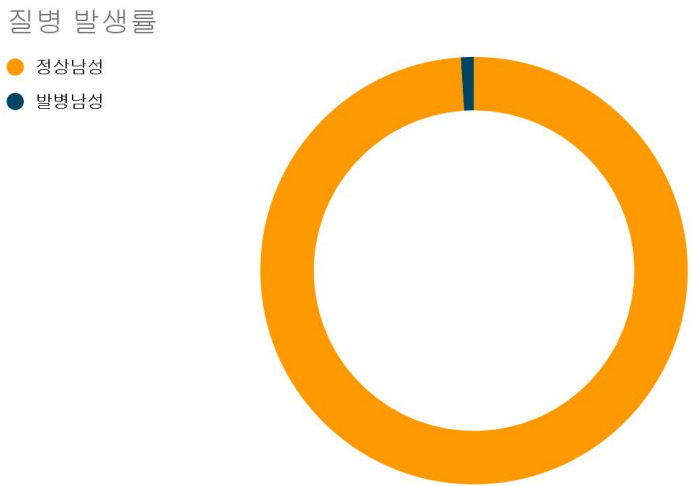
# 불균형 Class 데이터 처리

- Sampling은 왜 할까요?
- 클래스 불균형 문제란, 분류를 목적으로 데이터 셋에 클래스 라벨의 비율이 균형을 맞추지 않고, 한쪽으로 치우친 경우를 말합니다
- 이런 경우 모델이 각 클래스의 데이터를 제대로 학습하기 어려워집니다

- 우리나라 질병 발생률

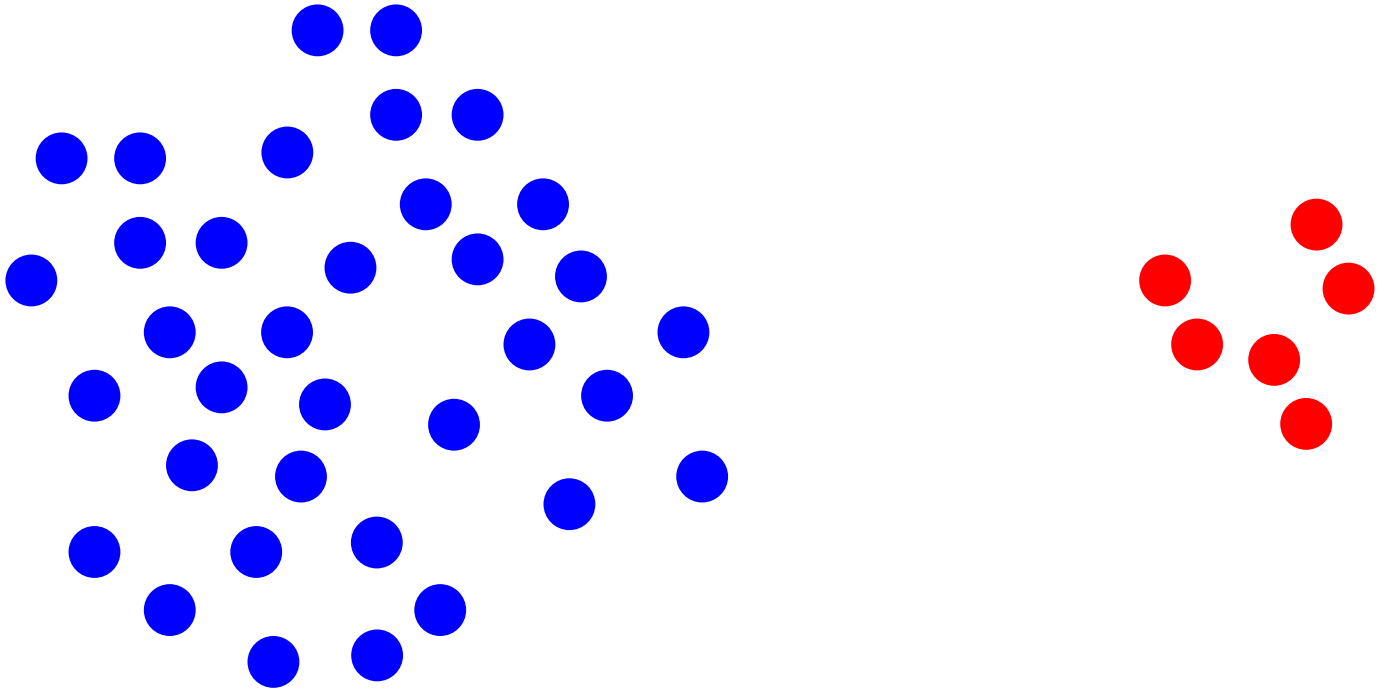
예를 들어, 발병남성의 1%이고 정상남성 99%라면 불균형 데이터입니다



- 제조업의 불량 판정

제조업에서는 불량보다는 정상인 제품이 훨씬 많으 것이므로, 불균형 데이터입니다

정상 이상

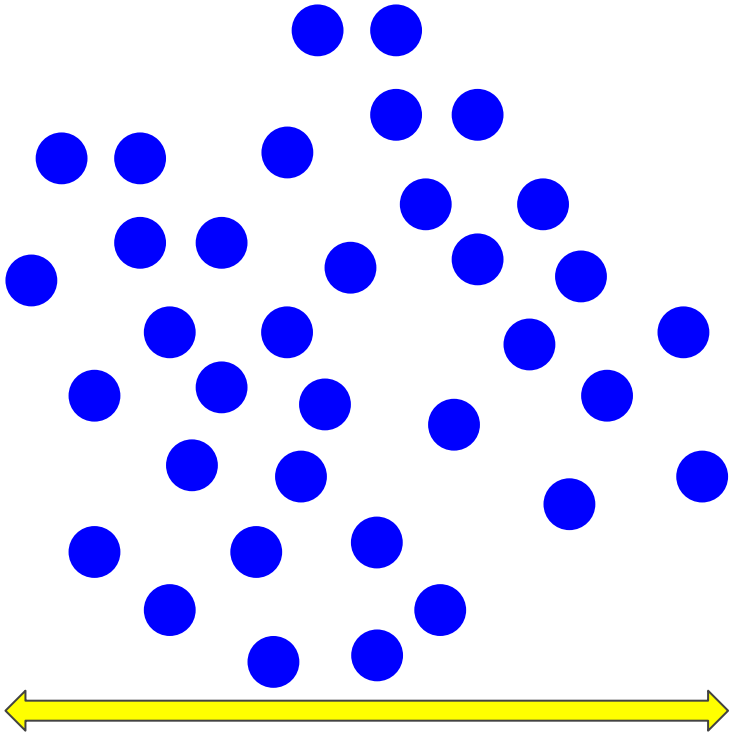


- 불균형 데이터는 무엇일까요?
- 정상 범주의 관측치 수와 이상 범주의 관측치 수의 차이가 크게 나타나는 경우
- 클래스 별 관측치의 수가 현저하게 차이가 나는 데이터

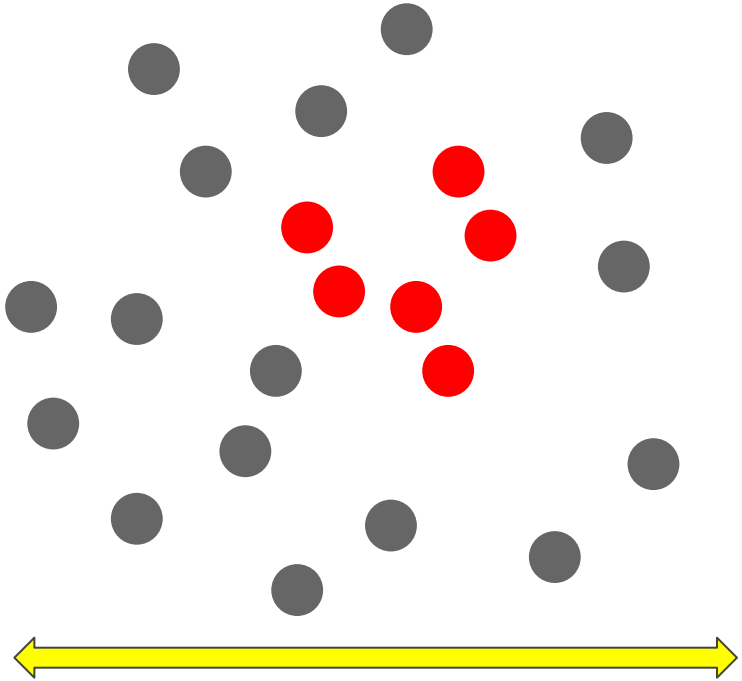
- 정상(다수)을 정확히 분류 VS 이상(소수)을 정확히 분류를 한다면
- 일반적으로 이상(소수)을 정확히 분류하는 것이 중요합니다
- 적절한 분류경계선이 형성되지 못하면 이상(소수)을 정확하게 찾아내지 못합니다

# 모델링 관점에서 생각하기

정상 이상



정상 관측치 분포

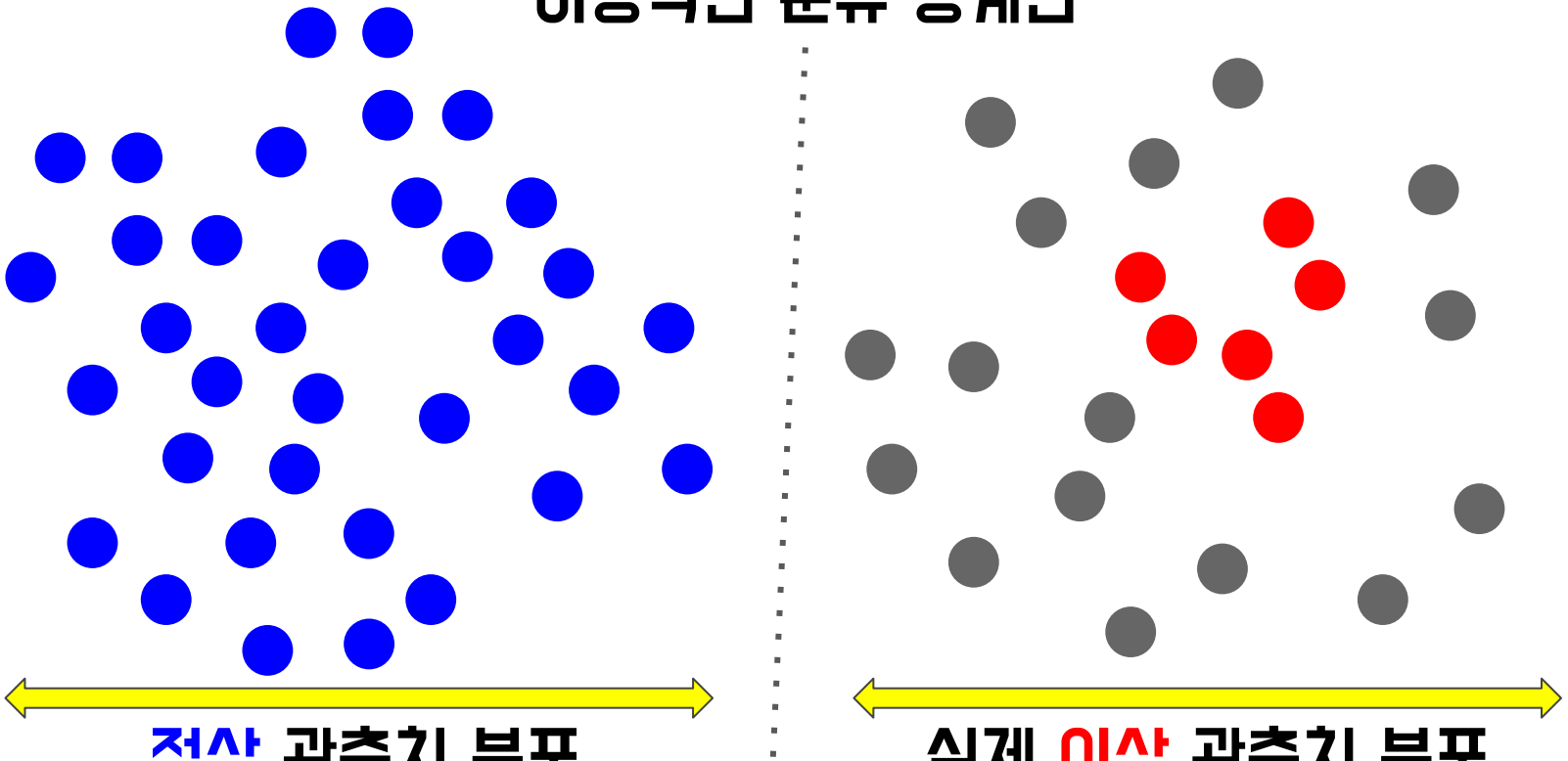


실제 이상 관측치 분포

# 모델링 관점에서 생각하기

정상 이상

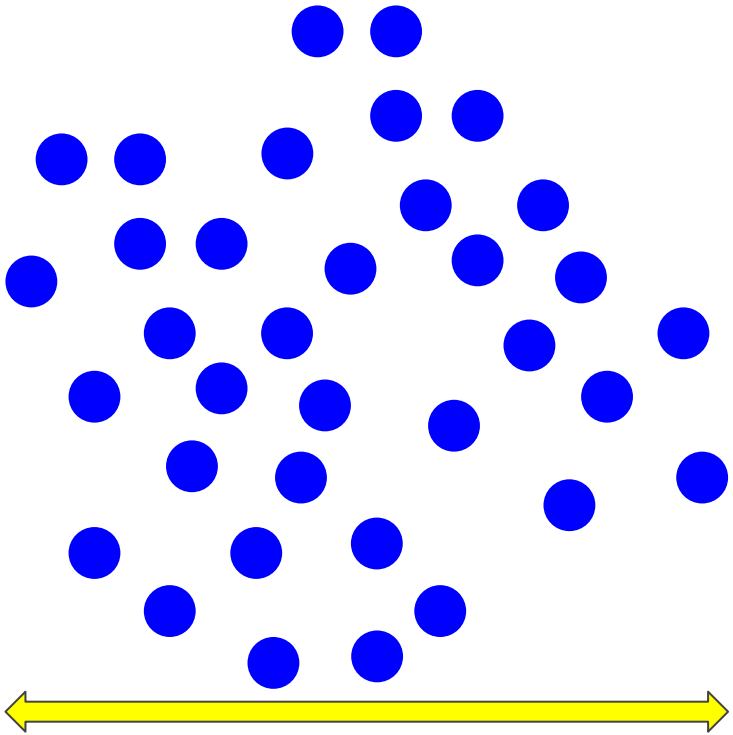
이상적인 분류 경계선



정상 관측치 분포

실제 이상 관측치 분포

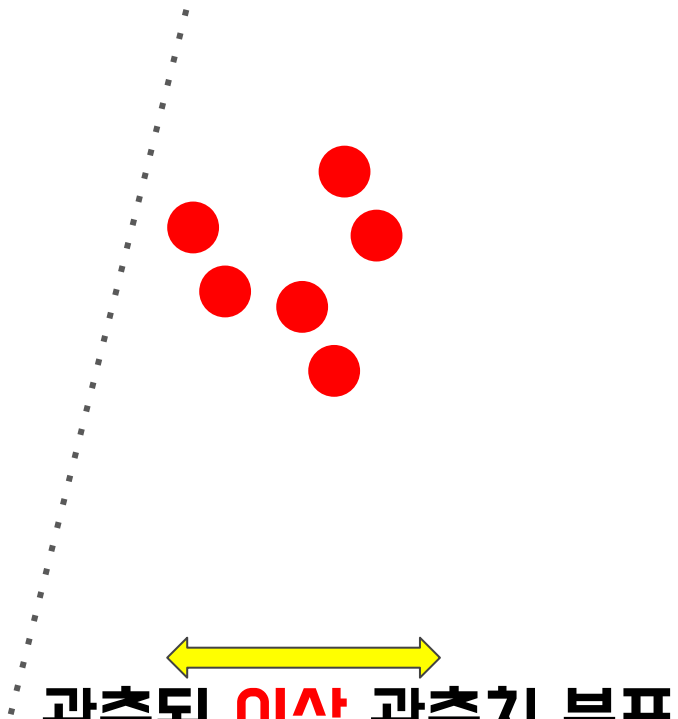
# 모델링 관점에서 생각하기



정상 관측치 분포

## 정상 이상

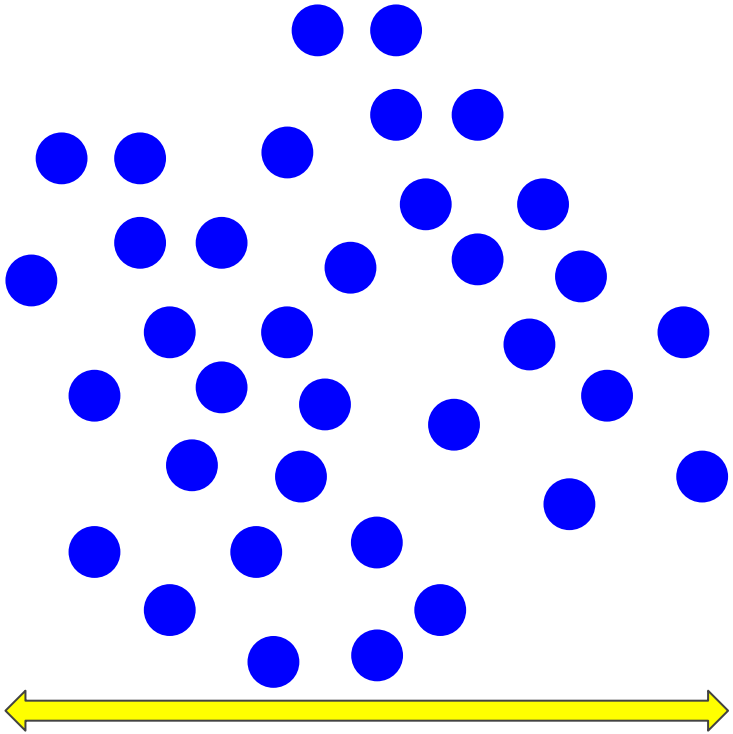
실제 분류 경계선



관측된 이상 관측치 분포

# 모델링 관점에서 생각하기

## 정상 이상



정상 관측치 분포


실제 이상 관측치 분포

| Confusion matrix(정오행렬) | | 예측 | |
|---|---|---|---|
| | | 이상 | 정상 |
| 실제 | 이상 | 10 | 10 |
| | 정상 | 0 | 80 |

- 예측정확도(Accuracy)

10+80 / 10+10+0+80로 0.9로 정확도가 높지만
하지만 이상은 반밖에 예측을 못한 것입니다

- 불균형 데이터를 사용하면 높은 예측정확도를 보일 수는 있지만
- 모델 성능에 대한 왜곡이 있을 수 있습니다
- 이상을 반박에 예측하지 못하는 모델을 공장에서 사용할 수 있을까요?

- 많은 클래스 데이터 수를 감소시키는 Undersampling 기법
- 언더샘플링 다수 범주 관측치 제거해서 계산 시간을 감소
- 데이터 클랜징으로 클래스 오버랩 감소가 가능
- 데이터 제거로 인한 정보 손실 발생

- Undersampling 기법
  - Random undersampling
  - Tomek links
  - Condensed Nearest Neighbor Rule
  - One-sided selection

- 적은 클래스 데이터 수를 증가시키는 Oversampling 기법
- 정보 손실이 없음
- 대부분의 경우 언더샘플링에 비해 높은 분류 정확도를 보임
- 과적합 가능성
- 계산 시간이 증가

- SMOTE 기법을 사용해보겠습니다
- SMOTE(synthetic minority oversampling technique)는 데이터의 개수가 적은 클래스의 표본을 가져온 뒤 임의의 값을 추가하여 새로운 샘플을 만들어 데이터에 추가하는 오버샘플링 방식입니다
- imblearn 모듈을 사용합니다

```
#모듈 설치하기
!pip install imblearn

#모듈 사용하기
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
```

```
data = pd.read_csv('uci-secom.csv')
data = data.replace(np.NaN, 0)
data = data.drop(columns = ['Time'], axis = 1)
x = data.drop(columns = ['Pass/Fail'], axis = 1)
y = data['Pass/Fail']
y = y.to_numpy().ravel()
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.25,random_state=10)
```

```
#모델링 함수를 만듭니다.
def modeling(model,x_train,x_test,y_train,y_test):
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    metrics(y_test,pred)

#평가 지표를 만듭니다.
def metrics(y_test,pred):
    accuracy = accuracy_score(y_test,pred)
    precision = precision_score(y_test,pred)
    recall = recall_score(y_test,pred)
    print('정확도 : {0:.2f}, 정밀도 : {1:.2f}, 재현율 :
{2:.2f}'.format(accuracy,precision,recall))
```

```
#로지스틱 회귀 모델로 학습을 합니다
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
modeling(LR ,x_train,x_test,y_train,y_test)
```

```
#SMOTE로 사용해보겠습니다
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
#fit_sample 함수로 오버샘플링을 합니다
x_train_over,y_train_over = smote.fit_sample(x_train,y_train)

#결과를 확인합니다.
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', x_train.shape,
y_train.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ',
x_train_over.shape, y_train_over.shape)
```

```
#오버샘플링 한 것으로 테스트해봅니다.
LR = LogisticRegression()
modeling(LR, x_train_over,x_test,y_train_over,y_test)
```

- imblearn의 over_sampling과 under_sampling를 사용해보겠습니다

```
#모듈을 가져옵니다.
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

#변수를 만듭니다.
ros = RandomOverSampler()
rus = RandomUnderSampler()

#fit_resample를 사용합니다
oversampled_data, oversampled_label = ros.fit_resample(x, y)
oversampled_data = pd.DataFrame(oversampled_data)
```

```
undersampled_data, undersampled_label = rus.fit_resample(x, y)
undersampled_data = pd.DataFrame(undersampled_data)

#shape로 확인합니다
print(oversampled_data.shape, undersampled_data.shape)

#이것을 이용해서 학습을 다시 해서 확인해봅니다.
```