09 분류실습 - 캐글 신용카드 사기 검출

class는 0이 사기가 아닌 정상적인 신용카드 트랜잭션 데이터, 1은 신용카드 사기 트랜잭션을 의미한다
전체 데이터의 0.172%가 사기 트랜잭션이다
사기 검출이나 이상 검출 같은 데이터 세트는 이처럼 불균형한 분포를 가지기 쉽다

언더 샘플링과 오버 샘플링의 이해

지도학습에서 불균형한 레이블 값 분포로 인한 문제 해결을 위해 적절한 학습 데이터를 확보하는 방안

- 언더 샘플링: 많은 데이터 세트를 적은 데이터 세트 수준으로 감소시키는 방식
  정상 레이블 데이터가 10000건, 이상 레이블 데이터가 100건 있으면 정상 레이블 데이터를 100건으로 줄여 버리는 방식
  과도하게 정상 레이블로 학습 및 예측하는 부작용 개선 가능
  너무 많은 정상 레이블 데이터를 감소시키기 때문에 정상 레이블의 경우 오히려 제대로 된 학습을 수행할 수 없음

- 오버 샘플링: 이상 데이터와 같이 적은 데이터 세트를 증식하여 학습을 위한 충분한 데이터를 확보하는 방법
  동일한 데이터를 단순히 증식하는 것은 과적합이 되기 때문에 원본 데이터의 피처 값들을 아주 약간만 변경하여 증식
  SMOTE 방법: 적은 데이터 세트에 있는 개별 데이터의 K 최근접 이웃을 찾아 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어서 기존 데이터와 약간 차이가 나는 새로운 데이터들을 생성하는 방식

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.csv')
card_df.head(3)
```

get_preprocessed_df(): 불필요한 Time 피처만 삭제

```
from sklearn.model_selection import train_test_split

def get_preprocessed_df(df=None):
    df_copy = df.copy()
    df_copy.drop('Time', axis=1, inplace=True)
    return df_copy
```

get_train_test_dataset(): 학습 피처/레이블 데이터 세트, 테스트 피처/레이블 데이터 세트 반환
테스트 데이터 세트를 전체의 30%인 stratified 방식으로 추출해 학습 데이터 세트와 테스트 데이터 세트의 레이블 값 분포도를 서로 동일하게 만든다

```
def get_train_test_dataset(df=None):
    df_copy = get_preprocessed_df(df)
    X_features = df_copy.iloc[:, :-1]
    y_target = df_copy.iloc[:, -1]
    X_train, X_test, y_train, y_test = W
    train_test_split(X_features, y_target, test_size=0.3, random_state=0, stratify=y_target)
    return X_train, X_test, y_train, y_test

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

```
#학습 데이터 세트와 테스트 데이터 세트의 레이블 값 비율 확인
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.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f
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 = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)

    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},₩
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

```
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]

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
```

get_model_train_eval(): 인자로 사이킷런의 estimator 객체와 학습/테스트 데이터 세트를 입력 받아서 학습 예측/평가를 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)
```

```
#LightGBM으로 모델 학습
from lightgbm import LGBMClassifier

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

```

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

데이터 분포도 변환 후 모델 학습/예측/평가

```
#amount 피처의 분포도 확인
import seaborn as sns

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

카드 사용금액이 1000불 이하인 데이터가 대부분이며 꼬리가 긴 형태의 분포 곡선을 가짐
amount를 표준 정규 분포 형태로 변환 후 로지스틱 회귀의 예측 성능을 측정

```
from sklearn.preprocessing import StandardScaler
# 사이킷런의 StandardScaler를 이용하여 정규분포 형태로 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))
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time','Amount'], axis=1, inplace=True)
    return df_copy
```

```
# Amount를 정규분포 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=

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

```
### 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
[[85289     6]
 [   36   112]]
정확도: 0.9995, 정밀도: 0.9492, 재현율: 0.7568,    F1: 0.8421, AUC:0.9773
```

```
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    # 넘파이의 log1p( )를 이용하여 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

print('### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_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
[[85290     5]
 [   35   113]]
정확도: 0.9995, 정밀도: 0.9576, 재현율: 0.7635,    F1: 0.8496, AUC:0.9786
```

이상치 데이터 제거 후 모델 학습/예측/평가

IQR을 이용해 이상치 데이터를 검출하는 방식은 보통 IQR에 1.5를 곱해서 생성된 범위를 이용해 최댓값과 최소값을 결정한 뒤 최댓값을 초과하거나 최솟값에 미달하는 데이터를 이상치로 간주

먼저 어떤 피처의 이상치 데이터를 검출할 것인지 선택이 필요
매우 많은 피처가 있을 경우 이들 중 결정값과 가장 상관성이 높은 피처들을 위주로 이상치를 검출하는 것이 좋다

```
#각 피처별로 상관도를 구한 뒤 시각화
import seaborn as sns
```

```
3
4 plt.figure(figsize=(9, 9))
5 corr = card_df.corr()
6 sns.heatmap(corr, cmap='RdBu')
```

양의 상관관계가 높을수록 진한 파란색에 가깝고 음의 상관관계가 높을수록 진한 빨간색에 가까움
결정 레이블인 class 피처와 음의 상관관계가 높은 피처는 v14와 v17이다

get_outlier(): 이상치가 있는 데이터 프레임 인덱스 반환

```
1 def get_outlier(df=None, column=None, weight=1.5):
2
3     fraud = df[df['Class']==1][column]
4     quantile_25 = np.percentile(fraud.values, 25)
5     quantile_75 = np.percentile(fraud.values, 75)
6
7     iqr = quantile_75 - quantile_25
8     iqr_weight = iqr * weight
9     lowest_val = quantile_25 - iqr_weight
10    highest_val = quantile_75 + iqr_weight
```

```
    outlier_index = fraud[(fraud < lowest_val) | (fraud > highest_val)].index
    return outlier_index
```

```
#v14 칼럼에서 이상치 데이터 찾기
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_processed_df(): 이상치를 삭제하고 데이터 가공

```
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)
    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
print('### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_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
[[85291     4]
 [   25   121]]
정확도: 0.9997, 정밀도: 0.9680, 재현율: 0.8288,    F1: 0.8930, AUC:0.9831
```

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_sample(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
```

SMOTE 적용 후 2배에 가까운 데이터가 증식
레이블 값이 0과 1의 분포가 동일하게 생성

```
lr_clf = LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train_over, ftr_test=X_test, tgt_train=y_train_over,
```

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

로지스틱 회귀 모델의 경우 오버 샘플링된 데이터로 학습할 때 재현율 증가하지만 정밀도가 저하
이는 실제 원본 데이터의 유형보다 너무나 많은 class=1 데이터를 학습하며 실제 테스트 데이터 세트에서 예측을 지나치게 class=1로 적용해 정밀도가 급격히 떨어진 것

정밀도와 재현율 곡선을 통해 시각적으로 문제 확인

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.metrics import precision_recall_curve
%matplotlib inline

def precision_recall_curve_plot(y_test , pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve( y_test, pred_proba_c1)

    plt.figure(figsize=(8,6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary],label='recall')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))

    plt.xlabel('Threshold value'); plt.ylabel('Precision and Recall value')
    plt.legend(); plt.grid()
    plt.show()
```

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

로지스틱 회귀 모델의 경우 SMOTE 적용 후 올바른 예측 모델이 생성되지 못함

```
#LightGBM 모델을 오버 샘플링된 데이터 세트로 학습/예측/평가
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=Fal
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
```

SMOTE 적용시 재현율은 높아지나 정밀도는 낮아지는 것이 일반적이다
좋은 SMOTE 패키지일수록 재현율 증가율은 높이고 정밀도 감소율은 낮추도록 데이터 증식

✓ 0초 오후 9:13에 완료됨