[추가자료]

Class Imbalance 문제

Resampling

강사 : 한기영

Class Imbalances

✓ 일반적인 알고리즘들

- 데이터가 클래스 내에서 **고르게 분포**되어 있다고 가정
- 클래스 불균형 문제의 경우 보통 알고리즘이 **다수 클래스를 더 많이 예측하는 쪽으로 모델이 편향**되는 경향이 있음.

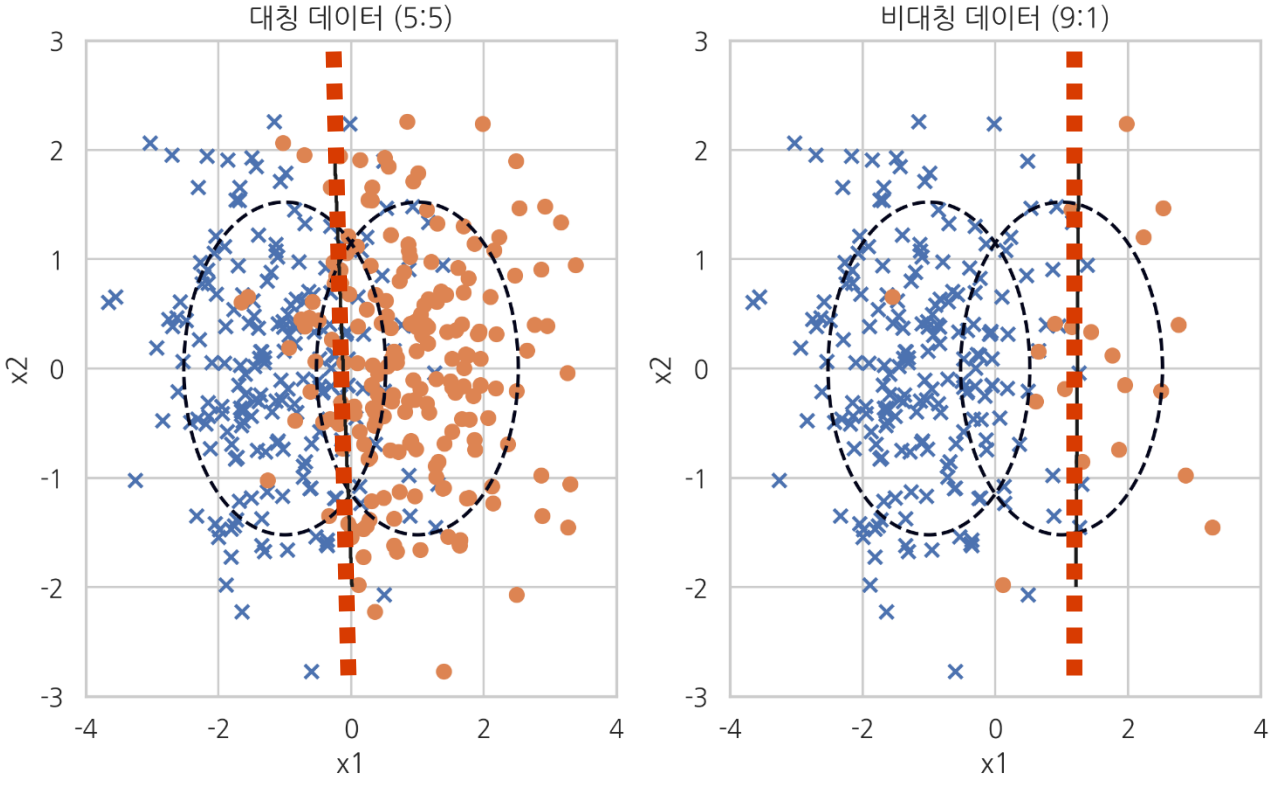
➔소수의 클래스에서 오분류 비율이 높아짐.

✓ 비즈니스 상황에서 Class Imbalance 는 흔한 현상

- 고객 이탈 예측 : 잔존 > 이탈
- 금융 비정상 거래 예측 : 정상 > 비정상(사기거래)
- 제조 공정간 불량 예측 : 정상 > 불량

Class Imbalances

✓왜 문제인가?



Class Imbalances

✓왜 문제인가?

- 결과로, Accuracy는 높지만 적은 Class 쪽 Recall은 형편없이 낮음.
- 경우에 따라 아래와 같이 나오기도 함.

예측값

실제값

```
      0    1
0 [[283   0]
1  [ 31   0]]
```

	precision	recall	f1-score	support
0	0.90	1.00	0.95	283
1	0.00	0.00	0.00	31
accuracy			0.90	314
macro avg	0.45	0.50	0.47	314
weighted avg	0.81	0.90	0.85	314

Class Imbalances : 해결을 위한 두가지 방법

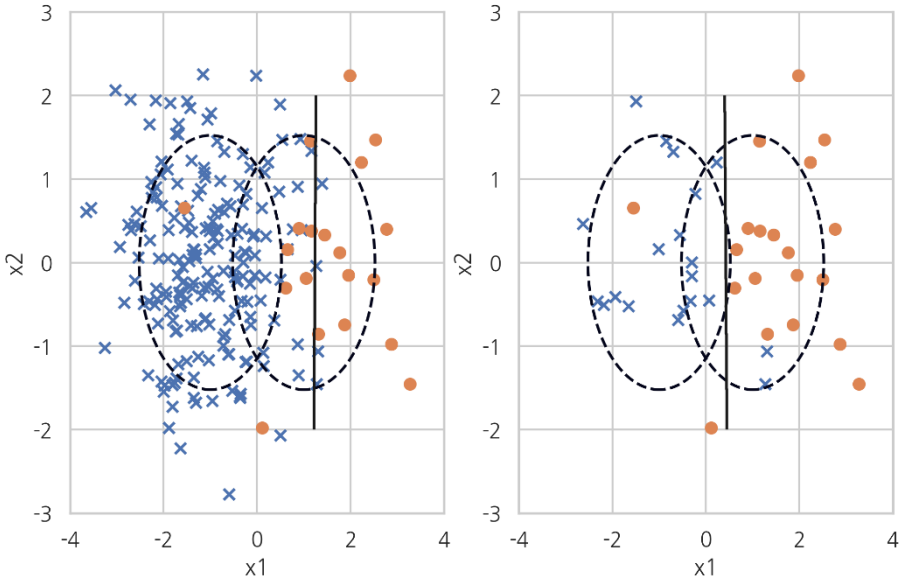
- ✓ **Resampling**
- ✓ **Class Weight 조정**

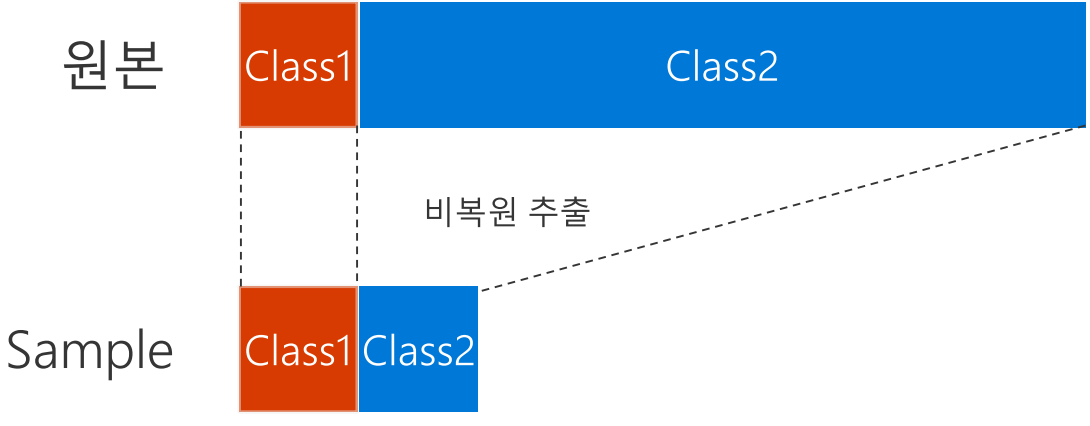
- ✓ **주의!**
  - 전반적인 성능을 높이기 위한 작업이 아니라,
  - **소수 Class의 성능을 높이기 위한 작업(다수 Class는 성능이 떨어짐)**

Resampling

✓ Down Sampling

- 다수 Class의 데이터를
- 소수 Class 수 만큼 random sampling

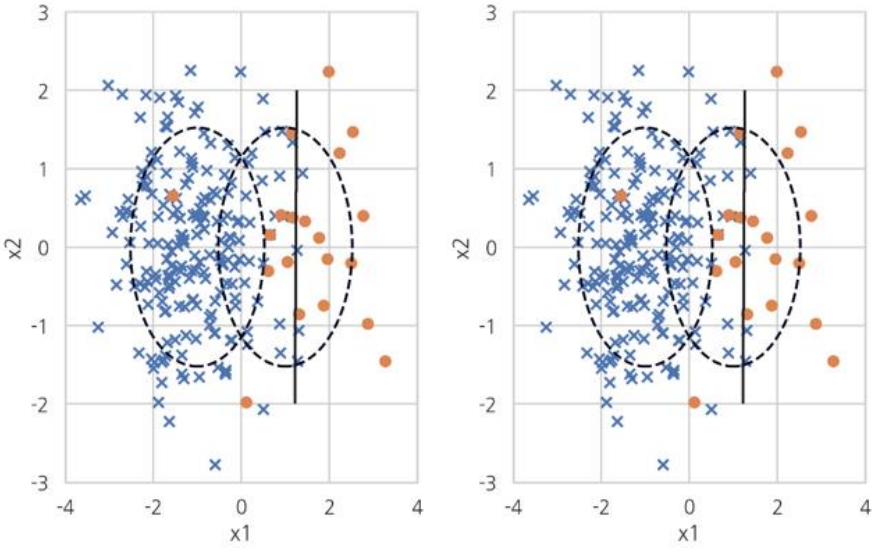


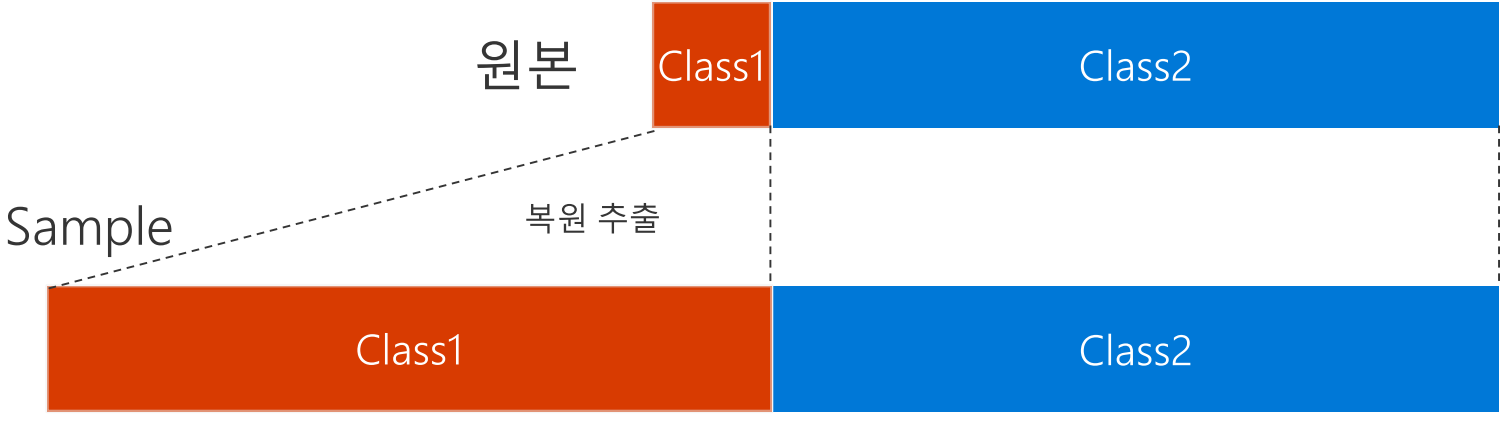


Resampling

✓ Up Sampling

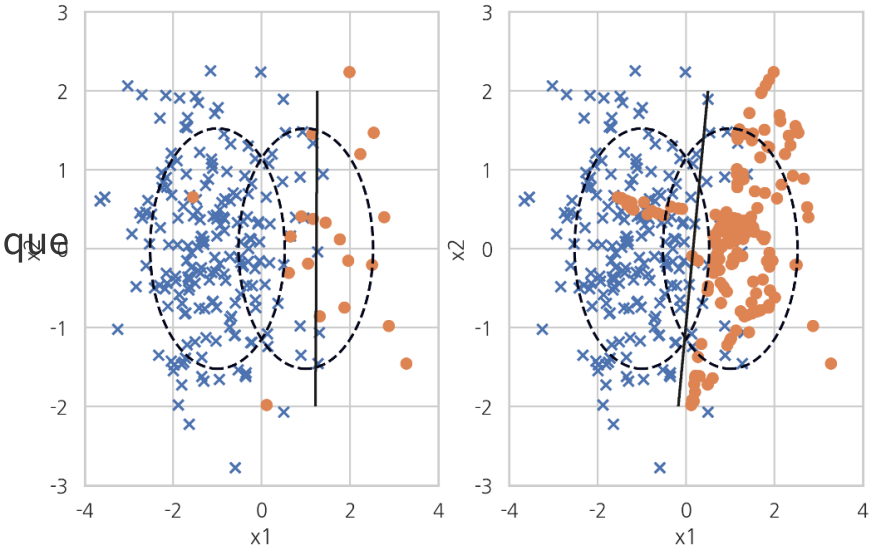
- 소수 Class의 데이터를
- 다수 Class 수 만큼
- random sampling(복원 추출)

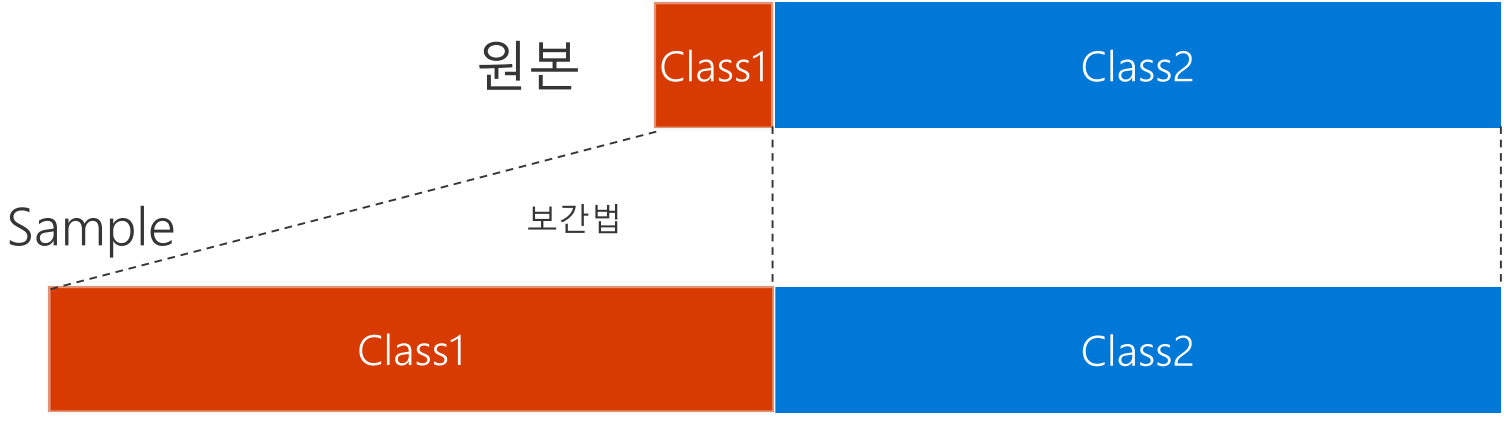




Resampling

- ✓ **SMOTE** Synthetic Minority Oversampling TEchnique
  - 소수 Class의 데이터를
  - 보간법(Interpolation)으로 새로운 데이터를 만들어 냄

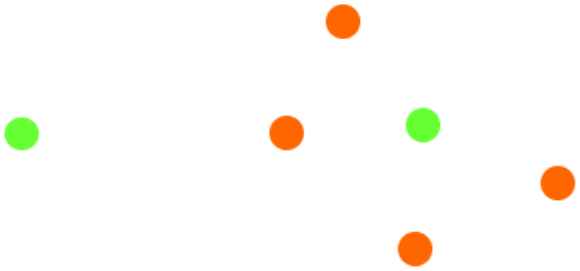




Resampling

✓ SMOTE(Synthetic Minority Oversampling TEchnique)

- 기존 소수 샘플을 보간법(Interpolation)으로 새로운 데이터를 만들어 냄



Step 1:
Identify a point from the minority class

Step 2:
Identify it's nearest neighbors. (Here 4 neighbors are taken. All minority neighbors)

Step 3:
Assume we want to add 2 times more synthetic observations
Select 2 nearest neighbor randomly
Draw a line between the point and the selected neighbors
Create a synthetic observation along the line

출처 : https://towardsdatascience.com/class-imbalance-smote-borderline-smote-adasyn-6e36c78d804

Class Imbalance 문제

Class Weight

모델링 절차 – Review

Logistic Regression

단계1 : 모델(함수)의 구조를 잡는다.
- 알고리즘마다 다르겠지요.

$$P(x) = \frac{1}{1 + e^{-f(x)}} \quad , f(x) = w_1 x_1 + w_0$$

단계2 : 초기값(parameter)을 할당한다.

$$[w_0, w_1] = [\ 0.2\ ,\ \ 0.91\ ]$$

단계3 : 모델을 만들어 예측 결과를 뽑는다.

$$pred : [\ 0.87\ ,\ \ 0.32, \ldots\ ]$$

단계4 : 오차를 계산한다.
- 알고리즘마다 오차 계산방식(***loss function***)이 달라집니다.
  - 회귀모델은 대체로 mse
  - 분류모델 중 tree기반 모델은 cross-entropy, 로지스틱 회귀는 Log Loss

$$log\ loss = \frac{1}{N}\sum\{ -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \}$$

단계5 : 오차를 줄이는 방향으로 Parameter를 조정
- ***Optimizer***

$$Gradient\ Decent, ADAM \ldots$$

단계6 : 다시 3단계로 올라가 반복한다.
- max iteration에 도달하거나, 오차의 변동이 (거의) 없으면 끝.

Class Weight

- ✓ **Resampling 없이 클래스에 가중치를 부여하여 Class Imbalance 문제 해결**
  - 학습하는 동안, 알고리즘의 **비용 함수에서 소수 클래스에 더 많은 가중치를 부여**하여
  - 소수 클래스에 더 높은 패널티를 제공함으로써,
  - 알고리즘이 소수 클래스에 대한 오류를 줄이게 됨.

- ✓ **sklearn 의 알고리즘들은 대부분 옵션 class_weight 제공**
  - class_weight = 'None' : 기본값
  - class_weight = 'balanced' : y_train 의 class 비율을 역으로 적용
  - class_weight = {0:0.2, 1:0.8} : 비율 지정. 단 비율의 합은 1

Class Weight 조절 – *Logistic Regression*

Class0 = wt_0 , Class1 = wt_1

단계1 : 모델(함수)의 구조를 잡는다.
- 알고리즘마다 다르겠지요.

$$P(x) = \frac{1}{1+e^{-f(x)}} \quad , f(x) = w_1 x_1 + w_0$$

단계2 : 초기값(parameter)을 할당한다.

$$[w_0, w_1] = [\ 0.2\ ,\ 1.7\ ]$$

단계3 : 모델을 만들어 예측 결과를 뽑는다.

$$pred : [\ 0.87\ ,\ 0.32, \ldots\ ]$$

단계4 : 오차를 계산한다.
- 알고리즘마다 오차 계산방식(***loss function***)이 달라집니다.
  - 회귀모델은 대체로 mse
  - 분류모델은 tree 기반 모델은 cross-entropy, 로지스틱 회귀는 Log Loss

$$log\ loss = \frac{1}{N}\sum\{ -(wt_1 y_i \log(\hat{y}_i) + wt_0 (1-y_i)\log(1-\hat{y}_i)) \}$$

단계5 : 오차를 줄이는 방향으로 Parameter를 조정
- ***Optimizer***

$$Gradient\ Decent, ADAM \ldots$$

단계6 : 다시 3단계로 올라가 반복한다.
- max iteration에 도달하거나, 오차의 변동이 (거의) 없으면 끝.