충북대학교
산업인공지능학과

어프렌티 스프로젝트
중간평가(후반)

2023254010
사수진

1. 데이터 선정과 문제 정의 및 이해

캐글의 "Credit Card Fraud Detection" 데이터셋을 기반으로 하여 이 데이터셋은 신용카드 거래의 여러 특징들과 해당 거래가 사기인지 정상 거래인지를 나타내는 레이블을 포함하고 있어 사기 탐지 문제에 관련된 데이터로 신용카드 거래 데이터를 기반으로 하여 사기 거래를 탐지하는 것입니다. 이는 신용카드 회사나 금융기관들이 신용카드 사기를 예방하고 고객들을 보호하기 위해 중요한 문제이며 데이터를 충분히 이해하고 명확한 문제 정의를 통해 적절한 머신러닝 모델을 개발하고 평가할 수 있을 것입니다.

1. 데이터 선정과 문제 정의 및 이해

```
import numpy as np
import pandas as pd
import lightgbm as lgb
from sklearn.metrics import confusion_matrix
import seaborn as sns
from matplotlib import pyplot as plt
```

✓ 9.9s

실습 데이터 - Credit Card Fraud Detection Data

- 데이터 설명 및 다운로드: https://www.kaggle.com/mlg-ulb/creditcardfraud
- 28개의 컬럼으로 구성
  - 28개의 변수(feature) 의미: V1, V2, ... V28은 PCA를 통해 얻은 주요 구성 요소입니다.
  - 다양한 데이터를 가지고
- 특징 'Amount': 거래금액
- 특징 'Class': 응답 변수로 사기일 경우 값 1, 그렇지 않으면 값 0을 사용합니다. → 타겟 기능, 클래스(0, 1; 이진분류)

```
data=pd.read_csv("./creditcard.csv")
data.tail()
```

✓ 3.9s

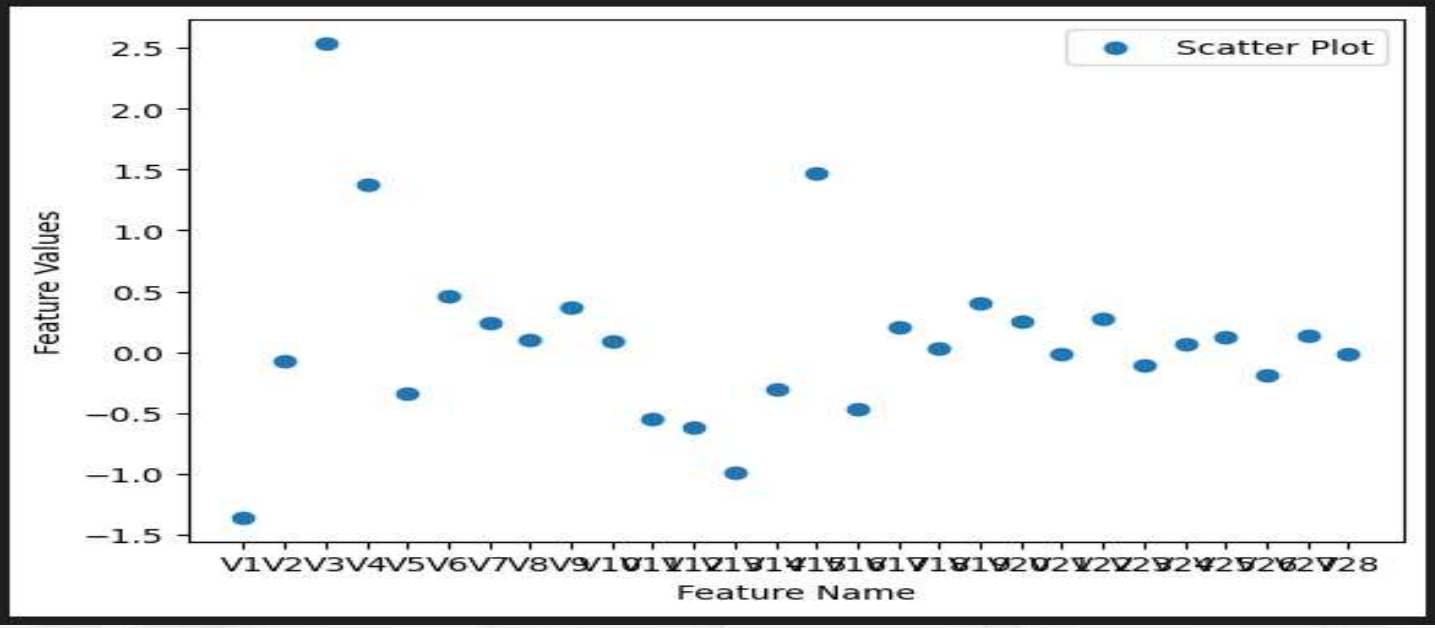
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

5 rows × 31 columns

2. 데이터 이해 및 시각화

```
# 데이터 이해 및 시각화1
plt.plot()
y = data_feature.iloc[0]
x = np.arange(len(y))
plt.scatter(x, y, label='Scatter Plot')
labels = data_feature.columns
plt.xticks(x, labels)
plt.xlabel('Feature Name')
plt.ylabel('Feature Values')
plt.legend()
plt.show()
```

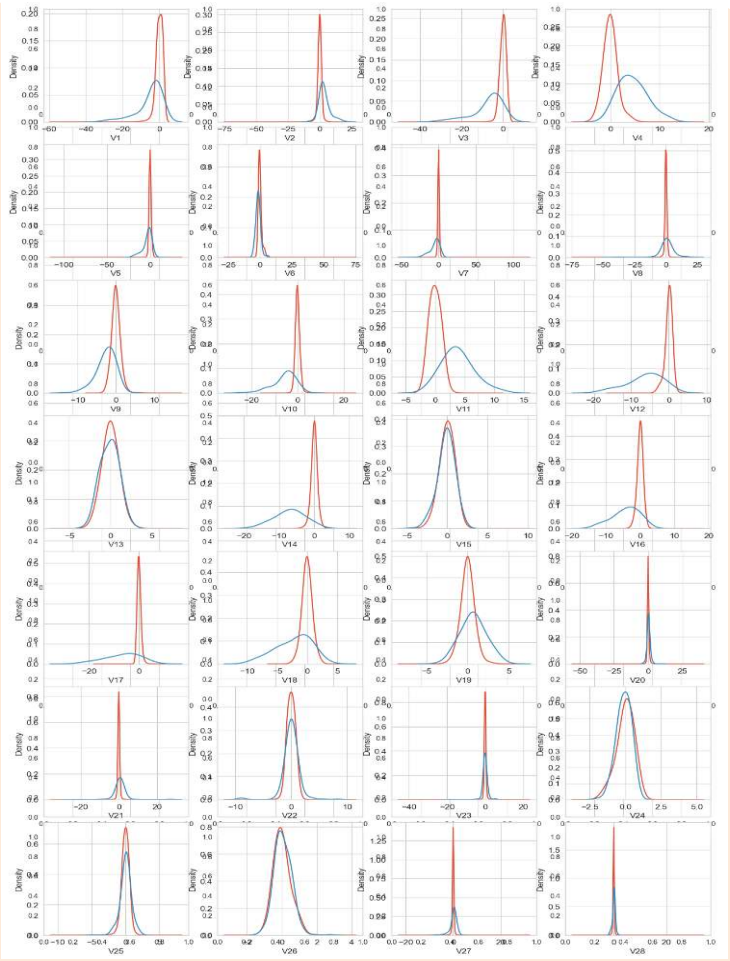
✓ 0.7s



2. 데이터 이해 및 시각화

```
var = data.columns.values[:-1] # V1 ~ V28(Class열 제외한 모든 열)
i = 0
t0 = data.loc[data['Class'] == 0] # Class : 0 인 행만 추출 --> 정상 거래
t1 = data.loc[data['Class'] == 1] # Class : 1 인 행만 추출 --> 사기 거래

sns.set_style('whitegrid') # 그래프 스타일 지정
plt.figure()
fig, ax = plt.subplots(8, 4, figsize = (16, 28)) # 축 지정

# KDE Plot
# Kernel Desntiy Estimation is a non-parametric estimation of a PDF,
# Probability Density Function of a random variable.
# It's a smoothing process of a discontinuous PDF

for feature in var:
    i += 1
    plt.subplot(7, 4, i) # 28개의 그래프
    sns.kdeplot(t0[feature], bw_method = 0.5, label = "Class = 0")
    sns.kdeplot(t1[feature], bw_method = 0.5, label = "Class = 1")
    plt.xlabel(feature, fontsize = 12) # 라벨 속성값
    locs, labels = plt.xticks()
    plt.tick_params(axis = 'both', which = 'major', labelsize = 12)
plt.show();
```

✓ 1m 27.2s



3. 데이터 전처리 및 정제

불필요한 데이터 제거

```
data=data.drop(['Time','Amount'],axis=1)
data.tail()
```

✓ 0.0s

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Class
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	1.475829	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.059616	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	0
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.001396	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	0
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.127434	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	0
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	...	0.382948	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	0

5 rows × 29 columns

```
# 데이터 내 NA값 여부 확인
data.isnull() # NA 확인
data.isnull().any() # 만약 존재한다면 0으로 대체 혹은, 해당 열을 제외하고 진행
```

✓ 0.0s

```
V1       False
V2       False
V3       False
V4       False
V5       False
V6       False
V7       False
V8       False
V9       False
V10      False
V11      False
V12      False
V13      False
V14      False
V15      False
V16      False
V17      False
V18      False
V19      False
V20      False
V21      False
V22      False
V23      False
V24      False
V25      False
V26      False
V27      False
V28      False
Class    False
dtype: bool
```

데이터 구조 확인

```
data.shape
```

✓ 0.0s

```
(284807, 29)
```

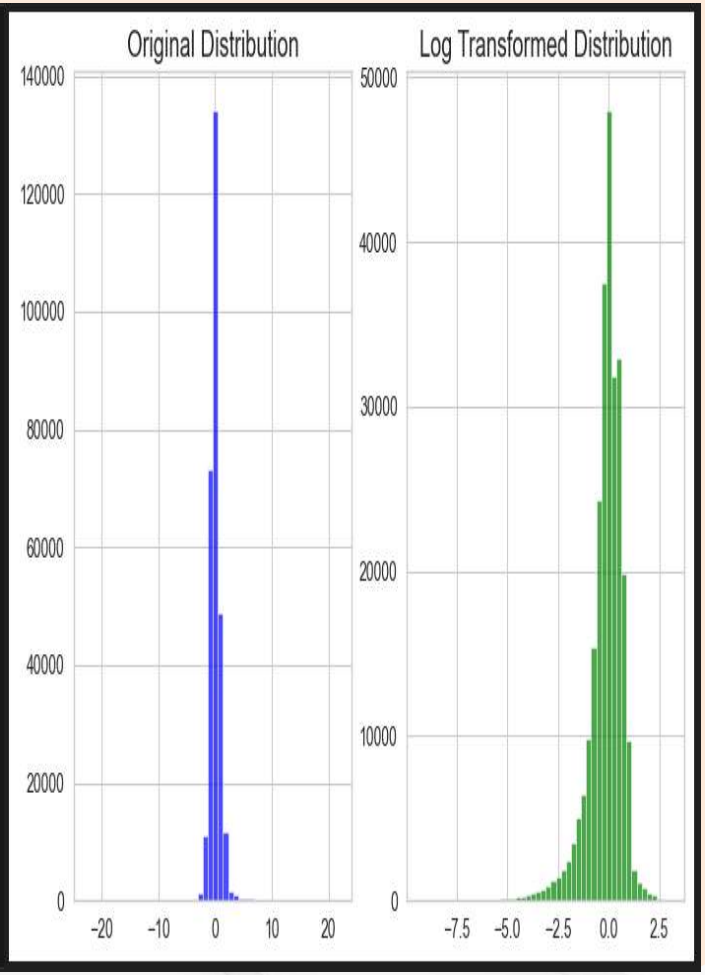
4. 텍스트와 범주형 특성 다루기

캐글의 "Credit Card Fraud Detection" 데이터셋에서 텍스트 데이터는 주로 설명 변수나 레이블로 사용되지 않습니다.
이미 숫자 형태의 변수들로 구성되어 있기 때문에 별도의 텍스트 데이터 처리가 필요하지 않아 생략하였습니다.

5. 특성 스케일링 및 변환

```python
# 특성 스케일링 및 변환은 표준화를 사용하는 것이 적합
from sklearn.preprocessing import StandardScaler

# StandardScaler 객체 생성
scaler = StandardScaler()

# 특성 데이터를 스케일링
scaled_features = scaler.fit_transform(data.drop('Class', axis=1))

# 스케일링된 데이터로 DataFrame 생성 (예시)
scaled_data = pd.DataFrame(data=scaled_features, columns=data.drop('Class', axis=1).columns)

# 스케일링된 데이터에 레이블 추가 (예시)
scaled_data['Class'] = data['Class']

# 로그 변환 적용 (표준화된 데이터에 적용하는 것을 예시로 보여줍니다)
log_scaled_data = np.log(1 + scaled_data.drop('Class', axis=1))

# 로그 변환 적용 후의 분포 시각화 예시 (한 개의 특성에 대해서만 시각화)
plt.figure(figsize=(8, 6))

# 로그 변환 전의 분포(예:V10)
plt.subplot(1, 2, 1)
plt.hist(scaled_data['V10'], bins=50, color='blue', alpha=0.7)
plt.title('Original Distribution')

# 로그 변환 후의 분포(예:V10)
plt.subplot(1, 2, 2)
plt.hist(log_scaled_data['V10'], bins=50, color='green', alpha=0.7)
plt.title('Log Transformed Distribution')

plt.show()
```

✓ 1.2s



6. 사용자 정의 변환기

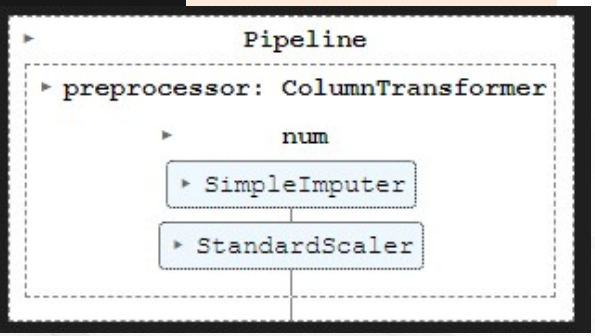
```python
# 사용자 정의 변환기 클래스 생성
from sklearn.base import BaseEstimator, TransformerMixin
import numpy as np

class LogTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, columns=None):
        self.columns = columns

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_transformed = X.copy()
        if self.columns is None:
            # 모든 열에 대해 로그 변환을 적용
            X_transformed = np.log1p(X_transformed)
        else:
            # 특정 열에 대해 로그 변환을 적용
            for col in self.columns:
                X_transformed[col] = np.log1p(X_transformed[col])
        return X_transformed

log_transformer = LogTransformer(columns=['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
                                          'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
                                          'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28'])
transformed_data = log_transformer.fit_transform(data)
```

✓ 0.3s

7. 변환 파이프라인

```python
# 변환 파이프라인
from sklearn import set_config
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# 파이프라인의 각 단계를 정의합니다.
numeric_features = data.columns.drop(['Class'])

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # 결측치 대체
    ('scaler', StandardScaler())  # 표준화
])

# ColumnTransformer 정의
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)
    ])

# 전체 파이프라인 정의
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# 파이프라인 시각화
set_config(display='diagram')
display(pipeline)
```

✓ 0.1s



8. 모델 선택과 훈련

캐글의 "Credit Card Fraud Detection" 데이터셋을 기반으로 모델은 Light GBM모델을 선택하였으며 데이터셋의 크기가 크고, 피처의 수가 많은 대규모 데이터셋으로 Light GBM은 대용량 데이터와 대규모의 피처를 처리하는 데 뛰어난 성능을 보이는 경사 부스팅 모델입니다. 이러한 대규모 데이터셋에서 빠른 학습과 예측이 가능합니다.

8. 모델 선택과 훈련

```
#학습데이터와 테스트데이터를 분리합니다
#학습데이터 80% 테스트데이터 20%로 설정합니다
#알고리즘별 학습결과를 비교하기 위해 random state를 0으로 설정합니다
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

✓ 0.3s

```
Number transactions X_train dataset:  (227845, 28)
Number transactions y_train dataset:  (227845, 1)
Number transactions X_test dataset:  (56962, 28)
Number transactions y_test dataset:  (56962, 1)
```

```
# 모델 성능평가 함수를 미리 만들어 놓기
# 이 함수는 confusion matrix(혼동 행렬)을 기반으로 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 스코어(F1-Score)를 계산하고 출력합니다.
def model_evaluation(label, predict):
    cf_matrix = confusion_matrix(label, predict)
    Accuracy = (cf_matrix[0][0] + cf_matrix[1][1]) / sum(sum(cf_matrix))
    Precision = cf_matrix[1][1] / (cf_matrix[1][1] + cf_matrix[0][1])
    Recall = cf_matrix[1][1] / (cf_matrix[1][1] + cf_matrix[1][0])
    F1_Score = (2 * Recall * Precision) / (Recall + Precision)
    print("Model_Evaluation with Label:1")
    print("Accuracy: ", Accuracy)
    print("Precision: ", Precision)
    print("Recall: ", Recall)
    print("F1-Score: ", F1_Score)
```

✓ 0.0s

8. 모델 선택과 훈련

Light GBM을 기본 모델로 설정

```
lgb_dtrain = lgb.Dataset(data = pd.DataFrame(X_train), label = pd.DataFrame(y_train)) # 학습 데이터를 LightGBM 모델에 맞게 변환
lgb_param = {'max_depth': 10, # 트리 깊이
             'learning_rate': 0.01, # Step Size
             'n_estimators': 50, # Number of trees, 트리 생성 개수
             'objective': 'binary'} # 파라미터 추가, Label must be in [0, num_class) -> num_class보다 1 커야한다.
lgb_model = lgb.train(params = lgb_param, train_set = lgb_dtrain) # 학습 진행
pred= np.repeat(0, len(y_test))
pred[lgb_model.predict(X_test) > 0.5] = 1
model_evaluation(y_test, pred)

#모델 평가 결과에 대한 설명:
#정확도 (Accuracy): 99.92%로 매우 높은 정확도를 보입니다. 그러나, 이 데이터는 신용카드 사기 탐지와 같은 불균형한 클래스 분포를 가지고 있기 때문에 정확도만으로 모델의 성능을 평가하는 것은 적절하지 않을 수 있습니다.
#정밀도 (Precision): 92.19%로 매우 높습니다. 정밀도는 모델이 사기 거래라고 예측한 경우 중에서 실제로 사기 거래인 비율을 나타냅니다. 높은 정밀도는 모델이 정상 거래를 사기 거래로 잘못 분류하는 경우가 적다는 것을 의미합니다.
#재현율 (Recall): 58.42%로 중간 정도입니다. 재현율은 실제 사기 거래 중에서 모델이 정확하게 감지한 비율을 나타냅니다. 중간 정도의 재현율은 실제 사기 거래를 놓치는 경우가 있다는 것을 의미합니다.
#F1 스코어 (F1-Score): 71.52%로 중간 정도입니다. F1 스코어는 정밀도와 재현율의 조화 평균으로, 두 지표가 균형을 이룰 때 높은 값을 갖습니다. 중간 정도의 F1 스코어는 모델의 정밀도와 재현율이 균형을 이루고 있다는 것을 나타냅니다.
#이 평가 지표 결과를 통해 모델은 사기 거래를 잘 감지할 수 있지만, 그 중에서도 일부 정상 거래를 사기 거래로 잘못 예측하는 경우가 있다는 것을 알 수 있습니다.
#클래스 불균형 문제에 대응하기 위해 리샘플링 기법이나 다양한 평가 지표를 사용하여 모델을 튜닝하고 평가해볼 필요가 있습니다.
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
c:\Users\MacBookPro\AppData\Local\Programs\Python\Python312\Lib\site-packages\lightgbm\engine.py:172: UserWarning: Found `n_estimators` in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of argument")
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Info] Number of positive: 391, number of negative: 227454
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.070697 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7140
[LightGBM] [Info] Number of data points in the train set: 227845, number of used features: 28
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001716 -> initscore=-6.365996
[LightGBM] [Info] Start training from score -6.365996
Model_Evaluation with Label:1
Accuracy:  0.9991748885221726
Precision:  0.921875
Recall:  0.5841584158415841
F1-Score:  0.7151515151515151
```

9. 모델 세부 튜닝

클래스 불균형 문제를 해결하기 위해 Oversampling을 수행

- 추가 성능향상이 가능한지 확인

```
# 기존 데이터 구조 재확인
print("X_train dataset: ", X_train.shape)
print("y_train dataset: ", y_train.shape)
print("X_test dataset: ", X_test.shape)
print("y_test dataset: ", y_test.shape)
```

✓ 0.0s

```
X_train dataset:  (227845, 28)
y_train dataset:  (227845, 1)
X_test dataset:  (56962, 28)
y_test dataset:  (56962, 1)
```

9. 모델 세부 튜닝

```
from imblearn.over_sampling import SMOTE

print("SMOTE 수행 이전 label '1' 개수: {}".format(sum(y_train == 1))) # y_train 중 레이블 값이 1인 데이터의 개수
print("SMOTE 수행 이전 label '0' 개수: {} \n".format(sum(y_train == 0))) # y_train 중 레이블 값이 0 인 데이터의 개수

sm = SMOTE(random_state=0, sampling_strategy=0.3) # SMOTE 알고리즘, 샘플링 비율 증가('1' 클래스의 데이터를 30배로 증가)

X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel()) # Over Sampling 진행

print("SMOTE 수행 결과 label '1' 개수: {}".format(sum(y_train_res==1)))
print("SMOTE 수행 결과 label '0' 개수': {}".format(sum(y_train_res==0)))
```

✓ 1.1s

```
SMOTE 수행 이전 label '1' 개수: [391]
SMOTE 수행 이전 label '0' 개수: [227454]

SMOTE 수행 결과 label '1' 개수: 68236
SMOTE 수행 결과 label '0' 개수': 227454
```

```
print("SMOTE 수행 이전 X_train: {}".format(X_train.shape)) # SMOTE 적용 이전 데이터 형태
print("SMOTE 수행 이전 y_train: {}".format(y_train.shape)) # SMOTE 적용 이전 데이터 형태
print('SMOTE 수행 결과 X_train: {}'.format(X_train_res.shape)) # SMOTE 적용 결과 확인
print('SMOTE 수행 결과 y_train: {}'.format(y_train_res.shape)) # # SMOTE 적용 결과 확인
```

✓ 0.0s

```
SMOTE 수행 이전 X_train: (227845, 28)
SMOTE 수행 이전 y_train: (227845, 1)
SMOTE 수행 결과 X_train: (295690, 28)
SMOTE 수행 결과 y_train: (295690,)
```

9. 모델 세부 튜닝

동일한 코드를 이용하여 LightGBM 다시 수행 후 성능향상 확인

- 위에서 사용한 코드 그대로 복사 -> 붙여넣기
- 입력값만 달라짐
- SMOTE 전.후 성능지표를 비교
  - 만약 성능이 향상된다면
  - 클래스 불균형이 Over Sampling으로 어느정도 극복된 것으로 판단.

```
lgb_dtrain2 = lgb.Dataset(data = pd.DataFrame(X_train_res), label = pd.DataFrame(y_train_res)) # 학습 데이터를 LightGBM 모델에 맞게 변환
lgb_param2 = {'max_depth': 10, # 트리 깊이
              'learning_rate': 0.01, # Step Size
              'n_estimators': 50, # Number of trees, 트리 생성 개수
              'objective': 'multiclass', # 목적 함수
              'num_class': len(set(pd.DataFrame(y_train_res))) + 1} # 파라미터 추가, Label must be in [0, num_class) -> num_class보다 1 커야한다.
lgb_model2 = lgb.train(params = lgb_param2, train_set = lgb_dtrain2) # 학습 진행
lgb_model2_predict = np.argmax(lgb_model2.predict(X_test), axis = 1) # 평가 데이터 예측, Softmax의 결과값 중 가장 큰 값의 Label로 예측
model_evaluation(y_test, lgb_model2_predict) # 모델 분류 평가 결과

#모델 평가 결과에 대한 설명:
#정확도 (Accuracy): 99.91%로 매우 높은 정확도를 보입니다. 그러나, 이 데이터는 신용카드 사기 탐지와 같은 불균형한 클래스 분포를 가지고 있기 때문에 정확도만으로 모델의 성능을 평가하는 것은 적절하지 않을 수 있습니다.
#정밀도 (Precision): 71.07%로 중간 정도입니다. 정밀도는 모델이 사기 거래라고 예측한 경우 중에서 실제로 사기 거래인 비율을 나타냅니다. 이 모델은 사기 거래로 예측한 경우 중에 약 71%만이 실제로 사기 거래입니다.
#재현율 (Recall): 85.15%로 높습니다. 재현율은 실제 사기 거래 중에서 모델이 정확하게 감지한 비율을 나타냅니다. 높은 재현율은 실제 사기 거래를 놓치는 경우가 적다는 것을 의미합니다.
#F1 스코어 (F1-Score): 77.48%로 높습니다. F1 스코어는 정밀도와 재현율의 조화 평균으로, 두 지표가 균형을 이룰 때 높은 값을 갖습니다. 높은 F1 스코어는 모델의 정밀도와 재현율이 균형을 이루고 있다는 것을 나타냅니다.
#이 평가 지표 결과를 통해 모델은 사기 거래를 잘 감지하고 있으며, 그 중에서도 대부분의 사기 거래를 식별할 수 있습니다. 하지만 여전히 약간의 과격한 예측이 있을 수 있으므로, 비즈니스 상황에 따라 재현율과 정밀도의 균형을 조절할 필요가 있을 것입니다.
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
c:\Users\MacBookPro\AppData\Local\Programs\Python\Python312\Lib\site-packages\lightgbm\engine.py:172: UserWarning: Found `n_estimators` in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of argument")
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.073871 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7140
[LightGBM] [Info] Number of data points in the train set: 295690, number of used features: 28
[LightGBM] [Info] Start training from score -0.262364
[LightGBM] [Info] Start training from score -1.466339
Model_Evaluation with Label:1
Accuracy:  0.9991222218320986
Precision:  0.7107438016528925
Recall:  0.8514851485148515
F1-Score:  0.7747747747747747
```

9. 모델 세부 튜닝

Over sampling이 통한다면...

그렇다면, Oversampling 크기를 증가시킨다면? (30% → 60%)

- 성능 변화를 직접 확인

```
print("SMOTE 수행 이전 label '1' 개수: {}".format(sum(y_train == 1))) # y_train 중 레이블 값이 1인 데이터의 개수
print("SMOTE 수행 이전 label '0' 개수: {} \n".format(sum(y_train == 0))) # y_train 중 레이블 값이 0 인 데이터의 개수

sm2 = SMOTE(random_state = 0, sampling_strategy=0.6) # SMOTE 알고리즘, 샘플링 비율 증가('1' 클래스의 데이터를 60배로 증가)
X_train_res2, y_train_res2 = sm2.fit_resample(X_train, y_train.ravel()) # Over Sampling 진행

print("SMOTE 수행 결과 label '1' 개수: {}".format(sum(y_train_res2==1)))
print("SMOTE 수행 결과 label '0' 개수: {}".format(sum(y_train_res2==0)))
```

```
SMOTE 수행 이전 label '1' 개수: [391]
SMOTE 수행 이전 label '0' 개수: [227454]

SMOTE 수행 결과 label '1' 개수: 136472
SMOTE 수행 결과 label '0' 개수: 227454
```

9. 모델 세부 튜닝

```
lgb_dtrain3 = lgb.Dataset(data = pd.DataFrame(X_train_res2), label = pd.DataFrame(y_train_res2)) # 학습 데이터를 LightGBM 모델에 맞게 변환
lgb_param3 = {'max_depth': 10, # 트리 깊이
              'learning_rate': 0.01, # Step Size
              'n_estimators': 50, # Number of trees, 트리 생성 개수
              'objective': 'multiclass', # 목적 함수
              'num_class': len(set(pd.DataFrame(y_train_res2))) + 1} # 파라미터 추가, Label must be in [0, num_class) -> num_class보다 1 커야한다.
lgb_model3 = lgb.train(params = lgb_param3, train_set = lgb_dtrain3) # 학습 진행
lgb_model3_predict = np.argmax(lgb_model3.predict(X_test), axis = 1) # 평가 데이터 예측, Softmax의 결과값 중 가장 큰 값의 Label로 예측
model_evaluation(y_test, lgb_model3_predict) # 모델 분류 평가 결과

#모델 평가 결과에 대한 설명:
#정확도 (Accuracy): 99.42%로 매우 높은 정확도를 보입니다. 그러나, 이 데이터는 신용카드 사기 탐지와 같은 불균형한 클래스 분포를 가지고 있기 때문에 정확도만으로 모델의 성능을 평가하는 것은 적절하지 않을 수 있습니다.
#정밀도 (Precision): 21.67%로 비교적 낮습니다. 정밀도는 모델이 사기 거래라고 예측한 경우 중에서 실제로 사기 거래인 비율을 나타냅니다. 이 모델은 사기 거래로 예측한 경우 중에 약 22%만이 실제로 사기 거래입니다.
#재현율 (Recall): 87.13%로 높습니다. 재현율은 실제 사기 거래 중에서 모델이 정확하게 감지한 비율을 나타냅니다. 높은 재현율은 실제 사기 거래를 놓치는 경우가 적다는 것을 의미합니다.
#F1 스코어 (F1-Score): 34.71%로 중간 정도입니다. F1 스코어는 정밀도와 재현율의 조화 평균으로, 두 지표가 균형을 이룰 때 높은 값을 갖습니다. 이 모델의 F1 스코어는 두 지표가 균형을 이루고 있음을 나타냅니다.
#이 평가 지표 결과를 통해 모델은 사기 거래를 상당히 잘 감지하고 있으며, 그 중에서도 대부분의 사기 거래를 식별할 수 있습니다. 그러나, 정밀도가 낮기 때문에 실제 사기가 아닌 거래를 사기로 잘못 분류하는 경우가 있을 수 있습니다.
#이 모델을 사용할 때에는 비즈니스 요구사항에 따라 재현율과 정밀도의 균형을 조절해야 할 것입니다.
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
c:\Users\MacBookPro\AppData\Local\Programs\Python\Python312\Lib\site-packages\lightgbm\engine.py:172: UserWarning: Found `n_estimators` in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of argument")
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.050941 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7140
[LightGBM] [Info] Number of data points in the train set: 363926, number of used features: 28
[LightGBM] [Info] Start training from score -0.470003
[LightGBM] [Info] Start training from score -0.980831
Model_Evaluation with Label:1
Accuracy:  0.9941891085284926
Precision:  0.21674876847290642
Recall:  0.8712871287128713
F1-Score:  0.34714003944773175
```

9. 모델 세부 튜닝

극단적인 경우 테스트 (100%)

- 1:1 비율로 Oversampling
- 수행 결과를 30%, 60% 경우의 성능지표와 비교

```
print("SMOTE 수행 이전 label '1' 개수: {}".format(sum(y_train == 1))) # y_train 중 레이블 값이 1인 데이터의 개수
print("SMOTE 수행 이전 label '0' 개수: {} \n".format(sum(y_train == 0))) # y_train 중 레이블 값이 0 인 데이터의 개수

sm3 = SMOTE(random_state = 0) # SMOTE 알고리즘, Default: 동등
X_train_res3, y_train_res3 = sm3.fit_resample(X_train, y_train.ravel()) # Over Sampling 진행

print("SMOTE 수행 결과 label '1' 개수: {}".format(sum(y_train_res3==1)))
print("SMOTE 수행 결과 label '0' 개수: {}".format(sum(y_train_res3==0)))
```

```
SMOTE 수행 이전 label '1' 개수: [391]
SMOTE 수행 이전 label '0' 개수: [227454]

SMOTE 수행 결과 label '1' 개수: 227454
SMOTE 수행 결과 label '0' 개수: 227454
```

9. 모델 세부 튜닝

```
lgb_dtrain4 = lgb.Dataset(data = pd.DataFrame(X_train_res3), label = pd.DataFrame(y_train_res3)) # 학습 데이터를 LightGBM 모델에 맞게 변환
lgb_param4 = {'max_depth': 10, # 트리 깊이
              'learning_rate': 0.01, # Step Size
              'n_estimators': 50, # Number of trees, 트리 생성 개수
              'objective': 'multiclass', # 목적 함수
              'num_class': len(set(pd.DataFrame(y_train_res3))) + 1} # 파라미터 추가, Label must be in [0, num_class) -> num_class보다 1 커야한다.
lgb_model4 = lgb.train(params = lgb_param4, train_set = lgb_dtrain4) # 학습 진행
lgb_model4_predict = np.argmax(lgb_model4.predict(X_test), axis = 1) # 평가 데이터 예측, Softmax의 결과값 중 가장 큰 값의 Label로 예측
model_evaluation(y_test, lgb_model4_predict) # 모델 분류 평가 결과

#모델 평가 결과에 대한 설명:
#정확도 (Accuracy): 98.34%로 높은 정확도를 보입니다. 그러나, 이 데이터는 신용카드 사기 탐지와 같은 불균형한 클래스 분포를 가지고 있기 때문에 정확도만으로 모델의 성능을 평가하는 것은 적절하지 않을 수 있습니다.
#정밀도 (Precision): 8.64%로 매우 낮습니다. 정밀도는 모델이 사기 거래라고 예측한 경우 중에서 실제로 사기 거래인 비율을 나타냅니다. 이 모델은 사기 거래로 예측한 경우 중에 약 8.64%만이 실제로 사기 거래입니다.
#재현율 (Recall): 87.13%로 높습니다. 재현율은 실제 사기 거래 중에서 모델이 정확하게 감지한 비율을 나타냅니다. 높은 재현율은 실제 사기 거래를 놓치는 경우가 적다는 것을 의미합니다.
#F1 스코어 (F1-Score): 15.73%로 매우 낮습니다. F1 스코어는 정밀도와 재현율의 조화 평균으로, 두 지표가 균형을 이룰 때 높은 값을 갖습니다. 이 모델의 F1 스코어가 낮다는 것은 정밀도와 재현율이 불균형하게 평가되고 있다는 것을 나타냅니다.
#이 평가 지표 결과를 통해 이 모델은 실제 사기 거래를 잘 감지하지만, 대부분의 정확도는 실제로 사기가 아닌 거래를 사기로 잘못 분류하고 있습니다. 이 모델을 실제 환경에서 사용할 때에는 정밀도와 재현율의 균형을 맞추는 것이 중요할 것입니다.
```

```
c:\Users\MacBookPro\AppData\Local\Programs\Python\Python312\Lib\site-packages\lightgbm\engine.py:172: UserWarning: Found `n_estimators` in params. Will use it instead of argument
  _log_warning(f"Found `{alias}` in params. Will use it instead of argument")
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.113427 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7140
[LightGBM] [Info] Number of data points in the train set: 454908, number of used features: 28
[LightGBM] [Info] Start training from score -0.693147
[LightGBM] [Info] Start training from score -0.693147
Model_Evaluation with Label:1
Accuracy:  0.9834451037533795
Precision:  0.08644400785854617
Recall:  0.8712871287128713
F1-Score:  0.1572832886505809
```

다양한 실험 결과 → 결론: 30%가 가장 적합