

# Bitamin 12주차 세션

review

1 0 기   4 조   노 지 예   부 도 현   임 청 수   한 세 림

# 복습과제 오답 리뷰

1 1 주 차 리 뷰

# 복습과제 분석

- sklearn 모듈을 이용하여 예측하는 부분은 오답률이 높지 않다.
- 데이터 가공 및 데이터셋 반환 부분의 오답률이 높다. (특히 사용자 정의 함수 코드에 오답 多.)
- 직접 함수를 정의하고 사용하는 문제가 많았어서, 함수 이름과 변수가 생소하게 느껴졌을 것으로 판단된다.

```
def get_train_test_dataset(df=None):
    # 인자로 입력된 DataFrame의 사전 데이터 가공이 완료된 복사 DataFrame 반환
    df_copy = get_preprocessed_df(df)
    # DataFrame의 맨 마지막 컬럼이 레이블, 나머지는 피쳐들
    X_features = df_copy.iloc[:, :-1]
    y_target = df_copy.iloc[:, -1]
    # 데이터 분할, test_size = .3, stratify 포함, random_state = 0
    X_train, X_test, y_train, y_test = train_test_split(X_features, y_target,
                                                        test_size = 0.3,
                                                        random_state = 0)

    # 학습과 테스트 데이터 세트 반환
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = get_train_test_dataset(data)
```

```
# get_preprocessed_df( )를 로그 변환 후 V14 피쳐의 이상치 데이터를 삭제하는 로직으로 변경.
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    amount_n = np.log1p(df_copy['Amount'])
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)
    # 이상치 데이터 삭제하는 로직 추가
    outlier_index = get_outlier(_____)
    #행기준 이상치 drop
    -----
    return df_copy
```

# 복습과제 분석

- 사용자 정의 함수의 정의

```
# 사전 데이터 가공 후 학습과 테스트 데이터 세트를 반환하는 함수.  
def get_train_test_dataset(df=None):  
    # 인자로 입력된 DataFrame의 사전 데이터 가공이 완료된 복사 DataFrame 반환  
    df_copy = get_preprocessed_df(df)  
    # DataFrame의 맨 마지막 컬럼이 레이블, 나머지는 피쳐들  
    X_features = df_copy.iloc[:, :-1]  
    y_target = df_copy.iloc[:, -1]  
    # train_test_split()으로 학습과 테스트 데이터 분할. stratify=y_target으로 Stratified 기반 분할  
    X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.3, random_state=0, stratify=y_target)  
    # 학습과 테스트 데이터 세트 반환  
    return X_train, X_test, y_train, y_test
```

```
X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)
```

Pandas.DataFrame.iloc[row, column]

→ 행은 그대로 (:), X 열은 마지막 줄 제외(:-1), y열은 마지막 줄만(-1)

Stratify=y

→ Train set과 Test set에 y 데이터가 동일 비율로 들어가게 한다.

→ Train이나 test 한쪽에 데이터가 몰리는 것을 방지.

# 복습과제 분석

- 사용자 정의 함수의 사용

```
# get_processed_df( )를 로그 변환 후 V14 피처의 이상치 데이터를 삭제하는 로직으로 변경.
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    amount_n = np.log1p(df_copy['Amount'])
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time', 'Amount'], axis=1, inplace=True)
    # 이상치 데이터 삭제하는 로직 추가
    outlier_index = get_outlier(df=df_copy, column='V14', weight=1.5)
    df_copy.drop(outlier_index, axis=0, inplace=True)
    return df_copy
```

```
def get_outlier(df=None, column=None, weight=1.5):
    # fraud에 해당하는 column 데이터만 추출, 1/4 분위와 3/4 분위 지점을 np.percentile로 구함.
    fraud = df[df['Class']==1][column]
    quantile_25 = np.percentile(fraud.values, 25)
    quantile_75 = np.percentile(fraud.values, 75)
    # IQR을 구하고, IQR에 1.5를 곱하여 최대값과 최소값 지점 구함.
    iqr = quantile_75 - quantile_25
    iqr_weight = iqr * weight
    lowest_val = quantile_25 - iqr_weight
    highest_val = quantile_75 + iqr_weight
    # 최대값 보다 크거나, 최소값 보다 작은 값을 아웃라이어로 설정하고 DataFrame index 반환.
    outlier_index = fraud[(fraud < lowest_val) | (fraud > highest_val)].index
    return outlier_index
```

get\_outlier 함수, get\_model\_train\_eval 함수의 인자 작성

→ 모듈 이름이 없는 사용자 정의 함수는 위쪽에 정의된 부분을 참고하여 input을 입력해야 함.

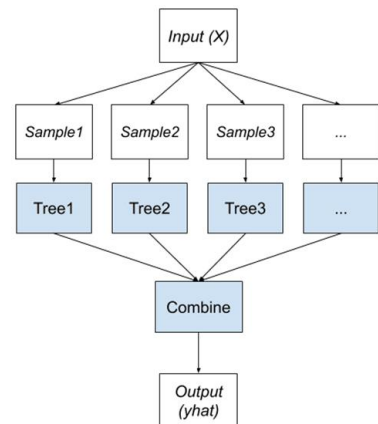
# 앙상블

1 1 주 차 리 뷰

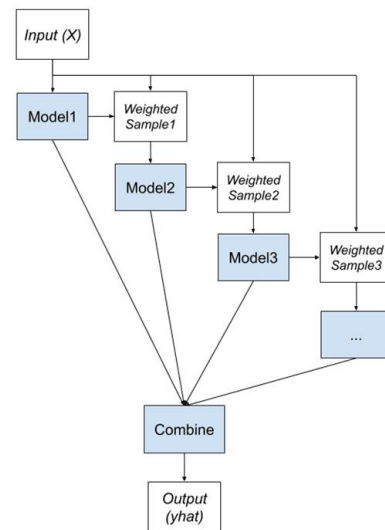
# 앙상블이란?

- 여러 개의 Weak Classifier를 결합하여 Strong Classifier를 만드는 기법.
- 결정 트리와 같이 Overfitting되기 쉬운 모델로 주로 구성한다.
- 앙상블의 종류: **배깅(Bagging)**, **부스팅(Boosting)**, **스태킹(Stacking)** 등.

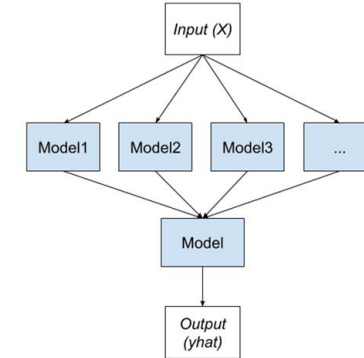
Bagging Ensemble



Boosting Ensemble



Stacking Ensemble



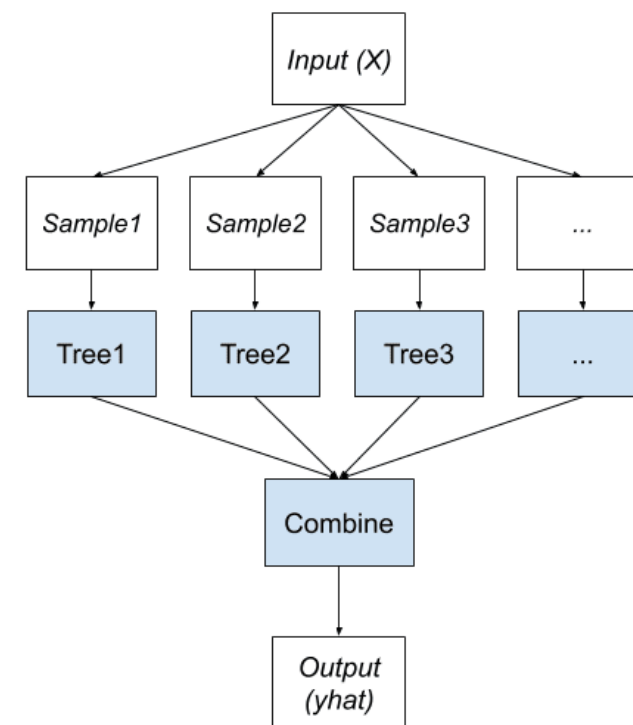
# 배깅 (Bagging, Bootstrap AGgregatING)

- 데이터셋을 **부트스트랩(Bootstrap)** 기법을 이용하여 데이터셋을 나누고, 서로 다른 데이터셋으로 학습을 수행한 다수의 Weak Classifier의 결과를 **집계(Aggregating)**한다.

ex. **랜덤 포레스트(Random Forest)**:

각 데이터셋을 (1) 부트스트랩하고, (2) 부트스트랩한 데이터셋으로 결정 트리를 구성하고, (3) 각 결정 트리의 예측 결과를 Voting하여 집계하는 방식.

Bagging Ensemble





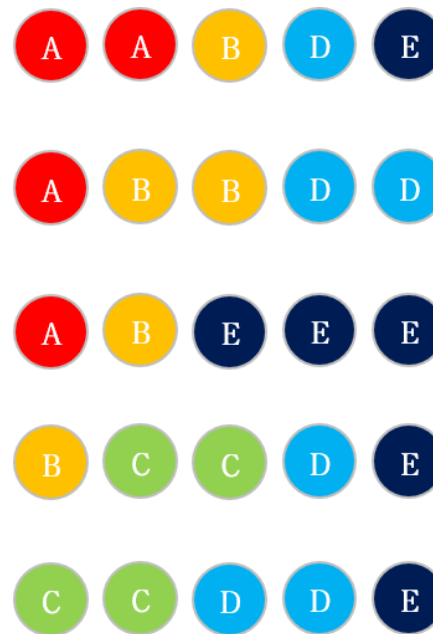
양상블

# 부트스트랩 (Bootstrap)

- 중복된 샘플을 허용하여 N개의 샘플을 모델의 개수만큼 Random Sampling하는 방식.
- 학습 시 Overfitting을 방지할 수 있다.

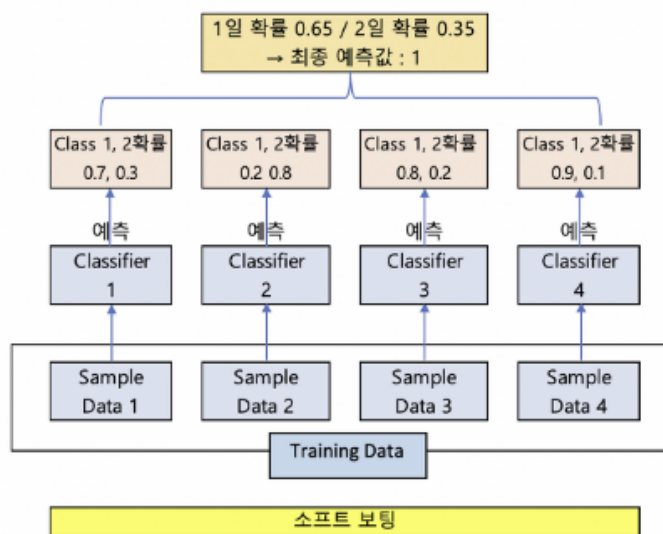
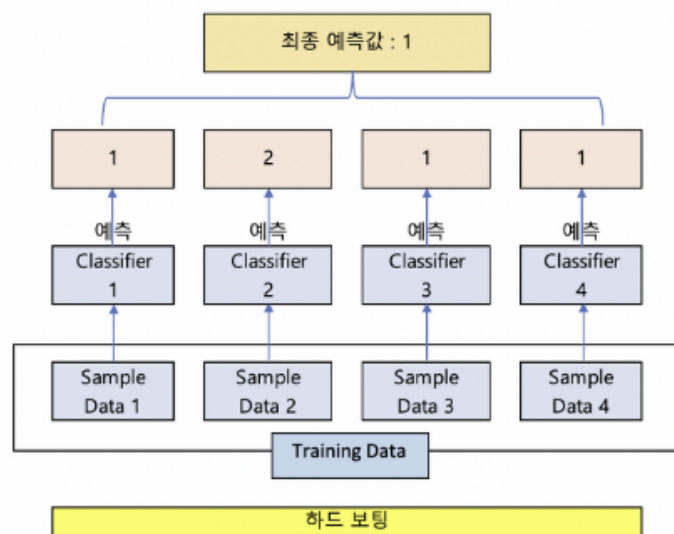


Bootstrap  
→



# 보팅 (Voting)

- Aggregating의 대표적인 방법으로, 각 분류 모델의 예측 결과를 **투표를 통해 결합**한다.
- **Hard Voting**: 다수결에 따라 가장 많은 예측값이 최종 예측값이 된다.
- **Soft Voting**: 각 분류기에 가중치를 부여한 후 예측값을 산정한다.
- 일반적으로 Soft Voting이 성능이 좋아 많이 쓰인다.



# 부스팅 (Boosting)

- 이전 분류기의 학습 결과를 토대로 다음 분류기의 가중치를 조정해가며 순차적으로 학습하는 방법이다.
- **장점** - 오답에 가중치를 두어 오답을 보강하고 정확도를 높이는 데 유리함.
- **단점** - Outlier가 있을 경우 영향을 많이 받음.

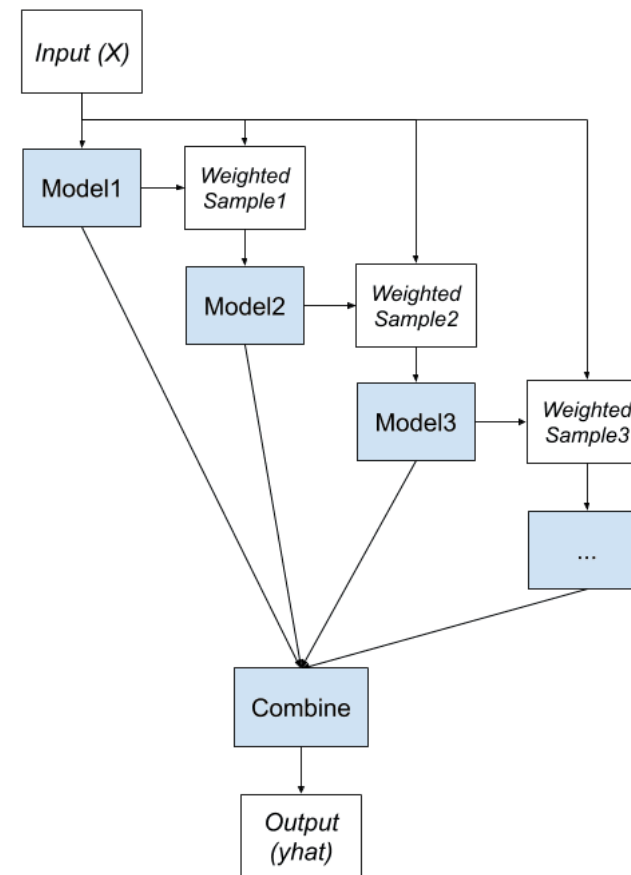
ex.

**AdaBoost**: 개별 분류기에 가중치를 주어 최종 모델을 만듦.

**Gradient Boost(GBM)**: 경사하강법을 이용하여 손실함수를 최소화 하는 쪽으로 학습함.

**XGBoost**: GBM의 단점을 정규화, 병렬 처리, 교차 검증 등을 이용해 보완함.

Boosting Ensemble



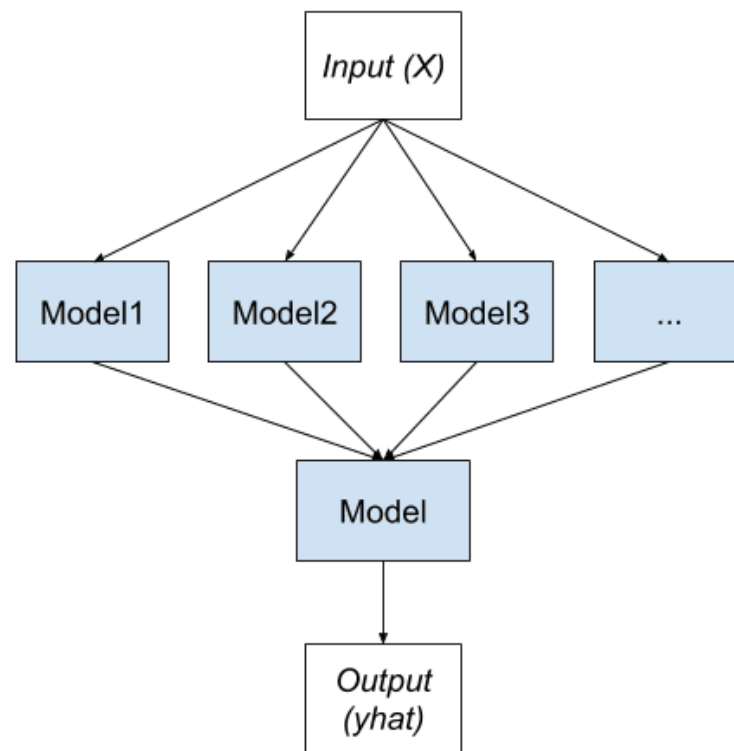
# 스태킹 (Stacking)

- 개별 모델이 예측한 데이터를 다시 하나의 학습 데이터셋으로 사용하여 최종 분류기로 학습하는 방법이다.
- 기본 스택킹 모델은 Overfitting의 위험이 높기 때문에 잘 사용하지 않고, 주로 CV 기반 스택킹 앙상블을 이용한다.

ex. CV 기반 스택킹 앙상블:

개별 모델로 학습된 데이터를 **교차 검증(CV, Cross Validation)**한 후 스택킹을 진행하는 방식.

Stacking Ensemble



# 불균형 데이터 처리

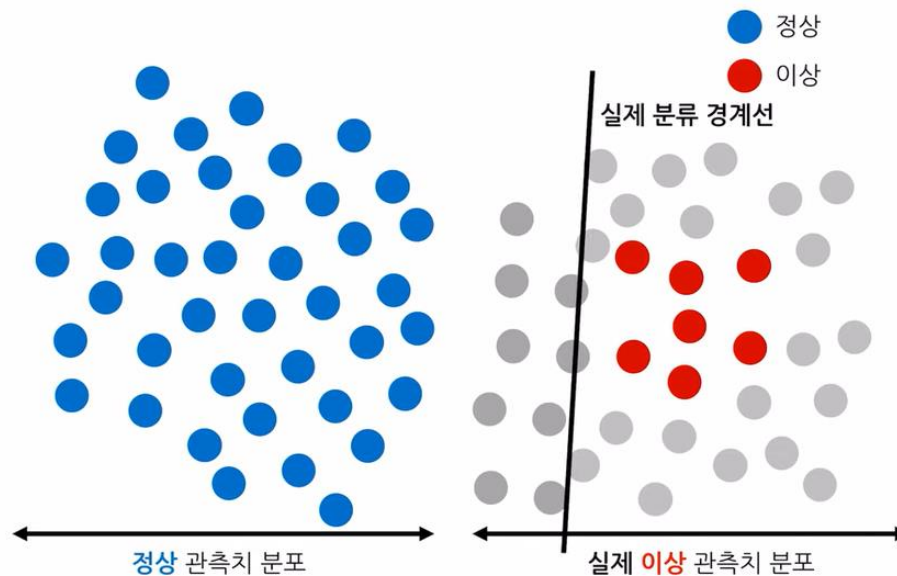
1 1 주 차 리 뷰

# 불균형 데이터

- **불균형 데이터**: 타겟 클래스의 데이터 관측치 수와 타겟이 아닌 클래스 데이터 관측치 수가 차이 나는 데이터.

ex. 암 환자의 수 vs 암 환자가 아닌 사람의 수

→ 일반적으로 타겟 데이터가 더 적기 때문에, 타겟 분류 기준을 잘 학습하기 위해 불균형 데이터 처리가 필요하다.



# 불균형 데이터 처리 기법

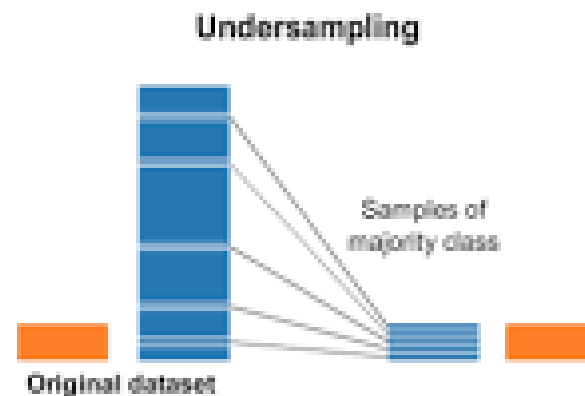
## (1) Undersampling

: 다수 클래스의 데이터를 소수 클래스의 데이터 수에 따라 줄이는 방법

장점 - 데이터 처리 및 학습 시간 단축

단점 - 정보 손실 발생 가능

ex. Random Sampling, Tomek Links, CNN 등



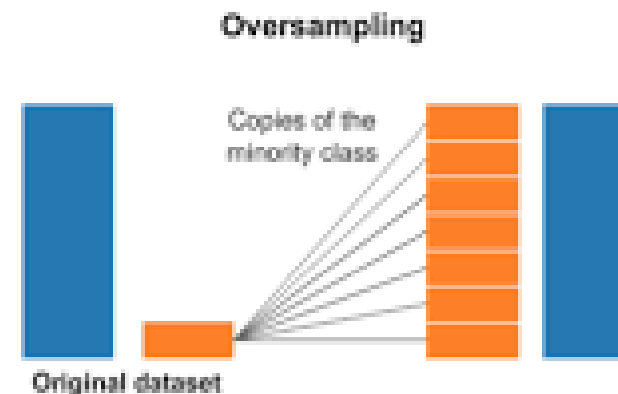
## (2) Oversampling

: 소수 클래스의 데이터를 다수 클래스의 데이터 수에 따라 늘리는 방법

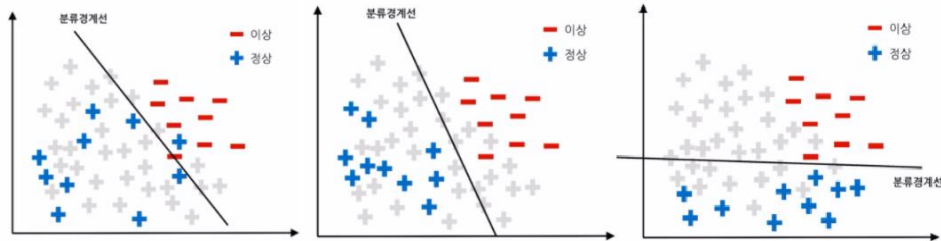
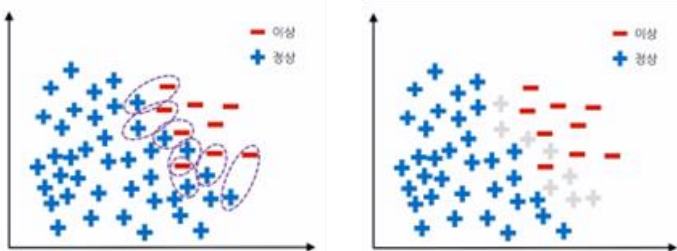
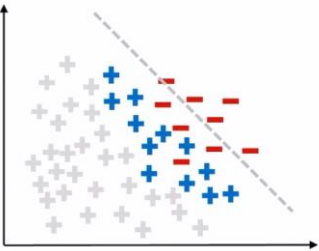
장점 - 정보 손실이 없음. 비교적 정확도가 높음.

단점 - 처리 시간 증가. Overfitting 가능성 有.

ex. Resampling, SMOTE, ADASYN, GAN 등

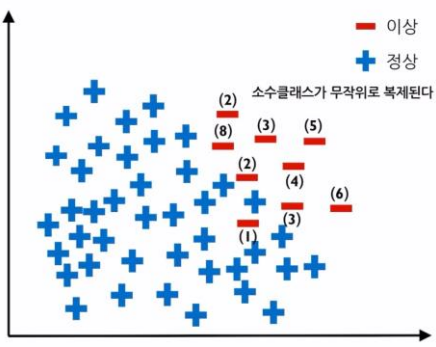
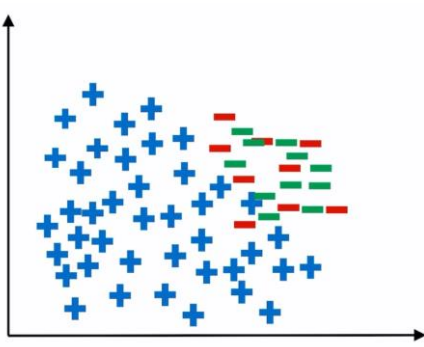
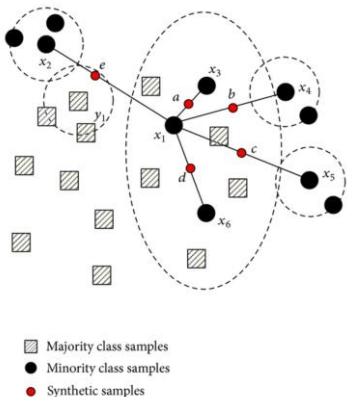
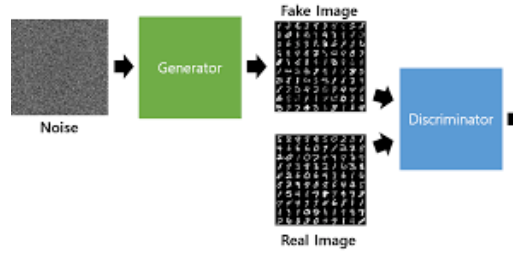


# 언더샘플링 (Undersampling)

종류		특징
Random Sampling		<ul style="list-style-type: none"> <li>- 다수 클래스에서 무작위로 샘플을 추출하는 방법.</li> <li>- 단점 - 매 시행마다 다른 결과를 얻음.</li> </ul>
Tomek Links		<ul style="list-style-type: none"> <li>- 서로 다른 클래스의 데이터 중 가장 거리가 짧은 데이터를 묶어 제거하는 방법.</li> </ul>
CNN		<ul style="list-style-type: none"> <li>- 다수 클래스 데이터 중 소수 클래스와 멀리 있는 데이터를 제거하는 방법.</li> </ul> <p>소수 클래스 데이터 전체와 무작위로 선택된 다수 클래스 데이터 하나로 sub-data를 구성한 후, 나머지 다수 클래스 데이터를 1-NN 방법으로 분류한다. 이후 소수 클래스와 가깝지 않은 쪽으로 분류된 샘플을 제거한다.</p>



# 오버샘플링 (Oversampling)

종류	Resampling	SMOTE	ADASYN	GAN
				
특징	<ul style="list-style-type: none"> <li>- 소수 클래스의 데이터를 <b>무작위로 복제</b>하는 방법.</li> <li>- 단점 - 소수 클래스에 Overfitting 발생 가능.</li> </ul>	<ul style="list-style-type: none"> <li>- 임의의 소수 클래스 데이터와 가까운 데이터 중 하나를 골라 <b>Synthetic 공식을 적용</b>해 가상의 데이터를 생성하는 방법.</li> <li>- <b>Borderline SMOTE</b>: 경계선에만 SMOTE를 적용하는 방법.</li> </ul>	<ul style="list-style-type: none"> <li>- Borderline SMOTE와 유사한 방법.</li> <li>- (차이점) 샘플링 개수를 <b>데이터 위치에 따라 다르게 설정</b>할 수 있음.</li> </ul>	<ul style="list-style-type: none"> <li>- 무작위로 노이즈를 생성해 가짜 샘플을 만들고, <b>가짜 샘플과 진짜 샘플을 판별</b>하여 판별하기 어려운 가짜 샘플을 남기는 방식.</li> <li>- 딥러닝을 이용한 최신 기법.</li> </ul>

# Thank You

1 2 주 차 복 습 세 셴