3

[Week3] 파머완 4장(2)

CHAPTER 04 분류

05 GBM(Gradient Boosting Machine)

GBM 개요 및 실습

부스팅 알고리즘

- 여러 개의 약한 학습기를 순차적으로 학습, 예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식
- AdaBoost(Adaptive boosting) / 그래디언트 부스트

GBM

- 가중치 업데이트를 경사 하강법(Gradient Descent)를 이용
- 경사 하강법 : 오류식 h(x) = y - F(x)를 최소화하는 방향성을 가지고 반복적으로 가중치 값을 업데이트 하는 것 ** y : 분류의 실제 결괏값 / F(x) : 피처 x1, … , xn에 기반한 예측 함수
- 사이킷런 GradientBoostingClassifier 클래스
- 일반적으로 랜덤 포레스트보다 예측 성능 뛰어남, 수행시간 더 오래 걸림

GBM 하이퍼 파라미터 소개

loss = 'deviance' : 경사 하강법에서 사용할 비용 함수 지정

learning_rate = 0.1 : 학습을 진행할 때마다 적용하는 학습률. 오류를 보정해 나가는 데 적용하는 계수. 0~1. → n_estimators와 상호 보완적으로 조합해 사용

n_estimators = 100 : weak learner의 개수, 개수가 많을수록 일정 수준까지 예측 성능 향상 but 오랜 수행 시간

subsample = 1 : 데이터 샘플링 비율, 과적합 염려 시 1보다 작은 값으로 설정

06 XGBoost(eXtra Gradient Boost)

XGBoost 개요

장점

- 뛰어난 예측 성능 : 분류, 회귀
- GBM 대비 빠른 수행 시간 : 병렬 수행
- 과적합 규제
- Tree pruning(나무 가지치기) : 더 이상 긍정 이득이 없는 분할을 가지치기하여 분할 수 감소
- 자체 내장된 교차 검증 : 최적화되면 반복 중단하는 조기 중단 기능
- 결손값 자체 처리

파이썬 래퍼 XGBoost 모듈 : 고유의 API와 하이퍼 파라미터 이용

일반 파라미터 → 디폴트 파라미터 값을 변경하는 경우 거 학습 태스크 파라미터

의 없음

- booster : gbtree(tree based model) 또는 gblinear(linear model) 선택
- silent = 0 : 출력 메시지를 나타내고 싶지 않으면 1
- nthread : CPU 실행 스레드 개수 조정

부스터 파라미터 → 트리 최적화, 부스팅, 과적합 규제 등 대부분의 하이퍼 파라미터

- eta = 0.3 [alias : learning_rate = 0.1] : GBM의 learning_rate와 같은 파라미터, 0~1 사이의 값, 0.01~0.2 사이의 값 선호
- num_boost_rounds : GBM의 n_estimators와 같은 파라미터
- min_child_weight = 1 : 트리에서 추가적으로 가지를 나눌지를 결정하기 위해 필요한 데이터들의 가중치 총합, 클수록 분할 자제, 과적합 조절
- gamma = 0 [alias : min_split_loss] : 해당 값보다 큰 손실이 감소된 경우 리프 노드 분리, 클수록 과적합 감소
- max_depth = 6 : 트리 기반 알고리즘의 max_depth와 동일, 0 → 깊이 제한 X, 클수록 과적합 가능성 증가, 보통 3~10 적용
- sub_sample = 1 : GBM의 subsample과 동일, 과적합 제어를 위해 샘플링 비율 지정, 0.5 → 전체 데이터의 절반을 트리 생성에 사용, 보통 0.5~1 사용
- colsample_bytree = 1 : GBM의 max_features와 유사, 트리 생성에 필요한 피처를 임의로 샘플링, 과적합 조정
- lambda = 1 [alias : reg_lambda] : L2 Regularization, 클수록 과적합 감소
- alpha = 0 [alias : reg_alpha] : L1 Regularization, 클수록 과적합 감소
- scale_pos_weight = 1 : 비대칭한 클래스로 구성된 데이터셋 균형을 유지하기 위한 파라미터
- objective : 최솟값을 가져야 할 손실 함수 정의, 이진분류/다중분류에 따라 달라짐
  - binary:logistic : 이진 분류 시 적용
  - multi:softmax : 다중 분류 시 적용, num_class 파라미터로 레이블 클래스 개수 지정 필요
  - multi:softprob : 개별 레이블 클래스의 해당되는 예측 확률 반환
- eval_metric : 검증에 사용되는 함수 정의, 기본값 → rmse(회귀) / error(분류)
  - 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

과적합 조절

- eta 값 낮추기(0.01~0.1) → num_round(n_estimators) : 높여줘야 함
- max_depth 값 낮추기
- min_child_weight 값 높이기
- gamma 값 높이기
- subsample, colsample_bytree 조정하기 → 트리 복잡도 조절

❗조기 중단(Early Stopping) : n_estimators에 지정한 부스팅 반복 횟수에 도달하지 않더라도 예측 오류가 더 개선되지 않으면 반복 중지하여 수행 시간 개선

파이썬 래퍼 XGBoost 적용 - 위스콘신 유방암 예측

plot_importance : xgboost에서 피처 중요도를 시각화해주는 모듈

DMatrix : XGBoost 만의 전용 데이터 객체

- data : 피처 데이터 세트
- label : 레이블 데이터 세트(분류) / 숫자형인 종속값 데이터 세트(회귀)

사이킷런 래퍼 XGBoost 개요 및 적용

XGBClassifier : 사이킷런 XGBoost 분류 클래스

- eta → learning_rate : 학습률
- sub_sample → subsample : 과적합 제어를 위한 샘플링 비율
- lambda → reg_lambda : L2 Regularization

조기 중단 성능 평가

- 주로 별도의 검증 데이터 세트를 이용
- early_stopping_rounds
- 평가용 데이터셋 지정 & eval_metric 설정 필요

to_graphvix( ) : 규칙 트리 구조 시각화

cv( ) : 교차 검증 수행 후 최적 파라미터를 구하는 API, DataFrame 형태로 반환

- params : 부스터 파라미터
- dtrain : 학습 데이터
- num_boost_rount : 부스팅 반복 횟수
- nfold : CV 폴드 개수
- stratified : CV 수행 시 층화 표본 추출 수행 여부
- metrics : CV 수행 시 모니터링할 성능 평가 지표
- early_stopping_rounds : 조기 중단 활성화, 반복 횟수 지정
- alpha → reg_alpha : L1 Regularization

07 LightGBM

LightGBM

- XGBoost에 비해 학습 시간 짧고 메모리 사용량 적음 but 예측 성능 차이 없음
- 적은 데이터셋(10,000건 이하)에 적용할 경우 과적합 발생하기 쉬움
- 리프 중심 트리 분할 방식 사용 : 최대 손실 값을 가지는 리프노드를 지속적으로 분할 → 깊고 비대칭적인 규칙 트리 생성 → 예측 오류 손실 최소화

  ** 대부분 균형 트리 분할 방식 사용 : 오버피팅에 보다 강한 구조 but 균형 맞추는 시간 필요
- 카테고리형 피처 지동 변환 & 최적 분할

LightGBM 하이퍼 파라미터

→ 트리의 깊이가 깊어지기 때문에 트리 특성에 맞는 하이퍼 파라미터 설정 필요

num_iterations = 100 [n_estimators] : 반복 수행 트리 개수 지정, 크게 지정할수록 예측 성능 상승(과적합 우려)

learning_rate = 0.1 : 0~1 사이의 학습률 값, n_estimators를 크게 learning_rate를 작게 하여 예측 성능 향상(과적합 이슈, 학습시간 증가 고려)

max_depth = -1 : 트리 기반 알고리즘에서와 동일, 0보다 작은 값 지정하면 깊이 제한 X

min_data_in_leaf = 20 [min_child_samples] : 결정 트리의 min_samples_leaf와 동일, 최소한으로 필요한 레코드

하이퍼 파라미터 튜닝 방안

1. num_leaves 개수⬆️ → 정확도⬆️ → 트리 깊이⬆️ → 복잡도⬆️ → 과적합 영향도⬆️
2. min_data_in_leaf[min_child_samples]⬆️ → 트리 깊이⬇️ → 과적합 제어
3. max_depth 위 요소들과 결합해 과적합 개선
4. learning_rate⬇️ & n_estimators⬆️
5. regularization 적용 / colsample_bytree, subsample 적용

파이썬 래퍼 LightGBM과 사이킷런 래퍼 XGBoost, LightGBM 하이퍼 파라미터 비교

파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런
num_iterations	n_estimators	n_estim
learning_rate	learning_rate	learning
max_depth	max_depth	max_de
min_data_in_leaf	min_child_samples	N/A
bagging_fraction	subsampe	subsam
feature_fraction	colsamply_bytree	colsamp
lambda_l2	reg_lambda	re
lambda_l1	reg_alpha	reg_alp

수(과적합 제어)

num_leaves = 31 : 하나의 트리가 가질 수 있는 최대 리프 개수

boosting = gbdt : 부스팅 트리 생성 알고리즘 기술

- gdbt : 일반적은 그래디언트 부스팅 결정 트리
- rf : 랜덤포레스트

bagging_fraction = 1.0 [subsample] : 데이터 샘플링 비율 지정, 과적합 제어

feature_fraction = 1.0 [colsample_bytree] : 개별 트리를 학습할 때마다 무작위로 선택하는 피처 비율, 과적합 제어

lambda_l2 = 0.0 [reg_lambda] : L2 Regulation 제어값, 피처 개수 많을 경우 적용 검토, 클수록 과적합 감소

lambda_l1 = 0.0 [reg_alpha] : L1 Regulation 제어값

objective : 최솟값을 가져야 할 손실함수 정의

early_stopping_round	early_stopping_rounds	early_st
num_leaves	num_leaves	N/A
min_sum_hessian_in_leaf	min_child_weight	min_chi

08 베이지안 최적화 기반의 HyperOpt를 이용한 하이퍼 파라미터 튜닝

베이지안 최적화 개요

목적함수 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 확률 기반 → 새로운 데이터를 입력받았을 때 최적 함수를 예측하는 사후 모델 개선 → 최적 함수 모델 생성

대체 모델(Surrogate Model) : 획득 함수로부터 최적 함수를 예측할 수 있는 입력값을 추천 받은 뒤 이를 기반으로 최적 함수 모델 개선 → 가우시안 프로세스 / 트리 파르젠 사용

획득 함수(Acquisition Function) : 개선된 대체 모델을 기반으로 최적 입력값 계산

1. 랜덤하게 하이퍼 파라미터 샘플링하여 성능 결과 관측
2. 관측값 기반으로 대체모델이 최적 함수 추정 → 최적 관측값은 y축 value에서 가장 높은 값을 가질 때의 하이퍼 파라미터
3. 추정된 최적 함수를 기반으로 획득함수가 다음으로 관측할 하이퍼 파라미터 값 계산 → 더 큰 최댓값을 가질 가능성이 높은 지점을 찾아 대체 모델에 전달
4. 획득 함수로부터 전달된 하이퍼 파라미터 수행 → 대체 모델 갱신 ⇒ 반복

HyperOpt 사용

1. 입력 변수명 & 입력값 검색 공간 설정
   - hp.quniform(label, low, high, q) : label에 low에서 high까지 q 간격으로 설정
   - hp.uniform(label, low, high) : low에서 high까지 정규 분포 형태 설정
   - hp.randint(label, upper) : 0부터 upper까지 random한 정숫값으로 설정
   - hp.loguniform(label, low, high) : exp(uniform(low, high)) 값 반환, 반환 값의 log 변환된 값은 정규 분포 형태 설정
   - hp.choice(label, options) : 검색 값이 문자열/문자열과 숫자가 섞여 있을 경우 설정, Options는 리스트/튜플 형태로 제공
2. 목적 함수 설정

- 특정 값을 반환하는 구조
- 딕셔너리 반환 : 'loss'와 'status' 키 값 설정 필요

3. 목적 함수의 반환 최솟값을 가지는 최적 입력값 유추
   - fmin(objective, space, algo, max_evals, trials)
     - fn : 목적 함수
     - space : 검색 공간 딕셔너리
     - algo = tpe.suggest : 베이지안 최적화 적용 알고리즘
     - max_evals : 최적 입력값을 찾기 위한 시도 횟수
     - trials : 최적 입력값을 찾기 위해 시도한 입력값 & 해당 입력값의 목적 함수 반환값 결과 저장
     - rstate : 랜덤 시드 값
   - results : {'loss' : 함수 반환값, 'status' : 반환 상태값}
   - vals : {'입력변수명' : 개별 수행 시마다 입력된 값의 리스트}

HyperOpt를 이용한 XGBoost 하이퍼 파라미터 최적화

❗하이퍼 파라미터 입력 시 형변환 필요 → HyperOpt 실수형

❗성능 값이 클수록 좋은 성능 지표일 경우 -1 곱해줘야 함

09 분류 실습 - 캐글 산탄데르 고객 만족 예측

클래스 레이블 명 TARGET : 불만(1), 만족(0)

모델 성능 평가 ROC-AUC 평가 : 대부분이 만족 데이터 이므로

데이터 전처리

1. 학습 데이터 로딩, 만족/불만족 비율 확인, 분포 확인
2. 예외값 수정, 필요없는 피처 드롭
3. 학습 데이터 / 테스트 데이터 분리 → TARGET 값 분포도가 분리 전과 비슷한지 확인
4. 검증 데이터 분리

XGBoost/LightGBM 모델 학습과 하이퍼 파라미터 튜닝

1. XGBoost 학습 모델 생성, 예측 결과 ROC-AUC 평가
2. HyperOpt를 이용해 검색 공간 설정
3. 목적함수 생성
4. fmin( )함수로 최적 하이퍼 파라미터 도출 → 모델 재학습
5. 테스트 데이터셋에서 ROC-AUC 측정
6. 튜닝 모델에서 각 피처 중요도 확인

❗LightBGM도 같은 방법으로 학습&튜닝❗

10 분류 실습 - 캐글 신용카드 사기 검출

레이블 Class 속성 : 정상적인 신용카드 트랜잭션(0), 신용카드 사기 트랜잭션(1) → 매우 불균형한 분포

❗이상 레이블을 가지는 데이터 건수가 매우 적어 제대로 다양한 유형 학습이 어렵고 정상 레이블로 치우친 학습 수행으로 이상 데이터 검출이 어려움

❗LGBMClassifier 객체 생성 시 boost_from_average = False 설정

언더 샘플링

- 많은 레이블을 가진 데이터셋을 적은 레이블을 가진 데이터셋 수준으로 감소
- 정상 데이터 10,000건 vs 이상 데이터 100건 ⇒ 정상 데이터 100건으로 줄여 버리는 방식

- 정상 레이블로 과도하게 학습하는 부작용 개선 가능 but 정상 레이블의 경우 제대로 된 학습 수행이 어려운 문제

오버 샘플링

- 적은 레이블을 가진 데이터셋을 많은 레이블을 가진 데이터셋 수준으로 증식
- 원본 데이터의 피처 값들을 아주 약간 변경하여 증식
- SMOTE : 적은 데이터 셋에 있는 개별 데이터들의 K 최근접 이웃을 찾아 데이터와 K개 이웃들의 차이를 일정값으로 만들어서 기존 데이터와 약간 차이가 나는 새로운 데이터를 생성
  - imbalanced-learn 패키지

❗로지스틱 회귀 모델의 경우 SMOTE 오버 샘플링을 진행할 경우 재현율은 크게 증가하지만 정밀도가 저하

❗SMOTE 적용 시 재현율은 높아지나 정밀도는 낮아지는 것이 일반적

이상치 데이터(Outlier)

- 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터
- IQR : 사분위 값의 편차를 이용해 이상치 탐색
  - 사분위 : 전체 데이터를 값이 높은 순으로 정렬 → 1/4씩 구간 분할
  - IQR : Q1 ~ Q3 범위
  - IQR에 1.5를 곱해 생성된 범위에 속하지 않은 데이터를 이상치로 간주

11 스태킹 앙상블

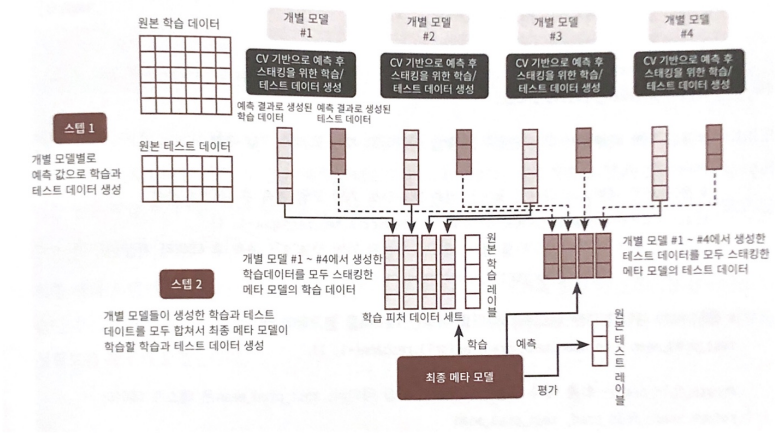
스태킹(Stacking)

- 개별적인 여러 알고리즘을 서로 결합해 예측 결과 도출 → 배깅, 부스팅과 공통점
- 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측 수행
- 개별적인 기반 모델과 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어 학습하는 최종 메타 모델이 필요
- 많은 개별 모델이 필요

CV 세트 기반 스태킹

- 과적합 개선을 위해 최종 메타 모델을 위한 데이터 세트를 만들 때 교차 검증 기반으로 예측된 결과 데이터셋 이용

1. 각 모델별 원본 학습/테스트 데이터를 예측한 결과값을 기반으로 메타 모델을 위한 학습/테스트 데이터 생성
2. 개별 모델의 학습용 데이터를 모두 스태킹 형태로 합쳐서 메타 모델이 학습할 최종 학습용 데이터셋 생성(테스트 데이터셋도 마찬가지로 생성)



12 정리

1. 앙상블 : 결정 트리 기반의 다수의 약한 학습기를 결합해 변동성을 줄여 예측 오류를 줄이고 성능 개선
2. 결정 트리 알고리즘 : 정보의 균일도에 기반한 규칙 트리로 예측 수행 → 트리가 깊어지고 복잡해지면서 과적합 쉽게 발생
3. 배깅 : 학습 데이터 중복을 허용하면서 다수의 세트로 샘플링하여 최종 결과 결합해 예측하는 방식 → 랜덤포레스트
4. 부스팅 : 학습기들이 순차적으로 학습을 진행하면서 예측이 틀린 데이터에 대해 가중치를 부여해 높은 정확도 예측이 가능하도록 하는 방식 → GBM
5. XGBoost & LightGBM
6. 스태킹 모델 : 여러 개의 개별 모델들이 생성한 예측 데이터를 기반으로 최종 메타 모델이 학습할 별도의 학습/테스트 데이터 셋을 재생성하는 기법