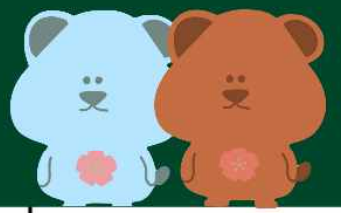


4주차 발표

조혜빈 차수빈 이가영

EWHA EURON

목차

EWHA EURON

01 산탄데르 고객 만족 예측

EWHA EURON

1.1 대회 소개

산탄데르 은행에서 고객의 만족도를 높이기 위해 1:1 맞춤 금융 상품을 추천하는 머신러닝 알고리즘을 사용하고자 함

프로젝트 목적

•자사의 금융 서비스를 이용하는 고객들의 특성 분석

•자사의 고객들을 대상으로 고객 맞춤형 상품 추천을 제공

=> 고객의 과거 이력과 유사한 고객군들의 데이터를 기반으로 다음달에 해당 고객이 무슨 상품을 사용할지 예측

=> 고객의 만족도를 높임과 동시에 은행 매출에 기여

✓ 만족/불만족한 고객의 데이터를 분류하는 이진 분류 문제

1.2 data description

피처 & 클래스

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings('ignore')

cust_df = pd.read_csv("./train_santander.csv", encoding='latin-1')
print('dataset shape:', cust_df.shape)
cust_df.head(3)
```

```
dataset shape: (76020, 371)
```

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op
0	1	2	23	0.0	0.0	0.0	
1	3	2	34	0.0	0.0	0.0	
2	4	2	23	0.0	0.0	0.0	

3 rows × 371 columns

피처 타입

```
cust_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76020 entries, 0 to 76019
Columns: 371 entries, ID to TARGET
dtypes: float64(111), int64(260)
memory usage: 215.2 MB
```

클래스 분포

```
print(cust_df['TARGET'].value_counts())
unsatisfied_cnt = cust_df[cust_df['TARGET'] == 1].TARGET.count()
total_cnt = cust_df.TARGET.count()
print('unsatisfied 비율은 {0:.2f}'.format((unsatisfied_cnt / total_cnt)))
```

```
0    73012
1     3008
Name: TARGET, dtype: int64
unsatisfied 비율은 0.04
```

불만족의 비율이 0.04로 훨씬 낮음
Imbalance한 데이터

EWHA EURON

1.3 핵심 모델링/아이디어

XGBoost 학습 모델

XGBoost: GBM에 기반해서 느린 수행 시간 및 과적합 규제 부재 등의 단점을 보완한
분류에 있어서 뛰어난 예측 성능을 나타내는 알고리즘

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score

# n_estimators는 500으로, random state는 예제 수행 시마다 동일 예측 결과를 위해 설정.
xgb_clf = XGBClassifier(n_estimators=500, random_state=156)

# 성능 평가 지표를 auc로, 조기 중단 파라미터는 100으로 설정하고 학습 수행.
xgb_clf.fit(X_train, y_train, early_stopping_rounds=100,
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])
```

n_estimators=500
early_stopping_rounds=100
eval_metric=auc

EWHA EURON

1.3 핵심 모델링/아이디어

LightGBM 학습 모델

LightGBM: XGBoost와 함께 부스팅 계열 알고리즘에 속하며

XGBoost보다 빠른 학습과 예측 수행 시간, 더 작은 메모리 사용량을 자랑하는 알고리즘

```
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=500)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
             verbose=True)
```

n_estimators=500
early_stopping_rounds=100
eval_metric=auc

EWHA EURON

1.3 핵심 모델링/아이디어

early_stopping_rounds

- 사이킷런 래퍼 XGBoost에서 조기 중단 관련 파라미터 중 하나
  평가 지표가 향상될 수 있는 반복 횟수를 정의
  (ex. early_stopping_rounds=100: 조기 중단할 수 있는 최소 반복 횟수가 100)

- 조기 중단 값을 너무 급격하게 줄이면 성능이 향상될 가능성이 있음에도 반복이 멈춰버려서
  충분한 학습이 되지 않아 예측 성능이 저하될 수 있음

```
[243]   validation_0-auc:0.91387        validation_1-auc:0.83197
[244]   validation_0-auc:0.91395        validation_1-auc:0.83204
[245]   validation_0-auc:0.91402        validation_1-auc:0.83196
ROC AUC: 0.8429
```

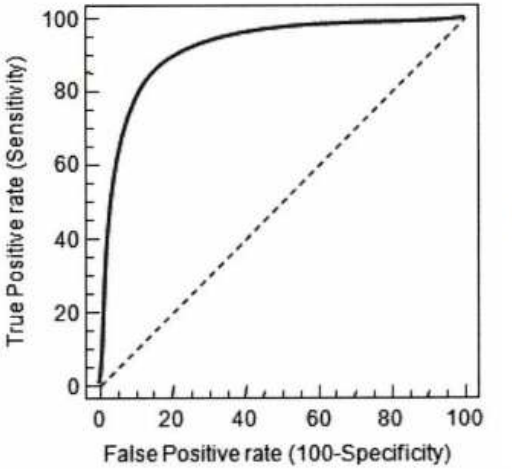
n_estimators=5000이지만
245번 반복한 후 학습을 완료함

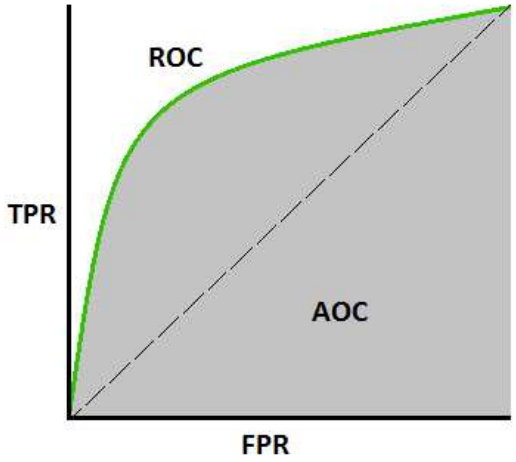
EWHA EURON

1.3 핵심 모델링/아이디어

ROC 곡선과 AUC

- ROC 곡선: FPR이 변할 때 TPR이 어떻게 변하는지 나타내는 곡선
- AUC: ROC 곡선의 넓이





		실제 정답	
		True (Positive)	False (Negative)
분류 결과	True (Positive)	TP (True Positive)	FP (False Positive)
	False (Negative)	FN (False Negative)	TN (True Negative)

FPR: False Positive Rate. 실제 값이 negative인데 예측 값은 positive
TPR: True Postive Rate. 실제 값과 예측 값 모두 positive

- ✓ ROC 곡선이 가운데 직선에서 멀어질수록 성능이 뛰어남
- ✓ AUC는 1에 가까울수록 좋은 수치
- ✓ AUC가 커지려면 FPR이 작은 상태에서 얼마나 큰 TPR을 얻을 수 있는지가 관건

EWHA EURON

1.4 코드/결과 분석

data preprocessing

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1
count	76020.000000	76020.000000	76020.000000	76020.000000	76020.000000
mean	75964.050723	-1523.199277	33.212865	86.208265	72.363067
std	43781.947379	39033.462364	12.956486	1614.757313	339.315831
min	1.000000	-999999.000000	5.000000	0.000000	0.000000
25%	38104.750000	2.000000	23.000000	0.000000	0.000000
50%	76043.000000	2.000000	28.000000	0.000000	0.000000
75%	113748.750000	2.000000	40.000000	0.000000	0.000000
max	151838.000000	238.000000	105.000000	210000.000000	12888.030000

8 rows × 371 columns

```
cust_df['var3'].value_counts()
```

```
2          74165
8            138
-999999      116
9            110
3            108
           ...
231            1
188            1
168            1
135            1
87             1
Name: var3, Length: 208, dtype: int64
```

일괄적 업데이트한 것으로 추정

```
# var3 피처 값 대체 및 ID 피처 드롭
cust_df['var3'].replace(-999999, 2, inplace=True)
cust_df.drop('ID', axis=1, inplace=True)

# 피처 세트와 레이블 세트분리. 레이블 컬럼은 DataFrame의 맨 마지막에 위치해 컬럼 위치
X_features = cust_df.iloc[:, :-1]
y_labels = cust_df.iloc[:, -1]
print('피처 데이터 shape:{0}'.format(X_features.shape))
```

다른 값과 차이가 많이 나기 때문에
-999999을 최빈값 2로 변환

EWHA EURON

1.4 코드/결과 분석

XGBoost 하이퍼 파라미터 튜닝

```
xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

【Output】

```
ROC AUC: 0.8419
```

```
from sklearn.model_selection import GridSearchCV

# 하이퍼 파라미터 테스트의 수행 속도를 향상시키기 위해 n_estimators를 100으로 감소
xgb_clf = XGBClassifier(n_estimators=100)

params = {'max_depth':[5, 7], 'min_child_weight':[1, 3], 'colsample_bytree':[0.5, 0.75] }

# cv는 3으로 지정
gridcv = GridSearchCV(xgb_clf, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="auc",
           eval_set=[(X_train, y_train), (X_test, y_test)])

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)

xgb_roc_score = roc_auc_score(y_test, gridcv.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

```
GridSearchCV 최적 파라미터: {'colsample_bytree': 0.75, 'max_depth': 7, 'min_child_weight': 1}
ROC AUC: 0.8448
```

```
# n_estimators는 1000으로 증가시키고, learning_rate=0.02로 감소, reg_alpha=0.03으로 추가함.
xgb_clf = XGBClassifier(n_estimators=1000, random_state=156, learning_rate=0.02, max_depth=7,
                        min_child_weight=1, colsample_bytree=0.75, reg_alpha=0.03)

# 성능 평가 지표를 auc로, 조기 중단 파라미터 값은 200 으로 설정하고 학습 수행.
xgb_clf.fit(X_train, y_train, early_stopping_rounds=200,
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])

xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

【Output】

```
ROC AUC: 0.8456
```

성능이 0.037 향상됨

max_depth: 트리의 최대 깊이
min_child_weight: 분할을 결정하는 데이터들의 weight 총합
colsampe_bytree: GBM의 max_feature 최대 피처 갯수

EWHA EURON

1.4 코드/결과 분석

LightGBM 하이퍼 파라미터 튜닝

```
lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

【Output】

```
ROC AUC: 0.8396
```

```
from sklearn.model_selection import GridSearchCV

# 하이퍼 파라미터 테스트의 수행 속도를 향상시키기 위해 n_estimators를 200으로 감소
lgbm_clf = LGBMClassifier(n_estimators=200)

params = {'num_leaves': [32, 64 ],
          'max_depth':[128, 160],
          'min_child_samples':[60, 100],
          'subsample':[0.8, 1]}

# cv는 3으로 지정
gridcv = GridSearchCV(lgbm_clf, param_grid=params, cv=3)
gridcv.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="auc",
           eval_set=[(X_train, y_train), (X_test, y_test)])

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)
lgbm_roc_score = roc_auc_score(y_test, gridcv.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

```
GridSearchCV 최적 파라미터: {'max_depth': 128, 'min_child_samples': 100, 'num_leaves': 32,
'subsample': 0.8}
ROC AUC: 0.8442
```

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=32, sumbsample=0.8, min_child_samples=100,
                          max_depth=128)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
             verbose=True)

lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

【Output】

```
ROC AUC: 0.8442
```

성능이 0.046 향상됨

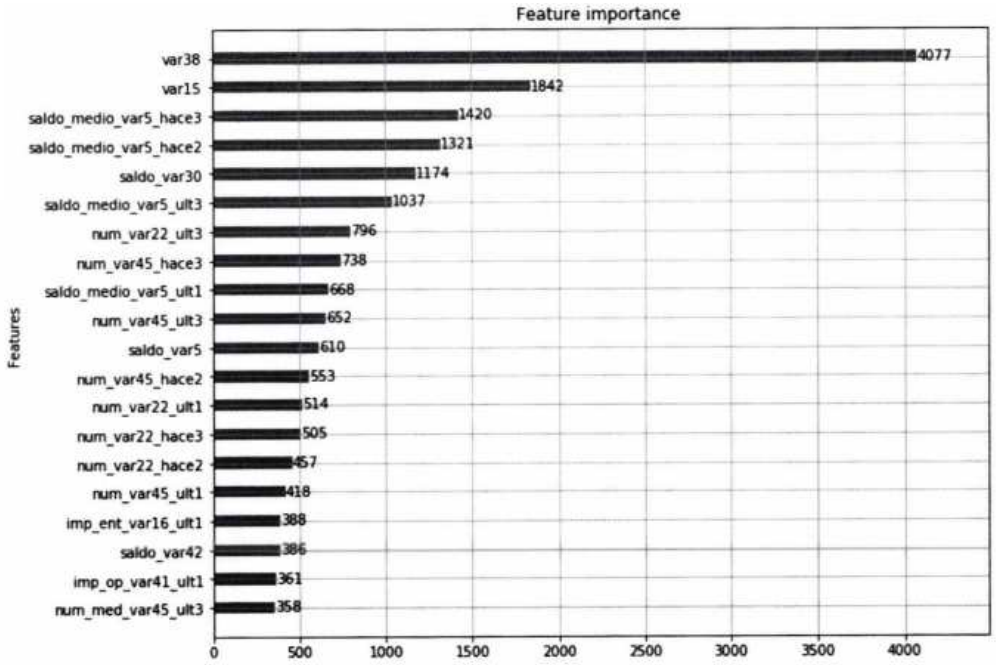
max_depth: 트리의 최대 깊이
num_leaves: 하나의 트리가 가질 수 있는 최대 리프 개수
min_child_samples: 최종 리프 노드가 되기 위한 레코드 수
subsample: 데이터 샘플링 비율

EWHA EURON

1.4 코드/결과 분석

각 피처의 중요도

```
from xgboost import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
plot_importance(xgb_clf, ax=ax, max_num_features=20, height=0.4)
```



Var 38, var 15 순으로 XGBoost의 성능을 좌우함

1.5 결론

1.target이 만족/불만족인 이진분류 문제

```
print(cust_df['TARGET'].value_counts())
unsatisfied_cnt = cust_df[cust_df['TARGET'] == 1].TARGET.count()
total_cnt = cust_df.TARGET.count()
print('unsatisfied 비율은 {0:.2f}'.format((unsatisfied_cnt / total_cnt)))
```

```
0    73012
1     3008
Name: TARGET, dtype: int64
unsatisfied 비율은 0.04
```

2. target의 분포가 imbalance한 data

```
# var3 피처 값 대체 및 ID 피처 드롭
cust_df['var3'].replace(-999999, 2, inplace=True)
cust_df.drop('ID', axis=1, inplace=True)

# 피처 세트와 레이블 세트분리. 레이블 컬럼은 DataFrame의 맨 마지막에 위치해 컬럼 위치
X_features = cust_df.iloc[:, :-1]
y_labels = cust_df.iloc[:, -1]
print('피처 데이터 shape:{0}'.format(X_features.shape))
```

3. GridSearchCV 이용해서 성능 향상

```
# n_estimators는 1000으로 증가시키고, learning_rate=0.02로 감소, reg_alpha=0.03으로 추가함.
xgb_clf = XGBClassifier(n_estimators=1000, random_state=156, learning_rate=0.02, max_depth=7,
                        min_child_weight=1, colsample_bytree=0.75, reg_alpha=0.03)

# 성능 평가 지표를 auc로, 조기 중단 파라미터 값은 200 으로 설정하고 학습 수행.
xgb_clf.fit(X_train, y_train, early_stopping_rounds=200,
            eval_metric="auc", eval_set=[(X_train, y_train), (X_test, y_test)])

xgb_roc_score = roc_auc_score(y_test, xgb_clf.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(xgb_roc_score))
```

[Output]

```
ROC AUC: 0.8456
```

4. lightGBM보다 XGBoost의 성능이 조금 더 높음

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=32, sumbsample=0.8, min_child_samples=100,
                          max_depth=128)

evals = [(X_test, y_test)]
lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="auc", eval_set=evals,
             verbose=True)

lgbm_roc_score = roc_auc_score(y_test, lgbm_clf.predict_proba(X_test)[:, 1], average='macro')
print('ROC AUC: {0:.4f}'.format(lgbm_roc_score))
```

[Output]

```
ROC AUC: 0.8442
```

02 MoA 예측

EWHA EURON

2.1 대회 소개

하버드 연구실에서 신약에 대한 MoA를 예측하는 알고리즘을 발전시키고자 함

(MoA: Mechanism of Action. 약물이 작용하는 메커니즘)

프로젝트 목적

•유전자 발현 데이터 및 세포 생존율 데이터와 같은 다양한 입력이 주어지면 MoA를 예측

•MoA 예측 알고리즘의 개선을 통해 약물 개발을 발전

✓약물에는 여러 MoA가 있으므로 다중 레이블 분류 문제

2.2 data description

```
train_features.head()
```

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	...	c-90	c-91	c-92	c
0	id_000644bb2	trt_cp	24	D1	1.0620	0.5577	-0.2479	-0.6208	-0.1944	-1.0120	...	0.2862	0.2584	0.8076	0.5
1	id_000779bfc	trt_cp	72	D1	0.0743	0.4087	0.2991	0.0604	1.0190	0.5207	...	-0.4265	0.7543	0.4708	0.0
2	id_000a6266a	trt_cp	48	D1	0.6280	0.5817	1.5540	-0.0764	-0.0323	1.2390	...	-0.7250	-0.6297	0.6103	0.0
3	id_0015fd391	trt_cp	48	D1	-0.5138	-0.2491	-0.2656	0.5288	4.0620	-0.8095	...	-2.0990	-0.6441	-5.6300	-1.3
4	id_001626bd3	trt_cp	72	D2	-0.3254	-0.4009	0.9700	0.6919	1.4180	-0.8244	...	0.0042	0.0048	0.6670	1.0

5 rows × 876 columns

유전자 발현 데이터 및 세포 생존율 데이터

```
targets.head()
```

	sig_id	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylcholine_receptor_agonist	acetylcholine_recep
0	id_000644bb2	0	0	0	0	
1	id_000779bfc	0	0	0	0	
2	id_000a6266a	0	0	0	0	
3	id_0015fd391	0	0	0	0	
4	id_001626bd3	0	0	0	0	

5 rows × 207 columns

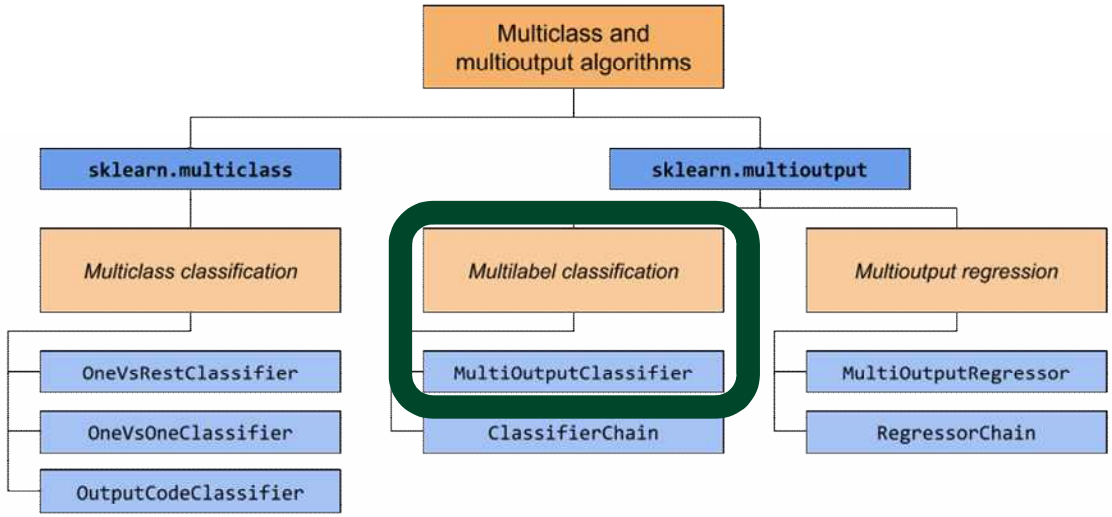
Mechanism of Action (MoA) responses
Y의 클래스가 206개 존재 -> 다중 분류

EWHA EURON

2.3 핵심 모델링/아이디어

다중 분류

✓클래스가 3개 이상인 분류



다중 클래스 문제 해결을 위해 sklearn이 제공하는 도구

2.3 핵심 모델링/아이디어

multi-label classification

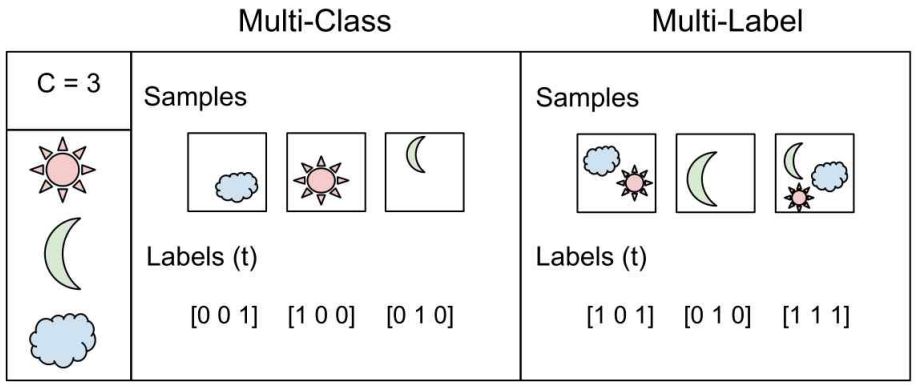
Multi-label classification이란?

3개 이상의 클래스를 구별하는 다중 분류 중에서도

하나의 샘플이 하나의 분류에만 속하는 것이 아니라 여러 개의 분류에 속하는 문제들

(ex. 영화 <토이스토리>는 애니메이션이면서 코미디 영화로 분류할 수 있음)

✓분류 체계가 상호 배타적이지 않다는 것이 특징!



✓Multi-class 와 multi-label의 차이점

2.3 핵심 모델링/아이디어

multi-label 문제에서 라벨의 형태

```
import pandas as pd

columns = ["SF", "Thriller", "Animation", "Comedy", "Kids"]

index = ["The Dark Knight", "Toy Story"]

dark_knight = [1, 1, 0, 0, 0]

toy_story = [0, 0, 1, 1, 1]

y = pd.DataFrame([dark_knight, toy_story], columns = columns, index =
index)
```

	SF	Thriller	Animation	Comedy	Kids
The Dark Knight	1	1	0	0	0
Toy Story	0	0	1	1	1

•n개의 클래스만큼의 0과 1로 이루어진 매트릭스

•n개의 클래스에 각각 이진 분류 작업을 실행하는 것

2.3 핵심 모델링/아이디어

MultiOutputClassifier

- 타겟 당 하나의 분류기를 학습시키는 전략
- 타겟(y1,y2,y3…,yn)을 예측하기 위해 예측 함수(f1,f2,f3…,fn)를 측정
- 각 레이블을 독립적으로 처리함

```
classifier = MultiOutputClassifier(XGBClassifier(tree_method='gpu_hist'))

clf = Pipeline([('encode', CountEncoder(cols=[0, 2])),
                ('classify', classifier)
               ])
```

추정기로 XGBClassifier를 사용한 다음
MultiOutputClassifier 클래스에 포함

```
params = {'classify__estimator__colsample_bytree': 0.6522,
          'classify__estimator__gamma': 3.6975,
          'classify__estimator__learning_rate': 0.0503,
          'classify__estimator__max_delta_step': 2.0706,
          'classify__estimator__max_depth': 10,
          'classify__estimator__min_child_weight': 31.5800,
          'classify__estimator__n_estimators': 166,
          'classify__estimator__subsample': 0.8639
         }

_ = clf.set_params(**params)
```

하이퍼 파라미터는 XGBoost와 동일

EWHA EURON

2.4 코드/결과 분석

modeling & evaluation

```
oof_preds = np.zeros(y.shape)
test_preds = np.zeros((test.shape[0], y.shape[1]))
oof_losses = []
kf = KFold(n_splits=NFOLDS)
for fn, (trn_idx, val_idx) in enumerate(kf.split(X, y)):
    print('Starting fold: ', fn)
    X_train, X_val = X[trn_idx], X[val_idx]
    y_train, y_val = y[trn_idx], y[val_idx]

    # drop where cp_type==ctl_vehicle (baseline)
    ctl_mask = X_train[:,0]=='ctl_vehicle'
    X_train = X_train[~ctl_mask,:]
    y_train = y_train[~ctl_mask]

    clf.fit(X_train, y_train)
    val_preds = clf.predict_proba(X_val) # list of preds per class
    val_preds = np.array(val_preds)[:,:,1].T # take the positive class
    oof_preds[val_idx] = val_preds

    loss = log_loss(np.ravel(y_val), np.ravel(val_preds))
    oof_losses.append(loss)
    preds = clf.predict_proba(X_test)
    preds = np.array(preds)[:,:,1].T # take the positive class
    test_preds += preds / NFOLDS
```

모델 성능 평가 지표: logloss

predict_proba: 불확실성을 추정하는 함수

각 샘플에 대해 어느 클래스에 속할지

그 확률을 0에서 1 사이의 값으로 돌려줌

```
# predict_proba 결과 중 앞부분 일부를 확인합니다
print("Predicted probabilities:\n{}".format(
      gbrt.predict_proba(X_test[:6])))
# Predicted probabilities:
# [[0.01573626 0.98426374]
#  [0.84575649 0.15424351]
#  [0.98112869 0.01887131]
#  [0.97406775 0.02593225]
#  [0.01352142 0.98647858]
#  [0.02504637 0.97495363]]
```

(예시)
[첫번째 클래스의 예측 확률, 두번째 클래스의 예측 확률]
합은 항상 1

EWHA EURON

2.4 코드/결과 분석

logloss

- 분류 모델 성능 평가 시 사용 가능한 지표
- 모델이 얼마의 확률을 가지고 예측했는지에 초점을 둠
- predict_proba가 계산해 준 확률과 음의 로그함수를 이용 값을 계산. 작을수록 좋은 모델

logloss 평균 =

$$\frac{\text{1번 문제 정답 확률에 대한 음의 로그 값} + \cdots\cdots + \text{n번 문제 정답 확률에 대한 음의 로그 값}}{\text{n (문제의 개수)}}$$

①고양이
②강아지
③팬더
④사자
⑤치타

	<모델 1>	<모델 2>
①	0.8	0.4
②	0.15	0.2
③	0.05	0.3
④	0	0.1
⑤	0	0

정답을 더 높은 확률로 예측할수록
좋은 모델이라고 평가

EWHA EURON

2.4 코드/결과 분석

modeling & evaluation

```
oof_preds = np.zeros(y.shape)
test_preds = np.zeros((test.shape[0], y.shape[1]))
oof_losses = []
kf = KFold(n_splits=NFOLDS)
for fn, (trn_idx, val_idx) in enumerate(kf.split(X, y)):
    print('Starting fold: ', fn)
    X_train, X_val = X[trn_idx], X[val_idx]
    y_train, y_val = y[trn_idx], y[val_idx]

    # drop where cp_type==ctl_vehicle (baseline)
    ctl_mask = X_train[:,0]=='ctl_vehicle'
    X_train = X_train[~ctl_mask,:]
    y_train = y_train[~ctl_mask]

    clf.fit(X_train, y_train)
    val_preds = clf.predict_proba(X_val) # list of preds per class
    val_preds = np.array(val_preds)[:,:,1].T # take the positive class
    oof_preds[val_idx] = val_preds

    loss = log_loss(np.ravel(y_val), np.ravel(val_preds))
    oof_losses.append(loss)
    preds = clf.predict_proba(X_test)
    preds = np.array(preds)[:,:,1].T # take the positive class
    test_preds += preds / NFOLDS
```

```
Starting fold:  0
Starting fold:  1
Starting fold:  2
Starting fold:  3
Starting fold:  4
[0.0169781773377249, 0.01704491710861325, 0.016865153552168475, 0.01700900926983899, 0.01717
882474706338]
Mean OOF loss across folds 0.017015216403081797
STD OOF loss across folds 0.00010156682747757948
```

K=5인 K 폴드 교차 검증을 했기 때문에 5개의 loss가 나옴

5개의 손실 값 평균은 0.017

EWHA EURON

2.5 결론

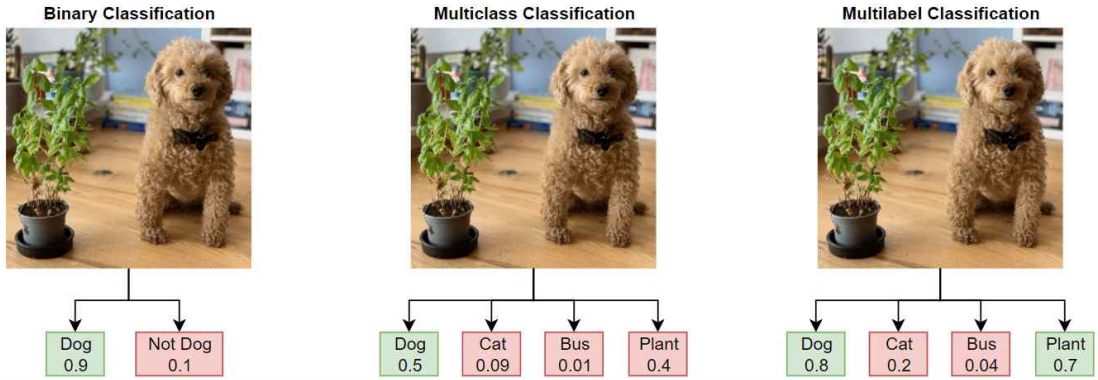
1. 타겟의 클래스가 206개 존재하는 다중분류 문제

```
targets.head()
```

	sig_id	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylcholine_receptor_agonist	ac
0	id_000644bb2	0	0	0	0	
1	id_000779bfc	0	0	0	0	
2	id_000a6266a	0	0	0	0	
3	id_0015fd391	0	0	0	0	
4	id_001626bd3	0	0	0	0	

5 rows × 207 columns

2. Multilabel classification



3. predict_proba를 이용한 확률 계산 & logloss

```
oof_preds = np.zeros(y.shape)
test_preds = np.zeros((test.shape[0], y.shape[1]))
oof_losses = []
kf = KFold(n_splits=NFOLDS)
for fn, (trn_idx, val_idx) in enumerate(kf.split(X, y)):
    print('Starting fold: ', fn)
    X_train, X_val = X[trn_idx], X[val_idx]
    y_train, y_val = y[trn_idx], y[val_idx]

    # drop where cp_type==ctl_vehicle (baseline)
    ctl_mask = X_train[:,0]=='ctl_vehicle'
    X_train = X_train[~ctl_mask,:]
    y_train = y_train[~ctl_mask]

    clf.fit(X_train, y_train)
    val_preds = clf.predict_proba(X_val) # list of preds per class
    val_preds = np.array(val_preds)[:,:,1].T # take the positive class
    oof_preds[val_idx] = val_preds

    loss = log_loss(np.ravel(y_val), np.ravel(val_preds))
    oof_losses.append(loss)
    preds = clf.predict_proba(X_test)
    preds = np.array(preds)[:,:,1].T # take the positive class
    test_preds += preds / NFOLDS
```

03. 심장병 발병 예측(Kaggle)

EWHA EURON

3-1. 대회 소개

-주제: 심장병 사례 분석을 위한 분류 모델 생성

-분류(Classification) 문제

-진행 단계

1. 데이터에 대한 상세한 탐색적 분석 수행
2. 어떤 metric을 사용할 지 결정
3. target data와 feature data들을 모두 분석(데이터 탐색)
4. 모형에 적용하기 위해 범주형 변수를 숫자로 변환(Scaling)
5. data leakage를 방지하기 위해 파이프라인 사용(make_column_transformer)
6. 각 모델의 결과를 보고 가장 적합한(정확도가 높은) 모델 선택
   => 개선 사항을 확인하기 위해 Optuna를 사용하여 Catboost의 하이퍼 파라미터를 튜닝
7. 각 feature의 중요도와 각 모델의 정확도 확인

References

•Kaggle site: https://www.kaggle.com/code/kaanboke/beginner-friendly-catboost-with-optuna

EWHA EURON

3-2. Data Description

📌 칼럼 확인하기

Feature Data

1. Age: 환자의 연령 [년]
2. Sex: 환자의 Sex [M: 남성, F: 여성]
3. ChestPainType: [TA: 전형적 협심증, ATA: 비정형 협심증, NAP: 비각각 통증, ASY: 무증상]
4. RestingBP: 휴식 혈압 [mmHg]
5. Cholestreol: 혈청 콜레스테롤 [mm/dl]
6. FatingBS: 공복 혈당 [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: 정지 심전도 결과
[정상: 정상, ST: ST-T파 이상 유무(T파 반전 및/또는 ST 상승 또는 하강 정도가 0.05 mV 초과),
LVH: Estes의 기준에 의한 추정 또는 확실한 좌심실 비대를 나타냄]
8. MaxHR: 최대 심박수 달성 [60~202 사이의 숫자 값]
9. ExerciseAngina: 운동성 협심증 [Y: Yes, N: No]
10. Oldpeak: ST [우울 상태에서 측정한 수치]
11. ST_Slope: 피크 운동 ST 세그먼트의 기울기 [Up: 상행, Flat: 평행, Down: 하행]

Label Data

12. HeartDisease: 출력 클래스 [1: 심장 질환, 0: 정상]

EWHA EURON

3-3. 탐색적 데이터분석

Step 1> 필요한 library & package import

References

1. Scikit-learn 공식 API: https://scikit-learn.org/stable/modules/classes.html
2. imblearn 공식 API: https://imbalanced-learn.org/stable/references/index.html#api
3. plotly 공식 API: https://plotly.com/python-api-reference/

💚 기본적인 것들

```
import numpy as np
import pandas as pd

# 시각화
import matplotlib.pyplot as plt
import seaborn as sns
```

💚 부스팅 모델

```
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

```
import optuna
```

EWHA EURON

3-3. 탐색적 데이터분석

Step 1> 필요한 library & package import

Scikit-learn

```
# 교차 검증
from sklearn.model_selection import train_test_split,KFold,StratifiedKFold,RepeatedStratifiedKFold,cross_val_score,
cross_val_predict,GridSearchCV

# 결측값 처리
from sklearn.impute import SimpleImputer

# 경계 설정
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # 선형 결정 경계 설정

# 데이터 전처리
from sklearn.preprocessing import OneHotEncoder,StandardScaler,PowerTransformer # 원핫 인코딩, 정규화, 정규화
from sklearn.compose import ColumnTransformer # 컬럼 변환

# Modeling
from sklearn.svm import SVC # Support Vector Machine
from sklearn.linear_model import LogisticRegression # 로지스틱 회귀
from sklearn.dummy import DummyClassifier # 분류기(input을 무시하고 예측 수행)
from sklearn.neighbors import KNeighborsClassifier

# Ensemble
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Pipelining
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_transformer

# 평가 지표
from sklearn.metrics import accuracy_score,classification_report
```

```
from imblearn.over_sampling import SMOTE  # 임의의 소수 클래스 데이터 사이에서 새로운 데이터를 생성
```

EWHA EURON

3-3. 탐색적 데이터분석

Step 1> 필요한 library & package import

차트 그리기

```
# 오프라인 모드로 import하기
import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline = False, world_readable = True)

import plotly
import plotly.express as px
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

import missingno as msno # 결측치를 어떻게 시각화 할 것인가
```

오류 메시지

```
import warnings
warnings.filterwarnings("ignore")
```

3-3. 탐색적 데이터분석

Step 2> 데이터 분포 형태 확인하기

```
### DataFrame 옵션 설정
pd.set_option('max_columns',100)
pd.set_option('max_rows',900)
pd.set_option('max_colwidth',200)
```

```
### 데이터 Load
df = pd.read_csv('./heart.csv') # csv data -> pd.DataFrame
df.head() # 상위 5개 데이터 확인
```

Feature data

Label data

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

3-3. 탐색적 데이터분석

Step 2> 데이터 분포 형태 확인하기

```
### 데이터 요약정보 확인
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

```
### 중복된 행(=데이터) 확인
df.duplicated().sum()
```

```
1]: 0
```

중복 데이터 X

```
### 결측치 파악하기(사용자 함수 정의)

def missing (df):
    missing_number = df.isnull().sum().sort_values(ascending = False) # 결측치 개수 파악
    missing_percent = (df.isnull().sum() / df.isnull().count()).sort_values(ascending = False) # 전체 데이터에서 결측치의 비율
    missing_values = pd.concat([missing_number, missing_percent], axis = 1, keys = ['Missing_Number', 'Missing_Percent'])

    return missing_values # 결측치 DataFrame 반환
```

```
missing(df)
```

```
3]:
```

	Missing_Number	Missing_Percent
Age	0	0.0
Sex	0	0.0
ChestPainType	0	0.0
RestingBP	0	0.0
Cholesterol	0	0.0
FastingBS	0	0.0
RestingECG	0	0.0
MaxHR	0	0.0
ExerciseAngina	0	0.0
Oldpeak	0	0.0
ST_Slope	0	0.0
HeartDisease	0	0.0

결측치 X

EWHA EURON

3-3. 탐색적 데이터분석

Step 2> 데이터 분포 형태 확인하기

수치형 변수와 범주형 변수

```
numerical= df.drop(['HeartDisease'], axis = 1).select_dtypes('number').columns # feature data들 중 수치형 데이터만 선택
categorical = df.select_dtypes('object').columns # 범주형 데이터들 선택

print(f'Numerical Columns:  {df[numerical].columns}')
print('₩n')
print(f'Categorical Columns: {df[categorical].columns}')
```

```
Numerical Columns:  Index(['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak'], dtype='object')


Categorical Columns: Index(['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope'], dtype='object')
```

```
df[categorical].nunique() # 각 범주형 변수별 범주의 개수 확인
```

```
Sex              2
ChestPainType    4
RestingECG       3
ExerciseAngina   2
ST_Slope         3
dtype: int64
```

EWHA EURON

3-3. 탐색적 데이터분석

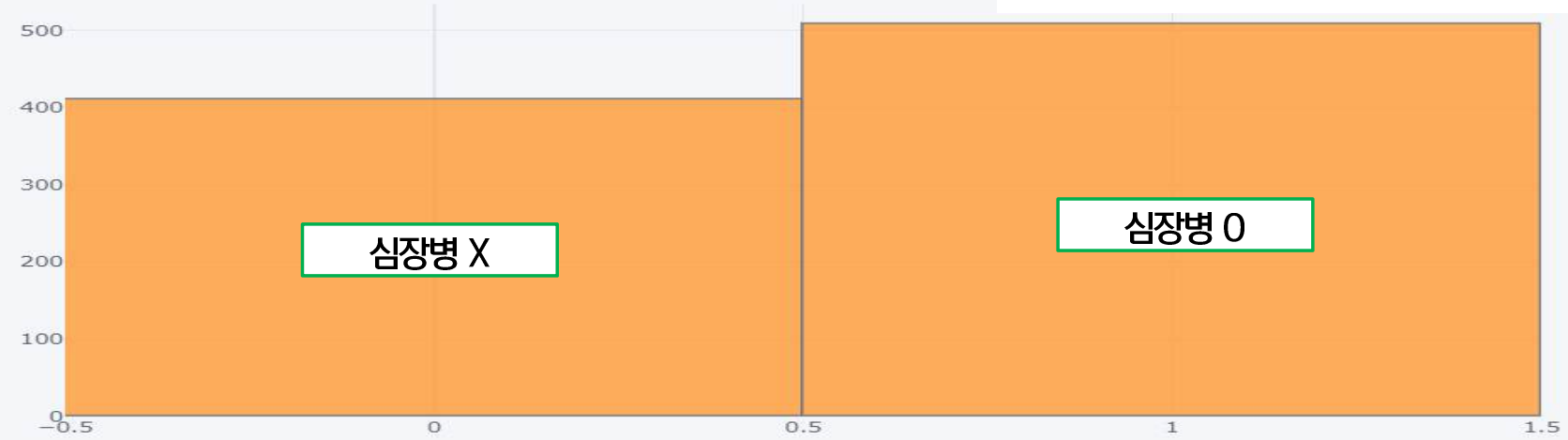
Step 3> Target 변수(=정답 데이터) 확인

```
y = df['HeartDisease'] # target data
print(f'Percentage of patient had a HeartDisease:  {round(y.value_counts(normalize = True)[1]*100, 2)} %  --> ({y.value_counts()[1]}
print(f'Percentage of patient did not have a HeartDisease: {round(y.value_counts(normalize = True)[0] * 100, 2)}  %  --> ({y.value_c
```

```
Percentage of patient had a HeartDisease:  55.34 %  --> (508 patient)
Percentage of patient did not have a HeartDisease: 44.66  %  --> (410 patient)
```

```
# 시각화
df['HeartDisease'].iplot(kind = 'hist')
```

- 약간의 불균형은 있다만 문제가 될 정도는 아님
- 평가 지표로 'accuracy'를 사용할 수 있음



EWHA EURON

3-3. 탐색적 데이터분석

Step 4> 수치형 변수들(Numerical Features)

```
df[numerical].describe() # 수치형 feature들의 기초통계량 확인
```

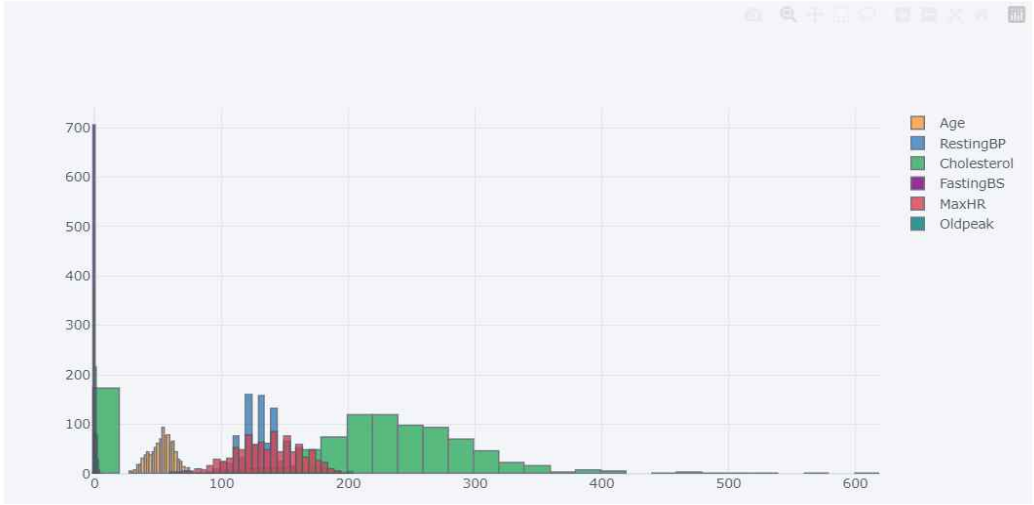
]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000

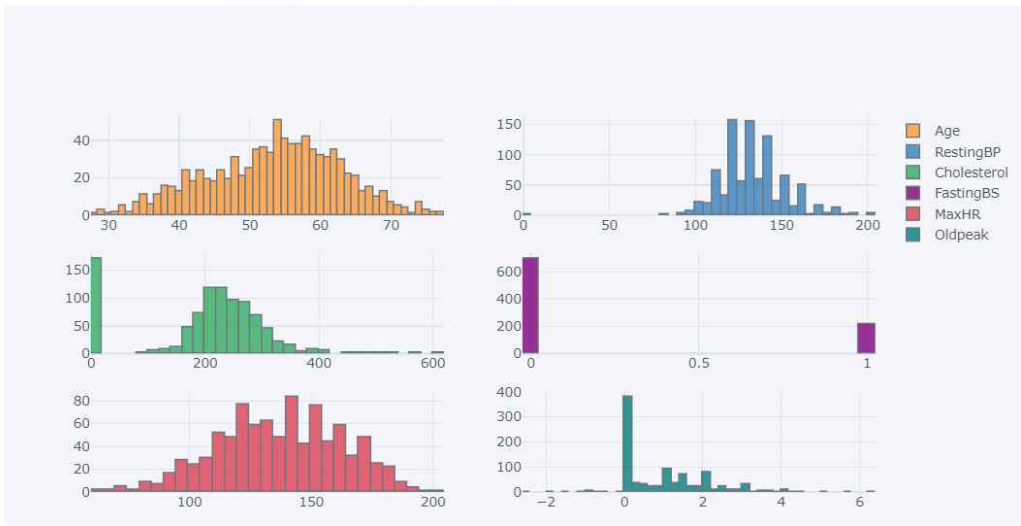
3-3. 탐색적 데이터분석

Step 4> 수치형 변수들(Numerical Features)

```
df[numerical].iplot(kind = 'hist')
```




```
df[numerical].iplot(kind = 'histogram',subplots = True, bins = 50)
```




```
skew_limit = 0.75 # 왜도를 평가하기 위한 한계치
                  # 전체적으로 아래 abs(1)은 선형 모형에 적합한 것으로 보임
skew_vals = df[numerical].drop('FastingBS', axis = 1).skew() # FastingBS는 범주형 변수로 보는 것이 합당해 보임
skew_cols= skew_vals[abs(skew_vals)> skew_limit].sort_values(ascending = False)
skew_cols # |왜도| > 한계치 인 컬럼 추출
```

```
]: Oldpeak    1.022872
   dtype: float64
```

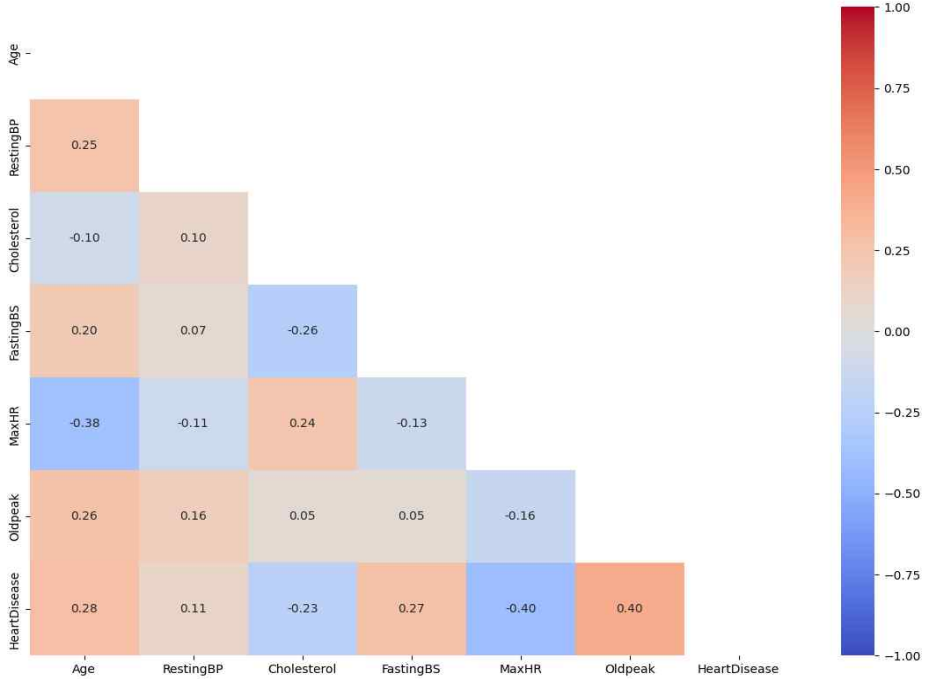
- 왜도에 별다른 문제는 없음
- 수치형 변수들은 정규분포에 근사한 분포를 가지고 있음

EWHA EURON

3-3. 탐색적 데이터분석

Step 4> 수치형 변수들(Numerical Features)

```
### heatmap으로 features 시각화

numerical1= df.select_dtypes('number').columns # 수치형 변수들 선택

matrix = np.triu(df[numerical1].corr()) # 상관계수 행렬
fig, ax = plt.subplots(figsize = (14,10))
sns.heatmap (df[numerical1].corr(), annot = True, fmt = '.2f', vmin = -1, vmax = 1, center = 0,
             cmap = 'coolwarm',mask = matrix, ax = ax)
```



- 행렬에 기초하여, 수치형 변수들과 목표 변수 사이에 약한 수준의 상관관계가 있음을 관찰할 수 있음.
- Oldpeak(-> 우울증 관련 수치)는 심장병(-> 목표 변수)과 양(+)의 상관관계를 가짐
- MaxHR(-> 최대 심박수)는 심장병(-> 목표 변수)과 음(-)의 상관관계를 가짐
- Cholesterol(-> 콜레스테롤)은 심장병(-> 목표 변수)과 음(-)의 상관관계를 가짐

EWHA EURON

3-3. 탐색적 데이터분석

Step 5> 범주형 변수들(Categorical Features)

```
### 데이터 미리보기
df[categorical].head()
```

]:

	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_Slope
0	M	ATA	Normal	N	Up
1	F	NAP	Normal	N	Flat
2	M	ATA	ST	N	Up
3	F	ASY	Normal	Y	Flat
4	M	NAP	Normal	N	Up

3-3. 탐색적 데이터분석

Step 5> 범주형 변수들(Categorical Features)

💚 성별에 따른 심장병 발병 확률

```
print (f'A female person has a probability of {round(df[df["Sex"]=="F"]["HeartDisease"].mean()*100,2)} % have a HeartDisease')
print()
print (f'A male person has a probability of {round(df[df["Sex"]=="M"]["HeartDisease"].mean()*100,2)} % have a HeartDisease')
print()
```

```
A female person has a probability of 25.91 % have a HeartDisease

A male person has a probability of 63.17 % have a HeartDisease
```

```
fig = px.histogram(df, x = "Sex", color = "HeartDisease",width = 400, height = 400)
fig.show()
```

0: 심장병 발병 x
1: 심장병 발병 O

- 남성이 여성보다 심장병에 걸릴 확률이 거의 2.44배 높다.

EWHA EURON

3-3. 탐색적 데이터분석

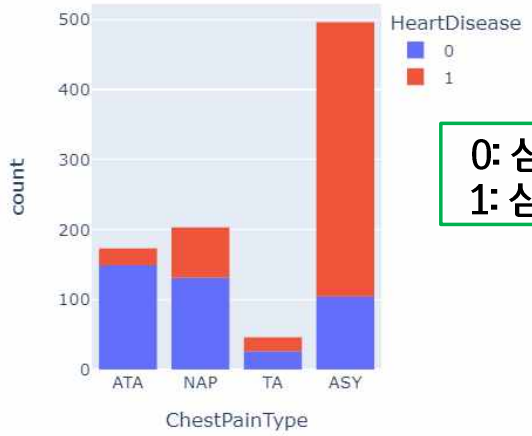
Step 5> 범주형 변수들(Categorical Features)

💚 **ChestPainType(가슴 통증 유형)**에 따른 심장병 발병 확률

```
df.groupby('ChestPainType')['HeartDisease'].mean().sort_values(ascending = False)
```

```
6]: ChestPainType
    ASY    0.790323
    TA     0.434783
    NAP    0.354680
    ATA    0.138728
    Name: HeartDisease, dtype: float64
```

```
fig = px.histogram(df, x = "ChestPainType", color = "HeartDisease",width = 400, height = 400)
fig.show()
```



0: 심장병 발병 x
1: 심장병 발병 O

- 통증 유형별로 뚜렷한 차이가 존재함
- ASY(무증상)인 사람: 무증상성 흉통은 ATA(비정형 협심증) 흉통을 가진 사람보다 심장 질환을 가질 가능성이 6배 정도 더 높다.

EWHA EURON

3-3. 탐색적 데이터분석

Step 5> 범주형 변수들(Categorical Features)

💚 **RestingECG(정지 심전도 결과)**에 따른 심장병 발병 확률

```
df.groupby('RestingECG')['HeartDisease'].mean().sort_values(ascending = False)
```

```
8]: RestingECG
ST        0.657303
LVH       0.563830
Normal    0.516304
Name: HeartDisease, dtype: float64
```

```
fig = px.histogram(df, x = "RestingECG", color = "HeartDisease",width = 400, height = 400)
fig.show()
```

0: 심장병 발병 x
1: 심장병 발병 0

- RestingECG(정지 심전도 결과)는 크게 다르지 않음
- ST인 사람: ST - T 파동 이상을 갖는 사람은 다른 사람들보다 심장병을 가질 가능성이 더 높음

EWHA EURON

3-3. 탐색적 데이터분석

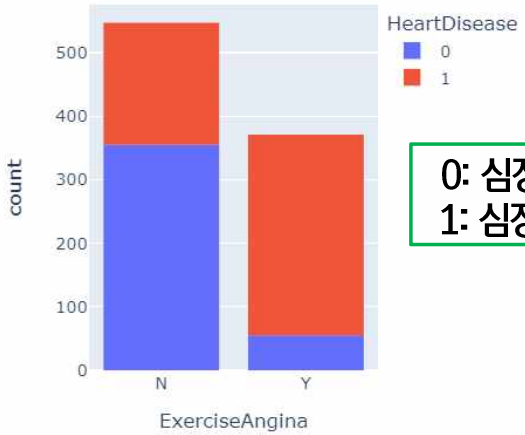
Step 5> 범주형 변수들(Categorical Features)

ExerciseAngina(운동성 협심증 여부)에 따른 심장병 발병 확률

```
df.groupby('ExerciseAngina')['HeartDisease'].mean().sort_values(ascending = False)
```

```
ExerciseAngina
Y    0.851752
N    0.351005
Name: HeartDisease, dtype: float64
```

```
fig = px.histogram(df, x = "ExerciseAngina", color = "HeartDisease",width = 400, height = 400)
fig.show()
```



0: 심장병 발병 x
1: 심장병 발병 0

- 운동성 협심증이 있는 사람이 없는 사람에 비해 심장 질환이 있을 확률이 거의 2.4배 높음

3-3. 탐색적 데이터분석

Step 5> 범주형 변수들(Categorical Features)

ST_Slope에 따른 심장병 발병 확률

```
df.groupby('ST_Slope')['HeartDisease'].mean().sort_values(ascending = False)
```

```
2]: ST_Slope
    Flat    0.828261
    Down    0.777778
    Up      0.197468
    Name: HeartDisease, dtype: float64
```

```
fig = px.histogram(df, x = "ST_Slope", color = "HeartDisease",width = 400, height = 400)
fig.show()
```

0: 심장병 발병 x
1: 심장병 발병 0

- 피크 운동 ST 세그먼트의 기울기에 따른 심장병 발병 비율의 차이가 있음
- ST_Slope가 Up인 사람은 다른 두 유형의 사람들에 비해 심장병에 걸릴 확률이 현저히 낮음

EWHA EURON

3-3. 탐색적 데이터분석

Step 6> 데이터 요약

•target data는 균형 있는 데이터에 가깝다고 볼 수 있음

•수치형 변수들은 대상 변수와 약한 상관관계를 가짐

•Oldpeak(우울증 관련 수치)는 심장 질환과 양(+)의 상관관계가 있음

•MaxHR은 심장 질환과 음(-)의 상관관계를 가짐

•Cholesterol(콜레스테롤)은 심장병과 음(-)의 상관관계를 가지고 있음

•By Sex: 남자가 여자보다 심장병에 걸릴 확률이 거의 2.44배 높음

•ChestPainType(흉통 유형)별로 뚜렷한 차이를 관찰할 수 있음

•=> ASY(무증상)인 사람: 무증상성 흉통은 ATA(비정형 협심증) 흉통을 가진 사람보다 심장 질환을 가질 가능성이 6배 정도 더 높음

•RestingECG: 휴식 심전도 결과는 크게 다르지 않음

•=> ST인 사람: ST - T 파동 이상을 갖는 사람은 다른 사람들보다 심장병을 가질 가능성이 더 높음

•ExerciseAngina(운동성 협심증)이 있는 사람이 없는 사람에 비해 심장 질환이 있을 확률이 거의 2.4배 높음

•ST_Slope(피크 운동 ST 세그먼트의 기울기)에 따른 심장병 발병 비율의 차이가 있음

•=> ST_Slope가 Up인 사람은 다른 두 유형의 사람들에 비해 심장병에 걸릴 확률이 현저히 낮음

3-4. Modeling

Step 0> 개요

•기준선 모델로 dummy Classifier를 사용

•이후 scaler 유무를 다르게 하여 각각 Logistic, Linear Discriminant(LDA), KNeighbors(KNN), Support Vector Machine(SVM) 모델을 적용

•이후 앙상블 모델링 기법인 Adaboost, Randomforest, Gradient Boosting and Extra Trees를 활용

•유명한 부스팅 모델들인 XGBoost,LightGBM, Catboost를 다룰 예정

•마지막으로 Catboost를 위한 hyper parameter tuning을 자세하게 알아볼 예정

3-4. Modeling

Step 1> Baseline Model(기준선 모델)

- sklearn.compose.make_column_transformer
  - 데이터에 수치형(numerical) 변수와 범주형(categorical) 변수가 섞여 있는 경우, 이들을 각각 따로 Encoding 해주기 위해 사용
  - 처리할 데이터를 분리한 후, 각각에 맞는 Encoding 기법을 넣어주면 됨

```
accuracy =[]
model_names =[]

### 데이터 준비
X = df.drop('HeartDisease', axis = 1) # feature datas -> HeartDisease를 제외한 모든 columns
y = df['HeartDisease'] # label(= target) data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)  # train_set : test_set = 7 : 3

### Encoding
ohe = OneHotEncoder() # 0(= False) or 1(= True)
ct = make_column_transformer((ohe,categorical),remainder = 'passthrough')
# 범주형 데이터는 transform(train data로부터 학습된 mean값과 variance값을 test data에 적용)을 하지 않고 데이터를 그대로 통과

### 모델 생성
model = DummyClassifier(strategy = 'constant', constant = 1) # 모델 적합 시 x에 대해 항상 1을 예측으로 반환
pipe = make_pipeline(ct, model) # Encoder와 model 지정
pipe.fit(X_train, y_train) # 학습
y_pred = pipe.predict(X_test) # 예측
accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가
print (f'model : {model} and  accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')

### 결과
model_names = ['DummyClassifier']
dummy_result_df = pd.DataFrame({'Accuracy':accuracy}, index = model_names)
dummy_result_df
```

	Accuracy
DummyClassifier	0.5942

EWHA EURON

3-4. Modeling

Step 2> Logistic & Linear Discriminant & SVC & KNN(Scaler)

```
accuracy =[]
model_names =[]

### 데이터 준비
X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

# Encoding
ohe= OneHotEncoder()
ct= make_column_transformer((ohe,categorical),remainder='passthrough') # Scaler 적용 x
# remainder='passthrough': transform을 하지 않고 데이터를 그대로 통과

### Modeling
lr = LogisticRegression(solver = 'liblinear')
lda= LinearDiscriminantAnalysis()
svm = SVC(gamma = 'scale')
knn = KNeighborsClassifier()

models = [lr,lda,svm,knn]
# 각 모델별로 작업 진행
for model in models:
    pipe = make_pipeline(ct, model) # Encoder와 model 지정
    pipe.fit(X_train, y_train) # 학습
    y_pred = pipe.predict(X_test) # 예측
    accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가
    print (f'model : {model} and  accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')

### 결과
model_names = ['Logistic','LinearDiscriminant','SVM','KNeighbors']
result_df1 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df1
```

	Accuracy
Logistic	0.8841
LinearDiscriminant	0.8696
SVM	0.7246
KNeighbors	0.7174

EWHA EURON

3-4. Modeling

Step 3> Logistic & Linear Discriminant & SVC & KNN(Scaler)

```
accuracy =[]
model_names =[]

### 데이터 준비
X= df.drop('HeartDisease', axis = 1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

### Encoding
ohe= OneHotEncoder()
s= StandardScaler() # Scaler 적용
ct1= make_column_transformer((ohe,categorical),(s,numerical))
# 범주형 변수는 One-hot Encoding, 수치형 변수는 StandardScaler 적용

### Modeling
lr = LogisticRegression(solver = 'liblinear')
lda = LinearDiscriminantAnalysis()
svm = SVC(gamma = 'scale')
knn = KNeighborsClassifier()

models = [lr,lda,svm,knn]
# 각 모델별로 작업 진행
for model in models:
    pipe = make_pipeline(ct1, model) # Encoder와 model 지정
    pipe.fit(X_train, y_train) # 학습
    y_pred = pipe.predict(X_test) # 예측
    accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가
    print (f'model : {model} and  accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')

### 결과
model_names = ['Logistic_scl','LinearDiscriminant_scl','SVM_scl','KNeighbors_scl']
result_df2 = pd.DataFrame({'Accuracy':accuracy}, index = model_names)
result_df2
```

Scale

	Accuracy
Logistic	0.8841
LinearDiscriminant	0.8696
SVM	0.7246
KNeighbors	0.7174

Scale

	Accuracy
Logistic_scl	0.8804
LinearDiscriminant_scl	0.8696
SVM_scl	0.8841
KNeighbors_scl	0.8841

예상대로 KNN과 SVM 모두 Scaler를 적용하기 이전보다 Scaler 적용 후에 성능이 향상됨

EWHA EURON

3-4. Modeling

Step 4> 앙상블 모델(AdaBoost, Gradient Boosting, Random Forest, Extra Trees)

```
accuracy =[]
model_names =[]

### 데이터 준비
X = df.drop('HeartDisease', axis = 1)
y = df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

### Encoding
ohe = OneHotEncoder()
ct = make_column_transformer((ohe,categorical), remainder = 'passthrough')  # Scaler 적용 x

### Ensemble
ada = AdaBoostClassifier(random_state = 0) # 오류 데이터에 가중치를 부여하면서 부스팅 수행
gb = GradientBoostingClassifier(random_state = 0) # 경사 하강법을 이용하여 가중치 업데이트
rf = RandomForestClassifier(random_state = 0) # 결정 트리 기반 배깅(샘플 중복 생성을 통한 결과 도출) 알고리즘,
                                              # 약한 학습기들을 생성한 후 이를 선형 결합하여 최종 학습기 생성
et=  ExtraTreesClassifier(random_state = 0) # 결정 트리 기반 배깅 알고리즘
                                            # 훈련에 사용할 특성과 임계값을 무작위로 선택 후 최적 선택

models = [ada,gb,rf,et]
# 각 모델별로 작업 진행
for model in models:
    pipe = make_pipeline(ct, model) # Encoder와 model 지정
    pipe.fit(X_train, y_train) # 학습
    y_pred = pipe.predict(X_test) # 예측
    accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가
    print (f'model : {model} and  accuracy score is : {round(accuracy_score(y_test, y_pred),4)}')

### 결과
model_names = ['Ada','Gradient','Random','ExtraTree']
result_df3 = pd.DataFrame({'Accuracy':accuracy}, index = model_names)
result_df3
```

	Accuracy
Ada	0.8659
Gradient	0.8768
Random	0.8877
ExtraTree	0.8804

- 정확도 점수는 서로 매우 비슷함
- RandomForest와 ExtraTree 모두 비슷한, 높은 정확도 점수를 보임
- 두 모델 모두 hyper parameter tuning으로 개선할 수 있음

EWHA EURON

3-4. Modeling

Step 5> 부스팅 모델(XGBoost, LightGBM, Catboost)

- 부스팅: 예측력이 약한 모형들을 결합하여 강한 예측모형을 만드는 방법
- 전처리 없이 범주형 변수를 처리하는 기능을 사용하여 이후 CatBoost를 단독으로 사용할 예정

```
accuracy = []
model_names = []

### 데이터 준비
X = df.drop('HeartDisease', axis=1)
y = df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

### Encoding
ohe = OneHotEncoder()
ct = make_column_transformer((ohe,categorical),remainder = 'passthrough') # Scaler 적용 x

### Boosting
xgbc = XGBClassifier(random_state = 0)
lgbmc = LGBMClassifier(random_state = 0)


models = [xgbc,lgbmc]
# 각 모델별로 작업 진행
for model in models:
    pipe = make_pipeline(ct, model) # Encoder와 model 지정
    pipe.fit(X_train, y_train) # 학습
    y_pred = pipe.predict(X_test) # 예측
    accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가

### 결과
model_names = ['XGBoost','LightGBM']
result_df4 = pd.DataFrame({'Accuracy':accuracy}, index=model_names)
result_df4
```

	Accuracy
XGBoost	0.8297
LightGBM	0.8732

EWHA EURON

3-4. Modeling

Step 6> CatBoost

•범주형 변수를 더 잘 처리하고, 과적합(overfitting) 문제를 개선한 알고리즘
•다른 모델들의 경우 범주형 변수를 사용하기 위해서는 One - Hot Encoding 등 데이터 전처리가 선행되어야 하지만, Catboost에서는 자동으로 이를 변환하여 사용할 수 있다는 장점이 있음
•내부적 알고리즘을 통한 과적합, sampling 다양성 등의 문제가 개선됨
•=> hyper parameter에 따른 영향이 적음

•사용 의도
  - 분류 문제에 대한 교육 및 모델 적용 => scikit-learn 도구와의 호환성을 제공

✔ 기본 최적화 목표는 다양한 조건에 따라 달라질 수 있다.
  - Logloss: 대상에 두 개의 다른 값만 있거나 target_border 매개 변수가 None이 아닌 경우
  - MultiClass: 대상에 두 개 이상의 다른 값이 있으며 border_count 매개 변수가 None인 경우

Reference

•CatBoost 공식 API: https://catboost.ai/en/docs/concepts/python-reference_catboostclassifier

3-4. Modeling

Step 6> CatBoost

```
accuracy = []
model_names = []

### 데이터 준비
X = df.drop('HeartDisease', axis = 1)
y = df['HeartDisease']
categorical_features_indices = np.where(X.dtypes != np.float)[0] # 범주형 변수의 index 반환
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

### Modeling
model = CatBoostClassifier(verbose = False, random_state = 0) # verbose = False: Silent logging level
                                                              # -> stdout에 로깅 정보를 출력하지 x

### 학습, 예측, 평가
model.fit(X_train, y_train,cat_features = categorical_features_indices, eval_set = (X_test, y_test)) # 학습
y_pred = model.predict(X_test) # 예측
accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가

### 결과
model_names = ['Catboost_default']
result_df5 = pd.DataFrame({'Accuracy':accuracy}, index = model_names)
result_df5
```

	Accuracy
Catboost_default	0.8804

EWHA EURON

3-4. Modeling

Step 7> Optuna를 활용한 Catboost의 hyper parameter tuning

•Optuna
- 하이퍼 파라미터 최적 프레임워크
- 파라미터의 범위를 지정해주거나 파라미터가 될 수 있는 목록 설정 시 매 trial마다 파라미터를 변경하면서 최적의 파라미터 탐색

•parameter의 범위/목록 설정하기
- suggest_int: 범위 내의 정수형 값을 선택
- suggest_float: 범위 내의 실수형 값을 선택
- suggest_categorical: list 내의 데이터 중 선택
- suggest_uniform: 범위 내의 균일 분포를 값으로 선택
- suggest_discrete_uniform: 범위 내의 이산 균등 분포를 값으로 선택
- suggest_loguniform: 범위 내의 로그 함수 선상의 값 선택

3-4. Modeling

Step 7> Optuna를 활용한 Catboost의 hyper parameter tuning

•Tuning을 진행할 Catboost의 Parameters

1. Objective: 과적합 감지 및 최상의 모델을 위해 지원되는 측정 기준
   - Logloss: 정답을 더 높은 확률로 예측할수록 좋은 모델이라고 평가
   - CrossEntropy: 훈련 데이터를 사용한 예측 모형에서 실제 값과 예측값의 차이 (dissimilarity)를 계산

2. colsample_bylevel: 학습 속도를 높임, 일반적으로 품질에 영향을 미치지 않음
   => 해당 대회에서는 0.01 ~ 0.1 사이

3. depth: 트리의 깊이
   => 해당 대회에서는 1 ~ 12 사이

3-4. Modeling

Step 7> Optuna를 활용한 Catboost의 hyper parameter tuning

•Tuning을 진행할 Catboost의 Parameters

4. boosting_type: 기본적으로 부스팅 유형은 작은 데이터 세트의 경우로 설정됨 -> 과적합을 방지하지만 계산 측면에서 많은 비용이 소모됨
=> 교육 속도를 높이기 위해 해당 parameter를 활용할 수 있음
  - Ordered: 일반적으로 작은 데이터 세트에서 더 나은 품질을 제공하지만 Plain 방식보다 느릴 수 있음.
  - Plain: 전형적인 GradientBoosting 방식

5. bootstrap_type: 객체의 가중치를 샘플링하는 방법을 정의
  - Bayesian: 앞서 나온 결과의 '경험'을 계속해서 반영하면서 최적 하이퍼파라미터 값에 빠르게 도달할 수 있게 함
  - Bernoulli: 확률분포를 베르누이 분포로 가정
  - MVS: 최소 분산 sampling(Minimal Variance Sampling)

3-4. Modeling

Step 7> Optuna를 활용한 Catboost의 hyper parameter tuning

•Optuna를 적용하기 위한 objective function 정의

```
def objective(trial): # 각 시도마다
    ### 데이터 준비
    X = df.drop('HeartDisease', axis=1)
    y = df['HeartDisease']
    categorical_features_indices = np.where(X.dtypes != np.float)[0] # 범주형 변수의 index 반환
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

    ### tuning을 진행할 hyper parameters 설정
    param = {
        "objective": trial.suggest_categorical("objective", ["Logloss", "CrossEntropy"]),
        "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.01, 0.1),
        "depth": trial.suggest_int("depth", 1, 12),
        "boosting_type": trial.suggest_categorical("boosting_type", ["Ordered", "Plain"]),
        "bootstrap_type": trial.suggest_categorical("bootstrap_type", ["Bayesian", "Bernoulli", "MVS"]),
        "used_ram_limit": "3gb",
    }
    # bootstrap 유형에 따라 옵션 설정하기
    if param["bootstrap_type"] == "Bayesian":
        param["bagging_temperature"] = trial.suggest_float("bagging_temperature", 0, 10)
    elif param["bootstrap_type"] == "Bernoulli":
        param["subsample"] = trial.suggest_float("subsample", 0.1, 1)

    ### Modeling
    cat_cls = CatBoostClassifier(**param) # 분류기 객체 생성/ **params: 인자의 개수가 가변적일 때 사용
    cat_cls.fit(X_train, y_train, eval_set=[(X_test, y_test)], cat_features = categorical_features_indices,
                verbose = 0, early_stopping_rounds = 100) # 학습
    preds = cat_cls.predict(X_test) # 예측
    pred_labels = np.rint(preds) # 예측값을 가장 가까운 정수로 반올림
    accuracy = accuracy_score(y_test, pred_labels) # 평가
    return accuracy
```

3-4. Modeling

Step 7> Optuna를 활용한 Catboost의 hyper parameter tuning

•parameters를 바꿔가며 학습 수행하기

```
if __name__ == "__main__":
    study = optuna.create_study(direction = "maximize") # objective가 최대화 되는 방향으로 학습 진행
    study.optimize(objective, n_trials = 50, timeout = 600) # 학습 진행 횟수와 시간 제한 걸기

    ### 결과
    print("Number of finished trials: {}".format(len(study.trials))) # 학습 완료 메시지
    print("Best trial:")
    trial = study.best_trial
    print("  Value: {}".format(trial.value))
    print("  Params: ")
    for key, value in trial.params.items():
        print("    {}: {}".format(key, value))
```

```
Number of finished trials: 45
Best trial:
  Value: 0.9021739130434783
  Params:
    objective: CrossEntropy
    colsample_bylevel: 0.07348231854712815
    depth: 12
    boosting_type: Plain
    bootstrap_type: Bayesian
    bagging_temperature: 0.39591467728098717
```

EWHA EURON

3-4. Modeling

Step 7> Optuna를 활용한 Catboost의 hyper parameter tuning

•Optuna를 통해 찾은 최적 parameters들을 적용하여 모델 생성하기

```
accuracy = []
model_names = []

### 데이터 준비
X = df.drop('HeartDisease', axis = 1)
y = df['HeartDisease']
categorical_features_indices = np.where(X.dtypes != np.float)[0] # 범주형 변수의 index 반환
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

### Modeling
model = CatBoostClassifier(verbose = False,random_state = 0,
                           objective = 'CrossEntropy',
                           colsample_bylevel = 0.04292240490294766,
                           depth = 10,
                           boosting_type = 'Plain',
                           bootstrap_type = 'MVS')
model.fit(X_train, y_train, cat_features = categorical_features_indices,eval_set = (X_test, y_test)) # 학습
y_pred = model.predict(X_test) # 예측
accuracy.append(round(accuracy_score(y_test, y_pred),4)) # 평가
print(classification_report(y_test, y_pred))

### 결과
model_names = ['Catboost_tuned']
result_df6 = pd.DataFrame({'Accuracy':accuracy}, index = model_names)
result_df6
```

	precision	recall	f1-score	support
0	0.88	0.90	0.89	112
1	0.93	0.91	0.92	164
accuracy			0.91	276
macro avg	0.90	0.91	0.91	276
weighted avg	0.91	0.91	0.91	276

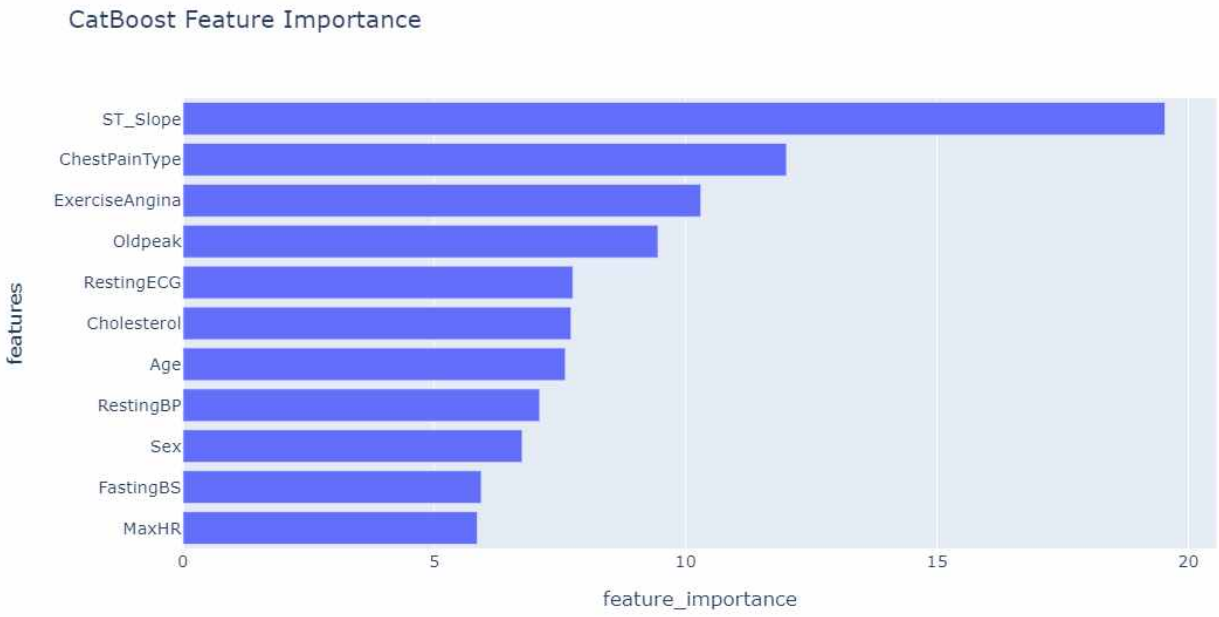
	Accuracy
Catboost_tuned	0.9094

EWHA EURON

3-5. 결론

1> 변수 별 중요도(Feature importances)

```
feature_importance = np.array(model.get_feature_importance())
features = np.array(X_train.columns)
fi = {'features':features,'feature_importance':feature_importance}
df_fi = pd.DataFrame(fi)
df_fi.sort_values(by = ['feature_importance'], ascending = True,inplace = True)
fig = px.bar(df_fi, x = 'feature_importance', y = 'features',title = "CatBoost Feature Importance",height = 500)
fig.show()
```

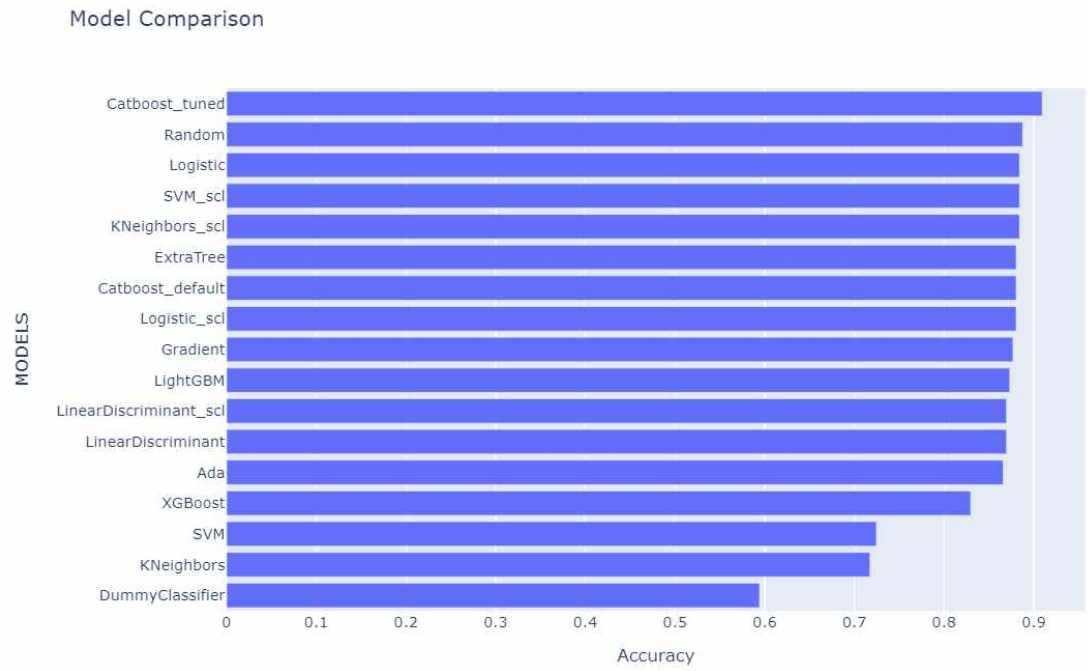


3-5. 결론

2> Model 별 정확도(Accuracy) 비교

```
result_final = pd.concat([dummy_result_df,result_df1,result_df2,result_df3,result_df4,result_df5,result_df6],axis = 0)
```

```
result_final.sort_values(by = ['Accuracy'], ascending = True,inplace = True)
fig = px.bar(result_final, x = 'Accuracy', y = result_final.index,title = 'Model Comparison',
             height = 600, labels = {'index':'MODELS'})
fig.show()
```



3-5. 결론

3> 마무리

•주제: 심장병 사례 분류 모델 생성하기
•Steps
1. 데이터에 대한 상세한 탐색적 분석 수행
2. 어떤 metric을 사용할 지 결정
3. target data와 feature data들을 모두 분석(데이터 탐색)
4. 모형에 적용하기 위해 범주형 변수를 숫자로 변환(Scaling)
5. data leakage를 방지하기 위해 파이프라인 사용(make_column_transformer)
6. 각 모델의 결과를 보고 가장 적합한(정확도가 높은) 모델 선택 => 개선 사항을 확인하기 위해 Optuna를 사용하여 Catboost의 하이퍼 파라미터를 튜닝
7. 각 feature의 중요도와 각 모델의 정확도 확인

04. 신용카드 사기 검출

EWHA EURON

#4.1 캐글 신용카드 사기 검출 대회 소개

MACHINE LEARNING GROUP - ULB · UPDATED 4 YEARS AGO

9401 | New Notebook | Download (69 MB)

Credit Card Fraud Detection

Anonymized credit card transactions labeled as fraudulent or genuine

Data | Code (3746) | Discussion (104) | Metadata

About Dataset

Context

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Content

The dataset contains transactions made by credit cards in September 2013 by European cardholders.
This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

Usability
8.53

License
Database: Open Database, Cont...

Expected update frequency
Not specified

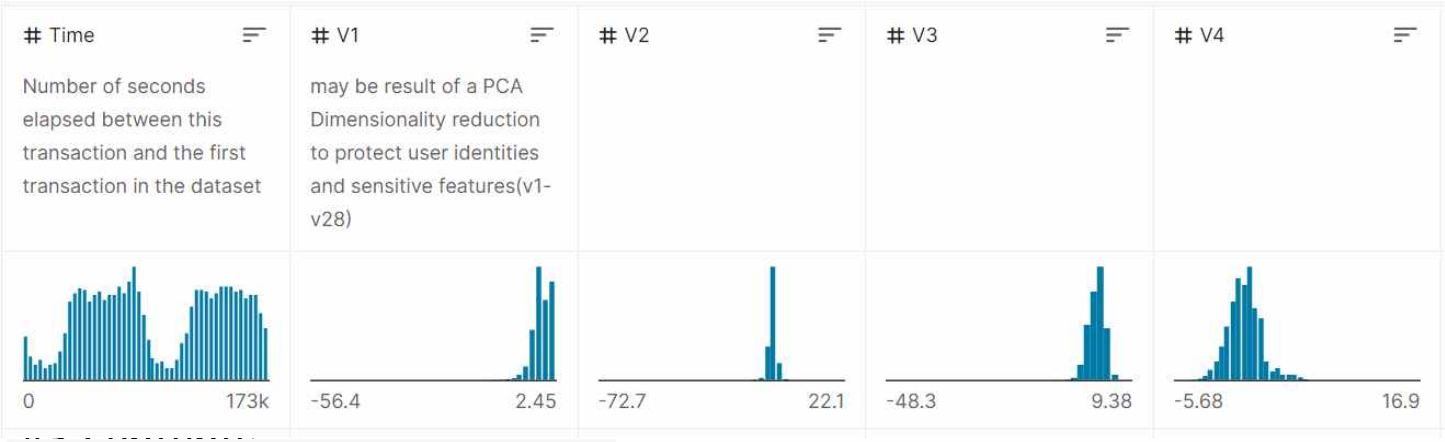
✓데이터 세트에는 유럽 카드 소지자가 2013년 9월에 신용 카드로 거래한 내용이 포함됨.

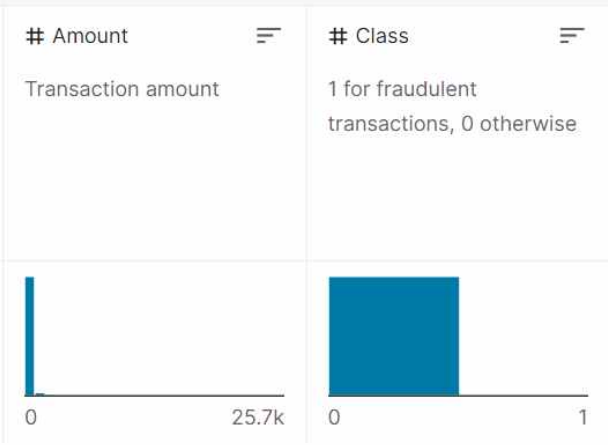
✓이 데이터 세트는 284,807건의 거래 중 492건의 사기가 발생한 이틀 동안 발생한 거래를 보여줌.

✓데이터 세트는 매우 불균형하며 긍정적 클래스(사기)는 모든 거래의 0.172% 차지.
- ✓ Class는 0과 1로 분류됨.
  - ✓ 0: 사기가 아닌 정상적인 신용카드 트랜젝션 데이터
  - ✓ 1: 신용카드 사기 트랜잭션

EWHA EURON

#4.2 Data Description





1) Time: 각 트랜잭션과 데이터 세트의 첫 째 트랜잭션 사이에 경과된 시간(초)이 포함.

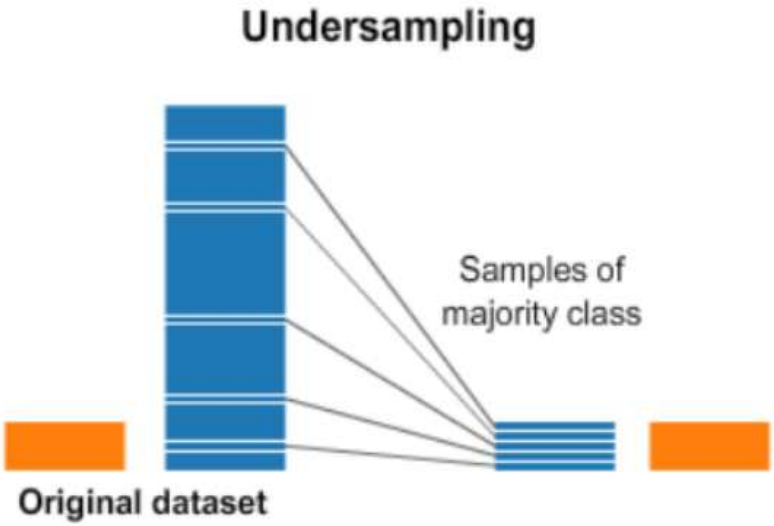
3) V1-V28: PCA로 얻은 주요 구성 요소이지만 기밀 문제로 인해 데이터에 대한 원래 기능과 추가 배경 정보를 제공하지 않음.

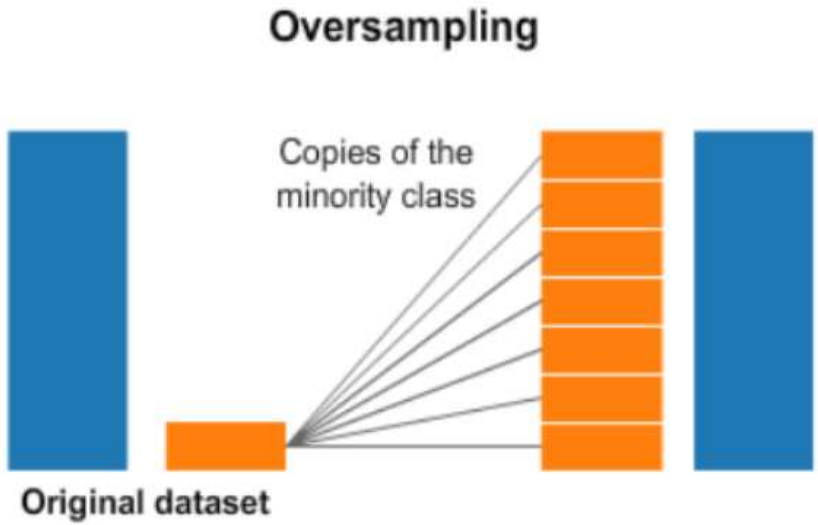
5) Amount: 거래 금액. 신용카드 트랜잭션 금액

7) Class: 0의 경우 정상, 1의 경우 사기 트랜잭션

EWHA EURON

#4.3 불균형 데이터 학습 방법(1)

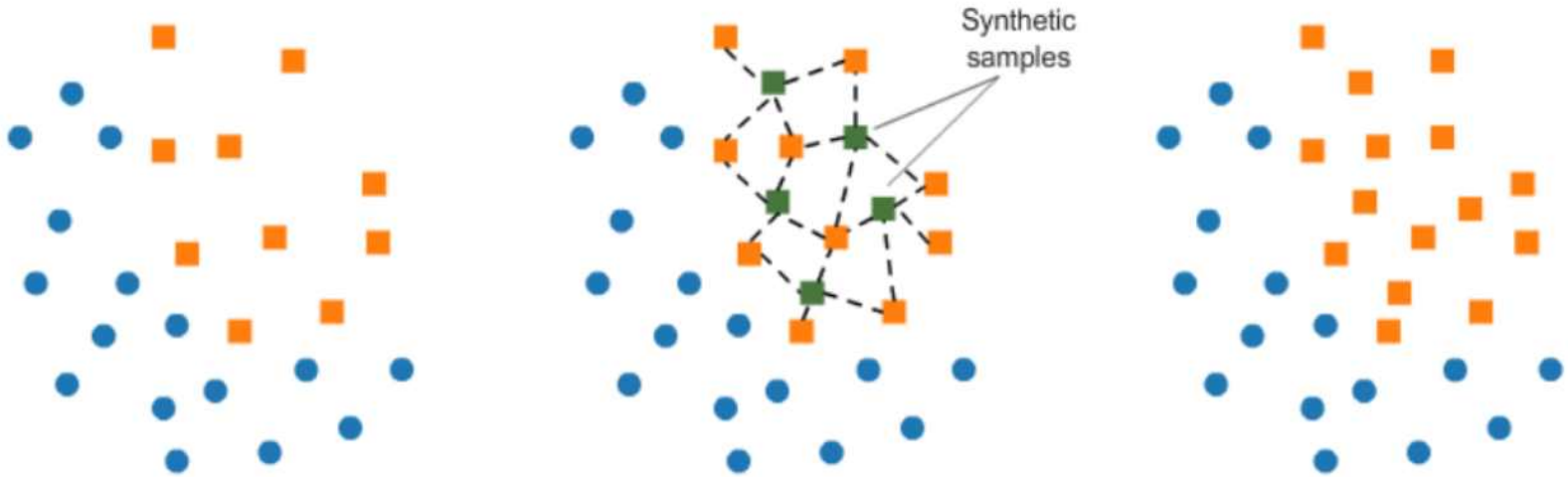




- ✓언더 샘플링: 많은 데이터 세트를 적은 데이터 세트 수준으로 감소시키는 방식.
  - ✓장점: 과도하게 정상 레이블로 학습/예측하는 부작용 개선
  - ✓단점: 너무 많은 정상 레이블 데이터를 감소시켜 정상 레이블의 경우 오히려 제대로 된 학습 수행 불가

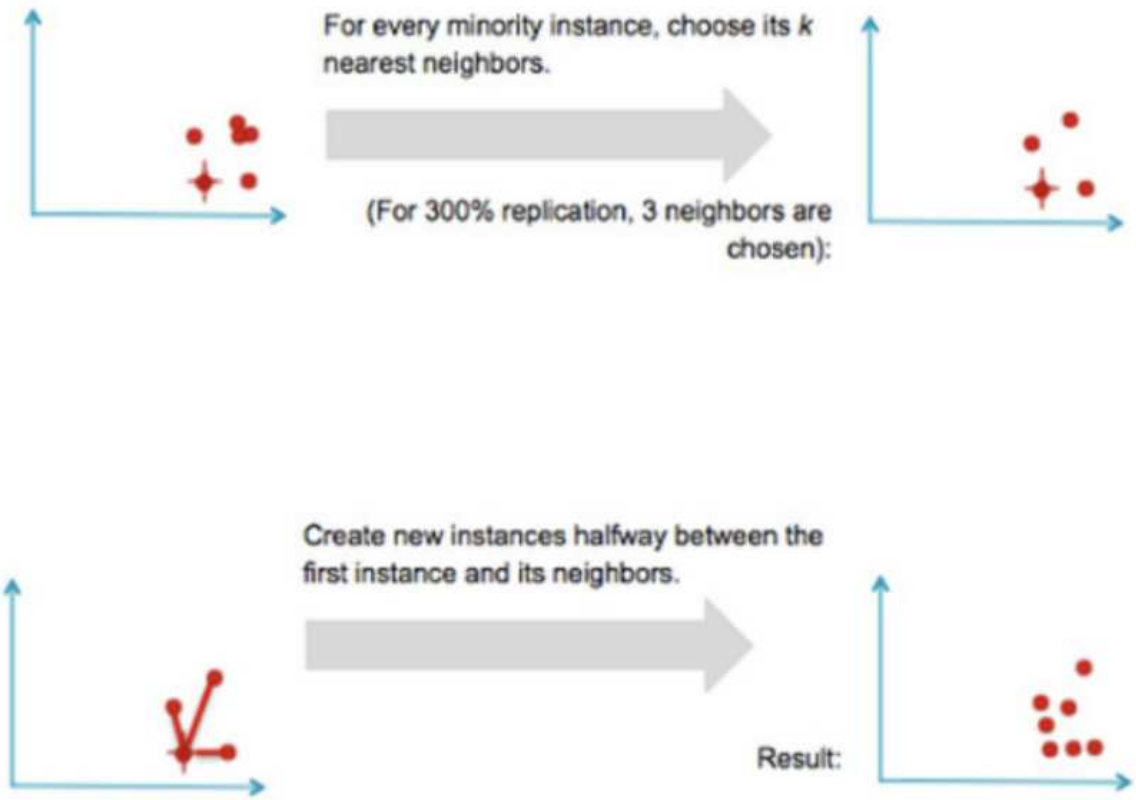
- ✓오버 샘플링: 이상 데이터와 같이 적은 데이터 세트를 증식하여 학습을 위한 충분한 데이터를 확보하는 방법.
  - ✓원본 데이터의 피처 값들을 아주 약간 변경하여 증식.
  - ✓대표적으로 SMOTE 방법이 있음.

#4.3 불균형 데이터 학습 방법(2) - SMOTE



- SMOTE(Synthetic Minority Over-sampling TEchnique)
  - 적은 데이터 세트에 있는 개별 데이터들의 K 최근접 이웃을 찾아 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어 기존 데이터와 약간 차이가 나는 새로운 데이터 생성

#4.3 불균형 데이터 학습 방법(2) - SMOTE



1)소수 클래스의 데이터 중 특정 벡터(샘플)와 가장 가까운 k개의 이웃 벡터 선정

3)기준 벡터와 선정한 벡터 사이를 선분으로 연결

5)선분 위의 임의의 점이 새로운 벡터(혹은 이 중 임의의 하나)

```
!pip install imbalanced-learn
```

Looking in indexes: https://pypi.

#4.4 데이터 1차 가공 및 모델 학습/예측/평가

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

card_df = pd.read_csv('./creditcard.csv')
card_df.head(3)
```

	Time	V1	V2	V3	V4	V5	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1

3 rows × 31 columns

Time 피처는 데이터 생성 관련한 작업용 속성으로서 큰 의미가 없기에 제거(drop).

```
from sklearn.model_selection import train_test_split

# 인자로 입력받은 DataFrame 복사한 뒤 Time 칼럼만 삭제하고 복사된 DataFrame 반환
def get_preprocessed_df(df=None):
  df_copy = df.copy()
  df_copy.drop('Time', axis=1, inplace=True)
  return df_copy
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

전체 284807개의 레코드에서 결측치 값은 없고, Class 테이블만 int형, 나머지 피처들은 모두 float형이다.

EWHA EURON

#4.4 데이터 1차 가공 및 모델 학습/예측/평가

```
from sklearn.model_selection import train_test_split

# 인자로 입력받은 DataFrame 복사한 뒤 Time 칼럼만 삭제하고 복사된 DataFrame 반환
def get_preprocessed_df(df=None):
  df_copy = df.copy()
  df_copy.drop('Time', axis=1, inplace=True)
  return df_copy
```

```
# 사전 데이터 가공 후 학습과 테스트 데이터 세트를 반환하는 함수
def get_train_test_dataset(df=None):
  # 인자로 입력된 DataFrame의 사전 데이터 가공이 완료된 복사 DataFrame 반환
  df_copy= get_preprocessed_df(df)
  # DataFrame의 맨 마지막 칼럼이 레이블, 나머지는 피처들
  X_features = df_copy.iloc[:,:-1]
  y_target = df_copy.iloc[:,-1]
  # train_test_split()으로 학습과 테스트 데이터 분할. stratify=y_target으로 Stratified 기반 분할
  X_train,X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.3, random_state=0, stratify=y_target)
  # 학습과 테스트 데이터 세트 반환
  return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)
```

Get_preprocessed_df(): 불필요한 Time 피처 삭제
Get_train_test_dataset(): 학습 피처/레이블 데이터 세트, 테스트 피처/레이블 데이터 세트 반환. 테스트 데이터 세트를 전체 30%인 Stratified 방식으로 추출.

```
print('학습 데이터 레이블 값 비율')
print(y_train.value_counts()/y_train.shape[0]*100)
print('테스트 데이터 레이블 값 비율')
print(y_test.value_counts()/y_test.shape[0]*100)
```

```
학습 데이터 레이블 값 비율
0    99.827451
1     0.172549
Name: Class, dtype: float64
테스트 데이터 레이블 값 비율
0    99.826785
1     0.173215
Name: Class, dtype: float64
```

```
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:,1]

# 3장에서 사용한 get_clf_eval() 함수 이용해 평가 수행
get_clf_eval(y_test, lr_pred, lr_pred_proba)
```

```
오차 행렬
[[85281    14]
 [   56    92]]
정확도: 0.9992, 정밀도: 0.8679, 재현율: 0.6216,    F1: 0.7244, AUC:0.9609
```

생성된 학습 데이터와 테스트 데이터가 큰 차이 없이 잘 분할됨.

로지스틱 회귀를 이용한 신용 카드 사기 여부 예측
: 재현율 0.6216, ROC-AUC 0.9609

EWHA EURON

#4.4 데이터 1차 가공 및 모델 학습/예측/평가

```
# 인자로 사이킷런의 Estimator 객체와 학습 테스트 데이터 세트를 입력 받아서 학습 예측 평가 수행
def get_model_train_eval(model, ftr_train=None, ftr_test=None, tgt_train=None, tgt_test=None):
  model.fit(ftr_train, tgt_train)
  pred=model.predict(ftr_test)
  pred_proba = model.predict_proba(ftr_test)[:,1]
  get_clf_eval(tgt_test, pred, pred_proba)
```

```
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=False)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test= X_test, tgt_train=y_train, tgt_test=y_test)
```

```
오차 행렬
[[85289     6]
 [   36   112]]
정확도: 0.9995, 정밀도: 0.9492, 재현율: 0.7568,    F1: 0.8421, AUC:0.9797
```

Get_model_train_eval()
: 인자로 사이킷런의 Estimator 객체와 학습/테스트 데이터 세트를 입력 받아 학습/예측/평가를 수행

로지스틱 회귀를 이용한 신용 카드 사기 여부 예측
: 재현율 0.6216, ROC-AUC 0.9609

LightGBM을 이용한 신용 카드 사기 여부 예측
: 재현율 0.7568, ROC-AUC 0.9797

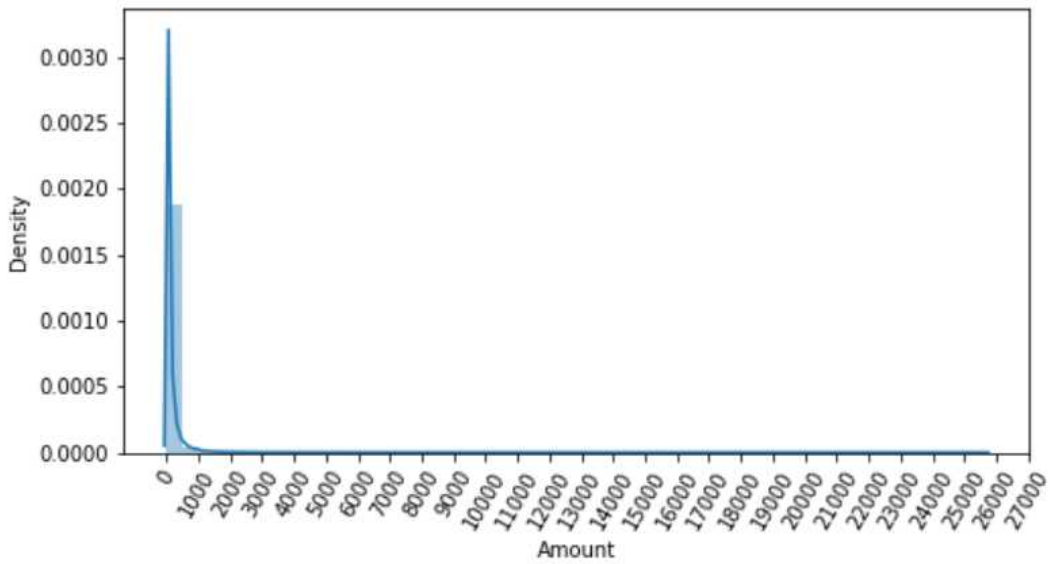
주의!!
LightGBM이 버전업되며 boost_from_average 파라미터의 디폴트값이 False에서 True로 변경됨.
극도로 불균형한 분포를 이루는 경우 boost_from_average=True설정은 재현률과 ROC-AUC 성능을 매우 크게 저하시키므로 주의!

EWHA EURON

#4.5 데이터 분포도 변환 후 모델 학습/예측/평가

```
import seaborn as sns
plt.figure(figsize=(8,4))
plt.xticks(range(0,30000,1000), rotation=60)
sns.distplot(card_df['Amount'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f87c6a73e90>
```



Creditcard.csv의 중요 피처 값 분포도 살펴보기

Amount 피처 시각화한 결과 카드 사용금액이 1000불 이하인 데이터가 대부분이며 27000불까지 드물지만 많은 금액을 사용한 경우가 발생하며 꼬리가 긴 형태의 분포 곡선을 가지고 있음.

EWHA EURON

#4.5 데이터 분포도 변환 후 모델 학습/예측/평가

```
from sklearn.preprocessing import StandardScaler
# 사이킷런의 StandardScaler을 이용해 정규 분포 현태로 Amount 피처값 변환하는 로직으로 수정.
def get_preprocessed_df(df=None):
  df_copy=df.copy()
  scaler=StandardScaler()
  amount_n = scaler.fit_transform(df_copy['Amount'].values.reshape(-1,1))
  # 변환된 amount를 amount_scaled로 피처명 변경 후 DataFrame맨 앞 칼럼으로 입력
  df_copy.insert(0,'Amount_Scaled', amount_n)
  df_copy.drop(['Time','Amount'], axis=1, inplace=True)
  return df_copy
```

```
# Amount를 정규 분포 형태로 변환 후 로지스틱 회귀 및 LightGBM 수행
X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)

print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)

print('### LightGBM 예측 성능 ###')
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[85281    14]
 [   58    90]]
정확도: 0.9992, 정밀도: 0.8654, 재현율: 0.6081,    F1: 0.7143, AUC:0.9702
### LightGBM 예측 성능 ###
오차 행렬
[[85263    32]
 [  107    41]]
정확도: 0.9984, 정밀도: 0.5616, 재현율: 0.2770,    F1: 0.3710, AUC:0.6383
```

Get_preprocessed_df(): StandardScaler 클래스를 이용해 Amount 피처를 정규 분포 형태로 변환.

데이터 분포 변환한 이후 로지스틱 회귀와 LightGBM 모델 모두 이전과 비교해 크게 성능이 개선되지는 않음.

```
def get_preprocessed_df(df=None):
  df_copy = df.copy()
  # 넘파이의 log1p()를 이용해 Amount를 로그 변환
  amount_n=np.log1p(df_copy['Amount'])
  df_copy.insert(0,'Amount_Scaled', amount_n)
  df_copy.drop(['Time','Amount'], axis=1, inplace=True)
  return df_copy
```

```
X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)

print('### 로지스틱 회귀 예측 성능 ###')
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)

print('### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

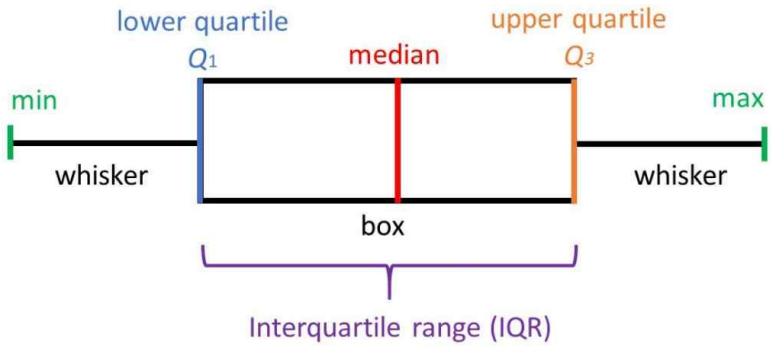
```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[85283    12]
 [   59    89]]
정확도: 0.9992, 정밀도: 0.8812, 재현율: 0.6014,    F1: 0.7149, AUC:0.9727
### LightGBM 예측 성능 ###
오차 행렬
[[85266    29]
 [   65    83]]
정확도: 0.9989, 정밀도: 0.7411, 재현율: 0.5608,    F1: 0.6385, AUC:0.7801
```

Get_preprocessed_df(): log1p()를 이용해 Amount 피처를 로그 변환.

데이터 분포 변환한 이후 로지스틱 회귀와 LightGBM 모델 모두 이전과 비교해 약간씩 성능이 개선됨.

EWHA EURON

#4.6 이상치 데이터 제거 후 모델 학습/예측/평가

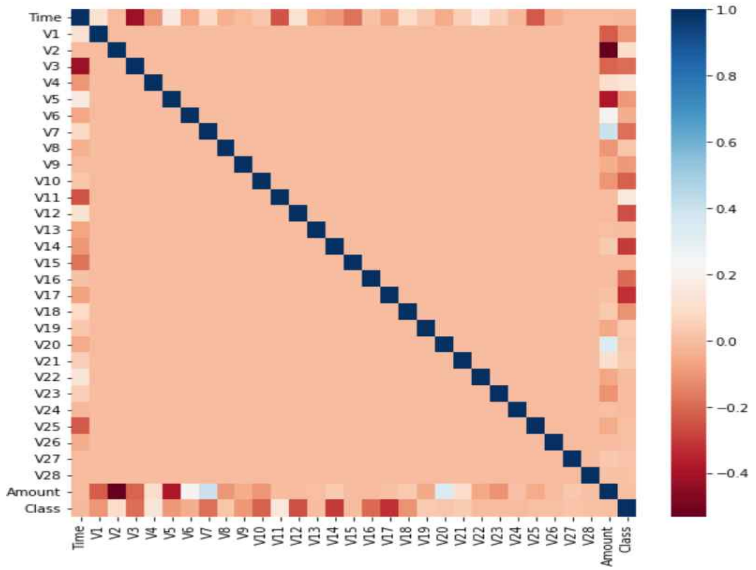


이상치 데이터(Outlier): 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터.

IQR 방식: 사분위 값의 편차를 이용하는 기법.
- 사분위: 전체 데이터를 값이 높은 순으로 정렬하고 이를 1/4씩으로 구간 분할.
- IQR: Q1~Q3의 구간 범위.
- 최댓값: Q3 + IQR * 1.5
- 최솟값: Q1 – IQR * 1.5

```
import seaborn as sns

plt.figure(figsize=(9,9))
corr=card_df.corr()
sns.heatmap(corr, cmap='RdBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f87c7937750>
```



검출할 이상치 데이터 선택
: 결정값과 가장 상관성이 높은 피처 위주로 이상치 검출

Cmap='RdBu' : 양의 상관관계가 높을수록 색깔이 진한 파랑, 음의 상관관계가 높을수록 색깔이 진한 빨강.

음의 상관관계가 높아보이는 V14 피처 이상치 제거하기로 선택.

#4.6 이상치 데이터 제거 후 모델 학습/예측/평가

```
import numpy as np

def get_outlier(df=None, column=None, weight=1.5):
  # fraud에 해당하는 column 데이터만 추출, 1/4 분위와 3/4 분위 지점을 np.percentile로 구함
  fraud=df[df['Class']==1][column]
  quantile_25=np.percentile(fraud.values,25)
  quantile_75=np.percentile(fraud.values,75)
  # IQR을 구하고 IQR에 1.5를 곱해 최댓값 최솟값 지점 구함
  iqr=quantile_75-quantile_25
  iqr_weight=iqr*weight
  lowest_val= quantile_25 - iqr_weight
  highest_val = quantile_75 + iqr_weight
  # 최댓값보다 크거나 최솟값보다 작은 값을 이상치 데이터로 설정하고 DataFrame index 반환
  outlier_index = fraud[(fraud<lowest_val)|(fraud>highest_val)].index
  return outlier_index
```

```
outlier_index= get_outlier(df=card_df, column='V14', weight=1.5)
print('이상치 데이터 인덱스:', outlier_index)
```

```
이상치 데이터 인덱스: Int64Index([8296, 8615, 9035, 9252], dtype='int64')
```

Get_outlier: 인자로 DataFrame과 이상치 검출할 칼럼 입력 받음. 넘파이의 percentile() 사용해 IQR 계산. 이상치가 있는 DataFrame 인덱스 반환.

총 4개 데이터가 이상치로 추출됨.

```
# get_processed_df()를 로그 변환 후 V14 피처의 이상치 데이터를 삭제하는 로직으로 변경
def get_preprocessed_df(df=None):
  df_copy = df.copy()
  amount_n=np.log1p(df_copy['Amount'])
  df_copy.insert(0,'Amount_Scaled', amount_n)
  df_copy.drop(['Time','Amount'], axis=1, inplace=True)
  # 이상치 데이터 삭제하는 로직 추가
  outlier_index=get_outlier(df=df_copy, column='V14', weight=1.5)
  df_copy. drop(outlier_index, axis=0, inplace=True)
  return df_copy

X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)
print('### 로지스틱 회귀 예측 성능 ###')
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
print('### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)
```

```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[85281    14]
 [   48    98]]
정확도: 0.9993, 정밀도: 0.8750, 재현율: 0.6712,    F1: 0.7597, AUC:0.9743
### LightGBM 예측 성능 ###
오차 행렬
[[85233    62]
 [  110    36]]
정확도: 0.9980, 정밀도: 0.3673, 재현율: 0.2466,    F1: 0.2951, AUC:0.6229
```

Get_processed_df(): 이상치 추출하고 이를 삭제하는 로직 추가.

로지스틱 회귀와 LightGBM 모두 예측 성능 크게 향상됨.

EWHA EURON

#4.7 SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
X_train_over, y_train_over = smote.fit_resample(X_train, y_train)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트:', X_train.shape, y_train.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트:', X_train_over.shape, y_train_over.shape)
print('SMOTE 적용 후 레이블 값 분포: ₩n', pd.Series(y_train_over).value_counts())
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트: (199362, 29) (199362,)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트: (398040, 29) (398040,)
SMOTE 적용 후 레이블 값 분포:
 0    199020
1    199020
Name: Class, dtype: int64
```

```
lr_clf = LogisticRegression()
# ftr_train 과 tgt_train 인자값이 SMOTE 증식된 X_train_over와 y_train_over로 변경됨에 유의
get_model_train_eval(lr_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over, tgt_test=y_test)
```

```
오차 행렬
[[82937  2358]
 [   11   135]]
정확도: 0.9723, 정밀도: 0.0542, 재현율: 0.9247,    F1: 0.1023, AUC:0.9737
```

주의!!
SMOTE 적용할 때 반드시 학습 데이터 세트만 오버 샘플링 해야함.

SMOTE 적용 후 레이블 분포가 동일함 확인.

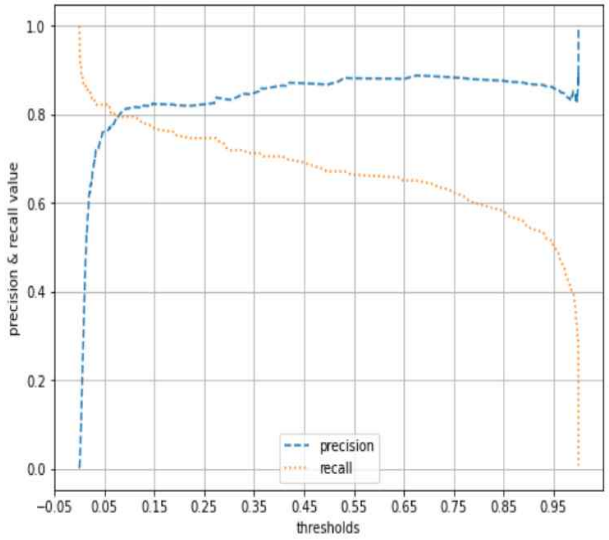
로지스틱 회귀를 이용한 신용 카드 사기 여부 예측
: 재현율 0.9247, ROC-AUC 0.9737, 정밀도: 0.0542

재현율이 크게 증가하지만 정밀도가 아주 급격하게 저하됨.
-> 로지스틱 회귀 모델이 오버 샘플링으로 인해 실제 원본 데이터의 유형보다 너무 많은 Class=1 데이터를 학습하며 실제 테스트 데이터 세트에서 예측을 지나치게 Class=1로 적용했기 때문.

#4.7 SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

```
# Precision-Recall Curve Plot 그리기
def precision_recall_curve_plot(y_test, pred_proba):
    # threshold ndarray와 이 threshold에 따른 정밀도, 재현율 ndarray 추출
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba)

    # x축을 threshold, y축을 정밀도, 재현율로 그래프 그리기
    plt.figure(figsize=(8, 6))
    thresholds_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[:thresholds_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[:thresholds_boundary], linestyle=':', label='recall')

    # threshold의 값 X축의 scale을 0.1 단위로 변경
    stard, end = plt.xlim()
    plt.xticks(np.round(np.arange(stard, end, 0.1), 2))

    plt.xlim()
    plt.xlabel('thresholds')
    plt.ylabel('precision & recall value')
    plt.legend()
    plt.grid()

precision_recall_curve_plot(y_test, lr_clf.predict_proba(X_test)[:,1])
```

https://foxtrotin.tistory.com/44




```
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=False)
get_model_train_eval(lgbm_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over, tgt_test=y_test)
```

```
오차 행렬
[[85286     9]
 [   22   124]]
정확도: 0.9996, 정밀도: 0.9323, 재현율: 0.8493,    F1: 0.8889, AUC:0.9789
```

임계값 0.99 이하: 재현율이 매우 좋고 정밀도가 극단적으로 낮음
임계값 0.99 이상: 재현율이 매우 낮고 정밀도가 높음

분류 결정 임계값을 조정하더라도 임계값의 민감도가 너무 심해 올바른 재현율 및 정밀도 성능을 얻을 수 없어 로지스틱 회귀 모델의 경우 SMOTE 적용으로 올바른 예측 모델이 생성되지 않음.

#4.7 SMOTE 오버 샘플링 적용 후 모델 학습/예측/평가

```
lgbm_clf=LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=False)
get_model_train_eval(lgbm_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over, tgt_test=y_test)
```

```
오차 행렬
[[85286     9]
 [   22   124]]
정확도: 0.9996, 정밀도: 0.9323, 재현율: 0.8493,    F1: 0.8889, AUC:0.9789
```

LightGBM을 이용한 신용 카드 사기 여부 예측
: 재현율 0.8493, 정밀도 0.9323

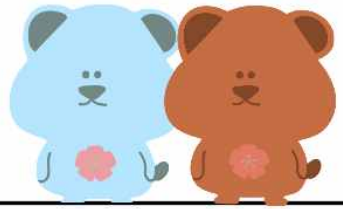
SMOTE 적용하면 재현율은 높이지나 정밀도는 낮아지는 것이 일반적.
좋은 SMOTE 패키지일수록 재현율 증가율은 높이고 정밀도 감소율은 낮출 수 있도록 효과적으로 데이터 증식.

EWHA EURON

#4.8 결론

데이터 가공 유형	머신러닝 알고리즘	정밀도	재현율	ROC-AUC
원본 데이터 가공 없음	로지스틱 회귀	0.8679	0.6216	0.9609
	LightGBM	0.9492	0.7568	0.9797
데이터 로그 변환	로지스틱 회귀	0.8812	0.6014	0.9727
	LightGBM	0.7411	0.5608	0.7801
이상치 데이터 제거	로지스틱 회귀	0.8750	0.6712	0.9743
	LightGBM	0.3673	0.2466	0.6229
SMOTE 오버 샘플링	로지스틱 회귀	0.0542	0.9247	0.9737
	LightGBM	0.9323	0.8493	0.9789

THANK YOU



EWHA EURON