

# 3주차 발표

여채운 여유진 문수인

# 목차

---

#01 배깅 vs 부스팅 vs 스태킹

#02 AdaBoost vs Gradient Boost

#03 GBM 하이퍼 파라미터 및 튜닝

#04 XGBOOST

#05 LGBM

#06 CatBoost

#07 스태킹 앙상블



## 01 배킹 vs 부스팅 vs 스타킹



# 1.1 앙상블이란?



머신러닝을 위한 다양한 학습 알고리즘들을 결합하여 학습시키는 것으로, 예측력의 보완은 물론, 각각의 알고리즘을 single로 사용할 경우 나타나는 단점들을 보완

## 특징 #1

앙상블 기법은 여러 학습 알고리즘을 사용하여 구성 학습 알고리즘만으로 얻을 수 있는 것보다 더 나은 예측 성능을 얻음

## 특징 #2

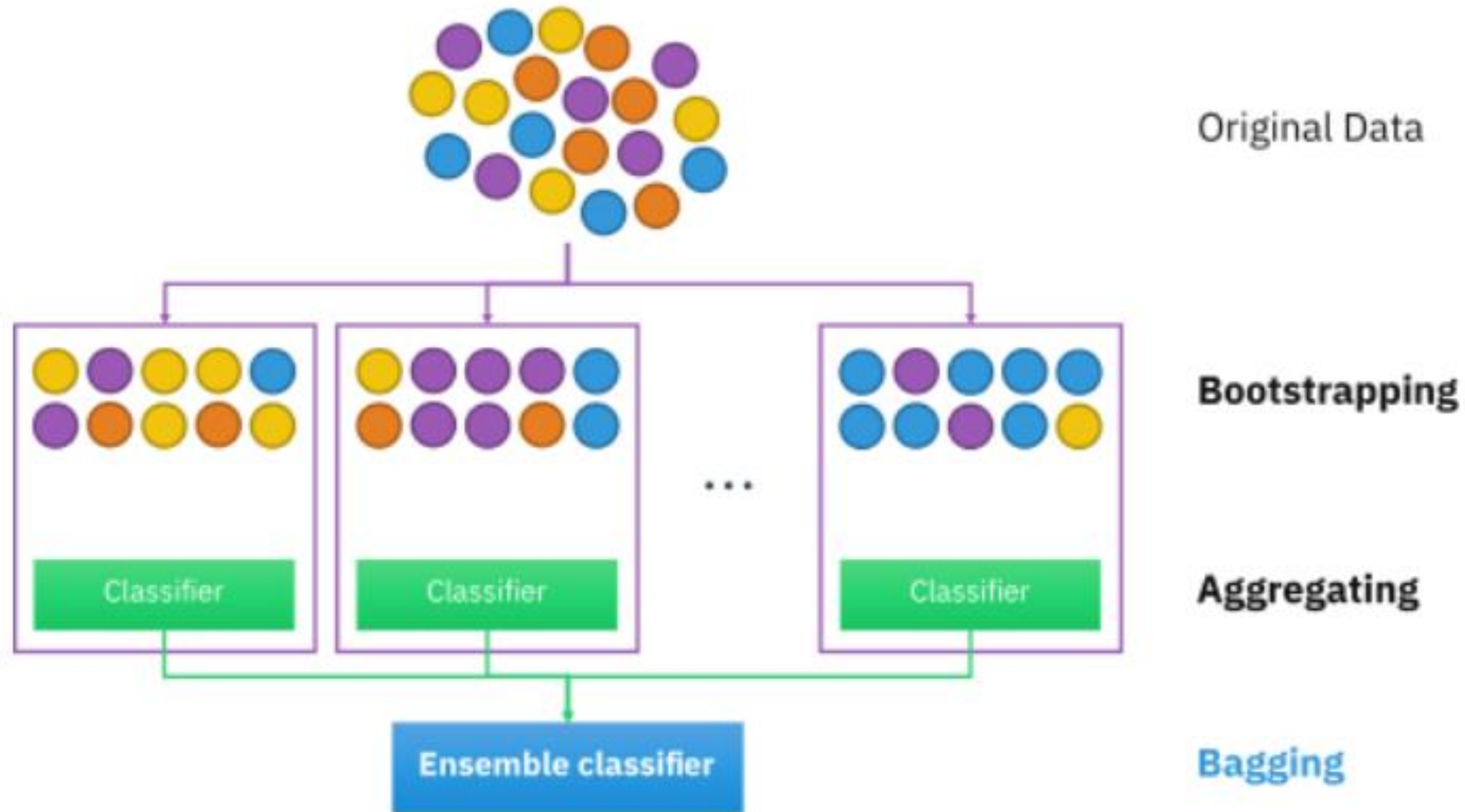
앙상블 기법의 예측력을 평가하기 위해서는 일반적으로 단일 모델의 예측을 평가하는 것보다 더 많은 계산이 필요

=> 앙상블 기법에는 배깅(Bagging) / 부스팅(Boosting) / 스택킹(Stacking) 이 있음

# 1.2 배깅



Bootstrap aggregating의 줄임말로 원 데이터 집합에서 크기가 같은 표본을 여러개 단순 임의 복원추출하여 각 표본에 대한 모델을 만들고 이를 결합하여 최종 모델을 만드는 방법

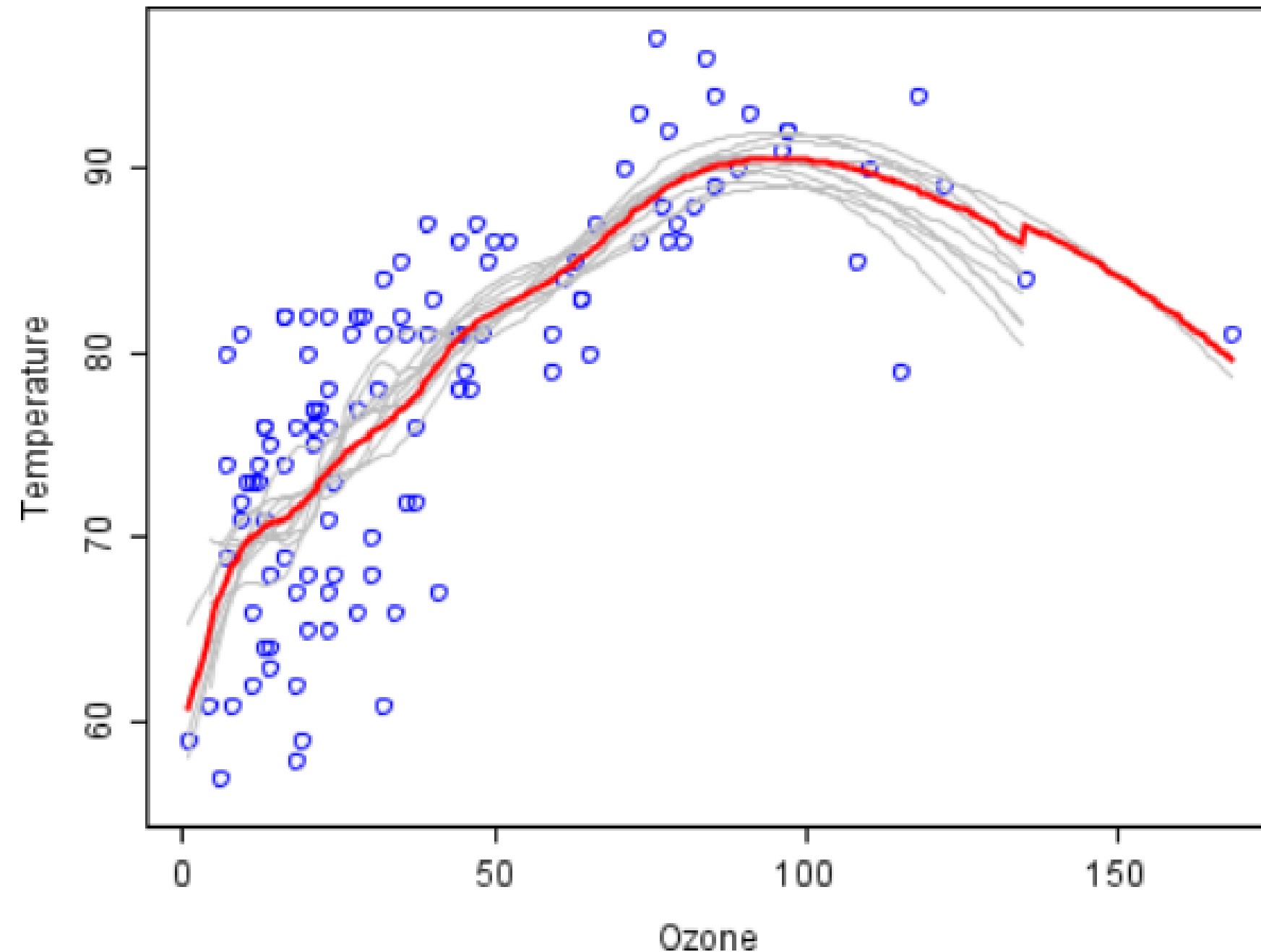


대표적인 알고리즘  
=> 랜덤 포레스트

배깅(Bagging)은 부트스트랩(Bootstrap)을 집계(Aggregating) 하는 것

# 1.3 부트스트랩

✧ random sampling을 적용하는 방법



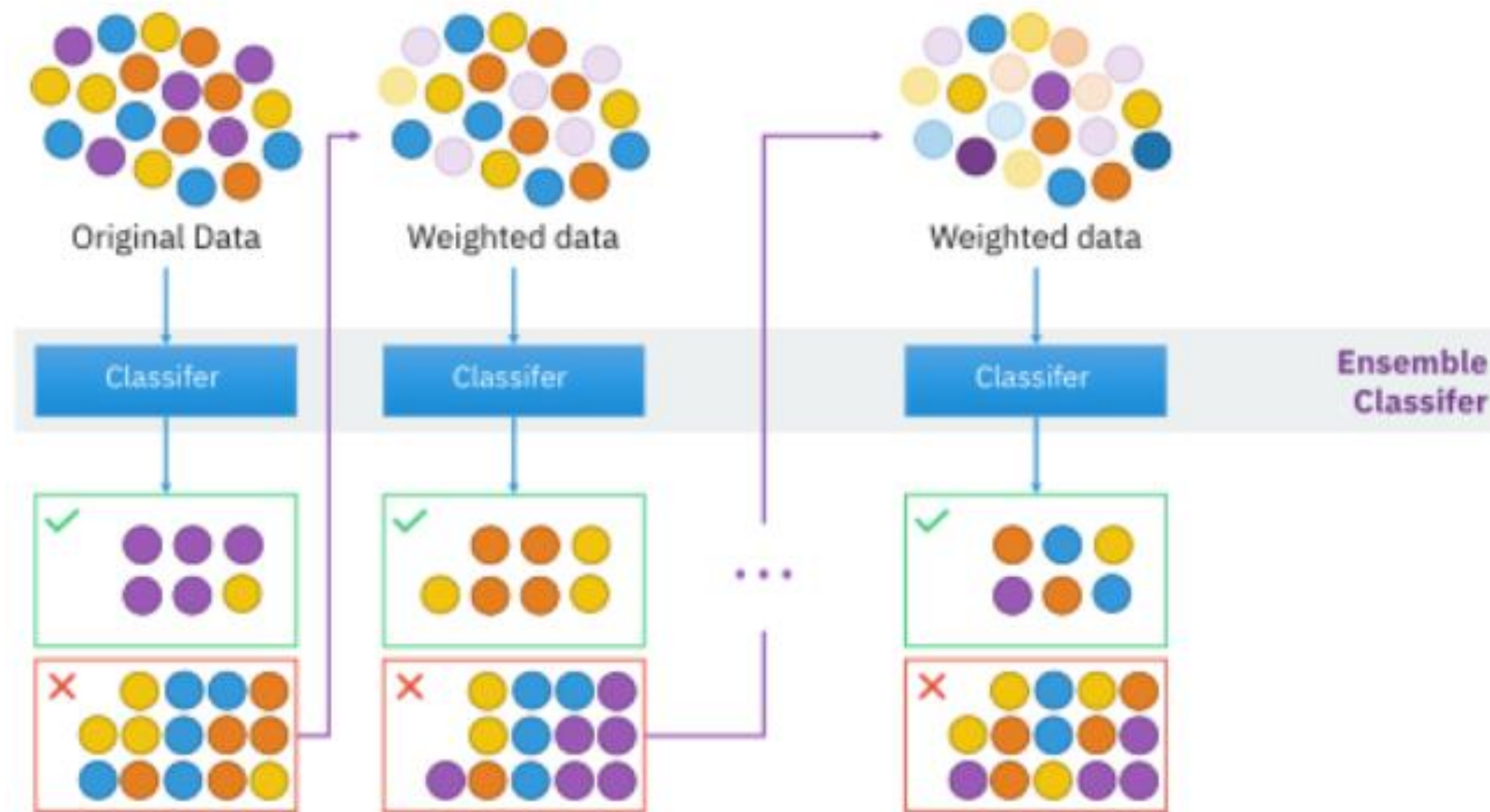
학습 데이터가 충분하지 않더라도  
충분한 학습효과를 주어 높은  
bias의 underfitting 문제나,  
높은 variance로  
인한 **overfitting** 문제를  
**해결**하는데 도움을 줌

회색으로 표시된 각각의 model들을 취합하여 overfit을 줄이는 예. (Wikipedia/Bootstrap\_aggregating)

# 1.4 부스팅



배깅과 유사하나 부트스트랩 표본을 구성하는 Sampling 과정에서 각 자료에 동일한 확률을 부여하는 것이 아니라, 분류가 잘못된 데이터에 **더 큰 가중을 주어** 표본을 추출하는 방법



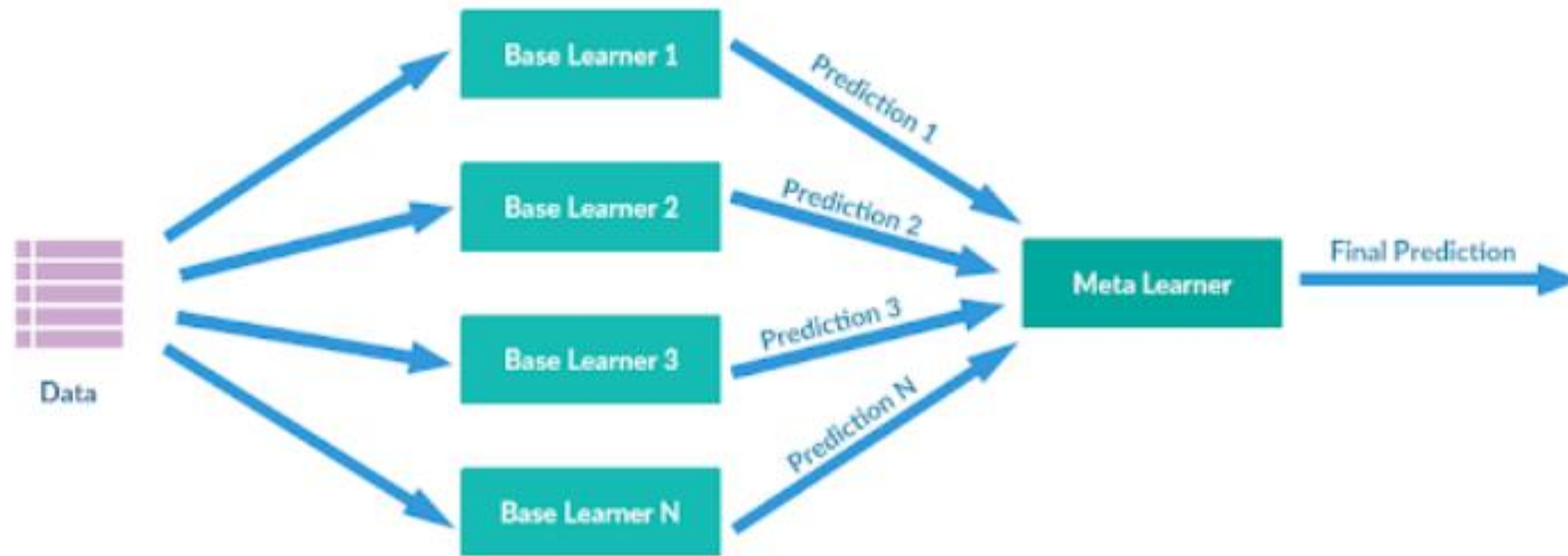
대표적인 알고리즘  
=> XGBoost, AdaBoost,  
GradientBoost

부스팅(Boosting)의 학습 방향. 배깅과는 다르게 순차적으로 진행된다.

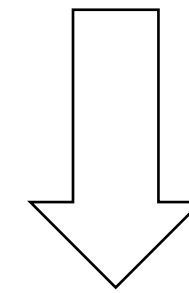


# 1.5 스택킹

✦ 개별 모델이 예측한 데이터를 다시 meta data set으로 사용해서 학습



기본적인 Stacking 방법의 경우 Base Learner들이 동일한 데이터 원본 데이터를 가지고 그대로 학습을 진행했기 때문에 **overfitting 문제**가 발생



크로스 벨리데이션(Cross Validation)

기본적인 Stacking 방법. 개별적인 모델이 예측한 데이터를 다시 training data set으로 활용하여 학습.



# 1.6 배깅 vs 부스팅 vs 스택킹

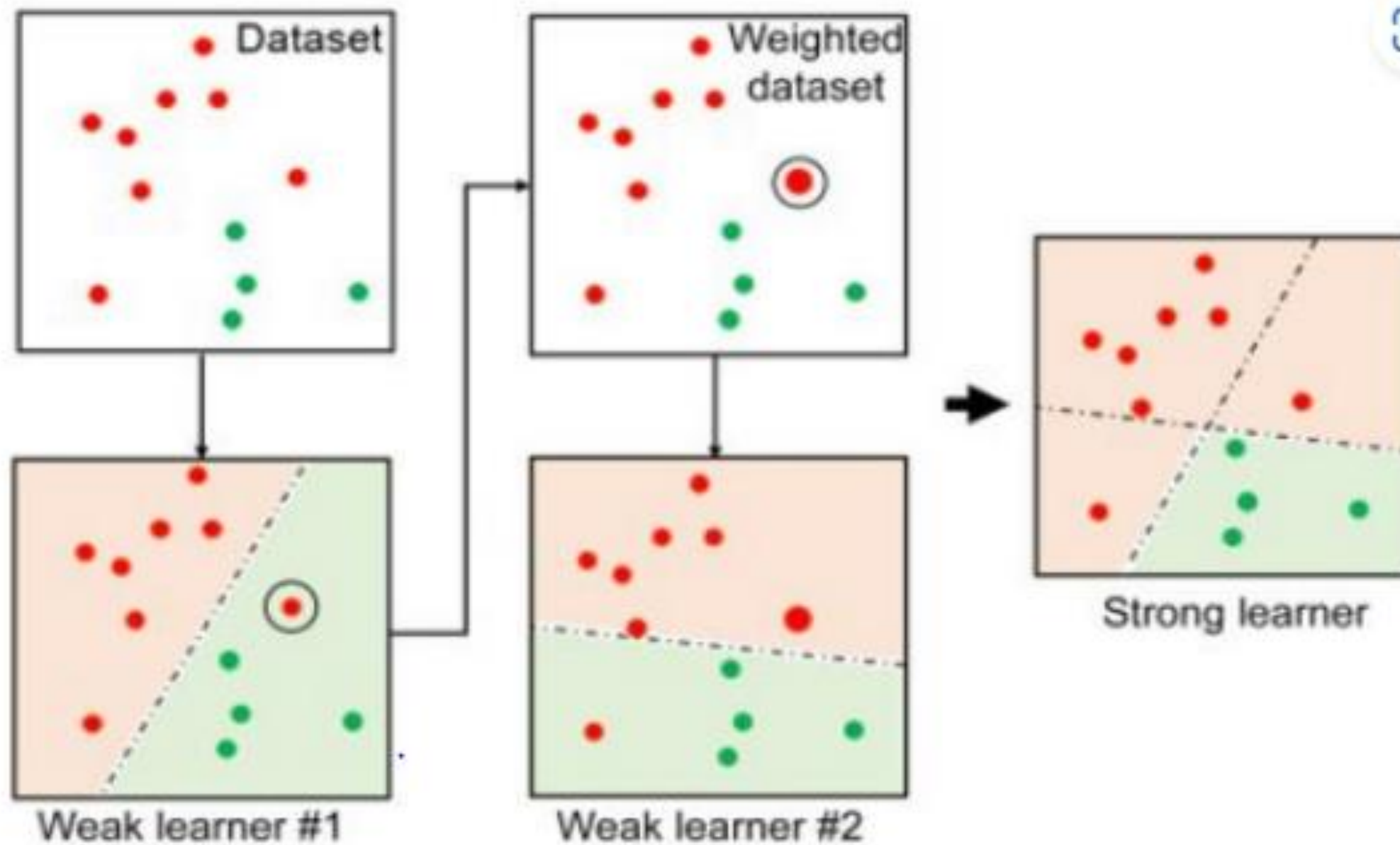
	Bagging	Boosting	Stacking
Partitioning of the data into subsets	Random	Giving <u>mis</u> -classified samples higher preference	Various
Goal to achieve	Minimize variance	Increase predictive force	Both
Methods where this is used	Random subspace	Gradient descent	Blending
Function to combine single models	(Weighted) average	Weighted majority vote	Logistic regression

## 02 AdaBoost vs Gradient Boost



# 2.1 AdaBoost

✦ 오류 데이터에 가중치를 부여하면서 부스팅을 수행



이전의 분류기에 의해 잘못 분류된 것들을 이어지는 약한 학습기들이 수정해줄 수 있다는 점에서 다양한 상황에 적용가능함

따라서 에이다 부스트는 잡음이 많은 데이터와 이상점(outlier)에 취약한 모습을 보임

## 2.1 AdaBoost

가속 분류기

$$F_T(x) = \sum_{t=1}^T f_t(x)$$

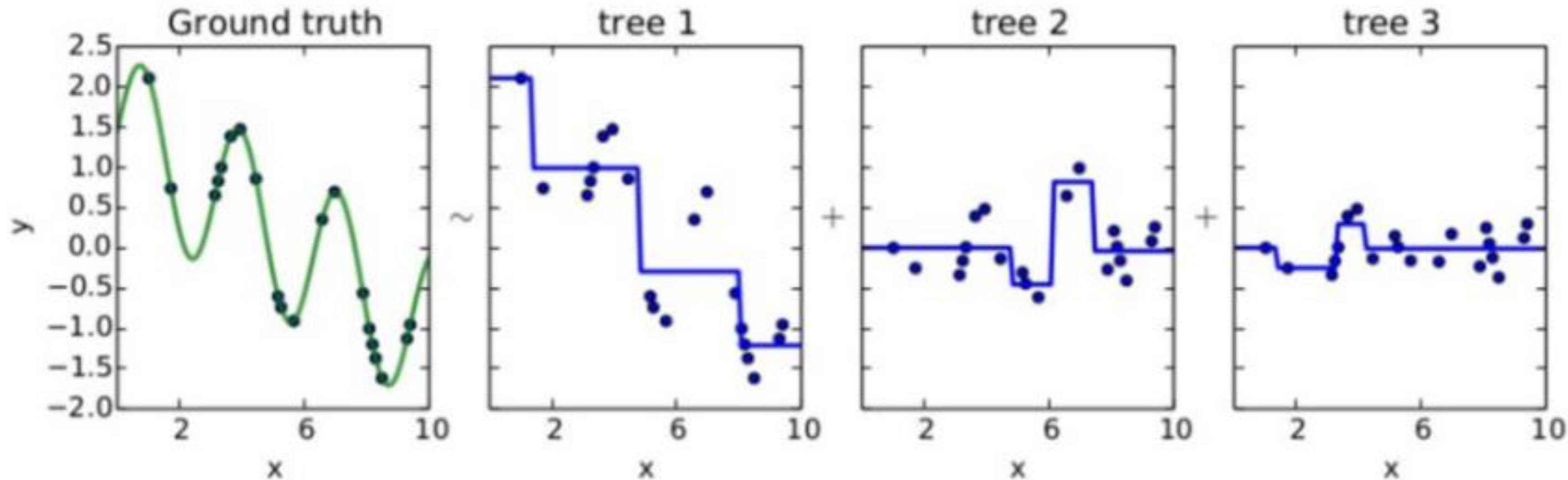
훈련 오류의 합

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

=> 최소가 되도록 하는 계수 알파를 배정 !

## 2.2 Gradient boost

가중치 업데이트를 경사 하강법을 이용함. 오류를 반복적으로 학습시키는 방식으로 오류를 줄여나감



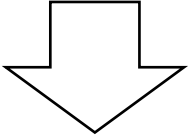
모델 A를 통해  $y$ 를 예측하고 남은 잔차 (residual)을 다시 B라는 모델을 통해 예측하고 A+B 모델을 통해  $y$ 를 예측한다면 A보다 나은 B 모델을 만들 수 있게 됨.

=> 이러한 방법을 계속하면 잔차는 계속해서 줄어들게 되고, training set을 잘 설명하는 예측 모델을 만들 수 있음

## 2.2 Gradient boost

residual은 loss function을 squared error로 설정하였을 때, negative gradient

증명 )    Loss function     $j(y_i, f(x_i)) = \frac{1}{2} (y_i - f(x_i))^2$

 편미분

$$\frac{\partial j(y_i, f(x_i))}{\partial f(x_i)} = \frac{\partial \left[ \frac{1}{2} (y_i - f(x_i))^2 \right]}{\partial f(x_i)} = f(x_i) - y_i$$

residual     $y_i - f(x_i)$

## 03 GBM 하이퍼 파라미터 및 튜닝





# 3.1 GBM 하이퍼 파라미터

**loss** : 경사 하강법에서 사용할 비용 함수 지정

**learning\_rate** : gbm이 학습을 진행할 때마다 적용하는 학습률. Weak learner가 순차적으로 오류 값을 보정해 나가는 데 적용하는 계수. 0~1 사이의 값을 지정할 수 있으며 기본값은 0.1

**n\_estimators** : weak learner의 개수. Weak learner가 순차적으로 오류를 보정하므로 개수가 많을수록 예측 성능이 일정 수준까지는 좋아질 수 있음. 하지만 개수가 많을수록 수행 시간이 오래걸림

**subsample** : weak learner가 학습에 사용하는 데이터의 샘플링 비율. 과적합이 염려되는 경우 subsample을 1보다 작은 값으로 설정

## 3.2 GBM 하이퍼 파라미터 튜닝

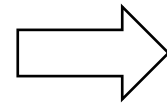
### GridSearchCV 이용

```
>>> gb_clf = GradientBoostingClassifier(random_state=0)
>>> gb_clf.fit(X_train, y_train)
>>> gb_pred = gb_clf.predict(X_test)
>>> gb_accuracy = accuracy_score(y_test, gb_pred)

>>> print('GBM 정확도: {:.4f}'.format(gb_accuracy))
>>> print('GBM 수행 시간: {:.1f} 초'.format(time.time() - start_time))
```

GBM 정확도: 0.9389

GBM 수행 시간: 502.1 초



```
>>> params = {
>>>     'n_estimators': [100, 500],
>>>     'learning_rate': [0.05, 0.1]
>>> }
```

```
>>> grid_cv = GridSearchCV(gb_clf, param_grid=params, cv=2, verbose=1)
>>> grid_cv.fit(X_train, y_train)
>>> print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
>>> print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 8 out of 8 | elapsed: 95.6min finished
```

최적 하이퍼 파라미터:

```
{'learning_rate': 0.1, 'n_estimators': 500}
```

최고 예측 정확도: 0.9011

```
>>> # GridSearchCV를 이용해 최적으로 학습된 estimator로 예측 수행
>>> gb_pred = grid_cv.best_estimator_.predict(X_test)
>>> gb_accuracy = accuracy_score(y_test, gb_pred)
>>> print('GBM 정확도: {:.4f}'.format(gb_accuracy))
```

GBM 정확도: 0.9420

## 3.2 GBM 하이퍼 파라미터 튜닝

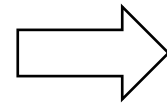
### GridSearchCV 이용

```
>>> gb_clf = GradientBoostingClassifier(random_state=0)
>>> gb_clf.fit(X_train, y_train)
>>> gb_pred = gb_clf.predict(X_test)
>>> gb_accuracy = accuracy_score(y_test, gb_pred)

>>> print('GBM 정확도: {:.4f}'.format(gb_accuracy))
>>> print('GBM 수행 시간: {:.1f} 초'.format(time.time() - start_time))
```

GBM 정확도: 0.9389

GBM 수행 시간: 502.1 초



```
>>> params = {
>>>     'n_estimators': [100, 500],
>>>     'learning_rate': [0.05, 0.1]
>>> }

>>> grid_cv = GridSearchCV(gb_clf, param_grid=params, cv=2, verbose=1)
>>> grid_cv.fit(X_train, y_train)
>>> print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
>>> print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))

Fitting 2 folds for each of 4 candidates, totalling 8 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 8 out of 8 | elapsed: 95.6min finished
최적 하이퍼 파라미터:
{'learning_rate': 0.1, 'n_estimators': 500}
최고 예측 정확도: 0.9411

>>> # GridSearchCV를 이용해 최적으로 학습된 estimator로 예측 수행
>>> gb_pred = grid_cv.best_estimator_.predict(X_test)
>>> gb_accuracy = accuracy_score(y_test, gb_pred)
>>> print('GBM 정확도: {:.4f}'.format(gb_accuracy))
```

GBM 정확도: 0.9420

하이퍼 파라미터를 최적화 한 결과, 정확도가 올라감을 확인

## 04 XGBoost



# 4.1 XGBoost 개요

XGBoost: eXtra Gradient Boost

GBM에 기반해서 단점을 보완한, 분류에 있어서 뛰어난 예측 성능을 나타내는 알고리즘

XGBoost의 주요 장점

- ① 분류와 회귀 영역에서 뛰어난 예측 성능
- ② 병렬 수행 등 다양한 기능으로 GBM에 비해 빠른 수행 속도
- ③ 과적합 규제 기능 (Regularization)
- ④ 나무 가지치기 (Tree pruning): 긍정 이득이 없는 분할을 줄인다
- ⑤ 최적화된 반복 수행 횟수와 최적화 되면 반복을 중단하는 조기 중단 기능
- ⑥ 결측치 자체 처리

# 4.1 XGBoost 개요

XGBoost의 파이썬 패키지명: xgboost

```
import xgboost
print(xgboost.__version__)
```

XGBoost 패키지의 변화

파이썬 래퍼 XGBoost	★ 사이킷런 래퍼 XGBoost
<ul style="list-style-type: none"><li>- 사이킷런과 호환되지 않는 독자적인 패키지</li><li>- 고유의 API와 하이퍼 파라미터 사용</li></ul>	<ul style="list-style-type: none"><li>- 사이킷런과 연동되어 표준 사이킷런 개발 프로세스 및 유틸리티 사용 가능</li><li>- 클래스: XGBClassifier XGBRegressor</li></ul>

## 4.2 파이썬 래퍼 XGBoost 하이퍼 파라미터

### 주요 일반 파라미터

✓ GBM와 유사한 하이퍼 파라미터에 조기 중단, 과적합을 규제하는 파라미터가 추가되었으나 파이썬 래퍼 XGBoost 모듈과 사이킷런 래퍼 XGBoost 모듈은 사이킷런 파라미터의 범용화된 이름 규칙에 따라 파라미터 명이 달라진다.

- ① booster: gbtree(default) or gblinear
- ② silent: 0이면 출력 메시지 표시(default), 원하지 않으면 1
- ③ nthread: CPU의 실행 스레드 개수 조정, 디폴트는 전체 스레드 사용



## 4.2 파이썬 래퍼 XGBoost 하이퍼 파라미터

### 주요 부스터 파라미터

- ① eta(default=0.1) : GBM의 learning rate와 같은 파라미터
- ② num\_boost\_rounds: GBM의 n\_estimators와 같은 파라미터
- ③ min\_child\_weight(default=1) : 분할을 결정하는 데이터들의 weight 총합
- ④ gamma(default=0): 트리의 리프 노드를 추가적으로 나눌지 결정하는 최소 손실 감소 값
- ⑤ max\_depth(default=6): 트리 기반 알고리즘의 max\_depth과 같은 파라미터
- ⑥ sub\_sample(default=1): GBM의 subsample 와 같은 파라미터

## 4.2 파이썬 래퍼 XGBoost 하이퍼 파라미터

### 주요 부스터 파라미터

- ① `colsample_bytree(default=1)`: GBM의 `max_features`
- ② `lambda(default=1)`: L2 Regulation 적용 값
- ③ `alpha(default=0)`: L1 Regulation 적용 값
- ④ `scale_pos_weight(default=1)`: 비대칭한 클래스로 구성된 데이터 세트의 균형을 위한 파라미터

# 4.2 파이썬 래퍼 XGBoost 하이퍼 파라미터

## 학습 태스크 파라미터

① objective: 최소값을 가져야할 손실 함수를 정의

binary:logistic: 이진 분류

multi:softmax: 다중 분류

multi:softprob: multi:softmax와 유사하나 개별 레이블 클래스에 해당되는 예측 확률 반환

② eval\_metric: 검증에 사용되는 함수

유형:rmse, mae, logloss, error, merror, mlogloss, auc

# 4.3 파이썬 래퍼 XGBoost 적용 – 위스콘신 유방암 예측

## 위스콘신 유방암 예측

### # 데이터, 패키지 불러오기, 데이터 split

```
import xgboost as xgb
from xgboost import plot_importance
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

# 데이터 불러오기
dataset = load_breast_cancer()
X_features = dataset.data
y_label = dataset.target

cancer_df = pd.DataFrame(data=X_features, columns=dataset.feature_names)
cancer_df['target'] = y_label

# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test = train_test_split(X_features, y_label,
                                                    test_size=0.2, random_state=156)
```

### # DMatrix

```
dtrain = xgb.DMatrix(data=X_train, label=y_train)
dtest = xgb.DMatrix(data=X_test, label=y_test)
```

### # 하이퍼 파라미터 설정

```
params = { 'max_depth':3, # 트리 최대 깊이는 3
           'eta': 0.1, # 학습률은 0.1
           'objective':'binary:logistic', # 데이터를 이진 분류하므로 이진 로지스틱
           'eval_metric':'logloss' # 오류 함수 평가 성능 지표
         }
num_rounds = 400
```

```
# train 데이터 셋은 'train', evaluation(test) 데이터 셋은 'eval' 로 명기합니다.
wlist = [(dtrain,'train'),(dtest,'eval')]
# 하이퍼 파라미터와 early stopping 파라미터를 train() 함수의 파라미터로 전달
xgb_model = xgb.train(params = params, dtrain=dtrain, num_boost_round=num_rounds,
                      early_stopping_rounds=100, evals=wlist)
```

### # 예측

```
pred_probs = xgb_model.predict(dtest)
print('predict() 수행 결과값을 10개만 표시, 예측 확률 값으로 표시됨')
print(np.round(pred_probs[:10],3))

# 예측 확률이 0.5 보다 크면 1,
# 그렇지 않으면 0 으로 예측값 결정하여 List 객체인 preds에 저장
preds = [ 1 if x > 0.5 else 0 for x in pred_probs ]
print('예측값 10개만 표시:',preds[:10])
```

```
predict() 수행 결과값을 10개만 표시, 예측 확률 값으로 표시됨
[0.934 0.003 0.91  0.094 0.993 1.    1.    0.999 0.997 0.    ]
예측값 10개만 표시: [1, 0, 1, 0, 1, 1, 1, 1, 1, 0]
```



# 4.3 파이썬 래퍼 XGBoost 적용 – 위스콘신 유방암 예측

## 예측 성능 평가

### ① get\_clf\_eval

```
get_clf_eval(y_test , preds, pred_probs)
```

오차 행렬

```
[[35  2]  
 [ 1 76]]
```

정확도: 0.9737,

정밀도: 0.9744,

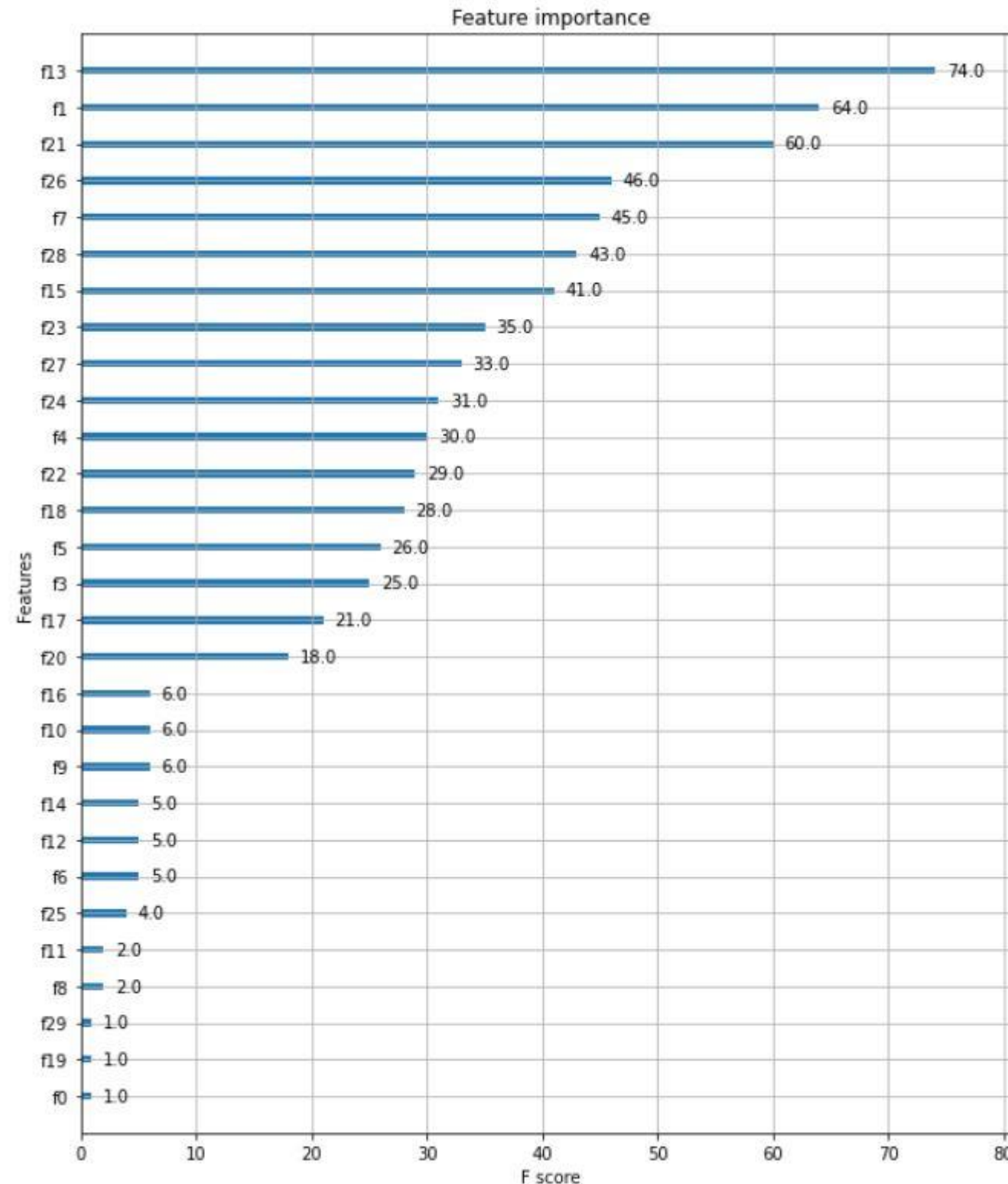
재현율: 0.9870,

F1: 0.9806,

AUC:0.9951

### ② 시각화

```
import matplotlib.pyplot as plt  
%matplotlib inline  
  
fig, ax = plt.subplots(figsize=(10, 12))  
plot_importance(xgb_model, ax=ax)
```



# 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

하이퍼 파라미터의 변경

파이썬 래퍼 XGBoost	사이킷런 래퍼 XGBoost
eta	learning_rate
sub_sample	subsample
lambda	reg_lambda
alpha	reg_alpha
num_boost_round	n_estimators

서로 동일한 파라미터지만 동시에 사용되면  
파이썬 래퍼에서는 num\_boost\_round, 사이킷런 래퍼에서는 n\_estimators

## 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

하이퍼 파라미터 비교, 위스콘신 유방암 예측

- 파이썬 래퍼

```
num_rounds = 400

params = { 'max_depth':3,
          'eta': 0.1,
          'objective':'binary:logistic',
          'eval_metric':'logloss'
        }
```

- 사이킷런 래퍼

```
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400,
                           max_depth=3,
                           learning_rate=0.1)

xgb_wrapper.fit(X_train, y_train)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[: , 1]
```



## 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

### 사이킷런 래퍼 XGBoost

#### - 사이킷런 래퍼

```
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400,
                             max_depth=3,
                             learning_rate=0.1)

xgb_wrapper.fit(X_train, y_train)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[:, 1]
```

사이킷런의 기본 Estimator를 그대로 상속해 만들었기 때문에 fit(), predict()만으로 학습과 예측이 가능하고, GridSearchCV, Pipeline 등 사이킷런의 유틸리티를 사용할 수 있다.

## 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

### 예측 성능 평가

```
get_clf_eval(y_test , w_preds, w_pred_proba)
```

오차 행렬

```
[[35  2]
```

```
 [ 1 76]]
```

정확도: 0.9737, 정밀도: 0.9744, 재현율: 0.9870, F1: 0.9806, AUC:0.9951

- ✓ 파이썬 래퍼와 동일한 평가 결과를 가짐

## 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

### 조기 중단 관련 파라미터

- ✓ 사이킷런 래퍼 XGBoost에서도 fit()에 조기 중단 관련 파라미터를 입력해서 조기 중단을 수행할 수 있다
  - early\_stopping\_rounds: 평가 지표가 향상될 수 있는 반복 횟수 정의
  - eval\_metric: 조기 중단을 위한 평가 지표
  - eval\_set: 성능 평가를 수행할 데이터 세트 – 학습 데이터가 아닌 별도의 데이터 세트여야 한다

## 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

조기 중단 예제 (eval\_metric = logloss, eval\_set = [(X\_test, Y\_test)])

- early\_stopping\_rounds = 100

```
xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
evals = [(X_test, y_test)]
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="logloss",
                eval_set=evals, verbose=True)

ws100_preds = xgb_wrapper.predict(X_test)
ws100_pred_proba = xgb_wrapper.predict_proba(X_test)[:, 1]
```

```
get_clf_eval(y_test, ws100_preds, ws100_pred_proba)
```

오차 행렬

```
[[34  3]
```

```
 [ 1 76]]
```

정확도: 0.9649, 정밀도: 0.9620, 재현율: 0.9870, F1: 0.9744, AUC:0.9954

```
[309] validation_0-logloss:0.085877
[310] validation_0-logloss:0.085923
[311] validation_0-logloss:0.085948
Stopping. Best iteration:
[211] validation_0-logloss:0.085593
```

✓ n\_estimators = 400이지만  
311번 반복한 후 학습을 완료

✓ get\_clf\_eval()을 통해 조기 중단이 적용되지 않은 결과와 큰 차이가 없다는 것을 알 수 있다.

## 4.4 사이킷런 래퍼 XGBoost의 개요 및 적용

조기 중단 예제 (eval\_metric = logloss, eval\_set = [(X\_test, Y\_test)])

- early\_stopping\_rounds = 10

- ✓ 조기 중단값을 너무 급격하게 줄이면 성능이 향상될 가능성이 있음에도 10번 반복하는 동안 성능 평가 지표가 향상되지 않으면 반복이 멈춰버려서 충분한 학습이 되지 않아 예측 성능이 저하될 수 있다

```
xgb_wrapper.fit(X_train, y_train, early_stopping_rounds=10,  
                eval_metric="logloss", eval_set=evals, verbose=True)  
  
ws10_preds = xgb_wrapper.predict(X_test)  
ws10_pred_proba = xgb_wrapper.predict_proba(X_test)[: , 1]  
get_clf_eval(y_test , ws10_preds, ws10_pred_proba)
```

오차 행렬

```
[[34  3]  
 [ 2 75]]
```

정확도: 0.9561, 정밀도: 0.9615, 재현율: 0.9740, F1: 0.9677, AUC:0.9947

```
[60]    validation_0-logloss:0.091939  
[61]    validation_0-logloss:0.091461  
[62]    validation_0-logloss:0.090311  
Stopping. Best iteration:  
[52]    validation_0-logloss:0.089577
```

- ✓ early\_stopping\_rounds = 100일 때보다 정확도가 낮다.

# 05 LightGBM





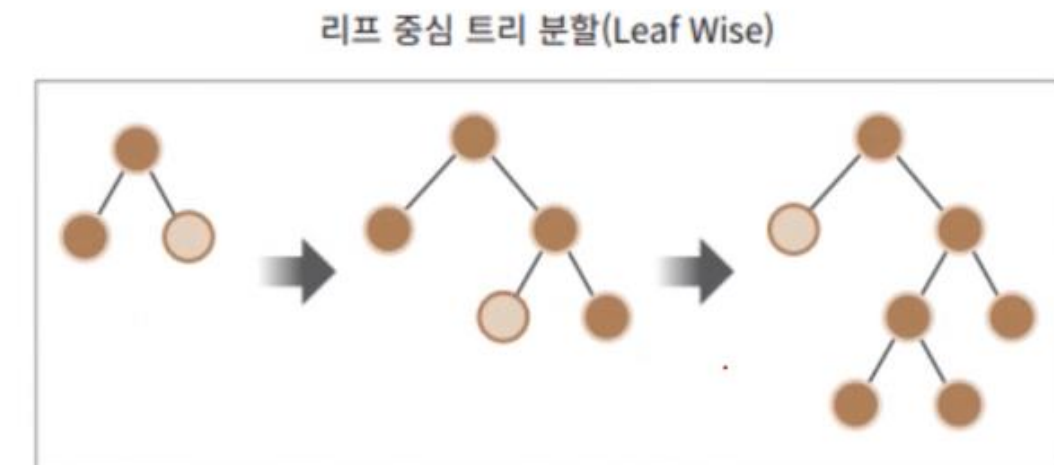
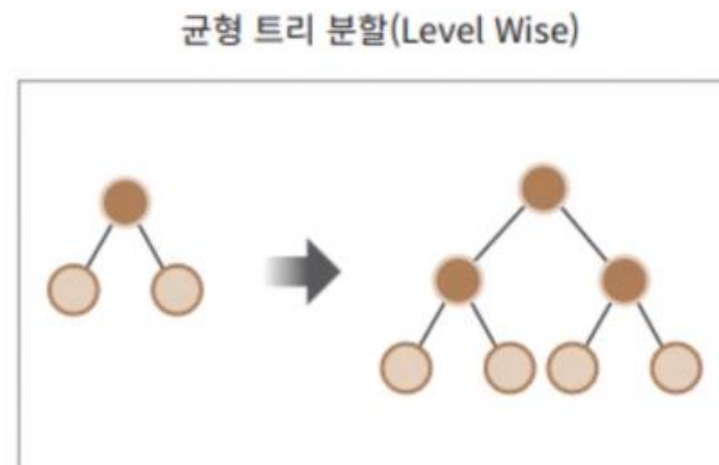
# 5.1 LGBM 개요 및 설치

## LGBM의 XGBoost 대비 장점

- ① 빠른 학습과 예측 수행 시간, 더 작은 메모리 사용량
- ② XGBoost와 예측 성능은 비슷하지만 기능상으로는 더 다양하다
- ③ 카테고리형 피처의 자동 변환과 최적 분할
- ④ 리프 중심 트리 분할(Leaf Wise): 일반적인 트리 분할 방법처럼 트리의 균형을 맞추는 것 대신

최대 손실 값(max delta loss)을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고

비대칭적인 규칙 트리가 생성





# 5.2 LGBM 하이퍼 파라미터

## 주요 파라미터

- ① num\_iterations(default=100): 반복 수행하려는 트리의 개수를 지정.
- ② learning\_rate(default=0.1): 부스팅 스텝을 반복적으로 수행할 때 업데이트되는 학습률 값이다.
- ③ max\_depth(default=-1): 트리 기반 알고리즘의 max\_depth
- ④ min\_data\_in\_leaf(default=20): 결정 트리의 min\_samples\_leaf와 같은 파라미터
- ⑤ num\_leaves(default=31): 하나의 트리가 가질 수 있는 최대 리프 개수
- ⑥ boosting(default=gbdt): 부스팅의 트리를 생성하는 알고리즘을 기술
  - gbdt : 일반적인 그래디언트 부스팅 결정 트리
  - rf : 랜덤 포레스트

# 5.2 LGBM 하이퍼 파라미터

## 주요 파라미터

- ① `bagging_fraction(default=1.0)`: 과적합을 제어하기 위해서 데이터를 샘플링하는 비율
- ② `feature_fraction(default=1.0)`: 개별 트리를 학습할 때마다 무작위로 선택하는 feature의 비율
- ③ `lambda_l2(default=0.0)`: L2 regulation 제어를 위한 값
- ④ `lambda_l1(default=0.0)`: L1 regulation 제어를 위한 값

## Learning Task 파라미터

- ① `objective`: 최솟값을 가져야 할 손실함수를 정의. 애플리케이션 유형, 즉 회귀, 다중 클래스 분류, 이진 분류인지에 따라서 `objective`인 손실 함수가 지정

## 5.2 LGBM 하이퍼 파라미터

### 하이퍼 파라미터 튜닝 방안

- ① num\_leaves의 개수를 높이면 정확도가 높아지지만, 반대로 트리의 깊이가 깊어지고 모델이 복잡도가 커져 과적합 영향도가 커진다.
- ② min\_data\_in\_leaf는 num\_leaves와 학습 데이터의 크기에 따라 달라지지만, 보통 큰 값으로 설정하면 트리가 깊어지는 것을 방지한다.
- ③ max\_depth는 깊이의 크기를 제한한다.

## 5.3 파이썬, 사이킷런 래퍼 LightGBM, 사이킷런 래퍼 XGBoost 하이퍼 파라미터 비교

- ✓ 사이킷런 래퍼 LightGBM의 하이퍼 파라미터는 사이킷런 XGBoost에 맞춰서 변경하여 많은 하이퍼 파라미터가 똑같다.

유형	파이썬 래퍼 LightGBM	사이킷런 래퍼 LightGBM	사이킷런 래퍼 XGBoost
파라미터명	num_iterations	n_estimators	n_estimators
	learning_rate	learning_rate	learning_rate
	max_depth	max_depth	max_depth
	min_data_in_leaf	min_child_samples	N/A
	bagging_fraction	subsample	subsample
	feature_fraction	colsample_bytree	colsample_bytree
	lambda_l2	reg_lambda	reg_lambda
	lambda_l1	reg_alpha	reg_alpha
	early_stopping_round	early_stopping_rounds	early_stopping_rounds
	num_leaves	num_leaves	N/A
	min_sum_hessian_in_leaf	min_child_weight	min_child_weight

## 5.4 LGBM 적용 – 위스콘신 유방암 예측

### 위스콘신 유방암 예측

# 데이터, 패키지 불러오기, 데이터 split

```
import lightgbm

from lightgbm import LGBMClassifier

import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

dataset = load_breast_cancer()
ftr = dataset.data
target = dataset.target

# 전체 데이터 중 80%는 학습용 데이터, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test=train_test_split(ftr, target,
                                                  test_size=0.2, random_state=156 )

# 앞서 XGBoost와 동일하게 n_estimators는 400 설정.
lgbm_wrapper = LGBMClassifier(n_estimators=400)

# LightGBM도 XGBoost와 동일하게 조기 중단 수행 가능.
evals = [(X_test, y_test)]
lgbm_wrapper.fit(X_train, y_train, early_stopping_rounds=100, eval_metric="logloss",
                eval_set=evals, verbose=True)

preds = lgbm_wrapper.predict(X_test)
pred_proba = lgbm_wrapper.predict_proba(X_test)[:, 1]
```

### # 예측 성능 평가

```
get_clf_eval(y_test, preds, pred_proba)
```

```
오차 행렬
[[33  4]
 [ 1 76]]
정확도: 0.9561,
정밀도: 0.9500,
재현율: 0.9870,
      F1: 0.9682,
AUC:0.9905
```

- ✓ XGBoost를 사용했을 때(96.49%)보다 정확도가 작지만, 데이터 세트의 크기가 작아서 알고리즘 간 성능 비교는 큰 의미가 없다.

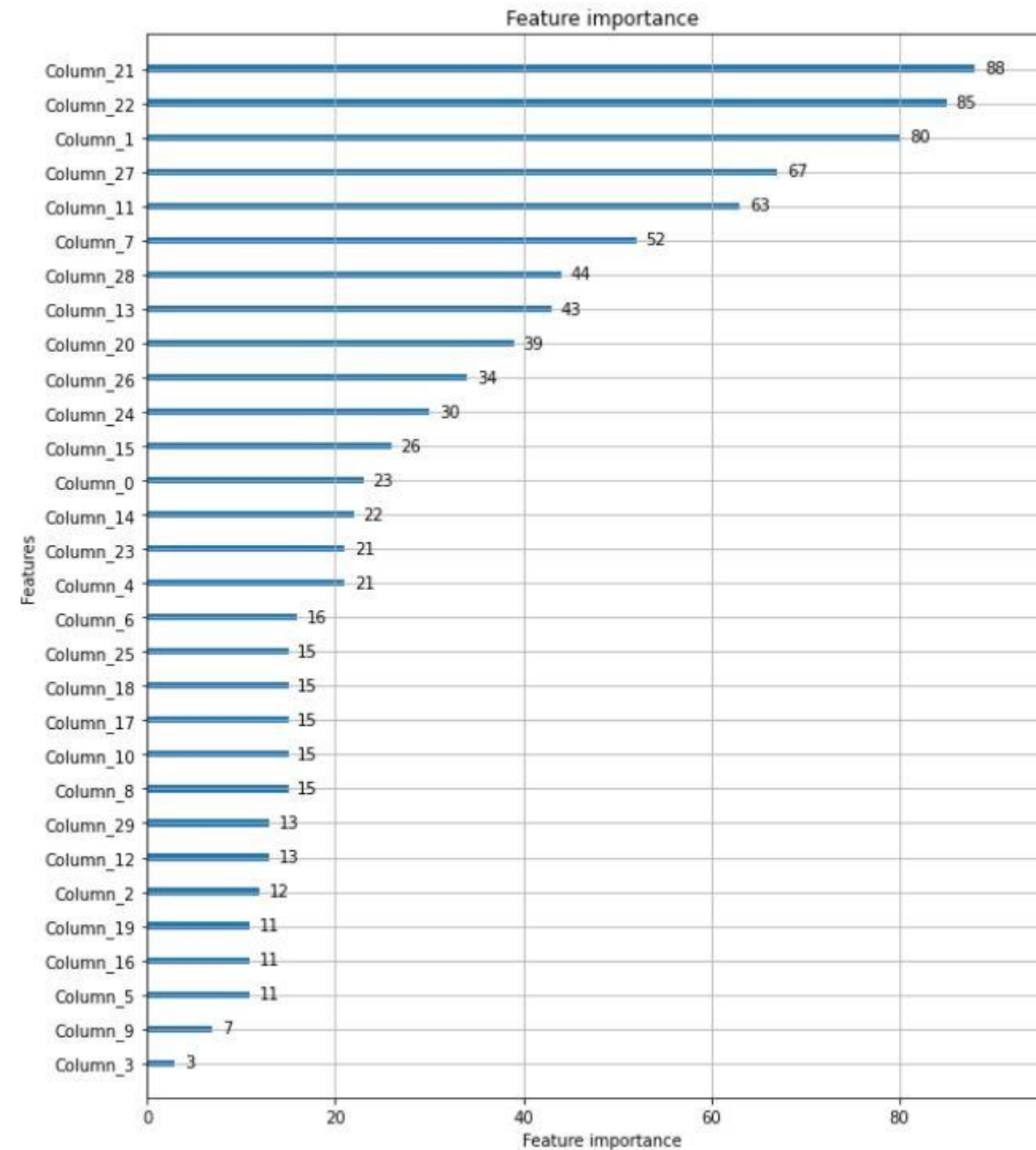
## 5.4 LGBM 적용 – 위스콘신 유방암 예측

위스콘신 유방암 예측

# 예측 성능 평가 - 시각화

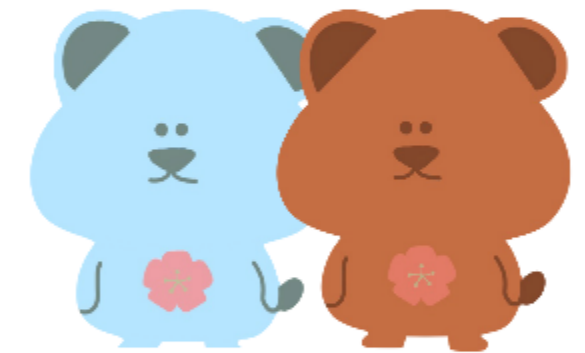
```
from lightgbm import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(lgbm_wrapper, ax=ax)
```





# 06 CatBoost



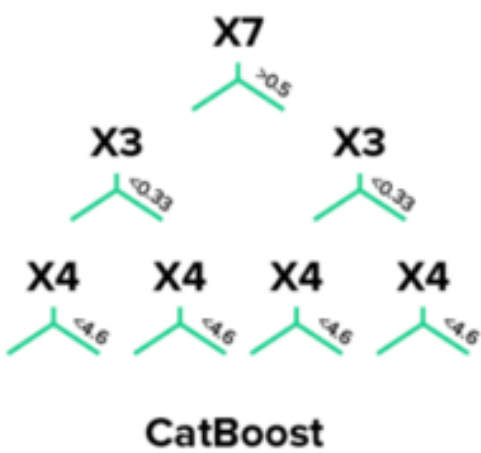
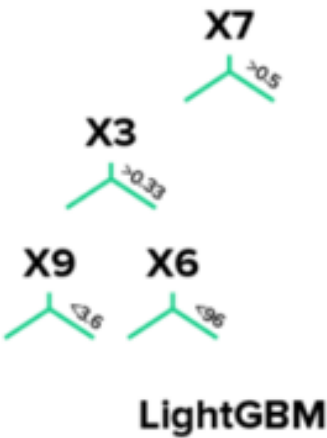
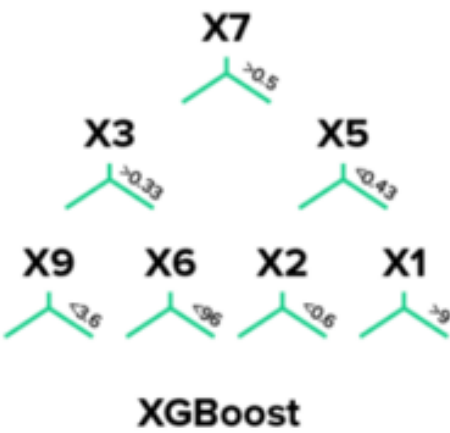
# 6.1 부스팅 모델 비교



- 병렬 학습 가능 -> 속도 개선
- Level-wise
- overfitting 규제 기능
- 결측치 처리 성능
- Cross Validation 기능 추가

- Leaf-wise
- Gradient-based One-Side Sampling : Information Gain이 적은 가지의 데이터를 증폭시켜 오답에 초점
- 범주형 변수의 처리, 최적 분할
- 학습 속도 및 성능 개선
- 데이터가 적을 경우 overfitting 위험

- 범주형 변수 처리 개선
- overfitting 개선
- Level-wise



# 6.2 CatBoost

## CatBoost (Categorical Boosting)

: Categorical feature를 더 잘 처리하고, overfitting 개선한 알고리즘

### 1) 장점

- 트리 기반 모델에서의 범주형 변수 처리 문제 개선
- Target leakage로 인한 overfitting 방지

### 2) 한계

- 수치형 변수가 많은 경우 속도가 느리다 : 범주형 변수가 많을 때 적합
- 결측치가 많거나, sparse matrix인 경우 잘 처리하지 못함

# 6.2 CatBoost

---

## 기존 알고리즘 한계

### Overfitting

1. Categorical feature 처리시
2. Gradient boosting 과정에서

“이전의 알고리즘에 존재하는 overfitting 문제를 해결하고 있다.”

# 6.3 범주형 변수 처리

## Categorical feature 처리 방법

- 1. One-hot encoding
  - level의 개수가 많은 경우 변수가 급격히 증가하는 단점이 있다.
- 2. Target Statistics (TS)
  - 같은 범주에 속하는 데이터들의 target 값의 평균값으로 인코딩
  - Target Encoding, Mean Encoding, Response Encoding 등으로 부른다.

time	feature 1	class_labels (max_temptarure on that day)
sunday	sunny	35
monday	sunny	32
tues	cloudy	15
wed	cloudy	14
thurs	mostly_cloudy	10
fri	cloudy	20
sat	cloudy	25

'cloudy' ->  $(15+14+20+25) / 4 = 18.5$

- Target leakage 로 인한 overfitting 문제
- Holdout TS, Leave-one-out TS, ordered TS 등의 개선

# 6.3 범주형 변수 처리

## Ordered target encoding

- k번째 데이터를 인코딩할 때, (k-1)번까지의 데이터의 target 값만 사용해서 평균값 계산  
→ 현재 데이터의 target 값 사용하지 않음 : Target leakage 방지
- Random permutation을 통해 순서에 다양성 부여

time	feature 1	class_labels (max_temptarure on that day)
sunday	sunny	35
monday	sunny	32
tues	couldy	15
wed	cloudy	14
thurs	mostly_cloudy	10
fri	cloudy	20
sat	cloudy	25

Friday : 5번째 row까지만 사용  
'Cloudy' ->  $(15+14) / 2 = 15.5$

Saturday : 6번째 row까지만 사용  
'Cloudy' ->  $(15+14+20) / 3 = 16.3$



# 6.4 Ordered Boosting

## 기존의 Gradient boosting 알고리즘

: 각 스텝마다 모든 학습 데이터를 사용하여 잔차 계산

-> 이전 스텝에서 사용한 데이터로 다시 계산하기 때문에 overfitting 문제

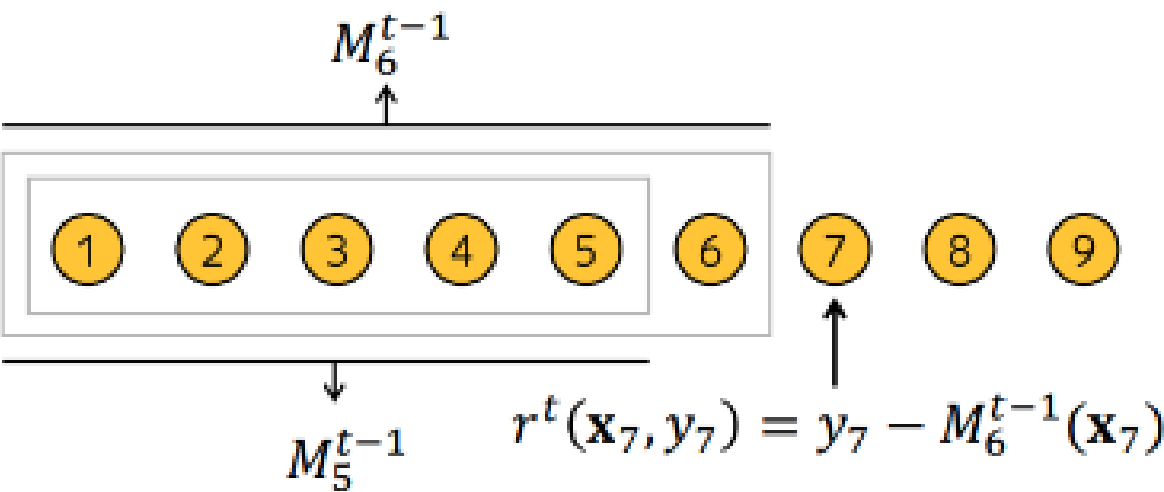
1.  $x_1 \sim x_{10}$ 에 대한 잔차를 가지고 모델을 만든다.
2.  $x_1 \sim x_{10}$ 에 대한 잔차를 가지고 모델을 만든다.
3. 위 과정을 반복

time	datapoint	class label
12:00	x1	10
12:01	x2	12
12:02	x3	9
12:03	x4	4
12:04	x5	52
12:05	x6	22
12:06	x7	33
12:07	x8	34
12:08	x9	32
12:09	x10	12

# 6.4 Ordered Boosting

## Ordered boosting

: 각 스텝마다 일부 데이터만 가지고 잔차 계산



- 1. 먼저 x1의 잔차만 계산하고, 이를 기반으로 모델을 생성한다.
- 2. x1, x2의 잔차를 가지고 모델을 만들고, x3, x4의 잔차를 모델로 예측한다.
- 3. x1~x4를 가지고 모델을 만들고, x5~x8의 잔차를 모델로 예측한다.
- 4. 위 과정을 반복

time	datapoint	class label
12:00	x1	10
12:01	x2	12
12:02	x3	9
12:03	x4	4
12:04	x5	52
12:05	x6	22
12:06	x7	33
12:07	x8	34
12:08	x9	32
12:09	x10	12

# 6.5 그 외

## 1. Feature combination

Information gain이 비슷한 두 categorical feature는 하나의 변수로 통합

country	hair color	class_label
India	black	1
India	black	1
India	black	1
india	black	1
russia	white	0
russia	white	0
russia	white	0
russia	white	0

## 2. One-hot encoding

level이 많지 않은 범주형 변수는 one-hot encoding (‘one\_hot\_max\_size’ 파라미터)

## 3. Hyperparameter tuning

CatBoost는 내부적인 알고리즘으로 overfitting, sampling 다양성 등의 문제 개선  
-> 하이퍼 파라미터에 따라 큰 영향을 받지 않는다.

# 6.6 CatBoost 적용 – 위스콘신 유방암 예측

## 1. CatBoost 설치

```
!pip install catboost
```

## 2. 위스콘신 유방암 데이터 로드

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

dataset = load_breast_cancer()
X = dataset.data
y = dataset.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
```

## 3. Pool 함수 -> categorical feature 처리

```
from catboost import Pool

pool_train = Pool(data = X_train, label = y_train)
pool_test = Pool(data = X_test, label = y_test)
```

Pool(data, label, **cat\_features**=[ 'col1', 'col2' ])

# 6.6 CatBoost 적용 – 위스콘신 유방암 예측

## 4. CatBoost 모델 학습, 예측

```
from catboost import CatBoostClassifier

cb_clf = CatBoostClassifier(iterations=400, learning_rate=0.1, max_depth=3)
cb_clf.fit(pool_train)
pred = cb_clf.predict(pool_test)
pred_prob = cb_clf.predict_proba(pool_test)[: , 1]
```

CatBoost accuracy : 0.9737

# 07 Stacking





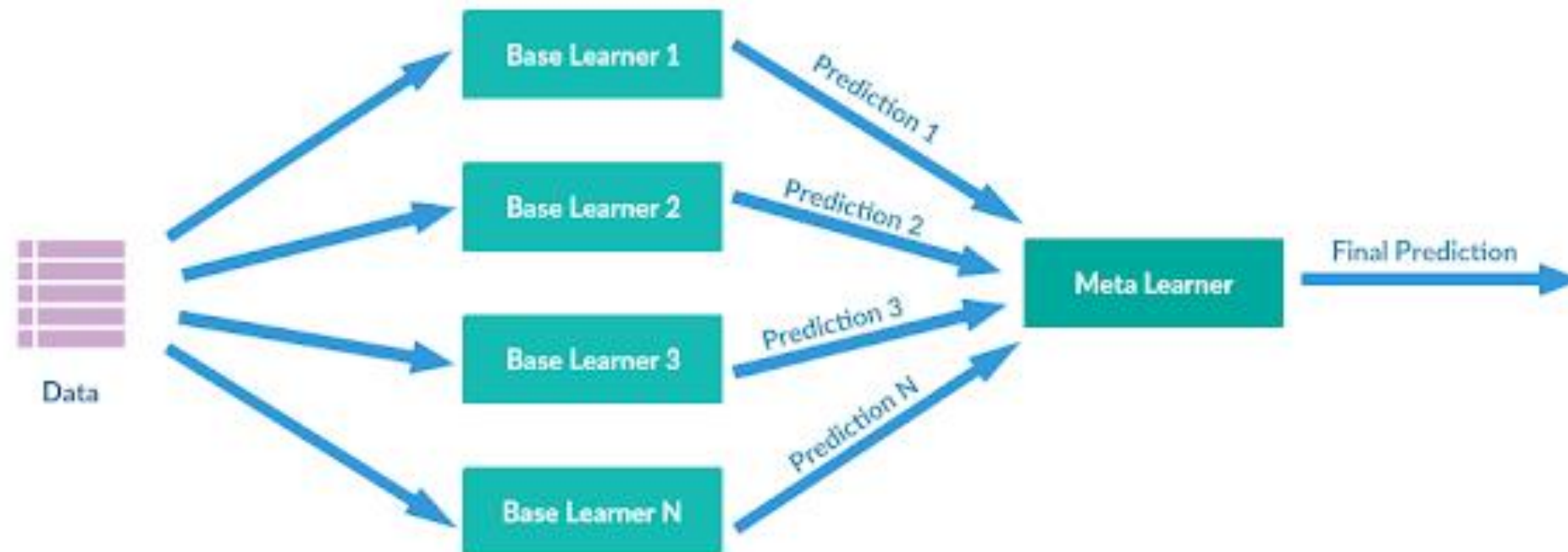
# 7.1 Stacking

## Stacking 앙상블이란?

개별 모델로 예측한 결과를 기반으로 다시 최종 예측을 수행하는 방식

## 다른 앙상블과의 차이점

여러 개별 모델들의 예측 결과를 **학습**해서 최종 결과를 도출하는 meta 모델

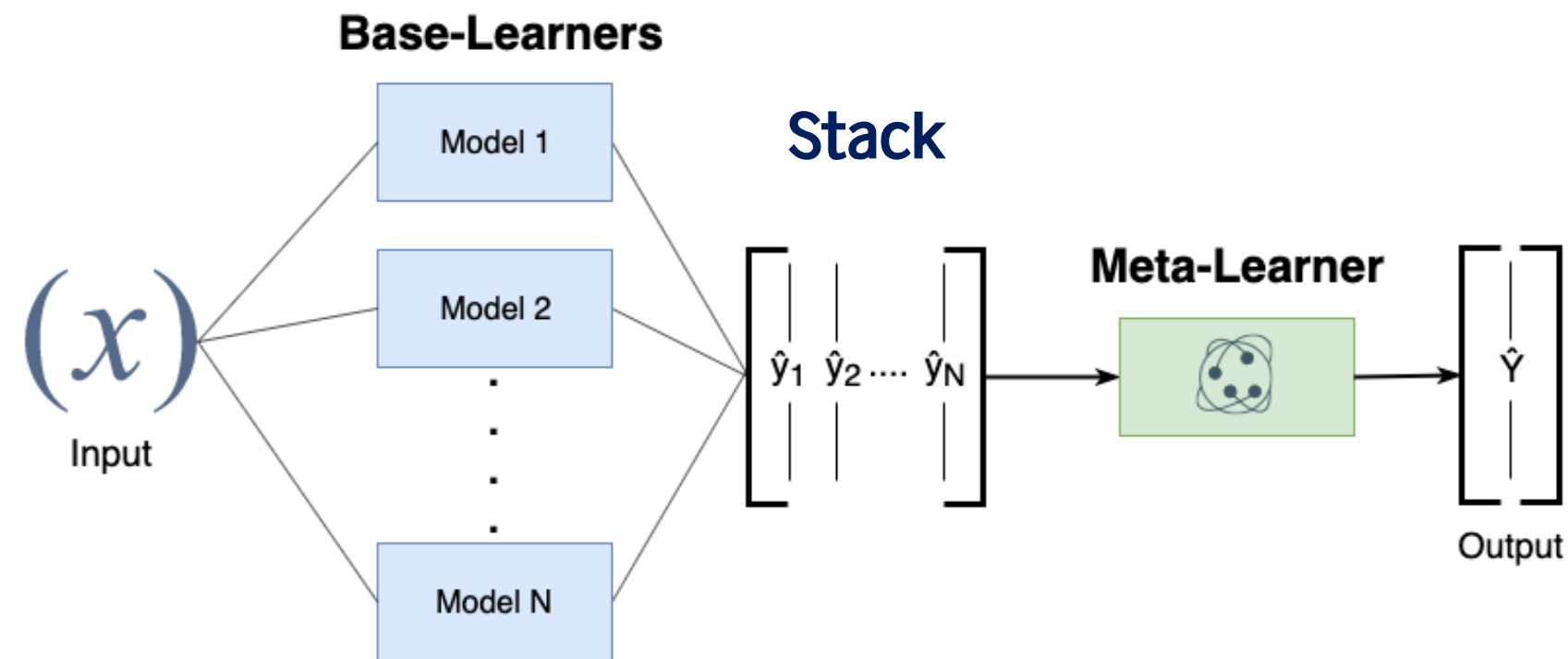


# 7.1 Stacking

## 구조

- 개별 base 모델
- 최종 meta 모델 : 개별 base 모델의 예측 결과를 학습해 최종 결과를 냄

이때, 개별 모델의 예측 결과를 **stacking** 형태로 결합해서, 최종 meta 모델의 입력으로 사용



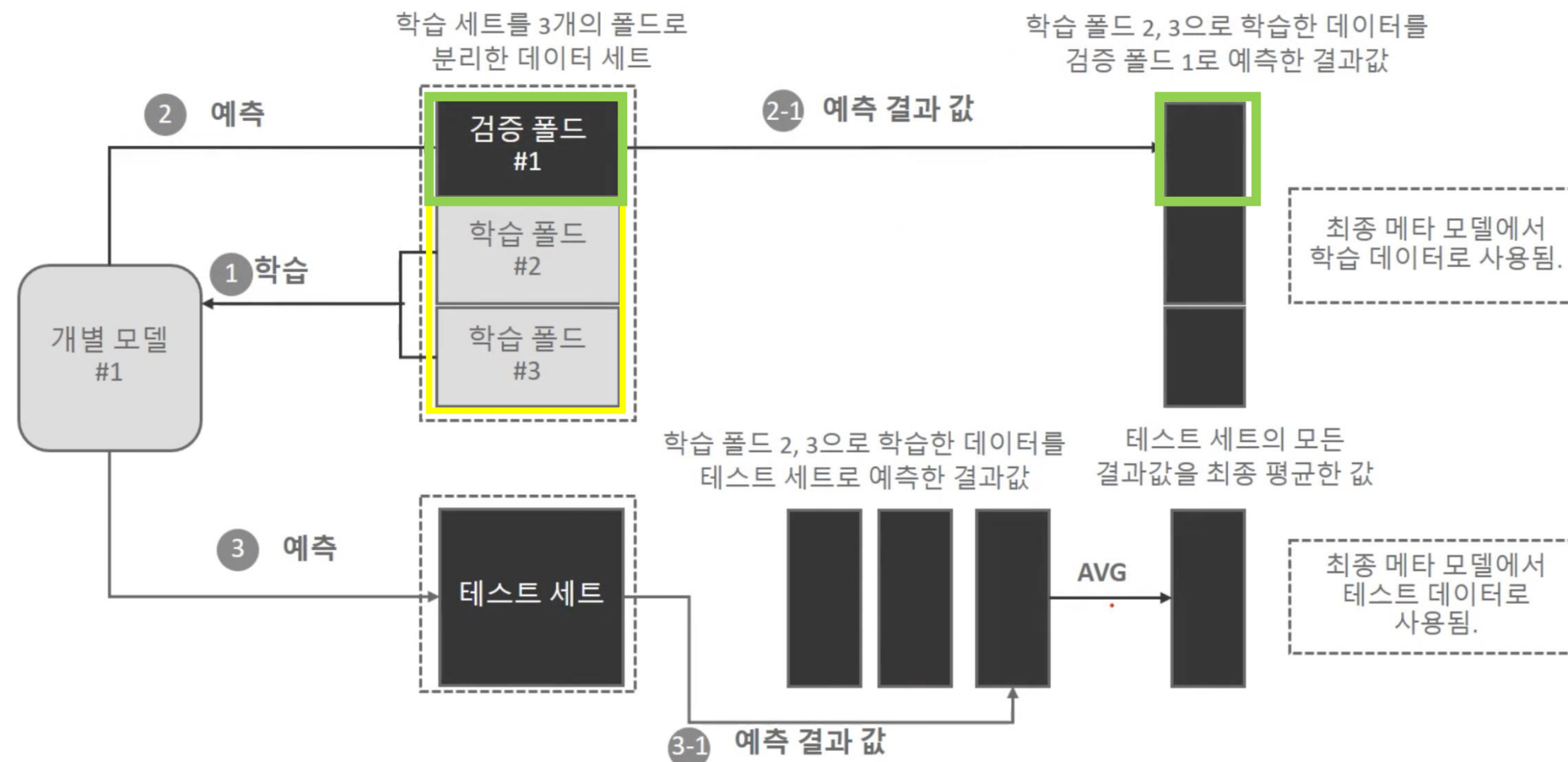
# 7.2 CV set 기반 Stacking

## 1) Step 1 : 개별 base 모델 학습, 예측값 도출

- 학습 데이터를 K개의 fold로 나눔
- K-1 개의 fold를 학습 데이터로 하여 base 모델 학습 (K번 반복)
  - 검증 fold 1개를 예측한 결과 (K fold)
  - 테스트 데이터를 예측한 결과 (K개) 의 평균

-> 최종 meta 모델의 학습용 데이터

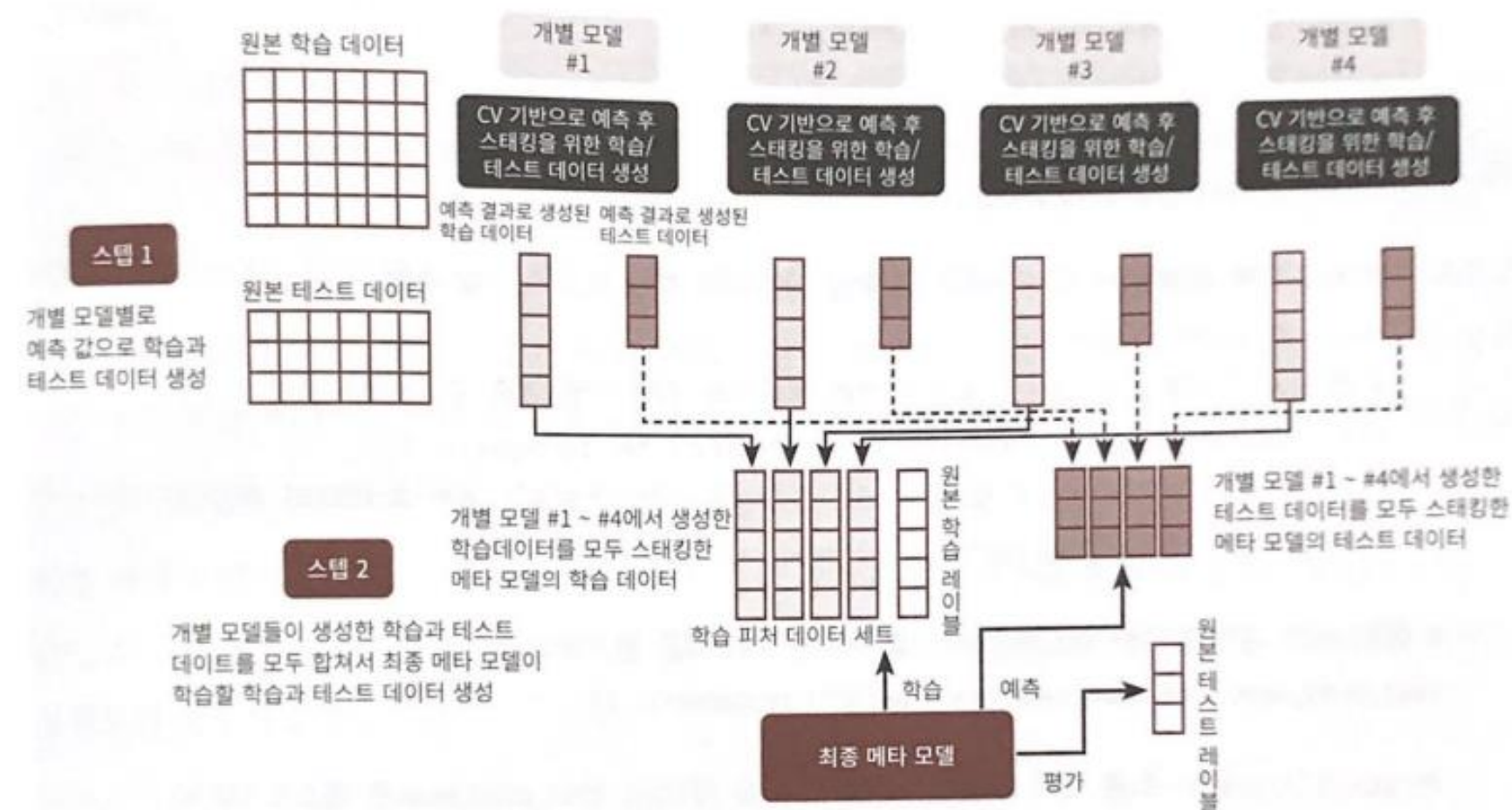
-> 최종 meta 모델의 테스트용 데이터



# 7.2 CV set 기반 Stacking

## 2) Step 2 : 최종 meta 모델 학습

- 각 base 모델이 생성한 학습용 데이터를 stacking -> 최종 meta 모델의 학습용 데이터 세트
- 각 base 모델이 생성한 테스트용 데이터를 stacking -> 최종 meta 모델의 테스트용 데이터 세트
- 최종 학습용 데이터 + 원본 학습 레이블 데이터로 학습
- 최종 테스트용 데이터로 예측 -> 원본 테스트 레이블 데이터로 평가



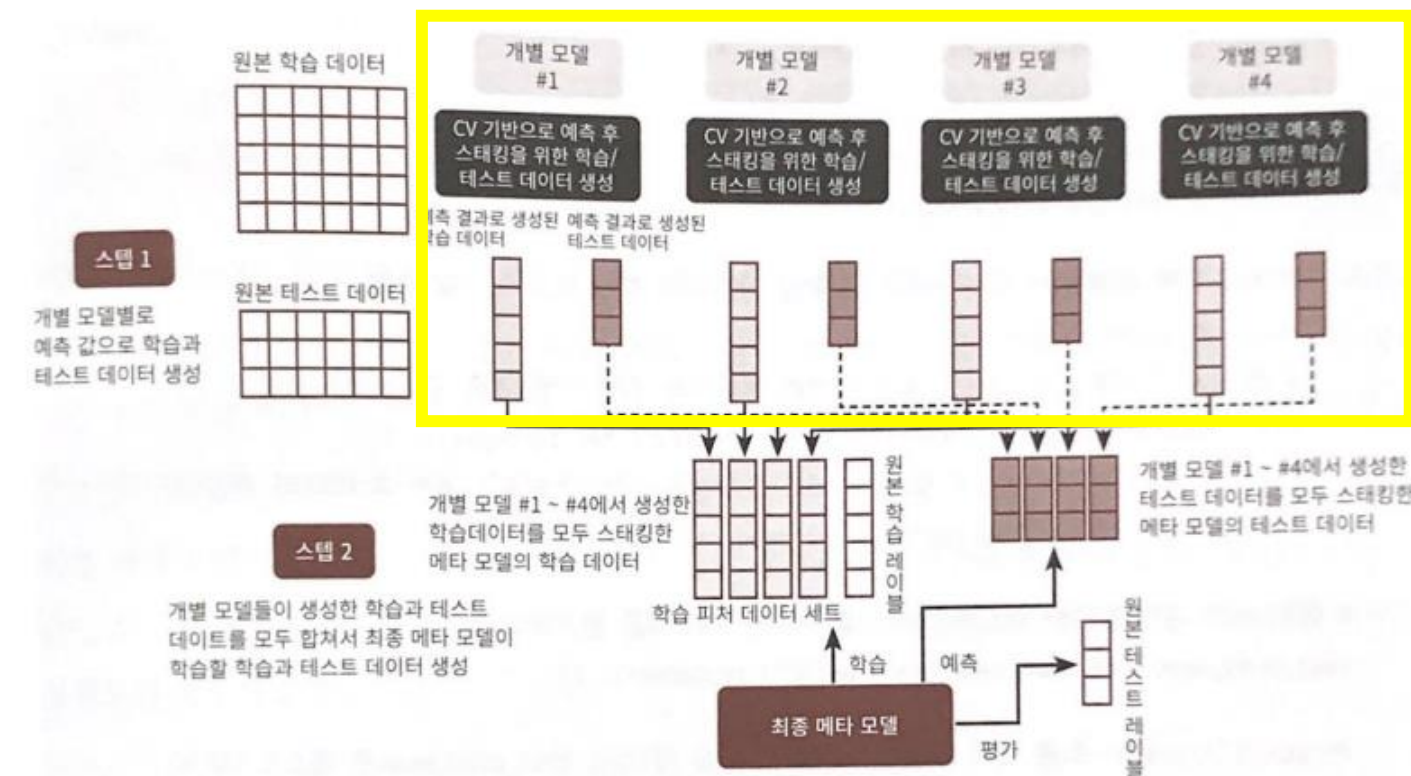
# 7.3 Stacking 적용 – 위스콘신 유방암 예측

## 1) Step 1을 위한 함수

```
def get_meta_dataset(model, X_train, y_train, X_test, n_folds):  
  
    kfold = KFold(n_splits=n_folds, shuffle=False)  
    train_meta = np.zeros((X_train.shape[0], 1))  
    test_meta = np.zeros((X_test.shape[0], n_folds))  
  
    for fold_cnt, (train_index, valid_index) in enumerate(kfold.split(X_train)):  
        X_tr = X_train[train_index]  
        y_tr = y_train[train_index]  
        X_val = X_train[valid_index]  
  
        model.fit(X_tr, y_tr)  
        train_meta[valid_index, :] = model.predict(X_val).reshape(-1, 1)  
        test_meta[:, fold_cnt] = model.predict(X_test)  
  
    test_meta = np.mean(test_meta, axis=1).reshape(-1, 1)  
  
    return train_meta, test_meta
```

## 2) Step 1 : 개별 base 모델별로 함수 실행

```
dt_train, dt_test = get_meta_dataset(dt_clf, X_train, y_train, X_test, 7)  
knn_train, knn_test = get_meta_dataset(knn_clf, X_train, y_train, X_test, 7)  
rf_train, rf_test = get_meta_dataset(rf_clf, X_train, y_train, X_test, 7)  
ab_train, ab_test = get_meta_dataset(ab_clf, X_train, y_train, X_test, 7)
```





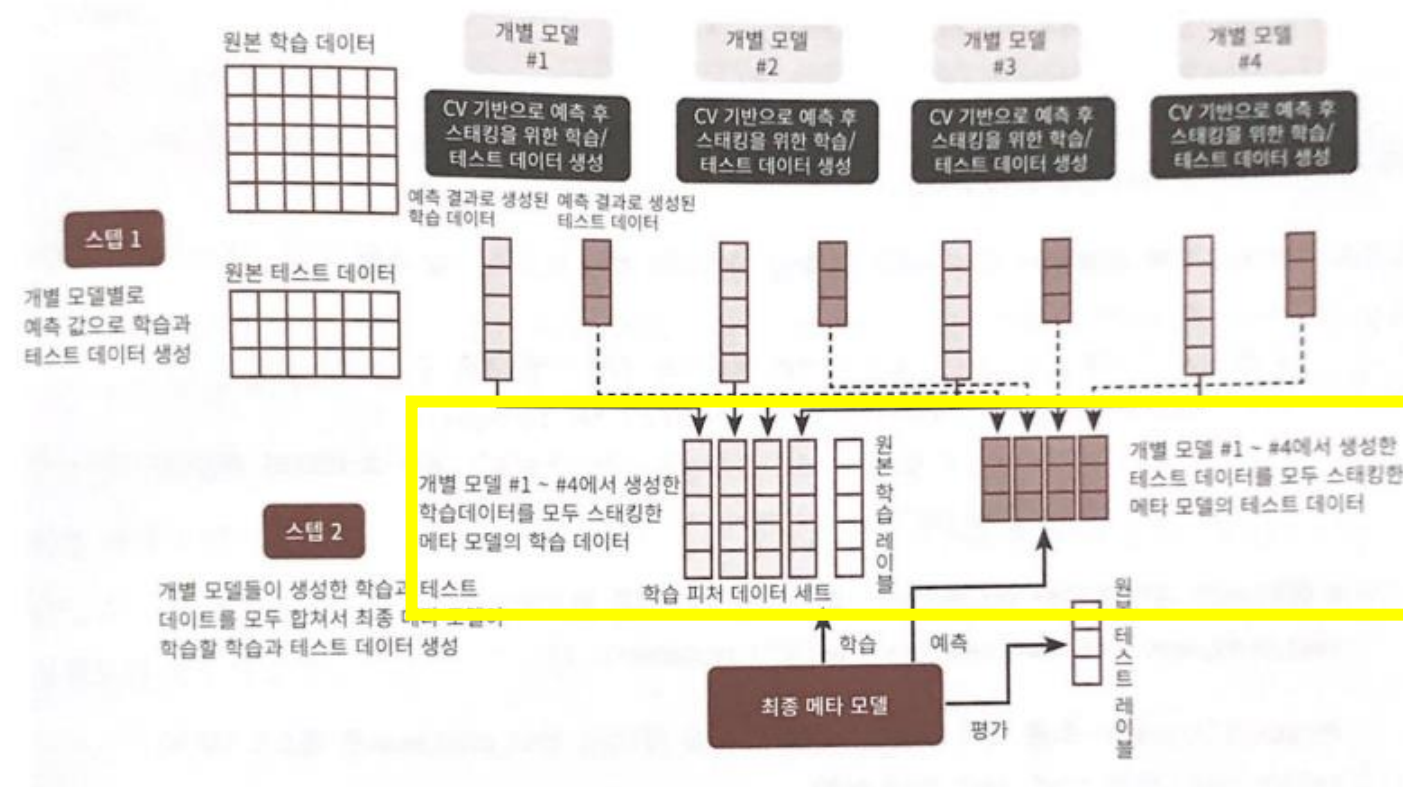
## 7.3 Stacking 적용 – 위스콘신 유방암 예측

### 3) Step 2 : meta 모델이 사용할 학습용, 테스트용 데이터 생성 (stacking)

```
final_X_train_stacked = np.concatenate((dt_train, knn_train, rf_train, ab_train), axis=1)
final_X_test_stacked = np.concatenate((dt_test, knn_test, rf_test, ab_test), axis=1)

print("Original train X :", X_train.shape, "Original test X :", X_test.shape)
print("Stacking train X :", final_X_train_stacked.shape, "Stacking test X :", final_X_test_stacked.shape)
```

Original train X : (455, 30) Original test X : (114, 30)  
Stacking train X : (455, 4) Stacking test X : (114, 4)





## 7.3 Stacking 적용 – 위스콘신 유방암 예측

### 4) Step 2 : meta 모델 학습 & 예측

```
lr_meta = LogisticRegression()  
lr_meta.fit(final_X_train_stacked, y_train)  
pred_meta = lr_meta.predict(final_X_test_stacked)
```

```
print("최종 meta model accuracy: {:.4f}".format(accuracy_score(y_test, pred_meta)))
```

최종 meta model accuracy: 0.9825

스태킹 앙상블

```
rf_meta = RandomForestClassifier()  
rf_meta.fit(final_X_train_stacked, y_train)  
pred_meta = rf_meta.predict(final_X_test_stacked)
```

```
print("최종 meta model accuracy: {:.4f}".format(accuracy_score(y_test, pred_meta)))
```

최종 meta model accuracy: 0.9912

Decision Tree accuracy: 0.9123

KNN accuracy: 0.9386

Random Forest accuracy: 0.9474

AdaBoost accuracy: 0.9561

개별 모델

# THANK YOU

