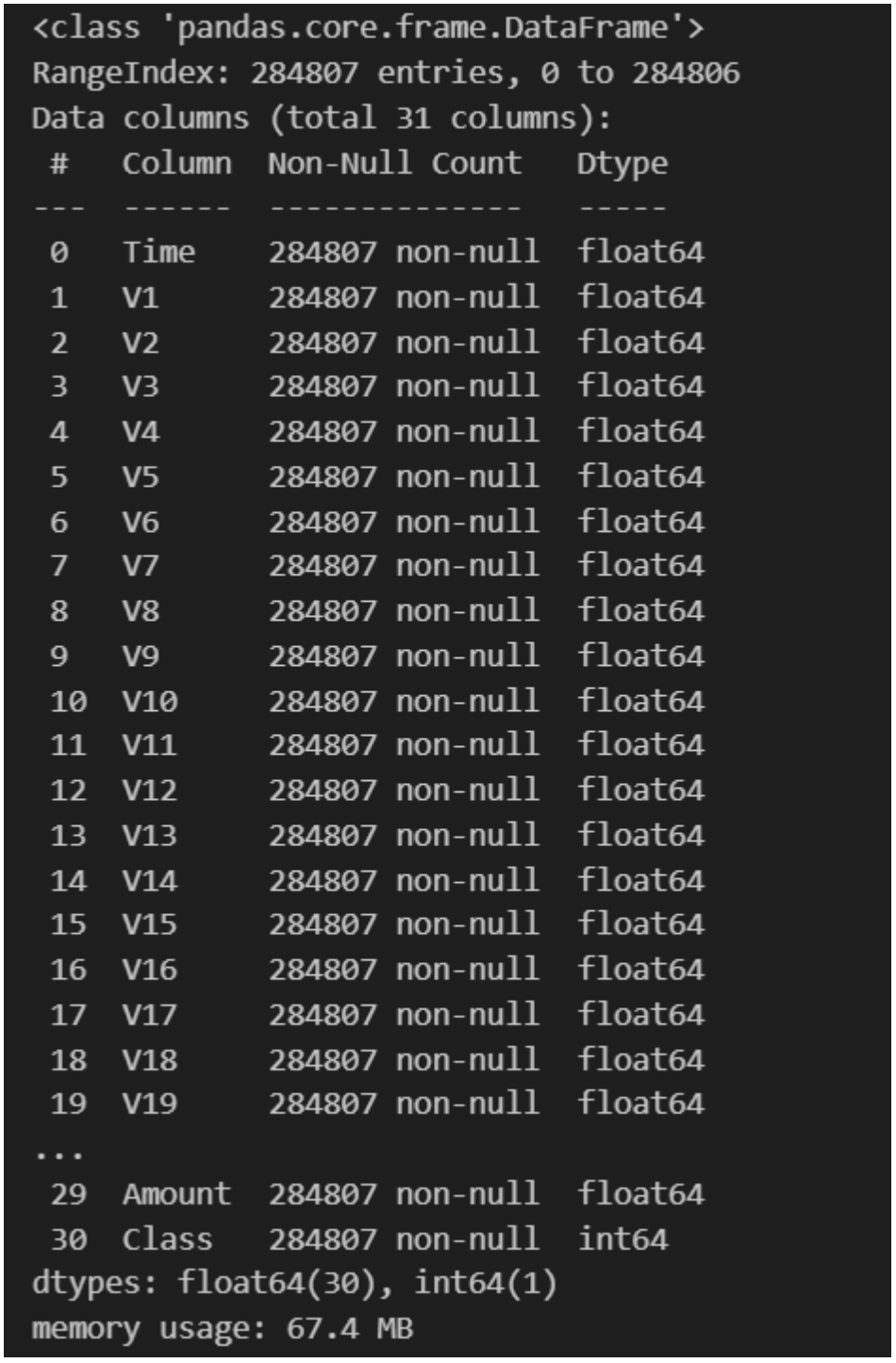
[5회차] ML 과제

Created	@2026년 1월 25일 오후 2:28
Tags	

1. 데이터 로드 및 기본 탐색

```
df = pd.read_csv("creditcard.csv")
# 데이터 구조 확인
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
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
...
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
# 정상 거래(Class: 0), 사기 거래 건수(Class: 1) 확인
print(df['Class'].value_counts())
```

Class
0 (정상 거래 건수) 2843
1 (사기 거래 건수)   492
Name: count, dtype: int64

2. 샘플링

```
normal_sample = df[df['Class'] == 0].sample(n=10000, random_state=42)
fraud = df[df['Class'] == 1]
new_df = pd.concat([normal_sample, fraud], axis=0)

print(new_df['Class'].value_counts(normalize=True))
```

- 샘플링 후 Class 비율

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

3. 데이터 전처리

```
# Amount 표준화
scaler = StandardScaler()
new_df["Amount_Scaled"] = scaler.fit_transform(new_df[["Amount"]])

# Amount 원본 변수 제거
new_df = new_df.drop(columns=['Amount'])

# X, y로 데이터프레임 분리
X = new_df.drop(columns=['Class'])
y = new_df['Class']
```

x

0.1s

Python

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	
138028	82450.0	1.314539	0.590643	-0.666593	0.716564	0.301978	-1.125467	0.388881	-0.288390	-0.132137	...	-0.058040	-0.170307	-0.429655	-0.141341	-0.200195	0.639491	0.399476	-0
63099	50554.0	-0.798672	1.185093	0.904547	0.694584	0.219041	-0.319295	0.495236	0.139269	-0.760214	...	-0.081298	0.202287	0.578699	-0.092245	0.013723	-0.246466	-0.380057	-0
73411	55125.0	-0.391128	-0.245540	1.122074	-1.308725	-0.639891	0.008678	-0.701304	-0.027315	-2.628854	...	0.065716	-0.133485	0.117403	-0.191748	-0.488642	-0.309774	0.008100	0
164247	116572.0	-0.060302	1.065093	-0.987421	-0.029567	0.176376	-1.348539	0.775644	0.134843	-0.149734	...	-0.169706	0.355576	0.907570	-0.018454	-0.126269	-0.339923	-0.150285	-0
148999	90434.0	1.848433	0.373364	0.269272	3.866438	0.088062	0.970447	-0.721945	0.235983	0.683491	...	-0.282777	0.103563	0.620954	0.197077	0.692392	-0.206530	-0.021328	-0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	1.252967	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.226138	0.370612	0.028234	-0.145640	-0.081049	0.521875	0.739467	0
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.247968	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.306271	0.583276	-0.269209	-0.456108	-0.183659	-0.328168	0.606116	0
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.017652	-0.164350	-0.295135	-0.072173	-0.450261	0.313267	-0.289617	0

10492 rows × 30 columns

y

0.0s

```
138028    0
63099     0
73411     0
164247    0
148999    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 10492, dtype: int64
```

4. 학습 데이터와 테스트 데이터 분할

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Train Class Distribution")
print(y_train.value_counts(normalize=True))

print("\nTest Class Distribution")
print(y_test.value_counts(normalize=True))
```

Train Class Distribution
Class
0 0.953056
1 0.046944
Name: proportion, dtype: float64

Test Class Distribution
Class

0 0.953311
1 0.046689
Name: proportion, dtype: float64

5. SMOTE 적용

```
# SMOTE 적용 전 사기 거래 건수
print("Before SMOTE")
print("Fraud count (Class=1):", (y_train == 1).sum())

# SMOTE 적용
smote = SMOTE(random_state=42)
new_X_train, new_y_train = smote.fit_resample(X_train, y_train)

# SMOTE 적용 후 사기 거래 건수
print("\nAfter SMOTE")
print("Fraud count (Class=1):", (new_y_train == 1).sum())
```

Before SMOTE
Fraud count (Class=1): 394

After SMOTE
Fraud count (Class=1): 7999

Credit Card Fraud Detection 데이터는 정상 거래에 비해 사기 거래의 비율이 매우 낮은 심각한 클래스 불균형 문제를 가지고 있다.

SMOTE 적용 전 학습 데이터에서 사기 거래는 394건에 불과하며 모델이 사기 거래의 패턴을 충분히 학습하기 어렵다.

이와 같은 불균형 데이터에서 모델을 그대로 학습한다면, 모델은 다수 클래스인 정상 거래 위주로 학습하게 되어 사기 거래를 정상 거래로 잘못 분류하는 경우가 증가하게 된다. 이는 사기 탐지 문제에서 치명적인 오류이며 Recall 저하로 이어진다.

SMOTE는 기존 소수 클래스 샘플 간의 특성 공간을 활용하여 새로운 합성 데이터를 생성함으로써 소수 클래스의 분포를 확장하는 오버샘플링 기법이다. 이를 통해 모델은 사기 거래의 다양한 패턴을 학습할 수 있다.

SMOTE 적용 후 사기 거래는 394건에서 7999건으로 증가하였으며 이를 통해 학습 데이터의 클래스 불균형이 완화되었다. 이를 통해 모델이 사기 거래를 적극적으로 탐지할 수 있으며 이후 Recall 및 F1-score 개선에도 기여할 수 있다.

6. 모델 학습

```
model = LogisticRegression(max_iter=5000, random_state=42)

# 모델 학습
model.fit(new_X_train, new_y_train)

# 테스트셋 예측값과 예측 확률
y_predict = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

print("Predict: ", y_predict)
print("Predict probability: ", y_proba)

print("\nClassification Report")
print(classification_report(y_test, y_predict))

pr_auc = average_precision_score(y_test, y_proba)
print("\nPR-AUC: ", pr_auc)
```

Predict: [0 0 0 ... 0 0 0]
Predict probability: [1.37682533e-01 7.48687431e-02 4.04165625e-02 ...
3.52845933e-04
5.23227636e-04 4.63449062e-07]

Classification Report

```
              precision  recall  f1-score  support
    0      1.00     0.99     0.99     2001
    1      0.81     0.93     0.87       98

accuracy                    0.99     2099
```

PR-AUC: 0.954877187327478

7. 최종 성능 평가

```
# Train을 학습/검증으로 한 번 더 나누기 (threshold 고르기)
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.2, rando
m_state=42, stratify=y_train)

# SMOTE는 학습 데이터에만 적용
smote = SMOTE(random_state=42)
X_tr_sm, y_tr_sm = smote.fit_resample(X_tr, y_tr)

# model, hyperparameter 튜닝
base_model = LogisticRegression(solver="liblinear", max_iter=5000, rando
m_state=42)
param_grid = {"C": [0.01, 0.1, 1, 10, 100], "penalty": ["l2"]}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(estimator=base_model, param_grid=param_grid, scor
ing="average_precision", cv=cv, n_jobs=-1, error_score="raise")
grid.fit(X_tr_sm, y_tr_sm)

best_model = grid.best_estimator_

print("Best parameters: ", grid.best_params_)
print("Best CV PR-AUC: ", grid.best_score_)
```

Best parameters: {'C': 0.01, 'penalty': 'l2'}
Best CV PR-AUC: 0.988370160551408

```
# Threshold 조정
# 검증 확률
val_proba = best_model.predict_proba(X_val)[:, 1]

thresholds = np.linspace(0.0, 1.0, 1001)
```

```
best_threshold = None
best_f1 = -1

for threshold in thresholds:
    val_predict = (val_proba >= threshold).astype(int)
    r1 = recall_score(y_val, val_predict, pos_label=1)
    f1 = f1_score(y_val, val_predict, pos_label=1)

    # 목표 조건을 만족하는 threshold 중에서 F1이 최대인 걸 선택
    if (r1 >= 0.80) and (f1 >= 0.88):
        if f1 > best_f1:
            best_f1 = f1
            best_threshold = threshold

print("Chosen threshold:", best_threshold)
print("Validation F1:", best_f1)
```

Chosen threshold: 0.886
Validation F1: 0.896551724137931

```
# 최종 테스트 평가
test_proba = best_model.predict_proba(X_test)[:, 1]

y_test_pred = (test_proba >= best_threshold).astype(int)

print("Final threshold: ", best_threshold)
print("\nClassification Report(Class 0, Class 1)")
print(classification_report(y_test, y_test_pred, digits=4))

pr_auc = average_precision_score(y_test, test_proba)
print("PR-AUC: ", pr_auc)
```

Final threshold: 0.886

Classification Report(Class 0, Class 1)

```
              precision recall  f1-score   support

           0     0.9916    0.9990    0.9953      2001
           1     0.9759    0.8265    0.8950        98

    accuracy                         0.9909      2099
```

PR-AUC: 0.9061781365968122

Logistic Regression 모델을 선정한 후 GridSearchCV를 통해 하이퍼파라미터 튜닝을 수행하고 PR-AUC를 기준으로 최적의 모델을 선정하였다. 이후 검증 데이터에서 Recall과 F1-score를 동시에 만족하는 threshold를 탐색하여 최종 threshold를 0.886으로 설정하였다.

테스트 데이터 평가 결과, 사기 거래에 대해 Recall=0.8265, F1-score=0.8950을 달성하였으며 PR-AUC=0.9061로 목표 기준을 모두 충족하였다. 또한 정상 거래에 대해서도 Recall=0.9990, F1-score=0.9953을 만족하였다. 이를 통해 본 모델은 클래스 불균형 환경에서도 효과적인 사기 거래 탐지 성능을 달성하였다.