

# XGBoost / LightGBM

## XGBoost?

- 트리 기반의 알고리즘 앙상블 학습에서 각광받는 알고리즘 중 하나
- GBM에 기반하고 있지만, GBM의 단점인 느린 수행시간, 과적합 규제 부재 등을 해결한 알고리즘

### 장점

- 뛰어난 예측 성능
- 병렬 CPU 환경에서 병렬 학습이 가능하여 기존 GBM 대비 빠른 수행 시간
- 과적합 규제(overfitting Regularization) > 좀 더 강한 내구성을 가질 수 있음
- Tree pruning(나무 가지치기)  
> max\_depth parameter로 분할 깊이를 조정하기도 하지만, tree pruning으로 더 이상 긍정 이득이 없는 분할을 가지치기해서 분할 수를 더 줄이는 추가적인 장점을 가지고 있음
- 자체 내장된 교차 검증  
> 반복 수행 시마다 내부적으로 학습셋과 평가셋에 대한 교차 검증 수행하여 최적화된 반복 수행 횟수를 가질 수 있음, 지정된 반복 횟수가 아니라 교차 검증을 통해 평가셋의 평가 값이 최적화되면 반복을 중간에 멈출 수 있는 조기 중단 기능이 있음
- 결손값 자체 처리

하이퍼 파라미터는 유형별로 일반, 부스터, 학습 태스크 파라미터로 나뉨

### 일반 파라미터 (일반 설정)

- 일반적으로 실행 시 스레드 개수나 silent 모드 등의 선택을 위한 파라미터로 디폴터 값을 바꾸는 경우는 거의 없음
- booster(gbtree(default) 또는 gblinear 선택), silent(default 0), nthread(CPU 실행 스레드 개수 조정, default는 전체 사용)가 있음

### 부스터 파라미터 (모델 파라미터)

- learning\_rate [default = 0.1] : 학습 단계별로 이전 결과를 얼마나 반영할지 설정, 보통 0.01 ~ 0.2 사이 값, 너무 작게 하면 시간 오래
- n\_estimators [default = 100] : 트리 모델 갯수

## LightGBM?

- Gradient Boosting 프레임워크로 tree 기반 학습 알고리즘
  - GBM의 학습 방식을 쉽게 말하자면, 틀린 부분에 가중치를 더하면서 진행한다고 할 수 있음

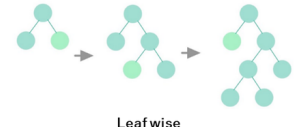
### 특징

- 다른 Tree 기반 알고리즘과 차이점은 확장 방향인데, 기존 GBM 계열 트리 기반 알고리즘은 XGBoost와 같이 level-wise(균형 트리 분할)로 Tree가 수평적으로 확장
  - 최대한 균형잡힌 트리를 유지하면서 분할하기 때문에 tree 깊이가 최소화될 수 있었음 == 균형을 맞추기 위한 시간이 필요하다
- LightGBM은 leaf-wise Tree 분할 방식으로 수직적으로 확장하는데, tree 균형을 고려하지 않고, 최대 손실 값(max data loss)을 가지는 leaf node를 지속적으로 분할하면서 tree 깊이가 깊어지고 비대칭적인 tree가 생성됨
- 이처럼 최대 손실값을 가지는 leaf node를 반복 분할하는 방식은 level wise tree 분할 방식보다 예측 오류 손실을 최소화할 수 있음

균형 트리 분할



리프 중심 트리 분할



### 장점 (XGBoost 대비)

- 빠른 학습 속도, 적은 메모리 사용
- 큰 스케일의 데이터를 핸들링할 수 있음
- categorical feature의 자동 변환과 최적 분할 (윗-하트 인코딩 사용하지 않고도 카테고리형 피쳐 최적으로 변환하고 이에 따라 노드 분할 수행)
- GPU에서 학습 가능

### 단점

- 작은 dataset을 사용할 경우 과적합 가능성이 큼 (일반적으로 10,000개 이하의 데이터를 적다고 함)

- min\_child\_weight [default = 1] : 트리에서 추가적으로 가지를 나눌지 결정하기 위한 데이터들의 가중치(weight) 최소화, 클수록 분할을 자제하며 과적합 조절을 위해 사용됨(단 너무 크게 설정하면 Under-fitting될 수 있음)
- max\_depth [default = 6] : 트리 최대 깊이, 보통은 3 ~ 10값 적용
- gamma [default = 0, alias : min\_split\_loss] : 트리 리프 노드를 추가적으로 나눌지 결정하는 최소 손실 감소 값, 값이 클수록 과적합 감소 효과
- subsample [default = 1] : 각 트리마다 데이터 샘플링 비율 over-fitting 방지, 일반적으로 0.5 ~ 1 사용
- colsample\_bytree [default = 1] : 각 트리마다 feature 샘플링 비율, 일반적으로 0.5 ~ 1 사용
- reg\_lambda [default = 1] : L2 규제 적용 값(Ridge), 피쳐 개수 많을 경우 적용 검토, 값이 클수록 과적합 감소 효과
- reg\_alpha [default = 0] : L1 규제 적용 값(Lasso), 피쳐 개수 많을 경우 적용 검토, 값이 클수록 과적합 감소 효과
- scale\_pos\_weight [default = 1] : 데이터가 불균형할 때 사용, 보통은 음성 데이터 수 / 양성 데이터 수 값으로 함

#### 학습 태스크 파라미터

- objective [default = reg:linear] : 최소값을 가져야할 손실 함수
  - binary:logistic : 이진 분류를 위한 로지스틱 회귀, 클래스가 아닌 예측 확률 반환
  - multi:softmax : softmax를 사용한 다중 클래스 분류, 클래스 반환
    - 손실함수가 해당 함수일때는 label 클래스 개수인 num\_class 파라미터를 지정해야 함
  - multi:softprob : softmax를 사용, 클래스에 대한 예상 확률 반환
- eval\_metric [목적함수에 따라 default가 다른 회귀 - rmse / 분류 - error] : 검증에 사용되는 함수
  - rmse : Root Mean Square Error
  - mae : Mean Absolute error
  - logloss : Negative log-likelihood
  - error : Binary classification error rate (임계값 0.5)
  - merror : multiclass classification error rate
  - mlogloss : multiclass logloss
  - auc : area under curve

#### 하이퍼 파라미터

- num\_iterations [default = 100] : 반복 수행하려는 트리 개수, 너무 크게 지정해도 과적합으로 성능이 저하될 수 있음 (사이킷런 호환 클래스의 n\_estimators와 같은 파라미터)
- learning\_rate [default = 0.1] : 학습률, 일반적으로 n\_estimators를 크게 하고 learning\_rate를 작게 해서 예측 성능을 향상시킬 수 있음
- max\_depth [default = -1] : 트리 최대 깊이 설정, lightgbm은 leaf wise 기반이므로 깊이가 상대적으로 더 깊음
- min\_data\_in\_leaf [default = 20] : 최종 결정 클래스인 리프 노드가 되기 위해 최소한으로 필요한 레코드 수이며 과적합을 제어하기 위한 파라미터
- num\_leaves [default = 31] : 하나의 트리가 가질 수 있는 최대 리프 개수
- boosting [default = gbdt] : 부스팅 트리를 생성하는 알고리즘 기술타
  - gbdt : 일반적인 그래디언트 부스팅 결정 트리 / rf : 랜덤포레스트
- bagging\_fraction [default = 1.0] : 과적합 제어를 위해 데이터를 샘플링하는 비율 지정
- feature\_fraction [default = 1.0] : 개별 트리 학습할 때마다 무작위로 선택하는 피쳐 비율, 과적합 제어용
- lambda\_l2 [default = 0] : L2 규제 적용 값, 피쳐 개수 많을 경우 적용 검토, 값이 클수록 과적합 감소 효과
- lambda\_l1 [default = 0] : L1 규제 적용 값, 클수록 과적합 감소 효과
- objective [default = regression] : 최소값을 가져야 할 손실함수 정의
  - regression : 회귀
  - binary : 이진분류
  - multiclass : 다중분류

## 테이블 정보 (temp\_new2)

Column Name	Description	Type	Length	Decimal	Note
USER_ID_hash	User ID	VARCHAR2	32		

REG_DATE	Registered date	DATE			Sign up date
SEX_ID	Gender	CHAR	1		f = female m = male
AGE	Age	NUMBER	4	0	
Tr_Pref_Name	Residential Prefecture	VARCHAR2	2		[KOR] Not registered if empty
Tr_small_area_name	Small area name of shop location	VARCHAR2	30		[KOR]
Translated_capsule_text	Capsule text	VARCHAR2	20		[KOR]
Translated_genre_name	Category name	VARCHAR2	50		[KOR]
VIEW_COUPON_ID_hash	Browsing Coupon ID	VARCHAR2	128		
usable_date_sum		CHAR			
view_count		INTEGER	1		
PRICE_RATE	Discount rate	NUMBER	4	0	
DISCOUNT_PRICE	Discount price	NUMBER	10	0	
VALIDPERIOD	Validity period (day)	NUMBER	4	0	
DISPPERIOD	Sales period (day)	NUMBER	4	0	
PURCHASE_FLG	Purchased flag	CHAR	1		0 : Not purchased  1 : Purchased
dispfrom	Sales release date	DATE			
dispend	Sales end date	DATE			
usable_date_mon	Is available on Monday	CHAR	1		
usable_date_tue	Is available on Tuesday	CHAR	1		
usable_date_wed	Is available on Wednesday	CHAR	1		
usable_date_thu	Is available on Thursday	CHAR	1		
usable_date_fri	Is available on Friday	CHAR	1		
usable_date_sat	Is available on Saturday	CHAR	1		
usable_date_sun	Is available on Sunday	CHAR	1		
usable_date_holiday	Is available on holiday	CHAR	1		
usable_date_before_holiday	Is available on the day before holiday	CHAR	1		
Male		CHAR	1		

Female		CHAR	1		
--------	--	------	---	--	--

## 파이썬 코드 진행 [↗](#)

### 파일 불러오기 [↗](#)

```

1 import pandas as pd
2 import numpy as np
3
4 df = pd.read_csv('/content/drive/MyDrive/000000000000/0000000000/0612_000000000000/data/temp_new2.csv')
5 df.head()
6
7 # 중복 데이터 제거
8 df2 = df.drop_duplicates().reset_index(drop = True)
9 df2.head()

```

### 데이터 전처리 [↗](#)

#### 날짜 데이터 컬럼 타입 변경 및 변수 생성 [↗](#)

```

1 df2['REG_DATE'] = pd.to_datetime(df2['REG_DATE'])
2 df2['YEAR'] = df2['REG_DATE'].dt.year
3 df2['MONTH'] = df2['REG_DATE'].dt.month
4 df2['DAY'] = df2['REG_DATE'].dt.day
5 df2.head()

```

#### x, y 데이터 추출 [↗](#)

```

1 y = df2['Translated_genre_name']
2 x = df2.drop(['Translated_genre_name', 'REG_DATE', 'Male', 'Female'], 1)

```

#### 컬럼 타입별로 리스트 생성 [↗](#)

컬럼은 랜덤포레스트 모델과 동일한 변수 사용 (랜덤포레스트 모델에서 변수 사이 상관관계 0.8 이상인 것을 제외한 남은 변수)

```

1 COL_DEL=['USER_ID_hash']
2 COL_DATE=['YEAR', 'MONTH', 'DAY']
3 COL_NUM=['PURCHASE_FLG', 'DISCOUNT_PRICE']
4 COL_CAT=['SEX_ID', 'Tr_Pref_Name', 'Tr_small_area_name']

```

#### 데이터 분할 [↗](#)

```

1 from sklearn.model_selection import train_test_split
2
3 # 전체에서 train, test 셋 분할
4 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42, stratify = y)
5
6 # train셋을 train, validation 셋으로 분할
7 x_tr, x_val, y_tr, y_val = train_test_split(x_train[COL_NUM + COL_CAT + COL_DATE], y_train, test_size = 0.2, rand

```

## 스케일링

x 변수(피쳐 데이터) - 구매여부와 가격은 단위가 동일하지 않으므로 단위 통일성을 주기 위해 진행

```
1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler()
4
5 x_tr[COL_NUM] = scaler.fit_transform(x_tr[COL_NUM])
6 x_val[COL_NUM] = scaler.transform(x_val[COL_NUM])
7 x_test[COL_NUM] = scaler.transform(x_test[COL_NUM])
```

## 인코딩

문자값으로 들어가있는 변수는 분석 진행에 어려움이 있어 인코딩 작업 진행

```
1 from sklearn.preprocessing import LabelEncoder
2
3 le = LabelEncoder()
4
5 X = pd.concat([x_train[COL_CAT], x_test[COL_CAT]])
6
7 for col in COL_CAT:
8     le.fit(X[col])
9     x_tr[col] = le.transform(x_tr[col])
10    x_val[col] = le.transform(x_val[col])
11    x_test[col] = le.transform(x_test[col])
```

## 모델링

먼저, y 변수값 인코딩 진행 :: y값이 문자형이면 오류 발생하여 y 변수(타겟 데이터) 라벨 인코딩 진행

```
1 from sklearn.preprocessing import LabelEncoder
2
3 label = LabelEncoder()
4
5 y_tr_ = label.fit_transform(y_tr)
6 y_val_ = label.transform(y_val)
7 y_test_ = label.transform(y_test)
```

### 1) XGBoost

```
1 from xgboost import XGBClassifier
2 from sklearn.metrics import accuracy_score, classification_report
3
4 # 모델 선언
5 # objective로 다중분류(multi:softmax)임을 선언했을 때는 num_class로 타겟 데이터 분류 개수를 설정해줘야 함
6 xgb_clf = XGBClassifier(random_state = 42, objective = 'multi:softmax', num_class = 13)
7
8 # 모델 학습
9 xgb_clf.fit(x_tr, y_tr_)
10
11 # 예측
12 xgb_pred = xgb_clf.predict(x_val)
13
```

```

14 # 정확도 확인
15 xgb_accuracy = accuracy_score(y_val_, xgb_pred)
16
17 # 클래스별 분류보고서 생성
18 xgb_report = classification_report(y_val_, xgb_pred)
19
20 print('xgb_accuracy :', xgb_accuracy)
21 print('xgb_report :\n', xgb_report)
22
23
24 # 변수 중요도 확인 및 시각화
25 import xgboost
26 import matplotlib.pyplot as plt
27 xgboost.plot_importance(xgb_clf, xlabel = '')

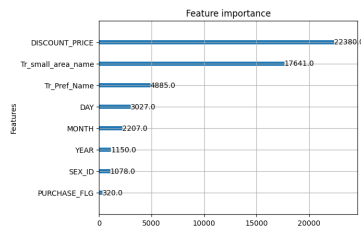
```

```

xgb_accuracy : 0.8092555582398114
xgb_report :

```

	precision	recall	f1-score	support
0	0.83	0.66	0.73	295
1	0.84	0.58	0.68	19376
2	0.61	0.78	0.69	7197
3	0.62	0.29	0.39	2781
4	0.89	0.61	0.72	2972
5	0.83	0.74	0.78	8665
6	0.41	0.15	0.22	199
7	0.66	0.15	0.24	3247
8	0.75	0.80	0.77	66505
9	0.81	0.87	0.84	87600
10	0.60	0.45	0.51	4733
11	0.91	0.96	0.94	64284
12	0.78	0.20	0.32	5144
accuracy			0.81	273198
macro avg	0.73	0.56	0.60	273198
weighted avg	0.81	0.81	0.80	273198



## 1-1) 오버샘플링 진행 후 XGBoost

```

1 from imblearn.over_sampling import SMOTE
2
3 # 오버샘플링 객체 생성
4 smote = SMOTE(random_state = 42)
5
6 # 샘플링 적용
7 x_tr_over, y_tr_over = smote.fit_resample(x_tr, y_tr_)
8 -----
9 # 모델 선언
10 xgb_clf_over = XGBClassifier(random_state = 42, objective = 'multi:softmax', num_class = 13)
11
12 # 모델 학습
13 xgb_clf_over.fit(x_tr_over, y_tr_over)
14
15 # 예측
16 xgb_pred_over = xgb_clf_over.predict(x_val)
17
18 # 정확도 확인
19 xgb_accuracy = accuracy_score(y_val_, xgb_pred_over)
20
21 # 클래스별 분류보고서 생성

```

```

22 xgb_report = classification_report(y_val_, xgb_pred_over)
23
24 print('xgb_accuracy :', xgb_accuracy)
25 print('xgb_report :\n', xgb_report)
26
27
28 # 변수 중요도 확인 및 시각화
29 import xgboost
30 import matplotlib.pyplot as plt
31 xgboost.plot_importance(xgb_clf_over, xlabel = '')

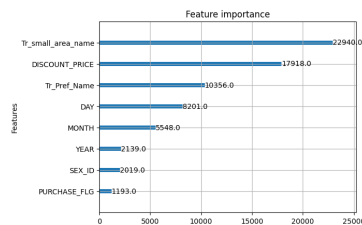
```

```

xgb_accuracy : 0.6711359526790094
xgb_report :

```

	precision	recall	f1-score	support
0	0.07	0.77	0.13	295
1	0.57	0.55	0.56	19376
2	0.45	0.92	0.61	7197
3	0.18	0.55	0.27	2781
4	0.32	0.67	0.44	2972
5	0.42	0.80	0.55	8665
6	0.03	0.54	0.05	199
7	0.15	0.27	0.19	3247
8	0.75	0.59	0.66	66605
9	0.84	0.64	0.73	87600
10	0.28	0.51	0.36	4733
11	0.93	0.85	0.89	64284
12	0.17	0.26	0.20	5144
accuracy			0.67	273198
macro avg	0.40	0.61	0.43	273198
weighted avg	0.75	0.67	0.70	273198



	오버샘플링 전	오버샘플링 후
xgboost 모델 정확도	0.81	0.67

## 2) LightGBM [↗](#)

발표 자료에는 생략

```

1 import lightgbm as lgb
2 from sklearn.metrics import accuracy_score
3
4 # 모델 선언
5 lgb_clf = lgb.LGBMClassifier(application = 'multiclass', random_state = 42)
6
7 # 모델 학습
8 lgb_clf.fit(x_tr, y_tr_)
9
10 # 예측
11 lgb_pred = lgb_clf.predict(x_val)
12
13 # 정확도 확인
14 lgb_accuracy = accuracy_score(y_val_, lgb_pred)
15
16 # 클래스별 분류보고서 생성

```

```

17 xgb_report = classification_report(y_val_, lgb_pred)
18
19 print('lgb_accuracy :', lgb_accuracy)
20 print('lgb_report :\n', lgb_report)
21
22
23 # 변수 중요도 확인 및 시각화
24 import lightgbm
25 import matplotlib.pyplot as plt
26 lightgbm.plot_importance(lgb_clf)

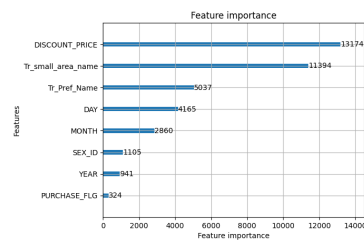
```

```

lgb_accuracy : 0.7368941207475896
lgb_report :

```

	precision	recall	f1-score	support
0	0.19	0.28	0.23	295
1	0.81	0.46	0.58	19376
2	0.59	0.75	0.66	7197
3	0.45	0.29	0.35	2781
4	0.75	0.56	0.64	2372
5	0.77	0.63	0.69	8865
6	0.00	0.14	0.01	199
7	0.51	0.13	0.20	3247
8	0.69	0.69	0.69	66505
9	0.75	0.80	0.77	87600
10	0.57	0.36	0.44	4733
11	0.87	0.93	0.90	64284
12	0.66	0.16	0.26	5144
accuracy			0.74	273198
macro avg	0.59	0.48	0.50	273198
weighted avg	0.75	0.74	0.74	273198



## 2-1) 오버샘플링 진행 후 LightGBM

```

1 # 오버샘플링 진행은 xgboost 모델 진행 전 실행한 데이터 이용
2
3 # 모델 선언
4 import lightgbm as lgb
5 lgb_clf_over = lgb.LGBMClassifier(application = 'multiclass', random_state = 42)
6
7 # 모델 학습
8 lgb_clf_over.fit(x_tr_over, y_tr_over)
9
10 # 예측
11 lgb_pred_over = lgb_clf_over.predict(x_val)
12
13 # 정확도 확인
14 lgb_accuracy = accuracy_score(y_val_, lgb_pred_over)
15
16 # 클래스별 분류보고서 생성
17 lgb_report = classification_report(y_val_, lgb_pred_over)
18
19 print('lgb_accuracy :', lgb_accuracy)
20 print('lgb_report :\n', lgb_report)
21
22
23 # 변수 중요도 확인 및 시각화
24 import lightgbm

```

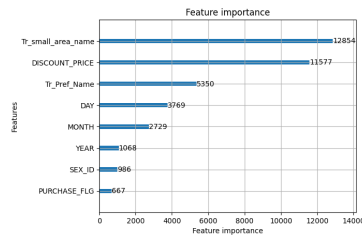


```
25 import matplotlib.pyplot as plt
26 lightgbm.plot_importance(lgb_clf_over)
```


```
lgb_accuracy : 0.6182878351964509
lgb_report :
      precision    recall  f1-score   support

     0       0.05      0.76      0.10        295
     1       0.51      0.52      0.51       19376
     2       0.43      0.92      0.59        7197
     3       0.16      0.57      0.25        2781
     4       0.28      0.66      0.39        2972
     5       0.34      0.77      0.47        8865
     6       0.02      0.55      0.05         199
     7       0.12      0.23      0.16        3247
     8       0.72      0.53      0.61       66505
     9       0.81      0.57      0.67       87600
    10       0.24      0.49      0.32        4733
    11       0.91      0.80      0.85       64284
    12       0.14      0.24      0.18        5144

 accuracy          0.62      273198
 macro avg         0.36      0.59      0.40      273198
 weighted avg      0.72      0.62      0.65      273198
```



	오버샘플링 전	오버샘플링 후
lightgbm 모델 정확도	0.74	0.62

 xgboost와 lightgbm 두 모델을 실행해본 결과, 오버샘플링 전후 결과를 모두 보면 xgboost 모델의 정확도가 더 좋게 나온 것을 볼 수 있었다.

## 개선 사항 및 보완점 [↗](#)

- 특정 카테고리 항목으로 치우친 데이터 조정
- 하이퍼 파라미터 튜닝하여 예측 성능 향상
  - 분류 보고서 결과를 토대로 과대적합을 완화시키기 위해 오버샘플링 방법만 진행해보았는데, 하이퍼 파라미터 튜닝 작업까지 진행 후 결과 도출