클래스 불균형 처리: Undersampling, Oversampling (SMOTE), Class Weight

- 분류(classification) 문제에서 각 클래스(label)에 속하는 데이터의 양에 심각한 차이가 있는 상황
- 신용카드 사기 탐지 데이터에서 99%가 정상 거래이고 1%만이 사기 거래인 경우
- 모델은 예측 정확도(accuracy)를 높이기 위해 모든 데이터를 다수 클래스(정상 거래)로 예측하려는 경향을 보임
- 결과적으로 소수 클래스(사기 거래)에 대한 재현율(recall)이 매우 낮아지는 문제가 발생
- **언더샘플링 (Undersampling)**
  - 다수 클래스의 데이터 개수를 줄여 소수 클래스의 데이터 개수와 맞추는 방법
  - 가장 간단한 방법 : 다수 클래스의 데이터를 무작위로 제거하는 **Random Undersampling**
- **오버샘플링 (Oversampling)**
  - 소수 클래스의 데이터 개수를 늘려 다수 클래스의 데이터 개수와 맞추는 방법
  - **Random Oversampling**: 소수 클래스의 데이터를 무작위로 복제하여 추가
  - **SMOTE (Synthetic Minority Over-sampling Technique)**
    - 소수 클래스의 데이터 포인트를 선택하고, 그 점과 가장 가까운 이웃들 사이의 공간에 새로운 합성(synthetic) 데이터를 생성하는 가장 대표적인 오버샘플링 기법
    - 단순 복제가 아니기 때문에 오버피팅 문제 완화 가능
- **Class Weight (가중치 조절)**
  - 모델 학습 시 각 클래스의 중요도에 다른 가중치를 부여하는 방법
  - 소수 클래스에 더 높은 가중치를 부여하여, 해당 클래스의 데이터를 잘못 분류했을 때 모델에 더 큰 페널티(손실)를 줌
  - 모델이 소수 클래스에 더 집중하도록 유도

적용 가능한 상황

- **사기 탐지, 불량품 검출, 질병 진단** 등 소수 클래스를 탐지하는 것이 매우 중요한 모든 불균형 데이터셋에 적용됩니다.
- **언더샘플링**: 전체 데이터의 양이 매우 많아서, 다수 클래스의 데이터를 일부 잃어도 정보 손실이 크지 않을 때 사용을 고려할 수 있습니다. 계산 비용을 줄이는 효과가 있습니다.
- **오버샘플링 (특히 SMOTE)**: 전체 데이터의 양이 충분하지 않아 정보 손실을 피하고 싶을 때 가장 일반적으로 사용됩니다. 언더샘플링보다 정보 손실이 없다는 큰 장점이 있습니다.
- **Class Weight**: 데이터를 직접 샘플링하지 않고 모델 자체에서 불균형을 처리하고 싶을 때 사용합니다. `scikit-learn`의 많은 분류 모델(`LogisticRegression`, `SVC`, `RandomForestClassifier` 등)이 `class_weight` 파라미터를 지원하여 간편하게 적용할 수 있습니다.

예제 데이터 생성 및 라이브러리 설치

- SMOTE를 사용하기 위해서는 `imbalanced-learn` 라이브러리가 필요
- `pip install imbalanced-learn`

```
from collections import Counter
from sklearn.datasets import make_classification
import pandas as pd

# 불균형 데이터셋 생성
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,
                           n_redundant=0, n_classes=2, n_clusters_per_class=1,
                           weights=[0.99, 0.01], flip_y=0, random_state=42)

print(f"Original dataset shape %s" % Counter(y))
# Original dataset shape Counter({0: 990, 1: 10})
```

1. Random Undersampling

- **용도**: 다수 클래스의 데이터를 무작위로 제거하여 클래스 균형을 맞춥니다.
- **주의사항**: 중요한 정보를 담고 있는 데이터가 제거될 수 있어 모델 성능이 저하될 위험이 있습니다.
- **코드 예시**

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42)
X_resampled_under, y_resampled_under = rus.fit_resample(X, y)

print(f"Resampled dataset shape %s" % Counter(y_resampled_under))
# Resampled dataset shape Counter({0: 10, 1: 10})
```

- **결과 해석**
  - 다수 클래스(0)의 데이터 990개가 10개로 줄어들어, 소수 클래스(1)와 동일한 10개의 샘플을 갖게 되었습니다.

2. SMOTE (Oversampling)

- **용도**: 소수 클래스의 데이터를 기반으로 새로운 합성 데이터를 생성하여 클래스 균형을 맞춥니다.
- **주의사항**
  - 이웃을 기반으로 데이터를 생성하므로, 이상치나 노이즈가 많은 데이터에 적용하면 오히려 성능이 저하 가능
  - 샘플링은 **학습 데이터에만 적용**해야 함 (테스트 데이터는 원본 분포를 유지해야 함)
- **코드 예시**

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled_smote, y_resampled_smote = smote.fit_resample(X, y)
```

```
print(f"Resampled dataset shape %s" % Counter(y_resampled_smote))
# Resampled dataset shape Counter({0: 990, 1: 990})
```

- **결과 해석**
  - 소수 클래스(1)의 데이터 10개가 990개로 늘어나, 다수 클래스(0)와 동일한 수의 샘플을 갖게 되었습니다.
  - 늘어난 980개의 데이터는 기존 데이터를 복제한 것이 아니라, 새롭게 합성된 데이터입니다.

3. Class Weight

- **용도**: 모델 학습 시 소수 클래스에 더 높은 가중치를 부여합니다.
- **주의사항**
  - 데이터를 변경하지 않으므로 정보 손실이나 오버피팅 위험이 적습니다.
  - `class_weight='balanced'` 옵션은 클래스 빈도의 역수에 비례하여 자동으로 가중치를 계산해줍니다.
- **코드 예시**:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 1. 가중치 미적용
model_no_weight = LogisticRegression(solver='liblinear')
model_no_weight.fit(X_train, y_train)
pred_no_weight = model_no_weight.predict(X_test)
print("--- No Class Weight ---")
print(classification_report(y_test, pred_no_weight, zero_division=0))

# 2. 가중치 적용
model_with_weight = LogisticRegression(solver='liblinear',
class_weight='balanced')
model_with_weight.fit(X_train, y_train)
pred_with_weight = model_with_weight.predict(X_test)
print("\n--- With Class Weight ('balanced') ---")
print(classification_report(y_test, pred_with_weight, zero_division=0))
```

- **결과 해석**:
  - **가중치 미적용**: 모델이 대부분을 0으로 예측하여, 클래스 1에 대한 `recall`과 `f1-score`가 0으로 나옵니다. 즉, 소수 클래스를 전혀 탐지하지 못합니다.
  - **가중치 적용**: `accuracy`는 다소 감소할 수 있지만, 클래스 1에 대한 `recall`과 `f1-score`가 크게 향상됩니다. 이는 모델이 소수 클래스를 더 잘 탐지하게 되었음을 의미합니다.
  - 다만 예제 데이터셋이 잘 나눠져 있어 미적용 성능이 더 좋게 나온다.
    - 이런 특수한 경우가 있을 수 있으므로, 우선 전체적인 성능을 확인하고 진행하는 것이 좋을 것

장단점 및 대안

방법	장점	단점	대안
언더샘플링	계산 비용이 감소하고, 데이터 저장 공간이 절약됨.	다수 클래스의 중요한 정보를 잃을 수 있어 모델 성능 저하 위험이 큼.	**Tomek Links**: 서로 다른 클래스에 속하면서 가장 가까운 이웃인 데이터 쌍을 찾아, 다수 클래스의 데이터를 제거하는 방법. **Edited Nearest Neighbours (ENN)**: 다수 클래스의 데이터 중, 주변의 k개 이웃 대부분이 다른 클래스인 데이터를 제거하는 방법.
오버샘플링 (SMOTE)	정보 손실이 없음. 언더샘플링보다 일반적으로 성능이 좋음.	새로운 데이터를 생성하므로 계산 비용이 증가함. 노이즈나 이상치 주변에 데이터를 생성하여 클래스 간 경계를 모호하게 만들 수 있음.	**ADASYN (Adaptive Synthetic Sampling)**: 분류하기 더 어려운 소수 클래스 데이터 주변에 더 많은 합성 데이터를 생성하는 SMOTE의 발전된 형태. **Borderline-SMOTE**: 클래스 경계선에 있는 소수 클래스 데이터에 대해서만 SMOTE를 적용하여 더 효과적으로 학습을 도움.
Class Weight	구현이 매우 간편함. 데이터셋을 직접 변경하지 않아 정보 손실이나 부작용이 적음.	모든 모델이 `class_weight` 파라미터를 지원하지는 않음. 최적의 가중치를 직접 찾아야 할 수도 있음.	**Cost-Sensitive Learning**: Class Weight와 유사한 개념으로, 오분류 비용(misclassification cost)을 직접 정의하여 모델의 손실 함수에 반영하는 방법.

팁: 언더샘플링과 오버샘플링을 결합한 **하이브리드 방법**도 효과적입니다. 예를 들어, `SMOTEENN`은 SMOTE로 오버샘플링을 수행한 후, ENN으로 노이즈가 될 수 있는 데이터를 제거하여 두 방법의 장점을 결합합니다.