

05. GBM

부스팅 알고리즘: 여러 개의 약한 학습기를 결합 → 순차적 학습

Ada Boost (에이다 부스트)

- 오류 데이터에 가중치를 부여

- 개별적인 약한 학습기에 가중치를 부여 → 모두 결합

+	+	+	-
+	-	-	-

GBM (gradient Boost machine)

- 가중치 업데이트를 경사 하강법을 이용. → 반복을 이용하여 오류를 최소화할 수 있도록 가중치의 업데이트 값을 도출.

• 오류값: $(실제값 - 예측값)$ $h(x) = y - f(x)$
 • 분류, 회귀 모두 가능 오류의 절대 차이 예측 함수
 classifier regressor

- 일반적으로 랜덤 포레스트보다 예측 성능 ↑ but 시간 ↑, 투명 힐끔.

GBM 하이퍼 파라미터 / 튜닝

(결정 트리 기반으로 랜덤 포레스트 등과 경쟁)

- loss: 경사 하강법에 사용하는 비용함수 지정, default: deviance
- learning rate: GBM이 학습을 진행할 때마다 적용하는 학습률 (0~1, default: 0.1)
 - 너무 적을 경우: 업데이트 되는 정도 ↓, 예측성능 ↑, 시간 ↑, 최소 오류 값 찾는 어려움
 - 너무 클 경우: 예측성능 ↓, 시간 ↓
- n_estimators 와 같은 보통적으로 사용.
- subsample: weak learner 가 학습에 사용하는 데이터 샘플링 비율. (default: 1, 전체 학습 데이터를 이용)
 - 과적합을 방지하려고 보다 적은 값을 이용하기도.

06. XGBoost (extra gradient boost)

- 모의 기반의 양상을 학습에서 가장 각광 받고 있는 알고리즘.

- GBM 기반여지만 느린 수행시간, 과적합 규제 문제 등의 문제를 해결.
- 병렬 CPU 환경에서는 병렬 학습이 가능해 시간 학습. (상대적으로 빠르다는 의미)
- 분류, 회귀 모두 가능
- 자체적으로 과적합 규제 기능 가짐
 - 일반적으로 GBM은 부정 손실이 발생하면 분할을 더 이상 수행 X (즉나에게 많은 분할이 발생하기도..)
 - max_depth로 분할 깊이를 조정 + tree pruning 으로 긍정 이득이 없는 분할 가지치기 를 하여 분할 수를 줄인다.
- 자체 내장된 교차 검증: 최적화된 반복 수행
- 절손값 자체 처리 기능.

- ① 파이썬 라이브러리 XGBoost 모듈: 초기의 드라마적인 XGBoost 프로젝트 기반. (고급의 API, 하이퍼 파라미터 이용)
- ② 사이킷런 라이브러리 XGBoost 모듈: 사이킷런과 연동된다. (n_estimators와 사용법 동일)

① 파이썬 라이브러리 XGBoost와 하이퍼 파라미터

- ↳ GBM과 유사한 하이퍼 파라미터 + (조기 중단 / 과적합 규제)를 위한 하이퍼 파라미터 추가.
 - 일반 파라미터: 스레드의 개수, silent 모드선택, 주로 디폴트 값을 유지
 - 부스터 파라미터: 트리 회귀화, 부스팅, regularization 등
 - 학습 타스크 파라미터: 학습 수행 시의 객체 할당, 평가를 위한 지표 ..

1) 주요 일반 파라미터

- booster: gbtree (tree based model) / gbtlinear (linear model) 선택.
- silent: 디폴트는 0, 출력 X: 1
- nthread: CPU의 실행 스레드 개수를 조절. (디폴트: 모두 이용)

2) 주요 부스터 파라미터

- eta: 학습률에 해당. 파이썬 기본값 경우, 디폴트는 0.3. (0.6 ~ 0.2 이용)
- num_boost_rounds: n_estimators 와 동일.
- min_child_weight: 추가적으로 가치를 나눌지를 결정하기 위해 필요한 데이터들의 weight 총합.

값이 클수록 분할을 자제 / 과적합을 조절

- gamma: 트리의 리프 노드를 추가적으로 나눌지를 결정할 최소값. 값이 클수록 과적합 감소 효과 있음.
- colsample_bytree: max_feature 과 유사. 트리 생성에 필요한 피처를 몇으로 샘플링
- lambda: L2 Regularization 적용 값.] 피처 개수가 많을 경우, 적용을 걸통하고 값이 클수록 과적합 감소 효과가 있다.
- alpha: L1 "
- scale_pos_weight: 대칭, 균형을 유지.
- max_depth / sub_sample

3) 학습 타스크 파라미터

- objective: 최솟값을 가려야할 손실 함수. (손실함수: 이진분류 / 다중분류)

↳ binary: logistic : 이진분류

multi: softmax : 다중분류 (num_class를 지정해야 함.)

- multi: softprob : multi:softmax 와 유사하나 개별 레이블 클래스의 해당되는 예측 확률을 반환.
- eval_metric: 검증에 사용되는 함수. (default: 회귀일 경우: rmse, 분류일 경우: etar)

→ rmse / mae / logloss / error / merror / mlogloss / auc

<과적합이 심한 경우>

- eta를 낮춘다. num_round는 높여준다.
- max_depth를 ↓, min_child_weight를 ↑, gamma를 ↑
- subsample, colsample_tree를 조정.

XG Boost의 조기종단 (Early Stopping) 기능

: 부스팅을 반복하다 설정한 조기 종단 파라미터까지 학습 모수가 감소하지 않으면 부스팅을 중단.

- 워드로션 유방암 예측.

① import xgboost as xgb

```
from xgboost import Plot - Importance  
import pandas as pd  
import numpy as np  
data set / train - test - split 까지 로드
```

① 필요한 모듈/테이터 로드

② dataset = load_breast_cancer()

```
X_features = dataset.data  
y_label = dataset.target
```

② X, y 할당

③ X_train, X_test, y_train, y_test = train_test_split(X_features, y_label, test_size=0.2, random_state=156)

```
print(X_train.shape, X_test.shape)
```

③ 학습용, 테스트용 분할
80% 20%

④ dtrain = xgb.DMatrix(data=X_train, label=y_train) dtest = xgb.DMatrix(data=X_test, label=y_test)

④ DMatrix 설정 (주로 넘파이 형태)
XGBoost 전용 테이터 set.

⑤ params = {'max_depth': 3, # 트리 최대 깊이

'eta': 0.1, # 학습률

'objective': 'binary:logistic', # 예제 데이터가 0과 1이므로 이진분류

'eval_metric': 'log loss', # 평가할 속성의 평가 성능 지표

'early_stoppings': 100 # 조기종단 설정

}

⑤ 하이퍼파라미터 설정

num_rounds = 400 # 부스팅 반복 횟수

⑥ wlist = [(dtrain, 'train'), (dtest, 'eval')]

xgb_model = xgb.train(params=params,

⑥ early_stoppings_rounds를 이용하기 위해 설정

↳ eval_set 과 eval_metric 이 함께 설정.

XGBoost는 반복마다 eval_와 함께 설정된 데이터 세트에서 eval_metric의 지정된 평가지표로 예측 오류를 측정

eval_set은 성능 평가를 수행할 평가용 데이터, eval_metric: 평가세트에 적용할 방법 ex) error / logloss

* 사이킷런의 predict()는 예측 결과 클래스 값을 반환, Xgboost 의 predict()는 예측 결과를 추정할 수 있는 확률값을 반환한다.

④ XGBoost 의 예측 성능 평가

: get_clf_eval(y_test, preds, pred_probs)
plot_importance

⑤ 사이킷런 대비 XGBoost 의 개요 및 적용

: 사이킷런의 기본 Estimator를 이용.

↙ XGBClassifier
XGBRegressor

파라미터들: eta → learning_rate

sub_sample → Subsample

lambda → reg_lambda

alpha → reg_alpha

① from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=0.3)

xgb_wrapper.fit(X_train, y_train)

w_preds = xgb_wrapper.predict(X_test)

w_pred_proba = xgb_wrapper.predict_proba(X_test)[:, 1]

② from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)

evals = [(X_test, y_test)]

xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="logloss",
eval_set=evals, verbose=True)

ws100_preds = xgb_wrapper.predict(X_test)

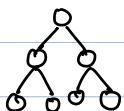
ws100_pred_proba = xgb_wrapper.predict_proba(X_test)[:, 1]

07. LightGBM

- XGBoost 보다 학습에 걸리는 시간이 훨씬 적다.
- 예측정밀도 good.
- 만건 이하의 작은 데이터셋의 경우, 과적합 되기 쉽다.
- 더 작은 memory 사용량, 카디고리컬 피처의 자동 변환과 최적 분할

일반 GBM 계열의 트리 분할

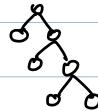
- 트리의 깊이를 효율적으로 줄이고자..
- 최대한 균형잡힌 tree → 시간↑



Light GBM 방식

VS

- 리프 중심 트리 분할 방식
- 균형을 맞추지 않고 최대 손실 값을 가지는 리프 노드를 지속적으로 분할 → 트리 깊어짐, 비약적
- 예측 오류 손실 최소화



LightGBM 하이퍼 파라미터

주요 파라미터

- num_iterations : 트리의 개수를 지정
- learning_rate, max_depth, min_data_in_leaf
- num_leaves : 하나의 트리가 가질 수 있는 최대 리프 개수
- boosting : 부스팅 트리를 생성하는 알고리즘 (gbdt(default), lgbf(엔SEMBLE드트))
- bagging_fraction : 레이어 Sampling 비율
- feature_fraction : 개별 트리를 학습할 때 무작위로 선택하는 피처의 비율
- lambda_l2, lambda_l1

Learning Task 파라미터

- objective: 최소값을 기려야하는 손실함수를 정의.

```
[1] from lightgbm import LGBMClassifier  
import pandas as pd  
import numpy as np  
from sklearn.datasets import load_breast_cancer  
from sklearn.model_selection import train_test_split  
dataset = load_breast_cancer()  
X = dataset.data  
y = dataset.target
```

이전 사이킷런과 동일

② $X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}} = \text{train_test_split}()$

`lgbm_wrapper = LGBMClassifier(n_estimators=400)`

`evals = [(X_test, y_test)]`

`lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="log loss",`

`eval_set=evals, verbose=True)`

`preds = lgbm_wrapper.predict(X_test)`

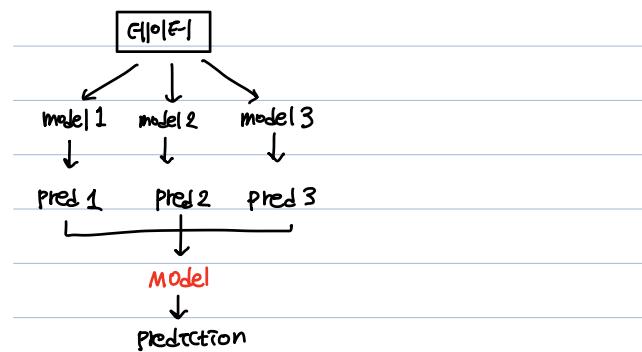
`pred_proba = lgbm_wrapper.predict_proba(X_test)[:, 1]`

10. 스태킹 앙상블

→ 개별 알고리즘으로 예측한 데이터 → 폐타임 데이터로 결합 → 별도의 ML 알고리즘으로 최종 학습.

model 1

model 2



① import numpy as np

KNN

RandomForestClassifier

AdaBoostClassifier

DecisionTreeClassifier

LogisticRegression ————— 최종 model

개별 model

③ 개별모델 학습 (fit)

④ 개별모델 예측 정확도 반환

⑤ 개별 예측값 칼럼 레벨 초기화 불여 피처

값은 상정

`pred = np.array([__개별 model__])`

`pred = np.transpose(pred)`

데이터셋 / train-test-split / accuracy-score 틀로 드.

② 개별 ML 모델 생성

`knn_clf =`

`rf_clf =`

`dt_clf =`

`ada_clf =`

최종 모델 생성

`lr_final = LogisticRegression()`

⑥ 위의 데이터로 최종 메타 모델인 로지스틱 회귀에 fit.

`lr_final.fit(pred, y_test)`

`final = lr_final.predict(pred)`

CV 세트 기반의 스태킹

: 과정함을 개선하고자 최종 메타 모델을 위한 데이터 set을 만들 때, 교차 검증 기반으로 예측된 결과를 이용

Step 1) 각 모델별로 원본 학습/테스트 데이터를 예측한 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터 생성

- 학습용 데이터를 3개의 폴드로 나눈다. (3개의 폴드)
- 학습용 데이터를 3개의 폴드로 나누되, 2개의 폴드는 학습을 위한 데이터 폴드 / 1개는 폴드 검증용.
→ 학습된 개별 모델은 검증 폴드 1개 데이터로 예측, 결과를 저장.
- 3번 동일하게 반복.
- 예측값의 평균으로 최종 결과값 생성.

Step 2) 메타 모델이 사용할 최종 학습 데이터, 원본 데이터의 레이블을 합쳐 최종 예측데이터