파머완 4장 Part 2 필사 & 개념정리

4-5 부스팅 계열

부스팅 알고리즘은 여러 개의 약한 학습기를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치를 부여해 오류를 개선해 나가면서 학습하는 방식.

대표적인 구현으로는 Adaboost와 Gradient boost가 있음.

- Ada → 약한 학습기가 순차적으로 오류 값에 대해 가중치를 부여한 후 예측 결정 기준을 모두 결합해 예측 수행. → 따라서 개별 약한 학습기보다 훨씬 정확도가 높아졌음을 알 수 있음
- GBM → 에이다와 유사하나, 가중치 업데이트를 경사 하강법(gradient descent)을 이용함.

경사 하강법? '반복 수행을 통해 오류를 최소화할 수 있도록 가중치의 업데이트 값을 도출하는 기법' 정도로만 알고 있으면 됨.

rf 와 비교해 봤을 때, 더 나은 예측 성능을 나타냄. 일반적으로 성능이 뛰어나나, 시간이 오래 걸리고 수행 시간도 김. → 순차적으로 가중치를 부여해야 하기 때문에 CPU로 하면 한계가 있음+하이퍼 파라미터도 많음.

GBM 하이퍼 파라미터 소개

하이퍼 파라미터	설명	기본값
loss	경사 하강법에서 사용할 비용함수 지정	deviance
learning_rate	GBM이 학습을 진행할 때마다 적용하는 학습률. Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수 (0~1 사이 값)	0.1
n_estimators	weak learner의 개수. 개수가 많을수록 예측 성능이 일정 수준까지 좋아질 수 있으나 수행 시간이 오래 걸림	-
subsample	weak learner가 학습에 사용되는 데이터의 샘플링 비율. 과적합이 염려될 때는 1보다 작은 값으로 설정	1

*learning_rate 특성:

- 너무 작은 값: 최소 오류값을 찾아 예측 성능이 높아질 가능성이 높으나, 너무 오래 걸림
- 너무 큰 값: 최소 오류값을 찾지 못할 수도 있으나 빠른 수행이 가능함

- n_estimators와 상호 보완적 관계: learning rate를 작게 하고, n_estimator을 크게 하면 더 이상 성능이 좋아지지 않는 한계점까지 예측 성능이 조금씩 좋아질 수 있음

4-6 XGBoost(Extra Gradient Boost)

앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나. 분류에 있어서 일반적으로 다른 머신러닝보다 뛰어난 예측 성능을 나타냄.

항목	설명
뛰어난 예측 성능	일반적으로 분류와 회귀 영역에서 뛰어난 예측 성능을 발휘합니다.
GBM 대비 빠른 수행 시간	일반적인 GBM은 순차적으로 Weak learner가 가중치를 증감하는 방법으로 학습하기 때문에 전반적으로 속도가 느립니다. 하지만 XGBoost는 병렬 수행 및 다양한 기능으로 GBM에 비해 빠른 수행 성능을 보장합니다. 아쉽게도 XGBoost가 일반적인 GBM에 비해 수행 시간이 빠르다는 것이지, 다른 머신러닝 알고리즘(예를 들어 랜덤 포레스트)에 비해서 빠르다는 의미는 아닙니다.
과적합 규제 (Regularization)	표준 GBM의 경우 과적합 규제 기능이 없으나 XGBoost는 자체에 과적합 규제 기능으로 과적합에 좀 더 강한 내구성을 가질 수 있습니다.
Tree pruning (나무 가지치기)	일반적으로 GBM은 분할 시 부정 손실이 발생하면 분할을 더 이상 수행하지 않지만, 이러한 방식도 자칫 지나치게 많은 분할을 발생할 수 있습니다. 다른 GBM과 마찬가지로 XGBoost도 max_depth 파라미터로 분할 깊이를 조정하기도 하지만, tree pruning으로 더 이상 긍정 이득이 없는 분할을 가지치기 해서 분할 수를 더 줄이는 추가적인 장점을 가지고 있습니다.
자체 내장된 교차 검증	XGBoost는 반복 수행 시마다 내부적으로 학습 데이터 세트와 평가 데이터 세트에 대한 교차 검증을 수행해 최적화된 반복 수행 횟수를 가질 수 있습니다. 지정된 반복 횟수가 아니라 교차 검증을 통해 평가 데이터 세트의 평가 값이 최적화 되면 반복을 중간에 멈출 수 있는 조기 중단 기능이 있습니다.
결손값 자체 처리	XGBoost는 결손값을 자체 처리할 수 있는 기능을 가지고 있습니다.

*원래는 C/C++로 작성되어 있으나 파이썬 패키지(xgboost)를 이용해서 사용할 수 있음.

XGBoost 패키지의 사이킷런 래퍼 클래스는 XGBClassifier와 XGBRegressor입니다.→ 이를 활용해서 사이킷런의 fit(),predict()사용 가능.

- 파라미터 요약

1. 기본

파라미터	설명
booster	트리(`gbtree` , 기본) / 선형(`gblinear`) 선택
nthread	사용할 CPU 스레드 수 (기본: 전체 사용)

2. 트리 파라미터

파라미터 (별칭)	설명 / 튜닝 포인트
learning_rate (eta)	학습률. 작게(0.01~0.2) 설정 + `n_estimators` 크게.
n_estimators (num_boost_round)	부스팅 반복 횟수(트리 개수).
max_depth	트리 깊이 제한. 깊을수록 복잡·과적합 위험 ↑. (3~10 권장)
min_child_weight	자식 분할 허용 최소 가중치 합. 클수록 보수적(과적합↓).
gamma (min_split_loss)	리프 분할 시 최소 손실 감소값. 크게 설정 시 과적합 완화.
subsample	학습 샘플(행) 비율. 0.5~1.0 (과적합 방지).
colsample_bytree	피처(열) 샘플링 비율. 피처 많을 때 과적합 억제.
reg_lambda (λ_2)	L2 정규화 강도. 클수록 과적합↓.
reg_alpha (λ_1)	L1 정규화 강도. 클수록 과적합↓.
scale_pos_weight	불균형 데이터에서 소수 클래스 가중치 조정.

3. 학습 태스크 파라미터

파라미터	설명
objective	손실함수: • `binary:logistic` (이진) • `multi:softmax` (다중, `num_class` 필요) • `multi:softprob` (다중, 확률 출력)
eval_metric	평가지표 선택: `rmse` , `mae` , `logloss` , `error` , `merror` , `mlogloss` , `auc` 등
early_stopping_rounds	지정 라운드 동안 개선 없으면 학습 중단 (검증 데이터 필요)

4. 튜닝 개념 요약

- **학습률–반복수 페어링**: `learning_rate ↓` ↔ `n_estimators ↑`
- **복잡도 제어**: `max_depth` , `gamma` , `min_child_weight` ↑ → 보수적 분할 → 과적합 완화
- **샘플링**: `subsample` , `colsample_bytree` < 1 → 다양성 확보, 과적합 억제
- **불균형 데이터**: `scale_pos_weight` 조정

- **평가지표**: 목적에 맞춰 `eval_metric` 선택

- 위스콘신 유방암 예측

-XGBoost만의 전용 데이터 객체인 DMatrix를 사용: 데이터 세트를 모두 이걸로 변환해서 전달해 주어야 함.

-xgboost 내의 시각화 할 수 있는 게 있음. plot_importance() API활용

-파이썬 래퍼 XGBoost는 사이킷런의 GridSearchCV와 유사하게 데이터 세트에 대한 교차 검증 수행 후 최적 파라미터를 구할 수 있는 방법을 cv() API로 제공

-조기 중단: 조기 중단 관련한 파라미터를 fit()에 입력하면 됩니다. 조기 중단 관련 파라미터는 평가 지표가 향상될 수 있는 반복 횟수를 정의하는 early_stopping_rounds, 조기 중단을 위한 평가 지표인 eval_metric, 그리고 성능 평가를 수행할 데이터 세트인 eval_set입니다.

*조기 중단값을 너무 급격하게 줄이면 예측 성능이 저하될 우려가 큽니다.

4-7 Light GBM

장점: XGBoost의 단점을 보완해서 나온 것이기 때문에 학습에 걸리는 시간이 훨씬 적음. 메모리 사용량도 적음.

단점: 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉽다는 것. 약 10,000건 이하의 데이터 세트 정도

주요 내용: 리프 중심 트리 분할 방식(leaf wise) 사용.

트리 깊이를 최소화하기 위해 기존의 대부분 트리는 균형 트리 분할 방식(level wise)을 사용함. 그러나 최대 손실 값을 가지는 리프 노드를 지속적으로 분할하면서 트리의 균형을 맞추지 않고, 트리가 깊어지고 비대칭적인 규칙 트리가 생성될 수 있음. → 예측 오류 손실을 최소화할 수 있다는 장점을 취함.

파이썬 패키지명: lightgbm

- 파라미터 요약

1. 일반 파라미터

파라미터	설명
boosting	부스팅 유형 선택. `gbdt` (기본), `dart` , `goss` 지원
num_iterations (n_estimators)	반복 학습 횟수(트리 개수)
learning_rate	학습률. 작게 설정(0.01~0.1) + 반복 수 크게

파라미터	설명
num_threads	CPU 스레드 수. 기본: 전체 사용

2. 트리 관련 파라미터 (핵심)

파라미터	설명 / 튜닝 포인트
max_depth	트리 최대 깊이. 기본 -1 (제한 없음). 3~12 범위 자주 사용
num_leaves	리프 노드 수 (모델 복잡도 제어 핵심). 너무 크면 과적합 ↑
min_data_in_leaf	리프 노드에 필요한 최소 데이터 수. 값이 클수록 단순화·과적합 방지
min_sum_hessian_in_leaf	리프 분할 시 최소 Hessian 합. 값이 크면 분할 덜 함
feature_fraction (colsample_bytree)	트리마다 사용할 피처 비율. 0.5~1.0
bagging_fraction (subsample)	데이터 샘플 비율. 0.5~1.0
bagging_freq	배깅 수행 주기(라운드 단위). 0이면 사용 안 함
lambda_l1	L1 정규화 강도. 클수록 가중치 희소화, 과적합 방지
lambda_l2	L2 정규화 강도. 클수록 과적합 방지
min_gain_to_split	분할이 일어나기 위한 최소 손실 감소량. 클수록 보수적 분할
max_bin	연속형 피처를 히스토그램으로 변환할 때 bin 개수. 값이 크면 정밀하지만 메모리↑, 속도↓

3. 학습 태스크 파라미터

파라미터	설명
objective	손실함수: • `binary` (이진 분류) • `multiclass` (다중 분류, `num_class` 필요) • `regression` (회귀) • `lambdarank` (랭킹)
metric	평가 지표: `binary_logloss` , `auc` , `multi_logloss` , `rmse` , `mae` 등
is_unbalance / scale_pos_weight	클래스 불균형 처리 옵션

4. 튜닝 개념 요약

- **주요 복잡도 제어 키**: `num_leaves` , `max_depth`

  → 과적합 방지 핵심: `num_leaves` 를 적절히 제어 (`2^(max_depth)` 보다 작게)
- **학습률–반복수 페어링**: `learning_rate ↓` ↔ `num_iterations ↑`
- **샘플링 기법**: `feature_fraction` , `bagging_fraction` , `bagging_freq` 를 조합해 과적합 억제

- **정규화:** `lambda_l1` , `lambda_l2` , `min_gain_to_split` 로 복잡도 제어
- **데이터 크기 최적화:** `max_bin` 조정으로 속도/메모리 트레이드오프 조절

*참고 파라미터 명 비교

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

- 위스콘신 유방암 예측

4-8 베이지안 최적화 기반의 HyperOpt를 이용한 하이퍼 파라미터 튜닝

지금까지의 하이퍼 파라미터 튜닝을 위해 사이킷런이 제공하는 Grid Search 방식을 적용했음

→ 단점: 튜닝해야 하는 하이퍼 파라미터 개수가 많을 경우, 최적화 수행 시간이 오래 걸린다는 것.

→ 따라서 실무의 대용량 학습 데이터에 XGBoost나 LightGbM의 하이퍼 파라미터 튜닝 시에 베이지안 최적화 기법 사용.

베이지안 최적화: 목적 함수 식을 제대로 알 수 없는 블랙 박스 형태의 함수에서 최대 또는 최소 함수 반환 값을 만드는 최적 입력값을 가능한 적은 시도를 통해 빠르고 효과적으로 찾

아주는 방식

베이지안 확률에서처럼 새로운 데이터를 입력받았을 때 최적 함수를 예측하는 사후 모델을 개선해 나가면서 최적 함수 모델을 만들어냄.

▼ 베이지안 확률 이해하기

- **빈도주의 확률**: 어떤 사건이 무한히 반복되었을 때의 상대적 빈도. (ex: 동전 던지기를 무한히 하면 앞면 확률은 0.5에 수렴)
- **베이지안 확률**: 사건이 일어날 가능성에 대한 **우리의 믿음(신뢰도)**을 확률로 표현. 새로운 정보가 들어오면 이 믿음을 업데이트함.

📌 **핵심 아이디어**

- "확률은 단순히 횟수가 아니라, 우리가 가진 지식(사전 확률 prior)에 기반한 믿음이다."
- 베이즈 정리:Posterior∝Prior×Likelihood

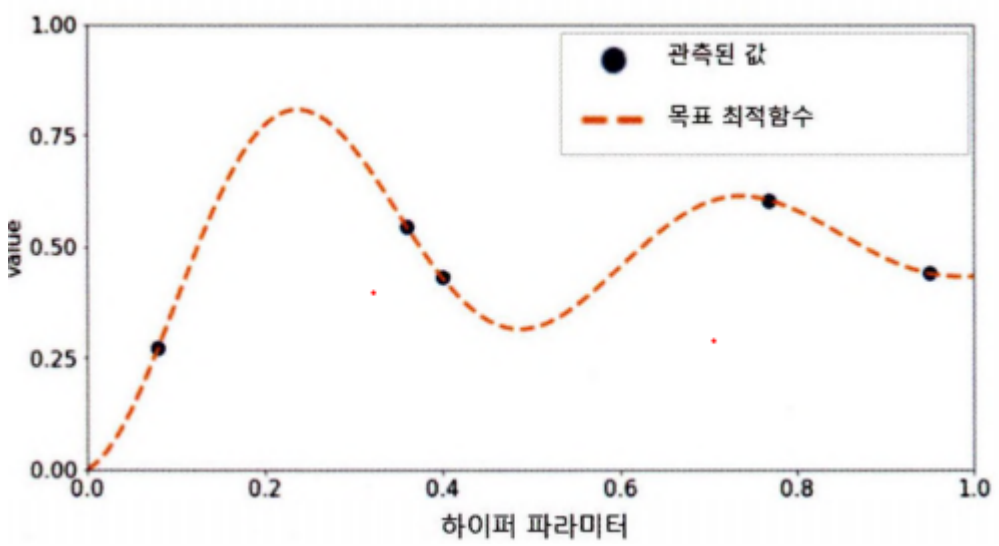
  Posterior∝Prior×LikelihoodPosterior \propto Prior \times Likelihood

  → 새로운 데이터를 관찰하면, 기존 믿음(prior)을 업데이트해서 **사후 확률(posterior)**을 구한다.

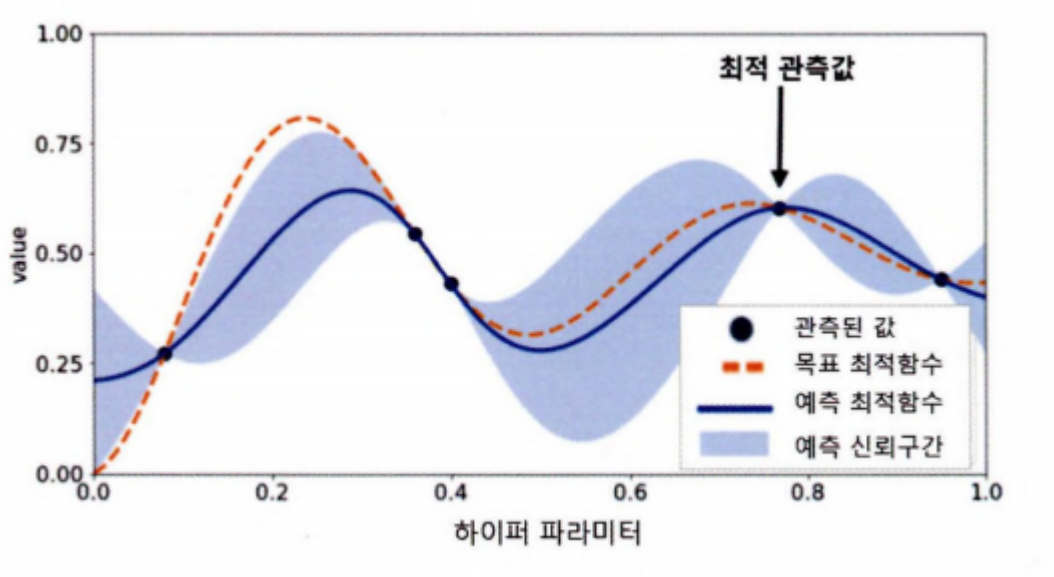
  - **랜덤/그리드 서치**: 마치 **보물찾기를 눈가리고 아무데나 찍어보는 것**.
  - **베이지안 최적화**:
    - "여태까지 땅을 파봤더니 금속 탐지기가 많이 울린 곳은 동쪽 근처였다."
    - → 그러면 "동쪽 근처를 더 파보자"라는 식으로 **지금까지의 정보(사후 확률)**를 바탕으로 전략적으로 탐색.
  - **베이지안 확률**: 확률은 "사건이 일어날 가능성에 대한 믿음", 새로운 정보가 들어오면 업데이트됨.
  - **베이지안 최적화**: 지금까지의 시도와 결과를 바탕으로, **확률 모델을 세워서 탐색을 더 똑똑하게 하는 방법**.
  - **HyperOpt**: 이를 구현한 라이브러리, TPE 방식을 사용해 효율적으로 좋은 하이퍼파라미터를 찾음.

베이지안 최적화 단계:

1. 최초에는 랜덤하게 하이퍼 파라미터 샘플링 및 성능 결과 관측.



2. 관측된 값을 기반으로 대체 모델은 최적 함수 추정. 옅은 파란색은 함수의 신뢰 구간으로, 추정된 함수의 결괏값 오류 편차를 의미하며 추정 함수의 불확실성을 나타냅니다. 최적 관측값은 y축 value에서 가장 높은 값을 가질 때의 하이퍼 파라미터입니다.



3. 추정된 최적 함수를 바탕으로 획득 함수는 다음으로 관측할 하이퍼 파라미터 값 계산. 이전의 최적 관측값보다 더 큰 최댓값을 가질 가능성이 높은 지점을 찾아서 다음에 과녁할 하이퍼 파라미터 대체 모델에 전달.
4. 다시 갱신해서 다시 최적 함수를 예측 추정.
5. 이런 방식으로 step3과 4를 반복하면 대체 모델의 불확실성이 개선되고 정확한 최적 함수 추정이 가능하게 됨. 대체 모델은 최적 함수를 추정할 때 가우시안 프로세스 적용.

- HyperOpt 사용하기
- HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화

4-10 분류 실습 -캐글 신용사기 검출

- 언더 샘플링과 오버 샘플링의 이해

레이블이 불균형한 분포를 가진 데이터 세트를 학습시킬 때, 예측 성능의 문제가 발생할 수 있음. 지도학습에서 극도로 불균형한 레이블 값 분포로 인한 문제점을 해결하기 위해 적절한 학습 데이터를 확보하는 방안→ 언더 샘플링과 오버 샘플링. 오버 샘플링이 예측 성능상 유리한 경우가 많아 상대적으로 더 많이 사용됨.

언더 샘플링: 많은 레이블을 가진 데이터 세트를 적은 레이블을 가진 데이터 세트 수준으로 감소

오버 샘플링: 적은 레이블을 가진 데이터 세트를 많은 레이블을 가진 데이터 세트 수준으로 증식

→ 대표적으로 SMOTE(Synthetic Minority Over-sampling Technique) 방법이 있습니다.
SMOTE는 적은 데이터 세트에 있는 개별 데이터들의 K 최근접 이웃(K Nearest Neighbor)을 찾아서 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어서 기존 데이터와 약간 차이가 나는 새로운 데이터들을 생성

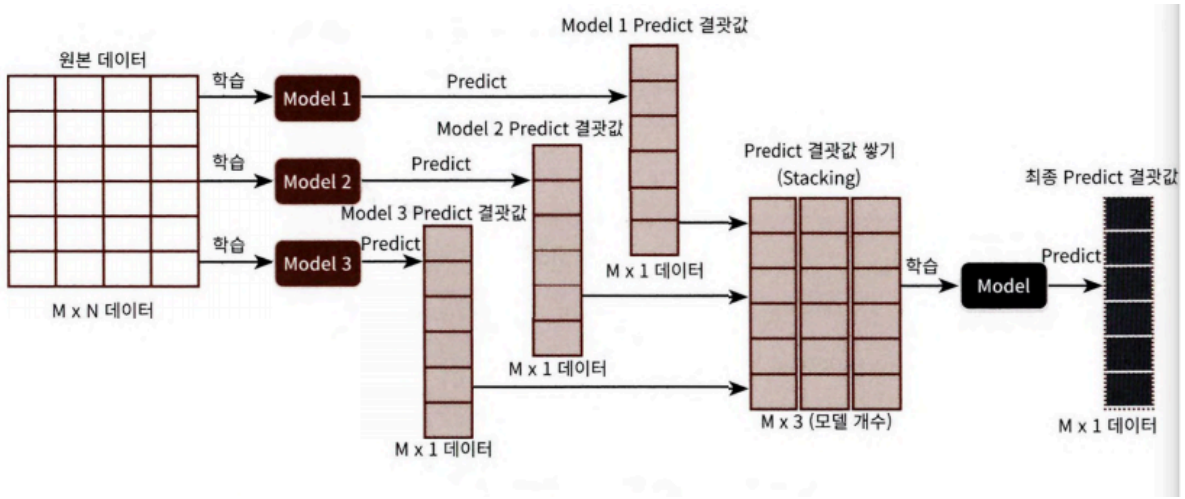
데이터 가공 유형	머신러닝 알고리즘	평가 지표		
		정밀도	재현율	ROC-AUC
원본 데이터 가공 없음	로지스틱 회귀	0.8679	0.6216	0.9702
	LightGBM	0.9573	0.7568	0.9790
데이터 로그 변환	로지스틱 회귀	0.8812	0.6014	0.9727
	LightGBM	0.9576	0.7635	0.9796
이상치 데이터 제거	로지스틱 회귀	0.8750	0.6712	0.9743
	LightGBM	0.9603	0.8288	0.9780
SMOTE 오버 샘플링	로지스틱 회귀	0.0542	0.9247	0.9737
	LightGBM	0.9118	0.8493	0.9814

4-11 스태킹 앙상블

배깅(Bagging) 및 부스팅(Boosting)과 가장 큰 차이점은 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행한다는 것입니다. 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 방식입니다(이렇게 개별 모델의 예측된 데이터 세트를 다시 기반으로 하여 학습하고 예측하는 방식을 메타 모델이라고 합니다).

→ 두 종류의 모델이 필요함. 1. 개별적인 기반 모델, 2. 이 개별 기반 모델의 예측 데이터(결과)를 가지고 학습 데이터로 만들어서 학습하는 최종 메타 모델.

쓰임새: 캐글과 같은 대회에서 높은 순위를 차지하기 위해 조금이라도 성능 수치를 높여야 할 경우 자주 사용됩니다. 스태킹을 적용할 때는 많은 개별 모델이 필요합니다. 2~3개의 개별 모델만을 결합해서는 쉽게 예측 성능을 향상시킬 수 없으며, 스태킹을 적용한다고 해서 반드시 성능 향상이 되리라는 보장도 없습니다. 일반적으로 성능이 비슷한 모델을 결합해 좀 더 나은 성능 향상을 도출하기 위해 적용됩니다.



- CV 세트 기반의 스태킹.

과적합을 개선하기 위해 최종 메타 모델을 위한 데이터 세트를 만들 때 교차 검증 기반으로 예측된 결과 데이터 세트를 이용합니다.

→ 개별 모델들이 각각 교차 검증으로 메타 모델을 위한 학습용 스태킹 데이터 생성과 예측을 위한 테스트용 스태킹 데이터를 생성한 뒤 이를 기반으로 메타 모델이 학습과 예측 수행.

1단계: 각 모델별로 원본 학습/테스트 데이터 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터를 생성

2단계: 모두 스태킹 형태로 합쳐서 최종 학습용 데이터 세트를 생성. 데스트용도 스태킹 형태로 합쳐져 메타 모델이 예측할 최종 테스트 데이터 세트 생성.

메타 모델은 최종적으로 생성된 학습 데이터 세트와 원본 학습 데이터의 레이블 데이터를 기반으로 학습한 뒤, 최종적으로 생성된 테스트 데이터 세트를 예측하고, 원본 테스트 데이터의 레이블 데이터를 기반으로 평가