

앙상블 Ensemble 2.

- 앙상블 실습
- Kaggle 산탄데르 은행 고객 분류 실습
- Kaggle 신용 카드 사기 검출 실습

Kaggle 산탄데르 은행 고객 분류

실습의 학습 목표

- 370개의 피처를 가진 고객 데이터를 학습하여 불만 고객, 만족 고객을 분류하는 예측기를 구현할 수 있다.
- 이상치 데이터를 판별하고 이를 처리할 수 있다.
- XGboost 알고리즘을 활용하여 머신러닝 모델을 구현할 수 있다.








[실습] 산탄데르 은행 고객 분류 캐글 데이터



Santander Customer Satisfaction

Which customers are happy customers?

\$60,000 · 5,123 teams · 3 years ago

Data (8 MB)		API	 kaggle competitions download -c santander-custom...	?	 Download All	
Data Sources		About this file		Columns		
 sample_submission.... 75.8k x 2		No description yet		# ID		
 test.csv 75.8k x 370				# var3		
 train.csv 76.0k x 371 				# var15		

[실습] 산탄데르 은행 고객 분류

개요: 370개의 피처로 주어진 데이터 세트 기반에서 고객 만족 여부를 예측

클래스 레이블 :

불만 고객 : 1

만족 고객 : 0

성능평가:ROC-AUC

train.csv를 다운로드 받은 후, 'train_Santander.csv'로 이름 변경

[실습] 산탄데르 은행 고객 분류

1. 데이터 전처리

클래스 레이블 비율 확인.

var3컬럼의 -999999를 모두 2로 변경

ID 컬럼 제거

2. 기본 모델 구축

test size=0.2

random_state=0

n_estimators=500

random_state=156

3. 하이퍼 파라미터 튜닝(1차, 2차로 나눔)

min_child_weight: 트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 weight총합. 클수록 분할을 자제(default=1)

sub_sample: 트리를 생성하는데 필요한 데이터의 비율(default=1)

colsample_bytree: 트리생성에 필요한 컬럼을 임의로 샘플링하는데 사용

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

데이터 전처리

```
▶ 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)

- 레이블 (Target) 의 분포 확인(불만족 고객 비율)

```
▶ 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
```

```

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

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

```

피쳐 데이터 shape:(76020, 369)

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_features, y_labels,
                                                    test_size=0.2, random_state=0)

train_cnt = y_train.count()
test_cnt = y_test.count()
print('학습 세트 Shape:{0}, 테스트 세트 Shape:{1}'.format(X_train.shape , X_test.shape))

print(' 학습 세트 레이블 값 분포 비율') # 불균형 데이터세트이므로 확인 필요
print(y_train.value_counts()/train_cnt)
print('ㄴ 테스트 세트 레이블 값 분포 비율')
print(y_test.value_counts()/test_cnt)

```



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

```
# n_estimators=500, random state=156
```

```
xgb_clf = XGBClassifier(n_estimators=500, random_state=156, ##
                        |tree_method='gpu_hist', use_label_encoder=False)
```

```
# 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)])
```

```
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))
```

첫 번째 하이퍼파라미터 튜닝 (0.8404 -> ??)

```
▶ from sklearn.model_selection import GridSearchCV

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

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))
```

두번째 하이퍼파라미터 튜닝.(0. 8455 -> 0.8453)

```
▶ # 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)  
  
# evaluation metric을 auc로, early stopping은 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))
```

- 시각화


```
▶ from xgboost import plot_importance
import matplotlib.pyplot as plt


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


-
- LightGBM으로도 모델을 구축하여 XGBoost 모델과 비교하라



Kaggle 신용카드 사기 검출


 Dataset




 8357


Credit Card Fraud Detection


Anonymized credit card transactions labeled as fraudulent or genuine


 Machine Learning Group - ULB • updated 4 years ago (Version 3)

[Data](#) [Tasks \(10\)](#) [Code \(3,388\)](#) [Discussion \(94\)](#) [Activity](#) [Metadata](#)

[Download \(151 MB\)](#) [New Notebook](#) 

 **Usability** 8.5

 **License** Database: Open Database, Contents: Database Contents

 **Tags** finance, crime


Description

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.



실습의 학습 목표

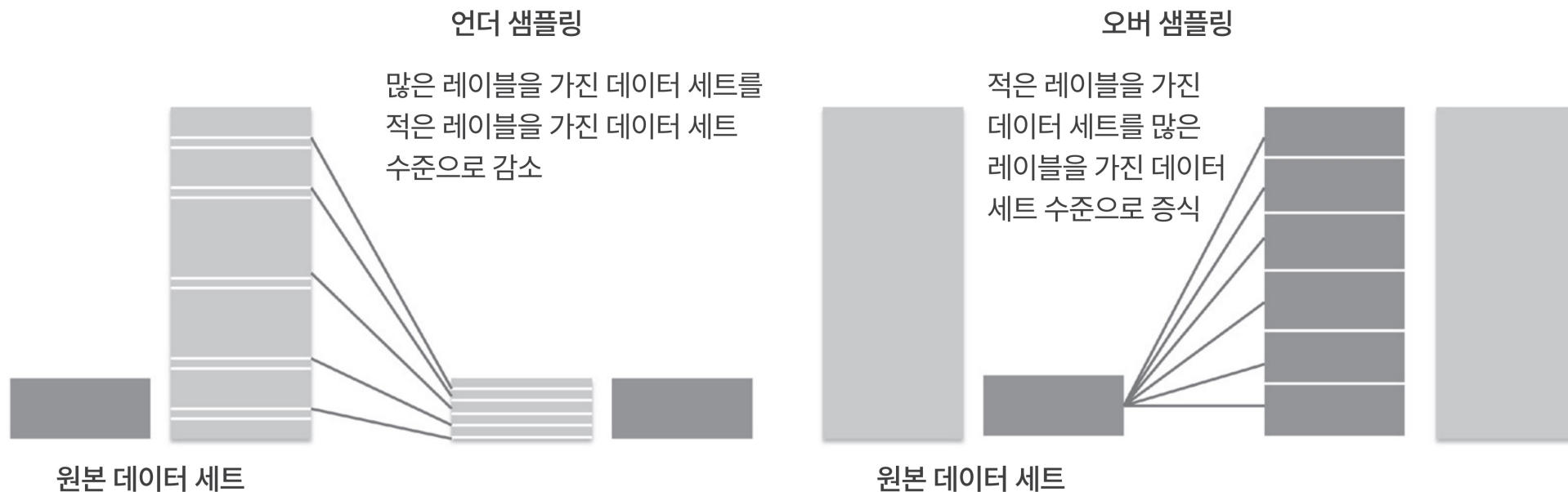
- 오버샘플링과 언더샘플링의 개념을 이해하고 sklearn에서 이를 구현하여 데이터를 준비할 수 있다
- 이상치 데이터의 개념을 이해하고 확보한 데이터셋에서 이를 구별하여 처리할 수 있다.
- 동일한 데이터셋을 대상으로 2가지 이상의 분류 알고리즘(로지스틱회귀, LGBM)을 적용해 본 후 모델별 성능을 비교하고 최선의 모델을 선택할 수 있다.

오버샘플링Oversampling VS 언더샘플링Undersampling

극도의 불균형 데이터세트의 학습: 학습을 제대로 수행하기 어려움

오버샘플링Oversampling, 언더샘플링Undersampling

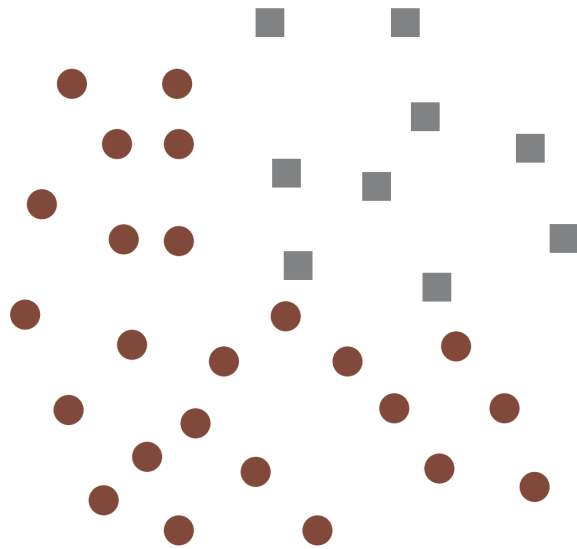
지도학습에서 극도로 불균형한 레이블 값의 분포로 인한 문제점을 해결하기 위해 적절한 학습 데이터를 확보하는 방안



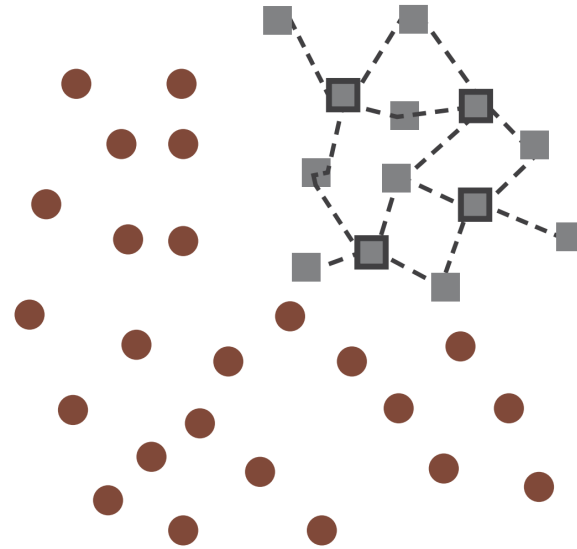
<이미지 출처: 파이썬 머신러닝 완벽가이드, 위키북스, 2021, p263>

오버샘플링 : SMOTE방식

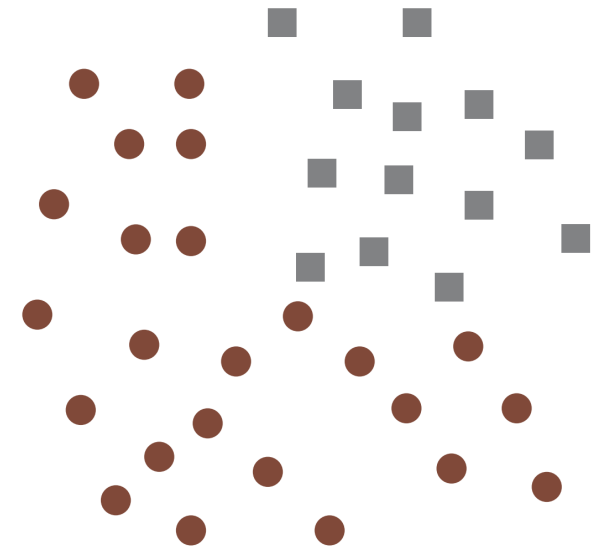
① 원본 데이터



② K 최근접 이웃으로 데이터 신규 증식 대상 설정



③ 신규 증식하여 오버 샘플링 완료



< SMOTE 수행 절차 >

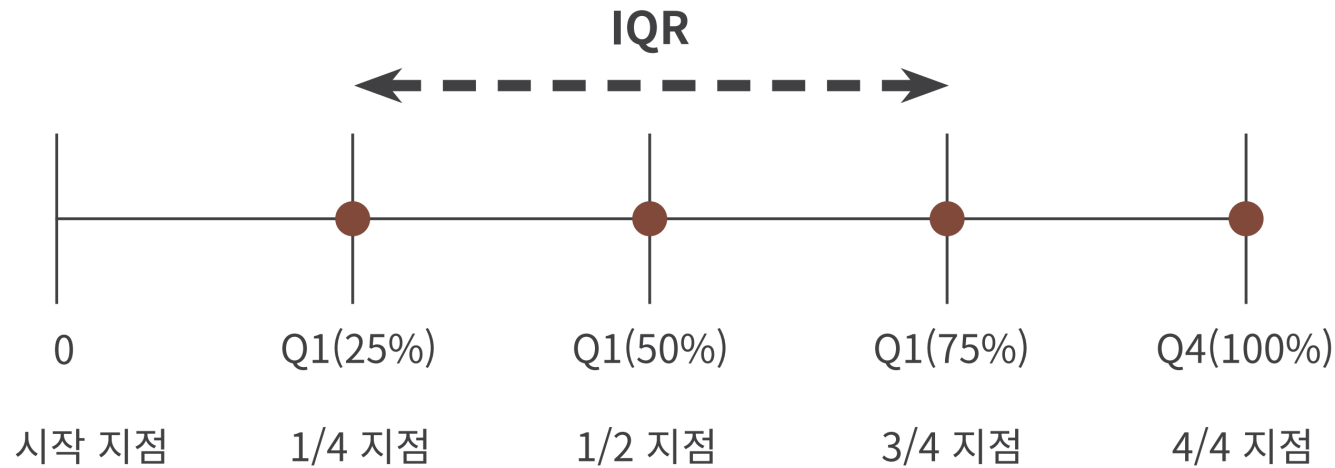
이상치(Outlier) 데이터

이상치 데이터: 전체 데이터의 패턴에서 벗어난 이상 값을 가진 데이터
이상치로 인해 머신러닝 모델의 성능에 영향을 받는 경우가 발생하기 쉽다.

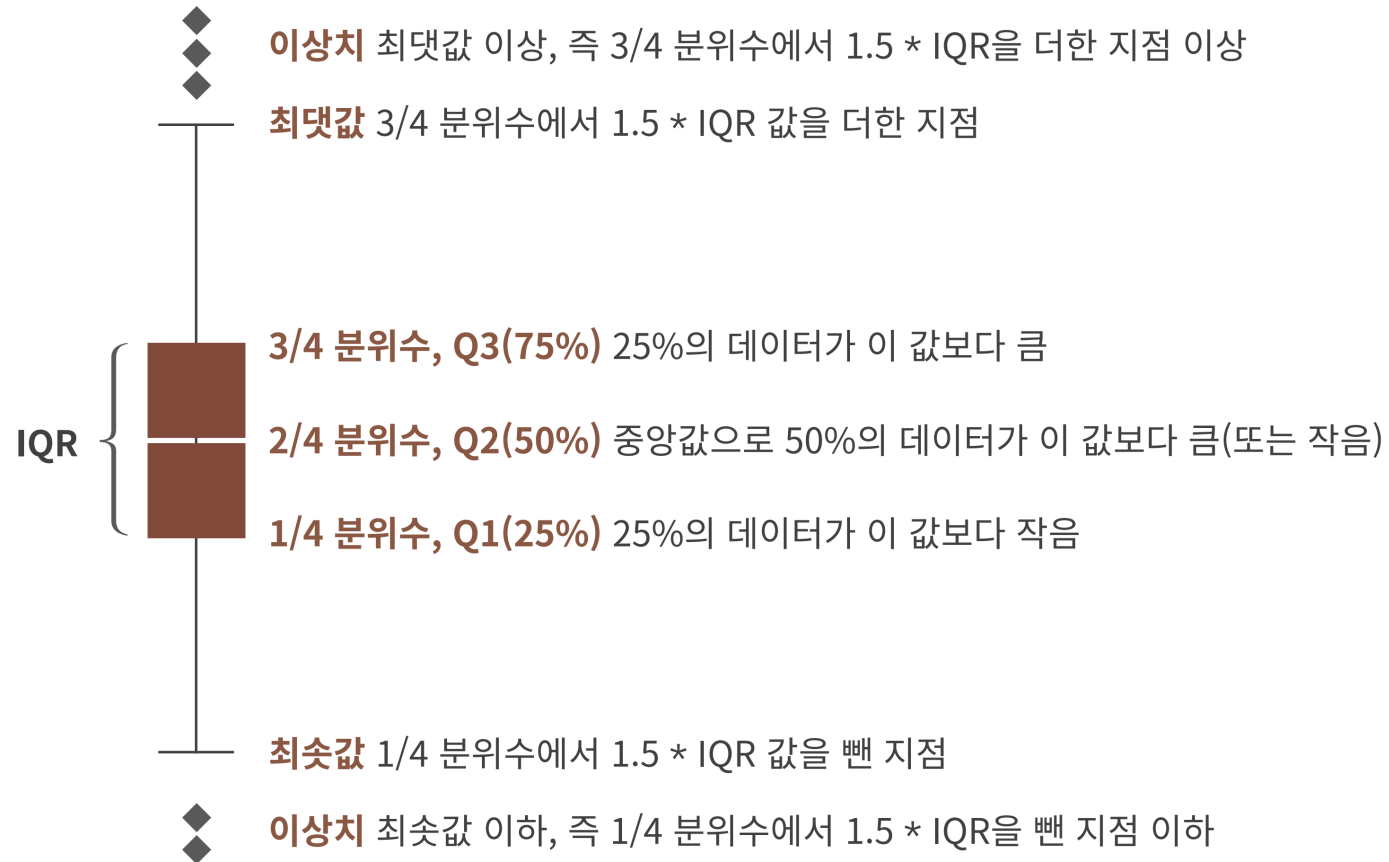
이상치를 찾는 방법: IQR(Inter Quantile Range)방식 , 사분위(Quantile)값의 편차를 이용하는 기법,
박스플롯으로 시각화

사분위: 전체 데이터를 정렬하고, 25%씩 구간으로 분할(순위제)

IQR: Q1 ~ Q3 구간



이상치(Outlier) 판단 기준



Kaggle 신용카드 사기 검출

kaggle 신용카드 사기 검출 데이터세트

- 데이터셋 위치 <https://www.kaggle.com/mlg-ulb/creditcardfraud>
- 사기(Fraud) 트랜잭션: 1 (전체의 0.172%)
- 정상 신용카드 트랜잭션: 0 성능평가: ROC-AUC (대부분이 정상 거래이고 사기거래 데이터는 소수인 극도로 불균형 데이터세트)
- 데이터셋 다운로드: <https://www.kaggle.com/mlg-ulb/creditcardfraud?select=creditcard.csv>
- 다운로드 후 데이터셋 압축풀기 creditcard.zip -> creditcard.csv
- LogisticRegression, LightGBM으로 예측

분석 순서

1. 로지스틱 분류
2. LightGBM 분류
3. 데이터 분포 왜곡 개선 후 모델 각각 재적용
 - 1) 정규 분포 형태로 변경 후 모델 각각 재적용
 - 2) 로그 변환 후 모델 각각 재적용
4. 이상치 데이터 제거 후 모델 각각 재적용
5. SMOTE 오버 샘플링 후 모델 각각 재적용

데이터 일차 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_fraud/creditcard.csv')
card_df.head(3)
```


평가함수 get_clf_eval()작성

```
from sklearn.metrics import confusion_matrix, accuracy_score,
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score

def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    f1 = f1_score(y_test, pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f}, #
    F1: {3:.4f}, AUC: {4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

데이터 준비

```
from sklearn.model_selection import train_test_split

df_copy = card_df.copy()
df_copy.drop('Time', axis=1, inplace=True)

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)

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)
```

1. LogisticRegression 모델을 활용한 분류

```
▶ from sklearn.linear_model import LogisticRegression
  # solver, max_iter 값 변경
  lr_clf = LogisticRegression(solver='lbfgs', max_iter=1000)
  lr_clf.fit(X_train, y_train)

  lr_pred = lr_clf.predict(X_test)
  lr_pred_proba = lr_clf.predict_proba(X_test)[:, 1]

  # get_clf_eval() 함수를 이용하여 평가 수행.
  get_clf_eval(y_test, lr_pred, lr_pred_proba)
```

2. LightGBM 모델 학습 및 평가

```
▶ from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=1000, #
                          num_leaves=64, n_jobs=-1, boost_from_average=False)

lgbm_clf.fit(X_train, y_train)

pred = lgbm_clf.predict(X_test)
pred_proba = lgbm_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba)
```

-
- 왜곡된 분포도를 가지는 데이터를 재가공한뒤 모델 재테스트

▶ *# 사용액 데이터의 분포 확인*

```
import seaborn as sns
```

```
plt.figure(figsize=(8, 4))  
plt.xticks(range(0, 30000, 1000), rotation=60)  
plt.hist(card_df['Amount'], bins=300);
```

1. 사이킷런의 `StandardScaler`를 이용하여 정규분포 형태로 `Amount` 피처값 변환하는 로직으로 수정.

```
from sklearn.preprocessing import StandardScaler

# Amount를 정규분포 형태로 변환 후 로지스틱 회귀 및 LightGBM 수행.
df_copy = card_df.copy()
scaler = StandardScaler()
amount_n = scaler.fit_transform(df_copy['Amount'].values.reshape(-1, 1))

# 변환된 Amount를 Amount_Scaled로 피처명 변경후 DataFrame맨 앞 컬럼으로 입력
df_copy.insert(0, 'Amount_Scaled', amount_n)

# 기존 Time, Amount 피처 삭제
df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)

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)
```

```
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegression(solver='lbfgs', max_iter=1000)
lr_clf.fit(X_train, y_train)

lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:, 1]

# get_clf_eval() 함수를 이용하여 평가 수행.
get_clf_eval(y_test, lr_pred, lr_pred_proba)

print('### LightGBM 예측 성능 ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, #
                          num_leaves=64, n_jobs=-1, boost_from_average=False)
lgbm_clf.fit(X_train, y_train)

pred = lgbm_clf.predict(X_test)
pred_proba = lgbm_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba)
```

- 2. numpy log1p를 사용하여 Amount 피쳐값을 로그 변환하는 로직으로 수정.
- 로그변환: 데이터 분포도가 심하게 왜곡되어 있을 경우 적용.
- 원래 값을 log 값으로 변환해 상대적으로 작은 값으로 변경, 데이터 분포의 왜곡을 개선

```
▶ # Amount를 로그 변환하는 로직으로 수정
df_copy = card_df.copy()

amount_n = np.log1p(df_copy['Amount'])
# 변환된 Amount를 Amount_Scaled로 피쳐명 변경후 DataFrame맨 앞 컬럼으로 입력
df_copy.insert(0, 'Amount_Scaled', amount_n)
# 기존 Time, Amount 피쳐 삭제
df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)

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)
```



```
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegression(solver='lbfgs', max_iter=1000)
lr_clf.fit(X_train, y_train)

lr_pred = lr_clf.predict(X_test)
lr_pred_proba = lr_clf.predict_proba(X_test)[:, 1]

# get_clf_eval() 함수를 이용하여 평가 수행.
get_clf_eval(y_test, lr_pred, lr_pred_proba)

print('### LightGBM 예측 성능 ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, #
                          num_leaves=64, n_jobs=-1, boost_from_average=False)
lgbm_clf.fit(X_train, y_train)

pred = lgbm_clf.predict(X_test)
pred_proba = lgbm_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba)
```

4. 이상치(Outlier) 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]
    #print(fraud.index)
    quantile_25 = np.percentile(fraud.values, 25)
    quantile_75 = np.percentile(fraud.values, 75)
    # IQR을 구하고, IQR에 1.5를 곱하여 최대값과 최소값 지점 구함.
    iqr = quantile_75 - quantile_25
    iqr_weight = iqr * weight
    lowest_val = quantile_25 - iqr_weight
    highest_val = quantile_75 + iqr_weight
    # 최대값 보다 크거나, 최소값 보다 작은 값을 아웃라이어로 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)
```

'Amount' 열의 데이터를 로그 변환 후 V14 피처의 이상치 데이터를 삭제하는 로직으로 변경.

```
df_copy = card_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)
```

데이터 준비

```
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)
```

```
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegression(solver='lbfgs', max_iter=1000)
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() 함수를 이용하여 평가 수행.
get_clf_eval(y_test, lr_pred, lr_pred_proba)
```

```
print('### LightGBM 예측 성능 ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, #
                          num_leaves=64, n_jobs=-1, boost_from_average=False)
lgbm_clf.fit(X_train, y_train)
```

```
pred = lgbm_clf.predict(X_test)
pred_proba = lgbm_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba)
```

5. 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 피쳐/레이블 데이터 세트: ', X_train.shape, y_train.shape)
print('SMOTE 적용 후 학습용 피쳐/레이블 데이터 세트: ', X_train_over.shape, y_train_over.shape)
# 레이블의 0과 1의 분포가 동일해짐. (이전에는 극단적 불균형)
print("SMOTE 적용 전 레이블 값 분포: \#n", pd.Series(y_train).value_counts())
print('SMOTE 적용 후 레이블 값 분포: \#n', pd.Series(y_train_over).value_counts())
```

- LogisticRegression() 모델을 사용한 분류

```
▶ lr_clf = LogisticRegression()  
# ftr_train과 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)  
print('### 로지스틱 회귀 예측 성능 ###')  
lr_clf = LogisticRegression(solver='lbfgs', max_iter=1000)  
lr_clf.fit(X_train_over, y_train_over)  
  
lr_pred = lr_clf.predict(X_test)  
lr_pred_proba = lr_clf.predict_proba(X_test)[:, 1]  
  
# get_clf_eval() 함수를 이용하여 평가 수행.  
get_clf_eval(y_test, lr_pred, lr_pred_proba)
```

- LGBM() 모델을 사용한 분류

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=False)

print('### LightGBM 예측 성능 ###')
# ftr_train과 tgt_train 인자값이 SMOTE 증식된 X_train_over와 y_train_over로 변경됨에 유의
lgbm_clf.fit(X_train_over, y_train_over)

pred = lgbm_clf.predict(X_test)
pred_proba = lgbm_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test, pred, pred_proba)
```

여러 가지 분석 방법 비교

데이터 가공유형	머신러닝 알고리즘	정밀도	재현율	ROC-AUC
원본 데이터 가공 없음	로지스틱 회귀	0.8667	0.6149	0.9703
	LightGBM	0.9573	0.7568	0.9790
데이터 로그 변환	로지스틱 회귀	0.8812	0.6014	0.9727
	LightGBM	0.9576	0.7635	0.9796
이상치 데이터 제거	로지스틱 회귀	0.8750	0.6712	0.9743
	LightGBM	0.9603	0.8288	0.9780
SMOTE오버 샘플링	로지스틱 회귀	0.0542	0.9247	0.9737
	LightGBM	0.9118	0.8493	0.9814

