## 05 GBM(Gradient Boosting Machine)

(1) GBM의 개요 및 실습

- 부스팅 알고리즘: 여러 개의 약한 학습기를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습
- 부스팅의 대표적인 구현

1. Adaboost
   - 약한 학습기가 잘못 분류한 오류 데이터 ⇒ 가중치 값 부여 반복
   - 약한 학습기가 순차적으로 가중치 부여한 예측 결정 기준 모두 결합해 예측 수행
2. 그래디언트 부스트
   - 에이다부스트와 유사
   - 경사 하강법(Gredient Descent)을 이용해 가중치 업데이트
   - 분류, 회귀 가능
   - 사이킷런 GredientBoostingClassifier 클래스

- 경사 하강법: 오류식을 최소화하는 방향성을 가지고 반복적으로 가중치 값 업데이트 하는 것

(2) GBM 하이퍼 파라미터 및 튜닝

- loss: 경사 하강법에서 사용할 비용 함수 지정 (디폴트 = deviance)
- learning_rate: GBM이 학습 진행할 때마다 적용하는 학습률
  - (0~1 사이 값 지정, 디폴트=0.1)
  - 약한 학습기가 순차적으로 오류 값을 보정하는데 적용하는 계수
  - 작은 값: 수행 시간↑, 최소 오류 값을 찾아 예측 성능↑
  - 큰 값: 수행 시간↓, 최소 오류 값 지나쳐 에측 성능↓
  - n_estimators와 상호 보완적으로 조합해 사용
- n_estimators: weak learner 수 (디폴트: 100)
  - 개수↑: 한계점까지 예측 성능이 일정 수준까지 좋아질 수 있음, but 수행 시간↑
- subsample: weak learner가 학습에 사용하는 데이터의 샘플링 비율

- 디폴트=1(전체 학습 데이터 기반의 학습)
- 과적합을 예방하려면 subsample을 1보다 작은 값으로 설정

## 06 XGBoost(eXtra Gredient Boost)

## XGBoost 개요

- GBM에 기반하며, GMB의 '느린 수행 시간, 과적합 규제 부재' 해결
- 멀티 CPU 환경에서 병렬 학습 가능 ⇒ 빠른 학습
- 사이킷런 XGBClassifier, XGBRegressor 래퍼 클래스
- XGBoost의 주요 장점

| 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. |

| 항목 | 설명 |
|---|---|
| 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.<br>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.<br>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. |

*초기의 독자적인 XGBoost 프레임워크 기반의 XGBoost: **파이썬 래퍼 XGBoost 모듈**

*사이킷런과 연동되는 모듈: **사이킷런 래퍼 XGBoost 모듈**

### 파이썬 래퍼 XGBoost 하이퍼 파라미터

- GBM 과 유사한 하이퍼 파라미터 가짐 + (조기 중단, 과적합 규제 하이퍼 파라미터 등이 추가)
- 파이썬 래퍼와 사이킷런 래퍼의 하이퍼 파라미터 이름이 약간 다르므로 주의

- 파이썬 래퍼 XGBoost 하이퍼 파라미터 유형
  - 일반 파라미터: 실행 시 스레드의 개수나 silent 모드 등의 선택을 위한 파라미터로

디폴트 파라미터 값을 바꾸는 경우가 거의 없음

- 부스터 파라미터: 트리 최적화, 부스팅, regularization 등과 관련 파라미터 등
- 학습 태스크 파라미터: 학습 수행 시의 객체 함수, 평가를 위한 지표 등을 설정

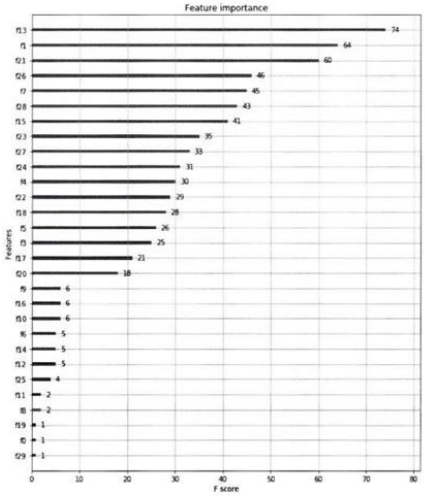
- 주요 일반 파라미터
  - booster: gbtree(tree based model, 디폴트) 또는 gblinear(linear model) 선택
  - silent: 디폴트=0, 1=출력 메세지X
  - nthread: CPU 실행 스레드 개수 조정 (디폴트=다 사용)/일부 CPU만 사용해 ML 애플리케이션 구동하는 경우 변경
- 주요 부스터 파라미터
  - eta [defualt=0.3, alias: learning_rate]: GBM의 학습률(0~1) 보통 0.01~0.2 적용
    사이킷런 래퍼 클래스 이용 시, learning_rate 파라미터, 디폴트 0.1
  - num_boost_rounds: GBM의 n_esimators
  - min_child_weight [default=1]: 트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 weight 총합
    클수록 분할 자제, 과적합 조절에 이용
  - gamma [default=0, alias: min_split_loss]: 트리의 리프 노드 추가적으로 나눌지 결정할 최소 손실 감소 값
    해당 값보다 큰 손실이 감소된 경우 리프 노드 분리, 클수록 과적합 감소
  - max_depth [default=6]: 트리 기반 알고리즘의 max_depth와 같음, 0 지정시 깊이 제한X
    높으면 특정 피처 조건에 특화되어 룰 조건 만들어져 과적합 가능성 증가
    보통 3~10 사이 값 적용
  - sub_sample [default=1]: GBM의 subsample, 트리 커져 과적합되는 것 제어 위해 데이터 샘플링하는 비율 지정
    보통 0.5~1
  - colsample_bytree [default =1]: GBM의 max_features와 유사, 트리 생성에 필요한 피처(칼럼)를 임의로 샘플링하는 데 사용
    피처 매우 많을 때 과적합 조정에 이용
  - lambada [default=1, alias: reg_lambda]: L2 Regularization 적용 값, 피처 개수 많을 때 적용 검토, 클수록 과적합 감소
  - alpha [default=0, alias: reg_alpha]: L1 Regularization 적용 값, 피처 개수 많을 때 적용 검토, 클수록 과적합 감소

- scale_pos_weight [default=1]: 특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트 균형 유지에 이용

- 학습 태스크 파라미터
  - objective: 최솟값을 가져야 할 손실 함수 정의
  - binary:logistic: 이진 분류일 때 적용
  - multi:softmax: 다중 분류일 때 적용, 손실 함수가 이거 일 경우 num_class 파라미터 지정해야 함
  - multi:softprob: 개별 레이블 클래스의 해당되는 예측 확률 반환
  - eval_metric: 검증에 사용되는 함수 정의, (디폴트: 회귀-rmse, 분류-error)
    rmse: Root Mean Square Error
    mae: Mean Absolute Error
    logloss: Negative log-likelihood
    error: Binary classfication error rate(0.5 threshold)
    merror: Multiclass classification error rate
    mlogloss: Multiclass logloss
    auc: Area under the curve

## 파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측

- eval_set: 성능 평가를 수행할 평가용 데이터 세트 설정
- eval_metric: 평가 세트에 적용할 성능 평가 방법. 분류일 경우 주로 'error', 'logloss' 적용
- plot_importance() : 피처의 중요도 막대그래프 형식으로 나타냄
  f스코어(해당 피처가 트리 분할 시 얼마나 자주 사용되었나) 기반으로 중요도 나타냄
  호출 시 파라미터로 학습이 완료된 모델 객체 및 맷플롯립의 ax 객체 입력

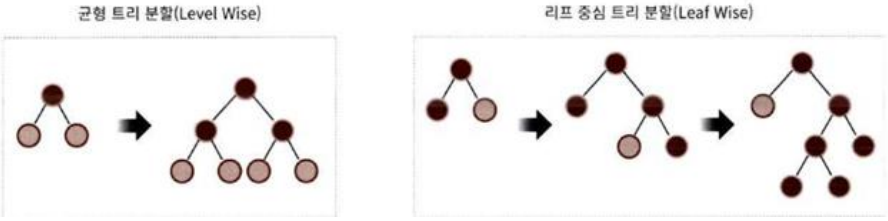


- to_graphviz() API 사용 (Graphviz 프로그램, 패키지 설치 필요)
  xgboost.to_graphviz()에 학습이 완료된 객체와 Graphviz가 참조할 파일명 입력
- cv() API: - 사이킷런의 GridSearchCV 처럼 데이터 세트에 대한 교차 검증 수행 후 최적 파라미터 구할 수 있는 방법 제공, 반환값 DF 형태
  - params (dict): 부스터 파라미터
  - dtrain (DMatrix): 학습 데이터
  - num_boost_round (int): 부스팅 반복 횟수
  - nfold (int): CV 폴드 개수
  - stratified (bool): CV 수행 시 층화 표본 추출(Stratified sampling) 수행 여부
  - metrics (string or list of string): CV 수행 시 모니터링할 성능 평가 지표
  - early_stopping_rounds (int): 조기 중단을 활성화시킴, 반복 횟수 지정

## 07 LightGBM

- XGBoost 대비 장점
  - 더 빠른 학습과 예측 수행 시간
  - 더 작은 메모리 사용량
  - 카테고리형 피처의 자동 변환과 최적 분할
- 단점
  - 적은 데이터 세트에 적용 시 과적합 발생하기 쉬움

- 리프 중심 트리 분할 방식 사용
  - 트리 균형 맞추지 않고, 최대 손실 값을 가지는 리프 노드 지속적으로 분할
    ⇒ 트리 깊이 깊어지고, 비대칭적 규칙 트리 생성
    ⇒ 결국 균형 트리 방식보다 예측 오류 손실 최소화할 것이다



- 사이킷런 래퍼 LightGBM 클래스 : 분류 LGBMClassifier, 회귀 LGBMRegressor

## LightGBM 하이퍼 파라미터

- XGBoost와 많이 유사
- 다른점 : 리프 노드가 계속 분할되며 트리 깊이 깊어지므로 이에 따른 하이퍼 파라미터 설정 필요
- 주요 파라미터
  - num_iterations [default=100]: 반복 수행하려는 트리의 개수 지정,<br>증가→예측 성능↑, (과도→과적합, 성능↓)<br>사이킷런 클래스: n_estimators
  - learning_rate [default=0.1]: 0~1 값, 학습률<br>보통 n_estimators↑, learning_rate↓ → 예측 성능↑<br>(과적합, 학습 시간 증가 주의)
  - max_depth [default=-1]: 트리 기반 알고리즘 max_depth과 같음<br>< 0 이면 깊이 제한X
  - min_data_in_leaf [default=20]: 리프 노드가 되기 위해서 최소한으로 필요한 레코드 수, 과적합 제어<br>결정 트리의 min_samples_leaf과 같음<br>사이킷런 클래스: min_child_samples
  - num_leaves [default=31]: 하나의 트리가 가질 수 있는 최대 리프 개수
  - boosting [default=gbdt]: 부스팅 트리 생성하는 알고리즘<br>gbdt: 일반적 그래디언트 부스팅 결정 트리 / rf: 랜덤 포레스트
  - bagging_fraction [default=1.0]: 데이터 샘플링 비율 지정, 과적합 제어<br>사이킷런 클래스: 동일
  - feature_fraction [default=1.0]: 개별 트리 학습할 때마다 무작위로 선택하는 피처 비율<br>사이킷런 클래스: 동일
  - lambda_l2 [default=0.0]: L2 regulation 제어, 과적합 제어(피처 많을 때 적용)<br>사이킷런 클래스: reg_lambda
  - lambda_l1 [default=0.0]: L1 regulation 제어, 과적합 제어(피처 많을 때 적용)

사이킷런 클래스: reg_alpha

- Learning Task 파라미터
  - objective: 최솟값 가져야 할 손실 함수 정의

### 하이퍼 파라미터 튜닝 방안

- 기본 튜닝 방안: 모델 복잡도 줄이기
  - num_leaves 개수 중심으로 min_child_samples, max_depth 조절
  - num_leaves 증가→정확↑, 깊이↑, 과적합 영향↑
  - min_child_samples 증가→트리 깊어지는 것 방지
  - max_depth 명시적으로 트리 깊이 제한
- 부스팅 계열에서 기본 튜닝: n_estimators↑ learning_rate↓ (과적합 주의)
  그 외: regularization 적용, colsample_bytree, subsample 파라미터 적용

## 08 베이지안 최적화 기반의 HyperOpt 이용 하이퍼 파라미터 튜닝

XGBoost, LightGBM의 경우 하이퍼 파라미터 개수가 많음 ⇒ Grid Search 방식의 최적화 수행 시간↑ ⇒ 베이지안 최적화 기법

### 베이지안 최적화 개요

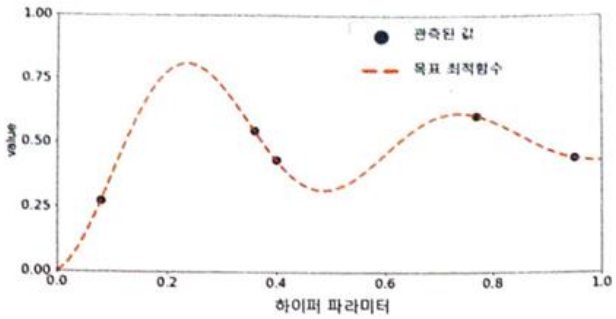
목적 함수 식을 제대로 알 수 없는 블랙 박스 형태의 함수에서 최대/최소 함수 반환 값을 만드는 최적 입력값을 가능한 적은 시도를 통해 찾아주는 방식

- 베이지안 확률에 기반(베이즈 확률)
- 새로운 데이터 입력받았을 때 최적 함수 예측하는 사후 모델 개선해 나가며 최적 함수 모델 만듦

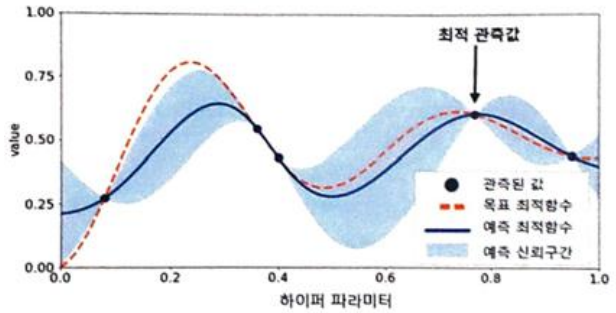
- 중요 구성 요소
  - 대체 모델(Surrogate Model): 획득 함수로부터 최적 입력 값 받음
    ⇒ 최적 함수 모델 개선
  - 획득 함수(Acquisition Function): 개선된 대체 모델 기반 ⇒ 최적 입력 값 계산
  - 두 개 왔다 갔다하면서 더 정확한 최적 입력 값 계산

- 튜닝에 사용될 때: 입력값 == 하이퍼 파라미터

- 베이지안 최적화 단계
  - 1 : 처음에 랜덤하게 하이퍼 파라미터들 샘플링, 성능 관측

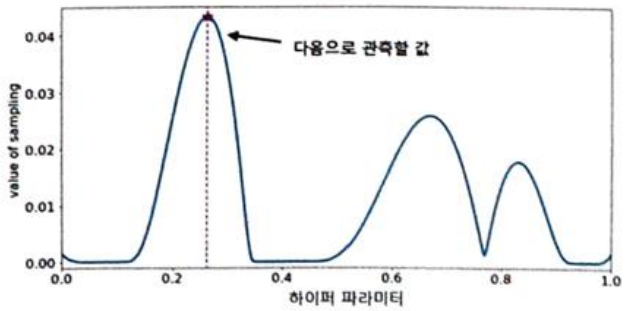



  - 2 : 대체 모델 - 관측 값 기반으로 최적 함수 추정



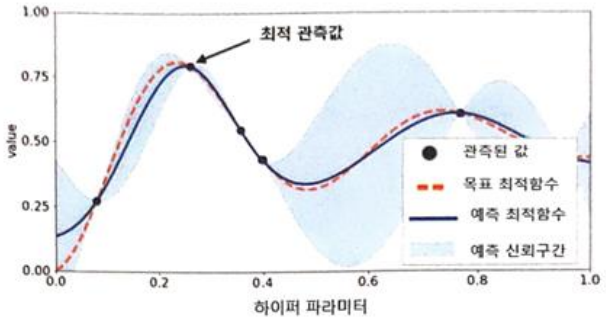

  - 3 : 획득 함수 - 추정된 최적 함수 기반으로 다음으로 관측할 하이퍼 파라미터 값 계산

    이전 최적 관측값보다 더 큰 최댓값을 가질 가능성이 높은 지점을 찾아 다음에 관측할 하이퍼 파라미터를 대체 모델에 전달




  - 4 : 대체 모델 - 갱신된 관측 값으로 최적 함수 추정

    3-4 반복 통해 대체 모델 불확실성 개선, 보다 정확한 최적 함수 추정




## HyperOpt 사용하기

- 주요 로직

1. 입력 변수명, 입력값의 검색 공간 설정
2. 목적 함수 설정
3. 목적 함수의 최솟값 가지는 최적 입력값 유추

- 입력값의 검색 공간 제공하는 함수
  - hp.quniform(label, low, high, q): low에서 high까지 q의 간격
  - hp.uniform(label, low, high): low에서 high까지 정규 분표 형태
  - hp.randint(label, upper): 0부터 upper까지 random한 정숫값
  - hp.loguniform(label, low, high): exp(uniform(low, high))값 반환, 반환 값의 log 변환된 값은 정규 분포 형태
  - hp.choice(label, options): 검색 값이 문자열/문자열+숫자값 일 경우 설정 Options: 리스트, 튜플 형태 ['gini', 'entropy'] 면 입력 변수 값을 각각 설정하여 입력

- fmin( ) 함수의 주요 인자
  - fn: 목적 함수
  - space: 검색 공간 딕셔너리
  - algo: 베이지안 최적화 적용 알고리즘 기본적으로 tpe.suggest 입력 (Tree of Parzen Estomator)
  - max_evals: 최적 입력값 찾기 위한 입력값 시도 횟수

- trials: 최적 입력값 탐색 시도한 입력값과 해당 입력값의 목적 함수 반환값 결과 저장 Trials 클래스를 객체로 생성한 변수명 입력
- rstate: fmin() 수행시 동일한 결괏값을 가지도록 설정하는 랜덤 시드 값 일반적으로 잘 적용X

- Trials 객체 주요 속성
  - results: 함수 반복 수행시마다 반환되는 반환값 가짐
    파이썬 리스트 형태, 개별 원소 딕셔너리 {'loss': 함수 반환값, 'status': 반환 상태값}
  - vals: 함수 반복 수행시마다 입력되는 입력 변숫값 가짐
    딕셔너리 형태, {'입력변수명': 개별 수행 시마다 입력된 값의 리스트}

**HyperOpt 이용 XGBoost 하이퍼 파라미터 최적화**

1. 적용해야 할 하이퍼 파라미터, 검색 공간 설정
2. 목적 함수에서 XGBoost 학습 후, 예측 성능 결과 반환
3. fmin() 함수에서 목적 함수를 하이퍼 파라미터 검색 공간의 입력값들 사용해 최적 예측 성능 결과 반환하는 최적 입력값 결정

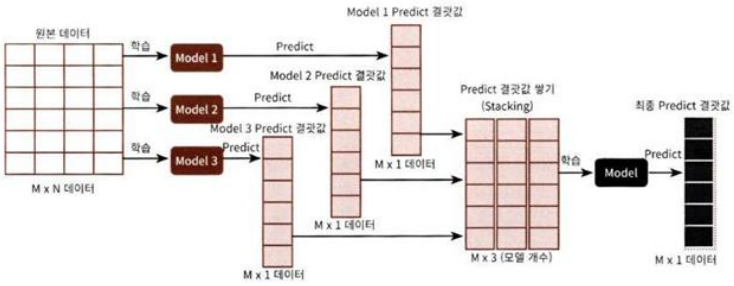
- 주의점
  - 하이퍼 파라미터 형변환 : 특정 하이퍼 파라미터 입력은 정수형, but HyperOpt는 입력값과 반환 값이 모두 실수형
  - -1 곱하기: 성능 값이 클수록 좋은 성능 지표일 경우에 -1 곱함 HyperOpt의 목적 함수는 최솟값을 반환할 수 있도록 최적화해야 하므로.

## 10 스태킹 앙상블

- 배깅, 부스팅과의 공통점: 여러 알고리즘 서로 결합해 예측 결과 도출
- 차이점: 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측 수행
- 스태킹 모델
  - 개별적 기반 모델 (많이 필요)
  - 최종 메타 모델: 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어 학습

⇒ 여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습용 / 테스

트용 피처 데이터 세트 만듦

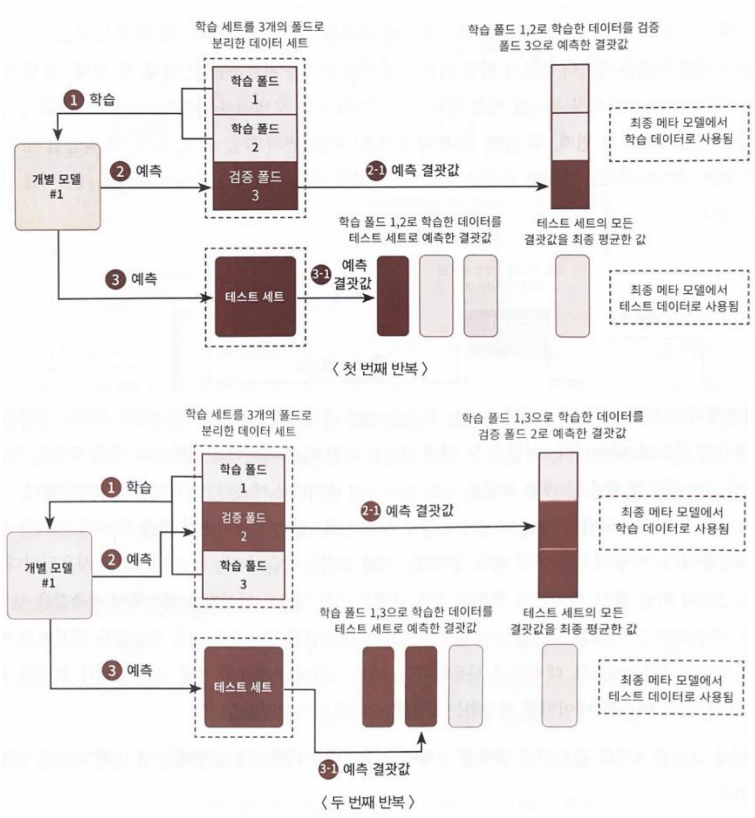


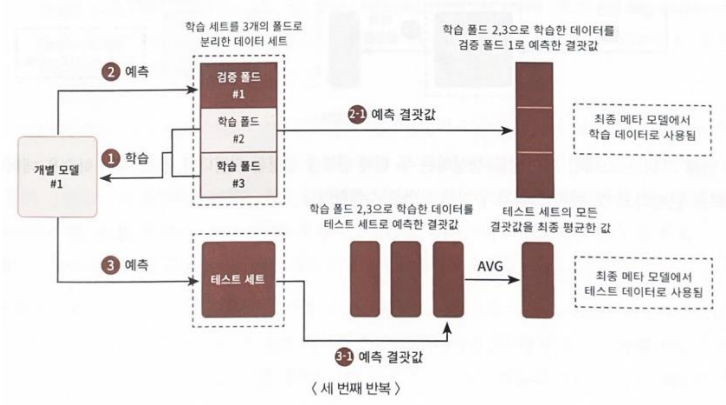

## CV 세트 기반의 스태킹

과적합 개선 위해 이용

⇒ 개별 모델들이 각각 교차 검증으로 학습용/테스트용 스태킹 데이터 생성
⇒ 기반으로 메타 모델이 학습, 테스트용 스태킹 데이터 세트를 예측,
원본 테스트 데이터이 레이블 데이터 기반 평가

1. 3개의 폴드 세트로 가정 시, 개별 모델이 3번 유사한 반복 작업 수행
   마지막 반복에서 개별 모델의 예측값으로 학습/테스트 데이터 생성






위 그림처럼 폴드별 예측 데이터를 합해 메타 모델 학습 데이터 만듦
개별 모델이 원본 테스트 세트로 예측한 결괏값 최종 평균해 메타 모델 테스트 데이터 만듦

2. 개별 모델들의 학습/테스트 데이터 합쳐 메타 모델 사용할 학습/테스트 데이터 생성