Chapter 04. 분류

05. GBM (Gradient Boosting Machine)

- 부스팅 알고리즘: 여러 개의 약한 학습기를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식
    - 대표적으로 AdaBoost와 그래디언드 부스트
        - AdaBoost: 오류 데이터에 가중치를 부여하면서 부스팅을 수행
            - 개별 약한 학습기는 각각 가중치를 부여해 결합
        - GBM: 에이다부스트와 유사하나, 가중치 업데이트에 경사 하강법 이용
- GBM의 개요
    - 오류 값은 실제 값 - 예측값
        - 실제 결과 값을 y, 피처를 x, 예측 함수를 F(x)라고 하면 오류식은 h(x) = y - F(x)
    - 경사 하강법: 오류식을 최소화하는 방향성을 가지고 반복적으로 가중치 값을 업데이트
    - 분류, 회귀 둘 다 가능
    - 사이킷런은 GradientBoostingClassifier 제공, 병렬 처리가 지원되지 않아서 대용량 데이터의 경우 학습에 매우 많은 시간 필요
    - 일반적으로 랜덤 포레스트보다 예측 성능이 조금 뛰어남
- GBM 하이퍼 파라미터 및 튜닝
    - loss: 경사 하강법에서 사용할 비용 함수를 지정, 기본값 'deviance'
    - learning_rate: GBM이 학습을 진행할 때마다 적용하는 학습률, Weak learner가 순차적으로 오휴 값을 보정해 나가는 데 적용하는 계수
        - n_estimators와 상호 보완적으로 조합해 사용
    - n_estimators: weak learner의 개수
    - subsample: weak learner가 학습에 사용하는 데이터의 샘플링 비

06. XGBoost(eXtra Gradient Boost)

- XGBoost 장점
    - GBM의 느린 수행 시간 및 과적합 규제 부재 등의 문제를 해결

        - 수행 속도 향상을 위한 조기 중단 기능 존재
    - 분류와 회귀 영역에서 뛰어난 예측 성능
    - Tree pruning(나무 가지치기)
    - 자체 내장된 교차 검증
    - 결손값 자체 처리
- XGBoost의 핵심 라이브러리는 C/C++로 작성
    - 파이썬 래퍼 XGBoost 모듈, 사이킷런 래퍼 XGBoost 모듈 존재

파이썬 래퍼 XGBoost

- 파이썬 래퍼 XGBoost 하이퍼 파라미터
    - 일반 파라미터: 일반적으로 실행 시 스레드의 개수나 silent 모드 등의 선택을 위함
        - booster: gbtree 또는 gblinear 선택
        - silent: 디폴트는 0, 출력 메시지를 나타내고 싶지 않으면 1
        - nthread: CPU의 실행 스레드 개수를 조정
    - 부스터 파라미터: 트리 최적화, 부스팅, regularization 등과 관련
        - eta [default=0.3,alias: learning_rate]: GBM의 학습률과 같음
        - num_boost_rounds: GBM의 n_estimators와 같음
        - min_child_weight[default=1]: 트리에서 추가적으로 가지를 나눌지를 결정하기 위해 필요한 데이터들의 weight 총합
        - gamma [default=0, alias: min_split_loss]: 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값
        - max_depth[default=6]: 트리 기반 알고리즘의 max_depth와 같음
        - sub_sample[default=1]: GBM의 subsample과 동일
        - colsample_bytree[default=1]: GBM의 max_features와 유사
        - lambda [default=1,alias:reg_lambda]: L2 Regularization 적용 값
        - alpha [default=0,alias: reg_alpha]: L1 Regularization 적용 값
        - scale_pos_weight[default=1]: 특정 값으로 치우친 비대칭한 클래스로 구성된 데이터 세트의 균형을 유지하기 위함
    - 학습 태스크 파라미터: 학습 수행 시의 객체 함수, 평가를 위한 지표 등을 설정
        - objective: 최솟값을 가져야할 손실 함수를 정의
        - binary:logistic: 이진 분류일 때 적용
        - multi:softmax: 다중 분류일 때 적용
        - multi:softprob: 개별 레이블 클래스의 해당되는 예측 확률 반환
        - eval_metric: 검증에 사용되는 함수를 정의

- 조기 중단 기능
    - early_stopping_rounds 파라미터를 입력하여 설정

        - eval_set: 성능 평가를 수행할 평가용 데이터 세트 설정
        - eval_metric: 평가 세트에 적용할 성능 평가 방법

- xgboost 패키지에 내장된 시각화 기능
    - plot_importance( ): 피처의 중요도를 막대그래프 형식으로 나타냄
    - to_graphiz( ): 규칙 트리 구조 그릴 수 있음
- cv( ): 데이터 세트에 대한 교차 검증 수행 후 최적화 파라미터를 구할 수 있는 방법
    - 파라미터: params, dtrain, num_boost_round, nfold, stratified, metrics, early_stopping_rounds

- 파이썬 래퍼 XGBoost와 사이킷런의 차이
    - 학습용과 테스트용 데이터 세트를 위해 별도의 객체인 DMatrix를 생성
        - DMatrix: 넘파이에서 입력 파라미터를 받아서 만들어지는 XGBoost만의 전용 데이터 세트
            - 주요 입력 파라미터: data, label
            - 넘파이 외에 libsvm txt 포맷 파일, xgboost 이진 버퍼 파일을 파라미터로 입력받아 반환 가능
    - 하이퍼 파라미터를 xgboost 모듈의 train( ) 함수에 파라미터로 전달
        - 사이킷런은 Estimator의 생성자를 하이퍼 파라미터로 전달
    - xgboost의 predict( )는 예측 결과를 추정할 수 있는 확률 값을 반환
        - 사이킷런은 예측 결과 클래스 값을 반환
    - plot_importance( )를 이용해 바로 피처 중요도 시각화
        - 사이킷런은 Estimator 객체의 feature_importances_ 속성을 이용해 직접 시각화 코드 작성

사이킷런 래퍼 XGBoost

- 분류를 위한 래퍼 클래스인 XGBClassifier, 회귀를 위한 래퍼 클래스인 XGBRegressor
    - XGBClassifier에서 변경한 파라미터
        - eta -> learning_rate
        - sub_sample -> subsample
        - lambda -> reg_lambda
        - alpha -> reg_alpha

- 조기 중단 기능
    - 조기 중단 관련한 파라미터를 fit( )에 입력

        - 파라미터: early_stopping_rounds, eval_metric, eval_set
    - 성능 평가를 수행할 데이터 세트는 학습 데이터가 아니라 별도의 데이터 세트

07. LightGBM

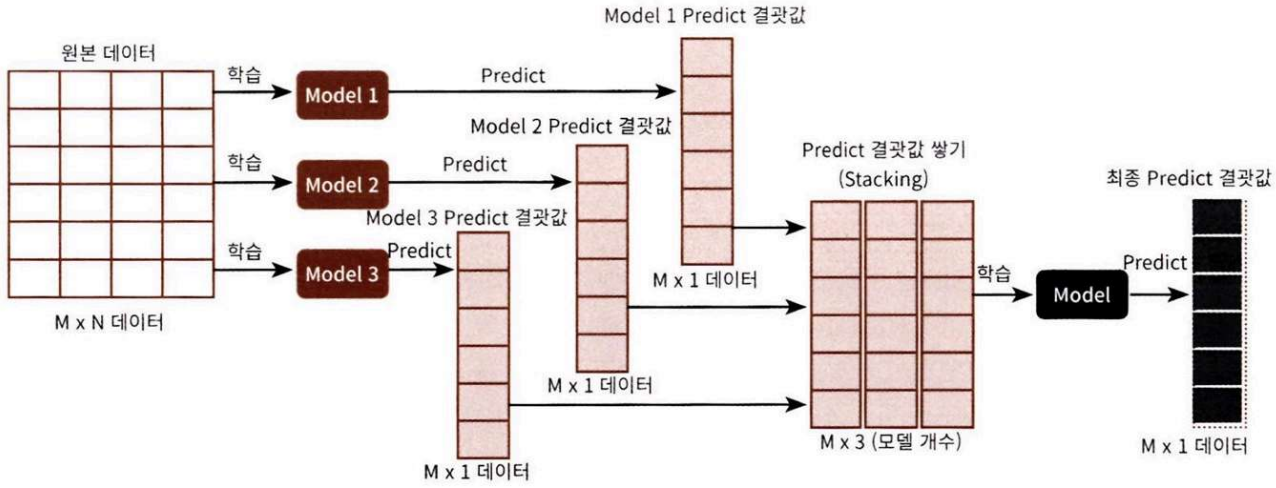
- XGBoost보다 학습에 걸리는 시간이 훨씬 적고, 메모리 사용량도 상대적으로 적음
- 일반 GBM 계열의 트리 분할 방법과 다르게 리프 중심 트리 분할 방식을 사용
    - 트리의 균형을 맞추지 않고, 최대 손실 값을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리가 생성됨
- 파이썬 패키지명 'lightgbm'
    - 파이썬 래퍼용과 사이킷런 래퍼 둘 다 존재
        - 사이킷런 래퍼 LightGBM 클래스는 분류를 위한 LGBMClassifier 클래스와 회귀를 위한 LGBMRegressor 클래스
            - fit( ), predict( ) 기반의 학습 및 예측 가능
- LightGBM 하이퍼 파라미터
    - 주요 파라미터
        - num_iterations [default=100]: 반복 수행하려는 트리의 개수 지정
        - learning_rate [default=0.1]: 0에서 1 사이의 값을 지정하며 부스팅 스텝을 반복적으로 수행할 때 업데이트 되는 학습률 값
        - max_depth [default=-1]: 트리 기반 알고리즘의 max_depth와 같음
        - min_data_in_leaf [default=20]: 결정 트리의 min_samples_leaf와 같은 파라미터
        - num_leaves [default=31]: 하나의 트리가 가질 수 있는 최대 리프 개수
        - boosting [default=gbdt]: 부스팅의 트리를 생성하는 알고리즘 기술
        - bagging_fraction [default=1.0]: 트리가 커져서 과적합되는 것을 제어하기 위해서 데이터를 샘플링하는 비율을 지정
        - feature_fraction [default=1.0]: 개별 트리를 학습할 때마다 무작위로 선택하는 피처의 비율
        - lambda_l2 [default=0.0]: L2 regulation 제어를 위한 값
        - lambda_l1 [default=0.0]: L1 regulation 제어를 위한 값
    - Learning Task 파라미터
        - objective: 최솟값을 가져야 할 손실함수를 정의
    - 하이퍼 파라미터 튜닝 방안
        - num_leaves의 개수를 높이면 정확도가 높아지지만, 트리의 깊이가 깊어지고 모델의 복잡도가 커져 과적합 영향도가 커짐
        - min_data_in_leaf는 과적합을 개선하기 위해 중요한 파라미터

        - max_depth는 num_leaves, min_data_in_leaf와 결합해 과적합을 개선하는 데 사용
- plot_importance( )로 피처 중요도 시각화

10. 스태킹 앙상블

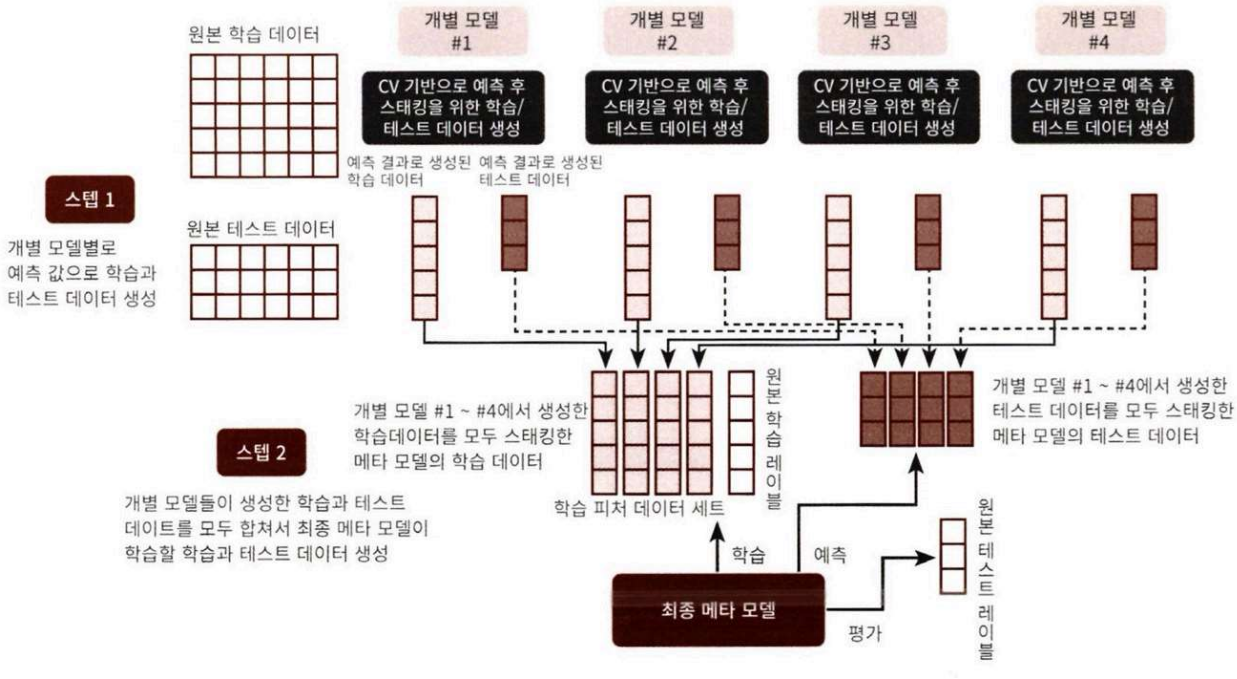
스태킹: 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출, 예측한 데이터를 기반으로 다시 예측을 수행

- 개별적인 기반 모델, 최종 메타 모델
    - 핵심은 여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습용 피처 데이터 세트와 테스트용 피처 데이터 세트를 만드는 것
- 2-3개의 개별 모델만을 결합해서는 쉽게 예측 성능을 향상시킬 수 없음



CV 세트 기반의 스태킹: 최종 메타 모델을 위한 데이터 세트를 만들 때 교차 검증 기반으로 예측된 결과 데이터 세트를 이용

- 스텝1: 각 모델별로 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터를 생성
- 스텝2: 스텝1에서 개별 모델들이 생성한 학습용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 최종 학습용 데이터 세트를 생성



HyperOpt

- XGBoost나 LightGBM에 Grid Search를 적용할 경우 기하급수적으로 늘어나는 하이퍼 파라미터 최적화 시간 때문에 어쩔 수 없이 하이퍼 파라미터 개수를 줄이거나 개별 하이퍼 파라미터의 범위를 줄여야 함
    - 하이퍼 파라미터 튜닝 시 Grid Search 방식보다는 베이지안 최적화 기법 이용

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

- 베이지안 확률에 기반을 둠
- 새로운 데이터를 입력 받았을 때 최적 함수를 예측하는 사후 모델을 개선해 나가면서 최적 함수 모델을 만들어 냄
- 두 가지 중요 요소: 대체 모델, 획득 함수
    - 대체 모델: 획득 함수로부터 최적 함수를 예측할 수 있는 입력값을 추천 받은 뒤 이를 기반으로 최적 함수 모델을 개선
    - 획득 함수: 개선된 대체 모델을 기반으로 최적 입력값을 계산
- 베이지안 최적화의 단계
    - Step1: 최초에는 랜덤하게 하이퍼 파라미터들을 샘플링하고 성능 결과를 관측
    - Step2: 관측된 값을 기반으로 대체 모델은 최적 함수를 추정
    - Step3: 추정된 최적 함수를 기반으로 획득 함수는 다음으로 관측할 하이퍼 파라미터 값을 계산

    - Step4: 획득 함수로부터 전달된 하이퍼 파라미터를 수행하여 관측된 값을 기반으로 대체 모델은 갱신되어 다시 최적 함수를 예측 추정
- 머신러닝 모델의 하이퍼 파라미터 튜닝에 적용할 수 있게 여러 파이썬 패키지 제공
    - HyperOpt, Bayesian Optimization, Optuna 등

HyperOpt

- 주요 로직
    - 입력 변수명과 입력값의 검색 공간 설정
        - hp 모듈 이용
        - 입력값의 검색 공간을 제공하는 대표 함수
            - hp.quniform, hp.uniform, hp.randint, hp.loguniform, hp.choice
    - 목적 함수 설정
        - 목적 함수는 반드시 변숫값과 검색 공간을 가지는 딕셔너리를 인자로 받고, 특정 값을 반환하는 구조로 만들어져야
    - 목적 함수의 반환 최솟값을 가지는 최적 입력값을 유추
        - 최댓값이 아니라 최솟값이라는 점에서 다른 패키지와 다름
        - fmin(objective, space, algo, max_evals, trials) 함수 제공
            - fmin( )의 주요 인자: fn, space, algo, max_evals, trials, rstate
            - fmin( ) 수행 시 인자로 들어가는 Trials 객체는 함수의 반복 수행 시마다 입력되는 변숫값들과 함수 반환값을 속성을 가짐
                - Trials의 중요 속성: results, vals
                    - results: 함수의 반복 수행 시마다 반환되는 반환값
                    - vals: 함수의 반복 수행 시마다 입력되는 입력 변숫값
- HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화
    - 적용해야 할 하이퍼 파라미터와 검색 공간 설정, 목적 함수에서 XGBoost 학습 후에 예측 성능 결과를 반환 값으로 설정
    - fmin( ) 함수에서 목적 함수를 하이퍼 파라미터 검색 공간의 입력값들을 사용하여 최적의 예측 성능 결과를 반환하는 최적 입력값들을 결정
    - 주의해야 할 점
        - 특정 하이퍼 파라미터들은 정숫값만 입력 받는데, HyperOpt는 입력값과 반환 값이 모두 실수형이기 때문에 하이퍼 파라미터 입력 시 형변환을 해줘야 함
        - HyperOpt의 목적 함수는 최솟값을 반환할 수 있도록 최적화해야 하기 때문에 성능 값이 클수록 좋은 성능 지표일 경우 -1을 곱해 줘야 함