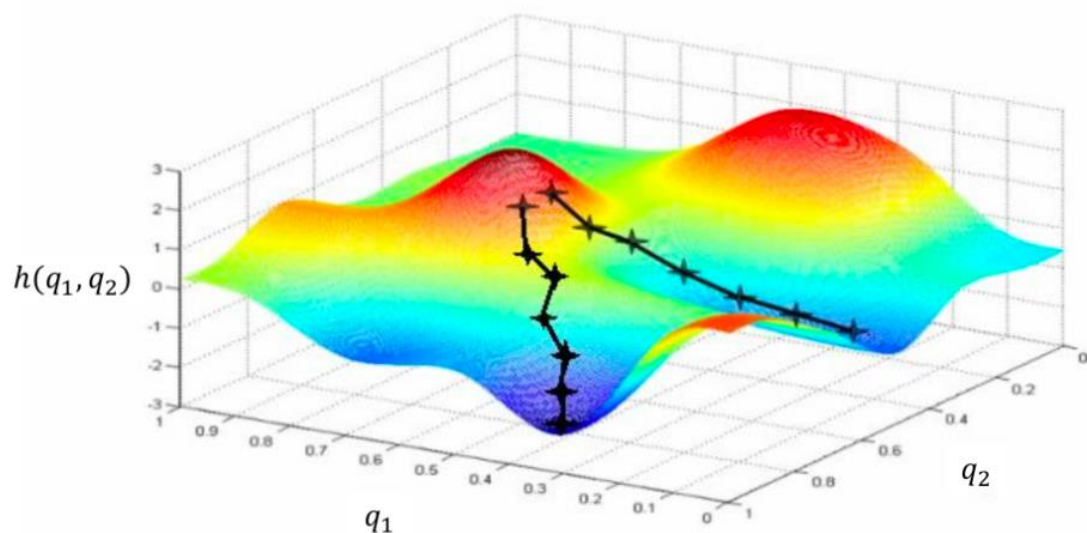
직관적으로 생각해 봤을 때 좋은 모델은 실제 값과 예측 값의 차이, 즉 오차가 작은 모형임.

Cost function은 모델의 오차를 도출해내는 함수이며

Cost function을 최소화함으로써 최적 파라미터를 찾을 수 있다

04. GBM

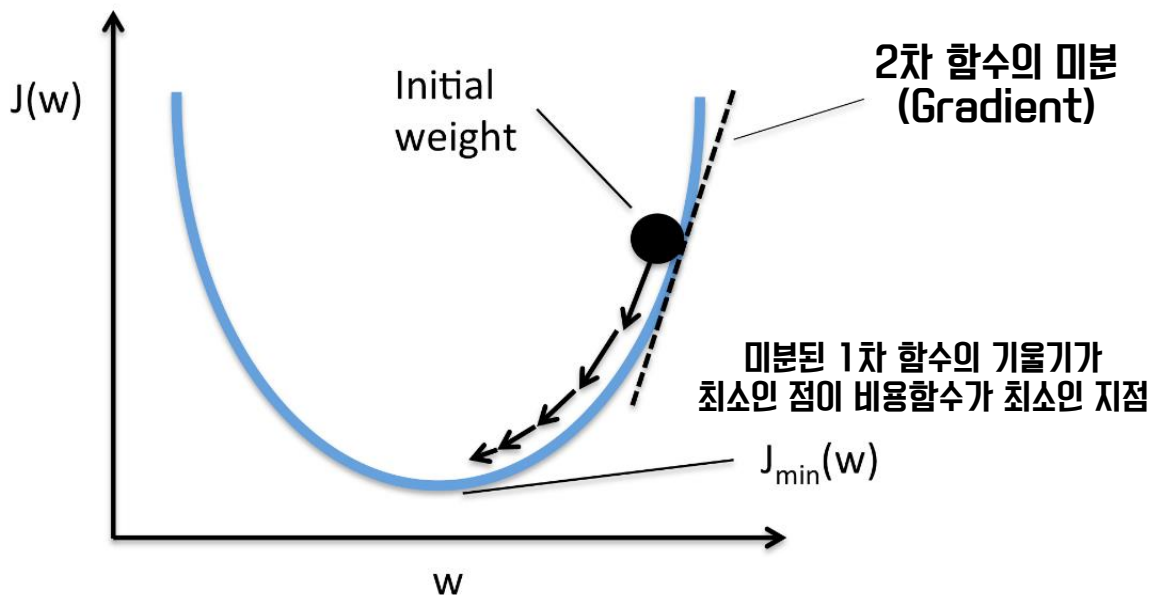
경사하강법(Gradient Descent)



- 경사하강법은 말 그대로 경사를 따라 가장 낮은 곳으로 이동하며 최소값을 찾는 방식
- 즉, 파라미터 값을 업데이트하며 cost function이 최소가 되는 값을 향해 점진적으로 나아가는 방식
- '데이터를 기반으로 알고리즘이 스스로 학습한다'는 머신러닝의 개념을 가능하게 만들어준 핵심기법
- 그렇다면 어떻게 오류가 작아지는 방향으로 파라미터 값을 보정할 수 있을까?

04. GBM

경사하강법(Gradient Descent)



- 왼쪽 이차함수를 $f(x) = x^2$ 라고 한다면, $\frac{dy}{dx} = 2x$ 임

- $$\underbrace{x_{new}}_{\text{업데이트된 } x \text{ 값}} = \underbrace{x_{old}}_{\text{기존의 } x \text{ 값}} - \underbrace{a}_{\substack{\text{Learning rate} \\ \rightarrow \text{파라미터가 움직이는} \\ \text{정도를 조절해줌}}} \times \underbrace{(2x_{old})}_{\substack{\text{기존 } x \text{의 gradient} \\ \text{(미분한 값)}}$$

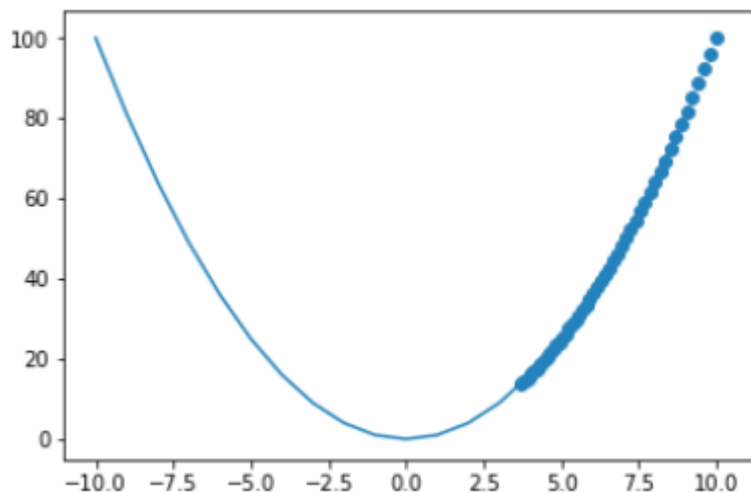
- x_{new} 에서의 미분 값(gradient) > 0 이면 왼쪽으로 이동
 - x_{new} 에서의 미분 값(gradient) < 0 이면 오른쪽으로 이동
- 즉, 파라미터는 gradient의 반대 방향으로 움직인다!**

04. GBM

경사하강법(Gradient Descent)

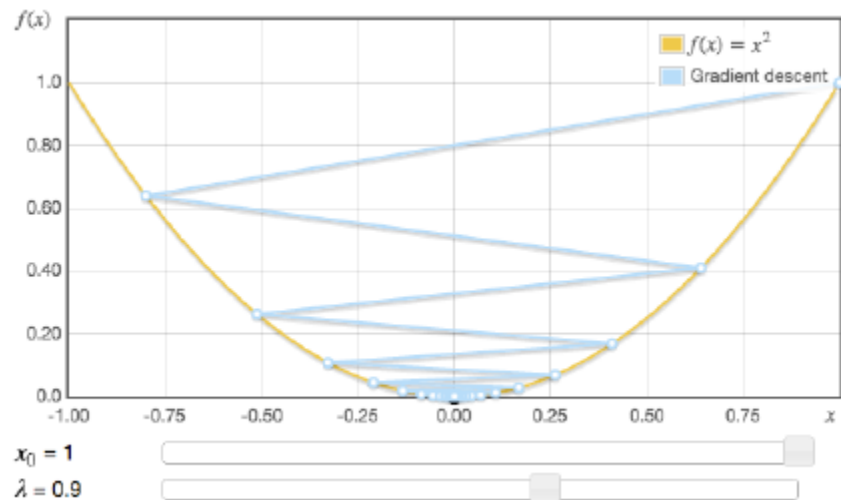
그렇다면 α (Learning rate)를 어떻게 설정해야 할까?

〈Learning rate가 너무 작은 경우〉



⇒ 파라미터가 천천히 이동하기 때문에
계산이 굉장히 오래 걸리고 끝까지 못감

〈Learning rate가 너무 큰 경우〉



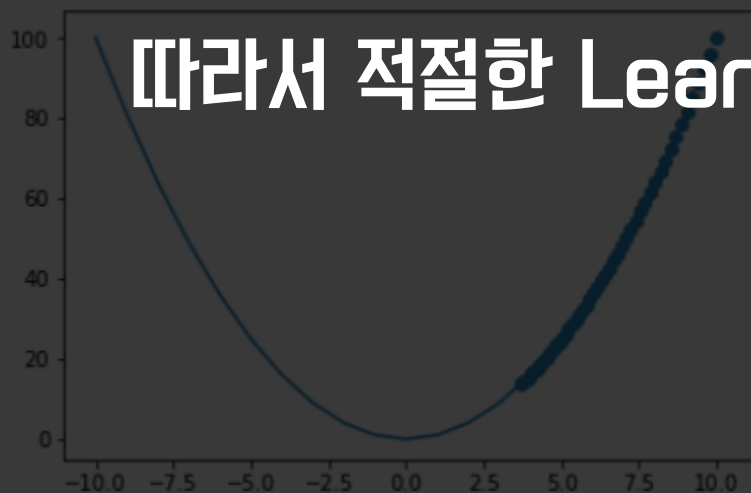
⇒ 파라미터가 최솟값으로 수렴하지 못하고
데이터가 튀는 문제가 생김

04. GBM

경사하강법(Gradient Descent)

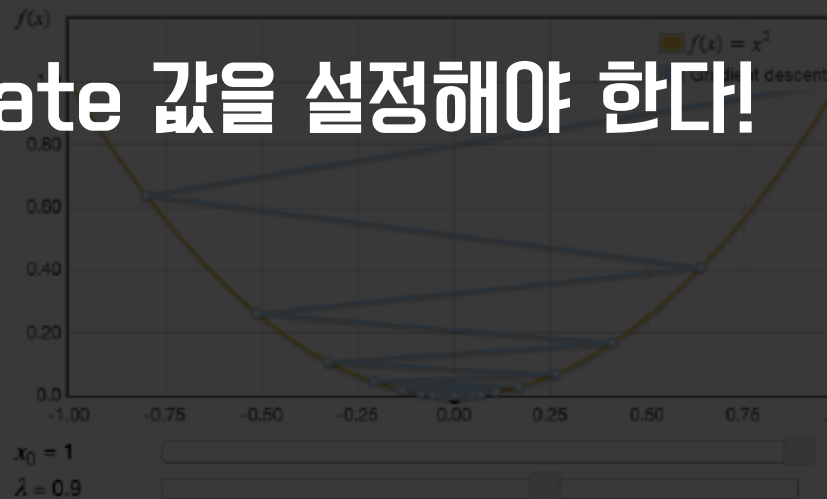
그렇다면 α (Learning rate)를 어떻게 설정해야 할까?

<Learning rate가 너무 작은 경우>



⇒ 파라미터가 천천히 이동하기 때문에
계산이 굉장히 오래 걸리고 끝까지 못감

<Learning rate가 너무 큰 경우>



⇒ 파라미터가 최솟값으로 수렴하지 못하고
데이터가 튀는 문제가 생김

04. GBM

GBM 알고리즘-예시 적용

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, and a differentiable **Loss Function** $L(y_i, F(x))$

- loss function = $\frac{1}{2}(y_i - f(x_i))^2$

⇒ 가정 회귀에서 가장 일반적으로 쓰이는 loss function인 Squared loss 공식

(loss function의 종류는 Squared loss 외에도 다양함)

04. GBM

GBM 알고리즘-예시 적용

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

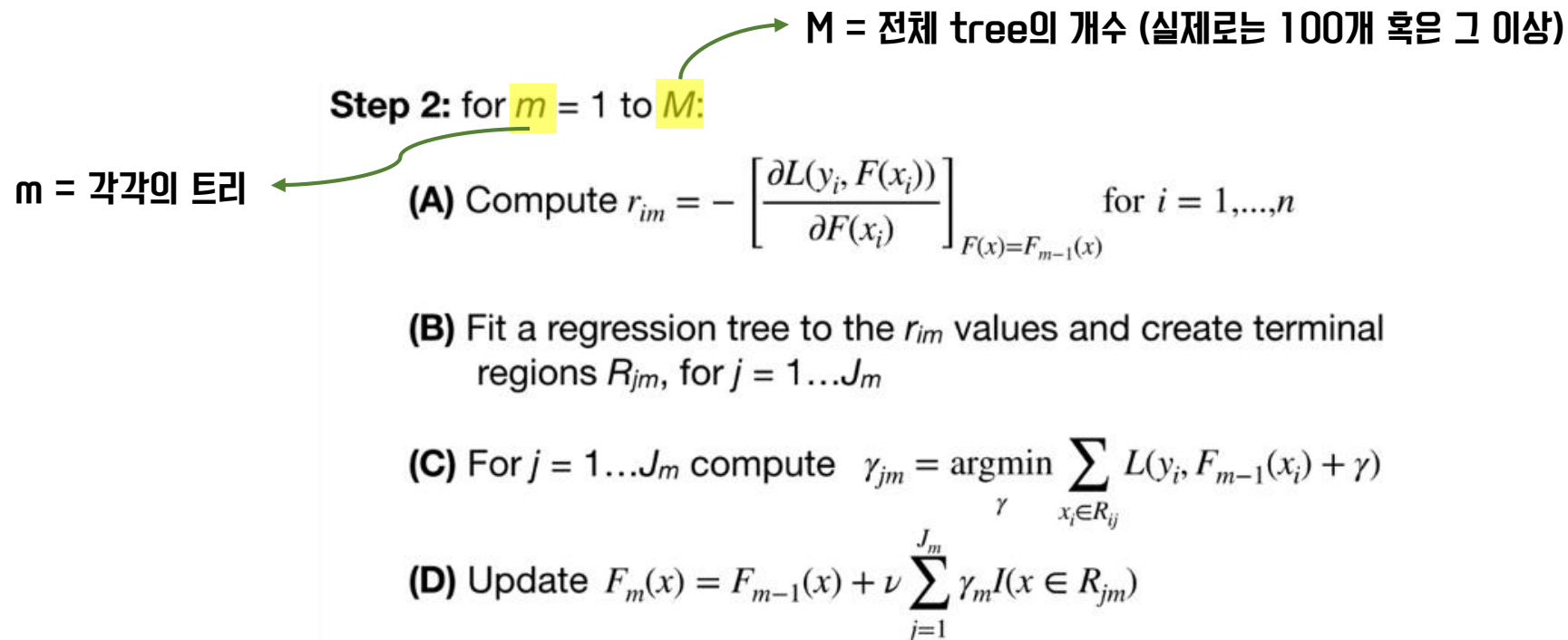
Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$

Loss function

- Initial model은 $\underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n \frac{1}{2} (y_i - r)^2$ 를 만족하는 γ 값, 즉, Loss function의 합을 최소화하는 $\gamma = F_0(x)$ 를 찾아야 함
- 따라서, 앞선 모델의 미분한 값 $\sum_{i=1}^n -(y_i - r) = 0$ 일 때의 γ 값을 찾으면 됨 \Rightarrow 이것이 바로 경사하강법
- 예제 데이터 대입
$$-(88 - \gamma) - (76 - \gamma) - (56 - \gamma) = 0,$$
$$\gamma = (88+76+56)/3 = y_i \text{ 값들의 평균} = 73.3 = F_0(x)$$
따라서, 73.3이 현재 데이터들에 대한 초기 예측값

04. GBM

GBM 알고리즘



일단 $m=1$ 이라고 생각하고 Step 2를 뜯어보자.

04. GBM

GBM 알고리즘

(A) Compute $r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \dots, n$

Gradient

(A)

- $F(x)$ 에 대해 Loss function을 미분하면

$$- \left(- (y - F(x)) \right) = y - F(x)$$

- 즉, loss function의 negative gradient는

잔차(residual)가 된다.

- 예제 데이터 대입

$$r_{i,1} = (\text{관측값} - 73.3)$$

(γ_{im} 에서 γ 은 residual, i 는 샘플 숫자, m 은 만들 트리)

Height (m)	Favorite Color	Gender	Weight (kg)	$r_{i,1}$	
1.6	Blue	Male	88	14.7	$r_{1,1}$
1.6	Green	Female	76	2.7	$r_{2,1}$
1.5	Blue	Female	56	-17.3	$r_{3,1}$

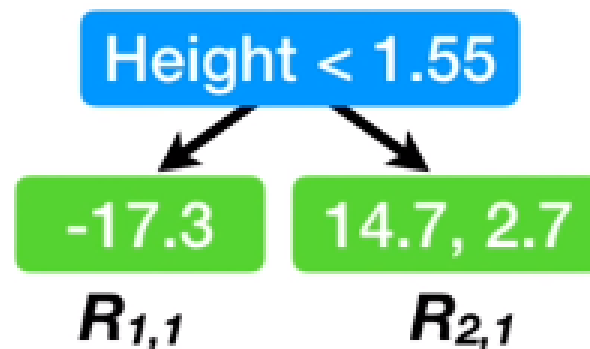
04. GBM

GBM 알고리즘

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm} , for $j = 1 \dots J_m$

(B)

- Tree 모형 생성



(원래 GBM에서는 stump tree가 아니라
제대로 된 tree 모형 사용하지만,
이해의 편의를 위해 가장 단순화한 모형 사용)

- R_{jm} 에서 m 은 몇 번째 트리인가를, j 는 각 리프의 인덱스를 의미
⇒ $R_{1,1}$ 은 첫 번째 트리에서 첫 번째 리프를,
 $R_{2,1}$ 은 첫 번째 트리에서 두 번째 리프를 의미
⇒ 해당 예시 트리에서는 리프가 2개 있으므로 $J_m = 2$

04. GBM

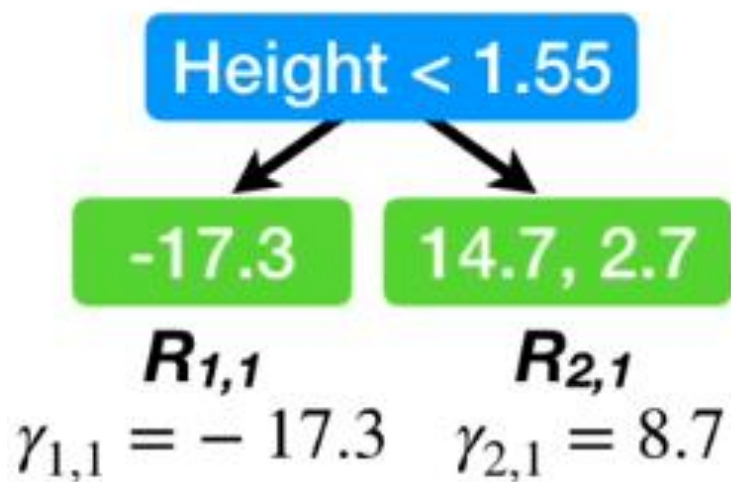
GBM 알고리즘

(C)

(C) For $j = 1 \dots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

앞서 언급한 $\frac{1}{2}(y_i - f(x_i))^2$ 에 대입하면

$$\frac{1}{2}(y_i - (F_{m-1}(x_i) + \gamma))^2$$



- 앞서 도출했듯이 $m = 1, F_0(x) = 73.3$
- 먼저, $R_{1,1}$ 은 오직 세 번째 행의 데이터(X_3)만 해당되므로 Σ 를 없애고 계산할 수 있음, $i = 3$
 $\Rightarrow \gamma_{1,1} = \underset{\gamma}{\operatorname{argmin}} \frac{1}{2}(y_1 - (F_0(x_3) + \gamma))^2$
 $\Rightarrow \gamma_{1,1} = -(56 - 73.3 - \gamma) = 0$
- 따라서 $\gamma_{1,1} = \gamma = -17.3$ 이므로 $R_{1,1}$ 의 output은 -17.3 임
- 같은 방식으로
 $\gamma_{2,1} = \underset{\gamma}{\operatorname{argmin}} \left[\frac{1}{2}(88 - (73.3 + \gamma))^2 + \frac{1}{2}(76 - (73.3 + \gamma))^2 \right]$
따라서 $\gamma_{2,1} = \gamma = 8.7$ 이므로 $R_{2,1}$ 의 output은 8.7 임
 \Rightarrow 결국 14.7 과 2.7 의 평균!

04. GBM

GBM 알고리즘

(D)

$$(D) \text{ Update } F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_m I(x \in R_{jm})$$

Learning rate를 0.1로 설정,
이 Learning rate는 정확도를 높여줌

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

● (D)는 각 샘플에 대해 새로운 예측을 적용하는 것

● $F_1(x) = \underbrace{73.3}_{F_0(x)} + 0.1 \times \underbrace{8.7}_{\gamma_{2,1}} = 74.2$

⇒ 처음에 예측했던 73.3에 비해 실제 값인 88에 더 가까워진 것을 알 수 있음

● 같은 방식으로 x_2, x_3 도 74.2, 71.6으로 업데이트 하면
처음 예측했던 73.3보다 실제 값에 더 가까워진 것을 확인할 수 있다!

04. GBM

GBM 알고리즘

Step 3: Output $\hat{f}(x) = f_M(x)$

앞선 과정을 M번 반복한 것이 최종 모형이 된다.

04. GBM

GBM 실습

하이퍼 파라미터 소개

- Weak learner로 tree 모형 사용하기 때문에 `n_estimator`, `max_depth`, `max_features` 같은 트리 기반 자체 파라미터도 존재함
- `loss` : 경사 하강법에서 사용할 비용함수 지정, default값은 'deviance' (회귀의 경우 ls)
- `learning rate` : learning rate 지정하는 파라미터, 0~1 사이의 값을 가지면 디폴트는 0.1
⇒ 앞서 설명했듯이 너무 작은 값은 정확한 예측을 할 확률이 높지만 그만큼 많은 시간이 필요함
반면 너무 큰 값은 속도는 빠르지만 최소 오류값에 도달하지 못할 수도 있음
따라서 `n_estimators` 파라미터와 상호보완적으로 조합하여 사용
- `n_estimators` : weak learner의 개수. weak learner가 순차적으로 오류를 보정하기 때문에 일정수준까지는 개수가 많을수록 예측성능이 향상됨. 하지만 개수가 많을수록 시간이 오래 걸리며 과적합 문제가 발생할 수 있음
- `Subsample` : weak learner가 학습에 사용하는 데이터 샘플링 비율

04. GBM

GBM 실습(회귀)

```
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3 * X[:, 0] ** 2 + 0.05 * np.random.randn(100)
```

X

```
array([[ -0.12545988],
       [  0.45071431],
       [  0.23199394],
       [  0.09865848],
       [ -0.34398136],
       [ -0.34400548],
       [ -0.44191639],
       [  0.36617615],
       [  0.10111501],
       [  0.20807258],
```

y

```
array([ 5.15728987e-02,  5.94479790e-01,  1.66051606e-01, -7.01779562e-02,
        3.43985933e-01,  3.72874939e-01,  6.59764984e-01,  3.76341398e-01,
       -9.75194335e-03,  1.04794741e-01,  7.35287787e-01,  6.78883363e-01,
        3.05066318e-01,  2.73909733e-01,  3.08559932e-01,  3.49130363e-01,
        7.98606436e-02, -1.45444646e-02, -5.71096619e-03,  5.75800683e-02,
        5.23392240e-02,  4.02946793e-01,  1.29867214e-01,  4.18481141e-02,
       -6.49789982e-02,  2.22943721e-01,  2.53451786e-01, -3.95060058e-02,
        1.75570720e-02,  6.37324227e-01,  1.29006981e-01,  3.34391950e-01,
        5.80417870e-01,  6.00772381e-01,  5.54501010e-01,  2.84001079e-01,
        1.17538848e-01,  6.08765289e-01,  9.22073759e-02,  2.58225391e-02,
        4.26829699e-01, -5.83641153e-02,  7.07523289e-01,  5.40226226e-01,
        2.14112889e-01,  3.37711060e-02,  1.76497872e-01, -6.88843767e-02,
        3.58884053e-02,  4.07472924e-01,  6.12002352e-01,  1.98779325e-01,
        5.84460527e-01,  4.42492127e-01, -4.87799540e-02,  5.37361759e-01,
        4.54900077e-01,  3.00958881e-01,  5.74483445e-01,  1.69025200e-01,
       -1.98442712e-03,  1.40740720e-01,  3.64880911e-01,  1.56124184e-05,
        1.55312061e-01,  7.08260013e-02,  9.06192075e-01,  2.89200798e-01])
```

실습위한 임의의 데이터 생성

04. GBM

GBM 실습(회귀)

```
# gbm 회귀 적용
from sklearn.ensemble import GradientBoostingRegressor
```

```
gb_regression_clf = GradientBoostingRegressor(max_depth=2, n_estimators=100,
                                              learning_rate=1.0, random_state=42)
gb_regression_clf.fit(X, y)
```

GradientBoostingRegressor(learning_rate=1.0, max_depth=2, random_state=42)

```
gb_regression_clf_slow = GradientBoostingRegressor(max_depth=2, n_estimators=200,
                                                    learning_rate=0.1, random_state=42)
gb_regression_clf_slow.fit(X, y)
```

GradientBoostingRegressor(max_depth=2, n_estimators=200, random_state=42)

- Learning rate와 weak learner 수에 따른 차이 비교
- Learning rate는 각각 1.0과 0.1 설정
- Weak learner의 수(n_estimators)는 100과 200으로 설정

04. GBM

GBM 실습(회귀)

```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

- 앞서 본 예시와 같은 그래프 시각화 위해 사용자 함수 생성
- 해당 코드를 일일이 뜯어볼 필요는 없음

04. GBM

GBM 실습(회귀)

```
fix, axes = plt.subplots(ncols=2, figsize=(10,4), sharey=True)

plt.sca(axes[0])
plot_predictions([gb_regression_clf], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="Ensemble predictions")
plt.title("learning_rate={}, n_estimators={}".format(gb_regression_clf.learning_rate,
                                                    gb_regression_clf.n_estimators), fontsize=14)

plt.xlabel("$x_1$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)

plt.sca(axes[1])
plot_predictions([gb_regression_clf_slow], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("learning_rate={}, n_estimators={}".format(gb_regression_clf_slow.learning_rate,
                                                    gb_regression_clf_slow.n_estimators), fontsize=14)

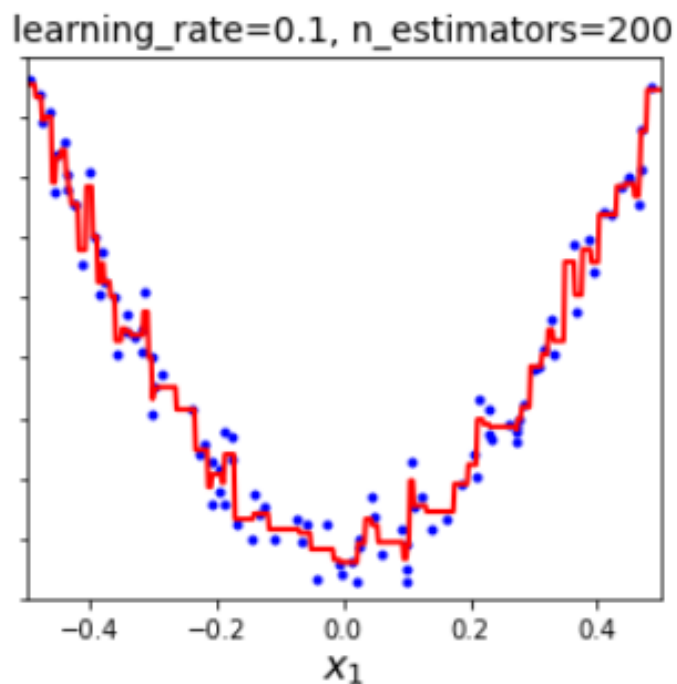
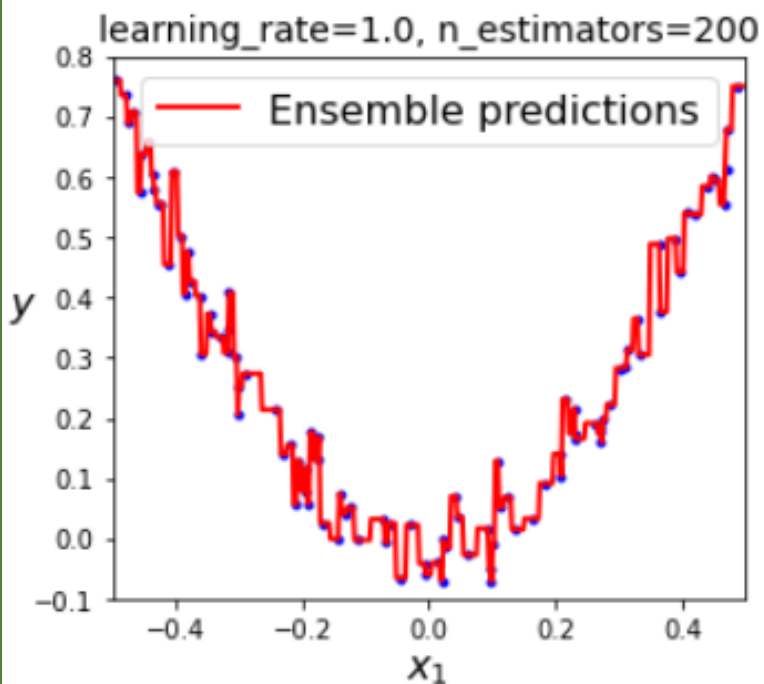
plt.xlabel("$x_1$", fontsize=16)

plt.show()
```

그래프 생성

04. GBM

GBM 실습(회귀)



- Learning rate가 1인 경우 반복 횟수가 적음에도 불구하고 모델을 빠르게 적합하여 과적합 문제가 심각하게 나타남을 확인할 수 있다
 - 반면, learning rate가 0.1인 경우 에러의 영향을 크게 받지 않고 비교적 잘 적합되었다. 하지만 이 역시도 weak learner의 수가 과도하게 많아지면 과적합 문제 발생할 수 있다.
- Gbm을 사용할 때는 과적합 문제에 주의하여 learning rate와 weak learner의 수를 적절히 조절해야 한다.

04. GBM

GBM 실습(분류)

```
from sklearn.ensemble import GradientBoostingClassifier

# GBM 수행 시간 측정을 위한. 시작 시간 설정.
start_time = time.time()

gb_clf = GradientBoostingClassifier(random_state=0)
gb_clf.fit(X_train , y_train)
gb_pred = gb_clf.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)

print('GBM 정확도: {0:.4f}'.format(gb_accuracy))
print("GBM 수행 시간: {0:.1f} 초 ".format(time.time() - start_time))
```

GBM 정확도: 0.9561
GBM 수행 시간: 0.5 초

**아까 Ada boost 실습 때 불러온
위스콘신 유방암 데이터 이어서!**

04. GBM

GBM 실습 (분류)

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100, 300, 500],
    'learning_rate': [0.01, 0.05, 0.1]
}
grid_cv = GridSearchCV(gb_clf, param_grid=params, cv=3, verbose=1)
grid_cv.fit(X_train, y_train)
print('최적 하이퍼 파라미터: \n', grid_cv.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

Fitting 3 folds for each of 9 candidates, totalling 27 fits

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 27 out of 27 | elapsed: 24.3s finished
```

최적 하이퍼 파라미터:

```
{'learning_rate': 0.05, 'n_estimators': 500}
```

최고 예측 정확도: 0.9583

```
# GridSearchCV를 이용하여 최적으로 학습된 estimator로 predict 수행.
gb_pred = grid_cv.best_estimator_.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)
print('GBM 정확도: {0:.4f}'.format(gb_accuracy))
```

GBM 정확도: 0.9649

그리드 서치 이용하여 최적의
하이퍼 파라미터값 서칭

05. XGBoost

XGBoost 개요

XGBoost (eXtra Gradient Boost)

- GBM 에 기반한, 트리 기반의 앙상블 학습

GBM 단점

느리다

과적합 이슈



과적합 규제 기능

GBM보다 빠르다.

Tree Purning
(나무 가지치기)

자체 내장 교차검증
→조기종료 기능

뛰어난 예측 성능

결손값 자체 처리

XGBoost 개요

XGBoost 알고리즘

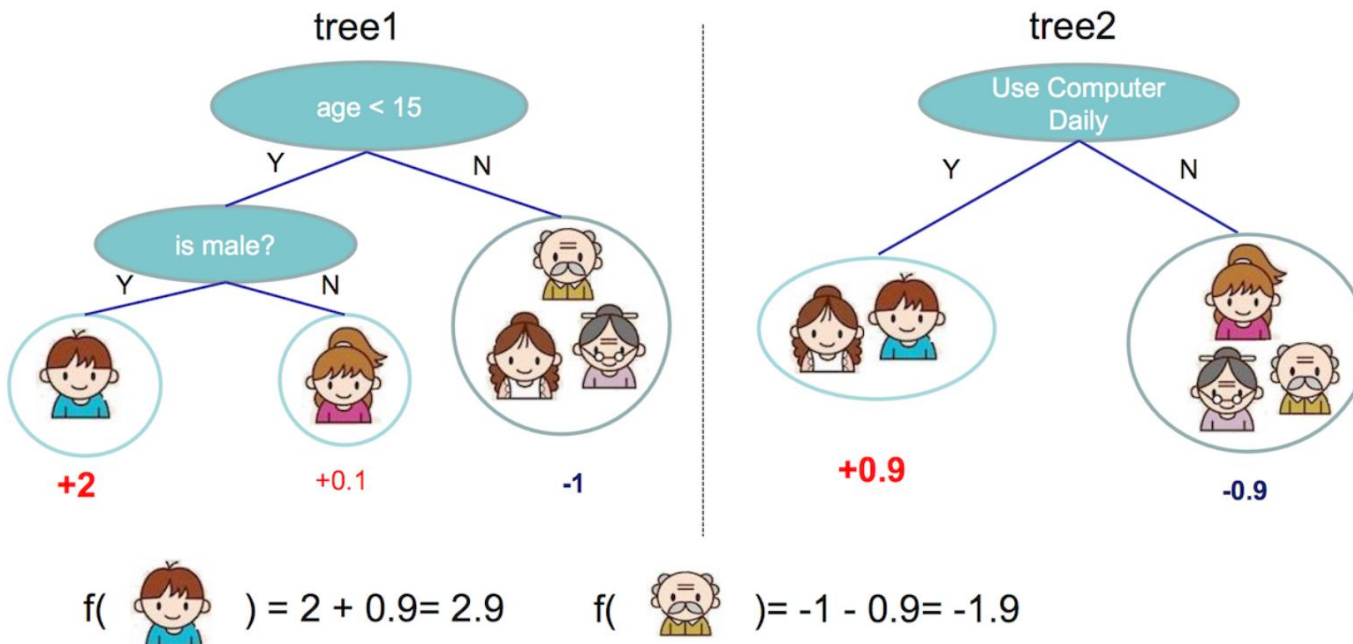
- GBM은 residual을 계속 줄이기 때문에 overfitting이 된다는 문제점이 있음
- 이를 해결하기 위해 GBM에 regularization term을 추가한 것이 XGBoost 알고리즘

$$\text{XGBoost loss function} = \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

XGBoost 개요

XGBoost 알고리즘

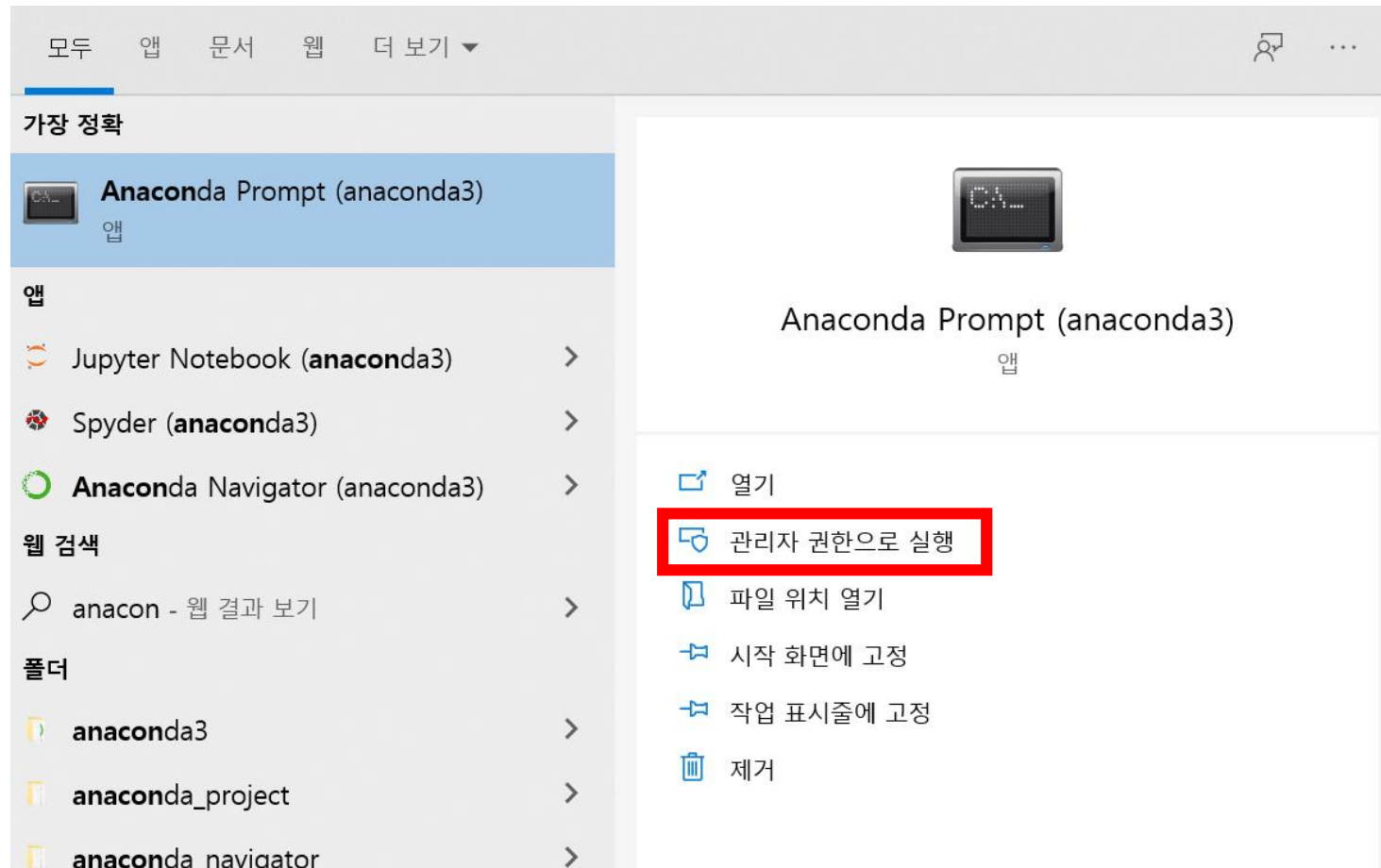
- CART(Classification And Regression Trees) 앙상블 모델을 이용
- 리프 노드 하나에 대해서만 값을 갖는 의사결정 트리와 달리, CART는 모든 리프들이 모델의 최종 스코어에 연관이 있음
- 따라서 의사결정 트리는 분류를 제대로 했는지 초점을 맞춘다면, CART는 **최종 스코어를 비교하면 됨**



Prediction of is sum of scores predicted by each of the tree

XGBoost 설치하기

1. 아나콘다 Command 창을 관리자 권한으로 실행



XGBoost 설치하기

2. conda install -c anaconda py-xgboost 명령어 입력

```
base) C:\Users\samsung> conda install -c anaconda py-xgboost
```

3. Proceed([y]/n)? 에서 Y를 입력

주피터 노트북에서 설치 여부 확인하기

```
In [2]: 1 import xgboost as xgboost
        2 from xgboost import XGBClassifier
```

설치가 되지 않을 경우!

pip install xgboost 입력

```
(base) C:\Users\samsung>pip install xgboost
Collecting xgboost
```

파이썬 래퍼 XGBoost 하이퍼 파라미터

일반 파라미터

- 스레드의 개수나 silent 모드 등의 선택을 위한 파라미터
- 디폴트 파라미터 값을 바꾸는 경우는 거의 X

부스터 파라미터

- 트리 최적화, 부스팅, regularization 등과 관련 파라미터 등을 지칭

학습 태스크 파라미터

- 학습 수행 시의 객체 함수, 평가를 위한 지표 등을 설정하는 파라미터

파이썬 래퍼 XGBoost 하이퍼 파라미터

① 주요 일반 파라미터

booster : gbtree(tree based model) 또는 gblinear(linear model) 선택
디폴트는 gbtree

silent : 출력 메시지 나타냄 여부를 결정하는 파라미터
디폴트는 0, 나타내고 싶지 않을 경우 1

nthread : CPU의 실행 스레드 개수를 조정
멀티 코어 / 스레드 CPU 시스템에서 전체 CPU를 사용하지 않고,
일부 CPU만 사용해 ML 애플리케이션을 구동하는 경우 변동
디폴트는 CPU의 전체 스레드를 다 사용하는 것

파이썬 래퍼 XGBoost 하이퍼 파라미터

② 주요 부스터 파라미터

eta[default=0.3, alias: learning_rate] : GBM의 학습률과 같은 파라미터

부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값

0과 1 사이에서 값을 지정하며, 디폴트는 0.3

보통 0.01~0.2 사이의 값을 선호

※ '사이킷런 래퍼 클래스'에서는 eta는 learning_rate 파라미터로 대체

num_boost_rounds : GBM의 n_estimators와 같은 파라미터

min_child_weight[default=1] : 트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한

데이터들의 weight 총합으로 값이 클수록 분할을 자제

→ 과적합을 조절하기 위해 사용

파이썬 래퍼 XGBoost 하이퍼 파라미터

② 주요 부스터 파라미터

`gamma[default=0, alias: min_split_loss]`

: 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값으로 값이 클수록 과적합 감소 효과가 있음
해당 값보다 큰 손실이 감소된 경우, 리프 노드를 분리
값이 클수록 과적합 감소 효과가 있음

`max_depth[default=6]` : 트리 기반 알고리즘의 max_depth과 같음

값이 높으면 특정 피쳐 조건에 특화되어 룰 조건이 만들어지므로 과적합 가능성이 높아짐
0을 지정하면 깊이에 제한이 없고, 보통 3~10 사이의 값을 적용

`sub_sample[default=1]` : GBM의 subsample과 동일

트리가 커져서 과적합 되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정
0에서 1 사이의 값이 가능하며, 일반적으로 0.5~1 사이의 값을 사용

파이썬 래퍼 XGBoost 하이퍼 파라미터

② 주요 부스터 파라미터

`colsample_bytree[default=1]` : GBM의 `max_features`와 유사

트리 생성에 필요한 피처를 임의로 샘플링 하는 데 사용
매우 많은 피처가 있는 경우 과적합을 조정하는 데 적용

`lambda[default=1, alias: reg_lambda]` : L2 Regularization 적용 값

피처의 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있음

`alpha[default=0, alias: reg_alpha]` : L1 Regularization 적용 값

피처의 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있음

`scale_pos_weight[default=1]` : 특정값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형을 유지하기 위한 파라미터

파이썬 래퍼 XGBoost 하이퍼 파라미터

③ 학습 태스크 파라미터

objective : G최솟값을 가져야 할 손실 함수를 정의

많은 유형의 손실함수를 사용할 수 있는데, 주로 이진 분류인지 다중 분류인지에 따라 결정하여 사용

binary:logistic : 이진 분류일 때 적용

multi:softmax : 다중 분류일 때 적용

손실함수가 multi:softmax일 경우에는 레이블 클래스의 개수인 num_class 파라미터를 지정

multi:softprob : multi:softmax와 유사하나 개별 레이블 클래스의 해당되는 예측 확률을

eval_metrics : 검증에 사용되는 함수를 정의

rmse(기본값) : Root Mean Square Error

Logloss : Negative log-likelihood

Merror : Multiclass classification error rate

Auc : Area under the curve

Mae : Mean Absolute Error

Error : Binary classification error rate(0.5 threshold)

Mlogloss : Multiclass logloss

파이썬 래퍼 XGBoost

과적합 문제가 심각하다면?

- eta값을 낮춘다. (0.01~0.1)

이 경우, num_round 또는 n_estimator는 반대로 높여줘야 한다.

- max_depth 값을 낮춘다.
- Min_child_weight 값을 높인다.
- Gamma 값을 높인다.
- Subsample과 colsample_bytree를 조정한다.

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

1. 데이터 로드 후 살펴보기

```
1 import xgboost as xgb
2 from xgboost import plot_importance
3 import pandas as pd
4 import numpy as np
5 from sklearn.datasets import load_breast_cancer
6 from sklearn.model_selection import train_test_split
7 import warnings
8 warnings.filterwarnings('ignore')
9
10 dataset = load_breast_cancer()
11 X_features = dataset.data
12 y_label = dataset.target
13
14 cancer_df = pd.DataFrame(data=X_features, columns=dataset.feature_names)
15 cancer_df['target'] = y_label
16 cancer_df.head(3)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness		worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	■ ■ ■	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.9	1326.0	0.08474	0.07864		0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.0	1203.0	0.10960	0.15990		0.4504	0.2430	0.3613	0.08758	0

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

2. 레이블 값의 분포 확인

```
1 print(dataset.target_names)
2 print(cancer_df['target'].value_counts())
```

```
['malignant' 'benign']
```

```
1    357
```

```
0    212
```

```
Name: target, dtype: int64
```

3. 80%를 학습용, 20%를 테스트용으로 분할

```
1 # 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
2 X_train, X_test, y_train, y_test=train_test_split(X_features, y_label,
3                                                    test_size=0.2, random_state=156 )
4 print(X_train.shape , X_test.shape)
5
```

```
(455, 30) (114, 30)
```

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

4. Dmatrix로 변환

→파이썬 래퍼 XGBoost는 학습용과 테스트용 데이터 세트를 위해 별도의 객체인 Dmatrix를 생성

```
1 dtrain = xgb.DMatrix(data=X_train , label=y_train)
2 dtest  = xgb.DMatrix(data=X_test  , label=y_test)
```

- Dmatrix는 넘파이 입력 파라미터를 받아 만들어지는 데이터 세트
- Data는 피치 데이터 세트
- Label은 분류의 경우에는 레이블 데이터 세트, 회귀의 경우에는 숫자형인 종속값 데이터 세트

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

5. 하이퍼 파라미터 설정

```
1 params = { 'max_depth':3,  
2            'eta': 0.1,  
3            'objective':'binary:logistic',  
4            'eval_metric':'logloss'  
5          }  
6 num_rounds = 400
```

① 트리 최대 깊이 max_depth → 3

② 학습률 eta → 0.1

③ 목적함수 → 이진 로지스틱(binary:logistic)
(∵ 0 또는 1 이진 분류)

④ 오류 함수의 평가 성능 지표 → logloss

⑤ 부스팅 반복 횟수(num_rounds) → 400

파이썬 래퍼 XGBoost - 위스콘신 유방암 예측

6. XGBoost 모델 학습

```
1 # train 데이터 셋은 'train' , evaluation(test) 데이터 셋은 'eval' 로 명기합니다.
2 wlist = [(dtrain, 'train'), (dtest, 'eval')]
3 # 하이퍼 파라미터와 early stopping 파라미터를 train( ) 함수의 파라미터로 전달
4 xgb_model = xgb.train(params = params , dtrain=dtrain , num_boost_round=num_rounds ,
5                       early_stopping_rounds=100, evals=wlist )

[0]    train-logloss:0.60969    eval-logloss:0.61352
[1]    train-logloss:0.54080    eval-logloss:0.54784
[2]    train-logloss:0.48375    eval-logloss:0.49425
[3]    train-logloss:0.43446    eval-logloss:0.44799
      ■

      ■

[307]   train-logloss:0.005470    eval-logloss:0.085998
[308]   train-logloss:0.005471    eval-logloss:0.085998
[309]   train-logloss:0.005464    eval-logloss:0.085877
[310]   train-logloss:0.005457    eval-logloss:0.085923
[311]   train-logloss:0.00545    eval-logloss:0.085948

Stopping. Best iteration:
[211]   train-logloss:0.006413    eval-logloss:0.085593
```

Early_stopping_rounds → 조기 중단 할 수

있는 최소 반복 횟수 : 100

※ 조기 중단 수행을 위해 반드시 eval_set와

eval_metric가 설정이 되어야 함

※ 반복 시마다 evals에 표시된 데이터 세트에 대해

평가 지표 결과가 출력 됨

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

7. 테스트 데이터 세트 예측 수행

```
1 pred_probs = xgb_model.predict(dtest)
2 print('predict( ) 수행 결과값을 10개만 표시, 예측 확률 값으로 표시됨')
3 print(np.round(pred_probs[:10],3))
4
5 # 예측 확률이 0.5 보다 크면 1, 그렇지 않으면 0 으로 예측값 결정하여 List 객체인 preds에 저장
6 preds = [ 1 if x > 0.5 else 0 for x in pred_probs ]
7 print('예측값 10개만 표시:',preds[:10])
```

predict() 수행 결과값을 10개만 표시, 예측 확률 값으로 표시됨
[0.934 0.003 0.91 0.094 0.993 1. 1. 0.999 0.997 0.]
예측값 10개만 표시: [1, 0, 1, 0, 1, 1, 1, 1, 1, 0]

Predict()는 예측 결과값이 아닌 예측 결과를 추정할 수 있는 확률 값으로 반환

➔ 이진 분류이므로 예측 확률이 0.5보다 크면 1, 그렇지 않으면 0으로 예측값을 결정하는 로직을 추가함

※사이킷런의 predict()는 예측 결과 클리스 0 또는 1을 반환

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

8. XGBoost 모델의 예측 성능 평가

```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score

def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    f1 = f1_score(y_test, pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, F1: {3:.4f}, AUC: {4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

```
get_clf_eval(y_test, preds, pred_probs)
```

오차 행렬

[[35 2]

[1 76]]

정확도: 0.9737, 정밀도: 0.9744, 재현율: 0.9870, F1: 0.9806, AUC: 0.9951

파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

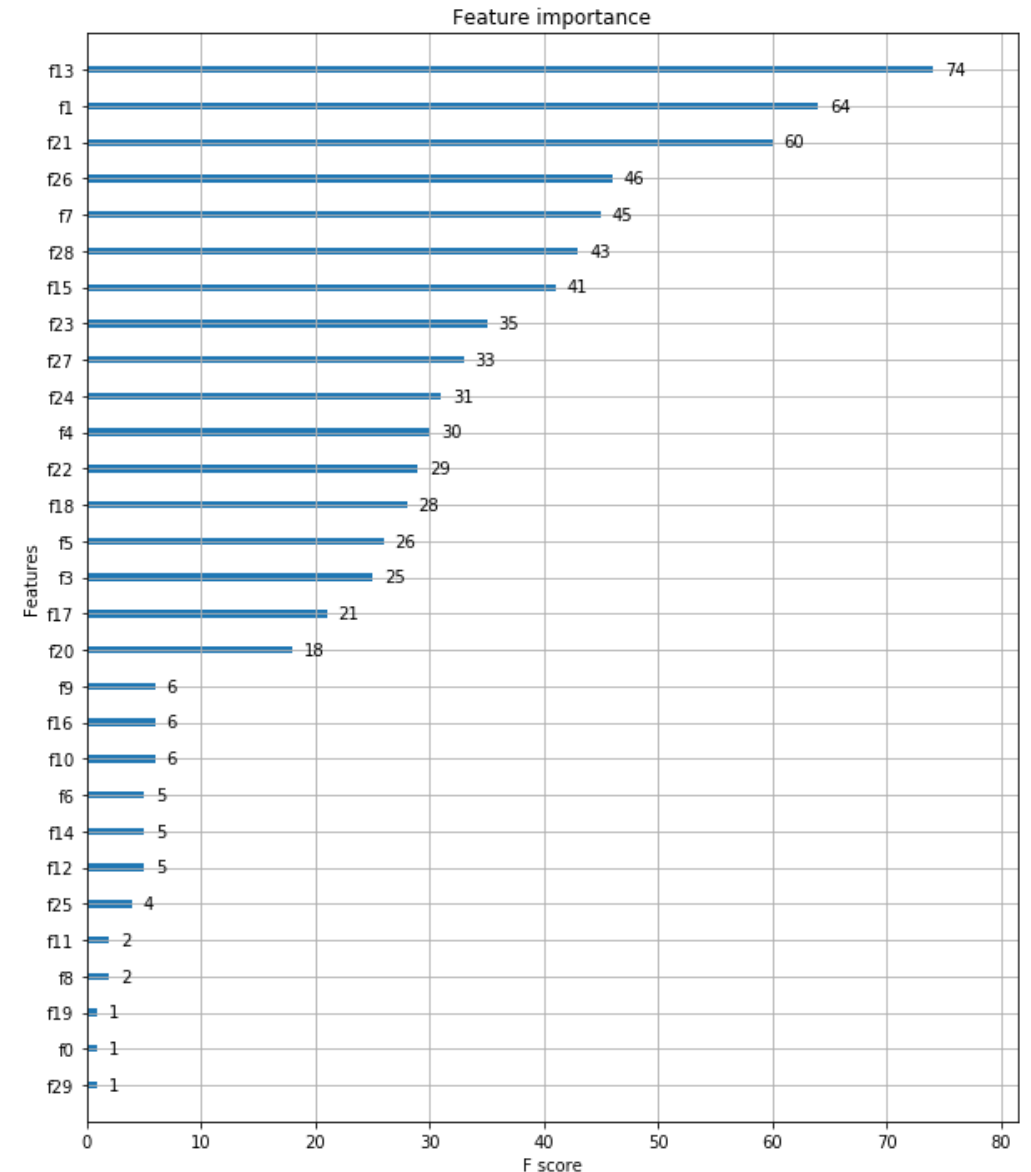
9. XGBoost 모델 시각화

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(xgb_model, ax=ax)
```

Plot_importance()

- 피쳐의 중요도를 막대그래프 형식으로 표현
- F1 스코어를 기준으로 피쳐의 중요도 표현
- 학습이 완료된 모델 객체와 ax객체 입력



파이썬 래퍼 XGBoost – 위스콘신 유방암 예측

※ **xgboost**는 사이킷런의 **GridSearchCV**와 유사하게 데이터 세트에 대한 교차 검증 수행 후 최적 파라미터를 구하는 방법을 **cv()**를 API로 제공

```
xgb.cv(params, dtrain, num_boost_round=10, nfold=3, stratified=False, folds=None, metrics=(), obj=None, feval=None, maximize=False, early_stopping_rounds=None, fpreproc=None, as_pandas=True, verbose_eval=None, show_stdv=True, seed=0, callbacks=None, shuffle=True)
```

- **params(dict)**: 부스터 파라미터
- **dtrain(DMatrix)** : 학습 데이터
- **num_boost_round(int)** : 부스팅 반복횟수
- **nfold(int)** : CV폴드 개수
- **stratified(bool)** : CV수행시 샘플을 균등하게 추출할지 여부
- **metrics(string or list of strings)** : CV 수행시 모니터링할 성능 평가 지표
- **early_stopping_rounds(int)** : 조기중단을 활성화시킴. 반복횟수 지정

사이킷런 래퍼 XGBoost 하이퍼 파라미터

사이킷런 래퍼 XGBoost 특징

Fit()과 predict()만으로 학습과 예측이 가능

XGBClassifier – 분류를 위한 래퍼 클래스 & XGBRegressor – 회귀를 위한 래퍼 클래스

파이썬 래퍼 XGBoost 하이퍼 파라미터 vs 사이킷런 래퍼 XGBoost

- eta → learning_rate
- sub_sample → subsample
- lambda → reg_lambda
- alpha → reg_alpha

사이킷런 래퍼 XGBoost – 위스콘신 유방암 예측

1. 모델 학습 및 예측 수행, 성능 평가

```
1 # 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
2 from xgboost import XGBClassifier
3
4 xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
5 xgb_wrapper.fit(X_train, y_train)
6 w_preds = xgb_wrapper.predict(X_test)
7 w_pred_proba = xgb_wrapper.predict_proba(X_test)[: , 1]
```

```
1 get_clf_eval(y_test , w_preds, w_pred_proba)
```

오차 행렬

[[35 2]

[1 76]]

정확도: 0.9737, 정밀도: 0.9744, 재현율: 0.9870, F1: 0.9806, AUC:0.9951

파이썬 래퍼 XGBoost와 동일한 평가 결과가 나옴

오차 행렬

[[35 2]

[1 76]]

정확도: 0.9737, 정밀도: 0.9744, 재현율: 0.9870, F1: 0.9806, AUC:0.9951

사이킷런 래퍼 XGBoost - 위스콘신 유방암 예측

2. XGBoost 모델 학습 - early_stopping_rounds=10

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
```

```
from xgboost import XGBClassifier
```

```
xgb_wrapper_classifier = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
```

```
evals = [(X_test, y_test)]
```

```
xgb_wrapper_classifier.fit(X_train, y_train, early_stopping_rounds=10, eval_metric="logloss", eval_set=evals, verbose=True)
```

```
ws10_preds = xgb_wrapper_classifier.predict(X_test)
```

```
ws10_pred_proba = xgb_wrapper_classifier.predict_proba(X_test)[:, 1]
```

```
[0] validation_0-logloss:0.61352
```

```
[1] validation_0-logloss:0.54784
```

```
[2] validation_0-logloss:0.49425
```

```
[3] validation_0-logloss:0.44799
```

```
[4] validation_0-logloss:0.40911
```

```
[5] validation_0-logloss:0.37498
```

```
.
```

```
.
```

```
.
```

```
[60] validation_0-logloss:0.09194
```

```
[61] validation_0-logloss:0.09146
```

```
[62] validation_0-logloss:0.09031
```

Early_stopping_rounds → 평가 지표가 향상될

수 있는 반복 횟수

Eval_metric → 조기 종단을 위한 평가 지표

Eval_set → 성능 평가를 수행할 데이터 세트

사이킷런 래퍼 XGBoost – 위스콘신 유방암 예측

2. XGBoost 모델 학습 – early_stopping_rounds=10

```
# 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
```

```
from xgboost import XGBClassifier
```

```
xgb_wrapper_classifier = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
```

```
evals = [(X_test, y_test)]
```

```
xgb_wrapper_classifier.fit(X_train, y_train, early_stopping_rounds=10, eval_metric="logloss", eval_set=evals, verbose=True)
```

```
ws10_preds = xgb_wrapper_classifier.predict(X_test)
```

```
ws10_pred_proba = xgb_wrapper_classifier.predict_proba(X_test)[: , 1]
```

```
[0] validation_0-logloss:0.61352
```

```
[1] validation_0-logloss:0.54784
```

```
[2] validation_0-logloss:0.49425
```

```
[3] validation_0-logloss:0.44799
```

```
[4] validation_0-logloss:0.40911
```

```
[5] validation_0-logloss:0.37498
```

```
⋮
```

```
⋮
```

```
⋮
```

```
[60] validation_0-logloss:0.09194
```

```
[61] validation_0-logloss:0.09146
```

```
[62] validation_0-logloss:0.09031
```

62번에서 72번까지,

10번의 반복 동안 성능 평가 지수가 향상되지 않아 멈춤

→ n_estimators인 400까지 수행하지 않고 72에서

완료

사이킷런 래퍼 XGBoost – 위스콘신 유방암 예측

3. XGBoost 모델 예측 성능 – `early_stopping_rounds=10`

```
: 1 get_clf_eval(y_test, ws10_preds, ws10_pred_proba)
```

오차 행렬

```
[[34  3]
```

```
 [ 2 75]]
```

정확도: 0.9561, 정밀도: 0.9615, 재현율: 0.9740, F1: 0.9677, AUC:0.9947

약간 저조한 성능

➔ 따라서 `early_stopping_rounds`를 100으로 재설정

사이킷런 래퍼 XGBoost – 위스콘신 유방암 예측

3. XGBoost 모델 예측 성능 – early_stopping_rounds=100

```
1 xgb_wrapper_classifier.fit(X_train,y_train,early_stopping_rounds=100,  
2                             eval_metric="logloss",eval_set=evals,verbose=True)  
3  
4 ws100_preds = xgb_wrapper_classifier.predict(X_test)  
5 ws100_pred_proba = xgb_wrapper_classifier.predict_proba(X_test)[: , 1]  
6 get_clf_eval(y_test, ws100_preds, ws100_pred_proba)
```

```
[0]    validation_0-logloss:0.61352  
[1]    validation_0-logloss:0.54784  
[2]    validation_0-logloss:0.49425  
[3]    validation_0-logloss:0.44799  
[4]    validation_0-logloss:0.40911
```

⋮

```
[309]    validation_0-logloss:0.08592  
[310]    validation_0-logloss:0.08592  
[311]    validation_0-logloss:0.08595
```

오차 행렬

```
[[34  3]  
 [ 1 76]]
```

정확도: 0.9649, 정밀도: 0.9620, 재현율: 0.9870, F1: 0.9744, AUC:0.9954

→ 311번 반복까지 수행 후 종료

→ 정확도는 약 0.9649로,

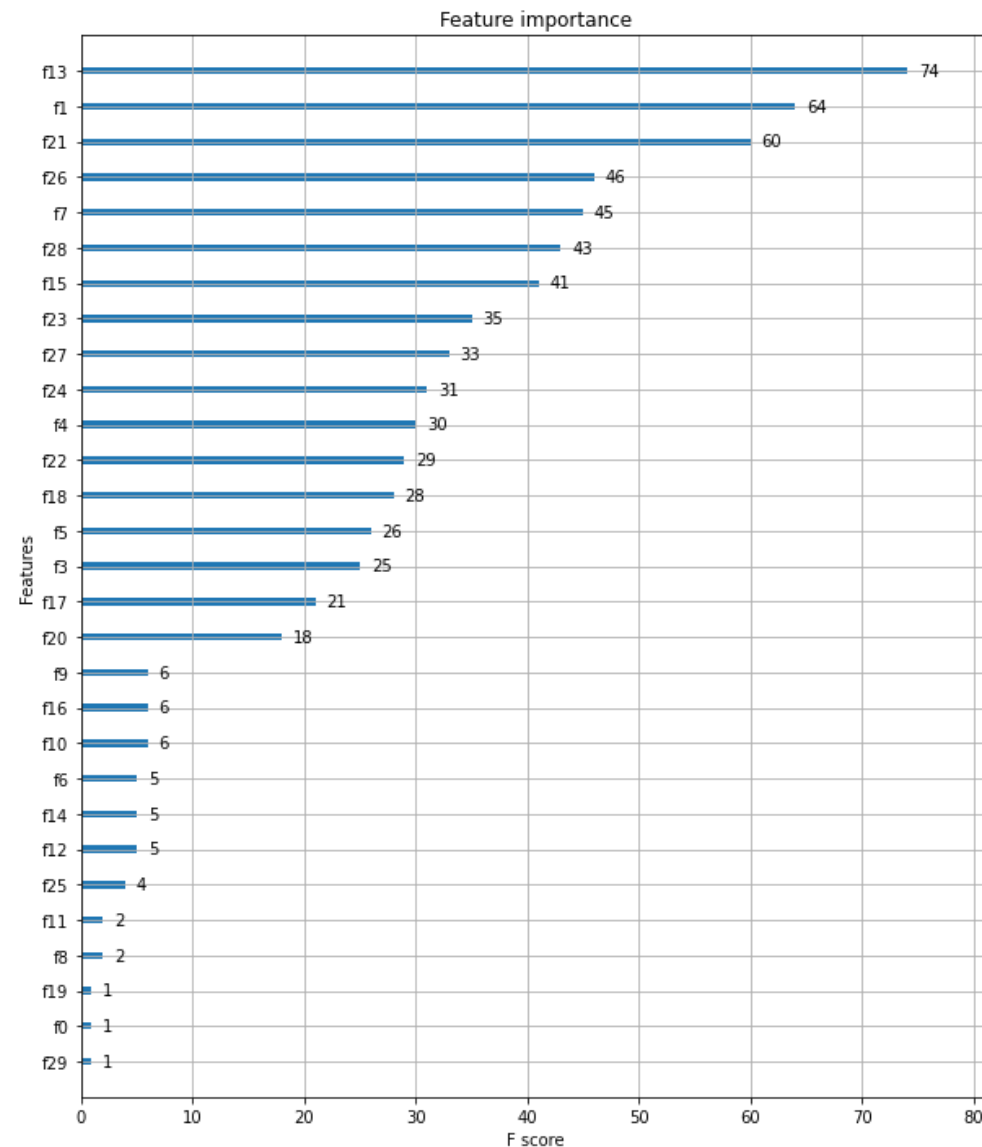
Early_stopping_rounds가 10일 때보다

정확도가 높음

사이킷런 래퍼 XGBoost – 위스콘신 유방암 예측

4. XGBoost 모델 시각화

```
1 from xgboost import plot_importance
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4
5 fig, ax = plt.subplots(figsize=(10, 12))
6 # 사이킷런 래퍼 클래스를 입력해도 무방.
7 plot_importance(xgb_wrapper_classifier, ax=ax)
```



사이킷런 래퍼 XGBoost – 위스콘신 유방암 예측

4. XGBoost 모델 시각화

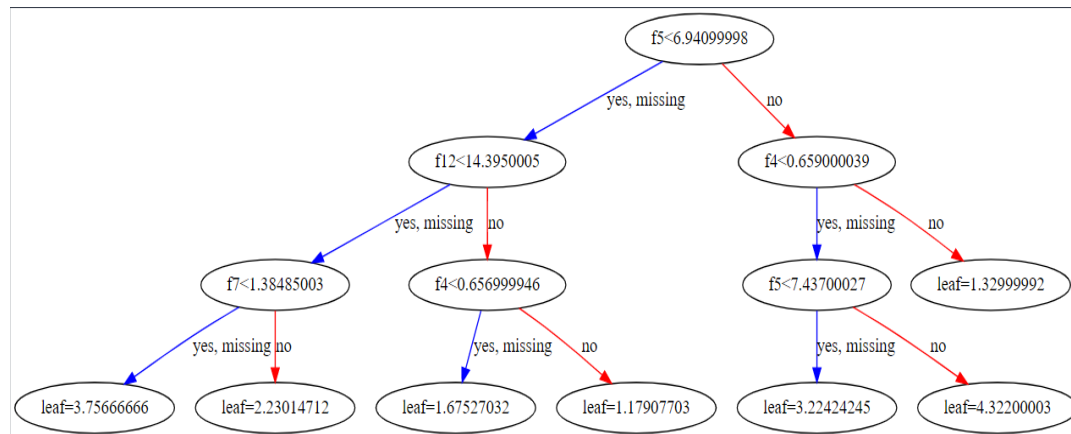
```
1 cross_val = cross_validate(estimator=xgb_wrapper_classifier,  
2                             X=X_features,  
3                             y=y_label,  
4                             cv=5)  
5  
6 print(cross_val['test_score'])  
7 print('avg test score: {:.4f} (+/-{:.4f})'.format(  
8       cross_val['test_score'].mean(), cross_val['test_score'].std()))
```

[0.96491228 0.96491228 0.99122807 0.97368421 0.97345133]
avg test score: 0.9736 (+/-0.0096)

```
1 import graphviz  
2 import matplotlib.pyplot as plt  
3 plt.style.use(['seaborn-whitegrid'])  
4  
5 dot_data = xgb.to_graphviz(xgb_wrapper_classifier)  
6 graph = graphviz.Source(dot_data)  
7 graph
```

<graphviz.files.Source at 0x1f2cab62c88>

```
1 # plot_tree로도 그릴 수 있음  
2 from xgboost import plot_tree  
3  
4 plot_tree(xgb_wrapper_classifier)
```



➔ XGBoost 트리의 의사결정 / 스코어 모델

사이킷런 래퍼 XGBoost – Regressor

■ XGBoost Regressor

```
import xgboost as xgb
from xgboost import plot_importance
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split, cross_validate
import warnings
warnings.filterwarnings('ignore')

dataset = load_boston()
X_features = dataset.data
y_label = dataset.target

cancer_df = pd.DataFrame(data=X_features, columns=dataset.feature_names)
cancer_df['target'] = y_label
cancer_df.head(3)
```

```
1 # 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
2 X_train, X_test, y_train, y_test = train_test_split(X_features,
3                                                     y_label,
4                                                     test_size=0.2,
5                                                     random_state=156)
```

```
1 # 사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
2 from xgboost import XGBRegressor
3
4 xgb_wrapper_regressor = XGBRegressor(n_estimators=400,
5                                     learning_rate=0.1,
6                                     max_depth=3,
7                                     objective='reg:squarederror')
8 xgb_wrapper_regressor.fit(X_train, y_train)
9 w_preds = xgb_wrapper_regressor.predict(X_test)
```

```
1 cross_val = cross_validate(estimator=xgb_wrapper_regressor,
2                             X=X_features,
3                             y=y_label,
4                             cv=5)
5
6 print(cross_val['test_score'])
7 print('avg test score: {:.4f} (+/-{:.4f})'.format(
8     cross_val['test_score'].mean(), cross_val['test_score'].std()))
```

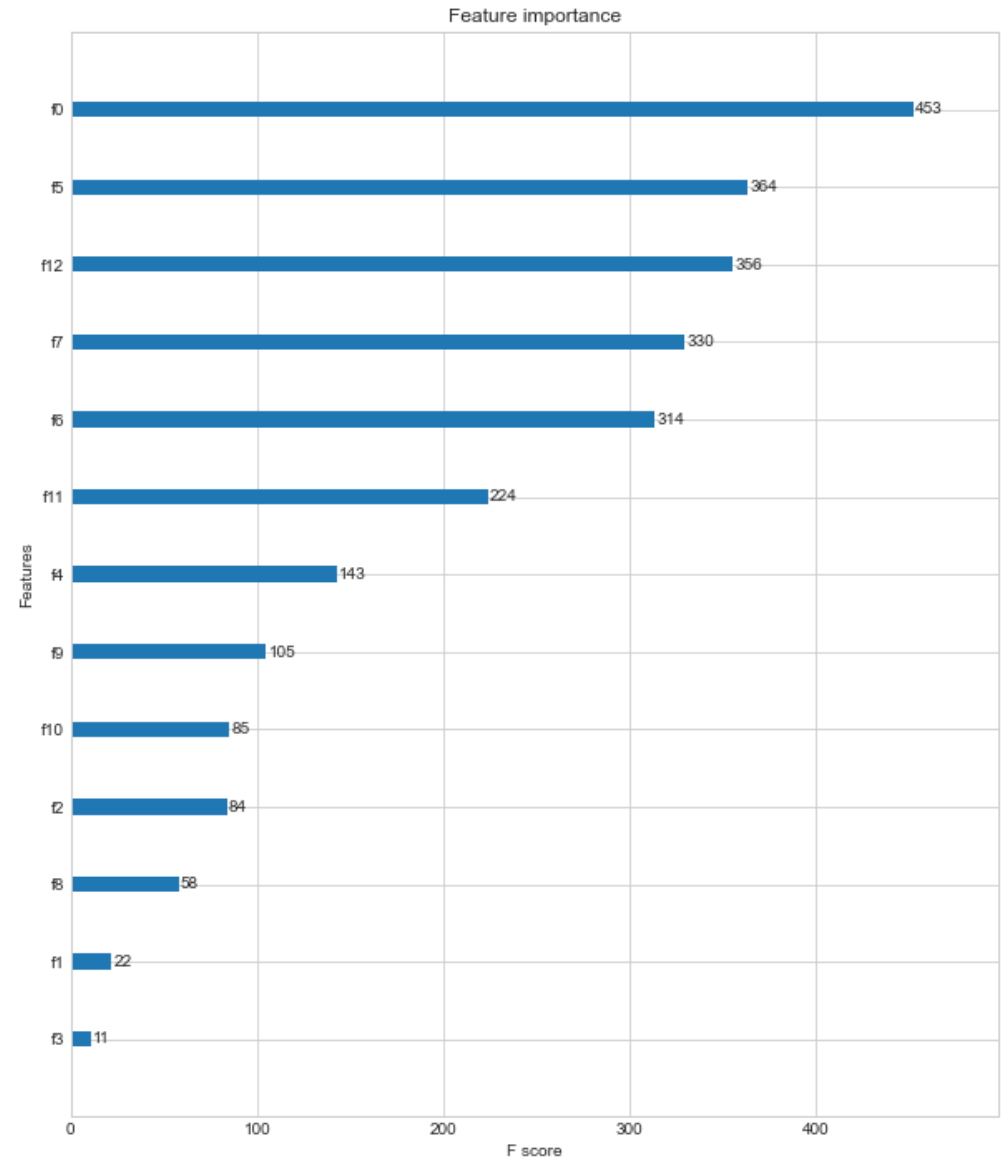
```
[0.78394641 0.83822739 0.82128978 0.58716097 0.41157073]
avg test score: 0.6884 (+/-0.1650)
```

사이킷런 래퍼 XGBoost – Regressor

■ XGBoost Regressor

```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(xgb_wrapper_regressor, ax=ax)
```



06. LightGBM

LightGBM 개요

LightGBM (Light Gradient Boost)

- 중심 트리 분할(Leaf Wise) 방식

GBM 단점

느리다

과적합 이슈

XGBoost 단점

GMB보다 빠르지만
여전히 느리다

많은 하이퍼
파라미터

빠른 학습과 예측 시간

더 적은 메모리 사용

범주형 피처의
자동 변환과 최적 분할

LightGBM 개요

LightGBM 장/단점

– 장점

- (1) XGBoost 대비 **대용량 데이터**에 대해 더 **빠른 학습**과 **예측 수행 시간** (GPU 지원)
- (2) **더 작은 메모리** 사용량
- (3) **범주형 피처**의 자동 변환과 최적 분할
 - : 원핫 인코딩 등 사용하지 않고도 범주형 피처를 최적으로 변환하고 이에 따른 노드 분할 수행

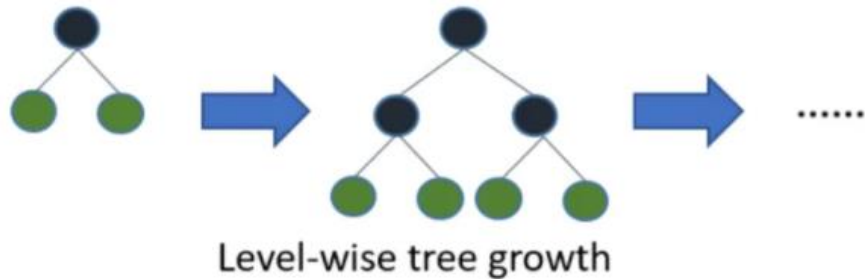
– 단점

- (1) **적은 데이터 세트**에 적용할 경우 **과적합 발생 쉬움**
 - * 행 수에 대한 제한은 없지만 10,000 이상의 행을 가진 데이터에 사용하는 것을 권유

LightGBM 개요

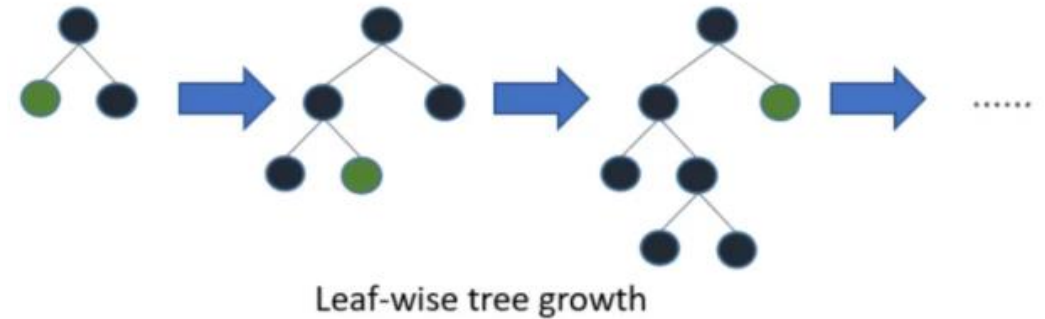
LightGBM 알고리즘

다른 GBM 계열 트리 분할 방식 (Level-Wise)



- tree가 수평적으로 확장
 - 균형을 잡아주기 때문에 depth를 줄어듦
- => 과적합에 강한 구조
=> but 균형을 잡아주기 위해 연산이 추가적으로 들어감

LightGBM 트리 분할 방식 (Leaf-Wise)



- tree가 수직적으로 확장 (균형 맞추기X)
 - max delta loss를 가진 leaf를 선택해서 분할
- => 비대칭적이고 깊은 tree가 생성
=> but 예측 오류 손실 최소화

LightGBM 설치하기

conda install -c conda-forge lightgbm 명령어 입력 후
Procced([y]/n)? 에서 Y를 입력

```
(base) C:\₩₩₩₩₩₩>conda install -c conda-forge lightgbm
```

설치 여부 확인

```
import lightgbm  
  
print(lightgbm.__version__)
```

3.1.1

LightGBM 하이퍼 파라미터

① 주요 파라미터

num_iterations [default=100] : 반복 수행하려는 트리 개수 지정, 너무 크면 과적합 발생

learning_rate [default= 0.1] : 학습률

max_depth [default=-1] : 최대 깊이 (0보다 작은 값 입력 시 깊이 제한 없음)

min_data_in_leaf [default=20] : 최종 리프 노드가 되기 위한 레코드 수, 과적합 제어용

num_leaves [default=31] : 하나의 트리가 가지는 최대 리프 개수

boosting [default=gbdt] : 부스팅 트리 생성 알고리즘 (gbdt - 일반적인 그래디언트 부스팅 트리 / rf - 랜덤 포레스트)

bagging_fraction [default=1.0] : 데이터 샘플링 비율 지정, 과적합 제어용

feature_fraction [default=1.0] : 개별 트리 학습 시 무작위로 선택되는 피처 비율, 과적합 제어용

lambda_l2 [default=0.0] : L2 Regularization 적용 값, 피처 개수 많을 때 적용 검토, 클수록 과적합 감소 효과

lambda_l1 [default=0.0] : L1 Regularization 적용 값, 피처 개수 많을 때 적용 검토, 클수록 과적합 감소 효과

② 학습 태스크 파라미터 **objective** : 최솟값을 가져야할 손실함수 정의

LightGBM 하이퍼 파라미터

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

LightGBM 하이퍼 파라미터

하이퍼 파라미터 튜닝 방안

기본 튜닝 방안

: `num_leaves`의 개수를 중심으로 `min_child_samples`(`min_data_in_leaf`), `max_depth`를 함께 조절하면서 모델의 복잡도를 줄이는 것

- `num_leaves`를 늘리면 정확도가 높아지지만 **트리가 깊어지고 과적합되기 쉬움**
- `min_child_samples`(`min_data_in_leaf`)를 크게 설정하면 **트리가 깊어지는 것을 방지**
- `max_depth`는 명시적으로 깊이를 제한. 위의 두 파라미터와 함께 **과적합을 개선하는데 사용**

또한, `learning_rate`을 줄이면서 `n_estimator`를 크게하는 것은 부스팅에서의 기본적인 튜닝 방안

LGBMClassifier – 위스콘신 유방암 예측

1. 모델 학습 수행

```
# LightGBM의 파이썬 패키지인 lightgbm에서 LGBMClassifier 임포트
from lightgbm import LGBMClassifier
from lightgbm import plot_importance, plot_metric, plot_tree

import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_validate

dataset = load_breast_cancer()
ftr = dataset.data
target = dataset.target

# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test = train_test_split(ftr,
                                                    target,
                                                    test_size=0.2,
                                                    random_state=156)

# 앞서 XGBoost와 동일하게 n_estimators는 400 설정.
lgbm_wrapper_classifier = LGBMClassifier(n_estimators=400)

# LightGBM도 XGBoost와 동일하게 조기 중단 수행 가능.
evals = [(X_test, y_test)]
lgbm_wrapper_classifier.fit(X_train,
                            y_train,
                            early_stopping_rounds=100,
                            eval_metric="logloss",
                            eval_set=evals,
                            verbose=True)

preds = lgbm_wrapper_classifier.predict(X_test)
pred_proba = lgbm_wrapper_classifier.predict_proba(X_test)[:, 1]
```

```
[142] valid_0's binary_logloss: 0.196367
[143] valid_0's binary_logloss: 0.19869
[144] valid_0's binary_logloss: 0.200352
[145] valid_0's binary_logloss: 0.19712
Early stopping, best iteration is:
[45] valid_0's binary_logloss: 0.122818
```

→ 145번 반복까지 수행 후 종료

LGBMClassifier – 위스콘신 유방암 예측

2. 예측 성능 평가

```
get_clf_eval(y_test, preds, pred_proba)
```

오차 행렬

```
[[33  4]
```

```
 [ 1 76]]
```

정확도: 0.9561, 정밀도: 0.9500, 재현율: 0.9870, F1: 0.9682, AUC: 0.9905

```
cross_val = cross_validate(estimator=lgbm_wrapper_classifier,  
                           X=ftr,  
                           y=target,  
                           cv=5)  
  
print(cross_val['test_score'])  
print('avg test score: {:.4f} (+/-{:.4f})'.format(cross_val['test_score'].mean(),  
                                                  cross_val['test_score'].std()))
```

```
[0.94736842 0.96491228 0.99122807 0.98245614 0.98230088]
```

```
avg test score: 0.9737 (+/-0.0157)
```

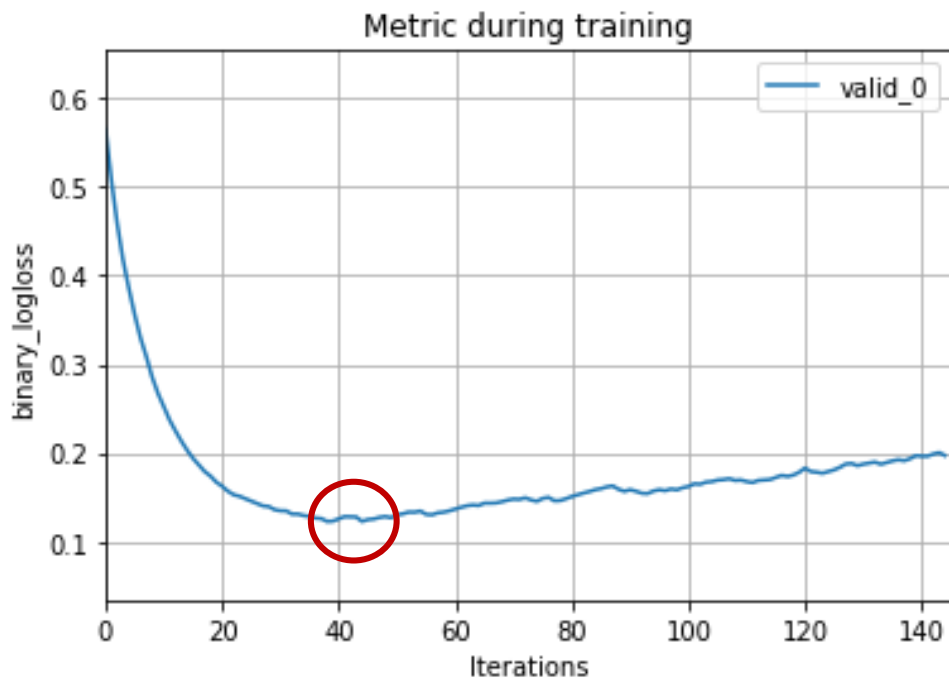
→ 정확도는 약 0.9561

→ 교차검증 시, 평균 검증 정확도 : 0.9737

LGBMClassifier – 위스콘신 유방암 예측

3. LightGBM 모델 시각화

```
plot_metric(lgbm_wrapper_classifier)
```



이전 코드 결과창 확인해보면 145(45)번째에서 값에서 종료

```
[142] valid_0's binary_logloss: 0.196367
[143] valid_0's binary_logloss: 0.19869
[144] valid_0's binary_logloss: 0.200352
[145] valid_0's binary_logloss: 0.19712
Early stopping, best iteration is:
[45] valid_0's binary_logloss: 0.122818
```

→ training 시 log-loss가 얼마나 떨어지는지 확인

→ 40 부근에서 log-loss값이 가장 떨어짐

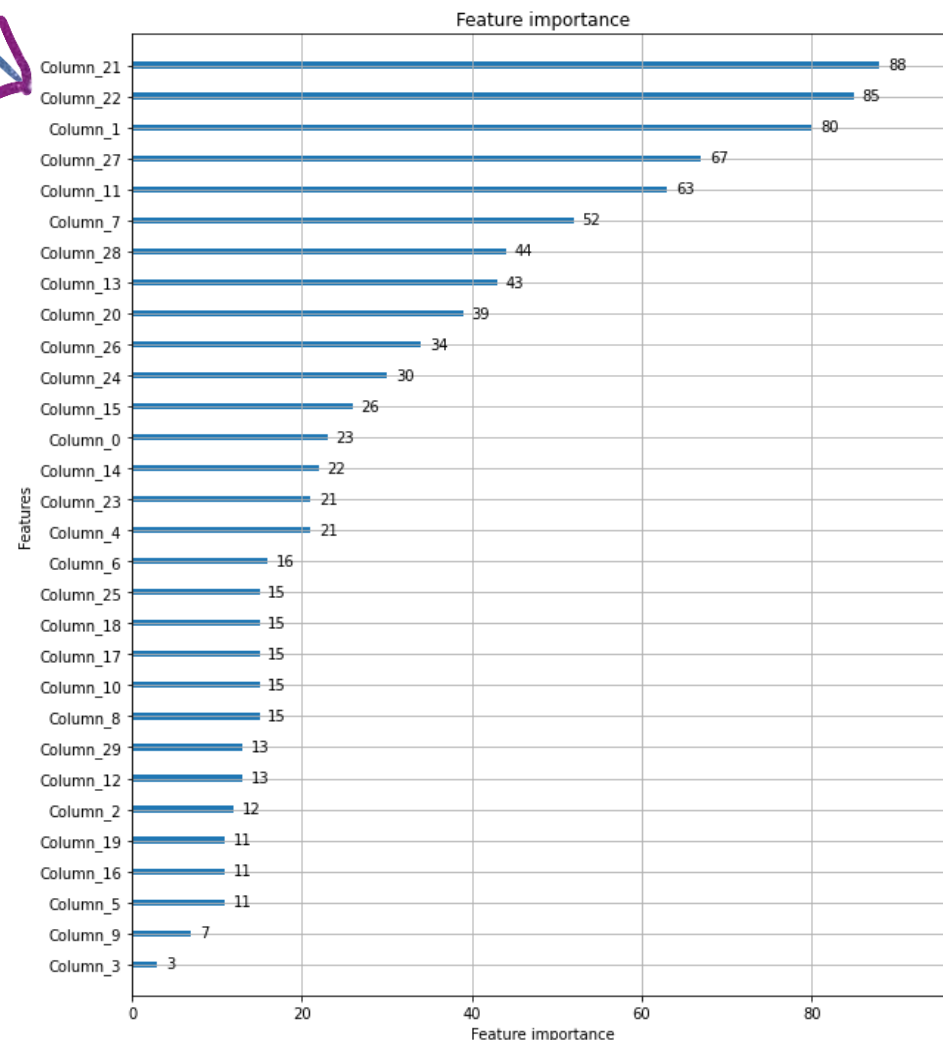
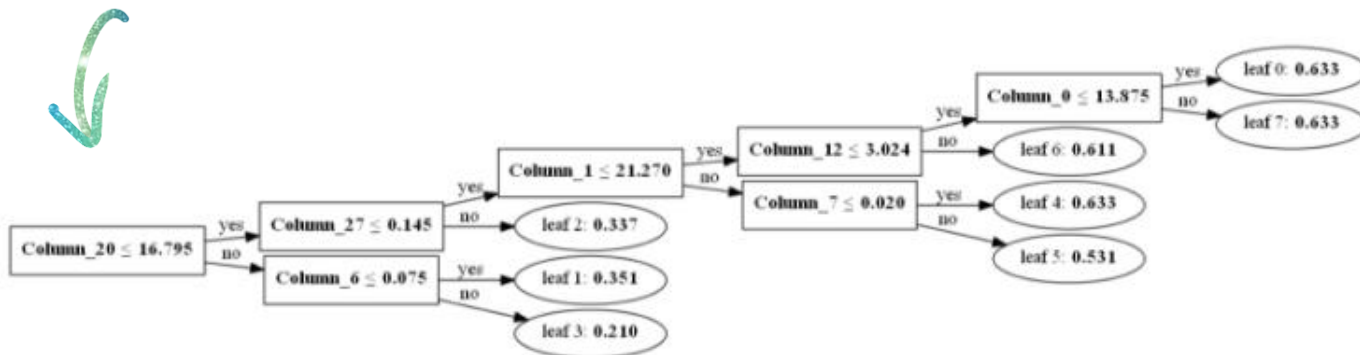
LGBMClassifier – 위스콘신 유방암 예측

3. LightGBM 모델 시각화

```
# plot_importance( )를 이용하여 feature 중요도 시각화
from lightgbm import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline
```

```
fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(lgbm_wrapper_classifier, ax=ax)
```

```
plot_tree(lgbm_wrapper_classifier, figsize=(28, 24))
```



LGBMRegressor - 보스턴 주택 가격 데이터

1. 모델 학습 수행

```
from lightgbm import LGBMRegressor
from lightgbm import plot_importance, plot_metric, plot_tree

import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split

dataset = load_boston()
ftr = dataset.data
target = dataset.target

# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test = train_test_split(ftr,
                                                    target,
                                                    test_size=0.2,
                                                    random_state=156)
```

```
# 앞서 XGBoost와 동일하게 n_estimators는 400 설정.
lgbm_wrapper_regressor = LGBMRegressor(n_estimators=400)

# LightGBM도 XGBoost와 동일하게 조기 중단 수행 가능.
evals = [(X_test, y_test)]
lgbm_wrapper_regressor.fit(X_train,
                           y_train,
                           early_stopping_rounds=100,
                           eval_metric="logloss",
                           eval_set=evals,
                           verbose=True)

preds = lgbm_wrapper_regressor.predict(X_test)
```

```
[190] valid_0's l2: 8.27979
[191] valid_0's l2: 8.26597
[192] valid_0's l2: 8.25721
[193] valid_0's l2: 8.25542
[194] valid_0's l2: 8.25451
Early stopping, best iteration is:
[94] valid_0's l2: 8.21052
```

→ 194번 반복까지 수행 후 종료

LGBMRegressor – 보스턴 주택 가격 데이터

2. 예측 성능 평가

```
cross_val = cross_validate(estimator=lgbm_wrapper_regressor,  
                           X=ftr,  
                           y=target,  
                           cv=5)  
  
print(cross_val['test_score'])  
print('avg test score: {:.4f} (+/-{:.4f})'.format(cross_val['test_score'].mean(),  
                                                  cross_val['test_score'].std()))
```

```
[0.73551531 0.80257021 0.78554793 0.5047667 0.01783397]  
avg test score: 0.5692 (+/-0.2957)
```

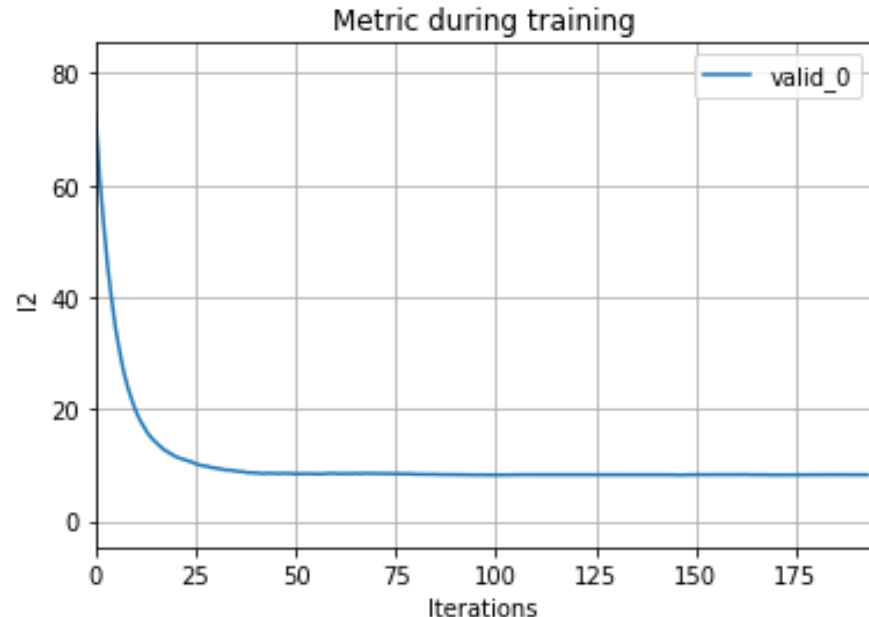
→ 교차검증 시, 평균 검증 정확도 : 0.5692

→ 회귀 데이터에서는 정확도가 떨어짐

LGBMRegressor – 보스턴 주택 가격 데이터

3. LightGBM 모델 시각화

```
plot_metric(lgbm_wrapper_regressor)
```



이전 코드 결과창 확인해보면 194(94)번째에서 값에서 종료

```
[190] valid_0's l2: 8.27979
[191] valid_0's l2: 8.26597
[192] valid_0's l2: 8.25721
[193] valid_0's l2: 8.25542
[194] valid_0's l2: 8.25451
Early stopping, best iteration is:
(94) valid_0's l2: 8.21052
```

→ training 시 loss가 얼마나 떨어지는지 확인

→ 94 부근에서 log-loss값이 가장 떨어짐

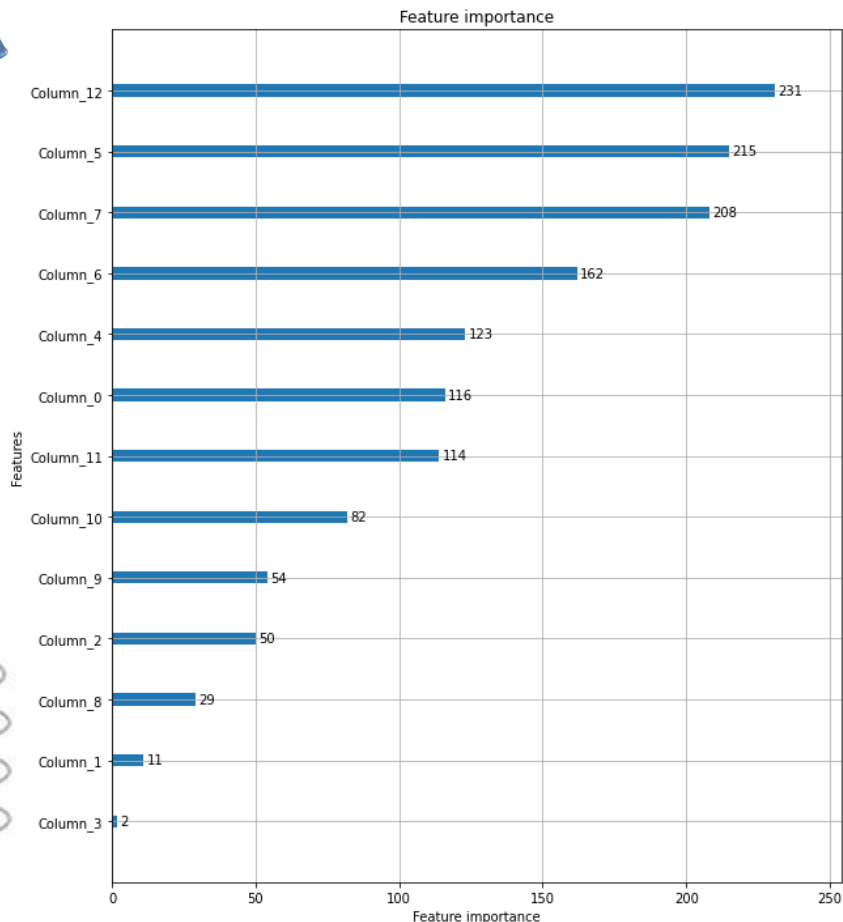
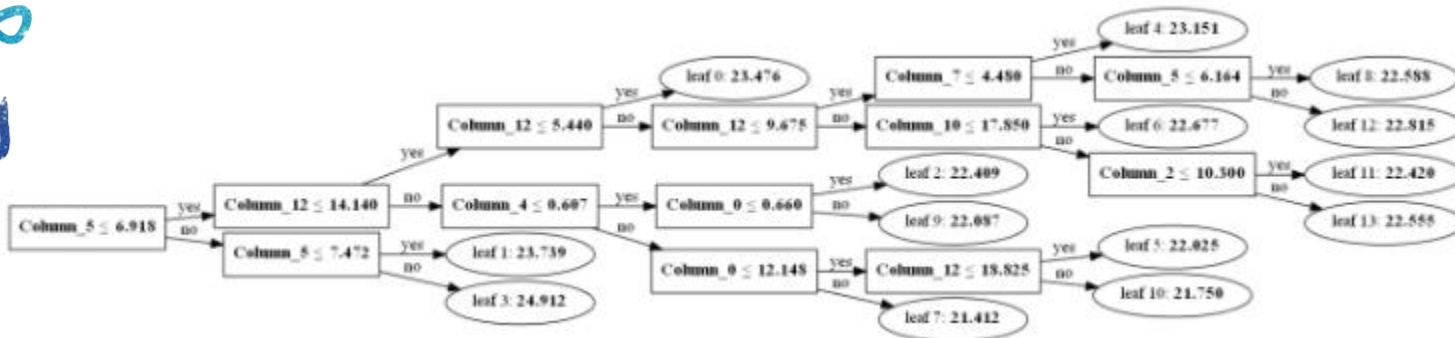
LGBMRegressor – 보스턴 주택 가격 데이터

3. LightGBM 모델 시각화

```
# plot_importance( )를 이용하여 feature 중요도 시각화
from lightgbm import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline
```

```
fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(lgbm_wrapper_regressor, ax=ax)
```

```
plot_tree(lgbm_wrapper_regressor, figsize=(28, 24))
```



Q&A

감사합니다.