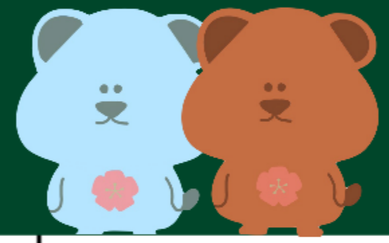
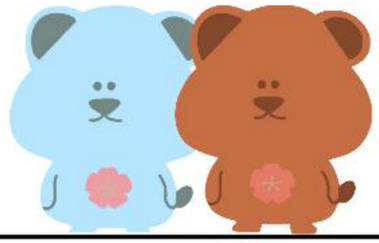


3주차 발표

DA팀 김예진 박지운 이의진

EWHA EURON

목차



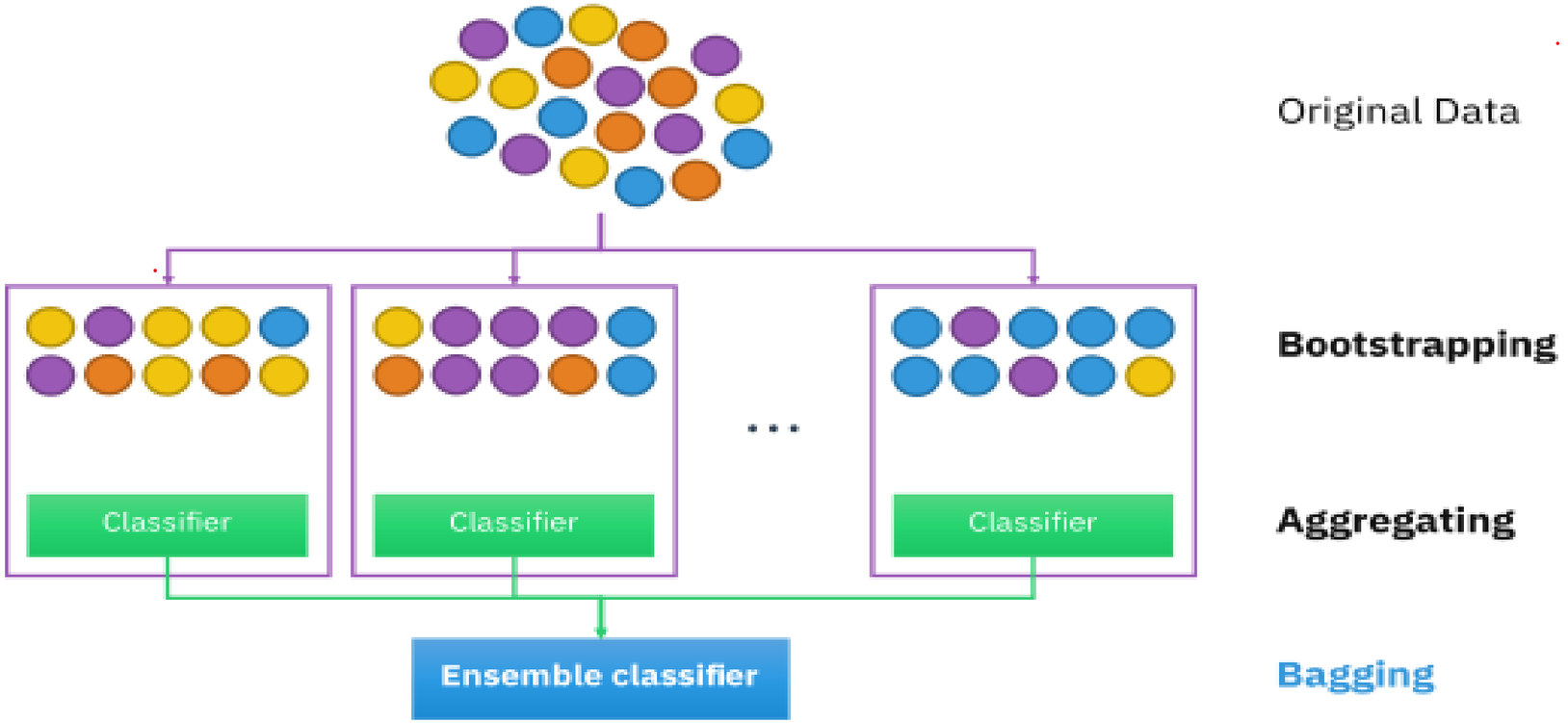
EWHA EURON

01 배깅 vs 부스팅 vs 스태킹

EWHA EURON

1.1 배깅

배깅(Bagging)이란 ? -> 부트스트랩을 집계하는 것

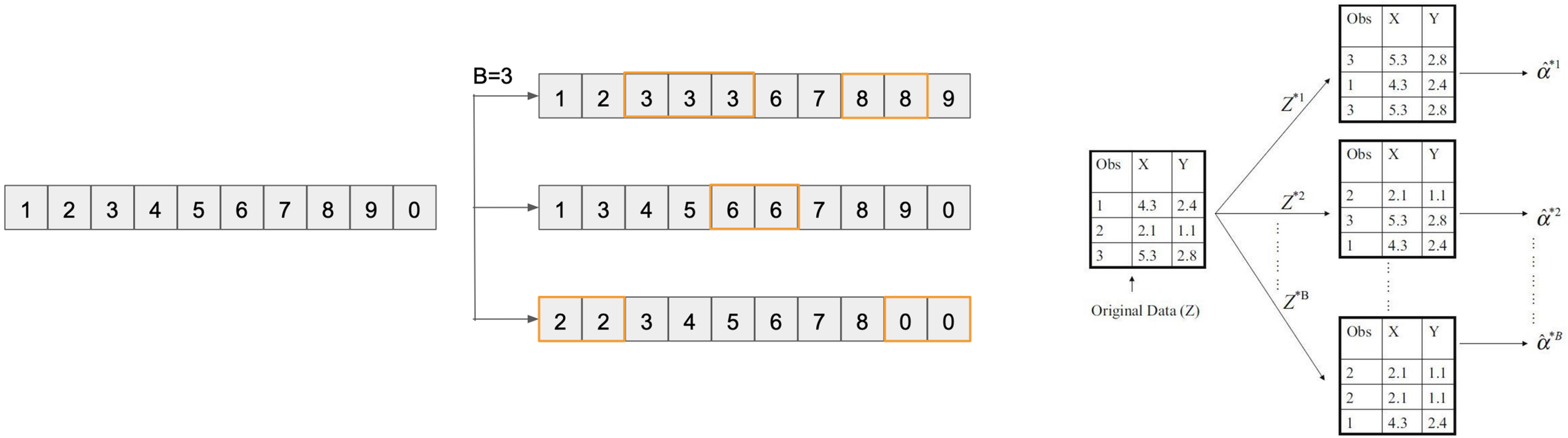


한가지 분류 모델을 여러 개 만들어 서로 다른 학습 데이터로 학습시킨 후 (bootstrap)
동일한 테스트 데이터에 대한 서로 다른 예측 값들을 투표를 통해 (aggregating)
가장 높은 예측 값으로 최종 결론을 내리는 앙상블 기법

1.2 부트스트랩

부트스트랩(bootstrap)

-> 여러 개의 데이터를 중첩되게 분리하는 것

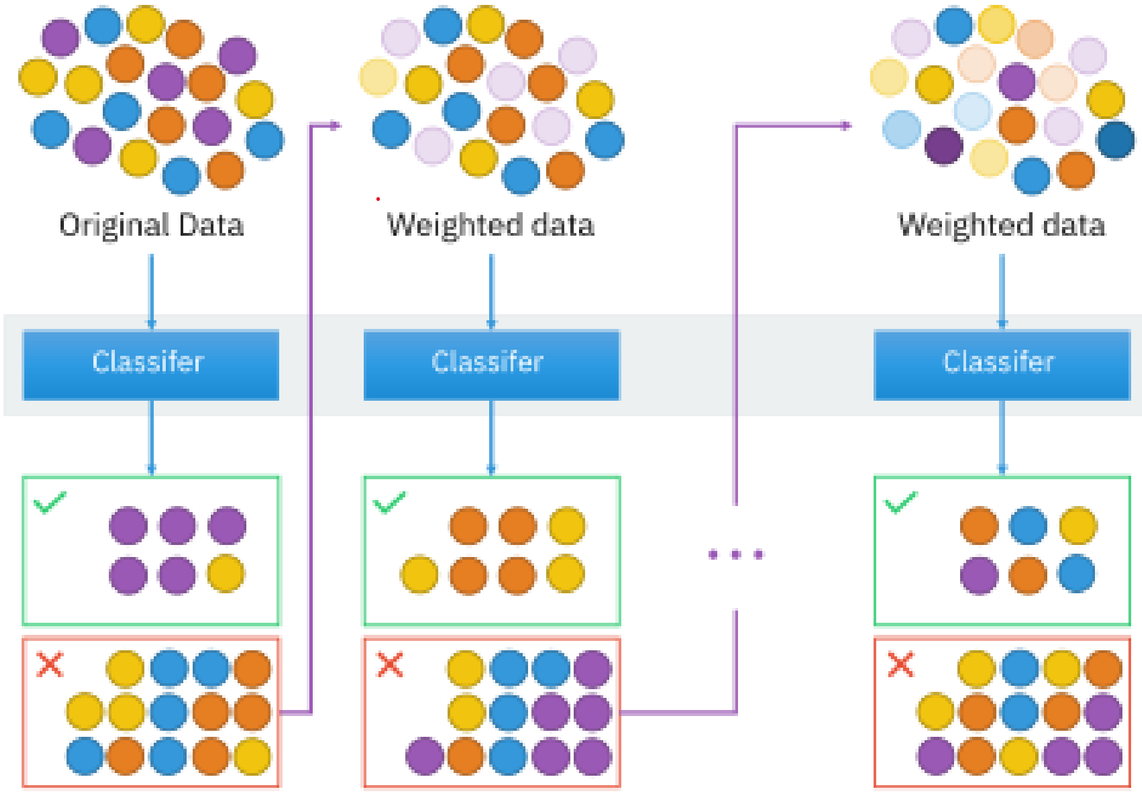


단순 복원 임의추출법을 통해 raw data로부터 같은 크기의 표본 자료들을 생성하는 것
-> 데이터 내에서 반복적으로 샘플을 사용하는 resampling 방법 중 하나

학습 데이터가 충분하지 않더라도 충분한 학습 효과를 주어 높은 bias의 underfitting 문제나
높은 variance로 인한 Overfitting 문제를 해결하는데 도움을 줌

1.3 부스팅

부스팅(Boosting)이란 ?

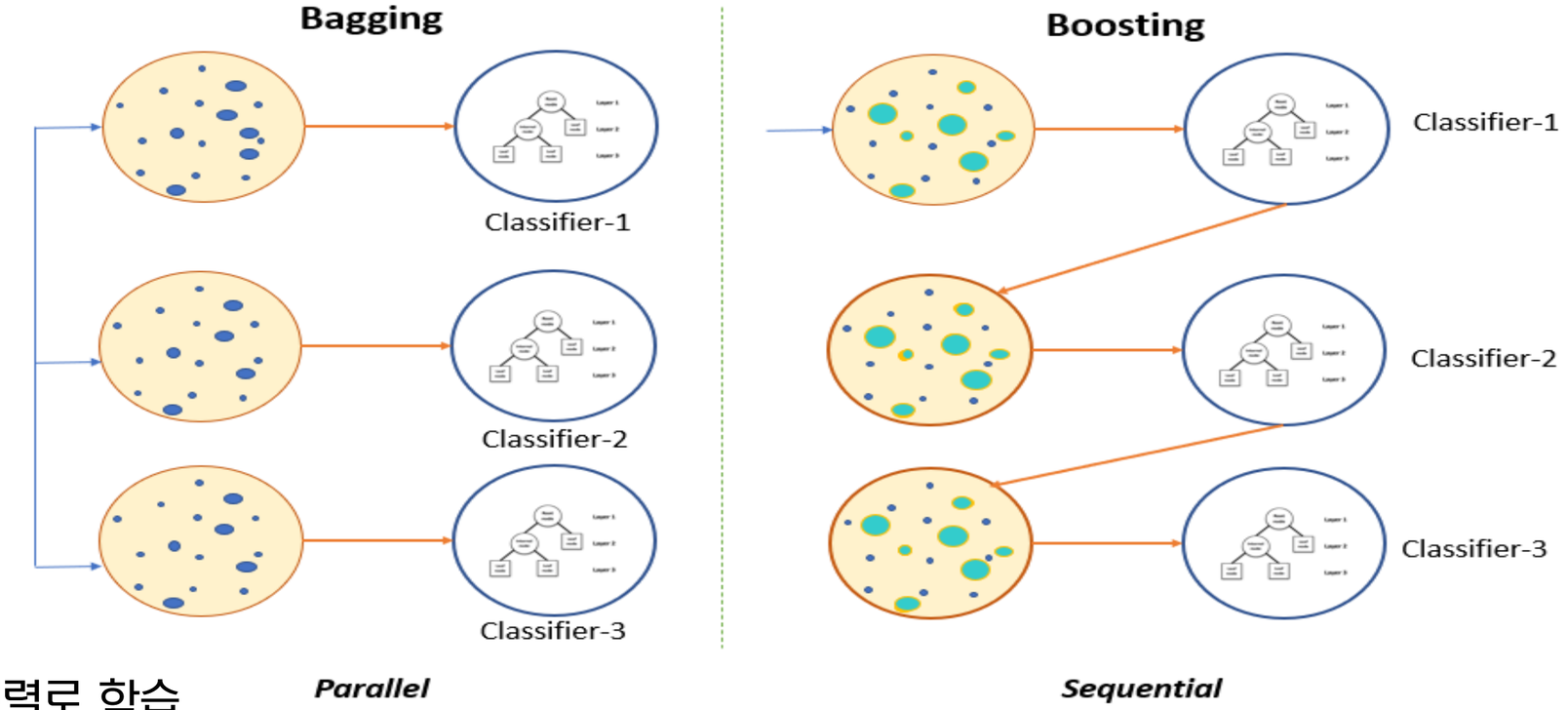


여러 개의 약한 학습기를 순차적으로 학습, 예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식

-> 대표적인 알고리즘:
XGBoost, AdaBoost, Gradient Boost

1.4 배깅 vs 부스팅

배깅 vs 부스팅

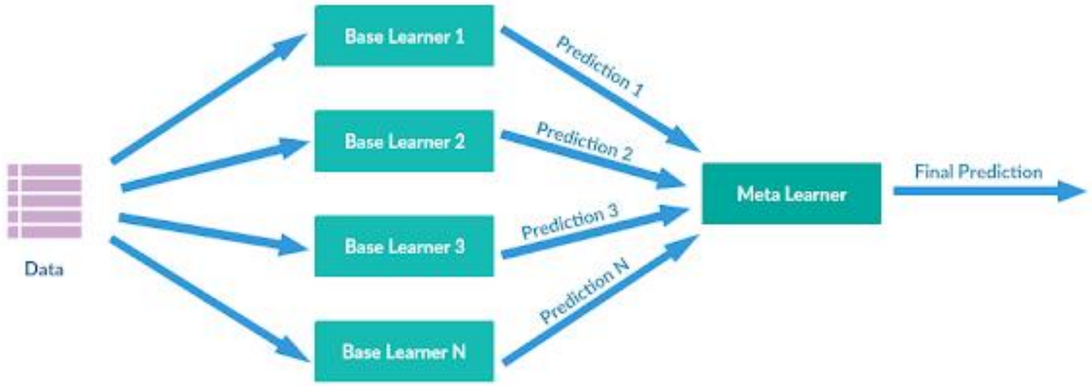


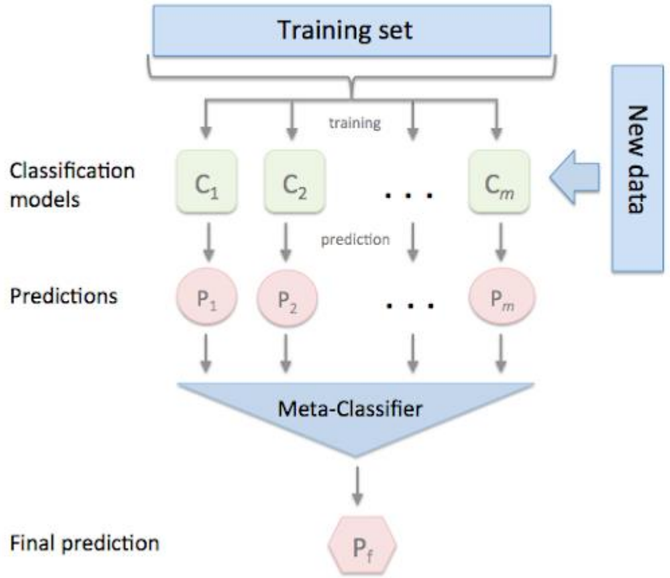
배깅은 병렬로 학습
부스팅은 순차적으로 학습하고 결과에 따라 가중치 부여

부스팅은 배깅에 비해 error가 적음 (성능이 좋다)
하지만 속도가 느리고 오버 피팅이 될 가능성이 있다
-> 개별 결정 트리의 낮은 성능이 문제라면 부스팅이 적합, 오버 피팅이 문제라면 배깅이 적합

1.5 스태킹

스태킹이란 ? 개별 알고리즘의 예측한 데이터를 기반으로 다시 예측을 수행하는 방법





모델이 예측한 결과를 다시 training set으로 활용
즉, base learner로 예측한 결과들을 다음 모델에 input data로 넣어 예측

스태킹 방식에 사용되는 모델은 크게 기반 모델과 메타 모델로 구분
기반 모델: 1차 예측을 수행하는 개별 알고리즘을 의미
메타 모델: 기반 모델의 예측 결과를 최종 데이터 세트로 학습하는 별도의 ML알고리즘

1.6 배깅 vs 부스팅 vs 스태킹

비교

	배깅	부스팅	스태킹
Partitioning of the data	랜덤	잘못 분류된 샘플에 가중치 부여	다양
목표	분산 감소	예측력 강화	both
사용되는 방법	random subspace	gradient descent	blending
Function to combine Single models	(weighted) average	weighted majority vote	logistic regression

02 AdaBoost vs Gradient Boost

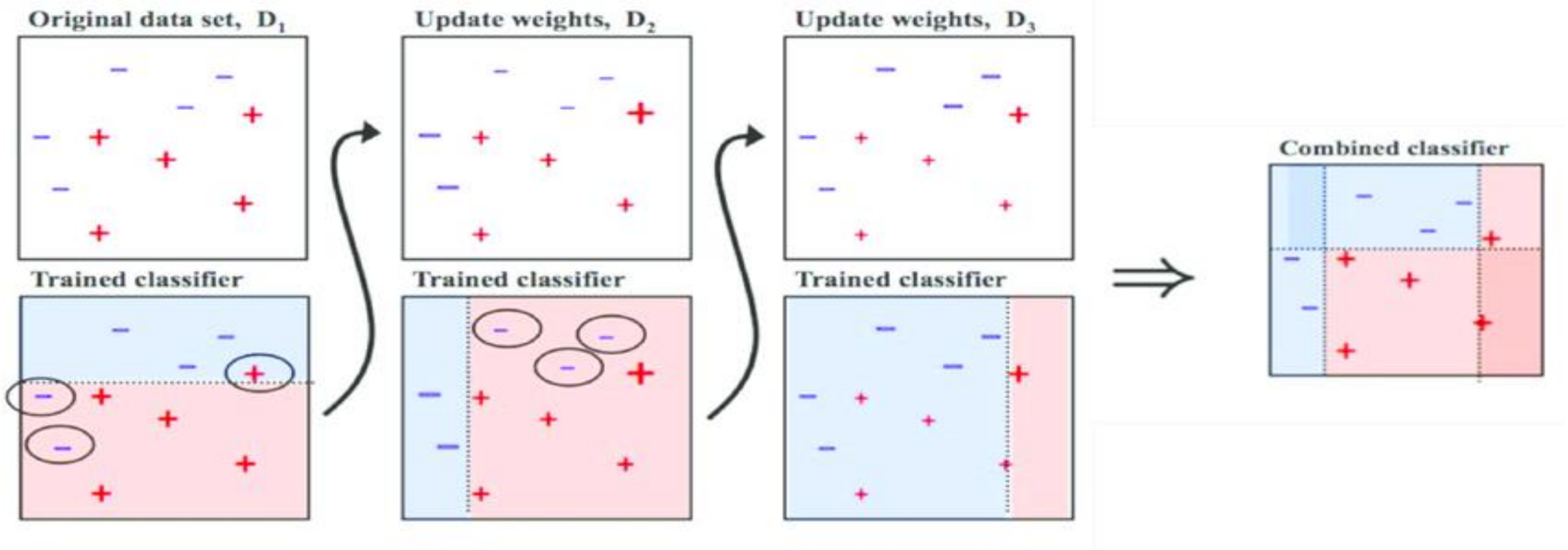
EWHA EURON

2.1 AdaBoost

에이다부스트(Adaboost)

Weak classifier 들이 상호보완 하도록 순차적으로 학습하고 이들을 조합하여 최종적으로 strong classifier의 성능을 향상시키는 것

Adaptive + boosting



먼저 학습된 분류기는 제대로 분류를 해내는 데이터와 제대로 분류해내지 못하는 데이터들이 발생한다.
먼저 학습된 분류기가 제대로 분류한 결과 정보와 잘못 분류한 결과 정보를 다음 분류기에 전달한다.
다음 분류기는 이전 분류기로부터 받은 정보를 활용하여 잘 분류해내지 못한 데이터들의 가중치를 높이며
잘못 분류되는 데이터에 더 집중하여 학습한다

2.1 AdaBoost

에이다부스트(Adaboost)

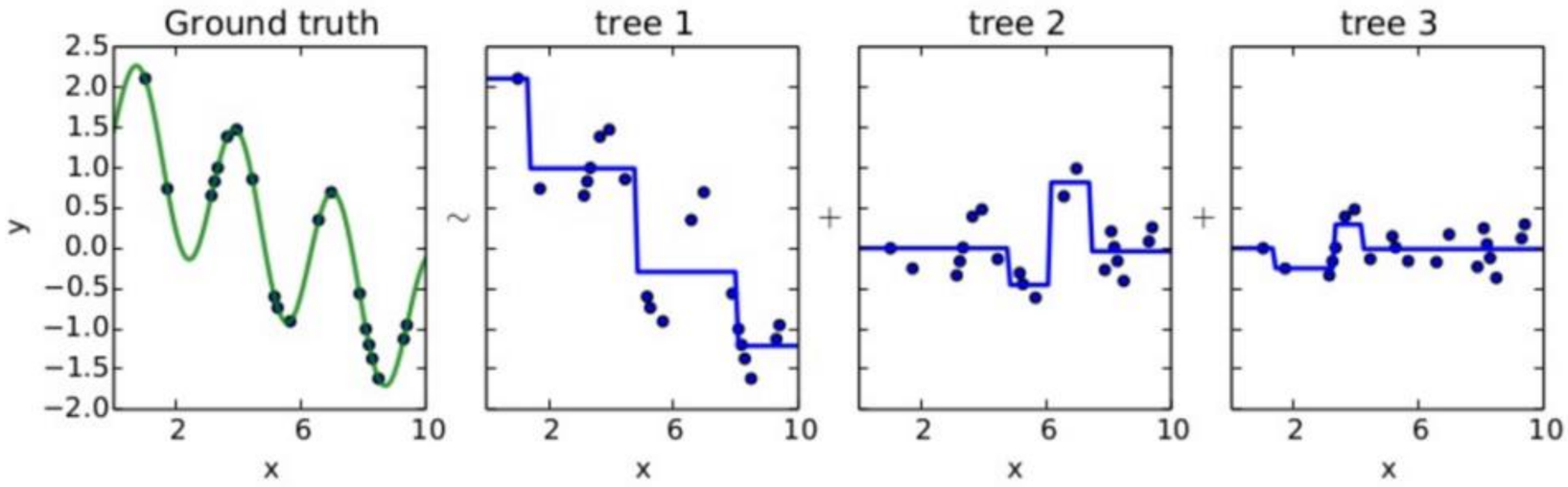
$$H(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + ... + \alpha_t h_t(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

H(x): 최종 강한 분류기
h: 약한 분류기
α: 약한 분류기의 가중치
t: 반복 횟수

2.2 Gradient Boost

Gradient Boost

Adaboost와 유사하나 가중치 업데이트를 경사 하강법을 이용함



에이다부스트처럼 앙상블에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가함
반복마다 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔여오차(residual error)에 새로운 예측기를 학습시킴
랜덤 포레스트와 달리 이진 트리의 오차를 보완하는 방식으로 순차적으로 트리를 만든다
-> 일반적으로 랜덤 포레스트보다는 예측 성능이 조금 뛰어나지만 시간이 오래걸리고 하이퍼 파라미터 튜닝 노력도 더 필요

EWHA EURON

2.2 Gradient Boost

Gradient Boost

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

-> Loss function을 다음과 같이 정의

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

-> 함수의 gradient는 다음과 같다

$$y_i - f(x_i)$$

-> 앞에 (-)를 붙여주면 다음과 같음

Residual은 실제값과 예측값의 차이이므로 이는 잔차를 나타내는 식과 같다.
따라서 loss function을 MSE로 정의했을 때 negative gradient=residual 이 성립하게 된다.

03 GBM 하이퍼 파라미터 및 튜닝

EWHA EURON

3.1 GBM 하이퍼 파라미터

loss	경사하강법에서 사용할 loss function 지정 기본값은 deviance
learning late	학습을 진행할 때마다 적용하는 학습률(0~1) 약한 학습기가 순차적으로 오류 값을 보정해나갈 때 적용하는 계수 작은 값 적용시 예측 성능이 높아질 가능성이 높지만 수행 시간이 오래 걸림 기본값은 0.1
subsample	개별트리가 학습에 사용하는 데이터 샘플링 비율 기본값은 1
n_estimators	weak learner 의 개수 지정 개수가 많을수록 성능이 좋아지지만 수행 시간이 오래 걸림 기본값은 100

3.1 GBM 하이퍼 파라미터

GBM 하이퍼 파라미터 및 튜닝

```
1   from sklearn.ensemble import GradientBoostingClassifier
2   import warnings
3   warnings.filterwarnings('ignore')
4
5   X_train, X_test, y_train, y_test = get_human_dataset()
6
7   gb_clf=GradientBoostingClassifier(random_state=0)
8   gb_clf.fit(X_train,y_train)
9   gb_pred=gb_clf.predict(X_test)
10  gb_accuracy=accuracy_score(y_test,gb_pred)
11
12  print('GBM 정확도: {0:.4f}'.format(gb_accuracy))
```

```
GBM 정확도: 0.9386
```

3.1 GBM 하이퍼 파라미터

GBM 하이퍼 파라미터 및 튜닝

```
1  from sklearn.model_selection import GridSearchCV
2
3  params = {
4      'n_estimators' : [100, 500],
5      'learning_rate' : [0.05, 0.1]
6  }
7  grid_cv = GridSearchCV(gb_clf, param_grid=params, cv=2, verbose=1)
8  grid_cv.fit(X_train, y_train)
9  print('최적 하이퍼 파라미터:₩n', grid_cv.best_params_)
10 print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

```
Fitting 2 folds for each of 4 candidates, totalling 8 fits
최적 하이퍼 파라미터:
 {'learning_rate': 0.05, 'n_estimators': 500}
최고 예측 정확도: 0.9002
```

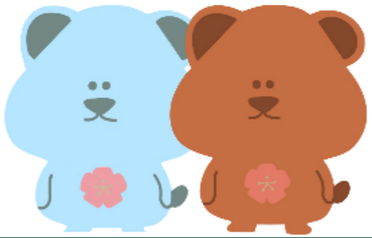
-> 최적 하이퍼 파라미터
learning rate: 0.05
n_estimator: 500

```
1  pred = grid_cv.best_estimator_.predict(X_test)
2  accuracy = accuracy_score(y_test, pred)
3  print('GBM 정확도: {0:.4f}'.format(accuracy))
```

```
GBM 정확도: 0.9393
```

EWHA EURON

04 XGBoost(eXtra Gradient Boost)

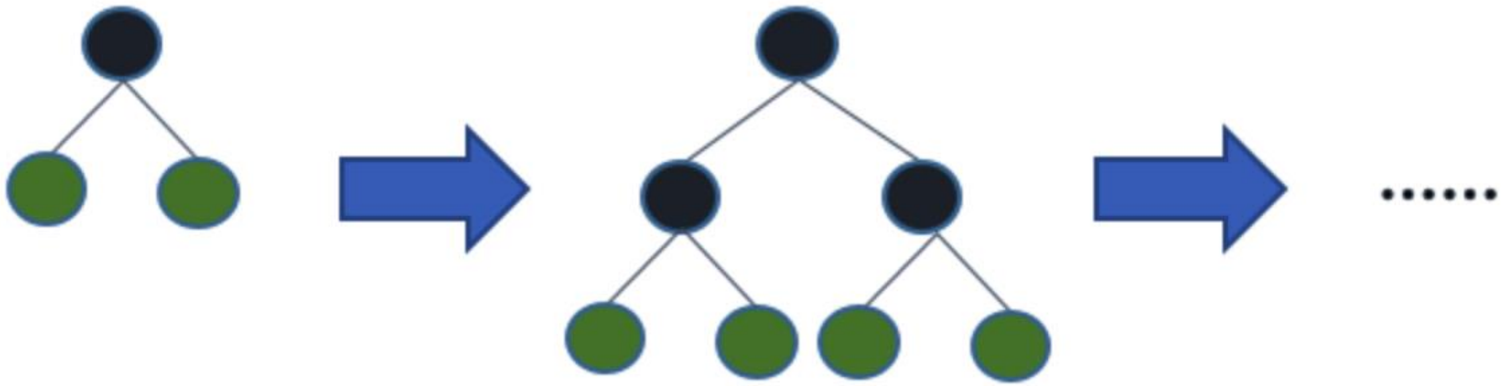


EWHA EURON

4.1 XGBoost

XGBoost란?

- 트리 기반의 앙상블 학습에서 가장 각광받고 있는 알고리즘 중 하나
- GBM에 기반하고 있지만 GBM의 단점인 **느린 수행 시간**(병렬 CPU 환경에서 병렬 학습 가능) 및 **과적합 규제(Regularization) 의 부재** 등의 문제를 해결함.



균형 트리 분할 (Level-Wise)
-> 최대한 균형 잡힌 트리를 유지하면서 분할(트리의 깊이 최소화, 과적합 방지)

4.2 XGBoost의 특징

1. **뛰어난 예측 성능**
2. **GBM 대비 빠른 수행 시간**

- 일반적인 GBM은 순차적으로 Weak learner가 가중치를 증감하는 방법으로 학습하기에 전반적인 속도가 느림
- BUT XGBoost는 병렬 수행 및 다양한 기능으로 GBM에 비해 빠른 수행 성능을 보장

3. 과적합 규제(Regularization)

- 표준 GBM은 과적합 규제 기능이 없으나 XGBoost는 자체에 과적합 규제 기능이 있음

4. Tree pruning(나무 가지치기)

- max_depth 파라미터로 분할 깊이를 조정하기도 하지만 tree pruning으로 더 이상 긍정 이득이 없는 분할을 가지치기하여 분할 수를 더 줄일 수 있음

5. 자체 내장된 교차 검증

- 반복 수행마다 내부적으로 학습 데이터 세트와 평가 데이터 세트에 대한 교차 검증을 수행하여 최적화된 반복 수행 횟수를 가질 수 있음
- 지정된 반복 횟수를 모두 수행하지 않고 평가 값이 최적화되면 반복을 중간에 멈출 수 있는 조기 중단 기능도 있음

6. 결손값 자체 처리

4.3 사이킷런 래퍼 XGBoost

파이썬 래퍼 XGBoost 모듈과 사이킷런 래퍼 XGBoost 모듈의 일부 하이퍼 파라미터는 동일한 기능을 하지만 파라미터 명이 다름.
우리는 **사이킷런 래퍼 XGBoost**를 중점적으로 설명

사이킷런 래퍼 XGBoost의 특징

1. GBM과 유사한 하이퍼 파라미터 + 조기중단(early stopping), 과적합을 규제하는 파라미터
2. 다른 Estimator와 동일하게 fit( )과 predict( )만으로 학습과 예측이 가능함
3. GridSearchCV, Pipeline 등 사이킷런의 다른 유틸리티를 그대로 사용할 수 있음

XGBClassifier(분류), XGBRegressor(회귀)

4.4 XGBoost 주요 파라미터

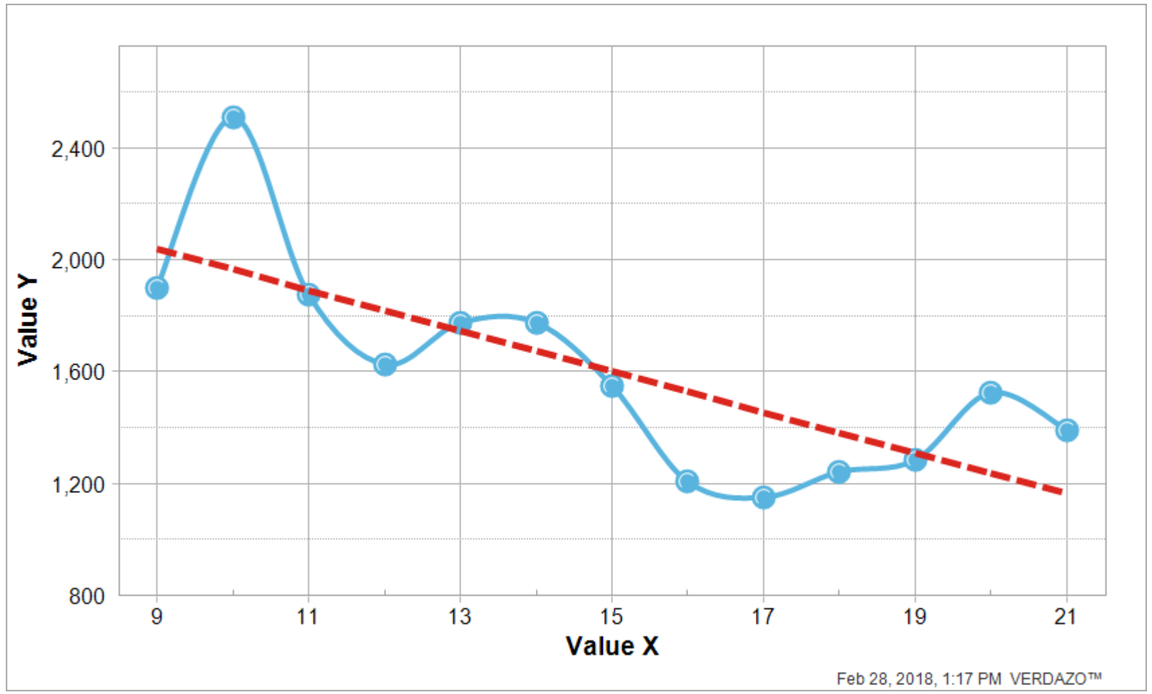
learning_rate (eta)	GBM의 학습률과 같은 파라미터 (0 ~ 1 사이의 값 지정), default=0.1 보통 0.01 ~ 0.2 사이의 값을 선호한다.
n_estimators (num_boost_rounds)	weak learner의 개수
min_child_weight	트리에서 추가적으로 가지를 나눌지 결정하기 위해 필요한 데이터들의 weight 총합. 값이 클수록 분할을 자제(과적합 조절용), default=1
gamma	트리의 리프 노드를 추가적으로 나눌지를 결정할 최소 손실 감소 값 해당 값보다 큰 손실이 감소된 경우에 리프 노드를 분리 값이 클수록 과적합 감소 효과 有, default=0
max_depth	0으로 지정하면 깊이에 제한이 없음 max_depth가 크면 과적합 가능성이 높아짐(보통 3~10 사이의 값 적용)
subsample (sub_sample)	트리가 커져 과적합되는 것을 제어하기 위해 데이터를 샘플링하는 비율을 지정 일반적으로 0.5~1사이의 값 적용
reg_lambda (lambda)	L2(Ridge) Regularization 적용값 피쳐 개수가 많은 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있음.
reg_alpha (alpha)	L1(Lasso) Regularization 적용값 피쳐 개수가 많은 경우 적용을 검토하며 값이 클수록 과적합 감소 효과가 있음.

4.5 Regularization(과적합 규제)

Regularization을 하는 이유

학습 데이터가 아닌 새로운 데이터를 만났을 때 **일반적으로 들어 맞는 모델을 만들기 위함**

(과적합 방지)



→ 우리가 기대하는 일반성을 띈 모델은 빨간 점선과 같음

→ 과적합이라는 것은 모델을 그래프로 그렸을 때, 빨간 점선과 같은 것이 아닌 파란 곡선과 같은 것 (모델의 차수가 너무 큰 것)

$y = w1x1 + w2x2 + w3x3 + x4x4 + .. wnxn + b$

→ 모델의 차수, 즉 피쳐의 개수(x1, x2,...)를 줄이지 않고 최대한 차수를 줄이는 방법은?

⇒ **W값, 즉 각 x에 대한 계수 값을 조정해나가자!(Regularization)**

4.5 Regularization(과적합 규제)

$$J(\theta) = \frac{1}{2m} \sum (h_\theta(x^{(i)}) - y^{(i)})^2$$ → 이 Cost function에 어떤 항이 추가적으로 붙느냐에 따라 L1, L2 Regularization을 구분

L1 Regularization(Lasso)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n} |\theta_j|$$

Gradient Descent 식에서 위 Cost function 을 접목

→ W를 업데이트 할 때, **계속해서 특정 상수를 빼주는 꼴**

→ W를 계속해서 업데이트 해나갈 때, W의 원소인 **어떤 wi 는 0 이 되도록 함.**(계속해서 특정상수를 빼나가기 때문)

→ **영향을 크게 미치는 핵심적인 피처 xi 들만 반영하도록 함!**

L2 Regularization(Ridge)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \theta_j^2$$

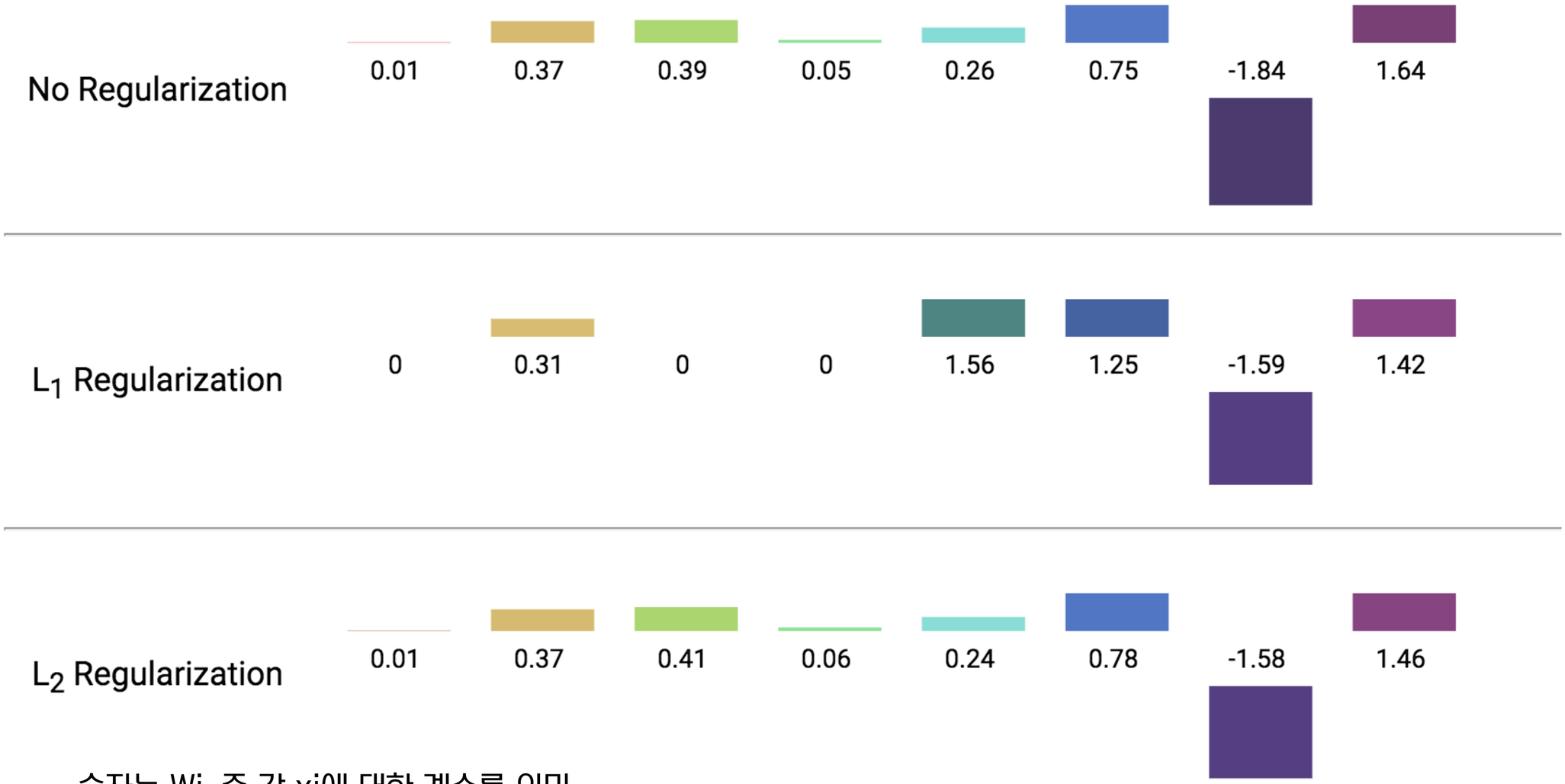
미분하여 Gradient Descent 식 에서 사용

→ **전체적으로 W 값이 작아지도록 함.**

→ Lasso 같이 일부 항의 계수를 0으로 만들어버리지는 않고,

전체적인 wi 값의 절대값을 감소시켜 덜 구불구불하게 하는 것

4.5 Regularization(과적합 규제)



→ 숫자는 Wi, 즉 각 xi에 대한 계수를 의미

4.6 사이킷런 래퍼 XGBoost 적용 - 위스콘신 유방암 예측

기본 사이킷런 래퍼 XGBoost 플로우

```
#사이킷런 래퍼 XGBoost 클래스인 XGBClassifier 임포트
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
xgb_wrapper.fit(X_train,y_train)
w_preds = xgb_wrapper.predict(X_test)
w_pred_proba = xgb_wrapper.predict_proba(X_test)[:,1]

#XGBoost 모델의 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:0.9665
```

+ 조기중단 수행 (조기 중단 관련 파라미터를 fit( )에 입력)

```
from xgboost import XGBClassifier

xgb_wrapper = XGBClassifier(n_estimators=400, learning_rate=0.1, max_depth=3)
evals = [(X_test, y_test)] #평가용 데이터 세트,
                           #원래는 완전히 알려지지 않은 데이터 세트를 사용해야 함!
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_preds_proba = xgb_wrapper.predict_proba(X_test)[:,1]
```

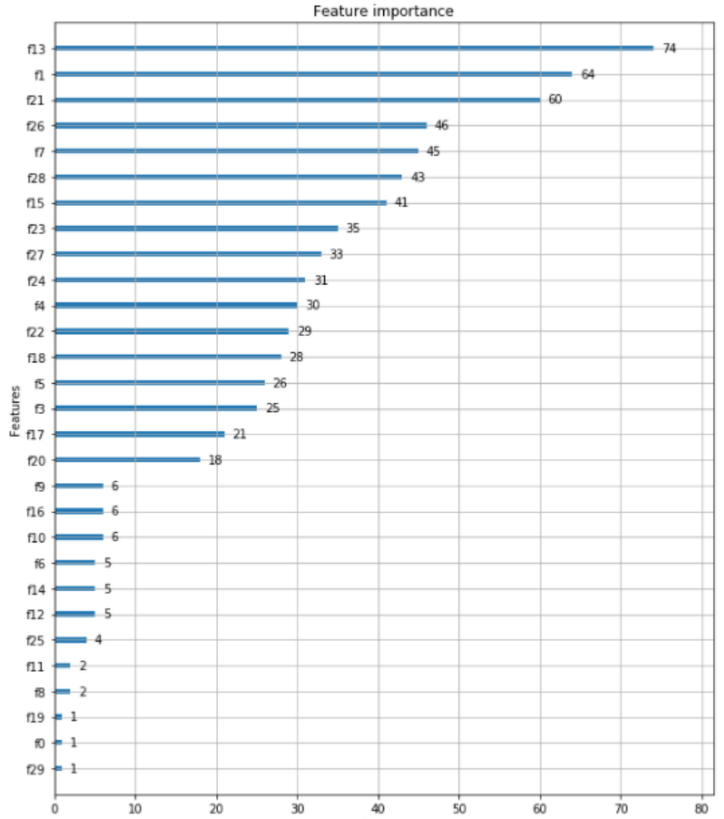
```
오차 행렬
[[34  3]
 [ 1 76]]
정확도: 0.9649, 정밀도: 0.9620, 재현율: 0.9870,    F1: 0.9744, AUC:0.9530
```

→ 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 세트**여야 함!)

⇒ 조기 중단값을 너무 급격하게 줄여버리면 예측 성능이 저하될 수 있음
(아직 성능이 향상될 여지가 있음에도 불구하고 적은 횟수 안에 성능 평가 지표가 향상되지 않으면 반복이 멈춰 버리기 때문)

4.7 XGBoost의 피쳐 중요도 시각화

```
from xgboost import plot_importance
import matplotlib.pyplot as plt

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



피쳐 중요도 시각화

→ **어느 데이터 요소가 확률값 계산에 중요하게 작용을 했느냐 하는 정도**를 나타내는 것

→ 기본 평가 지표로 f1 스코어를 기반으로 함

→ f0는 첫번째 피처, f1는 두번째 피처를 의미함

→ 13번째 피처가 가장 중요도가 높음을 알 수 있음

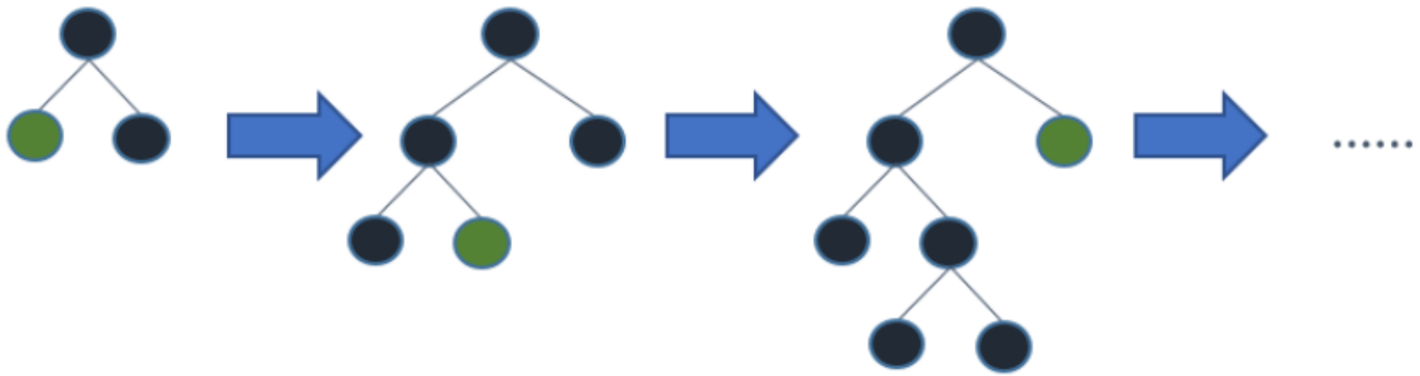
05 LightGBM

EWHA EURON

5.1 LightGBM

LightGBM 이란?

- 트리 기반의 앙상블 학습에서 XGBoost와 함께 가장 각광받고 있는 알고리즘 중 하나
- XGBoost보다 **학습에 걸리는 시간이 훨씬 적다!**
- 카테고리형 피쳐의 자동 변환과 최적 분할 지원(원-핫 인코딩 등 사용하지 않고도 카테고리형 피쳐를 최적으로 변환하고 이에 따른 노드 분할 수행)
- **적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉬움.**(10000건 이하의 데이터 세트)



Leaf-wise tree growth

리프 중심 트리 분할(Leaf Wise)

→ **트리의 균형을 맞추지 않고** 최대 손실 값을 가지는 리프 노드를 지속적으로 분할

⇒ 학습 반복을 통해 결국 균형 트리 분할 방식보다 **예측 오류 손실을 최소화** 할 수 있음 + 속도 UP!

5.2 사이킷런 래퍼 LightGBM

XGBoost와 마찬가지로 초기에는 파이썬 래퍼용 LightGBM만 개발
이후에 사이킷런과의 호환성을 위해 사이킷런 래퍼 LightGBM이 추가로 개발됨
우리는 **사이킷런 래퍼 LightGBM**을 중점적으로 설명

→ LGBMClassifier(분류), LGBMRegressor(회귀)

→ XGBoost와 매우 유사한 하이퍼 파라미터

→ BUT XGBoost와 다르게 리프노드가 계속 분할되면서 **트리의 깊이가 깊어지므로** 이러한 트리의 특성에 맞는

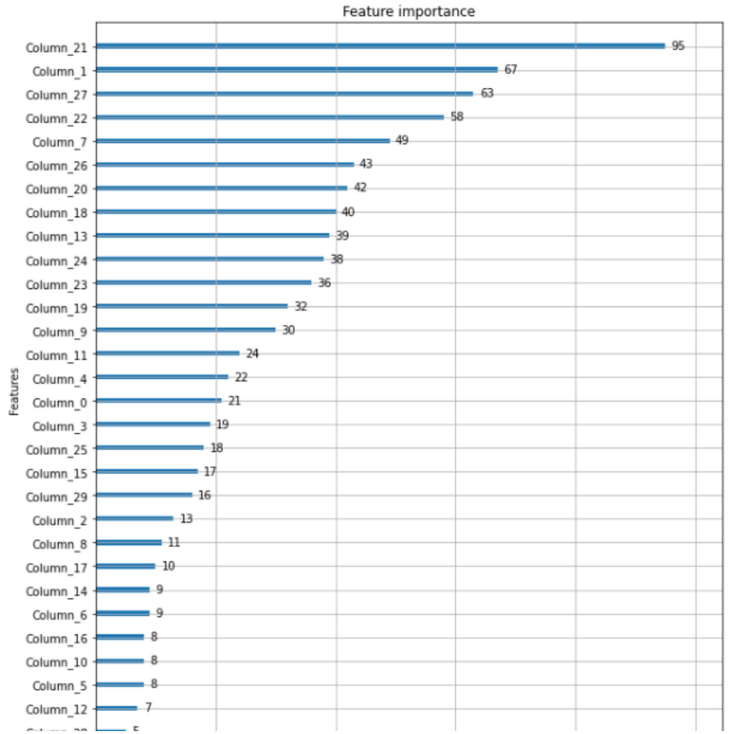
하이퍼 파라미터 설정이 필요함. (ex. max_depth를 크게 하기)

5.3 사이킷런 래퍼 LightGBM 적용 - 위스콘신 유방암 예측

```
#LightGBM의 파이썬 패키지인 lightgbm에서 LGBMClassifier 임포트
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, 20%는 테스트용 데이터 추출
X_train, X_test, y_train, y_test = train_test_split(ftr, target,
                                                    test_size=0.2, random_state=123)

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

#조기중단 수행
evals = [(X_test, y_test)] #원래는 이러면 안됨
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]
```

→ XGBoost와 동일하게 조기 중단(early stopping)가능

```
#plot_importance()를 이용해 피쳐 중요도 시각화
from lightgbm import plot_importance
import matplotlib.pyplot as plt

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



→ XGBoost와 동일하게 피쳐 중요도를 시각화 할 수 있는 plot_importance()제공

5.4 LGBM과 GridSearchCV

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators' : [100,200,300,400,500],
    'learning_rate' : [0.01,0.05, 0.1, 0.15, 0.2]

}
grid_cv = GridSearchCV(lgbm_wrapper, param_grid=params, cv=2, verbose=1)
grid_cv.fit(X_train,y_train)
print('최적 하이퍼 하라미터:', grid_cv.best_params_)
print('최고 예측 정확도:', grid_cv.best_score_)
```

```
최적 하이퍼 하라미터: {'learning_rate': 0.15, 'n_estimators': 200}
최고 예측 정확도: 0.9626420125202875
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    4.4s finished
```

→ GBM은 3시간째(..) 돌고 있음

→ LGBM은 몇초 안걸려서 바로 결과가 나온다!

06 CatBoost

EWHA EURON

6.1 CatBoost

CatBoost(Categorical Boosting)이란?

: Categorical feature를 처리하는데 중점을 둔 알고리즘

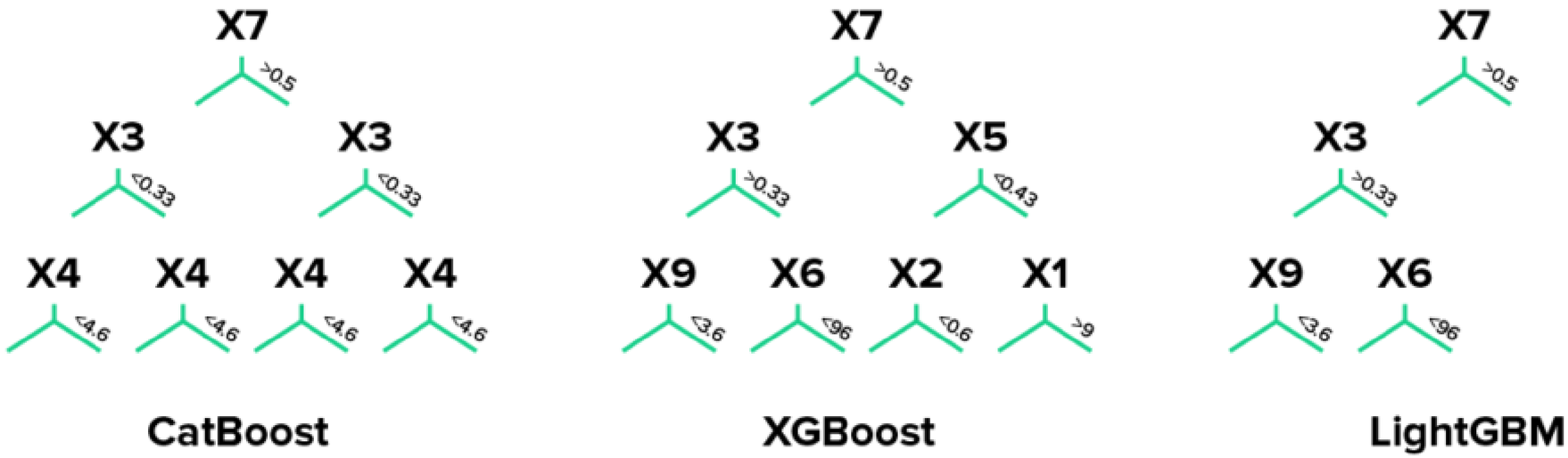
1) 장점
- 트리 기반 모델에서의 범주형 변수 처리 문제를 해결할 방식을 제공한다.
- Target leakage로 인한 과적합을 막아준다.
- 하이퍼 파라미터에 따라 성능이 달라지는 문제를 해결한다.

2) 한계
- 결측치가 많은 데이터는 잘 처리하지 못한다.
- 수치형 변수가 많은 데이터의 경우 학습속도가 느리다. 범주형 변수가 많을 때 적합한 알고리즘!

6.2 Level Wise

- CATB는 XGB와 같이 균형 트리 분할 방식(Level Wise)을 사용한다.
- 단순히 균형 잡힌 트리를 유지하는 것이 아니라 트리가 나누어지는 feature들이 대칭을 이룬다.
- 대칭 트리 구조는 예측 시간을 감소시켜 학습 속도를 빠르게 만든다.

Tree growth examples:



6.3 범주형 변수 처리

One-Hot Encoding 처리

- CATB는 level 개수가 적은 범주형 변수에 대해서 원-핫 인코딩을 적용한다.
- one_hot_max_size라는 하이퍼 파라미터 값으로 level 개수 설정 가능
- Level 개수가 많은 경우 변수의 수를 급격히 증가시키는 단점이 있다.

Target Statistics (TS) 추정

- 같은 범주에 속하는 데이터들의 y값들의 평균값을 추정하여 각 범주를 인코딩.
- 이 방식을 Targe Encoding, Mean Encoding, Response Encoding 등으로 부른다.
- Target leakage로 인한 과적합의 위험이 있다.

time	feature 1	class_labels (max_tempetarure 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

Cloudy = (15+14+20+25)/4 = 18.5

6.3 범주형 변수 처리

Ordered Boosting

- Step1. x1의 데이터로 학습한 모델을 만들고 이 모델로 x2의 잔차를 계산한다.
- Step2. x1, x2의 데이터로 학습한 모델을 만들고 이 모델로 x3의 잔차를 계산한다.
- Step3. 위 과정을 반복한다.
- 모든 데이터 포인트마다 학습을 하면 비용이 많이 들기에 log(num_of_datapoints) 모델을 사용하기도 한다.

Ordered Target Encoding

- Target leakage로 인한 과적합을 막기 위해 ordering principle을 Target Encoding에 적용한다.
- 현재 데이터를 인코딩할 때 과거 데이터의 y값만으로 평균값을 계산.

time	feature 1	class_labels (max_tempetarure 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:
Cloudy = (15+14)/2 = 15.5

Saturday:
Cloudy = (15+14+20)/3 = 16.3

6.3 범주형 변수 처리

Random Permutation

- 과거 데이터의 평균값을 추정할 때는 무작위 순열을 사용하여 인공적인 시간을 만들어냅니다.
- 과적합을 막기 위해 이 무작위 순열은 매 단계마다 섞어서 새로 뽑아줍니다.

Categorical Feature Combination

- 중복되는 변수가 2개 이상 존재할 경우 이를 하나의 변수로 통합한다.
- Feature 수를 줄여 비용을 줄인다.

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

6.4 적용 – 위스콘신 유방암 예측

1) CatBoost 설치

```
!pip install catboost
```

2) 데이터 로드 및 train set, test set 분리
3) 범주형 변수 처리 → train_pool, test_pool

```
import numpy as np

from catboost import Pool

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

cancer_data = load_breast_cancer()

X_data = cancer_data.data
y_label = cancer_data.target

X_train , X_test , y_train , y_test = train_test_split(X_data , y_label , test_size=0.2 , random_state=0)

train_pool = Pool(data=X_train, label=y_train)
test_pool = Pool(data=X_test, label=y_test)
```

Pool(data=X_train, label=y_train
cat_features = [‘categorical1’ ,’ categorical2’]

6.4 적용 – 위스콘신 유방암 예측

4) train_pool으로 CatBoost 모델 학습, test_pool로 예측

```
from catboost import CatBoostClassifier

catb = CatBoostClassifier(iterations=400, learning_rate=0.1, max_depth=3)
catb.fit(train_pool)
catb_preds = catb.predict(test_pool)
catb_pred_proba = catb.predict_proba(test_pool)[:, 1]
```

5) 정확도 측정

```
get_clf_eval(y_test, catb_preds, catb_pred_proba)
```

```
오차 행렬
[[45  2]
 [ 3 64]]
정확도: 0.9561, 정밀도: 0.9697, 재현율: 0.9552,    F1: 0.9624, AUC:0.9978
```

6.5 부스팅 모델별 비교

GBM → XGB

- 병렬 학습 가능
- Level Wise
- 과적합 규제 기능
- 자체 내장된 교차 검증 기능
- 결손값 자체 처리 기능

XGB → LGBM

- Leaf Wise
- GOSS 방식을 사용하여 Information Gain이 적은 가지의 데이터를 증폭시킴. 데이터 분포를 많이 바꾸지 않고 훈련이 잘 되지 않은 부분에 초점 맞추도록.
- 카테고리형 피처의 자동 변환, 최적 분할
- 학습 속도 및 성능 개선
- 작은 메모리 사용
- 데이터가 적을 경우 과적합 위험

LGBM → CATB

- 범주형 변수 처리 문제 해결
- 과적합을 줄여준다.
- 하이퍼 파라미터에 따라 성능이 달라지는 문제를 해결한다.
- 결측치가 많거나 수치형 변수가 많은 데이터에 부적합

07 스태킹 앙상블

EWHA EURON

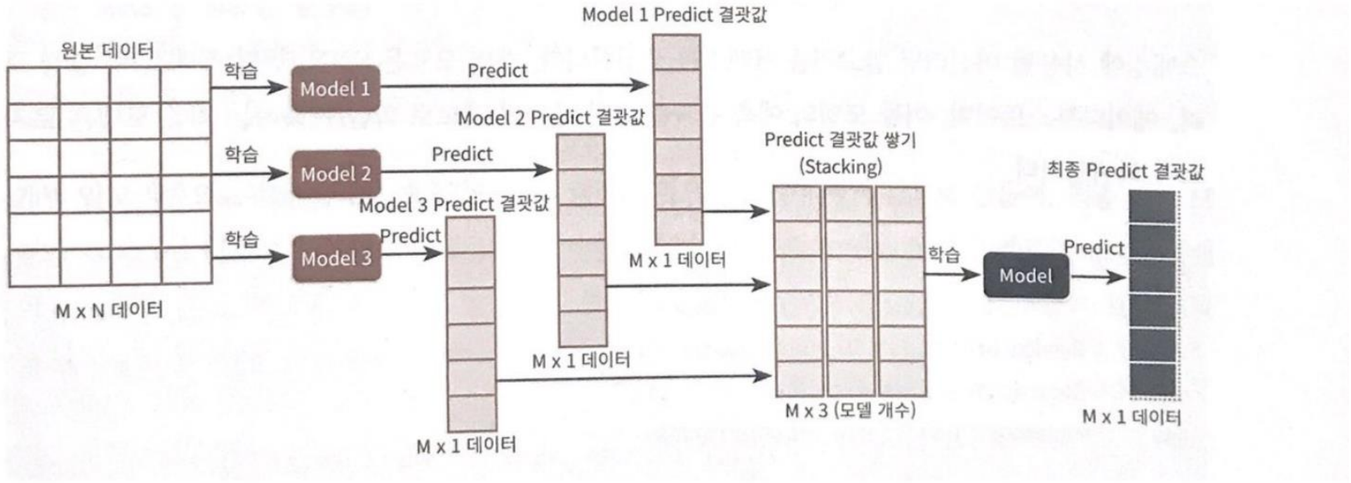
7.1 스태킹 앙상블

스태킹(Stacking)이란?

: 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행하는 방식

1) 개별적인 기반 모델
2) 최종 메타 모델: 개별 기반 모델의 예측 데이터를 학습 데이터로 만들어 최종 학습 수행
→ 테스트 데이터를 다시 기반으로 하여 학습 및 예측

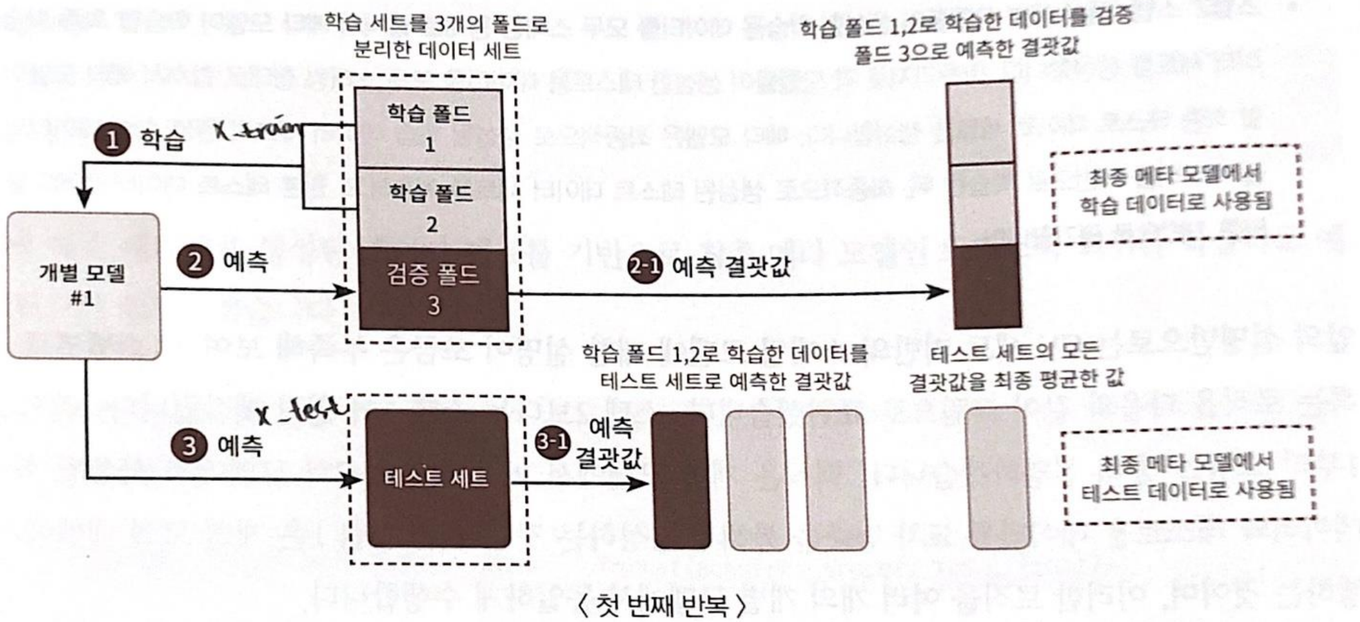
★ 개별 모델의 예측 데이터를 스태킹 형태로 결합하여 학습용, 테스트용 데이터세트를 만드는 것이 핵심!!



7.2 CV 세트 기반의 스태킹

CV 세트 기반의 스태킹

: 과적합 개선을 위해 개별 모델들이 각각 교차 검증으로 메타 모델을 위한 학습용, 테스트용 스태킹 데이터를 생성한다.



1) 스텝1

- 학습용 데이터를 N개의 폴드로 나눔
- N-1개의 폴드를 학습 데이터로 하여 개별 모델 학습

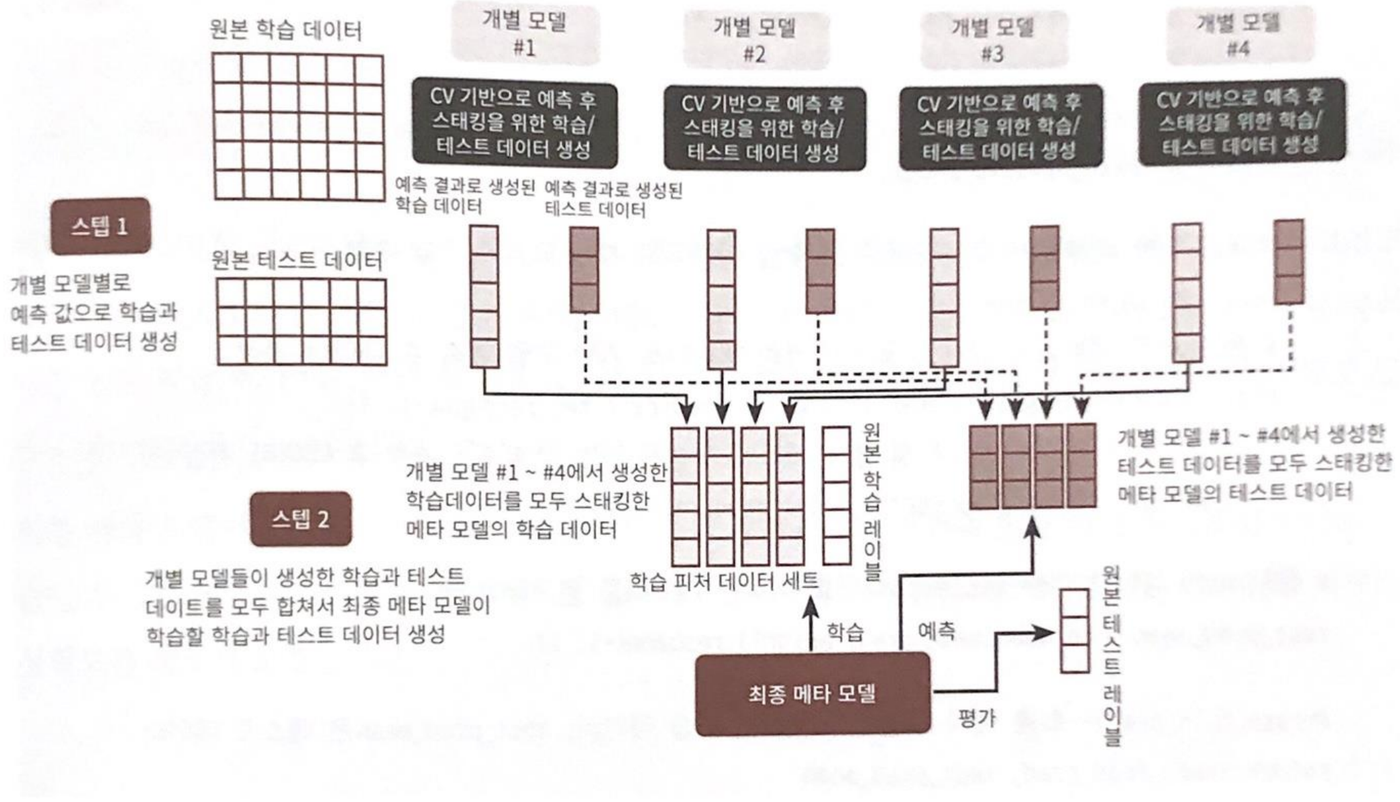
  검증 폴드 1개로 예측한 데이터 → 최종 메타 모델의 학습 데이터

  테스트 데이터를 예측한 데이터 → 최종 메타 모델의 테스트 데이터

7.2 CV 세트 기반의 스태킹

2) 스텝2

- 개별 모델들이 생성한 학습용 데이터를 스태킹 형태로 합침 → 최종 학습용 데이터 세트
- 개별 모델들이 생성한 테스트용 데이터를 스태킹 형태로 합침 → 최종 테스트용 데이터 세트
- 최종 학습용 데이터 세트 + 원본 학습 레이블 데이터로 학습
- 최종 테스트용 데이터 세트를 바탕으로 예측 → 원본 테스트 레이블 데이터로 평가



7.3 적용 – 위스콘신 유방암 예측

1) 스텝1을 진행하기 위한 함수 작성

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error

# 개별 기반 모델에서 최종 메타 모델이 사용할 학습 및 테스트용 데이터를 생성하기 위한 함수.
def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds ):
    # 지정된 n_folds값으로 KFold 생성.
    kf = KFold(n_splits=n_folds, shuffle=False, random_state=0)
    #추후에 메타 모델이 사용할 학습 데이터 반환을 위한 넘파이 배열 초기화
    train_fold_pred = np.zeros((X_train_n.shape[0] ,1 ))
    test_pred = np.zeros((X_test_n.shape[0],n_folds))
    print(model.__class__.__name__ , ' model 시작 ')

    for folder_counter , (train_index, valid_index) in enumerate(kf.split(X_train_n)):
        #입력된 학습 데이터에서 기반 모델이 학습/예측할 폴드 데이터 셋 추출
        print('\t 폴드 세트: ',folder_counter,' 시작 ')
        X_tr = X_train_n[train_index]
        y_tr = y_train_n[train_index]
        X_te = X_train_n[valid_index]

        #폴드 세트 내부에서 다시 만들어진 학습 데이터로 기반 모델의 학습 수행.
        model.fit(X_tr , y_tr)
        #폴드 세트 내부에서 다시 만들어진 검증 데이터로 기반 모델 예측 후 데이터 저장.
        train_fold_pred[valid_index, :] = model.predict(X_te).reshape(-1,1)
        #입력된 원본 테스트 데이터를 폴드 세트내 학습된 기반 모델에서 예측 후 데이터 저장.
        test_pred[:, folder_counter] = model.predict(X_test_n)

    # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
    test_pred_mean = np.mean(test_pred, axis=1).reshape(-1,1)

    #train_fold_pred는 최종 메타 모델이 사용하는 학습 데이터, test_pred_mean은 테스트 데이터
    return train_fold_pred , test_pred_mean
```

2) 개별 모델별로 함수 수행

```
knn_train, knn_test = get_stacking_base_datasets(knn_clf, X_train, y_train, X_test, 7)
rf_train, rf_test = get_stacking_base_datasets(rf_clf, X_train, y_train, X_test, 7)
dt_train, dt_test = get_stacking_base_datasets(dt_clf, X_train, y_train, X_test,  7)
ada_train, ada_test = get_stacking_base_datasets(ada_clf, X_train, y_train, X_test, 7)
```

```
KNeighborsClassifier  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
        폴드 세트:  5  시작
        폴드 세트:  6  시작
RandomForestClassifier  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
        폴드 세트:  5  시작
        폴드 세트:  6  시작
DecisionTreeClassifier  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
        폴드 세트:  5  시작
        폴드 세트:  6  시작
AdaBoostClassifier  model 시작
        폴드 세트:  0  시작
        폴드 세트:  1  시작
        폴드 세트:  2  시작
        폴드 세트:  3  시작
        폴드 세트:  4  시작
        폴드 세트:  5  시작
        폴드 세트:  6  시작
```

7.3 적용 – 위스콘신 유방암 예측

3) 각 모델별로 생성된 최종 학습용 데이터와 최종 테스트용 데이터 합치기

```
Stack_final_X_train = np.concatenate((knn_train, rf_train, dt_train, ada_train), axis=1)
Stack_final_X_test = np.concatenate((knn_test, rf_test, dt_test, ada_test), axis=1)
print('원본 학습 피처 데이터 Shape:',X_train.shape, '원본 테스트 피처 Shape:',X_test.shape)
print('스태킹 학습 피처 데이터 Shape:', Stack_final_X_train.shape,
      '스태킹 테스트 피처 데이터 Shape:',Stack_final_X_test.shape)
```

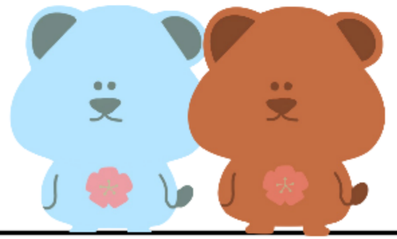
```
원본 학습 피처 데이터 Shape: (455, 30) 원본 테스트 피처 Shape: (114, 30)
스태킹 학습 피처 데이터 Shape: (455, 4) 스태킹 테스트 피처 데이터 Shape: (114, 4)
```

4) 최종 학습용 데이터 + 원본 학습 레이블 데이터로 학습
5) 최종 테스트용 데이터로 예측한 결과를 원본 테스트 레이블 데이터와 비교하여 정확도 측정

```
lr_final.fit(Stack_final_X_train, y_train)
stack_final = lr_final.predict(Stack_final_X_test)

print('최종 메타 모델의 예측 정확도: {0:.4f}'.format(accuracy_score(y_test, stack_final)))
```

```
최종 메타 모델의 예측 정확도: 0.9737
```

THANK YOU



EWHA EURON