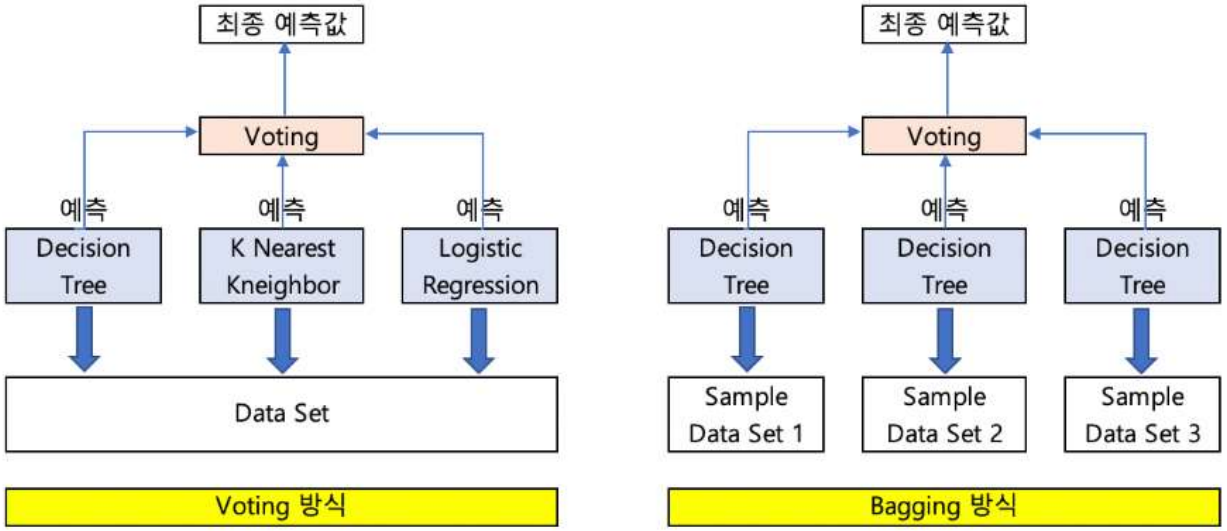
Tree와 랜덤 포레스트

➢ Ensemble Learning

- 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법
- 여러 개의 기본 모델 (weak learner, classifier, base learner, single learner)을 활용하여 하나의 새로운 모델을 만들어 내는 개념
- 다양한 learner를 만들면 전체의 모델의 성능이 증가 할 수 있음
- 다양한 분류기의 예측 결과를 결합함으로써 단일 분류기보다 신뢰성이 높은 예측값을 얻을 수 있습니다.
- 정형 데이터 분류 시에는 앙상블이 뛰어난 성능을 보임.

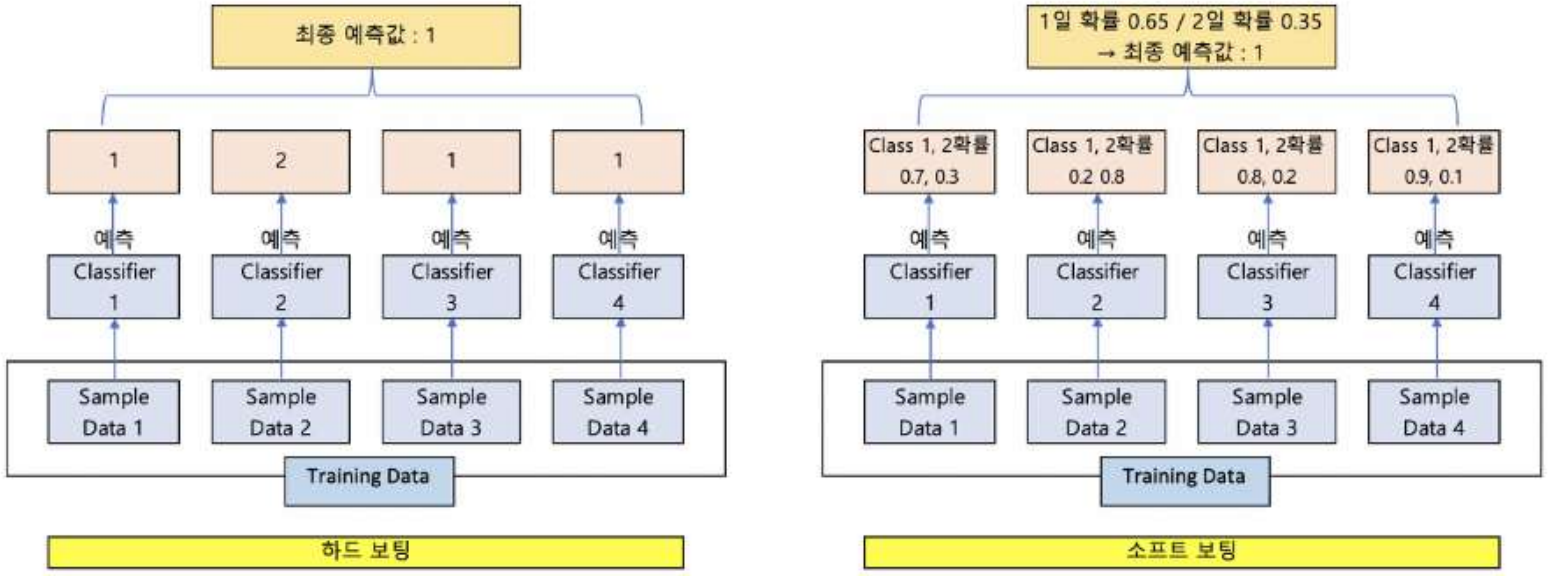


Tree와 랜덤 포레스트

- Ensemble Learning
  - 앙상블 학습의 유형 - 보팅(Voting), 배깅(Bagging), 부스팅(Boosting), 스태킹(Stacking)
  - 보팅 - 여러 종류의 알고리즘을 사용한 각각의 결과에 대해 투표를 통해 최종 결과를 예측하는 방식
  - 부트스트래핑(Bootstrapping) - 개별 Classifier에게 데이터를 샘플링해서 추출하는 방식
  - 배깅 앙상블 방식 – 개별 분류기가 부트 스트래핑 방식으로 샘플링된 데이터 세트에 대해서 학습을 통해 개별적인 예측을 수행한 결과를 보팅을 통해서 최종 예측 결과를 선정하는 방식
    같은 알고리즘에 대해 데이터 샘플을 다르게 두고 학습을 수행해 보팅을 수행하는 방식
    배깅의 대표적인 방식 : Random Forest
  - 부스팅 - 여러 개의 알고리즘이 순차적으로 학습을 하되, 앞에 학습한 알고리즘 예측이 틀린 데이터에 대해 올바르게 예측할 수 있도록, 그 다음번 알고리즘에 가중치를 부여하여 학습과 예측을 진행하는 방식
    그래디언트 부스트, XGBoost(eXtra Gradient Boost), LightGBM(Light Gradient Boost)
  - 스태킹 - 여러 가지 다른 모델의 예측 결과값을 다시 학습 데이터로 만들어 다른 모델(메타 모델)로 재학습시켜 결과를 예측하는 방법

Tree와 랜덤 포레스트

- 하드보팅(Hard Voting)과 소프트보팅(Soft Voting)
  - 하드보팅 - 다수결 원칙과 비슷 (다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정)
  - 소프트 보팅 - 각 알고리즘이 레이블 값 결정 확률을 예측해서, 이것을 평균하여 이들 중 확률이 가장 높은 레이블 값을 최종 값으로 예측



Tree와 랜덤 포레스트

- 보팅 분류기(Voting Classifier)
  - VotingClassifier 클래스의 생성자 매개변수 estimators는 리스트 값으로 보팅에 사용될 여러 개의 Classifier객체들을 튜플 형식으로 입력
  - VotingClassifier 클래스의 생성자 매개변수 voting은 보팅 방식 - 'hard', 'soft'

```
# 로지스틱회귀와 KNN을 기반으로 하여 소프트 보팅 방식으로 보팅 분류기 실습


#보팅 분류기의 정확도 출력
# LogisticRegression 정확도 출력
# KNeighborsClassifier 정확도 출력
```

Tree와 랜덤 포레스트

- 랜덤포레스트(RandomForest)
  - 배깅(Bagging)은 Bootstrap Aggregating의 약자로, 보팅(Voting)과는 달리 동일한 알고리즘으로 여러 분류기를 만들어 보팅으로 최종 결정하는 알고리즘
  - 배깅 진행 방식 :

(1) 동일한 알고리즘을 사용하는 일정 수의 분류기 생성

(2)각각의 분류기는 부트스트래핑(Bootstrapping)방식으로 생성된 샘플데이터를 학습

(3)최종적으로 모든 분류기가 보팅을 통헤 예측 결정

※ 부트스트래핑 샘플링은 전체 데이터에서 일부 데이터의 중첩을 허용하는 방식

Tree와 랜덤 포레스트

➢ 랜덤포레스트(RandomForest)

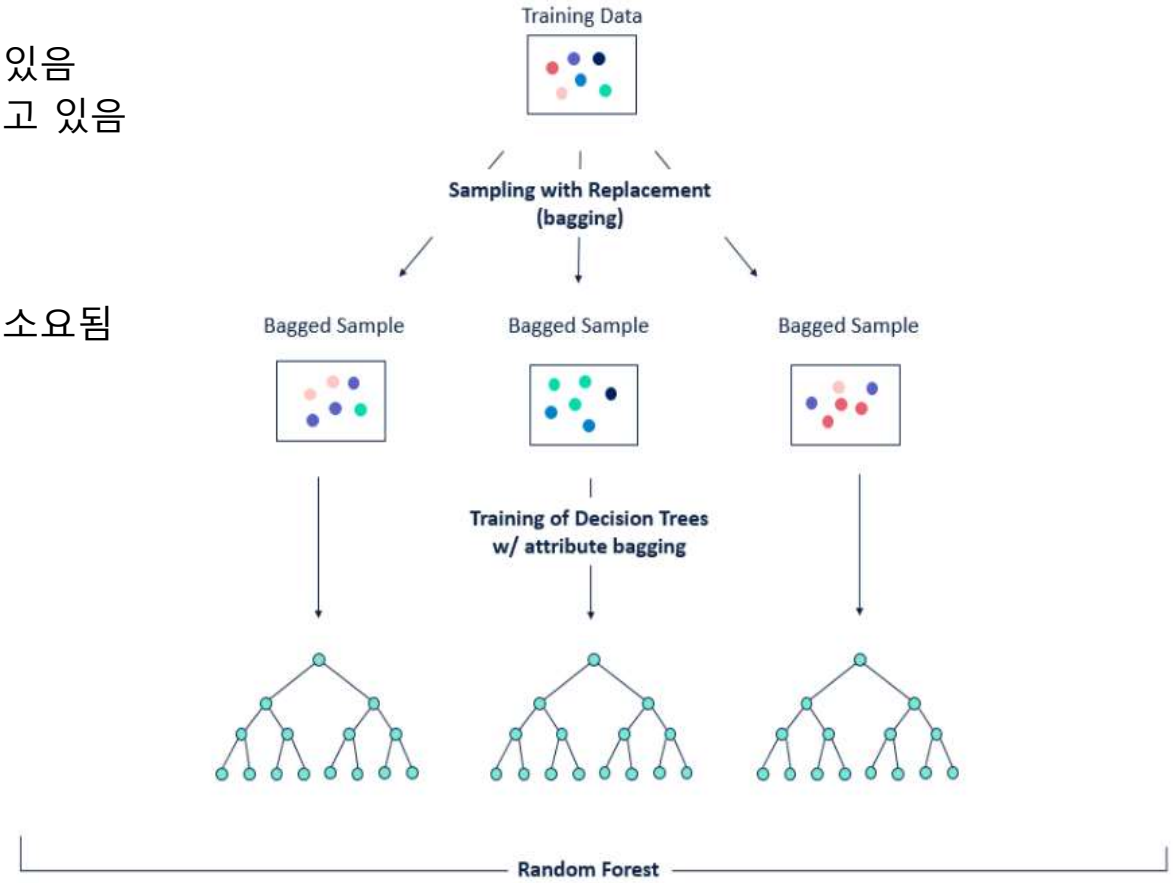
- 여러 개의 결정트리(Decision Tree)를 활용한 배깅 방식의 대표적인 알고리즘

- 장점

결정 트리의 쉽고 직관적인 장점을 그대로 가지고 있음
앙상블 알고리즘 중 비교적 빠른 수행 속도를 가지고 있음
다양한 분야에서 좋은 성능을 나타냄

- 단점

하이퍼 파라미터가 많아 튜닝을 위한 시간이 많이 소요됨



Tree와 랜덤 포레스트

- 랜덤포레스트(RandomForest) 하이퍼 파라미터 튜닝
  - 랜덤포레스트는 트리기반의 하이퍼 파라미터에 배깅, 부스팅, 학습, 정규화 등을 위한 하이퍼 파라미터까지 추가됩니다.

n_estimators	결정트리의 갯수를 지정 , Default(10), 트리 갯수를 늘리면 좋은 성능을 기대할 수 있지만, 학습 수행 시간이 오래 걸릴 수 있음
min_samples_split	노드를 분할하기 위한 최소한의 샘플 데이터수 , Default = 2 과적합을 제어하는데 사용, 작게 설정할 수록 분할 노드가 많아져 과적합 가능성 증가
min_samples_leaf	리프노드가 되기 위해 필요한 최소한의 샘플 데이터수, 과적합 제어 용도 불균형 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 작게 설정 필요
max_features	최적의 분할을 위해 고려할 최대 feature 개수 Default = 'auto' (결정트리에서는 default가 none이었음) int형으로 지정 →피처 갯수 / float형으로 지정 →비중 sqrt 또는 auto : 전체 피처 중 √(피처개수) 만큼 선정 log : 전체 피처 중 log2(전체 피처 개수) 만큼 선정
max_depth	트리의 최대 깊이 (default = None) 완벽하게 클래스 값이 결정될 때 까지 분할 또는 데이터 개수가 min_samples_split보다 작아질 때까지 분할 깊이가 깊어지면 과적합될 수 있으므로 적절히 제어 필요
max_leaf_nodes	리프노드의 최대 개수

Tree와 랜덤 포레스트

- Boosting Algorithm
  - 부스팅 알고리즘은 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치를 부여해 오류를 개선해나가며 학습하는 방식

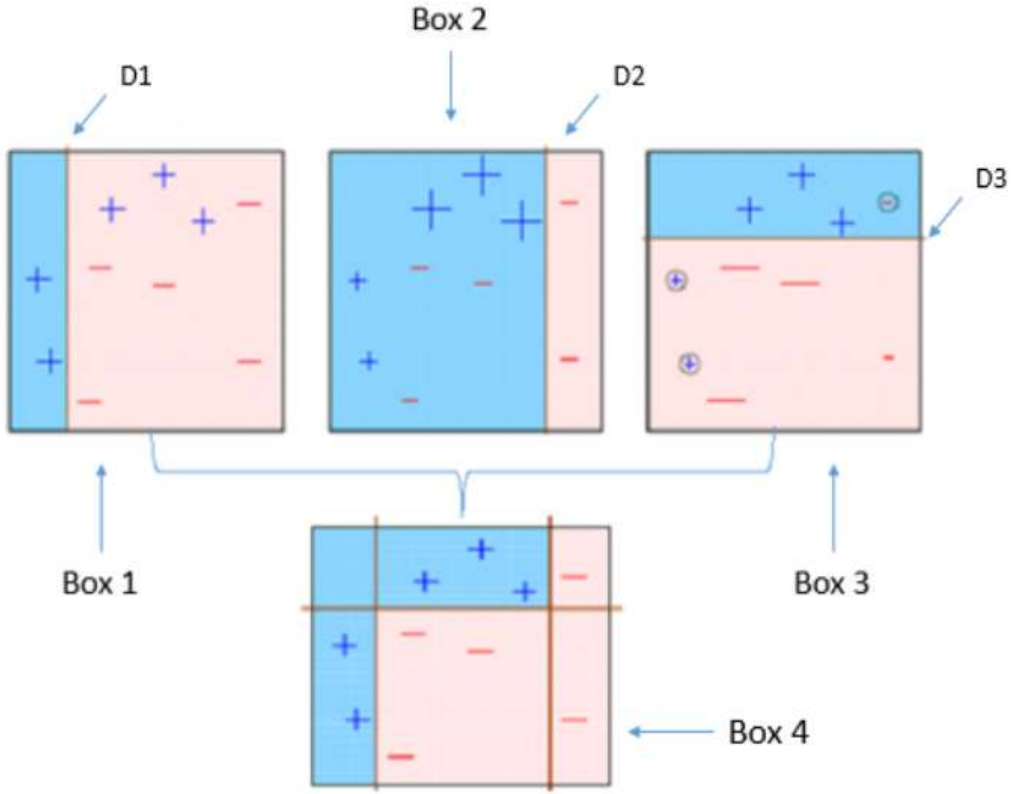
  - 부스팅 알고리즘
    AdaBoost
    Gradient Booting Machine(GBM)
    XGBoost
    LightGBM
    CatBoost

Tree와 랜덤 포레스트

- AdaBoost
  - Adaptive Boost의 줄임말로서 약한 학습기(weak learner)의 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 대표적인 알고리즘
  - 속도나 성능적인 측면에서 decision tree를 약한 학습기로 사용함

Step 1) 첫 번째 약한 학습기가 첫번째 분류기준(D1)으로 + 와 - 를 분류
Step 2) 잘못 분류된 데이터에 대해 가중치를 부여(두 번째 그림에서 커진 + 표시)
Step 3) 두 번째 약한 학습기가 두번째 분류기준(D2)으로 +와 - 를 다시 분류
Step 4) 잘못 분류된 데이터에 대해 가중치를 부여(세 번째 그림에서 커진 - 표시)
Step 5) 세 번째 약한 학습기가 세번째 분류기준으로(D3) +와 -를 다시 분류해서 오류 데이터를 찾음
Step 6) 마지막으로 분류기들을 결합하여 최종 예측 수행

→ 약한 학습기를 순차적으로 학습시켜, 개별 학습기에 가중치를 부여하여 모두 결합함으로써 개별 약한 학습기보다 높은 정확도의 예측 결과를 만듦



Tree와 랜덤 포레스트

- AdaBoost 하이퍼 파라미터 튜닝
  - 랜덤포레스트는 트리기반의 하이퍼 파라미터에 배깅, 부스팅, 학습, 정규화 등을 위한 하이퍼 파라미터까지 추가됩니다.

base_estimators	학습에 사용하는 알고리즘 Default = None DecisionTreeClassifier(max_depth=1)가 적용
n_estimators	생성할 약한 학습기의 갯수를 지정 Default = 50
learning_rate	학습을 진행할 때마다 적용하는 학습률(0~1) Weak learner가 순차적으로 오류 값을 보정해나갈 때 적용하는 계수 Default = 1.0

  - n_estimators를 늘린다면 생성하는 weak learner의 수는 늘어남
  - 이 여러 학습기들의 decision boundary가 많아지면서 모델이 복잡해짐

  - learning_rate을 줄인다면 가중치 갱신의 변동폭이 감소해서, 여러 학습기들의 decision boundary 차이가 줄어듦

  - n_estimators(또는 learning_rate)를 늘리고, learning_rate(또는 n_estimators)을 줄인다면 서로 효과가 상쇄된다.
  - 때문에 이 두 파라미터를 잘 조정하는 것이 알고리즘의 핵심입니다.

Tree와 랜덤 포레스트

- Gradient Boost Machine(GBM)
  - AdaBoost와 유사하지만, 가중치 업데이트를 경사하강법(Gradient Descent)를 이용하여 최적화된 결과를 얻는 알고리즘입니다.
  - GBM은 예측 성능이 높지만 Greedy Algorithm으로 과적합이 빠르게되고, 시간이 오래 걸린다는 단점이 있습니다.

※ Greedy Algorithm(탐욕 알고리즘)
미래를 생각하지 않고 각 단계에서 가장 최선의 선택을 하는 기법으로 각 단계에서 최선의 선택을 한 것이 전체적으로도 최선이길 바라는 알고리즘입니다.
물론 모든 경우에서 그리디 알고리즘이 통하지는 않습니다.
가령 지금 선택하면 1개의 마시멜로를 받고, 1분 기다렸다 선택하면 2개의 마시멜로를 받는 문제에서는, 그리디 알고리즘을 사용하면 항상 마시멜로를 1개밖에 받지 못합니다.
지금 당장 최선의 선택은 마시멜로 1개를 받는 거지만, 결과적으로는 1분 기다렸다가 2개 받는 게 최선이기 때문입니다.

Tree와 랜덤 포레스트

- Gradient Boost Machine(GBM) 하이퍼 파라미터 튜닝
  - Tree에 관한 하이퍼 파라미터

max_depth	트리의 최대 깊이 (default = 3) 깊이가 깊어지면 과적합될 수 있으므로 적절히 제어 필요
min_samples_split	노드를 분할하기 위한 최소한의 샘플 데이터수, 과적합을 제어하는데 사용 (Default = 2) 작게 설정할 수록 분할 노드가 많아져 과적합 가능성 증가
min_samples_leaf	리프노드가 되기 위해 필요한 최소한의 샘플 데이터수 (default = 1) min_samples_split과 함께 과적합 제어 용도 불균형 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 작게 설정 필요
max_features	최적의 분할을 위해 고려할 최대 feature 개수 Default = 'none' → 모든 피처 사용 int형으로 지정 →피처 갯수 / float형으로 지정 →비중 sqrt 또는 auto : 전체 피처 중 √(피처개수) 만큼 선정 log : 전체 피처 중 log2(전체 피처 개수) 만큼 선정
max_leaf_nodes	리프노드의 최대 개수 default = None → 제한없음

Tree와 랜덤 포레스트

- Gradient Boost Machine(GBM) 하이퍼 파라미터 튜닝
  - Boosting에 관한 하이퍼파라미터

loss	경사하강법에서 사용할 cost function 지정 특별한 이유가 없으면 default 값인 deviance 적용
n_estimators	생성할 트리의 갯수를 지정 (Default = 100) 많을수록 성능은 좋아지지만 시간이 오래 걸림
learning_rate	학습을 진행할 때마다 적용하는 학습률(0~1) Weak learner가 순차적으로 오류 값을 보정해나갈 때 적용하는 계수 (Default = 0.1) 낮은 만큼 최소 오류 값을 찾아 예측성능이 높아질 수 있음 하지만 많은 수의 트리가 필요하고 시간이 많이 소요
subsample	개별 트리가 학습에 사용하는 데이터 샘플링 비율(0~1) default=1 (전체 데이터 학습) 이 값을 조절하여 트리 간의 상관도를 줄일 수 있음

Tree와 랜덤 포레스트

- XGBoost(eXtra Gradient Boost)
  - 트리 기반의 알고리즘의 앙상블 학습에서 각광받는 알고리즘
  - GBM에 기반하고 있지만, GBM의 단점인 느린 수행시간, 과적합 규제 등을 해결한 알고리즘

  - XGBoost의 주요장점

(1) 뛰어난 예측 성능
(2) GBM 대비 빠른 수행 시간
(3) 과적합 규제(Overfitting Regularization)
(4) Tree pruning(트리 가지치기) : 긍정 이득이 없는 분할을 가지치기해서 분할 수를 줄임
(5) 자체 내장된 교차 검증

반복 수행시마다 내부적으로 교차검증을 수행해 최적회된 반복 수행횟수를 가질 수 있음
지정된 반복횟수가 아니라 교차검증을 통해 평가 데이트세트의 평가 값이 최적화되면 반복을 중간에 멈출 수 있는 기능이 있음
(6) 결손값 자체 처리

XGBoost는 독자적인 XGBoost 모듈과 사이킷런 프레임워크 기반의 모듈이 존재합니다.
독자적인 모듈은 고유의 API와 하이퍼파라미터를 사용하지만, 사이킷런 기반 모듈에서는 다른 Estimator와 동일한 사용법을 가지고 있습니다.

Tree와 랜덤 포레스트

- XGBoost 하이퍼 파라미터 튜닝
  - Boosting에 관한 하이퍼파라미터
  - 실행 시 스레드의 개수나 silent 모드 등의 선택을 위한 파라미터

booster	gbtree(tree based model) 또는 gblinear(linear model) 중 선택 Default = 'gbtree'
silent	Default = 1 출력 메시지를 나타내고 싶지 않을 경우 1로 설정
nthread	CPU 실행 스레드 개수 조정 Default는 전체 다 사용하는 것 멀티코어/스레드 CPU 시스템에서 일부CPU만 사용할 때 변경

Tree와 랜덤 포레스트

- XGBoost 하이퍼 파라미터 튜닝
  - 주요 부스터 파라미터
  - 트리 최적화, 부스팅, regularization 등과 관련된 파라미터를 지칭

파라미터명 (파이썬 래퍼)	파라미터명 (사이킷런 래퍼)	설명
eta	learning rate	GBM의 learning rate와 같은 파라미터 , 범위: 0 ~ 1
num_boost_around	n_estimators	생성할 weak learner의 수
min_child_weight	min_child_weight	GBM의 min_samples_leaf와 유사 관측치에 대한 가중치 합의 최소를 말하지만 GBM에서는 관측치 수에 대한 최소를 의미
gamma	min_split_loss	리프노드의 추가분할을 결정할 최소손실 감소값 해당값보다 손실이 크게 감소할 때 분리 값이 클수록 과적합 감소효과
max_depth	max_depth	트리 기반 알고리즘의 max_depth와 동일 0을 지정하면 깊이의 제한이 없음 너무 크면 과적합(통상 3~10정도 적용)
sub_sample	subsample	GBM의 subsample과 동일 데이터 샘플링 비율 지정(과적합 제어) 일반적으로 0.5~1 사이의 값을 사용

Tree와 랜덤 포레스트

- XGBoost 하이퍼 파라미터 튜닝
  - 주요 부스터 파라미터
  - 트리 최적화, 부스팅, regularization 등과 관련된 파라미터를 지칭

파라미터명 (파이썬 래퍼)	파라미터명 (사이킷런 래퍼)	설명
colsample_bytree	colsample_bytree	GBM의 max_features와 유사 트리 생성에 필요한 피처의 샘플링에 사용 피처가 많을 때 과적합 조절에 사용
lambda	reg_lambda	L2 Regularization 적용 값 피처 개수가 많을 때 적용을 검토 클수록 과적합 감소 효과
alpha	reg_alpha	L1 Regularization 적용 값 피처 개수가 많을 때 적용을 검토 클수록 과적합 감소 효과
scale_pos_weight	scale_pos_weight	불균형 데이터셋의 균형을 유지

Tree와 랜덤 포레스트

- XGBoost 하이퍼 파라미터 튜닝
  - 학습 태스크 파라미터
  - 학습 수행 시의 객체함수, 평가를 위한 지표 등을 설정하는 파라미터

파라미터명	파라미터명 (사이킷런 래퍼)
objective	reg:linear' : 회귀 binary:logistic : 이진분류 multi:softmax : 다중분류, 클래스 반환 multi:softprob : 다중분류, 확률반환
eval_metric	검증에 사용되는 함수정의 회귀 분석인 경우 'rmse'를, 클래스 분류 문제인 경우 'error' rmse : Root Mean Squared Error mae : mean absolute error logloss : Negative log-likelihood error : binary classification error rate merror : multiclass classification error rate mlogloss: Multiclass logloss auc: Area Under Curve

Tree와 랜덤 포레스트

- XGBoost 과적합 제어
  - eta 값을 낮춥니다.(0.01 ~ 0.1) → eta 값을 낮추면 num_boost_round(n_estimator)를 반대로 높여주어야 합니다.
  - max_depth 값을 낮춥니다.
  - min_child_weight 값을 높입니다.
  - gamma 값을 높입니다.
  - subsample과 colsample_bytree를 낮춥니다.

※ Early Stopping 기능
GBM의 경우 n_estimators에 지정된 횟수만큼 학습을 끝까지 수행하지만, XGB의 경우 오류가 더 이상 개선되지 않으면 수행을 중지
n_estimators 를 200으로 설정하고, 조기 중단 파라미터 값을 50으로 설정하면, 1부터 200회까지 부스팅을 반복하다가
50회를 반복하는 동안 학습오류가 감소하지 않으면 더 이상 부스팅을 진행하지 않고 종료합니다.
(가령 100회에서 학습오류 값이 0.8인데 101~150회 반복하는 동안 예측 오류가 0.8보다 작은 값이 하나도 없으면 부스팅을 종료)

Tree와 랜덤 포레스트

- LightGBM
  - XGBoost 대비 더 빠른 학습과 예측 수행 시간
  - 더 작은 메무리 사용량
  - 카테고리형 피처의 자동 변환과 최적 분할
  - 원-핫인코딩 등을 사용하지 않고도 카테고리형 피처를 최적으로 변환하고 이에 따른 노드분할 수행
  - 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉽습니다. (공식 문서상 대략 10,000건 이하의 데이터 세트)

  - GBM과의 차이점

  일반적인 균형트리분할 (Level Wise) 방식과 달리 리프중심 트리분할(Leaf Wise) 방식을 사용합니다.

  균형트리분할은 최대한 균형 잡힌 트리를 유지하며 분할하여 트리의 깊이를 최소화하여
  오버피팅에 강한구조이지만 균형을 맞추기 위한 시간이 필요합니다.
  리프중심 트리분할의 경우 최대 손실 값을 가지는 리프노드를 지속적으로 분할하면서
  트리가 깊어지고 비대칭적으로 생성합니다. 이로써 예측 오류 손실을 최소화하고자 합니다.

Tree와 랜덤 포레스트

- LightGBM 하이퍼 파라미터 튜닝
  - num_leaves의 개수를 중심으로 min_child_sampes(min_data_in_leaf), max_depth를
  - 함께 조절하면서 모델의 복잡도를 줄이는 것이 기본 튜닝 방안입니다.

  - num_leaves를 늘리면 정확도가 높아지지만 트리가 깊어지고 과접합되기 쉬움
  - min_child_samples(min_data_in_leaf)를 크게 설정하면 트리가 깊어지는 것을 방지
  - max_depth는 명시적으로 깊이를 제한. 위의 두 파라미터와 함께 과적합을 개선하는데 사용
  - 또한, learning_rate을 줄이면서 n_estimator를 크게하는 것은 부스팅에서의 기본적인 튜닝 방안