4.5 GBM(Gradient Boosting Machine)

GBM의 개요 및 실습

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

GBM(Gradient Boost Machine)

- 에이다 부스트와 유사, 가중치 업데이트를 경사 하강법(Gradient Descent)을 이용하는 것이 큰 차이
- 오류 값 = 실제 값 - 예측값 → 오류식: $h(x) = y - F(x)$
  (y: 실제 결괏값, $x1, x2, \cdots, xn$: 피처, $F(x)$: 예측 함수)
- 경사 하강법: 오류식을 최소화하는 방향성을 가지고 반복적으로 가중치 값 업데이트 하는 것
  ⇒ 반복 수행을 통해 오류를 최소화할 수 있도록 가중치의 업데이트 값을 도출하는 기법
- 분류(사이킷런 `GradientBoostingClassifier` 클래스), 회귀 가능
- 장점: 일반적으로 GBM이 랜덤 포레스트보다 예측 성능이 조금 뛰어난 경우 多,
  과적합에도 강한 뛰어난 예측 성능
- 단점: 수행 시간 오래 걸림, 하이퍼 파라미터 튜닝 노력 더 필요

GBM 하이퍼 파라미터 소개

- `loss`: 경사 하강법에서 사용할 비용함수 지정, default='deviance'

- `learning_rate`: GBM이 학습을 진행할 때마다 적용하는 학습률, 0~1, default=0.1
  작은 값 → 예측 성능이 높아질 가능성 높음, 수행 시간 오래 걸림
  큰 값 → 예측 성능이 떨어질 가능성 높아짐, 빠른 수행 가능
- `n_estimators`: weak learner의 개수, default=100
  큰 값 → 예측 성능이 일정 수준까지 좋아질 수 있음, 수행 시간 오래 걸림
- `subsample`: weak learner가 학습에 사용하는 데이터의 샘플링 비율, default=1
  1 → 전체 학습 데이터를 기반으로 학습, 0.5 → 학습 데이터의 50%로 학습
  과적합을 방지하기 위해서 1보다 작은 값으로 설정

4.6 XGBoost(eXtra Gradient Boost)

XGBoost 개요

- 트리 기반의 앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나
- 분류에 있어서 일반적으로 다른 머신러닝보다 뛰어난 예측 성능
- GBM에 기반, GBM의 단점인 느린 수행 시간 및 과적합 규제(Regularization) 부재 등의 문제를 해결

XGBoost의 주요 장점

항목	설명
뛰어난 예측 성능	일반적으로 분류와 회귀 영역에서 뛰어난 예측 성능 발휘
GBM 대비 빠른 수행 시간	병렬 수행 및 다양한 기능으로 일반적인 GBM에 비해 빠른 수행 성능 보장 다른 머신러닝 알고리즘(ex. 랜덤 포레스트)에 비해 빠르다는 의미는 아님
과적합 규제 (Regularization)	XGBoost는 자체에 과적합 규제 기능으로 과적합에 좀 더 강한 내구성
Tree pruning (나무 가지치기)	일반적으로 GBM은 분할 시 부정 손실이 발생하면 분할을 더 이상 수행하지 않지만 이러한 방식도 지나치게 많은 분할을 발생할 수 있음 max_depth 파라

항목	설명
	미터로 분할 깊이를 조정하기도 하지만, tree pruning 으로 더 이상 긍정 이득이 없는 분할을 가지치기 해서 분할 수를 더 줄이는 추가적인 장점을 지님
자체 내장된 교차 검증	반복 수행 시마다 내부적으로 학습 데이터 세트와 평가 데이터 세트에 대한 교차 검증을 수행해 최적화된 반복 수행 횟수를 가질 수 있음 지정된 반복 횟수가 아닌 교차 검증을 통해 평가 데이터 세트의 평가 값이 최적화 되면 반복을 중간에 멈출 수 있는 조기 중단 기능을 지님
결손값 자체 처리	결손값을 자체 처리할 수 있는 기능이 있음

- 파이썬 래퍼 XGBoost 모듈: 초기 독자적인 XGBoost 프레임워크 기반의 XGBoost
- 사이킷런 래퍼 XGBoost: 사이킷런과 연동되는 모듈

파이썬 래퍼 XGBoost 하이퍼 파라미터

- XGBoost는 GBM과 유사한 하이퍼 파라미터를 동일하게 가지고 있으며, 조기 중단(early stopping), 과적합을 규제하기 위한 하이퍼 파라미터 등이 추가됨.
- **일반 파라미터**: 일반적으로 실행 시 스레드의 개수나 silent 모드 등의 선택을 위한 파라미터, 디폴트 파라미터 값을 바꾸는 경우는 거의 없음
- **부스터 파라미터**: 트리 최적화, 부스팅, regulatization 등과 관련 파라미터 등
- **학습 태스크 파라미터**: 학습 수행 시의 객체 함수, 평가를 위한 지표 등을 설정하는 파라미터

세부 주요 파라미터는 파머완 p.229~230 참고

- 뛰어난 알고리즘일수록 파라미터를 튜닝할 필요 少, 피처의 수가 매우 많거나 피처 간 상관되는 정도가 많거나 데이터 세트에 따라 파라미터를 튜닝하기도 함.
- 과적합 문제가 심각하다면 다음과 같이 적용
  - eta 값 낮추기(0.01~0.1), num_round(또는 n_estimators)는 높여줘야 함
  - max_depth 값 낮추기

    - min_child_weight 값 높이기
    - gamma 값 높이기
    - subsample과 colsample_bytree 조정
      → 트리가 너무 복잡하게 생성되는 것을 막아 과적합 문제에 도움이 될 수 있음
- 조기 중단(Early Stopping) 기능: n_estimators에 지정한 부스팅 반복 횟수에 도달하지 않더라도 예측 오류가 더 이상 개선되지 않으면 반복을 끝까지 수행하지 않고 중지하는 기능 → 수행 시간을 개선할 수 있음

파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측

실습 코드 참고

사이킷런 래퍼 XGBoost의 개요 및 적용

- 사이킷런 래퍼 XGBoost: `XGBClassifier` (분류), `XGBRegressor` (회귀)
- XGBClassifier는 기존 사이킷런에서 일반적으로 사용하는 하이퍼 파라미터와 호환성을 유지하기 위해 기존의 xgboost 모듈에서 사용하던 네이티브 하이퍼 파라미터 몇 개를 다음과 같이 변경함
    - eta → learning_rate
    - sub_sample → subsample
    - lambda → reg_lambda
    - alpha → reg_alppha
    - n_estimators, num_boost_round 동시 사용
      파이썬 래퍼 XGBoost: num_boost_round 파라미터 적용
      사이킷런 래퍼 XGBoost: n_estimators 파라미터 적용

4.7 LightGBM

LightGBM

- 장점: XGBoost보다 학습에 걸리는 시간이 훨씬 少, 메모리 사용량도 상대적으로 少
- LightBGM과 XGBoost의 예측 성능은 별다른 차이가 없으며, 기능상의 다양성은 LightGBM이 약간 더 多
- 단점: 적은 데이터 세트(일반적으로 10,000건 이하의 데이터 세트)에 적용할 경우 과적합이 발생하기 쉬움
- 일반 GBM 계열의 트리 분할 방법: 균형 트리 분할(Level Wise) 방식
  LightGBM 트리 분할 방법: 리프 중심 트리 분할(Leaf Wise) 방식
  트리의 균형을 맞추기 않고, 최대 손실 값(max delta loss)을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리 생성 →
  LightGBM: 예측 오류 손실 최소화
- 사이킷런 래퍼 LightGBM: `LGBMlassifier` (분류), `LGBMRegressor` (회귀)

LightGBM의 XGBoost 대비 장점

- 더 빠른 학습과 예측 수행 시간
- 더 작은 메모리 사용량
- 카테고리형 피처의 자동 변환과 최적 분할(원-핫 인코딩 등을 사용하지 않고도 카테고리형 피처를 최적으로 변환하고 이에 따른 노드 분할 수행)

LightGBM 하이퍼 파라미터

- LightGBM 하이퍼 파라미터는 XGBoost와 많은 부분이 유사
- LightGBM은 리프 노드가 계속 분할되면서 트리의 깊이가 깊어지므로 이러한 트리 특성에 맞는 하이퍼 파라미터 설정이 필요함
- **주요 파라미터**: 파머완 p.247~248 참고
- **Learning Task 파라미터**: 파머완 p.248 참고

하이퍼 파라미터 튜닝 방안

- 기본 튜닝 방안: num_leaves의 개수를 중심으로
  min_child_samples(min_data_in_leaf), max_depth를 함께 조정하면서 모델의 복잡도를 줄이는 것
- num_leaves: 개별 트리가 가질 수 있는 최대 리프의 개수
  LightGBM 모델의 복잡도를 제어하는 파라미터
  일반적으로 개수↑ ⇒ 정확도 ↑, 트리의 깊이 깊어짐, 복잡도가 커져 과적합 영향도가 커짐
- min_data_in_leaf: 사이킷런 래퍼 클래스에서는 min_child_samples
  과적합을 개선하기 위한 중요한 파라미터
  보통 큰 값 ⇒ 트리가 깊어지는 것 방지
- max_depth: 명시적으로 깊이의 크기 제한
  num_leaves, min_data_in_leaf와 결합해 과적합을 개선하는데 사용
- learning_rate를 작게 하면서 n_estimators를 크게 하기, n_estimators를 너무 크게 하는 것은 과적합 가능성 ↑
- 과적합을 제어하기 위해서 reg_lambda, reg_alpha와 같은 regularization을 적용하거나 학습 데이터에 사용할 피처의 개수나 데이터 샘플링 레코드 개수를 줄이기 위해 colsample_bytree, subsample 파라미터를 적용

파이썬 래퍼 LightGBM과 사이킷런 래퍼 XGBoost, LightGBM 하이퍼 파라미터 비교

- 사이킷런 래퍼 LightGBM 클래서와 사이킷런 래퍼 XGBoost 클래스는 많은 하이퍼 파라미터가 똑같음

파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
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

파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
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

실습 코드 참고

4.8 베이지안 최적화 기반의 HyperOpt를 이용한 하이퍼 파라미터 튜닝

사이킷런 Grid Search 방식을 적용한 하이퍼 파라미터 튜닝 다음의 경우에 수행 시간이 오래 걸림

- 튜닝해야 할 하이퍼 파라미터 개수가 많을 경우
- 개별 하이퍼 파라미터 값의 범위가 넓거나 학습 데이터가 대용량일 경우

⇒ XGBoost나 LightGBM에 Grid Search를 적용할 경우, 기하급수적으로 늘어나는 하이퍼 파라미터 최적화 시간으로 인해 하이퍼 파라미터 개수를 줄이거나 개별 하이퍼 파라미터의 범위를 줄여야 함

베이지안 최적화 개요

- 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 두고 있는 최적화 기법
  → 새로운 데이터를 입력 받았을 때 최적 함수를 예측하는 사후 모델을 개선해 나가면서 최적 함수 모델 생성
- 베이지안 최적화를 구성하는 두 가지 중요 요소
  - 대체 모델(Surrogate Model), 획득 함수(Acquisition Function)
  - 대체 모델은 획득 함수로부터 최적 함수를 예측할 수 있는 입력값을 추천 받고 이를 기반으로 최적 함수 모델 개선하며, 획득함수는 개선된 대체 모델을 기반으로 최적 입력값을 계산

  ⇒ 입력 값은 하이퍼 파라미터에 해당하며, 대체 모델은 획득 함수로부터 하이퍼 파라미터를 추천받아 모델 개선 수행, 획득 함수는 개선된 모델을 바탕으로 더 정확한 하이퍼 파라미터를 계산함

베이지안 최적화 프로세스

- step1: 랜덤하게 하이퍼 파라미터들을 샘플링하고 성능 결과 관측
- step2: 관측된 값을 기반으로 대체 모델은 최적 함수 및 신뢰구간 추정
- step3: 추정된 최적 함수를 기반으로 획득 함수는 다음으로 관찰할 하이퍼 파라미터를 계산 후 이를 대체 모델에 전달
- step4: 획득 함수로부터 전달된 하이퍼 파라미터를 수행하여 관측된 값을 기반으로 대체 모델 다시 갱신
- step3, step4를 반복하면 대체 모델의 불확실성 개선 및 점차 정확한 최적 함수 추정 가능

HyperOpt 사용하기

- 대체 모델은 최적 함수를 추정할 때 일반적으로 가우시안 프로세스(Gaussian Process)를 적용하지만 HyperOpt는 트리 파르젠 Estimator(TPE, Tree-structure

Parzen Estimator)를 사용

- HyperOpt의 기본적인 사용법
  - 입력 변수명과 입력값의 검색 공간(Search Space) 설정
  - 목적 함수(Objective Function) 설정
  - 목적 함수의 반환 최솟값을 가지는 최적 입력값을 유추
    → `fmin()` 함수

HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화

- HyperOpt는 입력 값과 반환 값이 모두 실수형이기 때문에 하이퍼 파라미터 입력 시 형변환 필요
- HyperOpt는 목적 함수의 최솟값을 반환할 수 있도록 최적화해야 하기 때문에 성능 값이 클수록 좋은 성능 지표일 경우 -1을 곱해 주어함

4.11 스태킹 앙상블

스태킹(Stacking)

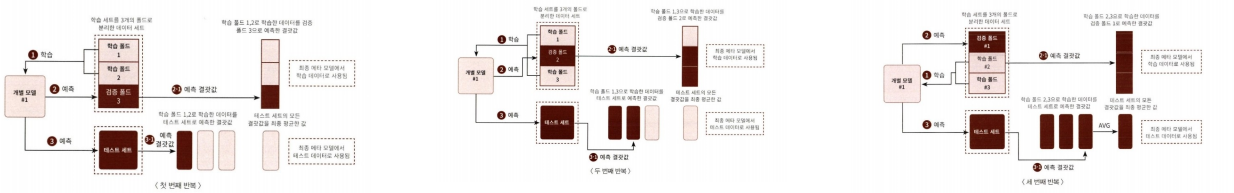
- 배깅(Bagging)과 부스팅(Boosting)과 비교
  공통점: 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출
  차이점: 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행
- 개별적인 기반 모델, 이 개별 기반 모델의 예측 데이터를 학습데이터로 만들어서 학습하는 최종 메타 모델이 필요함 ⇒ 스태킹 모델은 여러 개별 모델의 예측 데이터를 각각 스태킹 형태로 결합해 최종 메타 모델의 학습용 피처 데이터 세트와 테스트용 피처 데이터 세트를 만드는 것이 핵심

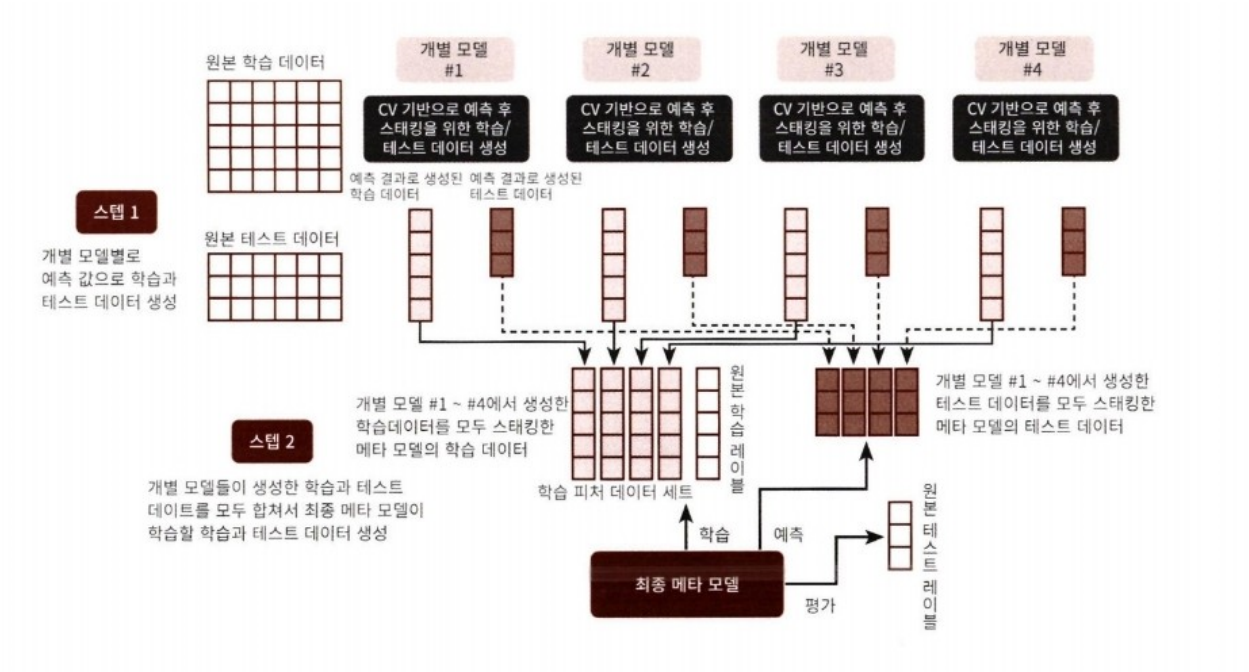
기본 스태킹 모델

- 데이터 로드
- 데이터 분리
- 개별 ML 모델 생성
- 스태킹으로 만들어진 데이터세트를 학습할 최종 모델 생성
- 개별 ML 모델 학습/예측, 정확도 확인
- 개별 ML 모델의 예측 결과를 피처로 생성
- 최종 메타 모델인 로지스틱 회귀로 학습 및 평가 수행

CV 세트 기반의 스태킹

- step1: 각 모델별로 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터를 생성
- step2: step1에서 개별 모델들이 생성한 학습용/테스트용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 최종 학습용/테스트용 데이터 세트 생성. 메타 모델은 최종적으로 생성된 학습 데이터 세트와 원본 학습 데이터의 레이블 데이터 기반으로 학습한 뒤, 최종적으로 생성된 테스트 데이터 세트를 예측하고, 원본 테스트 데이터의 레이블 데이터를 기반으로 평가




원본 학습 데이터
개별 모델 #1
개별 모델 #2
개별 모델 #3
개별 모델 #4
CV 기반으로 예측 후 스태킹을 위한 학습/테스트 데이터 생성
CV 기반으로 예측 후 스태킹을 위한 학습/테스트 데이터 생성
CV 기반으로 예측 후 스태킹을 위한 학습/테스트 데이터 생성
CV 기반으로 예측 후 스태킹을 위한 학습/테스트 데이터 생성
예측 결과로 생성된 학습 데이터
예측 결과로 생성된 테스트 데이터
스텝 1
개별 모델별로 예측 값으로 학습과 테스트 데이터 생성
원본 테스트 데이터
개별 모델 #1 ~ #4에서 생성한 학습데이터를 모두 스태킹한 메타 모델의 학습 데이터
원본 학습 레이블
개별 모델 #1 ~ #4에서 생성한 테스트 데이터를 모두 스태킹한 메타 모델의 테스트 데이터
스텝 2
개별 모델들이 생성한 학습과 테스트 데이트를 모두 합쳐서 최종 메타 모델이 학습할 학습과 테스트 데이터 생성
학습 피처 데이터 세트
학습
예측
원본 테스트 레이블
최종 메타 모델
평가