지도학습: 레이블, 즉 명시적인 정답이 있는 데이터가 주어진 상태에서 학습하는 머신러닝

* 분류: 머신러닝 알고리즘으로 학습데이터 학습해 모델 생성하고 미지의 레이블 값 예측
* 분류를 구현하는 알고리즘

1. 나이브 베이즈 Naïve Bayes: 베이즈 통계과 생성모델에 기반
2. 로지스틱 회귀 Logistic Regression: 독립변수와 종속변수의 선형 관계성에 기반
3. 결정 트리 Decision Tree: 데이터 균일도에 따른 규칙 기반

🡪 쉽고 유연, 데이터 스케일링이나 정규화 등 사전 가공의 영향이 적음

🡪 but 예측 성능 향상시키기 위해 복잡한 규칙 구조를 가져서 과적합 발생해 예측 성능 저하될 수 있음

1. 서포트 벡터 머신 SVM: 개별 클래스 간 최대 분류 마진을 효과적으로 찾아줌
2. 최소 근접 알고리즘 Nearest Neighbor: 근접 거리 기준으로
3. 신경망 Neural Network: 심층 연결 기반
4. 앙상블 Ensemble: 서로 다른(또는 같은) 머신러닝 알고리즘 결합

🡪 정형 데이터의 예측 분석 영역에서는 앙상블이 매우 높은 예측 성능 가짐

🡪 배깅(Bagging) 방식: 랜덤 포레스트 but 근래는 부스팅 방식으로 발전

🡪 부스팅(Boosting) 방식: 그래디언트 부스팅 – XgBoost, LightGBM

🡪 기본 알고리즘으로는 일반적으로 결정 트리 사용, 많은 여러 개의 약한 학습기를 결합해 확률적 보완과 오류 부분에 대한 가중치를 계속 업데이트하면서 예측 성능 향상 – 결정 트리가 좋은 약한 학습기임

**결정 트리**

* 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙 만드는 것
* 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인지가 성능 좌우
* 새로운 규칙마다 서브 트리 생성됨, Leaf Node: 결정된 클래스 값, 규칙 노드: 규칙 조건이 됨
* 트리의 깊이가 깊어질수록 과적합으로 이어지기 쉬워 예측 성능 저하

🡪 가능한 적은 결정 노드로 높은 예측 정확도 가지려면 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 규칙 정해져야 함

🡪 최대한 균일한 데이터 세트 구성하도록 분할

🡪 정보 균일도가 높은 데이터 세트 먼저 선택하도록 규칙조건 만듦, 서브 데이터에서 또 균일도 높은 자식 데이터 세트 쪼개는 방식으로

정보의 균일도 특정하는 방법: **엔트로피를 이용한 정보 이득 지수** 과 **지니 계수**

1. 정보 이득 지수 = 1 – 엔트로피 지수

엔트로피: 주어진 데이터 집합의 혼잡도, 서로 다른 값이 섞여 있으면 엔트로피 높음

**정보 이득이 높은 속성을 기준으로 분할**

1. 지니 계수: **지니 계수 낮을수록 데이터 균일도가 높음**

결정 트리 알고리즘을 구현한 DecisionTreeClassifier: 기본은 지니 계수 이용해 데이터 세트 분할

**결정 트리 모델 특징**

균일도를 기반으로 해서 알고리즘이 쉽고 직관적, 룰이 명확해 규칙 노드와 리프 노드가 어떻게 생성되는지 알 수 있고 시각화로 표현까지 가능, 피처의 스케일링과 정규화 같은 전처리 필요 X

But 과적합으로 정확도 떨어짐 – 피처가 많고 균일도가 다양할수록 트리 깊이 커지고 복잡

🡪 모든 상황 만족하는 완벽한 규칙 만들 수 없음. 트리의 크기를 사전에 제한하는 것이 성능 튜닝에 도움

**결정 트리 파라미터**

결정 트리 알고리즘 구현한 클래스: DecisionTreeClassifier(분류) DecisionTreeRegressor(회귀)

사이킷런의 결정 트리 구현은 CART 알고리즘 기반

DecisionTreeClassifier의 파라미터

1. **min\_samples\_split**: 노드를 분할하기 위한 최소한의 샘플 데이터 수, 디폴트 2, 과적합 제어

즉 자식 노드로 분할하려면 최소한 샘플 개수가 몇 개는 필요한데, 그보다 적으면 분할 X, 그냥 리프 노드가 됨.

1. **min\_samples\_leaf**: leaf node가 되기 위한 최소한의 샘플 데이터 수, 디폴트 1, 과적합 제어

디폴트 1 의미는 단독 클래스로만 되어 있거나 단 한 개의 데이터로 돼 있을 경우에는 리프 노드

비대칭적 데이터의 경우 특정 클래스의 데이터가 작을 수 있으므로 이때는 작게 설정

이 값을 키우면 리프 노드가 될 수 있는 조건이 완화됨, min\_samples\_leaf<=지정 값 만족하면 리프 노드 됨

1. **max\_features**: 최적의 분할 위해 고려할 최대 피처 개수, 디폴트 None(모든 피처 사용)

int면 대상 피처 개수, float면 전체 피처 중 대상 피처의 퍼센트, sqrt면 루트(전체 피처 개수),

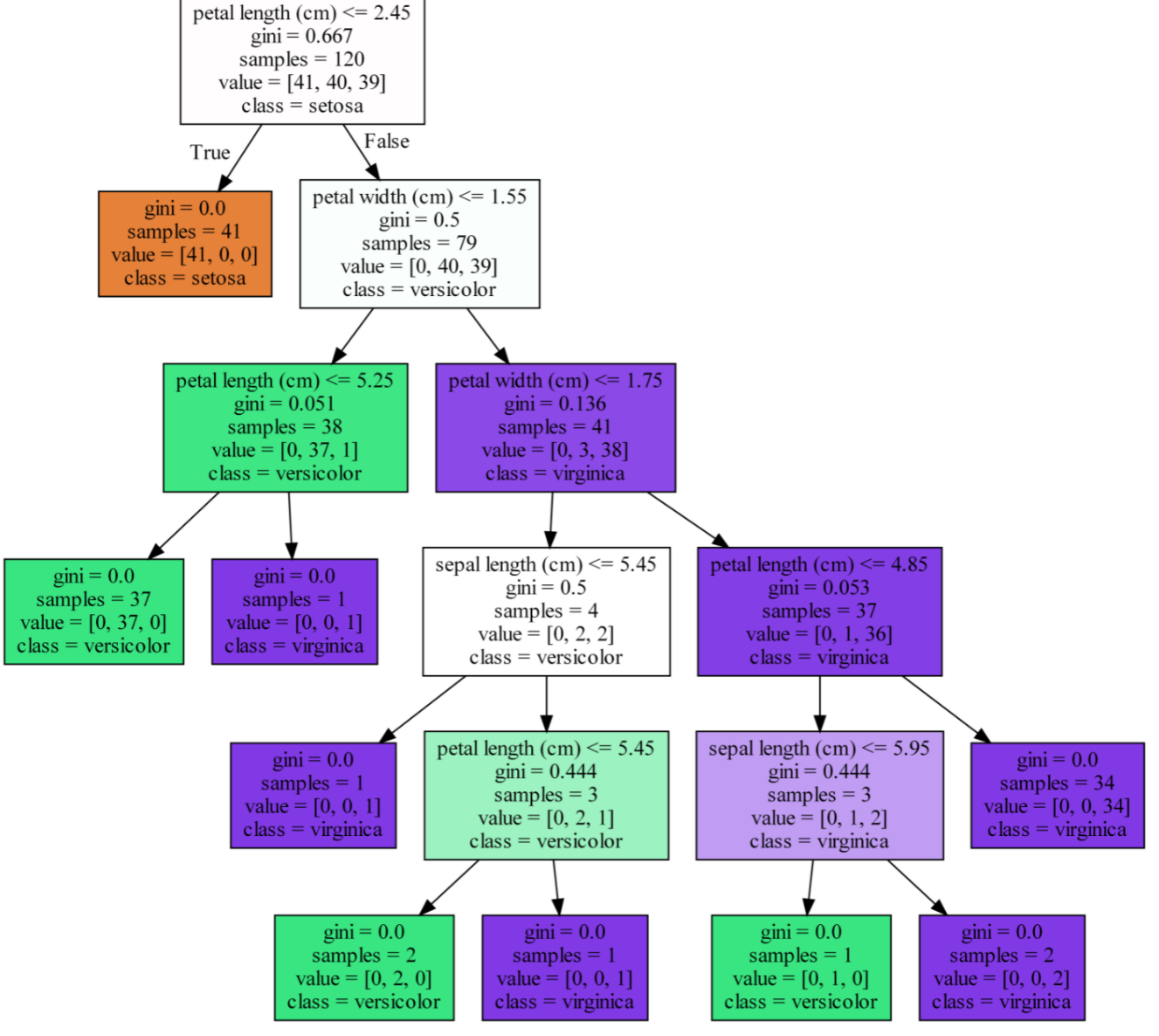
auto는 sqrt와 동일, log는 전체 피처 중 log2(전체 피처 개수) 선정

1. **max\_depth**: 최대 깊이, 디폴트 None,
2. **max\_leaf\_nodes**: leaf node의 최대 개수

**결정 트리 모델 시각화**

Graphviz - 결정 트리 알고리즘이 어떤 규칙을 가지고 트리 생성하는지 시각적으로 보여줌

export\_graphviz(학습 완료된 Estimator, 피처 이름 리스트, 레이블 이름 리스트) -> 학습된 결정 트리 규칙을 트리 형태로 시각화

리프 노드 – 자식 노드가 없는 노드, 최종 레이블 값이 결정되는 노드

브랜치 노드 – 자식 노드가 있는 노드, 자식 노드를 만들기 위한 분할 규칙 조건 가짐

Gini: value=[]으로 주어진 데이터 분포에서의 지니 계수

Samples는 현 규칙에 해당하는 데이터 건수, Class 는 하위 노드 가질 경우에 제일 많은 수의 레이블

Value = [] 는 클래스 값 기반의 데이터 건수 🡪 0: Setosa, 1: Versicolor, 2: Virginica 품종 가리킴

🡪 value=[41,40,39] 이면 0번 품종 41개, 1번 품종 40개, 2번 품종 39개라는 뜻

각 노드의 색깔은 붓꽃 데이터의 레이블 값 – 주황색 0: Setosa, 초록색1: Versicolor, 보라색2: Virginic 🡪 색깔이 짙어질수록 지니 계수 낮고 해당 레이블에 속하는 샘플 데이터가 많다.

규칙 생성 로직을 미리 제어하지 않으면 완벽하게 클래스 구별해내기 위해 트리 노드 계속 만듦 🡪 과적합 🡪 하이퍼 파라미터가 복잡한 트리 생성되는 것을 막음( max\_depth, min\_samples\_split, min\_samples\_leaf)

결정 트리는 균일도에 기반해 어떠한 속성을 규칙 조건으로 선택하느냐가 중요함

피처의 중요한 역할 지표를 DecisionTreeClassifier객체의 feature\_importances\_속성으로 제공

🡪 ndarray형태, 피처 순서대로 값 할당, 값이 높을수록 해당 피처의 중요도 높음

**앙상블 학습**

앙상블 학습을 통한 분류는 여러 개의 분류기를 생성하고 그 예측을 결합하여 보다 정확한 최종 예측 도출하는 기법

대부분의 정형 데이터 분류 시에 앙상블이 뛰어난 성능 🡪 랜덤 포레스트, 그래디언트 부스팅 🡪 뛰어난 성능, 쉬운 사용, 다양한 활용도 🡪 XGboost, LightGBM, Stacking, …

앙상블 학습 유형

1. 보팅(Voting): 여러 개의 분류기가 투표를 통해 최종 예측 결과 결정, 서로 다른 알고리즘을 가진 분류기 결합, 같은 데이터 세트에 대해 학습
2. 배깅(Bagging): 여러 개의 분류기가 투표를 통해 최종 예측 결과 결정, 각각 분류기가 모두 같은 유형의 알고리즘 기반 but 데이터 샘플링을 다르게 가져가서 학습 수행하고 보팅 수행 🡪 랜덤 포레스트

부트스트래핑 분할 방식: 개별 Classifier에게 데이터를 샘플링해 추출하는 방식(중첩 허용)

1. 부스팅(Boosting): 여러 분류기가 순차적으로 학습하되, 먼저 학습한 분류기가 예측이 틀린 데이터에 대해서는 올바르게 예측할 수 있도록 다음 분류기에게는 가중치 부여

🡪 그래디언트 부스팅, XGBoost, LightGBM

보팅 유형

1. 하드 보팅: 다수결 원칙과 비슷, 예측한 결과값 중 다수의 분류기가 결정한 예측값을 최종 보팅 결과값으로 선정
2. 소프트 보팅: 분류기의 레이블 값 결정 확률을 모두 더하고 평균해서 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 선정

**보팅 분류기**

보팅 방식의 앙상블 구현한 VotingClassifier

– 주요 생성 인자 estimators: 리스트값, 보팅에 사용된 여러 개의 Classifier 객체를 튜플 형식으로, voting: hard/soft

**랜덤 포레스트**

배깅(Bagging)의 대표적인 알고리즘

여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습 수행 후 보팅을 통해 예측 결정

학습 데이터 세트는 전체 데이터에서 일부 중첩되게 샘플링된 데이터 세트(부트스트래핑 분할 방식) 🡪 서브 세트의 데이터 건수는 전체 데이터 건수와 동일하지만 개별 데이터가 중첩됨 🡪 개별 데이터 세트에 결정 트리 분류기를 각각 적용: 랜덤 포레스트

**랜덤 포레스트 하이퍼 파라미터 및 튜닝**

트리 기반의 앙상블 알고리즘은 하이퍼 파라미터가 너무 많고, 그로 인해 튜닝 위한 시간이 많이 소모됨(배깅, 부스팅, 학습, 정규화 등을 위한 하이퍼 파라미터 추가됨)

1. n\_estimators: 결정 트리의 개수 지정, 디폴트 10
2. max\_features: 결정 트리의 max\_features와 동일, 분할을 위해 고려할 최대 피처 수, 디폴트 'auto(sqrt)'
3. max\_depth, min\_samples\_leaf, ...

**GBM(Gradient Boosting Machine)**

부스팅 알고리즘: 여러 개의 약한 학습기를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여하여 오류 개선

1. Adaptive boosting(AdaBoost): 오류 데이터에 가중치 부여하면서 부스팅 수행, 모두 결합해 예측 수행
2. 그래디언트 부스트: 가중치 업데이트에 경사 하강법(Gradient Descent) 이용 🡪 오류 값 (실제 값 – 예측 값) 최소화하는 방향성 가지고 반복적으로 가중치 업데이트

🡪 반복 수행을 통해 오류 최소화할 수 있도록 가중치의 업데이트 값 도출

분류와 회귀 가능, 분류를 위한GradientBoostingClassifier 클래스 제공

랜덤 포레스트보다 예측 성능 조금 뛰어나지만 오래 걸리고 하이퍼 파라미터 튜닝 노력 필요

**GBM 하이퍼 파라미터 및 튜닝**

n\_estimators, max\_depth, max\_features와 같은 트리 기반 자체의 파라미터

loss: 경사 하강법에서 사용할 비용 함수, 기본값 ‘deviance’

learning\_rate: GBM이 학습 진행할 때마다 적용하는 학습률, weak learner가 순차적으로 오류 값 보정해 나가는 데 적용하는 계수, 기본값 0.1

* 너무 작으면 최소 오류 값 찾아 예측 성능 높아질 가능성 있지만 오래걸리고 최소 오류 값 찾지 못할 수도 있음.
* 너무 크면 최소 오류 값 찾지 못하고 그냥 지나쳐서 예측 성능 떨어지지만 빠름
* n\_estimators와 상호 보완적으로 사용

n\_estimators: weak learner의 개수, 기본값 100

subsample: weak learner가 학습에 사용하는 데이터의 샘플링 비율, 기본값 1(전체 학습 데이터를 기반으로 학습) 🡪 0.5이면 학습 데이터의 50%

과적합에도 강한 뛰어난 예측 성능 가지지만 수행 시간 오래 걸림

**XGBoost(eXtra Gradient Boost)**

* GBM 기반
* 병렬 수행 가능하여 GBM 대비 빠른 수행 시간
* 과적합 규제 기능으로 과적합에 좀 더 강한 내구성 가짐
* Max\_depth로 분할 깊이 조정하기도 하지만, tree pruning(가지치기)로 더 이상 긍정 이득이 없는 분할을 가지지기 해서 분할 수를 줄임
* 자체 내장된 교차 검증, 평가 값이 최적화되면 조기 중단
* 결손값 자체 처리

초기에는 XGBoost 고유의 프레임워크를 파이썬 언어 기반에서 구현 – 사이킷런 적용 X

🡪 하지만 사이킷런 많이 사용되고 있어서 사이킷런과 연동할 수 있는 Wrapper class 제공 (XGBClassifier, XGBRegressor), 표준 사이킷런 개발 프로세스 및 다양한 유틸리티 활용 가능

**파이썬 래퍼 XGBoost 하이퍼 파라미터**

파이썬 래퍼 XGBoost 하이퍼 파라미터와 사이킷런 래퍼 XGBoost 하이퍼 파라미터는 동일한 기능을 의미하는 파라미터지만 사이킷런의 범용화된 이름 규칙에 따라 파라미터 명이 달라짐

1. 일반 파라미터: 디폴트 바꾸는 경우 거의 없음

* booster: gbtree(tree based model) or gblinear(linear model), 디폴트 gbtree
* silent: 디폴트 0, 출력 메세지 나타내고 싶지 않으면 1
* nthread: cpu의 실행 스레드 개수, 디폴트는 전체 다 사용

1. 부스터 파라미터: 트리 최적화, 부스팅 등 관련 – 대부분의 하이퍼 파라미터는 여기

* eta: gbm의 학습률과 같은 파라미터, 디폴트 0.3

부스팅 스텝을 반복적으로 수행할 때 업데이트 되는 학습률 값

* num\_boost\_rounds: GBM의 n\_estimators
* min\_child\_weight: 트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 weight 총합, 디폴트 1, 클수록 분할 자제
* gamma: 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값, 디폴트 0, gamma 보다 큰 손실이 감소된 경우 리프노드 분리, 클수록 과적합 감소 효과
* max\_depth: 디폴트 6
* sub\_sample: GBM의 subsample과 동일, 데이터를 샘플링하는 비율 지정, 디폴트 1
* colsample\_bytree: GBM의 max\_features와 유사, 디폴트 1
* lambda: L2 Regularization 적용 값, 디폴트 1
* alpha: L1 Regularization 적용 값, 디폴트 0
* scale\_pos\_weight: 특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형 유지하기 위한 파라미터, 디폴트 1

1. 학습 태스트 파라미터: 학습 수행 시 객체함수, 평가 위한 지표 등

* objective: 최솟값을 가져야할 손실 함수 정의
  + binary:logistic: 이진 분류일 때 적용
  + multi:softmax: 다중 분류일 때 적용 -> 레이블 클래스 개수인 num\_class 지정
  + multi:softprob: multi:softmax와 유사하나 개별 레이블 클래스의 해당되는 예측 확률 반환
* eval\_metric: 검증에 사용되는 함수 정의, 기본값은 회귀인 경우는 rmse, 분류인 경우는 error

XGBoost 자체적으로 교차검증, 성능 평가, 피처 중요도 등의 시각화 기능을 가진다.

수행 속도 향상시키기 위한 대표 기능으로 조기 중단 기능 있음 🡪 기존 GBM은 n\_estimators에 지정된 횟수만큼 반복적으로 학습 오류 감소시키며 진행하여서 중간에 멈출 수 없음

과적합 문제 심각하다면

1. eta 값 낮추기, 이 경우 num\_round(n\_estimaotrs) 높여주기
2. max\_depth 낮추기
3. min\_child\_weight 높이기
4. gamma 높이기
5. subsample, colsample\_bytree 조정

**사이킷런 래퍼 XGBoost**

사이킷런의 기본 Estimator을 그대로 상속해 fit(), predict()로 학습/예측 가능, 사이킷런의 다른 유틸리티 그대로 사용 가능

분류를 위한 래퍼 클래스인 XGBClassifier, 회귀를 위한 래퍼 클래스인 XGBRegressor

파라미터

eta -> learning\_rate

sub\_sample -> subsample

lambda -> reg\_lambda

alpha -> reg\_alpha

n\_estimators

**LightGBM**

XGBoost에서 GridSearchCV로 하이퍼 파라미터 튜닝 수행하면 수행시간 너무 오래 걸림

* GBM보다는 빠르지만 많은 CPU 코어 가진 시스템에서 높은 병렬도로 학습 진행해야 함.
* LightGBM의 장점은 XGBoost보다 학습에 걸리는 시간 훨씬 적음, 예측 성능도 비슷함
* But, 적은 데이터 세트에 적용할 때(10,000건 이하)는 과적합이 발생하기 쉬움

일반 GBM계열 트리 분할 방법과 다르게 **리프 중심 트리 분할(Leaf Wise)** 방식 사용

(대부분 트리 기반 알고리즘은 최대한 균형 잡힌 트리를 유지하면서 트리 깊이가 최소화되는 균형 트리 분할 방식(Level Wise)을 사용함 -> 오버 피팅에 강하지만 균형 맞추기 위한 시간 필요함)

트리의 균형을 맞추지 않고, 최대 손실 값을 가지는 리프 노드를 지속적으로 분할하면서 트리 깊이 깊어지고 비대칭적인 규칙 트리 생성됨, 예측 오류 손실 최소화할 수 있음.

XGBoost 대비 장점

* 더 빠른 학습과 예측 수행시간
* 더 작은 메모리 사용량
* 카테고리형 피처의 자동 변환과 최적 분할(원-핫 인코딩 등을 사용하지 않아도 됨)

XGBoost와 동일하게 초기에 파이썬 래퍼용 LightGBM만 개발됐지만 사이킷런 래퍼 LightGBM도 추가로 개발됨. -> 사이킷런 래퍼 LightGBM 클래스는 LGBMClassifier(분류), LGBMRegressor(회귀)

**LightGBM 하이퍼 파라미터**

XGBoost와 유사하지만 LightGBM은 리프 노드가 계속 분할되면서 트리 깊이 깊어지므로 이러한 특성 가지는 하이퍼 파라미터 설정 필요 -> max\_depth 크게 가짐

* num\_iterations: 반복 수행하려는 트리 개수, 디폴트 100, 사이킷런 호환 클래스에서는 n\_estimators로 이름이 변경됨
* learning\_rate
* max\_depth
* min\_data\_in\_leaf: 결정트리의 min\_samples\_leaf와 동일, 최종 결정 클래스인 리프 노드가 되기 위해서 최소한으로 필요한 레코드 수, 디폴트 20
* num\_leaves: 하나의 트리가 가질 수 있는 최대 리프 개수, 디폴트 31
* boosting: 부스팅의 트리 생성하는 알고리즘 -> gbdt, rf
* bagging\_fraction: 과적합 제어하기 위해 데이터를 샘플링 하는 비율 지정, GBM의 subsample과 동일 -> LightGBMClassifier에서는 subsample로 이름 변경, 디폴트 1.0
* feature\_fraction: 개별 트리 학습할 때마다 무작위로 선택하는 피처의 비율, LightGBM에서는 colsample\_bytree로 변경, 디폴트 1.0
* lambda\_l2: L2 regulation 제어를 위한 값, 클수록 과적합 감소, LightGBMClassifier에서는 reg\_lambda로 이름 변경, 디폴트 0.0
* lambda\_l1: L1 regulation 제어를 위한 값, 클수록 과적합 감소, LightGBMClassifier에서는 reg\_alpha로 이름 변경

Learning Task 파라미터

* objective: 최솟값을 가져야 할 손실함수, 회귀/다중 클래스 분류/이진분류인지에 따라 지정, Xgboost의 objective와 동일

**하이퍼 파라미터 튜닝 방안**

num\_leaves를 중심으로 min\_data\_in\_leaf, max\_depth를 조정하면서 복잡도 줄이기

* num\_leaves 개수 높이면 정확도 높아지지만 반대로 깊이 깊어져 복잡도 커져 과적합 영향도 커짐
* min\_data\_in\_leaf는 사이킷런 래퍼 클래스에서 min\_child\_samples로 이름 바뀜. 보통 큰 값으로 설정하면 트리 깊어지는 것 방지
* max\_depth는 깊이의 크기 제한, 위의 파라미터와 결합해 과적합 개선
* learning\_rate 작게하면서 n\_estimators 크게 하는 것도 하나의 방안, 너무 크게하면 안됨
* reg\_lambda, reg\_alpha 적용 등등

**파라미터 명 비교**

|  |  |  |
| --- | --- | --- |
| **파이썬 래퍼 LightGBM** | **사이킷런 래퍼 LightGBM** | **사이킷런 래퍼 XGBoost** |
| 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 | X |
| 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 | X |
| min\_sum\_hessian\_in\_leaf | min\_child\_weight | min\_child\_weight |

**언더 샘플링과 오버 샘플링의 이해**

이상 레이블을 가지는 데이터 건수가 정상 레이블을 가진 데이터 건수에 비해 너무 적어서 다양한 유형을 학습하지 못하는 반면 정상 레이블 데이터 건수는 매우 많아 일방적으로 정상 레이블로 치우친 학습을 수행해 예측 성능의 문제가 발생할 수 있다.

지도 학습에서 극도로 불균형한 레이블 값 분포로 인한 문제점을 해결하기 위해 적절한 학습 데이터를 확보해야 함. 대표적으로 **오버 샘플링(Oversampling)**과 **언더 샘플링(Undersampling)**

1. 오버 샘플링

적은 데이터 세트를 증식하여 학습을 위한 충분한 데이터 확보하는 방법, 단순히 증식하면 과적합이 되기 때문에 원본 데이터의 피처 값들을 아주 약간만 변경하여 증식

* SMOTE(Synthetic Minority Over-sampling Technique): 적은 데이터 세트에 있는 개별 데이터들의 K 최근접 이웃을 찾아서 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어서 기존 데이터와 약간 차이가 나는 새로운 데이터 생성

1. 언더 샘플링

많은 데이터 세트를 적은 데이터 세트 수준으로 감소, 과도하게 정상 레이블로 학습하는 부작용을 개선하지만 너무 많은 정상 레이블을 감소시키기 때문에 오히려 정상 레이블이 제대로 된 학습을 할 수 없는 단점 -> 잘 적용하지 않음

**이상치 데이터(Outlier)**: 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터

IQR(Inter Quantile Range): 사분위 값의 편차를 이용한 기법, 박스 플롯방식(Box Plot)으로 시각화

사분위: 전체 데이터를 값이 높은 순으로 정렬하고 1/4씩으로 구간 분할, Q1~Q4 존재

🡪 Q1 ~ Q3의 범위 = IQR

IQR을 이용해 이상치 데이터 검출하는 방식은 보통 IQR에 1.5를 곱해서 생성된 범위를 이용해 최댓값과 최솟값 결정한 뒤 최댓값을 초과하거나 최솟값에 미달하는 데이터를 이상치로 간주, 경우에 따라 1.5가 아닌 다른 값 적용

**스태킹 앙상블**

스태킹(Stacking): 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출한다는 점에서 배깅 및 부스팅과 공통점 가짐

* 차이점은 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측 수행
* 개별 알고리즘의 예측 결과 데이터를 최종 메타 데이터로 만들어 별도의 알고리즘으로 최종 학습 수행하고 테스트 데이터로 다시 최종 예측 수행

스태킹 모델은 두 종류의 모델이 필요

1. 개별적인 기반 모델
2. 이 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어서 학습하는 최종 메타 모델

스태킹 적용할 때는 많은 개별 모델이 필요, 일반적으로 성능 비슷한 모델 결합해 조금 더 나은 성능 향상 도출하기 위해 적용됨

**cv 세트 기반의 스태킹**

과적합을 개선하기 위해 최종 메타 모델을 위한 데이터 세트 만들 때 교차 검증 기반으로 예측된 결과 데이터 세트 이용

스텝1: 각 모델별로 원복 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터 생성(교차 검증을 통해서)

🡪 학습용 데이터를 n개의 fold로 나누고 그것을 가지고 학습 및 예측 수행, 예측 결과값을 모아 최종 메타 모델에서 학습데이터로 사용, 각 학습된 개별 모델로 테스트 데이터 예측한 결과값을 모아 평균을 내어 최종 메타 모델에서 테스트 데이터로 사용

스텝 2: 스텝 1에서 개별 모델들이 생성한 학습용/테스트용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 최종 학습용/예측할 테스트용 데이터 세트 생성

🡪 **최종 생성된 학습 데이터 세트**와 **원본 학습 데이터의 레이블 데이터** 기반으로 학습한 뒤, **최종 생성된 테스트 데이터 세트**를 예측하고 **원본 테스트 데이터의 레이블 데이터**를 기반으로 평가

**정리**

대부분의 앙상블 기법은 결정 트리 기반의 다수의 약한 학습기를 결합해 변동성 줄여 예측 오류 줄이고 성능 개선

결정 트리: 정보의 균일도에 기반한 규칙 트리 만들어 예측 수행, 직관적이지만 과적합 쉽게 발생

앙상블

* 배깅: 학습 데이터를 중복 허용하여 다수의 세트로 샘플링 후 다수의 약한 학습기가 학습한 뒤 최종 결과를 결합해 예측 🡪 랜덤 포레스트
* 부스팅: 학습기들이 순차적으로 학습하면서 예측 틀린 데이터들에 대해 가중치 부여해 다음 학습기가 학습할 때 예측 틀린 데이터들에 대해서는 보다 높은 정확도로 예측 가능

🡪 GBM, XGBoost, LightGBM, 스태킹 모델, …