Ensemble Model 2
(Boosting)

Jeonghun Yoon

Terms

Weak learner

Boosting

Adaboost

Stagewise Additive Modeling

Gradient boost

XGBoost

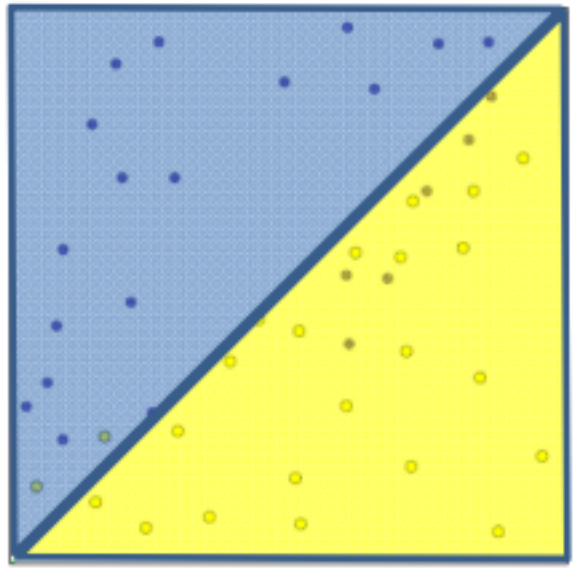
Stacking

Weak learner

Simple learner 또는 weak learner

- Decision stumps : depth가 1인 decision tree
- Shallow decision trees
- Naïve Bayes
- Logistic regression

Weak learner의 분류 결과 예



Bias-variance tradeoff를 다루기 위하여 필요

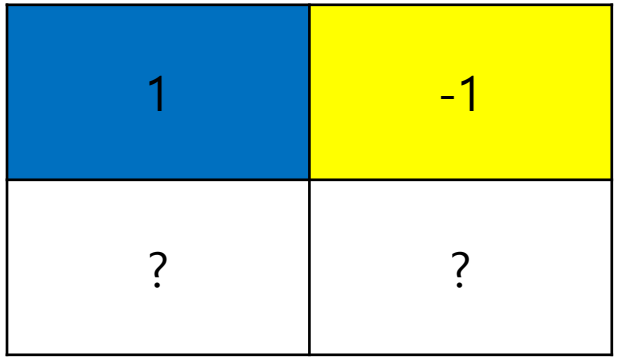
- 낮은 variance를 가지며 거의 overfit 되지 않는다.
- 높은 bias를 가지기 때문에 어려운 학습 문제를 풀수가 없다. (데이터가 복잡한 경우나 모델의 높은 capacity가 필요한 경우)

Ensemble of weak learners

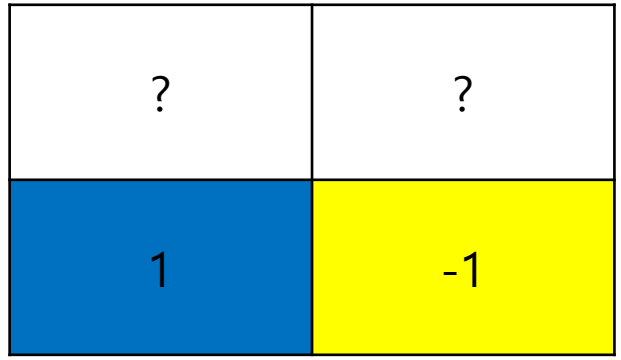
Single weak classifier

다수의 weak classifier를 학습하면, input space의 다른 부분들을 보완해 줄 수 있다.

Weak classifier 1



Weak classifier 2



Bagging vs Boosting

Bagging

- 훈련 데이터에서 다수의 표본을 추출하고, 개별 트리의 결과를 병합하여 단일 예측 모델을 생성
- 각 bootstrap 과정이 독립적이므로 병렬 처리가 가능

Boosting

- Bagging과는 달리 순차적으로 시행되며 bootstrap 과정이 없음
- Original dataset에서 약간의 수정을 거친 데이터를 개별적으로 학습한 후, 결합되어 강력한 분류기 생성

Boosting

Reference : (http://rob.schapire.net/papers/Schapire99c.pdf)

Idea

- Weak learner을 (re-weighted) training data 위에서 여러 번 학습시킨 후, 학습된 classifiers 또는 regressors의 결과를 결합하여(combine) 최종적으로 strong learner를 만든다.
- Classifiers를 결합하는 방법 : (weighted) majority vote
- Regressors를 결합하는 방법 : 평균

Boosting

Weak learner(분류기) 앙상블 내, t번째 분류기 c_t와 $t+1$번째 분류기 c_{t+1}이 **연관성을 가지고 생성**

① X의 샘플을 c_t가 옳게 분류하는 것과 틀리게 분류하는 것으로 분류

② 옳게 분류하는 샘플은 이제 인식이 가능하므로 가중치를 낮춤

③ 틀리게 분류하는 샘플은 아직 "까다로운" 상태이므로 가중치를 높임

④ c_{t+1}를 학습시키기 위한 sample 집합의 resampling 과정에서, 가중치가 높은 샘플이 뽑힐 확률이 높아지는 정책 사용

Resampling 시, replacement를 허용하지 않음.

- 한 번 뽑힌 샘플은 다시 사용하지 않음.

Adaboost

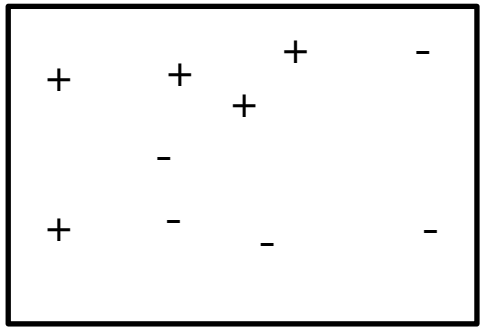
Adaptive **Boost**ing(Adaboost) : 대표적인 Boosting tree이다.

Yoav Freund & Robert Schapire가 제안하였다.

Weak learner를 반복적으로 사용하고, 그 결과를 **합하여** 모델의 accuracy를 향상시킨다.

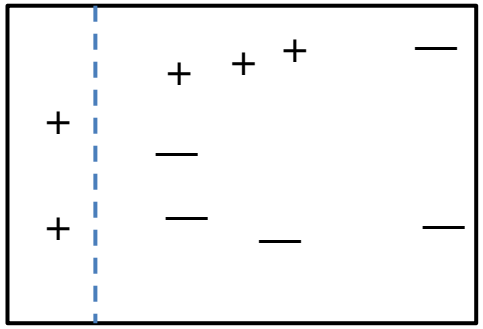
Adaboost 동작 원리

다음과 같이 데이터 셋이 주어졌다고 가정하자.

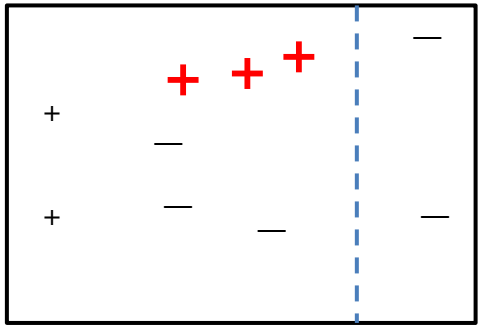


Adaboost 동작 원리

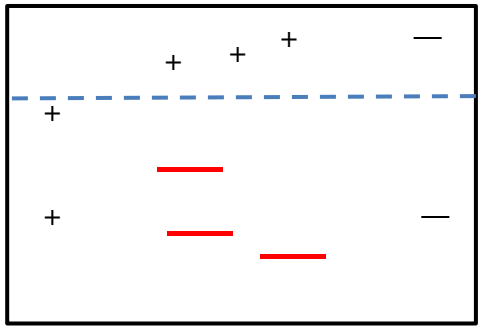
첫 번째 약 분류기(Decision stumps / 결정 크루터기)



두 번째 약 분류기



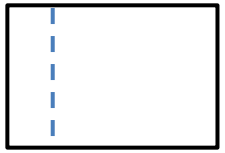
세 번째 약 분류기



오른쪽 3개 + : 오분류

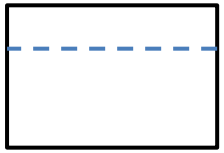
왼쪽 3개 — : 오분류

weight 1 ×

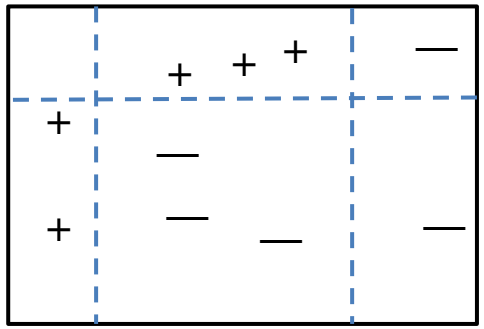


+ weight 2 ×

+ weight 3 ×



=



최종 (강) 분류기

Adaboost 동작 원리

개별 decision stump : $y(\mathbb{x}; \Theta_i)$, 분류기별 중요도 or 가중치 or 예측력 : α_i

최종 강 분류기 : $\hat{y}(\mathbb{x}) = \alpha_1 y(\mathbb{x}; \Theta_1) + \cdots + \alpha_m y(\mathbb{x}; \Theta_m)$

Adaboost의 각 단계에서는 수정된 가중치를 바탕으로 개별 분류기를 훈련시킨다. 이전에 훈련된 분류기의 결과에서 오분류된 데이터 포인트들에 더 높은 가중치를 부여하는 방식으로 가중치를 수정하게 된다. 원하는 수만큼의 개별 분류기를 훈련시키고 나면 이들을 앙상블하여 최종 강 분류기를 생성한다.

Adaboost 알고리즘

입력 : 훈련 집합 $X = \{(\mathbb{x}_1, t_1), (\mathbb{x}_2, t_2), \ldots, (\mathbb{x}_N, t_N)\}$
출력 : 분류기 앙상블(Classifier Ensemble) $\{c_1, \ldots\}$, 분류기 신뢰도 $\{\alpha_1, \ldots\}$

알고리즘 :

$t = 1$, $C = \emptyset$

for (j=1 to N) $w_t(j) = \frac{1}{N}$

repeat {

w_t를 고려하여 분류기 c_t를 학습(가중치가 큰 샘플을 고려하여 학습)

ϵ_t = X에 대하여 c_t가 잘못 분류하는 샘플의 가중치 합을 계산(if $(c_t(\mathbb{x}_i) \neq t_i)$)

if ($\epsilon_t < 0.5$) {

c_t의 신뢰도 $\boldsymbol{\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})}$

for (i=1 to N) {

if ($c_t(\mathbb{x}_i) \neq t_i$) $\boldsymbol{w_{t+1}(i) = w_t(i) * e^{\alpha_t}}$

else $\boldsymbol{w_{t+1}(i) = w_t(i) * e^{-\alpha_t}}$

}

가중치 벡터 $\mathbb{w}_{t+1}$을 정규화

$C = C \cup \{c_i\}$

}

$t = t + 1$

} until (멈춤 조건)

return C

Adaboost 알고리즘

$\mathbb{w}_t$를 고려하여 분류기 c_t를 학습(가중치가 큰 샘플을 고려하여 학습)

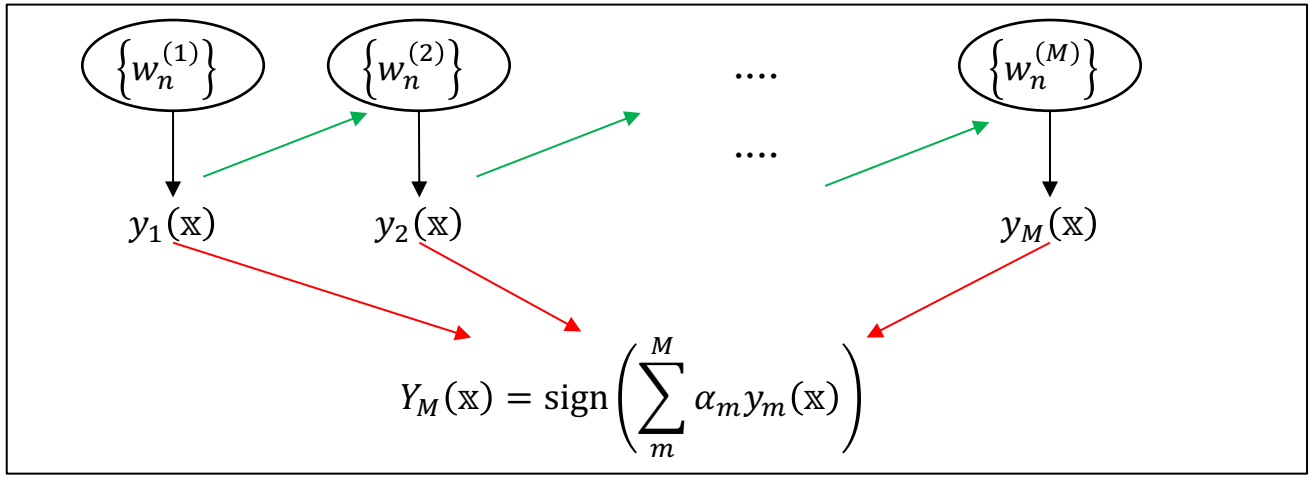
방법 1

- X에서 ρN개의 샘플을 뽑아 X_t라 한다. 이 때 $\mathbb{x}_j$가 뽑힐 확률이 $w_t(j)$가 되도록 한다. replacement는 허용하지 않는다. X_t로 c_t를 학습시킨다.

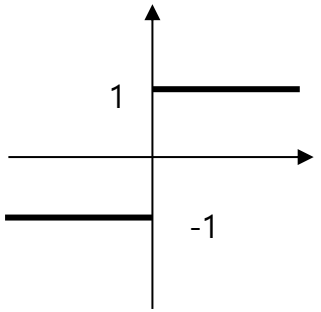
방법 2

- X 전체를 사용한다. $\epsilon_t = \sum_{j=1,\, c_t(\mathbb{x}_j) \neq t_j}^{N} w_t(j)$ 가 최소가 되도록 c_t를 학습시킨다.

Adaboost 알고리즘 (PRML)



$$\text{sign}(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



Adaboost 알고리즘 (PRML)

훈련 데이터는 입력 벡터 $\mathbb{x}_1, \ldots, \mathbb{x}_N$와 해당 binary target 변수가 $t_1, \ldots, t_N \in \{-1,1\}$ 이다.

약 분류기의 개수는 M이다.

- 데이터 가중치 $\{w_n\}$을 $w_n^{(1)} = \frac{1}{n}$으로 초기화한다.
- 개별 분류기의 수 $m = 1, \ldots, M$에 대하여 다음을 시행한다.
  - 다음의 가중 오류 함수를 최소화하는 방식으로 분류기 $y_m(\mathbb{x})$를 훈련 데이터에 피팅한다.

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)$$

여기서 $I(y_m(x_n) \neq t_n)$은 지시 함수(indicator function)로서 $y_m(x_n) \neq t_n$이면 1, 아니면 0 값을 가진다.

Adaboost 알고리즘 (PRML)

- 다음의 값을 계산한다.

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

- 위의 결과를 이용하여 다음을 계산한다.

$$\alpha_m = \ln(\frac{1-\epsilon_t}{\epsilon_t})$$

- 데이터 가중 계수를 업데이트한다.

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(x_n) \neq t_n)\}$$

- 최종 모델을 이용해서 다음과 같이 예측을 시행한다.

$$Y_M(\mathbb{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbb{x})\right)$$

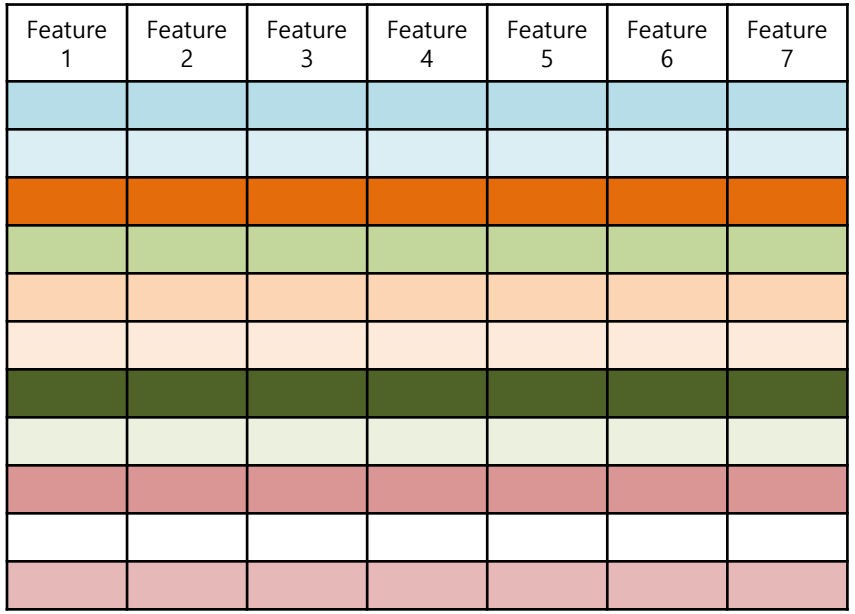
Adaboost

동일
가중값

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7

학습진행
(여러개의 weak learner)

개별 샘플의
가중치가 달라짐



Stagewise Additive Modeling

Boosting은 additive model을 생성한다.

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

여기서 $b(x; \gamma_m)$는 tree이고, γ_m는 node의 split을 나타내는 parameter이다.

Additive model은 통계학에서 자주 볼 수 있다.

- GAMs : $f(x) = \sum_m f_m(x_m)$
- Basis expansions : $f(x) = \sum_{m=1}^{M} \theta_m h_m(x)$

보통 f_m, θ_m들은 최소자승법lesat square, 최대우도법maximum likelihood처럼 함께 학습된다(단순선형회귀모형을 생각).

하지만, Boosting은 parameter (β_m, γ_m)를 **stagewise** 방식으로 학습한다.

이 방식은 학습 전체의 처리 속도를 늦추기는 하지만, 모델이 덜 빠르게 오버피팅 되는 경향이 있다.

Stagewise 예제*(Advanced)*

Stagewise Lesat Squares

Adaboost: Stagewise Modeling

- Boosting을 간단하게 해석하는 방법
  - 지수 오류의 최소화 (PRML 14.3.1 참고)
  - 부스팅의 오류 함수 (PRML 14.3.1 참고)

What is Gradient Boosting

Stochastic Gradient Descent

- SGD를 이용하여 모델(=함수, F) 을 optimization 할 때, 모델의 구조는 고정되어 있다. 즉, SGD를 이용하여 optimization 하려는 것은 모델의 매개변수$_{\text{parameter}}$이다.
  - θs of Logistic regression
  - θs of Linear regression
- $F(x|\Theta) = \min_{\Theta} Loss\big(y, F(x|\Theta)\big)$

Gradient Boosting

- GB는 모델의 구조가 고정되어 있다고 가정하지 않는다. GB는 데이터에 가장 근접한 함수를 찾는 것이다.
- $F(x|\Theta) = \min_{F,\Theta} Loss\big(y, F(x|\Theta)\big)$
- Best parameter와 best function을 찾는 것이 목표이다. → hard to solve

SGD는 1개의 복잡한 모델을 학습하지만, GB는 여러개의 간단한 모델의 ensemble을 학습한다.

What is Gradient Boosting

Gradient Boosting Machine(https://statweb.stanford.edu/~jhf/ftp/trebst.pdf)은 Friedman에 의하여 1999년도에 소개되었다. MART(Multiple Additive Regression Trees) 그리고 GRBT(Gradient Boosted Regression Trees)로도 알려져있다.

GB은 boosting 알고리즘이다. 그 기본 원칙대로 약한 분류기를 반복적으로 학습시킨다.

약한 분류기가 이전 학습에서 발견하기 어려웠던 문제성 관측값 즉, 예측이 틀린 관측값에 집중하게 한다.

다른 boosting 기법처럼 모델을 단계적으로 구축해 나가는 것은 같지만 임의의 미분 가능한 손실 함수를 최적화하는 문제로 일반화한 방법이다.

Motivation of Gradient Boosting

게임을 시작해보자!

$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 총 n개의 데이터가 있고, 이 데이터를 이용하여 회귀모형 $F(x)$를 학습하는 프로젝트를 진행한다고 생각해보자. 팀원이 모델 F를 만들었다. 하지만 성능이 그다지 좋지 않다. $F(x_1) = 0.8$의 예측값을 생성한다. 하지만 실제 $y_1 = 0.9$이다. $y_2 = 1.3$인데, $F(x_2) = 1.4$의 값이 나온다. 이 모델의 성능을 향상시켜야 하는데 한가지 제약조건이 있다. 팀원이 만든 모델 F는 절대 건드리지 않고, 모델을 향상시켜야 한다. 어떤 방법이 있을까?

Motivation of Gradient Boosting

솔루션!

모델 F에 별도의 모델을 추가(add)하는 것이다. 즉, $F(x) + h(x)$를 만들고 이 모델을 이용하여 예측하는 것이다.

$$F(x_1) + h(x_1) = y_1$$
$$F(x_2) + h(x_2) = y_2$$
$$\dots$$
$$F(x_n) + h(x_n) = y_n$$

과연, 추가된 모델 h는 완벽하게 우리의 목적을 달성할 수 있을까? 아닐 것이다. 하지만 근사적으로는 달성할 수 있을 것이다.

Motivation of Gradient Boosting

그렇다면 추가된 모델 h는 어떻게 구할 수 있을까?

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \dots, (x_n, y_n - F(x_n))$을 학습하면 된다.

Residual잔차, 오차

Gradient Boosting 기본 원리

학습데이터를 이용하여 정확도 75%(예시)까지 모델을 학습하고, 나머지 미설명 부분은 오차항에 남겨둔다.

- $Y = F(x) +$ **오차항**

오차항을 이용하여 다른 모델을 학습시킨 후, 그 전 모델에서는 미설명 부분이었으나 이번 학습에서는 설명이 되는 부분을 찾아내 원시 모델에 추가한다. 단, 추가는 반드시 전체 정확도를 향상시켜야만 한다.

- **Gradient(오차항)** $= G(x) +$ **오차항2**

모델이 80%(예시)의 정확도를 갖게 되면 식은 다음과 같이 된다.

- $Y = F(x) + G(x) +$ **오차항2**

Gradient Boosting 기본 원리

오차항 2에 이 기법을 다시 한 번 적용해 추가적으로 설명이 가능한 부분을 찾아낸다.

- **Gradient(오차항2)** $= H(x) +$ **오차항3**

모델의 정확도가 85%가 되면 최종 모델 식은 다음과 같다.

- $Y = F(x) + G(x) + H(x) +$ **오차항3**

단순 합보다 가중 평균을 사용하여(다른 모델보다 정확도가 높은 예측 결과를 가진 모델에 더 높은 중요도 점수를 부여) 모델의 정확도를 더 개선할 수 있다.

- $Y = \alpha F(x) + \beta G(x) + \gamma H(x) +$ 오차항4

Gradient Boosting 기본 원리

예제

Gradient Boosted Trees

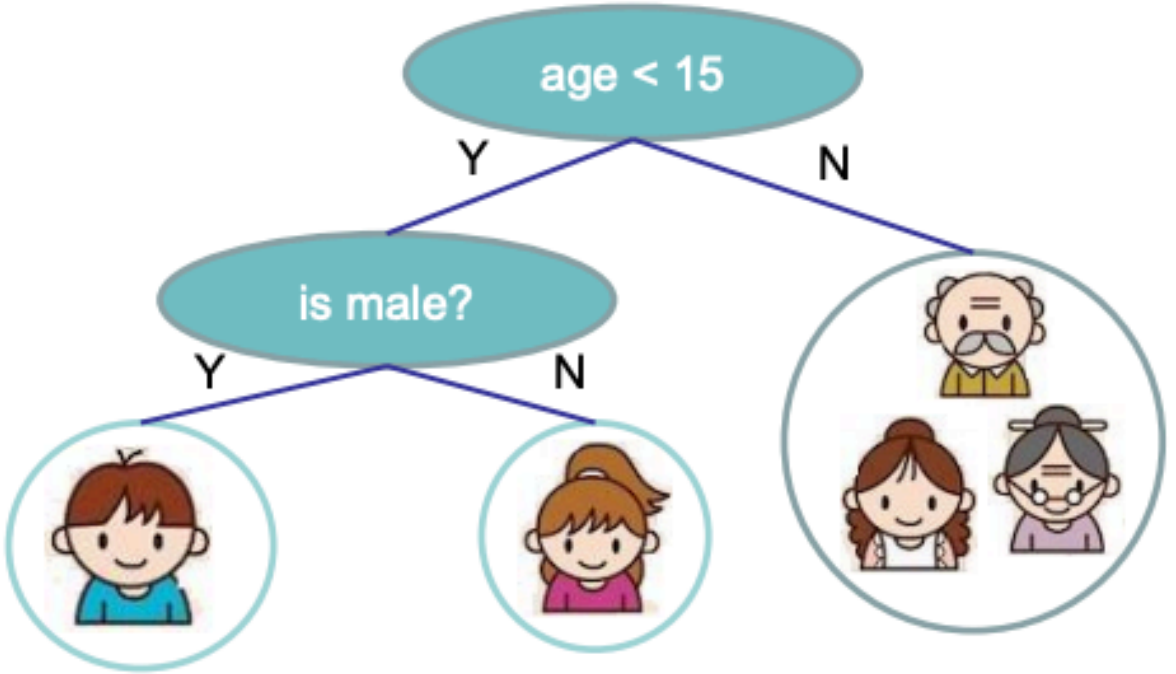
Gradient Boosted Trees의 3가지 요소

- 예측을 수행할 약한 분류기weak learner
  - 약한 분류기로 결정 트리를 사용한다.
- 최적화할 손실 함수 : 손실 함수는 해결하려는 문제에 따라 다르다. 부스팅에서는 처음부터 최적화를 하는 것이 아니라 각 단계별로 이전 단계에서 설명되지 못한 손실에 관해 최적화를 수행한다.
  - 회귀 문제 : Least squares method (최소 자승법)
  - 분류 문제 : Log loss function (로그 손실 함수)
- 손실 함수를 최소화하기 위해 약한 분류기를 추가할 가법 모델additive model
  - 기존 트리는 변동이 없고 새로운 트리가 하나씩 추가된다.
  - 기울기 하강 절차가 사용되어 트리가 추가될 때의 손실을 최소화한다.

Gradient Boosted Trees

Input: age, gender, occupation, …

Does the person like computer games

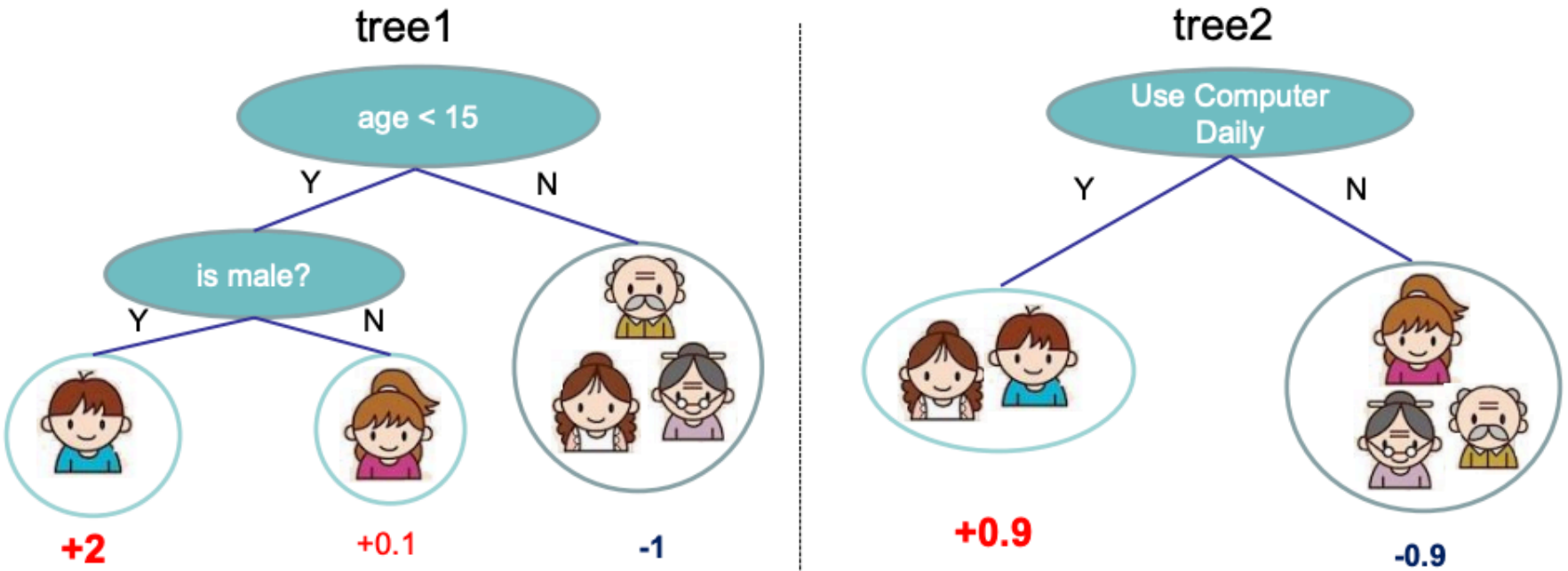


prediction score in each leaf →

$$\mathbb{w} = [2, 0.1, -1]$$

https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Gradient Boosted Trees



f( ) = 2 + 0.9= 2.9 f( )= -1 - 0.9= -1.9

Prediction of is sum of scores predicted by each of the tree

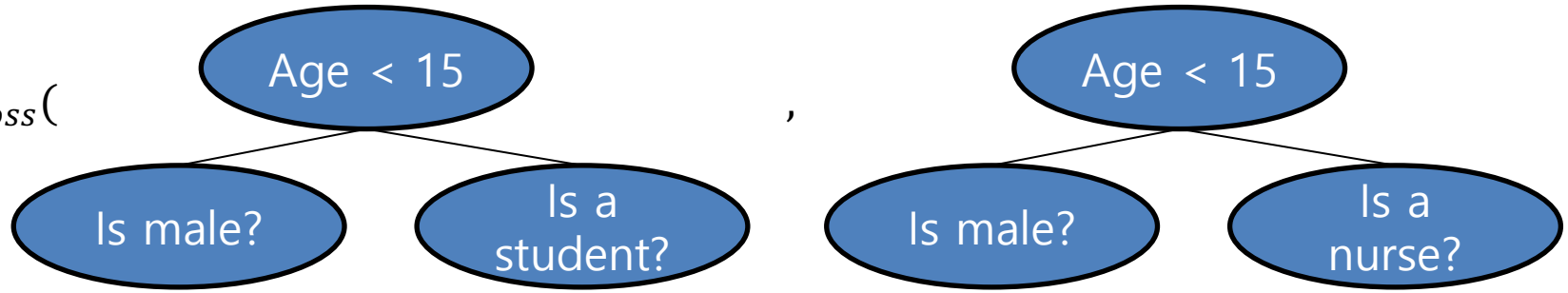
https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Gradient Boosted Trees

Split criterion : Loss function이 최소가 되는 splitting

- 분류 / 회귀 : Sklearn에서는 (friedman) mse를 사용

New tree = $\min_{Loss}$(



,

….)

Gradient Boosting Algorithm

Negative gradient : $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$

초기값은 $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$ 한다.

수렴할때까지 다음을 반복한다.

- Negative gradient $-g(x_i)$를 계산한다.
- $-g(x_i)$를 타겟변수로 Regression tree h를 학습한다.
- $F := F + \rho h$로 업데이트한다. ρ는 가중치이다.

Adaboost vs Gradient Boosting

Adaboost	Gradient Boosting
두 방법 모두 약한 학습자를 기본으로 반복적으로 예측하기 힘들었던 문제성 관측값에 더 집중해 성능을 부스트하는 방법이다. 마지막에는 여러 약한 학습자가(가중값에 의해) 합쳐져 하나의 강한 학습자가 만들어진다.	
이전에 잘못 분류된 관측값에 더 높은 가중값을 부여해 전환이 일어나게 한다.	이전 반복에서 큰 잔차를 남긴 관측값을 찾아낸다.
높은 가중값 데이터 포인트에 의해 단점을 찾아낸다.	기울기를 통해 단점을 찾아낸다.
지수 손실 함수를 통해 적합화 정도가 나쁜 표본에 관해 더 많은 가중값이 부여된다.	기울기 부스팅이 오차항(잔차항)을 더 분해해 더 많은 설명이 가능하다.
기울기 부스팅의 특별한 경우로 지수 손실함수를 사용한다.	기울기의 개념이 보다 더 일반적이다.

XGBoost (극단 기울기 부스팅)

XGBoost 는 2014년 티앤치 첸에 의해 개발된 gradient boosting 기반의 방법이다.

(https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf)

이 방법은 발표와 동시에 데이터 과학 커뮤니티에 엄청난 반향을 불러일으켰다.

XGBoost는 시스템 최적화와 머신 러닝 원칙을 모두 깊이 고려해 개발한 알고리즘이다.

라이브러리의 목표는 컴퓨터의 계산 능력을 그 한계값까지 사용해 확장 가능하고 포터블하며 정확한 결과를 낼 수 있도록 하는 것이다. (gradient boosting의 가장 빠른 구현체)

XGBoost (극단 기울기 부스팅)

모든 가능한 split에 대하여 loss값을 고려한 후, 새롭게 split 한다. Features의 수가 많다면, 그만큼 고려해야 할 가능한 split의 수도 늘어난다.

XGBoost는 모든 가능한 split을 검색해야하는 비효율성을, features의 분포를 먼저 살펴보고 그들의 정보를 이용하여 검색해야하는 범위를 줄임으로써 극복한다.

LightGBM, CatBoost : Gradient boosting 기법을 사용하는 좋은 라이브러리

앙상블들의 앙상블 (모델 스태킹)

앙상블들의 앙상블 혹은 모델 스태킹$_{stacking}$ 은 서로 다른 모델을 하나의 메타 모델로 합쳐 개별 모델로 독립된 성능을 발휘할 때보다 나은 일반적 성능을 낼 수 있도록 하는 기법이다.

스태킹의 종류

- 서로 다른 모델 형태 간의 앙상블들의 앙상블
- 동일 모델 형태끼리의 앙상블 그러나 다양한 부트스트랩 표본에서 개별적으로 구축

서로 다른 분류기 형태 간의 앙상블들의 앙상블

서로 다른 유형의 분류기(예를 들어, 로지스틱 회귀, 결정 트리, 랜덤 포레스트 등)를 동일 데이터 집합에 적합화하고 결과는

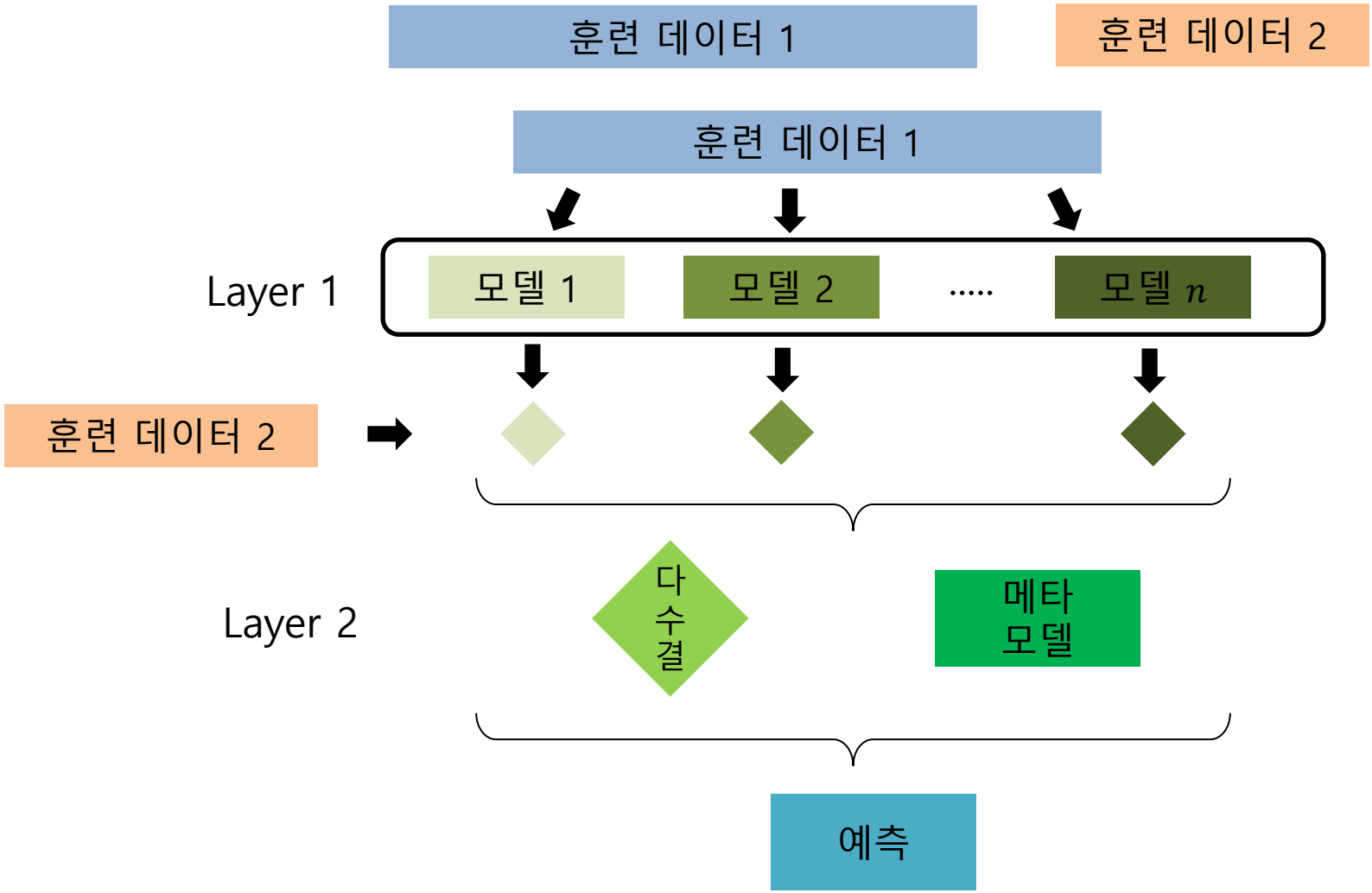
- **다수결이나 평균화**를 통해 합쳐진다. 합치는 방법은 분류 문제냐 회귀 문제냐에 달려 있다.
  - 분류 문제 : 최빈도 함수(가장 많이 나온 다수)
  - 회귀 문제 : 실제값을 평균
- **다른 분류기(메타 분류기라고도 한다)를 결과값에 적합**해 앙상블한다.
  - 메타 분류기의 입장에서 보면 X변수는 개별 모델들의 결과이고, Y변수는 앙상블된 최종 결과 0이나 1이 된다.
  - 메타 분류기를 사용하는 것의 장점은 각 분류기에 주어질 가중값을 얻을 수 있다는 것이다.
  - 메타 분류기를 사용하는 것의 단점은 메타 분류기가 불안정하고 유연하지 못하다는 것이다. 즉, 0 또는 1은 단지 결과값일 뿐, 확률과 같은 정확한 민감도를 제공하지 못한다.
- **확률에 관해 메타 분류기 적용**한다.
  - 개별 분류기에서 0와 1의 출력 대신 확률을 구하고 메타 분류기를 확률에 적용한다.

서로 다른 분류기 형태 간의 앙상블들의 앙상블

아래와 같이 구분하고 있지만, 사실 많은 사람들이 두 용어를 같은 의미로 사용

- Hold-out (Blending)
- Out-of-fold (Stacking)
  - Version 1
  - Version 2

Hold-out (Blending)

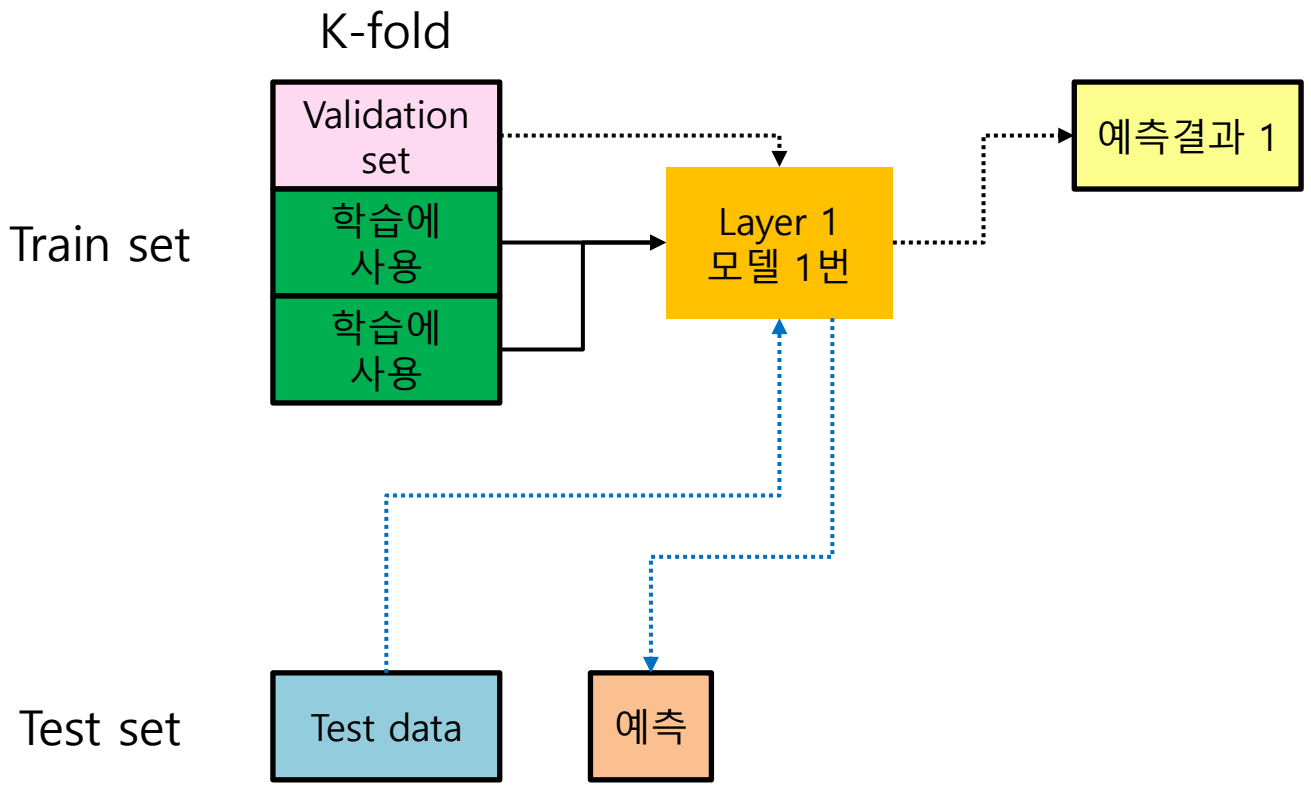


다른 모델에 같은 데이터에 적용

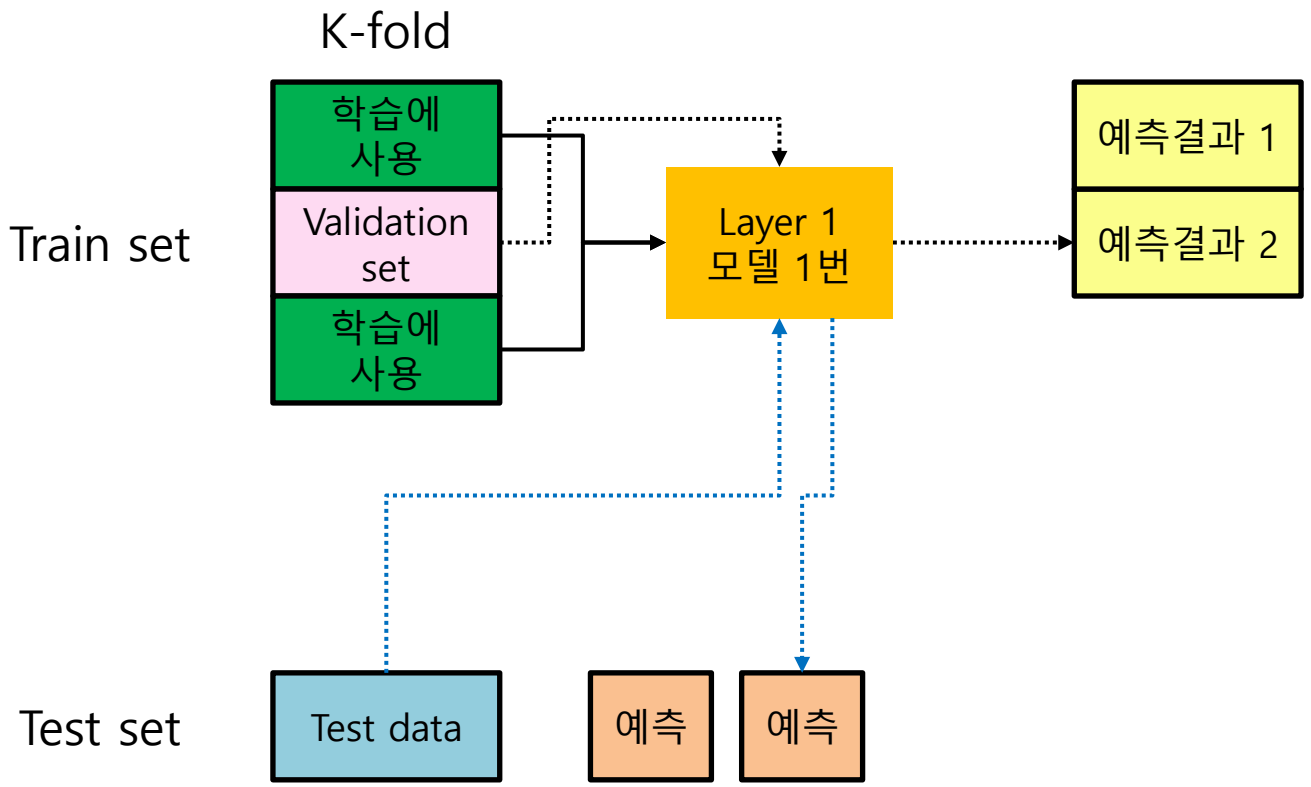
각 모델은 훈련 데이터2에 대하여 각각 예측 시행

다수결이나 메타 모델을 기본 모델들 결과에 적용

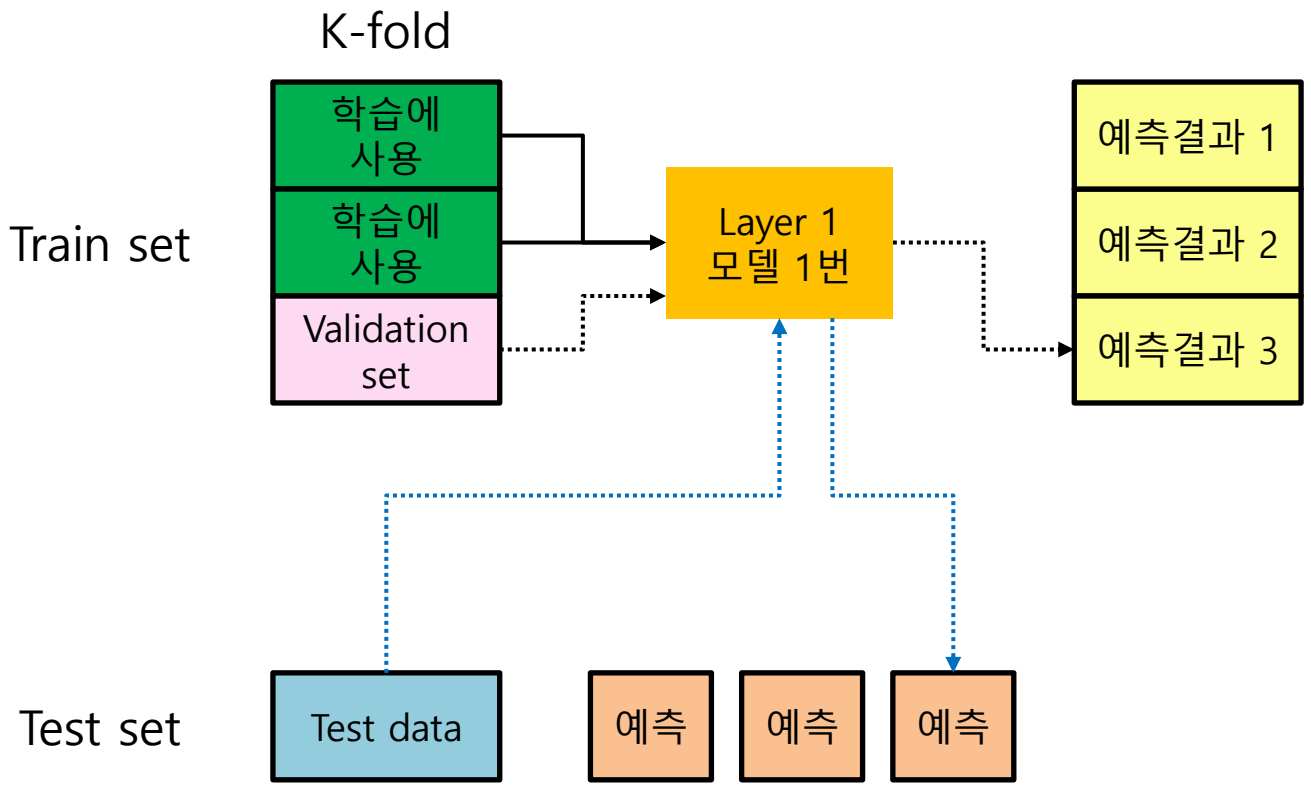
Out-of-fold (Stacking) version 1



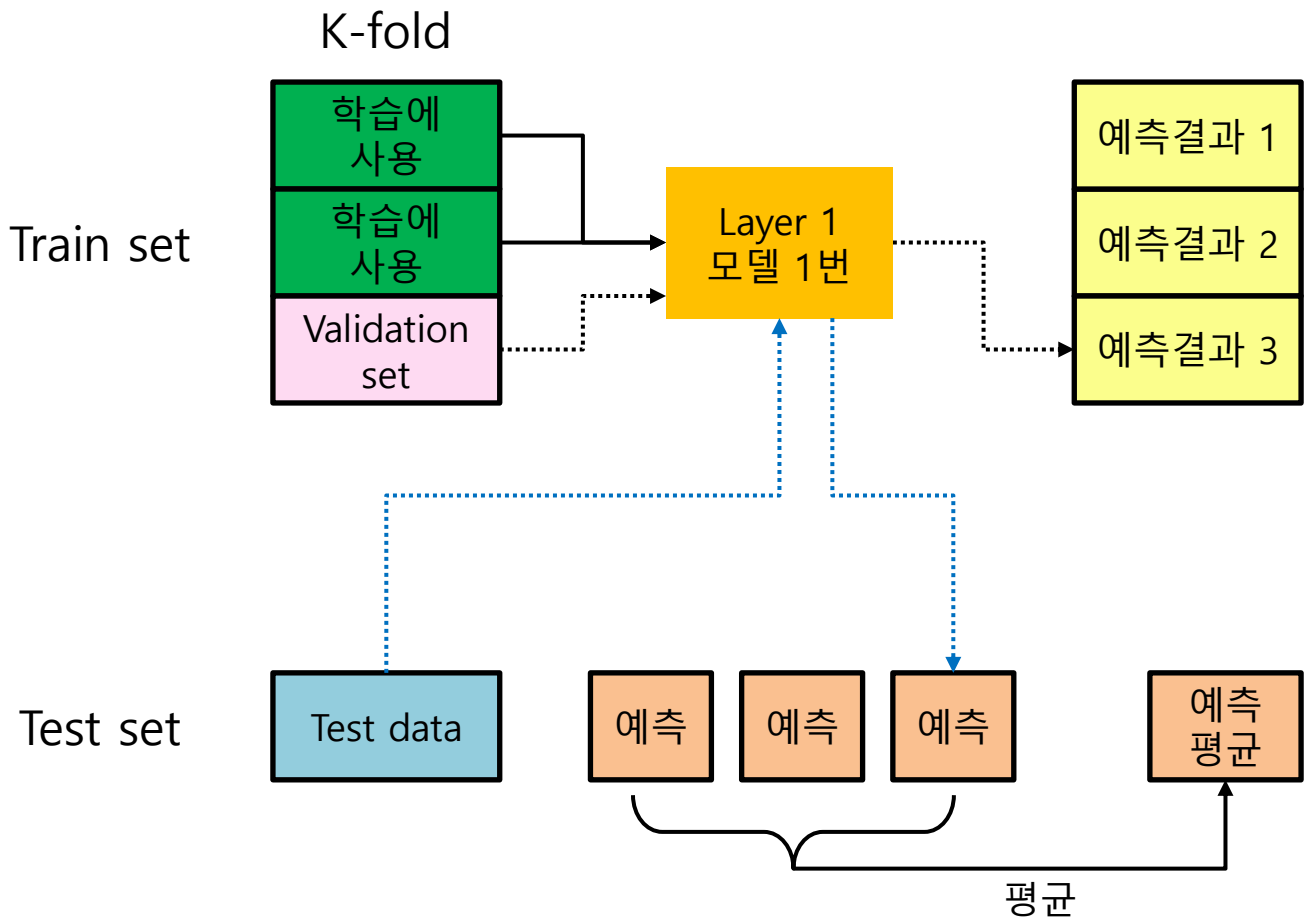
Out-of-fold (Stacking) version 1



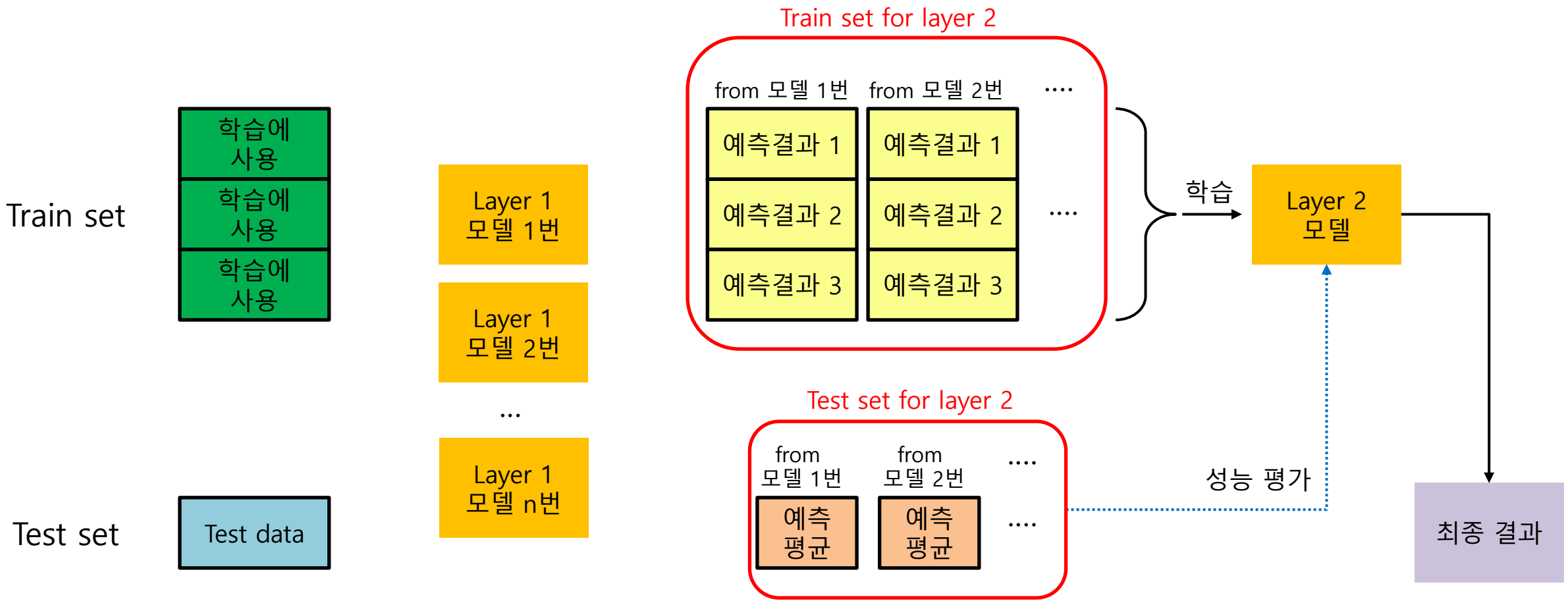
Out-of-fold (Stacking) version 1



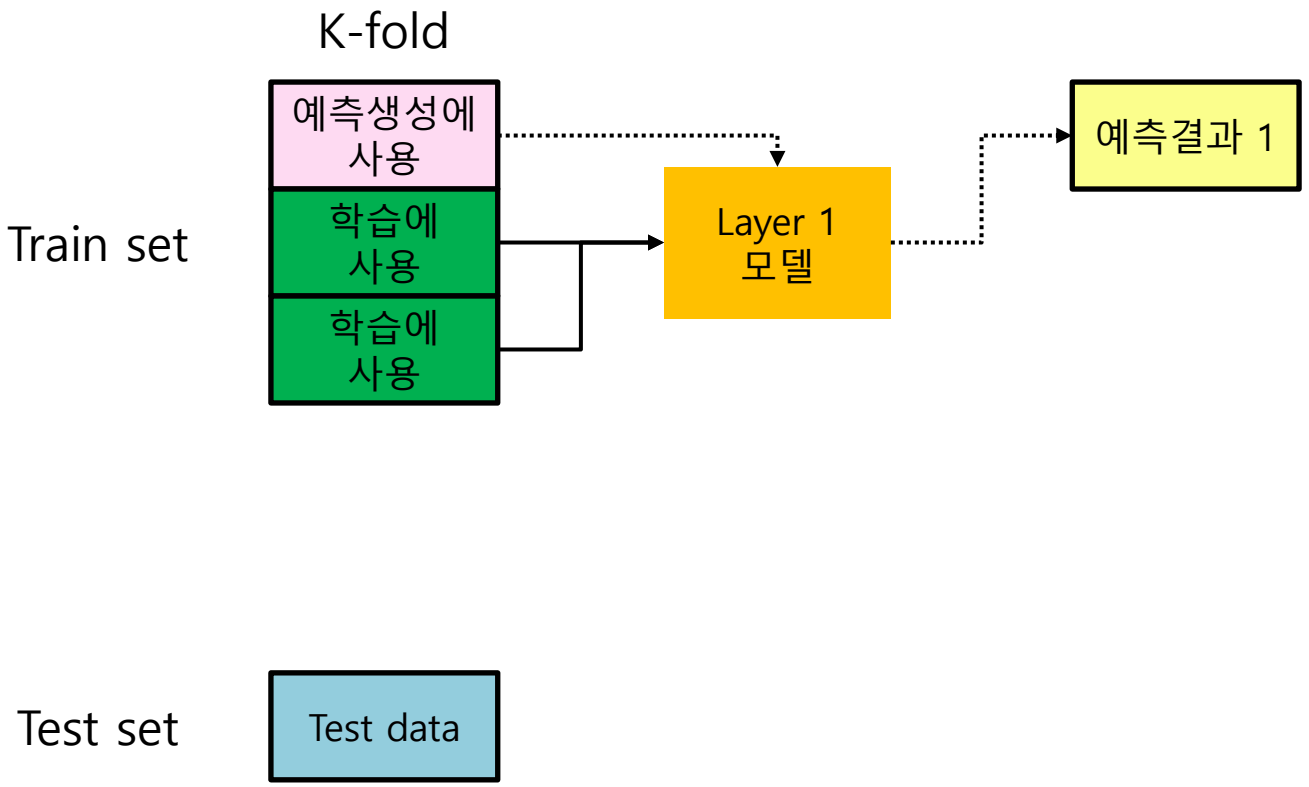
Out-of-fold (Stacking) version 1



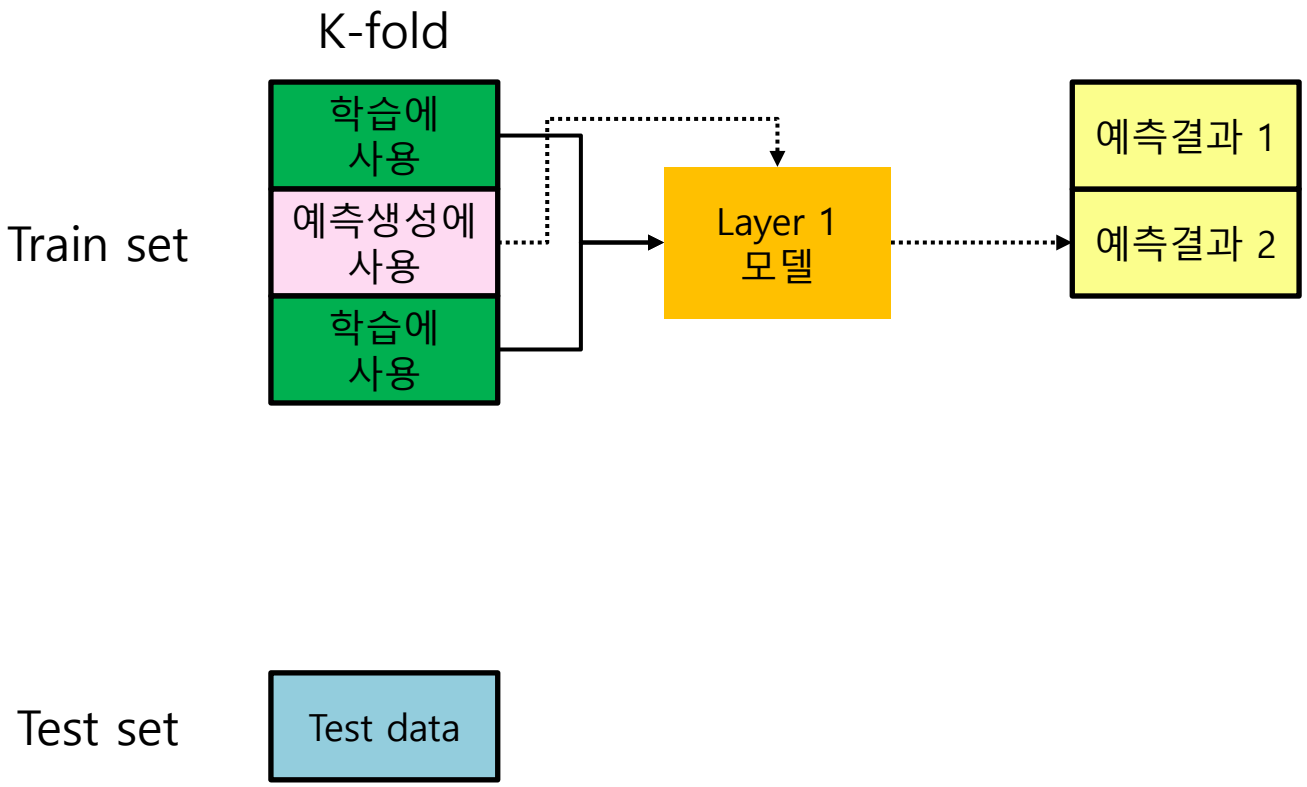
Out-of-fold (Stacking) version 1



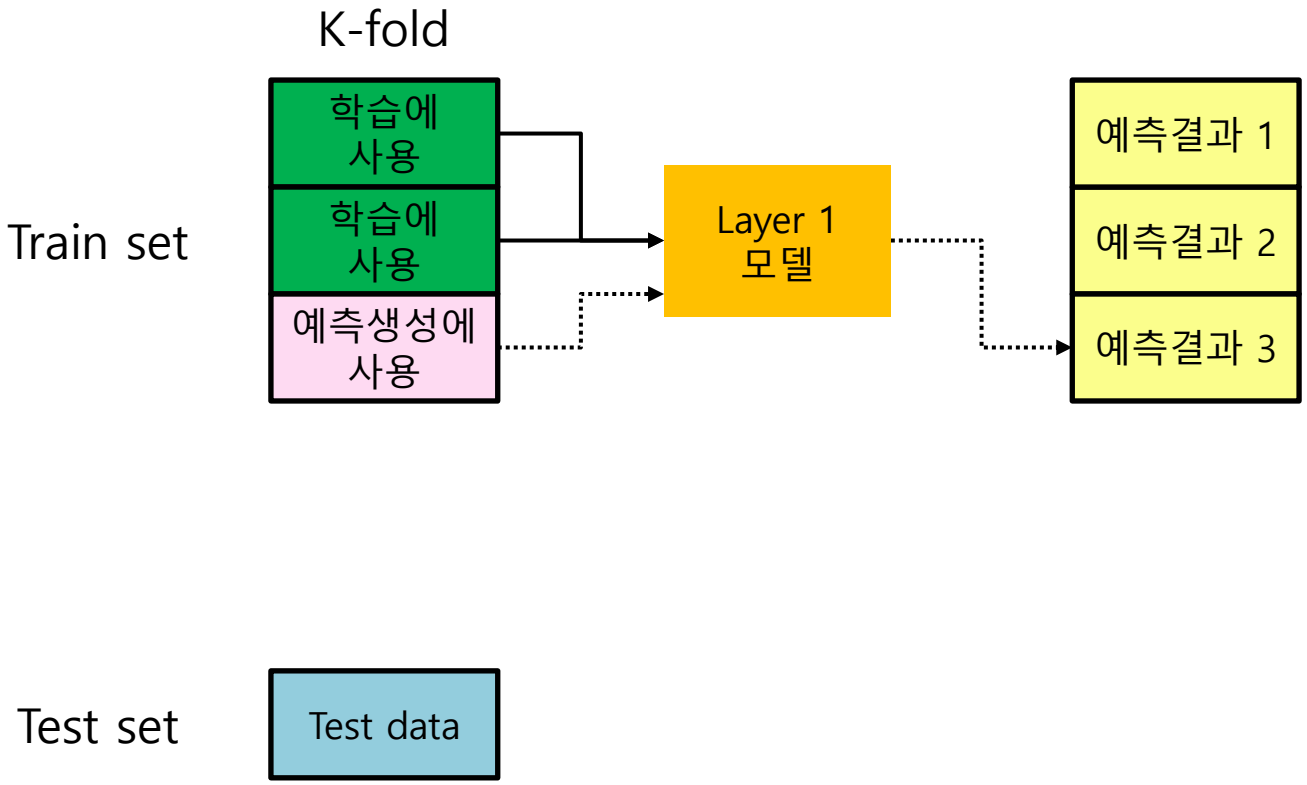
Out-of-fold (Stacking) version 2



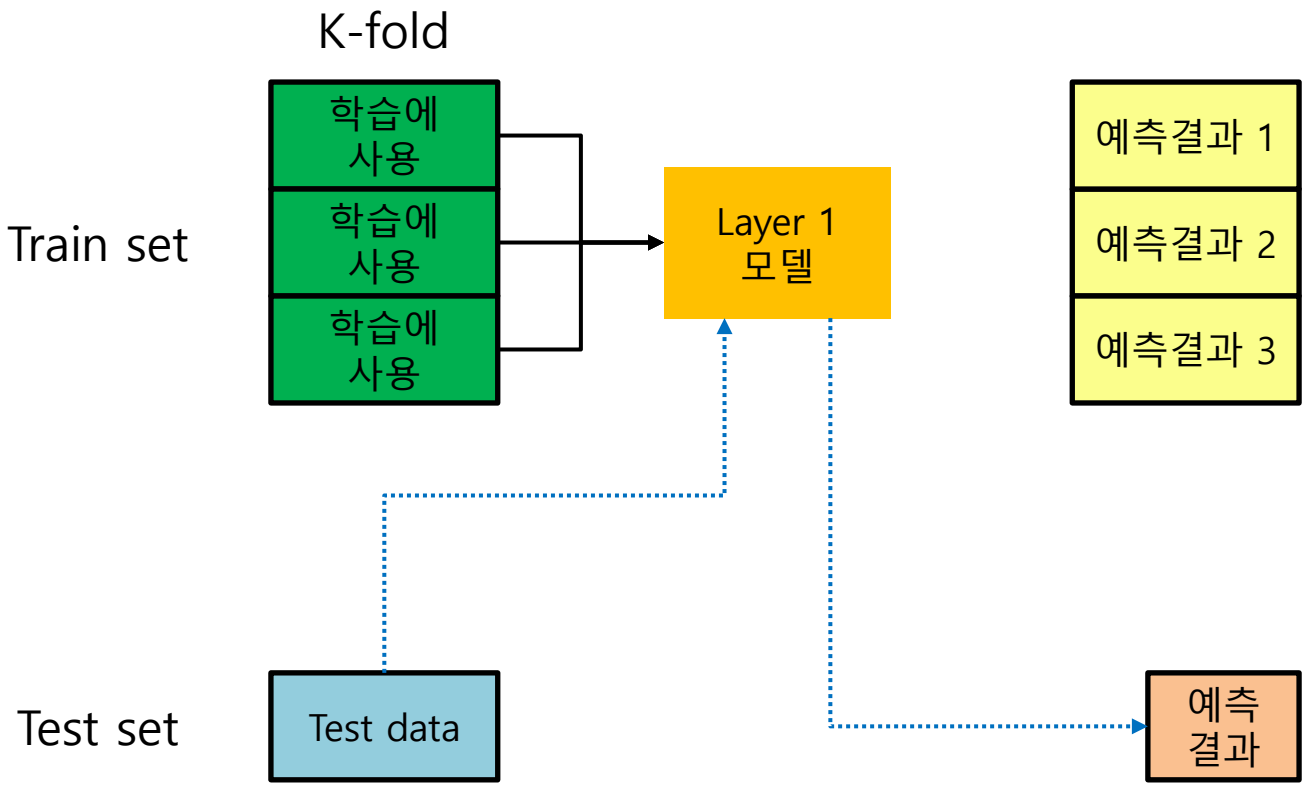
Out-of-fold (Stacking) version 2



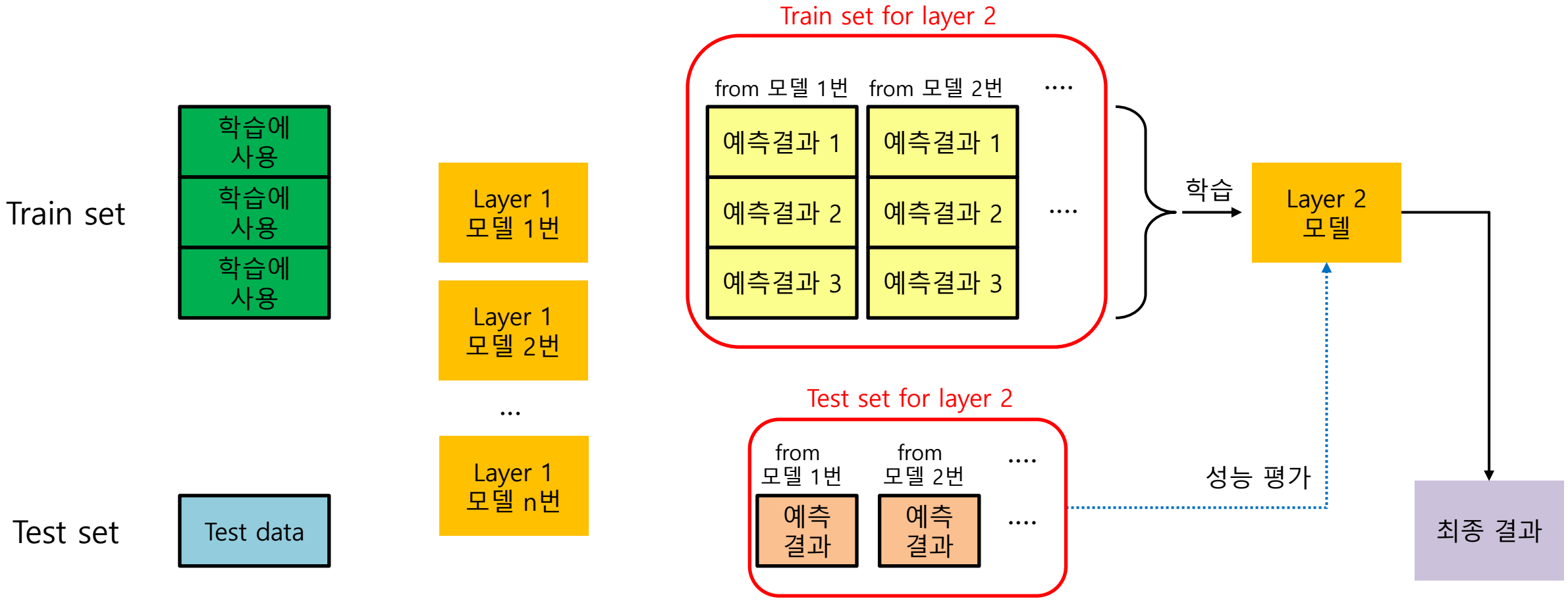
Out-of-fold (Stacking) version 2



Out-of-fold (Stacking) version 2



Out-of-fold (Stacking) version 2

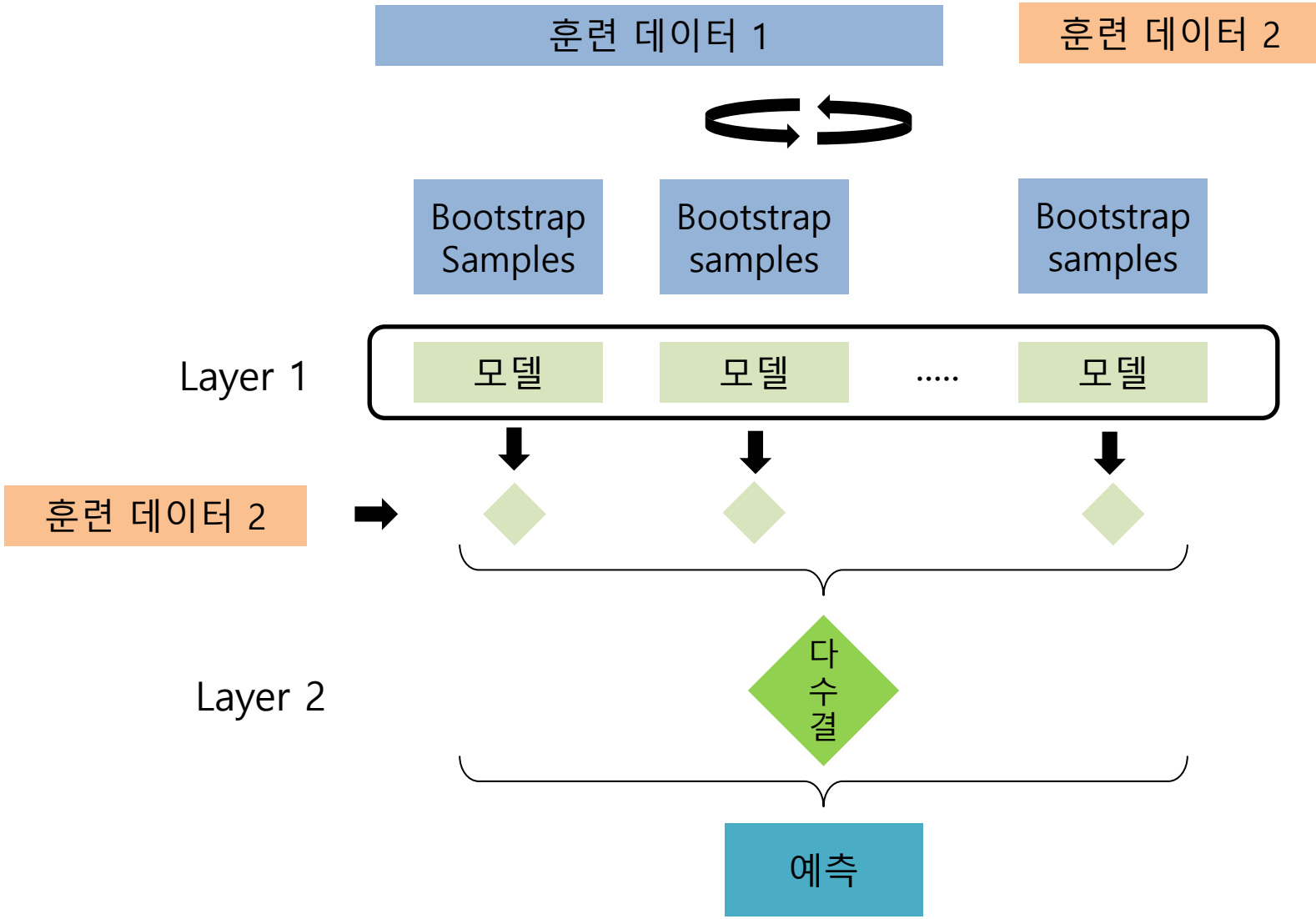


동일 형식 분류기와 부트스트래핑을 이용한 앙상블들의 앙상블

훈련 데이터에서 bootstrap 표본을 추출하고 매번 독립적으로 모델을 적합화한다(개별 모델로는 의사결정 트리나 랜덤 포레스트 등을 사용할 수 있다).

모든 결과는 마지막에 앙상블을 통해 합쳐진다. 이 방법은 여전히 분산의 감소가 여전히 모델의 성능을 향상시킬수 있는 아주 유연한 모델을 다룰 때 적합하다.

동일 형식 분류기와 부트스트래핑을 이용한 앙상블들의 앙상블



표본은 훈련 데이터로부터 복원 임의 추출 (Bootstraping)

동일 형태의 각기 다른 모델(결정 트리)를 각 부표본에 따로 적용

각 분류기는 테스트 데이터에 관해 각각 예측 시행

기본 분류기 최상단에서 다수결에 의해 결정

Appendix

MART

Gradient Boosting Algorithm

Multiple Additive Regression Trees*(Advanced)*

모델

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

여기서 $T(x;\Theta)$ 는 tree이다. Tree $T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I\left(x \in R_j\right)$. Tree의 parameters Θ는 $\{R_j, \gamma_j\}_1^J$이고, I는 지시함수이다.

학습 기준

Stage 1부터 $m-1$까지 이미 학습이 완료되어, tree의 parameter $\hat{\Theta}_1, \ldots, \hat{\Theta}_{m-1}$를 가지고 있는 상황에서 $\hat{\Theta}_m$을 stagewise 최적화를 통하여 학습시키는 기준은 다음과 같다.

$$\arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right)$$

일반적으로 loss function에서는 매우 어려운 최적화 문제이다. Gradient boosting에서는 이 문제를 해결하기 위하여 approximate gradient descent method를 사용한다.

Gradient Boosting Algorithm*(Advanced)*

(Stanford lecture note 참고)

초기화 : $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

$m = 1$ 부터 M까지

- $i = 1, \dots, N$에 관하여 다음을 계산 : $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$
- **타겟이 r_{im}인 회귀 트리를 학습**하여 단말 영역 $R_{jm}, j = 1, 2, \dots, J_m$을 생성

} 모델의 예측력 증가

- $j = 1, \dots, J_m$에 관하여 다음을 계산 : $r_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$

} Additive model

- Update : $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

출력 : $\hat{f}(x) = f_M(x)$