CH4

분류

1. **분류의 개요**

- 학습 데이터로 주어진 데이터의 피처와 레이블값을 학습해 모델 생성 후 새 데이터 값에 대한 마지막 레이블 값을 예측

2. **결정 트리**

- 데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 트리 기반의 분류 규칙을 만드는 것
- 규칙 노드 : 규칙 조건 , 리프 노드: 결정된 분류값
- 새로운 규칙 조건마다 서브 트리 생성
- 피처가 결합해 규칙 조건을 만들 때마다 규칙 노드 생성됨
- 트리의 깊이가 깊어질수록 결정 트리의 예측 성능이 저하될 가능성이 높음
- 적은 결정 노드로 높은 정확도를 가지려면 최대한 많은 데이터 세트가 해당 분류에 속하도록 결정 노드의 규칙이 정해져야 함

균일도

- 데이터 구분에 필요한 정보의 양에 영향
- 결정노드는 정보 균일도가 높은 데이터 세트를 먼저 선택할 수 있게 규칙 조건 만듦!
- 균일도 측정 방법
  - 엔트로피 이용한 정보이득 지수(1-엔트로피 지수 높을수록 균일)
  - 지니계수(낮을수록 균일) → DecisionTreeClassifier

특징

- 장점: 정보의 균일도라는 명확한 룰, 알고리즘 직관적, 보통 전처리 과정 불필요, 시각화 가능

- 단점: 과적합으로 인한 정확도 저하 → 피처 많고, 균일도 다양할 수록 깊이 커져 성능 저하

파라미터

min_samples_split	- 노드를 분할하기 위한 최소 샘플 데이터 수 - 디폴트 2, 작을수록 과적합의 가능성 증가 - 과적합 제어 용도
min_samples_leaf	- 말단 노드가 되기 위한 최소 샘플 데이터 수 - 과적합 제어 용도 - 비대칭적 데이터 → 특정 클래스 데이터가 극소일 가능성 존재하므로 작게 설정 필요
max_features	- 최적의 분할을 위한 최대 피처 개수 - 디폴트 none (전체 피처 사용) - int형→ 피처 개수, float형→ 퍼센트 - sqrt, log, auto 등
max_depth	- 트리의 최대 깊이 - 디폴트 none (완전히 클래스 결정값이 될 때까지 계속 분할 / 노드 데이터 개수가 min_samples_split보다 적을 때까지)
max_leaf_nodes	- 말단 노드의 최대 개수

시각화

- Graphviz 패키지
- 더 이상 자식 노드 없으면 → 리프 노드 → 최종 클래스 값이 결정되는 노드
- 자식 노드가 있는 노드 → 브랜치 노드
- 피처의 조건 → 규칙 조건 / 없다면 리프 노드
- value는 클래스 값 기반 데이터 건수 / 0,1,2, = setosa, versicolor, virginica
- 노드 색 : 레이블 값 / 주황,초록,보라
- 색이 짙을수록 지니계수 낮음!
- `feature_importances_` : 피처 중요도 반환

결정트리 과적합

- 이상치 데이터까지 분류하기 위해 결정 기준 경계가 복잡해질 수 있음
- 리프 생성 규칙 완화하면 좀 더 일반화 된 모습

3. **앙상블 학습**

- 정형 데이터 분류 시 성능 좋음
- 보팅 : 서로 다른 알고리즘을 가진 분류기 결합
    - 하드 보팅: 예측 결과값 중 다수 분류기가 결정한 예측을 최종 보팅 결과값으로

    - 소프트 보팅: 분류기의 레이블 결정 확률을 더하고 평균내어 확률이 가장 높은 레이블 값을 최종 보팅 결과값으로 → 주로 사용
- 배깅 : 같은 유형의 알고리즘 기반 분류기, 데이터 샘플링 다름
    - 개별 classifier 에게 데이터를 샘플링해서 추출 → 부트스트래핑 분할 방식
    - 중첩 허용
    - 랜덤포레스트
- 부스팅
    - 여러 분류기 순차 학습/ 예측 틀린 데이터에 다음 분류기에 가중치 전달
    - 그래디언트 부스팅, XGBoost, LightGBM

4. **랜덤 포레스트**

- 배깅 기반, 비교적 빠른 속도와 높은 예측 성능 보유
- 결정 트리의 직관적 특성 보유
- 부트스프래핑
- 트리 기반 앙상블 알고리즘 단점: 하이퍼 파라미터 너무 많아 튜닝 시간 너무 길다, 성능 향상도 미미
- n_estimators, max_features 등

[SVM]

5. **GBM(Gradient Boosting Machine)**

- 부스팅 알고리즘 : 약한 학습기 여러 개로 학습, 가중치 주면서 오류 개선
- 분류 회귀 모두 가능
- 랜덤 포레스트보다 성능 높은 경우 많음, 과적합에도 강함
- 수행 시간 오래 걸림 , 하이퍼 파라미터 튜닝 노력 필요

[종류]

`AdaBoost`

- 개별 약한 학습기로 돌려 잘못 분류된 데이터에 가중치를 부여해 결합

`그래디언트 부스트`

- 에이다부스트처럼 반복마다 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔여 오차에 새로운 학습기를 학습시킴
- 오류식 h(x)=y-F(x) (실제값 - 예측값)를 최소화하는 방향으로 가중치 값을 업데이트 → 경사하강법

[하이퍼 파라미터]

loss	경사하강법에서 사용할 비용함수. 기본값: deviance
learning_rate	학습률. 0~1사이 기본 0.1. 값이 너무 작다면 최소 오류 값을 찾아 예측 성능 높아질 가능성 높지만, 시간 오래 걸리고 최소 오류값을 못 찾을 수도 있음. 너무 크다면 최소 오류 값을 찾지 못하고 지나쳐 버려 예측 성능 떨어질 수도, 시간은 빠를 것. * n_estimaotrs 상호 보완 사용 → learning rate 작게, n_estimators 크게하면 예측 성능 조금씩 좋아질 수도
n_estimators	weak learner 개수. 기본값: 100. 많을 수록 성능 좋아지나 시간 오래 걸림.
subsample	weak learner가 학습에 사용하는 데이터 샘플링 비율. 기본값 :1 (전체 학습 데이터 기반 학습) * 과적합 걱정되는 경우 1보다 작게 설정.

6. **XGBoost(eXtra Gradient Boost)**

- 트리 기반
- 뛰어난 예측 성능
- GBM 기반이지만 수행 시간 및 과적합 규제 부재 문제 해결된 알고리즘
- 병렬 학습 가능
- Tree pruning : 긍정 이득 없는 분할을 가지치기해서 분할 수를 줄임
- 자체 내장된 교차 검증
- 조기 중단 기능 : 예측 오류가 더 이상 개선되지 않으면 반복을 끝까지 수행하지 않고 중지
- 결손값 자체 처리

  1) **파이썬 래퍼 XGBoost 모듈** : 초기 독자적인 프레임 워크 기반

  [주요 일반 파라미터] - 선택을 위한, 디폴트 값 그대로 사용

  - booster : gbtree(디폴트) or gblinear
  - silent : 디폴트 0. 출력 메세지 안 보려면 1.
  - nthread : CPU 실행 스레드 개수, 디폴트 모두

  [주요 부스터 파라미터] - 트리 최적화, 부스팅, 규제 등 관련

**괄호 안 : 디폴트, 호환 용어

- eta(0.3, learning_rate) : 파이썬 기반 기준. 사이킷런 래퍼→ 디폴트 0.1, 0.01~0.2 선호
- num_boost_rounds : =n_estimators
- min_child_weight(1) : 트리에서 추가로 가지 나눌지 결정하기 위해 필요한 데이터 weight 총합. 클수록 분할 자제. 과적합 제어 용도.
- gamma(0, min_split_loss) : 리프 노드 추가로 나눌지 결정할 최소 손실 감소 값. 해당 값보다 손실이 감소된 경우 분리. 클수록 과적합 감소.
- max_depth(6) : 0=깊이 제한 x. 높으면 과적합 가능성 높. 보통 3~10 사용
- subsample(1) : 트리가 커져 과적합 제어 위한 샘플링 비율. 보통 0.5~1 사용
- colsample_bytree(1) : =max_features. 트리 생성 필요한 피처 샘플링하는데 사용. 많은 피처있는 경우 과적합 제어 용도로 사용.
- lambda(1, reg_lambda) : L2 적용 값. 피처 개수 많으면 적용 검토, 클수록 과적합 감소 효과
- alpha(0, reg_alpha): L1 적용 값. 피처 개수 많으면 적용 검토, 클수록 과적합 감소 효과
- scale_por_weight(1) : 비대칭 클래스로 구성된 데이터 셋 균형 유지 위해 사용

[학습 테스크 파라미터] 학습 수행 시 객체 함수나 지표 등 설정

- objective : 최소값 가져야할 손실함수
- binary:logistic 이진분류
- multi:softmax 다중 분류 , num_class 지정해야 함.
- multi:softprob 위와 비슷, 예측 확률 반환
- eval_metric : 검증 사용 함수. 기본) 회귀 mse, 분류 error

  유형 → rmse, mae, logloss, error, merror, mlogloss, auc

[과적합 심하다면?]

eta 낮(이 경우 num_round | n_estimators 높) , max_depth 낮, min_child_weight 높, gamma 높, subsample이나 colsample_bytree 조정

2) **사이킷런 래퍼 XGBoost 모듈** : 사이킷런 전용 래퍼 클래스

- 용어 번경 전→후

eta	learning_rate
sub_sample	subsample
lambda	reg_lambda
alpha	reg_alpha

7. **lightGBM**

- XGBoost보다 학습 시간 짧음, 메모리 사용량도 더 적음
- 적은 데이터 세트 사용시 과적합 발생 쉬움 → 10000건 이하
- 리프 중심 트리 분할 : 손실 값을 가지는 리프 노드 지속적 분할, 트리 비대칭 깊어짐, 최대 손실값을 가지는 리프노드 분할하여 예측 오류 손실 최소화
  ↔ 균형 트리 분할(GBM 방식) : 오버피팅 강하나 시간 필요
- 카테고리형 피처 자동 변환&최적 분할

[하이퍼 파라미터]

num_iterations	반복 수행하려는 트리 개수. 디폴트 100. 클수록 예측 성능 높아질 수는 있으나 과적합 가능성도 증가. (=n_estimators)
learning_rate	학습률. 디폴트 0.1.
max_depth	디폴트 1. Leaf wise 기반이므로 깊이가 더 깊음.
min_data_in_leaf	최종 결정 클래스인 리프노드가 되기 위해 최소 필요한 레코드 수. 디폴트 20. (=min_samples_leaf=min_child_samples) 과적합 제어 용도.
num_leaves	하나의 트리가 가질 수 있는 최대 리프 개수. 디폴트 3.
boosting	부스팅 트리 디폴트 = gbdt(그래디언트 부스팅 결정 트리) / rf(랜덤포레스트)
bagging_fraction	과적합 제어하기 위한 데이터 샘플링 비율. 디폴트 1.0. (=subsample)
feature_fraction	개별 트리 학습할 때마다 무작위로 선택하는 피처 비율. 디폴트 1.0. 과적합 제어 용도. (=colsample_bytree)
lambda_l2	L2 규제를 위해. 디폴트 0.0. 피처 개수 많으면 적용 검토하고 값이 클수록 과적합 감소. (=reg_lambda)
lambda_l1	L1 규제를 위해. 디폴트 0.0. 과적합 제어 용도. (=reg_alpha)
objective (Learning task 파라미터)	최소값을 가져야 할 손실 함수

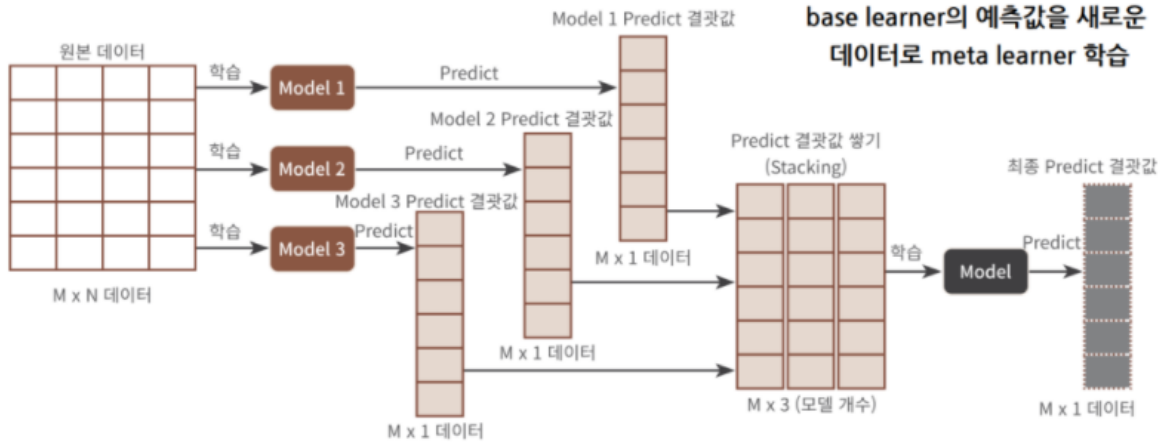
유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
파라미터명	num_iterations	n_estimators	n_estimators
	learning_rate	learning_rate	learning_rate
	max_depth	max_depth	max_depth
	min_data_in_leaf	min_child_samples	N/A
	bagging_fraction	subsample	subsample
	feature_fraction	colsample_bytree	colsample_bytree
	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

10. **스태킹 앙상블**

- 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측 수행

1) 개별적인 기반 모델

2) 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어 학습한 최종 메타 모델



CV 세트 기반 스태킹

1) 각 모델별 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습/테스트용 데이터 생성

2) 스텝1에서 생성한 데이터를 스태킹 형태로 합쳐 최종 학습용/테스트용 데이터 세트 생성. 원본/생성된 학습 데이터로 학습시킨 후 생성된 테스트 데이터 세트를 예측하고 원본 테스트 데이터의 레이블 데이터를 기반으로 평가