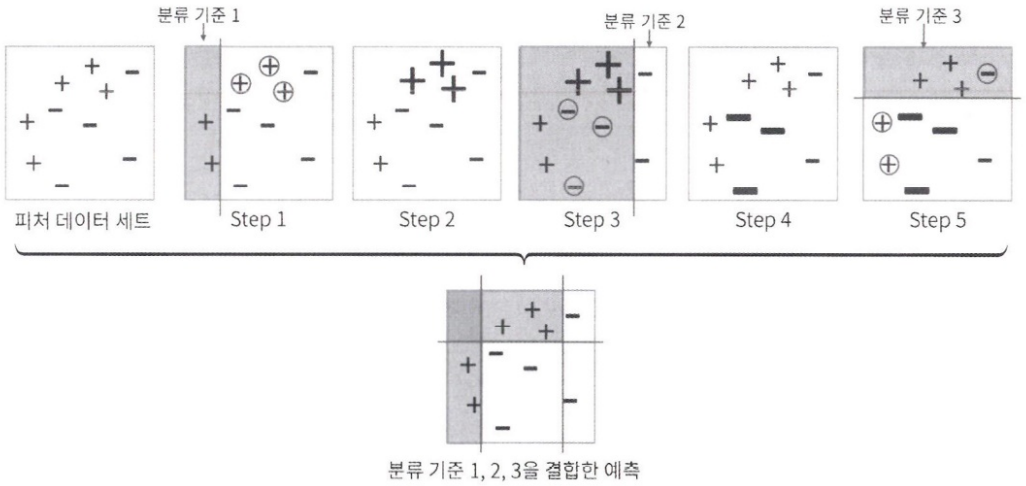
Week4_예습과제_우정연

4.5 GBM(Gradient Boosting Machine)

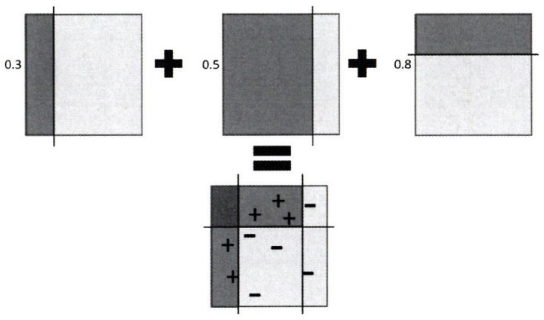
[GBM의 개용 및 실습]

- 부스팅 알고리즘
  - 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가며 학습하는 방식
  - 에이다 부스트(AdaBoost = Adaptive boosting), 그래디언트 부스트
- 에이다 부스트 (AdaBoost)
  - 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 대표적인 알고리즘



- Step 1은 첫 번째 약한 학습기(weak learner)가 분류 기준 1로 +와 -를 분류한 것입니다. 동그라미로 표시된 ⊕ 데이터는 + 데이터가 잘못 분류된 오류 데이터입니다.
- Step 2에서는 이 오류 데이터에 대해서 가중치 값을 부여합니다. 가중치가 부여된 오류 + 데이터는 다음 약한 학습기가 더 잘 분류할 수 있게 크기가 커졌습니다.
- Step 3은 두 번째 약한 학습기가 분류 기준 2로 +와 -를 분류했습니다. 마찬가지로 동그라미로 표시된 ⊖ 데이터는 잘못 분류된 오류 데이터입니다.
- Step 4에서는 잘못 분류된 이 - 오류 데이터에 대해 다음 약한 학습기가 잘 분류할 수 있게 더 큰 가중치를 부여합니다 (오류 - 데이터의 크기가 커졌습니다).
- Step 5는 세 번째 약한 학습기가 분류 기준 3으로 +와 -를 분류하고 오류 데이터를 찾습니다. 에이다부스트는 이렇게 약한 학습기가 순차적으로 오류 값에 대해 가중치를 부여한 예측 결정 기준을 모두 결합해 예측을 수행합니다.
- 마지막으로 맨 아래에는 첫 번째, 두 번째, 세 번째 약한 학습기를 모두 결합한 결과 예측입니다. 개별 약한 학습기보다 훨씬 정확도가 높아졌음을 알 수 있습니다.

  - 개별 약한 학습기는 각각 가중치를 부여해 결합



- GBM(Gradient Boost Machine)
  - 에이다부스트와 유사
  - 가중치 업데이트를 경사 하강법(Gradient Descent)을 이용
    - 반복 수행을 통해 오류를 최소화할 수 있도록 가중치의 업데이트 값을 도출하는 기법
    - 오류식 h(x) = y - F(x) 을 최소화하는 방향성을 가지고 반족적으로 가중치 값을 업데이트
      - 오류 값 = 실제 값 - 예측 값
      - 분류의 실제 결과값 = y
      - 피처 = x1, x2, x3...xn
      - 피처에 기반한 예측 함수 = F(x)
  - CART 기반의 다른 알고리즘과 마찬가지로 분류, 회귀 모두 가능

    - GradientBoostingClassifier 클래스
        - 사이킷런 - GBM 기반의 분류를 위해 제공
- 사이킷런의 GradientBoostingClassifier 실습
    - 일반적으로 GBM이 랜덤 포레스트보다는 예측 성능이 조금 더 좋음
    - But 수행시간이 오래 걸리고, 하이퍼 파라미터 튜닝 노력 더 필요
        - 약한 학습기의 순차적인 예측 오류 보정을 통해 학습을 수행 → 병렬 처리가 지원되지 않음 → 대용량 데이터의 경우 학습에 매우 많은 시간이 필요
        - 랜덤 포레스트의 경우 - 상대적으로 빠른 수행 시간 보장

[GBM 하이퍼 파라미터 및 튜닝]

- loss: 경사 하강법에서 사용할 비용 함수를 지정합니다. 특별한 이유가 없으면 기본값인 'deviance'를 그대로 적용합니다.
- learning_rate: GBM이 학습을 진행할 때마다 적용하는 학습률입니다. Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수입니다. 0~1 사이의 값을 지정할 수 있으며 기본값은 0.1입니다. 너무 작은 값을 적용하면 업데이트되는 값이 작아져서 최소 오류 값을 찾아 예측 성능이 높아질 가능성이 높습니다. 하지만 많은 weak learner는 순차적인 반복이 필요해서 수행 시간이 오래 걸리고, 또 너무 작게 설정하면 모든 weak learner의 반복이 완료돼도 최소 오류 값을 찾지 못할 수 있습니다. 반대로 큰 값을 적용하면 최소 오류 값을 찾지 못하고 그냥 지나쳐 버려 예측 성능이 떨어질 가능성이 높아지지만, 빠른 수행이 가능합니다.

  이러한 특성 때문에 learning_rate는 n_estimators와 상호 보완적으로 조합해 사용합니다. learning_rate를 작게 하고 n_estimators를 크게 하면 더 이상 성능이 좋아지지 않는 한계점까지는 예측 성능이 조금씩 좋아질 수 있습니다. 하지만 수행 시간이 너무 오래 걸리는 단점이 있으며, 예측 성능 역시 현격히 좋아지지는 않습니다.
- n_estimators: weak learner의 개수입니다. weak learner가 순차적으로 오류를 보정하므로 개수가 많을수록 예측 성능이 일정 수준까지는 좋아질 수 있습니다. 하지만 개수가 많을수록 수행 시간이 오래 걸립니다. 기본값은 100입니다.
- subsample: weak learner가 학습에 사용하는 데이터의 샘플링 비율입니다. 기본값은 1이며, 이는 전체 학습 데이터를 기반으로 학습한다는 의미입니다(0.5이면 학습 데이터의 50%). 과적합이 염려되는 경우 subsample을 1보다 작은 값으로 설정합니다.

- GridSearchCV를 이용해 하이퍼 파라미터 최적화
- GBM은 과적합에도 강한 뛰어난 예측 성능을 가진 알고리즘
- 수행 시간이 오래 걸린다는 단점 → 그래디언트 부스팅 기반 ML 패키지 XGBoost, LightGBM

4.6 XGBoost(eXtra Gradient Boost)

[XGBoost 개요]

- XGBoost 장점

- 트리 기반 앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나
- GBM의 단점인 느린 수행 시간 및 과적합 규제(Regularization) 부재 등의 문제를 해결한 GBM 기반 머신러닝
- 병렬 CPU 환경에서 병렬 학습 가능 → 기존 GBM보다 빠르게 학습 완료 가능

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.

- XGBoost의 파이썬 패키지명 'xgboost'
  - XGBoost 전용의 파이썬 패키지 ⇒ 파이썬 래퍼 XGBoost 모듈
    - 고유의 API와 하이퍼 파라미터 이용
  - 사이킷런 호환 래퍼용 XGBoost ⇒ 사이킷런 래퍼 XGBoost 모듈
    - XGBClassifier, XGBRegressor
    - 사이킷런 estimator가 학습을 위해 사용하는 fit(), predict()와 같은 표준 사이킷런 개발 프로세스 및 유틸리티 활용 가능

[파이썬 래퍼 XGBoost 하이퍼 파라미터]

- XGBoost는 GBM과 유사한 하이퍼 파라미터 + 조기 중단(early stopping), 과적합 규제
- 파이썬 래퍼 XGBoost 하이퍼 파라미터

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

주요 일반 파라미터

- booster: gbtree(tree based model) 또는 gblinear(linear model) 선택. 디폴트는 gbtree입니다.
- silent: 디폴트는 0이며, 출력 메시지를 나타내고 싶지 않을 경우 1로 설정합니다.
- nthread: CPU의 실행 스레드 개수를 조정하며, 디폴트는 CPU의 전체 스레드를 다 사용하는 것입니다. 멀티 코어/스레드 CPU 시스템에서 전체 CPU를 사용하지 않고 일부 CPU만 사용해 ML 애플리케이션을 구동하는 경우에 변경합니다.

주요 부스터 파라미터

- eta [default=0.3, alias: learning_rate]: GBM의 학습률(learning rate)과 같은 파라미터입니다. 0에서 1 사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값. 파이썬 래퍼 기반의 xgboost를 이용할 경우 디폴트는 0.3. 사이킷런 래퍼 클래스를 이용할 경우 eta는 learning_rate 파라미터로 대체되며, 디폴트는 0.1입니다. 보통은 0.01 ~ 0.2 사이의 값을 선호합니다.
- num_boost_rounds: GBM의 n_estimators와 같은 파라미터입니다.
- min_child_weight[default=1]: 트리에서 추가적으로 가지를 나눌지를 결정하기 위해 필요한 데이터들의 weight 총합. min_child_weight이 클수록 분할을 자제합니다. 과적합을 조절하기 위해 사용됩니다.
- gamma [default=0, alias: min_split_loss]: 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값입니다. 해당 값보다 큰 손실(loss)이 감소된 경우에 리프 노드를 분리합니다. 값이 클수록 과적합 감소 효과가 있습니다.
- max_depth[default=6]: 트리 기반 알고리즘의 max_depth와 같습니다. 0을 지정하면 깊이에 제한이 없습니다. Max_depth가 높으면 특정 피처 조건에 특화되어 룰 조건이 만들어지므로 과적합 가능성이 높아지며 보통은 3~10 사이의 값을 적용합니다.
- sub_sample[default=1]: GBM의 subsample과 동일합니다. 트리가 커져서 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정합니다. sub_sample=0.5로 지정하면 전체 데이터의 절반을 트리를 생성하는 데 사용합니다. 0에서 1사이의 값이 가능하나 일반적으로 0.5 ~ 1 사이의 값을 사용합니다.
- colsample_bytree[default=1]: GBM의 max_features와 유사합니다. 트리 생성에 필요한 피처(칼럼)를 임의로 샘플링하는 데 사용됩니다. 매우 많은 피처가 있는 경우 과적합을 조정하는 데 적용합니다.
- lambda [default=1, alias: reg_lambda]: L2 Regularization 적용 값입니다. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있습니다.
- alpha [default=0, alias: reg_alpha]: L1 Regularization 적용 값입니다. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있습니다.
- scale_pos_weight [default=1]: 특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형을 유지하기 위한 파라미터입니다.

학습 태스크 파라미터

- objective: 최솟값을 가져야할 손실 함수를 정의합니다. XGBoost는 많은 유형의 손실함수를 사용할 수 있습니다. 주로 사용되는 손실함수는 이진 분류인지 다중 분류인지에 따라 달라집니다.
- binary:logistic: 이진 분류일 때 적용합니다.
- multi:softmax: 다중 분류일 때 적용합니다. 손실함수가 multi:softmax일 경우에는 레이블 클래스의 개수인 num_class 파라미터를 지정해야 합니다.
- multi:softprob: multi:softmax와 유사하나 개별 레이블 클래스의 해당되는 예측 확률을 반환합니다.
- eval_metric: 검증에 사용되는 함수를 정의합니다. 기본값은 회귀인 경우는 rmse, 분류일 경우에는 error입니다. 다음은 eval_metric의 값 유형입니다.
  - rmse: Root Mean Square Error
  - mae: Mean Absolute Error
  - logloss: Negative log-likelihood
  - error: Binary classification error rate (0.5 threshold)
  - merror: Multiclass classification error rate
  - mlogloss: Multiclass logloss
  - auc: Area under the curve

    - 뛰어난 알고리즘일수록 파라미터를 튜닝할 필요 없음
    - 파라미터 튜닝에 들이는 공수 대비 성능 향상 효과가 높지 않은 경우가 대부분
- 과적합 문제가 심각할 경우

  - eta 값을 낮춥니다(0.01 ~ 0.1). eta 값을 낮출 경우 num_round(또는 n_estimators)는 반대로 높여줘야 합니다.
  - max_depth 값을 낮춥니다.
  - min_child_weight 값을 높입니다.
  - gamma 값을 높입니다.
  - 또한 subsample과 colsample_bytree를 조정하는 것도 트리가 너무 복잡하게 생성되는 것을 막아 과적합 문제에 도움이 될 수 있습니다.

- XGBoost 자체적으로 교차 검증, 성능 평가, 피처 중요도 등의 시각화 기능을 가짐
- 기본 GBM
  - n_estimators(or num_boost_rounds)에 지정된 횟수만큼 반복적으로 학습 오류를 감소시키며 학습을 진행 - 중간에 반복을 멈출 수 없음
- XGBoost, LightGBM

    - 조기 중단 기능이 있어 n_estimators에 지정한 부스팅 반복 횟수에 도달하지 않더라도 예측 오류가 더 이상 개선되지 않으면 반복을 끝까지 수행하지 않고 중지 → 수행시간 개선

[파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측]

- XGBoost의 파이썬 패키지인 xgboost
    - 자체적으로 교차 검증, 성능 평가, 피처 중요도 등의 시각화(plotting) 기능 가짐
    - 병렬 처리, 조기 중단 기능 → 수행 시간 개선 기능
- 위스콘신 유방암 데이터 세트
    - 종양의 크기, 모양 등의 다양한 속성값을 기반으로 악성 종양(malignant)인지 양성 종양(benign)인지 분류한 데이터 세트
- load_breast_cancer( )
    - 사이킷런에 내장된 데이터 세트 호출
    - 종양 크기와 모양에 관련된 속성이 숫자형 값으로 되어 있음
    - 타깃 레이블: malignant=0, benign=1
- 사이킷런과 파이썬 래퍼 XGBoost의 차이
    - 학습용과 테스트용 데이터 세트를 위해 별도의 객체인 DMatrix를 생성함
    - DMatrix
        - 넘파이 입력 파라미터를 받아 만들어지는 XGBoost 전용 데이터 세트
        - 주요 입력 파라미터: data, label
        - 넘파이 외 libsvm txt 포맷 파일, xgboost 이진 버퍼 파일을 파라미터로 입력받아 변환 가능
        - 판다스의 DataFrame으로 데이터 인터페이스 위해 DataFrame.values를 이용해 넘파이로 일차 변환한 뒤 이를 이용해 DMatrix 변환 적용
    - xgboost 모듈의 train( ) 함수에 파라미터로 하이퍼 파라미터를 전달
        - 사이킷런의 경우: Estimator의 생성자를 하이퍼 파라미터로 전달
- XGBoost 하이퍼 파라미터 (주로 딕셔너리 형태로 입력)
- 조기 중단
    - XGBoost가 수행 성능을 개선하기 위해 더 이상 지표 개선이 없을 경우 num_boost_round 횟수를 모두 채우지 않고 중간에 반복을 빠져 나올 수 있음

    - train( ) 함수에 early_stopping_rounds 파라미터를 입력하여 설정

        - eval_set는 성능 평가를 수행할 평가용 데이터 세트를 설정.
        - eval_metric은 평가 세트에 적용할 성능 평가 방법. 분류일 경우 주로 'error'(분류 오류), 'logloss'를 적용.

        - train( )은 학습이 완료된 모델 객체를 반환
        - 반복 시 train-error와 eval-logloss가 지속적으로 감소
- predict( ) 메서드를 이용해 모델 객체 예측
    - 예측 결과값이 아닌 예측 결과를 추정할 수 있는 확률 값 반환 (vs 사이킷런: 예측 결과 클래스 값 반환)
    - 예측 확률에 따른 예측 값을 결정하는 로직을 추가
- xgboost 패키지에 내장된 시각화 기능
    - plot_importance( ) API
        - 피처의 중요도를 막대그래프 형식으로 나타냄
        - 기본 평가 지표로 f1 스코어 기반으로 각 피처의 중요도 나타냄
        - 파라미터로 학습이 완료된 모델 객체 및 맷플롭립의 ax 객체를 입력
        - vs. 사이킷런은 Estimator 객제의 feature_importances_ 속성을 이용해 직접 시각화 코드 작성해야
        - 넘파이 기반의 피처 데이터로 학습 시에 피처명을 제대로 알 수 없으므로 f+순서로 피처 나열
    - xgboost.to_graphviz( ) API
        - 트리 기반 규칙 구조 시각화 가능
        - 파라미터로 학습이 완료된 모델 객체와 Graphviz가 참조할 파일명 입력
    - cv( ) API
        - 데이터 세트에 대한 교차 검증 수행 후 최적 파라미터를 구할 수 있는 방법 제공

```
xgboost.cv(params, dtrain, num_boost_round=10, nfold=3, stratified=False, folds=None, metrics=(),
obj=None, feval=None, maximize=False, early_stopping_rounds=None, fpreproc=None, as_pandas=True,
verbose_eval=None, show_stdv=True, seed=0, callbacks=None, shuffle=True)
```

- 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 strings): CV 수행 시 모니터링할 성능 평가 지표
- early_stopping_rounds (int): 조기 중단을 활성화시킴. 반복 횟수 지정.

    - 반환값은 DataFrame 형태

[사이킷런 래퍼 XGBoost 개요 및 적용 형태]

사이킷런 전용의 XGBoost 래퍼 클래스 → 사이킷런 기본 Estimator를 그대로 상속

- 다른 Estimator와 동일하게 fit( )과 predict( ), predict_proba( )만으로 학습과 예측 가능
- GridSearchCV, Pipeline등 사이킷런 다른 유틸리티 그대로 사용 가능
- 분류를 위한 XGBClassifier, 회귀를 위한 XGBRegressor
- xgboost 모듈에서 사용하던 네이티브 하이퍼 파라미터의 변경

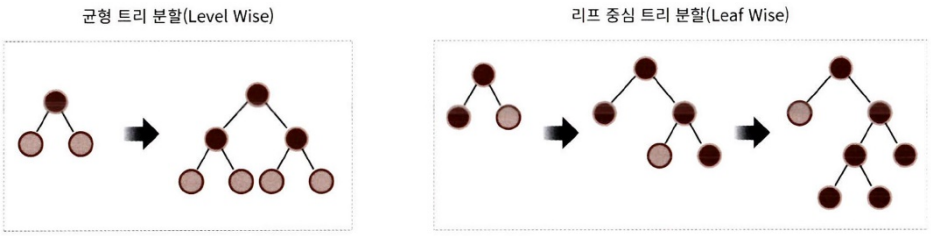
    - eta → learning_rate
    - sub_sample → subsample
    - lambda → reg_lambda
    - alpha → reg_alpha

- xgboost의 n_estimators = 사이킷런의 num_boost_round
- 조기 중단
    - 조기 중단 관련 파라미터를 fit( )에 입력하면 됨
    - eearly_stopping_rounds: 평가 지표가 향상될 수 있는 반복 횟수 정의
    - eval_metric: 조기 중단을 위한 평가 지표
    - eval_set: 성능 평가를 수행할 데이터 세트
    - 성능 평가를 수행할 데이터 세트는 학습 데이터가 아니라 별도의 데이터 세트여야 함

    - 조기 중단 값을 너무 급격하게 줄이면 예측 성능이 저하될 우려가 큼
- plot_importance( ) API
    - 피처의 중요도 시각화

4.7 LightGBM

- 장점
    - XGBoost보다 학습에 걸리는 시간이 훨씬 적음
    - 메모리 사용량도 상대적으로 적음
    - XGBoost 와 예측 성능은 별다른 차이가 없음
    - 기능상의 다양성이 약간 더 많음
    - 카테고리형 피처의 자동 변환과 최적 분할 (원-핫 인코딩 등을 사용하지 않고도 카테고리형 피처를 최적으로 변환하고 이에 따른 노드 분할 수행)
    - 대용량 데이터에 대한 뛰어난 예측 성능 및 병렬 컴퓨팅 기능을 제공, GPU 지원
- 단점
    - 적은 데이터 세트(10000건 이하)에 적용할 경우 과적합이 발생하기 쉬움
- 리프 중심 트리 분할 (Leaf Wise) 방식 사용
    - 기존 대부분의 트리 기반 알고리즘: 균형 트리 분할(Level Wise) 방식 사용
        - 트리의 깊이를 효과적으로 줄이기 위해
        - 최대한 균형 잡힌 트리를 유지하면서 분할 → 트리의 깊이가 최소화될 수 있음
        - 오버피팅에 보다 더 강한 구조를 가질 수 있다고 알려져 있음
        - but 균형을 맞추기 위한 시간이 필요하다는 상대적 단점이 있음
    - LightGBM의 경우
        - 트리의 균형을 맞추지 않고 최대 손실 값 (max data loss)을 가지는 리프 노드를 지속적으로 분할 → 트리 깊이가 깊어지고 비대칭적인 트리 생성
        - 균형 트리 분할 방식보다 예측 오류 손실을 최소화 할 수 있음



- LightGBM의 파이썬 패키지명: 'lightgbm'
  - 분류를 위한 LGBMClassifier, 회귀를 위한 LGBMRegressor 클래스

[LightGBM 하이퍼 파라미터]

- 리프 중심 트리 분할 특성에 맞는 하이퍼 파라미터 설정이 필요

주요 파라미터

- num_iterations [default = 100]: 반복 수행하려는 트리의 개수를 지정합니다. 크게 지정할수록 예측 성능이 높아질 수 있으나, 너무 크게 지정하면 오히려 과적합으로 성능이 저하될 수 있습니다. 사이킷런 GBM과 XGBoost의 사이킷런 호환 클래스의 n_estimators와 같은 파라미터이므로 LightGBM의 사이킷런 호환 클래스에서는 n_estimators로 이름이 변경됩니다.

- learning_rate [default = 0.1]: 0에서 1사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값입니다. 일반적으로 n_estimators를 크게 하고 learning_rate를 작게 해서 예측 성능을 향상시킬 수 있으나, 마찬가지로 과적합 이슈와 학습 시간이 길어지는 부정적인 영향도 고려해야 합니다. GBM, XGBoost의 learning rate와 같은 파라미터입니다.
- max_depth [default=-1]: 트리 기반 알고리즘의 max_depth와 같습니다. 0보다 작은 값을 지정하면 깊이에 제한이 없습니다. 지금까지 소개한 Depth wise 방식의 트리와 다르게 LightGBM은 Leaf wise 기반이므로 깊이가 상대적으로 더 깊습니다.
- min_data_in_leaf [default = 20]: 결정 트리의 min_samples_leaf와 같은 파라미터입니다. 하지만 사이킷런 래퍼 LightGBM 클래스인 LightGBMClassifier에서는 min_child_samples 파라미터로 이름이 변경됩니다. 최종 결정 클래스인 리프 노드가 되기 위해서 최소한으로 필요한 레코드 수이며, 과적합을 제어하기 위한 파라미터입니다.
- num_leaves [default = 31]: 하나의 트리가 가질 수 있는 최대 리프 개수입니다.
- boosting [default = gbdt]: 부스팅의 트리를 생성하는 알고리즘을 기술합니다.
  - gbdt: 일반적인 그래디언트 부스팅 결정 트리
  - rf: 랜덤 포레스트
- bagging_fraction [default = 1.0]: 트리가 커져서 과적합되는 것을 제어하기 위해서 데이터를 샘플링하는 비율을 지정합니다. 사이킷런의 GBM과 XGBClassifier의 subsample 파라미터와 동일하기에 사이킷런 래퍼 LightGBM인 LightGBMClassifier에서는 subsample로 동일하게 파라미터 이름이 변경됩니다.
- feature_fraction [default = 1.0]: 개별 트리를 학습할 때마다 무작위로 선택하는 피처의 비율입니다. 과적합을 막기 위해 사용됩니다. GBM의 max_features와 유사하며, XGBClassifier의 colsample_bytree와 똑같으므로 LightGBM Classifier에서는 동일하게 colsample_bytree로 변경됩니다.
- lambda_l2 [default=0.0]: L2 regulation 제어를 위한 값입니다. 피처 개수가 많을 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있습니다. XGBClassifier의 reg_lambda와 동일하므로 LightGBMClassifier에서는 reg_lambda로 변경됩니다.
- lambda_l1 [default = 0.0]: L1 regulation 제어를 위한 값입니다. L2와 마찬가지로 과적합 제어를 위한 것이며, XGBClassifier의 reg_alpha와 동일하므로 LightGBMClassifier에서는 reg_alpha로 변경됩니다.

Learning Task 파라미터

- objective: 최솟값을 가져야 할 손실함수를 정의합니다. Xgboost의 objective 파라미터와 동일합니다. 애플리케이션 유형, 즉 회귀, 다중 클래스 분류, 이진 분류인지에 따라서 objective인 손실함수가 지정됩니다.

[하이퍼 파라미터 튜닝 방안]

- num_leaves 개수를 중심으로 min_child_samples(min_data_in_leaf), max_depth를 함께 조정하면서 모델의 복잡도를 줄이는 것이 기본 튜닝 방안

- num_leaves는 개별 트리가 가질 수 있는 최대 리프의 개수이고 LightGBM 모델의 복잡도를 제어하는 주요 파라미터입니다. 일반적으로 num_leaves의 개수를 높이면 정확도가 높아지지만, 반대로 트리의 깊이가 깊어지고 모델이 복잡도가 커져 과적합 영향도가 커집니다.
- min_data_in_leaf는 사이킷런 래퍼 클래스에서는 min_child_samples로 이름이 바뀝니다. 과적합을 개선하기 위한 중요한 파라미터입니다. num_leaves와 학습 데이터의 크기에 따라 달라지지만, 보통 큰 값으로 설정하면 트리가 깊어지는 것을 방지합니다.
- max_depth는 명시적으로 깊이의 크기를 제한합니다. num_leaves, min_data_in_leaf와 결합해 과적합을 개선하는 데 사용합니다.

- learning_rate를 작게 하면서 n_estimators를 크게 하는 것은 부스팅 계열 튜닝에서 가장 기본적 튜닝 방안 → 이를 적용 가능
  - but n_estimators를 너무 크게 하면 과적합 우려
- 과적합 제어
  - reg_lambda, reg_alpha와 같은 regularization 적용
  - 학습 데이터에 사용할 피처의 개수나 데이터 샘플링 레코드 개수를 줄이기 위해 colsample_bytree, subsample 파라미터를 적용 가능

[파이썬 래퍼 LightGBM과 사이킷런 래퍼 XGBoost, LightGBM 하이퍼 파라미터 비교]

- 사이킷런 래퍼 LightGBM 클래스와 사이킷런 래퍼 XGBoost 클래스는 많은 하이퍼 파라미터 동일

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

[LightGBM 적용 - 위스콘신 유방암 예측]

- from lightgbm import LGBMClassifier
- 조기 중단 가능

    - LGBMClassifier의 fit( )에 조기 중단 관련 파라미터 설정
- plot_importance( )
    - 피처 중요도 시각화 내장 API

4.8 베이지안 최적화 기반의 HyperOpt를 이용한 하이퍼 파라미터 튜닝 (파머완 개정 2판)

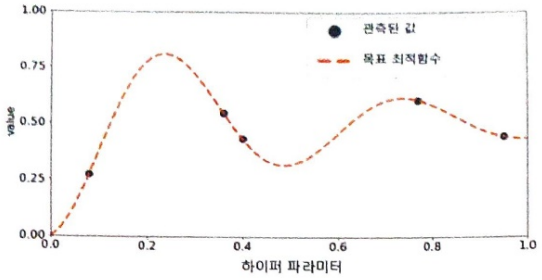
- 하이퍼 파라미터 튜닝을 위한 사이킷런 제공 GridSearch 방식 → 튜닝해야 할 하이퍼 파라미터 개수 많을 경우 최적화 수행 시간 오래 걸림
- XGBoost, LightGBM
    - 성능이 좋지만, 하이퍼 파라미터 개수가 다른 알고리즘에 비해 많음
    - 실무의 대용량 학습 데이터에 Grid Search 방식으로 최적 하이퍼 파라미터를 찾으려면 많은 시간이 소모됨
    - 하이퍼 파라미터 개수를 줄이거나 개별 하이퍼 파라미터의 범위를 줄여야 → 조금이라도 모델 성능 향상 필요할 경우 아쉬움

[베이지안 최적화 개요]

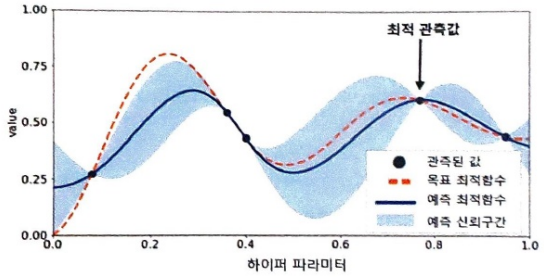
- 베이지안 최적화
    - 목적 함수 식을 제대로 알 수 없는 블랙 박스 형태의 함수에서 최대 또는 최소 함수 반환 값을 만드는 최적 입력값을 가능한 적은 시도를 통해 빠르고 효과적으로 찾아주는 방식
    - 함수 식 자체를 알 수 없고 입력값과 반환값만 알 수 있는 상황에서 함수 반환값의 최대/최소값을 찾기는 어려움 → 베이지안 최적화 이용 ⇒ 쉽고 빠르게 최적 입력값 찾을 수 있음
    - 베이지안 확률에 기반을 둔 최적화 기법
        - 새로운 데이터를 입력받았을 때 최적 함수를 예측하는 사후 모델을 개선해 나가며 최적 함수 모델 만듬
- 베이지안 최적화를 구성하는 두 중요요소
    - 대체 모델(Surrogate Model)
        - 획득 함수로부터 최적 함수를 예측할 수 있는 입력값을 추천 받은 뒤 이를 기반으로 최적 함수 모델 개선
    - 획득 함수(Acquisition Function)

        - 개선된 대체 모델을 기반으로 최적 입력값을 계산
- 하이퍼 파라미터 튜닝에 사용 시
    - 입력값 = 하이퍼 파라미터
    - 대체모델: 획득함수가 계산한 하이퍼 파라미터를 입력받으면서 점차 개선 → 개선된 대체 모델을 기반으로 획득 함수는 더 정확한 하이퍼 파라미터를 계산 가능
- 베이지안 최적화 단계

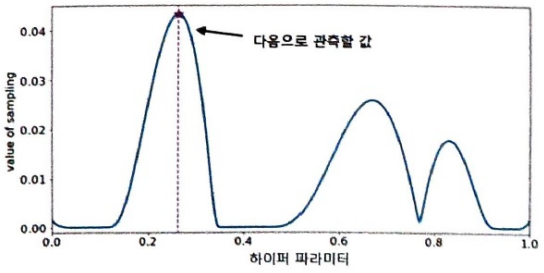
▪ Step 1: 최초에는 랜덤하게 하이퍼 파라미터들을 샘플링하고 성능 결과를 관측합니다. 아래 그림에서 검은색 원은 특정 하이퍼 파라미터가 입력되었을 때 관측된 성능 지표 결괏값을 뜻하며 주황색 사선은 찾아야 할 목표 최적함수 입니다.



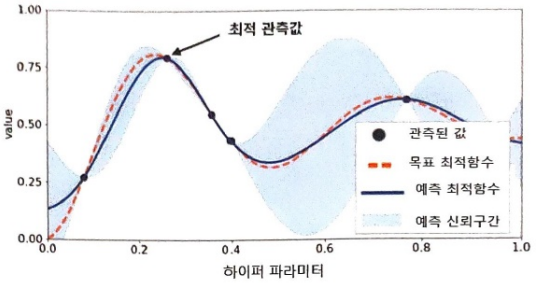
▪ Step 2: 관측된 값을 기반으로 대체 모델은 최적 함수를 추정합니다. 아래 그림에서 파란색 실선은 대체 모델이 추정한 최적 함수 입니다. 옅은 파란색으로 되어 있는 영역은 예측된 함수의 신뢰 구간입니다. 추정된 함수의 결괏값 오류 편차를 의미하며 추정 함수의 불확실성을 나타냅니다. 최적 관측값은 y축 value에서 가장 높은 값을 가질 때의 하이퍼 파라미터입니다.



▪ Step 3: 추정된 최적 함수를 기반으로 획득 함수(Acquisition Function)는 다음으로 관측할 하이퍼 파라미터 값을 계산합니다. 획득 함수는 이전의 최적 관측값보다 더 큰 최댓값을 가질 가능성이 높은 지점을 찾아서 다음에 관측할 하이퍼 파라미터를 대체 모델에 전달합니다.



- Step 4: 획득 함수로부터 전달된 하이퍼 파라미터를 수행하여 관측된 값을 기반으로 대체 모델은 갱신되어 다시 최적 함수를 예측 추정합니다.

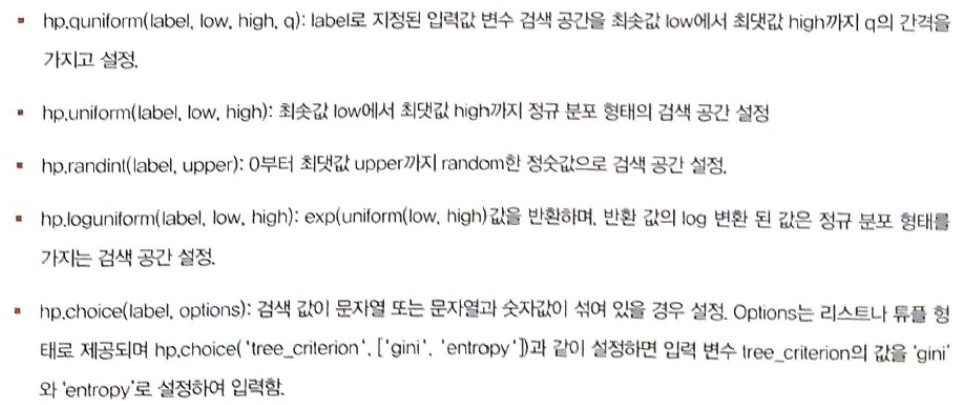


    - step3, step4 를 특정 횟수만큼 반복 → 대체 모델의 불확실성이 개선, 정확한 최적 함수 추정 가능
- 최적 함수 추정시 사용하는 알고리즘
    - 일반적으로 가우시안 프로세스(Gaussian Process) 적용
    - HyperOpt는 트리 파르젠 Estimator(TPE, Tree-structure Parzen Estimator) 사용

[HyperOpt 사용하기]

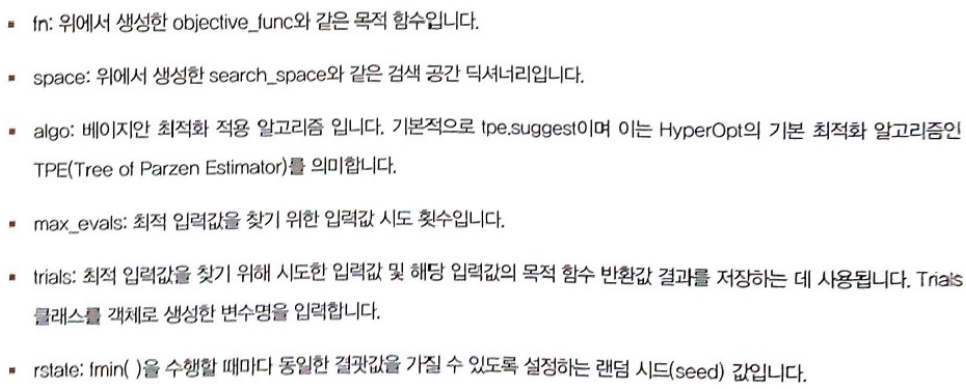
- 베이지안 최적화를 머신러닝 모델의 하이퍼 파라미터 튜닝에 적용할 수 있게 제공되는 패키지
    - HypterOpt, Bayesian, Optimization, Optuna 등
- HyperOpt 주요 로직
    1. 입력 변수명과 입력값의 검색공간(search space) 설정
    2. 목적 함수(objective function) 설정
    3. 목적 함수의 반환 최솟값을 가지는 최적 입력값 유추
- 다른 패키지와 다르게 목적 함수 반환 값의 최댓값이 아닌 최솟값을 가지는 최적 입력값을 유추함
- 입력 변수명과 입력값의 검색 공간은 파이썬 딕셔너리 형태로 설정되어야 함
- 키(key) 값으로 입력 변수면, 밸류(value) 값으로 해당 입력 변수 검색 공간이 주어짐
- hp.quniform('x', -10, 10, 1)
    - 입력 변수 x는 -10부터 10까지 1의 간격을 가지는 값들 (순차적으로 입력되지는 않음)
- 입력값의 검색 공간을 제공하는 대표적인 함수들

- 함수인자로 들어가는 label은 입력 변수명을 다시 적어줌
- low는 최솟값, high는 최댓값, q는 간격



- hp.quniform(label, low, high, q): label로 지정된 입력값 변수 검색 공간을 최솟값 low에서 최댓값 high까지 q의 간격을 가지고 설정.
- hp.uniform(label, low, high): 최솟값 low에서 최댓값 high까지 정규 분포 형태의 검색 공간 설정
- hp.randint(label, upper): 0부터 최댓값 upper까지 random한 정숫값으로 검색 공간 설정.
- hp.loguniform(label, low, high): exp(uniform(low, high)값을 반환하며, 반환 값의 log 변환 된 값은 정규 분포 형태를 가지는 검색 공간 설정.
- hp.choice(label, options): 검색 값이 문자열 또는 문자열과 숫자값이 섞여 있을 경우 설정. Options는 리스트나 튜플 형태로 제공되며 hp.choice('tree_criterion', ['gini', 'entropy'])과 같이 설정하면 입력 변수 tree_criterion의 값을 'gini'와 'entropy'로 설정하여 입력함.

- 목적 함수 생성
  - 반드시 변수값과 검색 공간을 갖는 딕셔너리를 인자로 받고, 특정 값을 반환하는 구조로 만들어져야 함
  - 딕셔너리 형태로 반환할 경우
    - {'loss': retval, 'status': STATUS_OK} 와 같이 loss와 status 키 값을 설정해서 반환해야 함
    - from hyperopt import STATUS_OK
- 목적 함수의 반환값이 최소가 될 수 있는 최적의 입력값을 베이지안 최적화 기법에 기반하여 찾아야 함
  - fmin(objective, space, algo, max_evals, trials) 함수



- fn: 위에서 생성한 objective_func와 같은 목적 함수입니다.
- space: 위에서 생성한 search_space와 같은 검색 공간 딕셔너리입니다.
- algo: 베이지안 최적화 적용 알고리즘 입니다. 기본적으로 tpe.suggest이며 이는 HyperOpt의 기본 최적화 알고리즘인 TPE(Tree of Parzen Estimator)를 의미합니다.
- max_evals: 최적 입력값을 찾기 위한 입력값 시도 횟수입니다.
- trials: 최적 입력값을 찾기 위해 시도한 입력값 및 해당 입력값의 목적 함수 반환값 결과를 저장하는 데 사용됩니다. Trials 클래스를 객체로 생성한 변수명을 입력합니다.
- rstate: fmin( )을 수행할 때마다 동일한 결괏값을 가질 수 있도록 설정하는 랜덤 시드(seed) 값입니다.

    - rstate 인자
      - 일반적으로 rstate를 잘 적용하지는 않음,
      - 일반적인 정수형 값을 넣지 않음

        - 버전별로 rstate인자값이 조금씩 다름
        - 넘파이의 random Generator를 생성하는 random.default_rng( ) 함수 이자로 seed 값 입력하는 방식
    - Trials 객체
        - 함수의 반복 수행 시마다 입력되는 변수값들과 반환값을 속성으로 가짐
        - 함수 수행 시마다 최적화되는 경과를 볼 수 있음
        - 중요 속성 - results, vals
            - result: 함수 반복 수행 시마다 반환되는 반환값
            - vals: 함수 반복 수해 시마다 입력되는 입력 변수값
- results 속성
    - 파이썬 리스트 형태
    - 리스트 내 개별 원소의 딕셔너리 모습
        - {'loss':함수 반환값, 'status':반환 상태값}
        - max_evals = 20 인 경우, results 속성은 20개의 딕셔너리를 개별 원소로 가지는 리스트로 구성
- vals 속성
    - 딕셔너리 형태
        - {'입력변수명': 개별 수행시마다 입력된 값의 리스트}
        - 입력 변수 x와 y를 키값으로 가짐
        - x와 y 키 값의 밸류는 20회의 반복 수행 시마다 사용되는 입력값들을 리스트 형태로 가짐
- 베이지안 최적화 방식으로 GridSearch 방식 보다 상대적으로 최적값을 찾는 시간을 많이 줄여줌

[HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화]

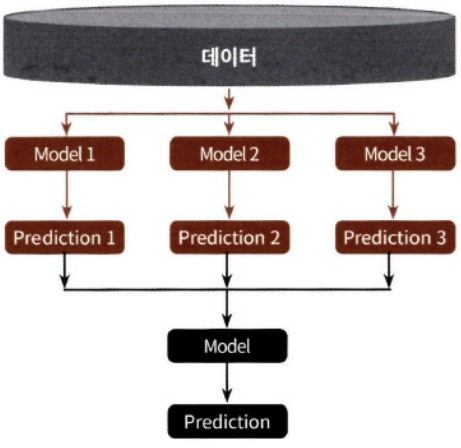
1. 적용해야 할 하이퍼 파라미터와 검색 공간 설정
2. 목적 함수에서 XGBoost를 학습 후에 예측 성능 결과를 반환 값으로 설정
3. fmin( ) 함수에서 목적 함수를 하이퍼 파라미터 검색 공간의 입력값들을 사용하여 최적의 예측 성능 결과를 반환하는 최적 입력값들을 결정

- 주의

    - 특정 하이퍼 파라미터 = 정수값만 입력 받음, HyperOpt는 입력값, 반환값이 모두 실수형
        - ⇒ 하이퍼 파라미터 입력시 형변환을 해야 함 int()
    - HyperOpt 목적 함수는 최솟값을 반환할 수 있도록 최적화 해야 함
        - ⇒ 성능 값이 클수록 좋은 성능 지표일 경우 -1을 곱해 줘야 함
        - but 회귀의 MAE, RMSE의 경우 성능 지표는 작을수록 좋기 때문에 반환 시 -1을 곱할 필요 없음
- cross_val_score( )를 XGBoost나 LightGBM에 적용할 경우 조기중단 지원 안됨
    - 조기 중단을 위해서는 KFold로 학습과 검증용 데이터 세트를 만들어 직접 교차 검증을 수행해야 함

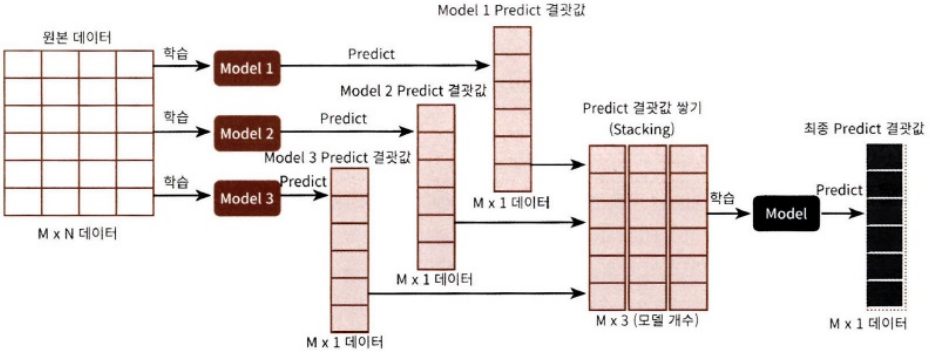
4.10 스태킹 앙상블

- 스태킹(Stacking)
    - 개별적인 여러 알고리즘을 서로 결합해 예측 결과 도출 (배깅, 부스팅과 공통점)
    - 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측 수행 (차이점)
    - 개별 알고리즘의 예측 결과 데이터 세트를 최종적인 메타 데이터 세트로 만들어 별도의 ML알고리즘으로 최종 학습을 수행 → 테스트 데이터를 기반으로 다시 최종 예측을 수행하는 방식 = 메타모델



- 개별적인 기반 모델과 개별 기반 모델의 예측데이터를 학습 데이터로 만들어 학습하는 최종 메타 모델 필요
- 여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습용 피처 데이터 세트와 테스트용 피처 데이터 세트를 만드는 것이 핵심

- 조금이라도 성능 수치 높여야 할 경우 자주 사용
- 많은 개별 모델 필요, 일반적으로 성능이 비슷한 모델을 결합해 좀 더 나은 성능 향상을 도출하기 위해 적용
- 스태킹 앙상블 모델 과정
  1. 모델 별로 각각 학습을 시킨 뒤 예측 수행 → 각각 M개의 로우를 가진 1개의 레이블 값 도출
  2. 모델별 도출된 예측 레이블 값을 다시 합(스태킹) → 새로운 데이터 세트 생성
  3. 스태킹된 데이터 세트에 대해 최종 모델을 적용해 최종 예측



[CV 세트 기반의 스태킹]

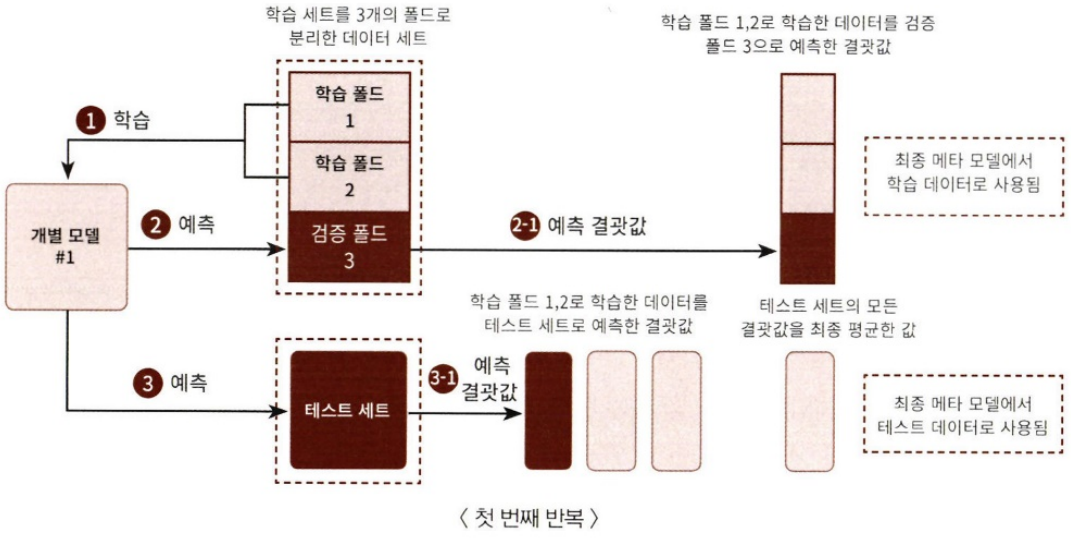
- CV 세트 기반의 스태킹 모델
  - 과적합을 개선하기 위해 최종 메타 모델을 위한 데이터 세트를 만들 때 교차 검증 기반으로 예측된 결과 데이터 세트를 이용
  - 개별 모델들이 각각 교차 검증으로 메타 모델을 위한 학습용 스태킹 데이터 생성과 예측을 위한 테스트용 스태킹 데이터를 생성 → 이를 기반으로 메타 모델이 학습과 예측을 수행

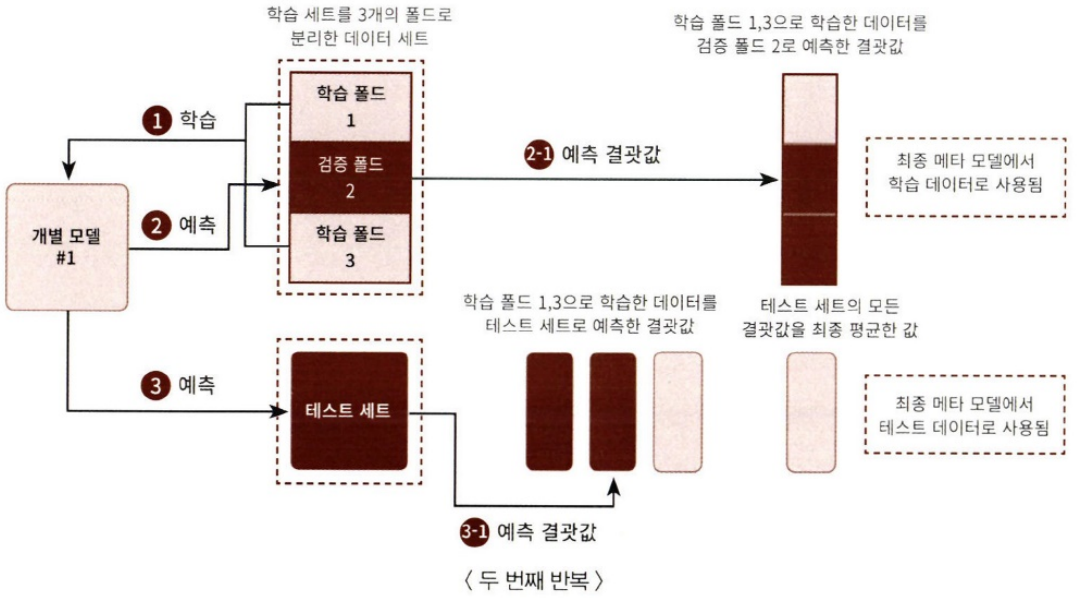
> - **스텝 1**: 각 모델별로 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터를 생성합니다.
>
> - **스텝2**: 스텝 1에서 개별 모델들이 생성한 학습용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 최종 학습용 데이터 세트를 생성합니다. 마찬가지로 각 모델들이 생성한 테스트용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 예측할 최종 테스트 데이트 세트를 생성합니다. 메타 모델은 최종적으로 생성된 학습 데이터 세트와 원본 학습 데이터의 레이블 데이터를 기반으로 학습한 뒤, 최종적으로 생성된 테스트 데이터 세트를 예측하고, 원본 테스트 데이터의 레이블 데이터를 기반으로 평가합니다

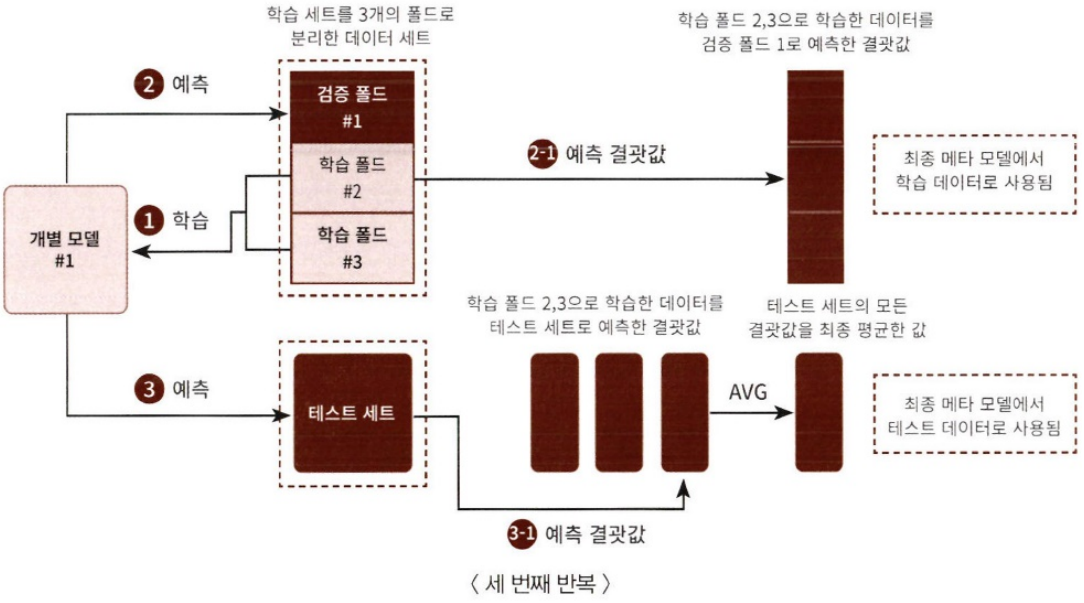
- 스텝 1

- 개별 모델에서 메타 모델인 2차 모델에서 사용될 학습용 데이터와 테스트용 데이터를 교차 검증을 통해 생성하는 것이 핵심
- 개별 모델 레벨에서 수행, 이러한 로직을 여러 개의 개별 모델에서 동일하게 수행해야

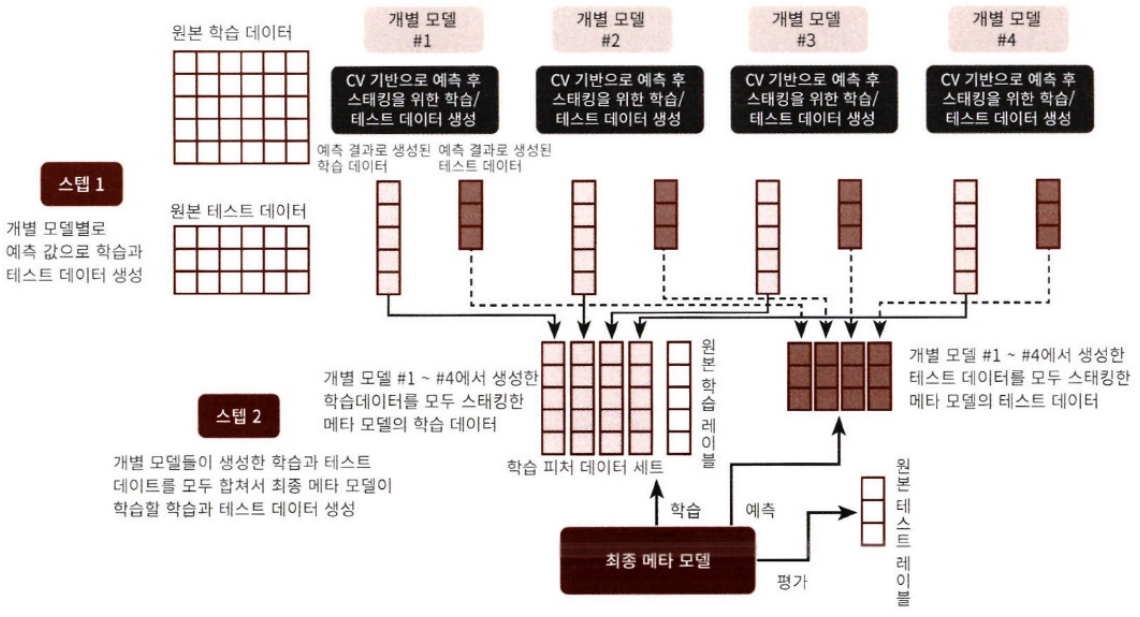
1. 학습용 데이터를 N개의 폴드(Fold)로 나눔, 2개의 폴드는 학습을 위한 데이터 폴드로, 나머지 1개의 폴드는 검증을 위한 데이터 폴드로 나눔
2. 두 개의 폴드로 나뉜 학습 데이터를 기반으로 개별 모델을 학습시킴
3. 학습된 개별 모델은 검증 폴드 1개 데이터로 예측하고 그 결과를 저장 → 예측 데이터는 메타 모델을 학습시키는 학습 데이터로 사용됨
4. 2개의 학습 폴드 데이터로 학습된 개별 모델은 원본 테스트 데이터를 예측하여 예측값을 생성







- 스탭 2
  - 각 모델들이 스텝1으로 생성한 학습과 테스트 데이터를 모두 합쳐서 최종적으로 메타 모델이 사용할 학습 데이터와 테스트 데이터를 생성하기만 하면 됨.
  - 메타 모델이 사용할 최종 학습 데이터와 원본 데이터의 레이블 데이터를 합쳐 메타 모델을 학습 → 최종 테스트 데이터로 예측 수행 → 최종 예측 결과를 원본 테스트 데이터의 레이블 데이터와 비교해 평가
- 전체 도식화



- 스태킹을 이루는 모델은 최적으로 파라미터를 튜닝한 상태에서 스태킹 모델을 만드는 것이 일반적임
    - 스태킹 모델의 파라미터 튜닝은 개별 알고리즘 모델의 파라미터를 최적으로 튜닝하는 것
- 스태킹 모델은 분류(Classification) 뿐만 아니라 회귀(Regression)에도 적용 가능