# 4.5

# GBM의 개요 및 실습
부스팅 알고리즘 : 여러 개의 약한 학습기를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식.
ex. AdaBoost, 그래디언트 부스트.

에이다 부스트 : 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 알고리즘.

GBM : 에이다부스트와 유사하나, 가중치 업데이트를 경사 하강법을 이용함.
(오류값 = 실제값 - 예측값)
(경사 하강법) : 오류식 h(x) = y - F(x)를 최소화 하는 방향성을 가지고 반복적으로 가중치 값을 업데이트 하는 것.

일반적으로 GBM이 랜덤 포레스트보다 예측 성능이 뛰어난 경우가 많음. 그러나 수행 시간이 오래 걸리고 하이퍼 파라미터 튜닝 노력도 더 필요함....

[GBM의 하이퍼 파라미터]
n_estimators, max_depth, max_features
loss : 경사 하강법에서 사용할 비용 함수를 지정함. 디폴트 : deviance
learning_rate : GBM이 학습을 진행할 때마다 적용하는 학습률. Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수임. 0~1로 지정할 수 있으며 디폴트는 0.1.
너무 작은 값을 적용하면 업데이트 되는 값이 작아져서 최소 오류 값을 찾아 예측 성능이 높아질 가능성이 높음. 하지만 수행 시간이 오래 걸리고, 값을 찾지 못할 수도 있음. 반대로 너무 큰 값을 적용하면 최소 오류 값을 찾지 못하고 지나쳐 예측 성능이 떨어질 수도 있음.
(learning_rate은 n_estimators와 사호 보완적으로 조합해 사용함.)
n_estimators : weak learner의 개수. 예측 성능이 많을수록 일정성능까지는 좋아질 수 있지만 수행 시간이 오래 걸림. 디폴트 : 100
subsample : weak learner가 학습에 사용하는 데이터의 샘플링 비율. 기본값은 1이며 전체 학습 데이터를 기반으로 학습한다는 의미임. 과저갑이 염려되는 경우 subsam;e을 1보다 작은 값으로 설정함.

# 4.6[XGBoost]

압도적인 수치의 차이는 아니지만, 분류에 있어서 일반적으로 다른 머신러닝보다 뛰어난 예측 성능을 나타냄. GBM에 기반하고 있지만, GBM의 단점인 느린 수행 시간 및 과적합 규제 부재 등의 문제를 해결해서 매우 각공을 받음.특히 병렬 CPU 환경에서 병렬 학습이 가능해 기존 GBM보다 빠르게 학습을 완료할 수 있음.
장점 : 뛰어난 예측 성능, GBM 대비 빠른 수행 시간, 과적합 규제, Tree pruning, 자체 내장된 교차 검증, 결손값 자체 처리.

하이퍼 파라미터
일반 파라미터 : 일반적으로 실행 시 스레드의 개수나 silent 모드 등의 선택을 위한 파라미터로서 디폴트 파라미터 값을 바꾸는 경우는 거의 없음.
부스터 파라미터 : 트리 최적화, 부스팅, regularization 등과 관련 파라미터 등을 지칭함
학습 태스크 파라미터 : 학습 수행 시의 객체 함수, 평가를 위한 지표 등을 설정하는 파라미터임.

[주요 일반 파라미터]
booster : gbtree 또는 gblinear 선택, 디폴트는 gbtree임.
silent : 디폴트는 0이며, 출력 메시지를 나타내고 싶지 않을 경우 1로 설정.
nthread : 체ㅓ의 실행 스레드 개수를 조정하며, 디폴트는 CPU 전체 스레드를 다 사용하는 거임.

[주요 부스터 파라미터]
eta[default=0.3,ailas:learning_rate] : GBM의 학습률과 같은 파라미터.파이썬 래퍼 기반의 xgboost를 이용할 경우 디폴트는 0.3, 사이킷런 래퍼 클래스를 이용할 경우 eta는 learning_rate 파라미터로 대체됨. 디폴트는 9,1
num_boost_rounds : GBM의 n_estimators와 같은 파라미터
min_child_seight[default = 1] : 트리에서 추가적으로 가지를 나눌지를 결정하기 위해 필요한 데이터들의 weight 총합. 클수록 분할을 자제함.
gamma[default=0, alias : min_split_loss] : 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 갑소 값. 해당 값보다 큰 손실이 감소된 경우에 리프 노드를 분리함. 값이 클수록 과적합 감소 효과가 있음.
max_depth[default=6] : 트리 기반 알고리즘의 max_depth와 같음. 0을 지정하면 깊이에 제한이 없음.
sub_sample[default=1] : GBM의 subsample과 동일함.
colsample_bytree[default=1] : GBM의 max_features와 유사함. 트리 생성에 필요한 피처를 임의로 샘플링하는데 사용됨.
lambda[default=1,alias:reg_lambda] : L2 Regularization 적용 값임. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있음.
alpha[default=0,alias:reg_alpha] : L1 Regularization 적용 값임. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있음.
scale_pos_weight[default=1] : 특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트를 균형을 유지하기 위한 파라미터.

[학습 태스크 파라미터]
objective: 최솟값을 가져야 할 손실 함수를 정의함. 주로 사용되는 손실함수는 이진 분류인지, 다중 분류인지에 따라 달라짐.
binary:logistic: 이진 분류일 때 적용.
multi:softmax: 다중 분류일 때 적용. 손실 함수가 multi:softmax일 경우에는 레이블 클래스의 개수인 num_class 파라미터를 지정해야 함.

multi:softprob :multi:softmax와 유사하나 개별 레이블 클래스의 해당되는 예측 확률을 반환함.
eval_metric : 검증에 상요되는 함수를 저용함. 회귀인 경우는 rmse, 분류일 경우에는 error.

사이킷런 래퍼 XGBoost 하이퍼 파라미터
eta -> learning_rate
sub_sample -> subsample
lambda -> reg_lambda
alpha -> reg_alpha

## 4.7[LightGBM]

XGBoost보다 학습에 걸리는 시간이 훨씬 적음.
메모리 사용량도 상대적으로 적음.
그러나 XGBoost의 예측 성능과 별다른 차이가 없음. 또한 기능상의 다양성은 LightGBM이 약간 더 많음.
(단점) : 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉬움.

LightGBM은 일반 GBM 계역의 트리 분할 방버과 다르게 리프 중심 트리 분할 방식을 사용함. (리프 중심 트리 분할 방식은 트리의 균형을 맞추지 않고, 최대 손실 값을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 ㅂ대칭적인 규칙 트리가 생성됨. 하지만, 이렇게 최대 손실값을 가지는 리프 노드를 지속적으로 분할해 생성된 규칙 트리는 학습을 반복할수록 결국은 균형 트리 분할 방식보다 예측 오류 손실을 최소화 할 수 있음.)

[주요 파라미터]
(p.247 ~ p.248 참고)

learning_rate을 작게 하면서 n_estimators를 크게 하는 것은 부스팅 계열 튜닝에서 가장 기본적인 튜닝 방안임.
이 밖에 과적합을 제어하기 위해서 reg_lambda, reg_alpha와 같은 regularization을 적용하거나 학습 데이터에 사용할 피처의 개수나 데이터 샘플링 레코드 개수를 줄이기 위해 colsample_bytree, subsample 파라미터를 적용할 수 있음.

*LightGBM도 XGBoost와 동일하게 조기 중단이 가능함.

## 4.11[스태킹 앙상블](개정판 4.10에 해당함)

스태킹은 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출한다는 점에서 앞에소개한 배깅 및 부스팅과 공통점을 가지고 있음. 하지만, 가장 큰 차이점은 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행한다는 거임!!

스태킹 모델은 두 종류의 모델이 필요함.

1. 개별적인 기반 모델
2. 개별 기반 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습용 피처 데이터 세트와 테스트용 피처 데이터 세트를 만드는 것.