신용카드 사기 탐지 하기

2020 산업공학 데이터 분석 대회

Data-DAY

홍혜림 현재웅

00
목차

1. 데이터 설명

- 캐글 제공 : 2013년 9월 유럽 카드 소지자의 2일 동안의 신용카드 거래 데이터
- 설명 변수
  → 개인 정보 보호로 인해 PCA로 이미 변환된 변수(V1~V28)
  → 시간(Time)과 거래 금액(Amount)
- 목표 변수 : Class(0은 정상, 1은 사기), 이진 변수

변수명	변수 설명	특징
Time	각 거래와 첫 거래 사이의 경과된 시간 (초)	PCA 변환X 설명 변수
Amount	거래 금액	
Class	응답 변수로 0이 정상 거래, 1이 사기	목표 변수

출처 : 캐글

2. 데이터 분석 목표

→ 신용카드 거래 데이터를 바탕으로 거래가 정상적인 결제인지, 사기인지 분류하는 모델 수립

3. 분석 이슈

1) 데이터의 불균형 문제

- 데이터에서 사기의 비율은 단 0.172%로 매우 불균형
- 따라서 정확도(accuracy)보다 confusion matrix의 f1-score, precision, recall score 점수 고려

2) 잘못된 사기 탐지의 위험성

- 정상 거래를 사기 탐지로 인식하여 해당 카드 정지를 내릴 경우 사용자는 큰 불만을 가질 것.
- 이를 방지하기 위해 정상 거래를 사기로 예측하는 경우(FP)를 최소화 하는 것을 추구

<Fig1. Confusion matrix>

4. 데이터 탐색

- 행과 열 개수: [284807, 31]
- 모두 수치형 변수

```
In [148]: df.shape
Out[148]: (284807, 31)
```

```
df.info()

<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
```

5. 결측치 처리

- 결측치 없음

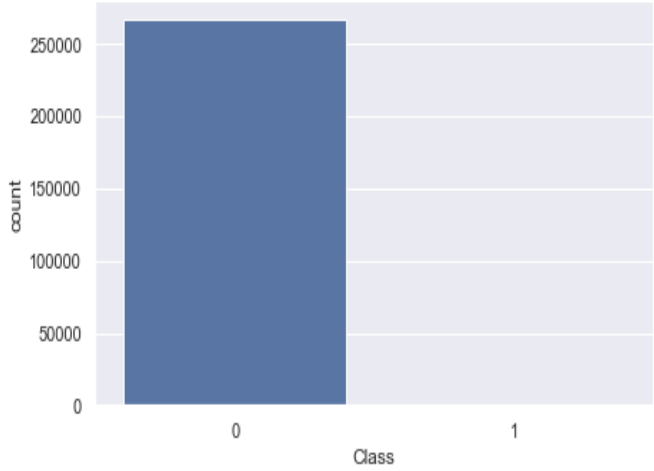
```
#결측치 확인
df.isna().sum().max()

0
```

```
#결측치 확인
df.isna().sum()

Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
```

6. 목표변수 확인

- Class : 0 (정상 거래), 1 (사기)
- 목표 변수 분포
- 정상 거래 → 266937건
- 사기 → 492건
- **매우 불균형함**

```
# 목표 변수의 분포 확인
df['Class'].value_counts()

0    266937
1       492
Name: Class, dtype: int64
```



7. 정규화

1) 정규화 전

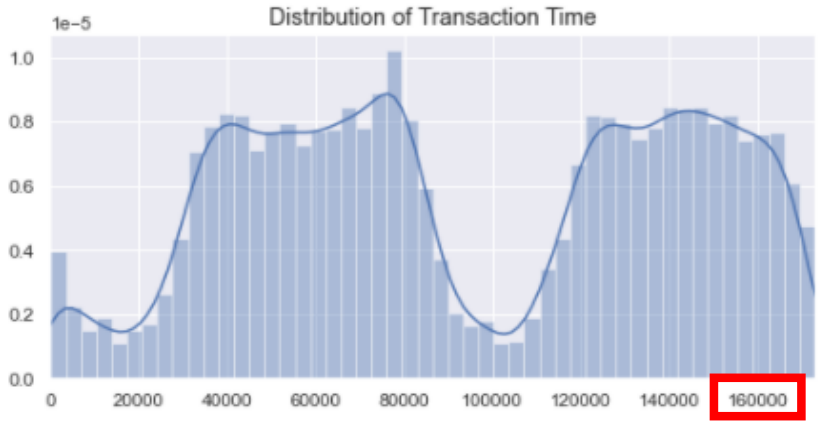
- PCA 변환되지 않은 변수인 Time와 Amount에 대한 정규화
- Time은 0~ 160000이 넘는 값까지 분포
- Amount는 0에 굉장히 밀집했고, 최대 25000 값까지 분포

2) 정규화 방법

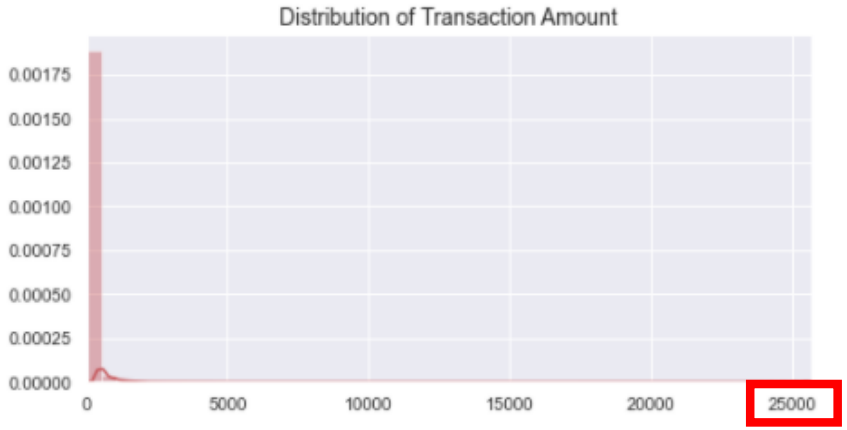
❖ StandardScaler: 기본 스케일, 평균과 표준편차 사용
❖ RobustScaler: 중앙값과 IQR 사용, 이상치의 영향을 최소화

→ 두 가지 방법을 사용하여 Time과 Amount를 정규화
→ 더 좋은 결과를 보인 것을 채택

<Time의 분포>



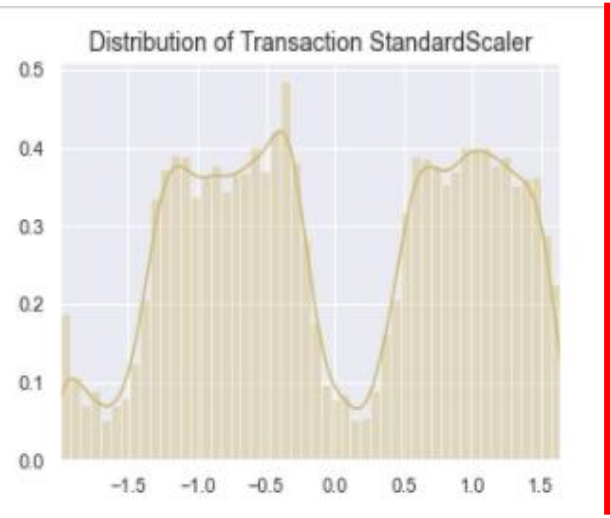
<Amount의 분포>



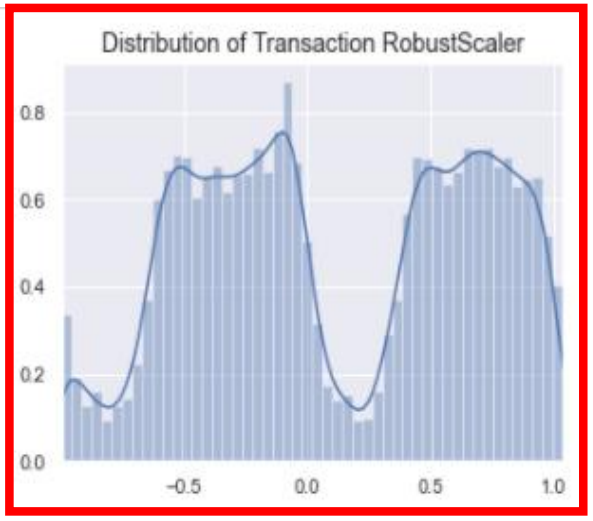
1) Time 정규화

→ 더 작은 분포 범위를 보이는 RobustScaler 채택

<StandardScaler 후 Time의 분포>



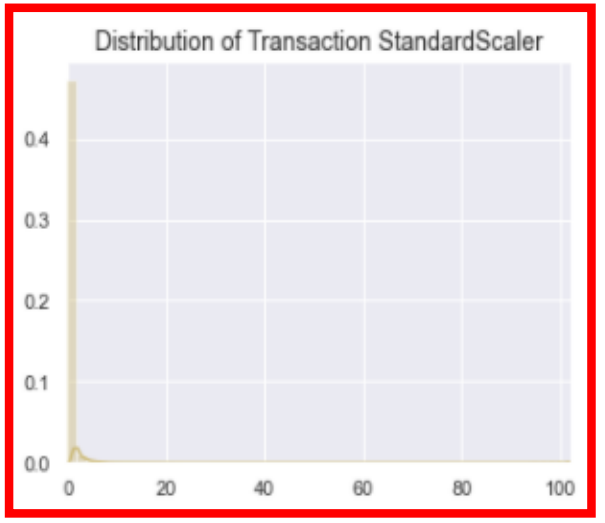
<RobustScaler 후 Time의 분포>



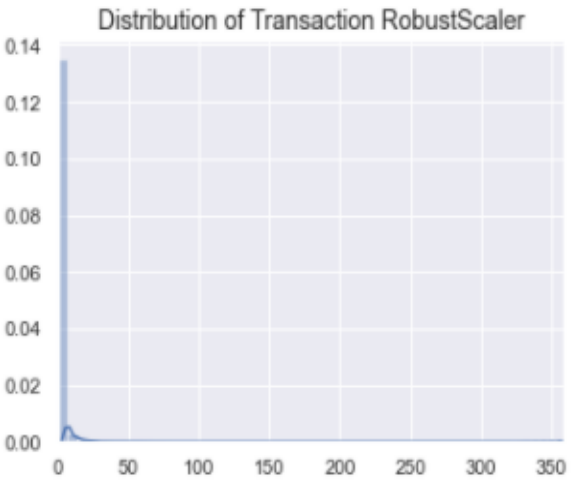
2) Amount 정규화

→ 더 작은 분포 범위를 보이는 StandardScaler 채택

<Standard Scaler 후 Amount의 분포>



<RobustScaler후 Amount의 분포>



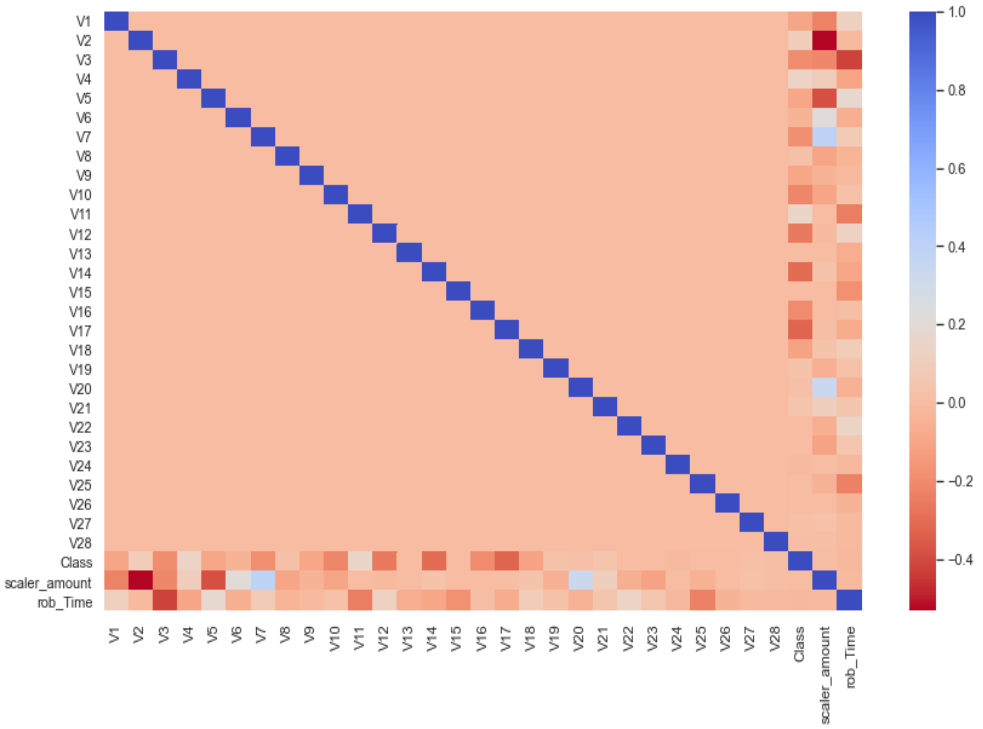
8. 이상치 처리

→ 모든 데이터에서 IQR을 통해 이상치 제거할 시 많은 데이터의 손실이 발생

✓ 해결 방법

- 목표 변수에 영향을 많이 미치는 변수를 상관 분석을 통해 찾고, 해당 변수에서만 이상치 제거
- Class(목표 변수)가 1(사기)인 데이터는 부족하기때문에 Class가 0(정상)인 데이터의 이상치 제거
- 상관 분석 결과
- 목표 변수와 연관성이 높은 변수는 V17, V14

<상관관계 그래프>

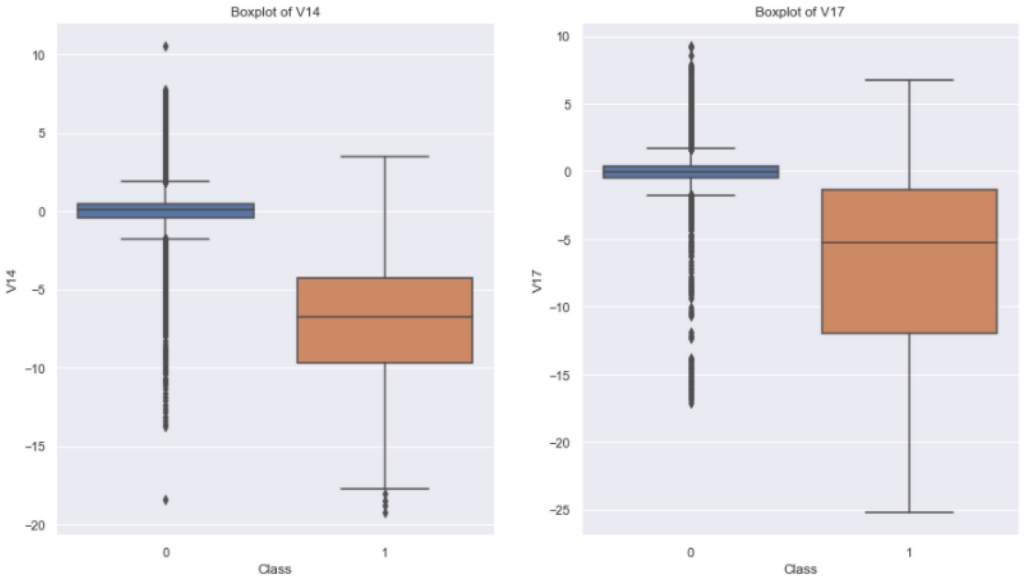


<상관관계 수치화>

	Class
V17	-0.326481
V14	-0.302544
V12	-0.260593
V10	-0.216883
V16	-0.196539

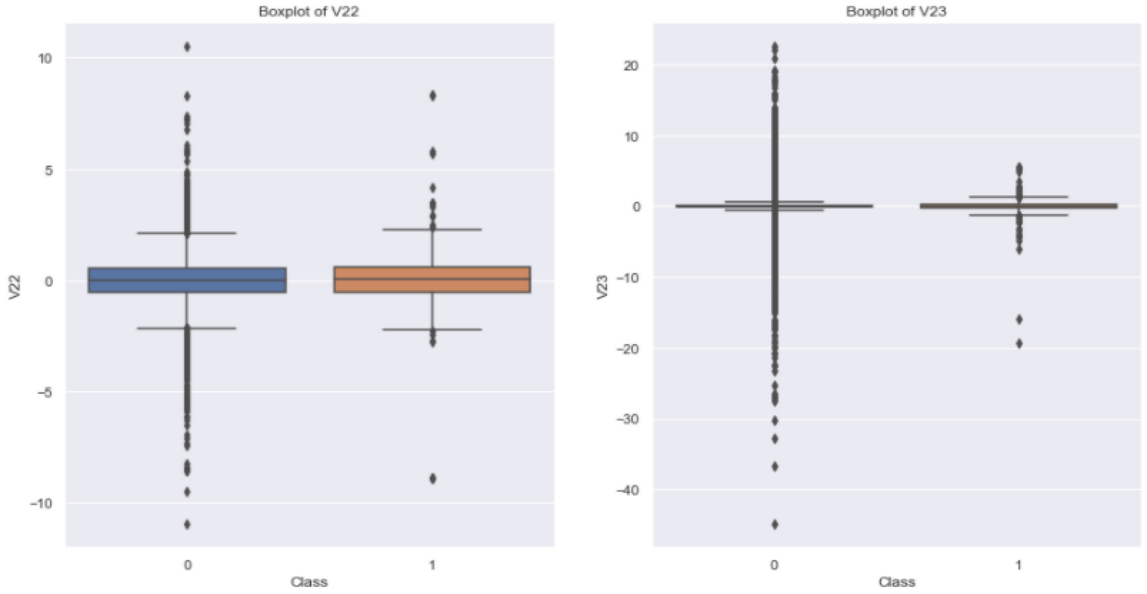
#3 EDA

<목표 변수와 연관성이 높은 V14, V17의 Boxplot>



0 class 1 0 class 1

<목표 변수와 연관성이 낮은 V22, V23의 Boxplot>



0 class 1 0 class 1

→ Boxplot의 비교를 통하여 V14와 V17은 목표 변수가 0, 1일 때 서로 분포가 확연히 다름을 확인

→ V14와 V17에 대하여 자세히 알 수는 없지만 정상 거래와 사기를 구분하는 중요한 Column임을 재차 확인

#3 EDA

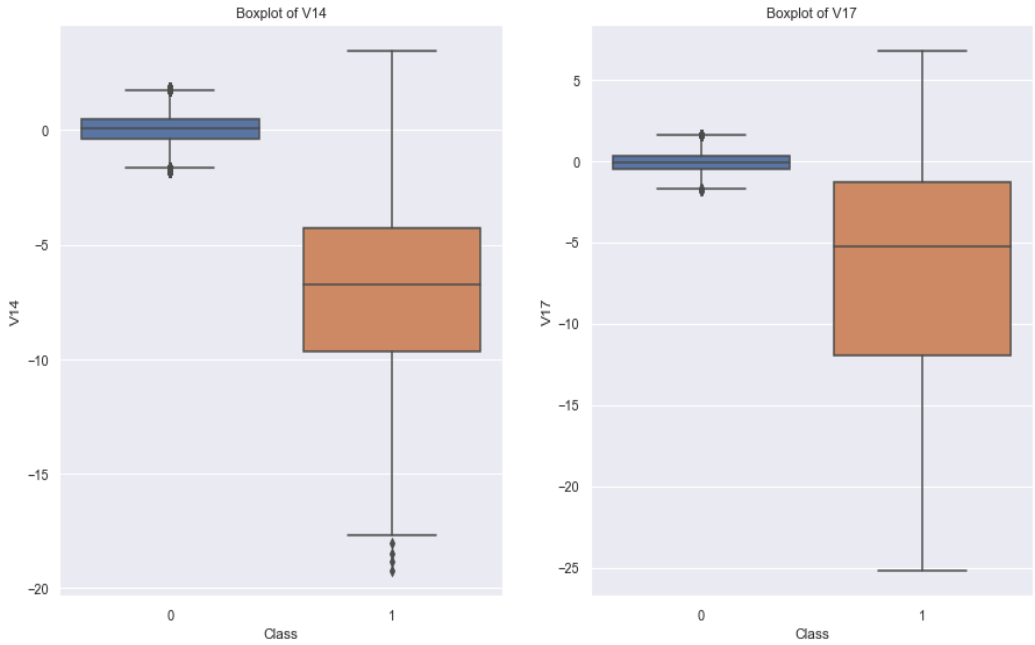
- V14, V17의 이상치 제거

```
#14, 17의 이상치 제거
def remove_outlier(d_cp, column):dusd
    fraud_column_data = d_cp[d_cp['Class']==0][column]
    quan_25 = np.percentile(fraud_column_data.values,25)
    quan_75 = np.percentile(fraud_column_data.values,75)

    iqr = quan_75-quan_25
    iqr = iqr*1.5
    lowest = quan_25 - iqr
    highest = quan_75 + iqr
    outlier_index = fraud_column_data[(fraud_column_data < lowest) | (fraud_column_data > highest)].index
    print(len(outlier_index))
    d_cp.drop(outlier_index, axis = 0, inplace = True)
    print(d_cp.shape)
    return d_cp
```

```
df = remove_outlier(df, 'V14')
df = remove_outlier(df, 'V17')
```

```
13800
(271007, 31)
3578
(267429, 31)
```

<이상치가 제거된 V14와 V17의 Boxplot>



→ 결과적으로 IQR을 통해 Class가 0 일 때의 V14, V17의 이상치 제거

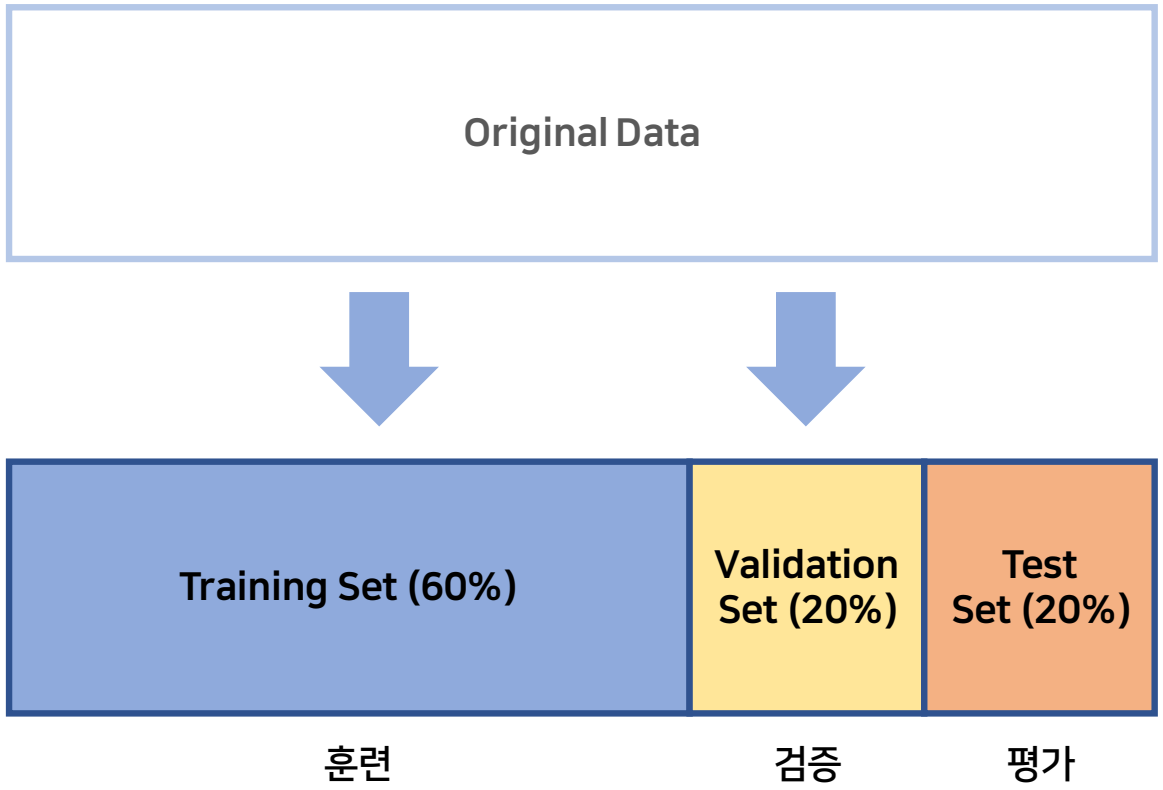
1. 과적합 방지

→ 데이터를 Training (60%), Validation (20%), Test (20%) 3개의 set으로 분할

1) Training set을 가지고 분류 모델을 훈련시킴

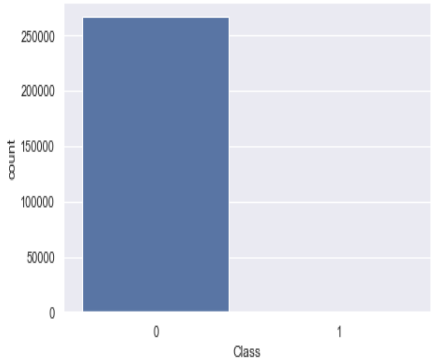
2) Validaion set을 통해 훈련중인 모델이 과적합 또는 과소적합 문제에 직면했는지 검증하며 모델링 구축

3) Test set을 사용하여 최종 모델에 대해 평가

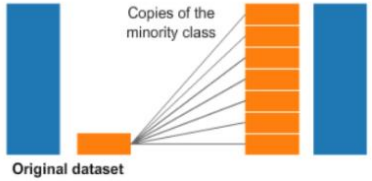


2. 모델링 방법

앞서 말했듯이 가장 큰 문제점은 목표 변수의 "데이터 불균형"



Under Sampling



Over Sampling

로지스틱 회귀
랜덤포레스트
XGBOOST
LGBM
신경망

로지스틱 회귀
랜덤포레스트
XGBOOST
LGBM
신경망

✓ 해결 방법

1) 언더 샘플링: 불균형한 데이터 셋에서 높은 비율을 차지하던 정상 거래 데이터를 줄임 → RandomUnderSampler 사용
2) 오버 샘플링: 불균형한 데이터 셋에서 낮은 비율을 차지하던 사기 데이터를 늘림 → SMOTE 사용

두 샘플링 방법을 통하여 데이터 불균형 문제를 해결하고, 이후 5개의 모델을 각각 사용하여 분류를 진행

#2 모델링 방법

3. 머신 러닝 분류 모델

```python
#모델링 함수
def modeling(model,X_train,X_val,y_train,y_val):
    model.fit(X_train,y_train)
    pred = model.predict(X_val)
    metrics(y_val,pred)
```

```python
#정확도, 정밀도, 재현율, f1-score, auroc 확인
def metrics(y_val,pred):
    accuracy = accuracy_score(y_val,pred)
    precision = precision_score(y_val,pred)
    recall = recall_score(y_val,pred)
    f1 = f1_score(y_val,pred)
    roc_score = roc_auc_score(y_val,pred,average='macro')
    print('정확도 : {0:.2f}, 정밀도 : {1:.2f}, 재현율 : {2:.2f}'.form
    print('f1-score : {0:.2f}, auc : {1:.3f}'.format(f1,roc_score)
```

→ 언더 샘플링과 오버 샘플링 동일하게 모델링 진행

```python
#로지스틱 회귀
lr = LogisticRegression(random_state =1,C=1000)
modeling(lr,X_train,X_test,y_train,y_test)
```

```
정확도 : 0.98, 정밀도 : 0.07, 재현율 : 0.94
f1-score : 0.12, auc : 0.957
```

```python
#랜덤포레스트
rfc = RandomForestClassifier(n_estimators=200,random_state=30, max_depth=8,n_jobs=-1)
modeling(rfc,X_train,X_test,y_train,y_test)
```

```
정확도 : 1.00, 정밀도 : 0.90, 재현율 : 0.91
f1-score : 0.90, auc : 0.953
```

```python
#xgb
xgb = XGBClassifier()
modeling(xgb,X_train,X_test,y_train,y_test)
```

```
정확도 : 1.00, 정밀도 : 0.86, 재현율 : 0.90
f1-score : 0.88, auc : 0.948
```

```python
#lgbm
lgb = LGBMClassifier(n_estimators=1000,num_leaves=64,n_jobs=-1,boost_from_average=False)
modeling(lgb,X_train,X_test,y_train,y_test)
```

```
정확도 : 1.00, 정밀도 : 0.97, 재현율 : 0.89
f1-score : 0.92, auc : 0.943
```

#2 모델링 방법

4. 신경망 모델

```
model = Sequential()
model.add(Dense(12, input_shape=(30, ), activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))
optimizer = Adam()
model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

```
history=model.fit(X_train,y_train,epochs=10,batch_size=5,validation_data=(X_val,y_val))
```

```
Epoch 1/10
68335/68335 [==============================] - 56s 815us/step - loss: 0.0351 - accuracy: 0.9870 - val_loss: 0.0127 - val_accuracy: 0.9962
Epoch 2/10
68335/68335 [==============================] - 54s 795us/step - loss: 0.0126 - accuracy: 0.9962 - val_loss: 0.0107 - val_accuracy: 0.9978
Epoch 3/10
68335/68335 [==============================] - 60s 877us/step - loss: 0.0102 - accuracy: 0.9970 - val_loss: 0.0064 - val_accuracy: 0.9981
Epoch 4/10
68335/68335 [==============================] - 58s 842us/step - loss: 0.0089 - accuracy: 0.9976 - val_loss: 0.0148 - val_accuracy: 0.9963
Epoch 5/10
68335/68335 [==============================] - 59s 870us/step - loss: 0.0081 - accuracy: 0.9978 - val_loss: 0.0057 - val_accuracy: 0.9984
Epoch 6/10
68335/68335 [==============================] - 54s 796us/step - loss: 0.0081 - accuracy: 0.9978 - val_loss: 0.0051 - val_accuracy: 0.9987
Epoch 7/10
68335/68335 [==============================] - 58s 852us/step - loss: 0.0076 - accuracy: 0.9979 - val_loss: 0.0054 - val_accuracy: 0.9986
Epoch 8/10
68335/68335 [==============================] - 62s 911us/step - loss: 0.0072 - accuracy: 0.9982 - val_loss: 0.0044 - val_accuracy: 0.9990
Epoch 9/10
68335/68335 [==============================] - 69s 1ms/step - loss: 0.0071 - accuracy: 0.9982 - val_loss: 0.0048 - val_accuracy: 0.9988
Epoch 10/10
68335/68335 [==============================] - 59s 862us/step - loss: 0.0065 - accuracy: 0.9983 - val_loss: 0.0065 - val_accuracy: 0.9987
```

✓ 다층 퍼셉트론 모델

- 은닉층 : 2개의 은닉층에 각각 12개의 노드 사용
- 과적합 방지 : dropout 사용
- 은닉층의 활성화 함수 : relu
- 출력층의 활성화 함수 : sigmoid
- Optimizer : Adam
- Loss : 이진 분류(binary_ctrossentropy)
- epoch : 10

5. 언더 샘플링 모델링 결과

- Test set로 평가한 모델의 confusion matrix

로지스틱 회귀 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	100	1600
	정상	4	52000

랜덤포레스트 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	99	420
	정상	6	53000

LGBM Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	100	750
	정상	5	53000

XGBOOST Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	100	910
	정상	4	53000

신경망 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	101	1084
	정상	4	52297

5. 언더 샘플링 모델링 결과

- confusion matrix의 결과 수치화

모델	정확도	정밀도	재현율	F1-score	Auc
로지스틱 회귀	0.97	0.06	0.96	0.11	0.966
랜덤 포레스트	0.99	0.19	0.94	0.32	0.967
XGBOOST	0.98	0.10	0.96	0.18	0.972
LGBM	0.99	0.12	0.95	0.21	0.969
신경망	0.98	0.09	0.96	0.16	0.970

→ 언더 샘플링에서는 랜덤포레스트 모델이 F1-score점수가 가장 높음
→ 또한, 정상을 사기로 판단할 오류도 가장 적음

6. 오버 샘플링 모델링 결과

- Test set로 평가한 모델 confusion matrix

로지스틱 회귀 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	90	1200
	정상	6	52000

랜덤포레스트 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	87	17
	정상	9	53000

LGBM Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	85	5
	정상	11	53000

XGBOOST Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	86	10
	정상	10	53000

신경망 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	86	102
	정상	10	53288

6. 오버 샘플링 모델링 결과

- confusion matrix의 결과 수치화

모델	정확도	정밀도	재현율	F1-score	Auc
로지스틱 회귀	0.98	0.07	0.94	0.12	0.957
랜덤 포레스트	1.00	0.81	0.90	0.85	0.948
XGBOOST	1.00	0.83	0.90	0.86	0.948
LGBM	1.00	0.93	0.89	0.91	0.943
신경망	1.00	0.46	0.89	0.60	0.946

→ 오버 샘플링에서는 LGBM 모델이 F1-score 점수가 가장 높음
→ 또한, 정상을 사기로 판단할 오류도 가장 적음

#1 최종 선정 모델

1. 최종 선정 모델

샘플링	모델	정확도	정밀도	재현율	F1-score	Auc
언더 샘플링	랜덤 포레스트	0.99	0.19	0.94	0.32	0.967
오버 샘플링	LGBM	1.00	0.93	0.89	0.91	0.943

✓ 최종 모델링 결과

- 오버 샘플링한 LGBM 모델이 언더 샘플링한 랜덤포레스트 보다
- F1-score 점수가 높으며, 정상 거래를 사기로 탐지할 오류가 적음

"" 최종 모델로 오버 샘플링한 LGBM 모델 선정 ""

<confusion matrix>

랜덤포레스트 Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	99	420
	정상	6	53000

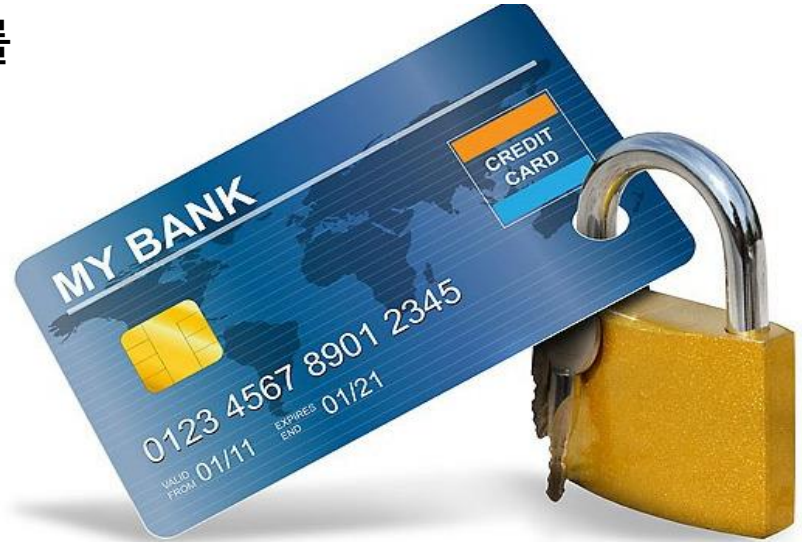
LGBM Confusion Matrix		실제 정답	
		사기	정상
분류 결과	사기	85	5
	정상	11	53000

2. 결론

- 불균형 데이터 셋에는 언더 샘플링보다 오버 샘플링 방법(SMOTE)이 더 효과적이고 분류 모델의 성능이 좋음

- 최종 모델의 LGBM은 정상을 사기로 판단할 오류면에서도 높은 성과를 보이고 있음

- 금융기관에서 우리 모델을 사용해 신용카드의 정상 거래와 사기를 91%의 성능으로 탐지할 수 있음



감사합니다 :)