ML

2023122054
오상호

데이터 로드 및 기본 탐색

```
# 데이터 로드 및 기본 탐색
df = pd.read_csv('creditcard.csv')
df['Class'].value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

다음 코드를 통해 data set에 정상 거래와 사기 거래의 비율이 어느정도인지 확인하였다. 정상 거래의 경우 284315, 사기거래의 경우 492개가 차지하고 있었고, 이를 통해 정상 거래에 비해 사기 거래의 케이스가 매우 적은 불균형한 경우라고 파악할 수 있다.

샘플링

```
#샘플링
df_true_raw = df[df['Class'] == 0]
df_true = df_true_raw.sample(n = 10000, random_state=42)
df_scam = df[df['Class'] == 1]

df_new = pd.concat([df_true, df_scam])
df_new['Class'].value_counts()
```

```
Class
0    10000
1      492
Name: count, dtype: int64
```

다음 코드를 통해 사기거래의 경우는 전부 유지시키고 정상 거래에서 10000건을 무작위로 샘플링하여, 새로운 데이터 프레임 df_new를 구성하였다. 해당 프레임에서 케이스의 비율을 확인한 결과 설계한대로 정상 거래의 경우가 10000, 사기 거래의 경우가 492개로 도출되었다.

데이터 전처리

```
#데이터 전처리
scaler = StandardScaler()
df_new['Amount'] = scaler.fit_transform(df_new[['Amount']])
df_new.rename(columns={'Amount': 'Amount_Scaled'}, inplace=True)
X = df_new.drop(['Time', 'Class'], axis=1)
y = df_new['Class']
```

StandardScaler를 통해서 Amount 변수를 표준화하였고, 이를 Amount_Scaled로 대체하였다.
X의 경우 Time과 Class를 제외한 것으로 입력변수 설정을 하였고,
Y의 경우 Class로 정답라벨 설정을 하였다.

학습 데이터와 테스트 데이터 분할

:

```
#학습 데이터와 테스트 데이터 분할
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=42,
    stratify=y
)

print("전체 데이터 Class 비율:")
print(y.value_counts(normalize=True))

print("\n학습 데이터 Class 비율:")
print(y_train.value_counts(normalize=True))

print("\n테스트 데이터 Class 비율:")
print(y_test.value_counts(normalize=True))
```

```
전체 데이터 Class 비율:
Class
0    0.953107
1    0.046893
Name: proportion, dtype: float64

학습 데이터 Class 비율:
Class
0    0.953056
1    0.046944
Name: proportion, dtype: float64

테스트 데이터 Class 비율:
Class
0    0.953311
1    0.046689
Name: proportion, dtype: float64
```

train_test_split을 사용해 학습셋:테스트셋 비율을 8:2로 나누고,stratify=y 옵션으로 클래스 비율 유지, 분할된 데이터의 Class 비율을 출력하였다.

SMOTE 적용

```
#SMOTE 적용
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("원본 학습 데이터:")
print(y_train.value_counts())

print("\nSMOTE 적용 후:")
print(pd.Series(y_train_resampled).value_counts())
```

```
원본 학습 데이터:
Class
0    7999
1     394
Name: count, dtype: int64

SMOTE 적용 후:
Class
0    7999
1    7999
Name: count, dtype: int64
```

사기 거래의 경우 정상 거래에 비해서 소수 클래스이므로, 불균형 상태이다.
실제로 원본 학습 데이터에서는 정상 거래의 경우는 7999건 이었지만 사기 거래의 경우는 394로 불균형이 심한 상태였다. 이러한 경우 사기 탐지 성능에 문제가 발생할 수 있기 때문에 SMOTE를 사용하여 소수 클래스인 사기거래에 대해서 오버 샘플링을 시행하였다.
SMOTE 시행 후 정상 거래와 사기 거래의 경우 같은 데이터 수를 가지게 되었다.

모델 학습

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, average_precision_score

model = RandomForestClassifier(random_state=42, n_estimators=100)
model.fit(X_train_resampled, y_train_resampled)

y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)

# Classification Report
print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred, target_names=['정상', '사기']))


# PR-AUC 계산
pr_auc = average_precision_score(y_test, y_pred_proba[:, 1])
print(f"PR-AUC Score: {pr_auc:.4f}")
```

```
=== Classification Report ===
              precision    recall  f1-score   support

          정상       0.99      1.00      1.00      2001
          사기       0.99      0.88      0.93        98

    accuracy                           0.99      2099
   macro avg       0.99      0.94      0.96      2099
weighted avg       0.99      0.99      0.99      2099

PR-AUC Score: 0.9493
```

RandomForest 모델을 사용하여서 모델학습을 진행하였다.
n_estimator를 100으로 설정하였고, 나머지는 기본 설정을 이용하여서 학습시켰고, 그 결과 precision, Recall, F1-score, PR-AUC에서 모두 과제 명세의 기준을 달성할 수 있었다.

최종 성능 평가

```python
#개선
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, average_precision_score

model = RandomForestClassifier(random_state=42, n_estimators=200, max_depth=20, min_samples_split = 10, min_samples_leaf=2)
model.fit(X_train_resampled, y_train_resampled)

y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)

# Classification Report
print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred, target_names=['정상', '사기']))


# PR-AUC 계산
pr_auc = average_precision_score(y_test, y_pred_proba[:, 1])
print(f"PR-AUC Score: {pr_auc:.4f}")
```

마찬가지로 RandomForest모델을 사용하였고,
이번에는 하이퍼파라미터들을 조정해보았다.
n_estimator를 200으로 증가시키고, smote를 사용했으므로 과적합을 방지하기 위해 Max_depth, min_samples_split, min_samples_leaf의 값을 기본값과 다르게 설정해주었다.
그 결과 precision, Recall, F1-score의 경우 이전과 큰 성능 저하없이 거의 동일하게 도출되었고, PR-AUC에서 성능 향상을 이룰 수 있었다.