4️⃣

# 04 _ 분류 part2

## 05. GBM(Gradient Boosting Machine)

### Boosting

> 💡 부스팅 알고리즘은 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측 한 데이터나 학습 트리에 가중치 부여를 통해 오류들 개선해 나가면서 학습하는 방식
> • 부스팅의 대표적인 구현은 AdaBoost(Adaptive boosting)와 그래디언트 부스트가 있음

### GBM 개요

**(1) AdaBoost**

- 개별 약한 학습기에 각각 가중치를 부여한 후, 모두 결합해 예측을 수행한다.

**(2) GBM**

- 에이다부스트와 유사하나, 가중치 업데이트를 경사 하강법을 이용하는 것이 큰 차이

📝 **경사 하강법(Gradient Descent)**

오류식을 최소화하는 방향성을 가지고 반복적으로 가중치 값을 업데이트한다.

즉, 반복 수행을 통해 오류를 최소화할 수 있도록 가중치의 업데이트 값을 도출하는 기법

> 오류식:

> $h(x) = y - F(x)$
>
> $y$: 실제 결괏값
>
> $x1, x2, \cdots, xn$: 피처
>
> $F(x)$: 피처에 기반한 예측 함수

## 사이킷런 GBM 주요 하이퍼 파라미터 및 튜닝

- `loss`: 경사 하강법에서 사용할 비용 함수를 지정한다. 특별한 이유가 없다면 디폴트 값 "deviance"를 적용한다.
- `learning_rate`: GBM이 학습을 진행할 때마다 적용하는 학습률로서 weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수.

  0~1 사이의 값을 지정할 수 있고 디폴트는 0.1

  **값이 작을수록 최소 오류 값을 찾아 예측 성능이 높아질 가능성이 크지만 수행시간이 오래 걸림.**
- `n_estimators`: weak learner의 갯수로 디폴트 값은 100이다.
- `subsample`: weak learner가 학습에 사용하는 데이터의 샘플링 비율로 디폴트 값은 1이다.

  과적합이 염려되는 경우 1보다 작은 값으로 설정한다.

# 06. XGBoost(eXtra Gradient Boost)

## XGBoost 개요

💡 트리 기반의 앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나

- 뛰어난 예측 성능
- GBM 대비 빠른 수행 시간: 병렬 학습 가능
- 과적합 규제 (Regularization): 과적합 규제 기능 제공
- 나무 가지치기 (Tree pruning): 더 이상 긍정 이득이 없는 분할 가지치기
- 자체 내장된 교차 검증: 조기 중단 가능
- 결측값 자체 처리

# 파이썬 래퍼 XGBoost 하이퍼 파라미터

### 일반 파라미터

일반적으로 실행 시 스레드의 갯수나 silent 모드 등의 선택을 위한 파라미터로서 디폴트 파라미터 값을 바꾸는 경우는 거의 없다.

- **booster**: gbtree 또는 gblinear 중 선택, 디폴트는 gbtree
- **silent**: 디폴트는 0이며, 출력 메시지를 나타내고 싶지 않은 경우 1로 설정
- **nthread**: CPU의 실행 스레드 개수로 디폴트는 전체 CPU의 스레드를 사용한다.

### 부스터 파라미터

트리 최적화, 부스팅, regularization 등과 관련 파라미터 등을 지칭한다.

- **eta [defalut: 0.3, alias: learning_rate]**: GBM의 학습률(learning_rate)와 같은 파라미터로 보통 0.01 ~ 0.2 사이의 값을 선호한다.
- **num_boost_round**: GBM의 n_estimators와 같은 파라미터
- **min_child_weight [default: 1]**: 트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 가중치 총합, 값이 클수록 분할을 자제한다. 과적합을 조절하기 위해 사용

- **gamma [default: 0, aias: min_split_loss]**: 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값, 해당 값보다 큰 손실(loss)이 감소된 경우 리프 노드를 분리하며 값이 클수록 과적합 감소 효과가 있다.
- **max_depth [default: 6]**: 트리 기반 알고리즘의 max_depth와 같다. 0을 지정하면 깊이 제한이 없다. 보통은 3 ~ 10 사이의 값을 적용한다.
- **sub_sample [default: 1]**: GBM의 subsample과 같은 파라미터로 일반적으로 0.5 ~ 1 사이의 값을 사용한다.
- **colsample_bytree [default: 1]**: GBM의 max_features와 유사하다. 트리 생성에 필요한 피처를 임의로 샘플링 하는데 사용, 피처가 매우 많은 경우 과적합을 조절하는데 사용
- **lambda [default: 1, alias: reg_lambda]**: L2 Regularization 적용 값으로 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있다.
- **alpha [default: 0, alias: reg_alpha]**: L1 Regularization 적용 값으로 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있다.
- **scale_pos_weight [default: 1]**: 특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 셋의 균형 유지를 위한 파라미터

### 학습 태스크 파라미터

학습 수행시의 객체 함수, 평가를 위한 지표 등을 설정하는 파라미터

- **objective**: 최솟값을 가져야할 손실함수를 정의한다. 주로 사용하는 손실함수는 이진 분류인지 다중분류인지에 따라 다르다.
- **binary:logistic**: 이진 분류일 때 사용
- **multi:softmax**: 다중 분류일 때 적용, 손실함수가 multi:softmax인 경우에는 레이블 클래스 개수인 num_class 파라미터를 지정해야 한다.
- **multi:softprob**: multi:softmax와 유사하지만 개별 레이블 클래스의 해당되는 예측 확률을 반환한다.
- **eval_metric**: 검증에 사용되는 함수를 정의한다. 기본값은 회귀의 경우 rmse, 분류인 경우 error이다.
    - rmse: Root Mean Square Error
    - mae: Mean Absolute Error
    - logloss: Negative log-likelihood

- error: Binary classification error rate (0.5 threshold)
- merror: Multiclass classification error rate
- mlogloss: Multiclass logloss
- auc: Area Under the Curve

**과적합 문제가 심각한 경우**

- eta 값을 낮춘다(0.01 ~ 0.1). eta 값을 낮출 경우 num_boost_rounds는 반대로 높여줘야 한다.
- max_depth 값을 낮춘다.
- min_child_weight 값을 높인다.
- gamma 값을 높인다.
- sub_sample과 colsample_bytree 등을 조정한다.

## 사이킷런 래퍼 XGBoost의 개요 및 적용

💡 사이킷런 래퍼 XGBoost의 하이퍼 파라미터는 파이썬 래퍼와 일부 차이가 있다.

- eta [defalut: 0.3] →→ learning_rate [defalut: 0.1]
- sub_sample →→ subsample
- lambda →→ reg_lambda
- alpha →→ reg_alpha
- num_boost_round →→ n_estimators

# 07. LightGBM

💡 **LightGBM 장단점**

- XGB보다도 학습에 걸리는 시간이 훨씬 적으며 메모리 사용량도 상대적으로 적다.
- 카테고리형 피처를 자동 변환하고, 예측 성능 역시 큰 차이가 없다.
- 데이터의 갯수가 적을 경우 과적합이 발생하기 쉽다.
- 적은 데이터의 갯수에 대한 기준은 애매하지만 LightGBM의 공식 문서에서 10,000건 이하라고 기술하였다.
-

**트리 기반 알고리즘 특징**

- 기존의 대부분 트리 기반 알고리즘은 트리의 깊이를 효과적으로 줄이기 위한 균형 트리 분할(Level Wise) 방식을 사용한다.
- 최대한 균형 잡힌 트리를 유지하면서 분할하기 때문에 깊이가 최소화 되며 오버피팅에 보다 강한 구조를 가진다.
- 하지만 균형을 맞추기 위한 시간이 오래 걸리는 단점이 있다.
-

**LightGBM 특징**

- LightGBM은 일반 GBM 계열의 트리 분할 방법과 다르게 리프 중심 트리 분할(Leaf Wise) 방식을 사용한다.
- 트리의 균형을 맞추지 않고 최대 손실 값(max delta loss)을 가지는 리프 노드를 지속적으로 분할하여 깊이가 증가하고 비대칭적인 트리를 생성한다.
- 이렇게 생성된 트리는 학습을 반복할수록 결국은 균형 트리 분할 방식보다 예측 오류 손실을 최소화 할 수 있다는 것이 LightGBM 구현사상이다.

# LightGBM 하이퍼 파라미터

*!! 사이킷런 기준*

**주요 파라미터**

- **n_estimators [default: 100]**: GBM과 XGB의 n_estimators와 같은 파라미터

- **learning_rate [defalut: 0.1]**: GBM과 XGB의 학습률(learning_rate)과 같은 파라미터, 일반적으로 n_estimators를 높이고 learning_rate를 낮추면 예측 성능이 향상하지만 마찬가지로 과적합 이슈 및 소요 시간 증가의 문제가 있다.
- **max_depth [default: 1]**: 트리 기반 알고리즘의 max_depth와 같다. 0보다 작은 값을 지정하면 깊이 제한이 없다. LightGBM은 Leaf Wise 방식이므로 깊이가 상대적으로 더 깊다.
- **min_child_samples [default: 20]**: 결정 트리의 min_samples_leaf와 같은 파라미터로 리프 노드가 되기 위해 최소한으로 필요한 샘플 수
- **num_leaves [default: 31]**: 하나의 트리가 가질 수 있는 최대 리프 개수
- **boosting [default: gbdt]**: 부스팅의 트리를 생성하는 알고리즘을 지정하며 gbdt는 일반적인 그래디언트 부스팅 결정 트리이며 rf는 랜덤 포레스트이다.
- **subsample [default: 1]**: GBM과 XGB의 subsample과 같은 파라미터
- **colsample_bytree [default: 1]**: XGB의 colsample_bytree와 같은 파라미터로 개별 트리를 학습할 때마다 무작위로 선택하는 피처의 비율
- **reg_lambda [default: 0]**: XGB의 reg_lambda와 같은 파라미터로 L2 regulation 제어를 위한 값이다. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있다.
- **reg_alpha [default: 0]**: XGB의 reg_alpha와 같은 파라미터로 L1 regulation 제어를 위한 값이다. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있다.
-

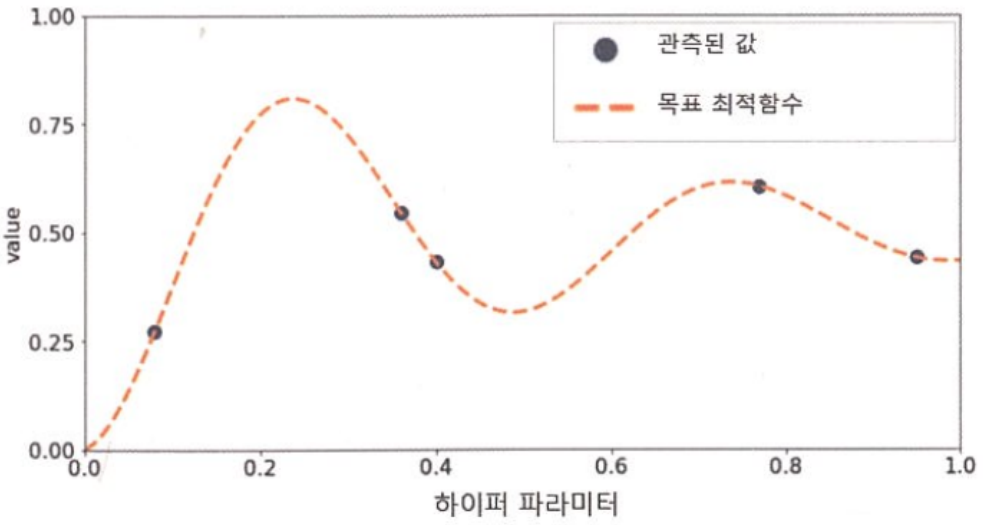
**학습 태스크 파라미터**

- **objective**: 최솟값을 가져야할 손실함수를 정의한다. XGB의 objective 파라미터와 동일하다.

# 08. 베이지안 최적화 기반의 HyperOpt를 이용한 하이퍼 파라미터 튜닝

# 베이지안 최적화 단계

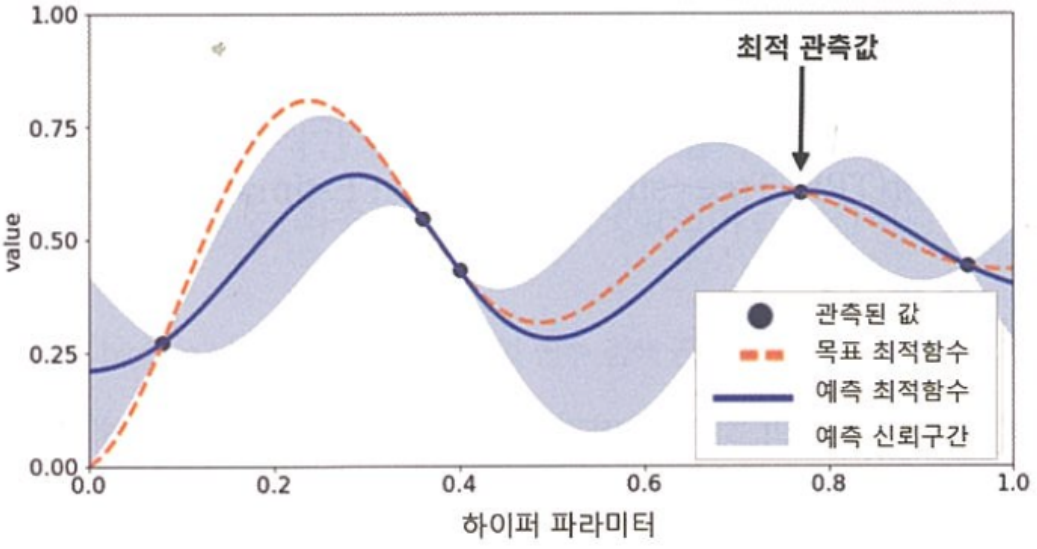
Step1) 랜덤하게 하이퍼 파라미터 샘플링, 성능 결과 관측




Step2) 관측 값 기반으로 대체 모델이 최적 함수 추정 (파란색 선)

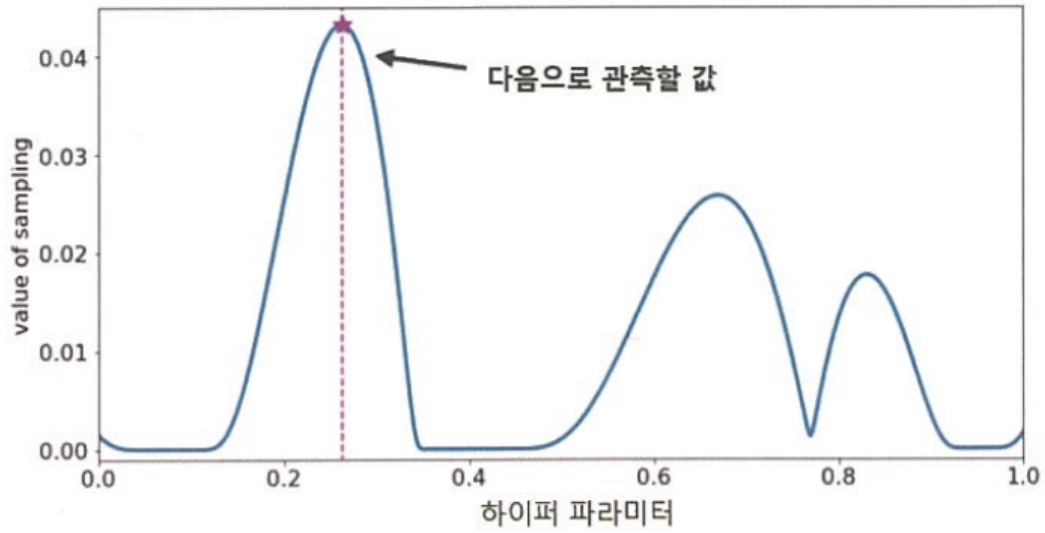
`신뢰 구간` : 추정 함수의 결과값 오류 편차, 추정 함수의 불확실성

`최적 관측값` : y축에서 가장 높은 값을 가질 때의 하이퍼 파라미터

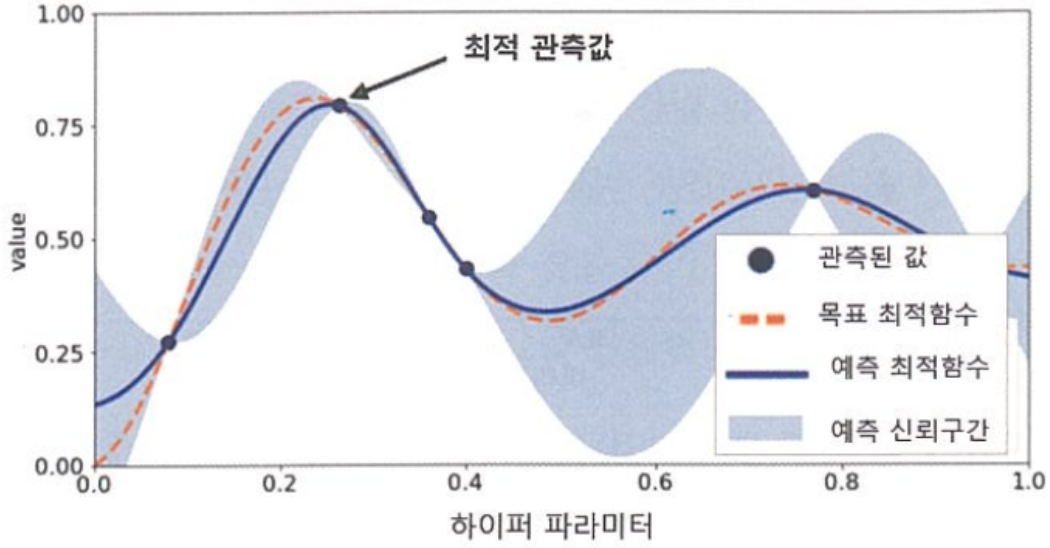



Step3) 추정 최적 함수를 기반, 획득 함수가 다음 관측 하이퍼 파라미터값 계산

획득 함수는 이전 하이퍼파라미터 최대 값보다 큰 값을 가질 가능성이 높은 지점 (최적 함수에서 가장 큰 값)의 값을(하이퍼파라미터) 대체 모델에 전달




Step4) 획득 함수로 받은 하이퍼파라미터를 기반으로 대체 모델 갱신, 최적 함수 예측 추정




Step3, Step4를 반복하면 대체 모델의 불확실성이 개선, 점차 정확한 최적 함수 추정이 가능함

## HyperOpt 사용하기

베이지안 최적화를 ML 모델의 하이퍼 파라미터 튜닝에 적용할 수 있게 제공되는 파이썬 패키지 중 하나

**활용 주요 로직**

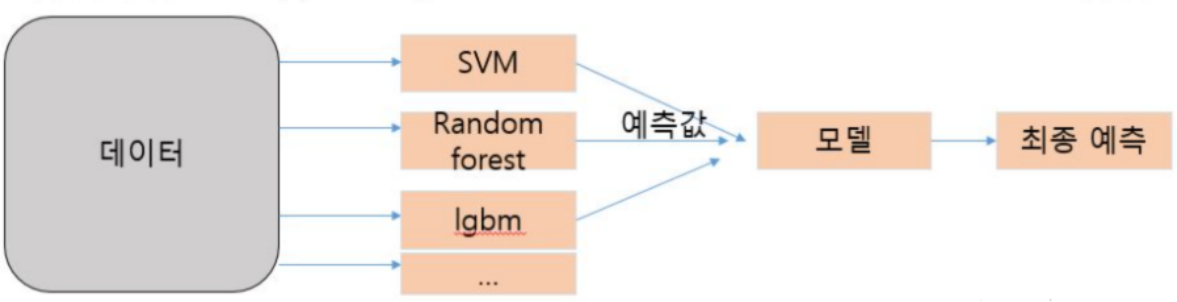
1. 입력 변수명, 입력값의 검색 공간 (Search Space) 설정

2. 목적 함수 (Objective Function) 설정

3. 목적 함수 반환 최솟값을 가지는 최적 입력값 유추

⏩HyperOpt는 목적 함수의 최솟값을 가지게하는 최적 입력값을 유추함 (최댓값 X)

# 10. 스태킹 앙상블




- 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출한다는 점에서 배깅(Bagging) 및 부스팅(Boosting)과 공통점을 가진다.
- 하지만 가장 큰 차이점은 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행한다는 것이다.
- 즉, 개별 알고리즘의 예측 결과 데이터 세트를 최종적인 메타 데이터 세트로 만들어 별도의 ML 알고리즘으로 최종 학습을 수행하고 테스트 데이터를 기반으로 다시 최종 예측을 수행하는 방식이다. (개별 모델의 예측된 데이터 세트를 다시 기반으로 하여 학습하고 예측하는 방식을 메타 모델이라고 한다.)
- 스태킹 모델은 두 종류의 모델이 필요하다.
    - 개별적인 기반 모델

    - 최종 메타 모델 (개별 기반 모델의 예측 데이터를 학습 데이터로 만들어서 학습)
- 스태킹 모델의 핵심은 여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습용 피처 데이터 세트와 테스트용 피처 데이터 세트를 만드는 것이다.
- 현실 모델에 적용하는 경우는 그렇게 많지 않으며 캐글과 같은 대회에서 조금이라도 성능 수치를 높여야 할 경우 자주 사용된다.
    - 2~3개의 개별 모델만을 결합해서는 쉽게 예측 성능을 향상시킬 수 없으며 많은 개별 모델이 필요하고, 반드시 성능 향상이 되리라는 보장도 없다.
    - 일반적으로 성능이 비슷한 모델을 결합해 좀 더 나은 성능 향상을 도출하기 위해 적용한다.