

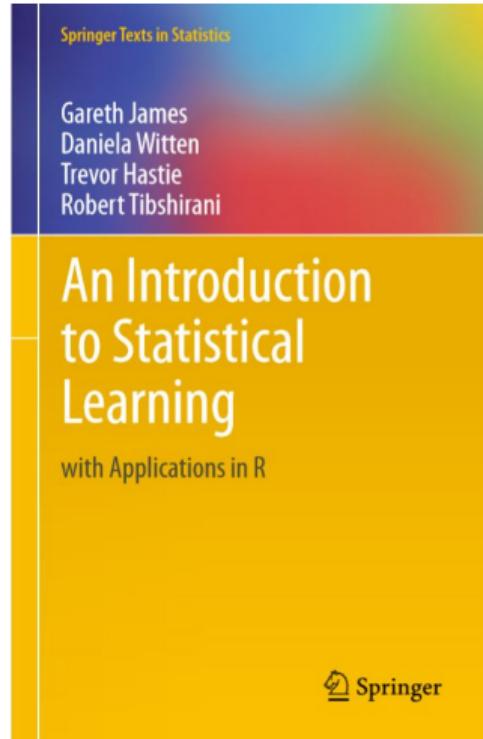
# 깊게 배우는 머신러닝

## Ch.8 Tree-Based Methods

훈 러닝 (Hun Learning)

January 20, 2020

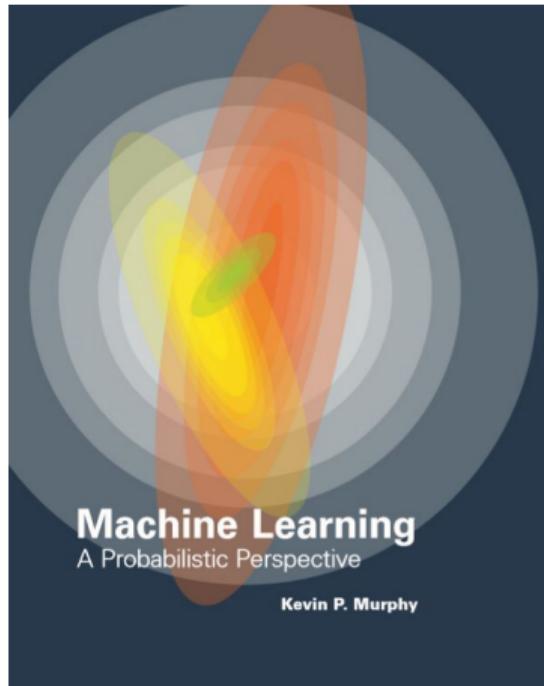
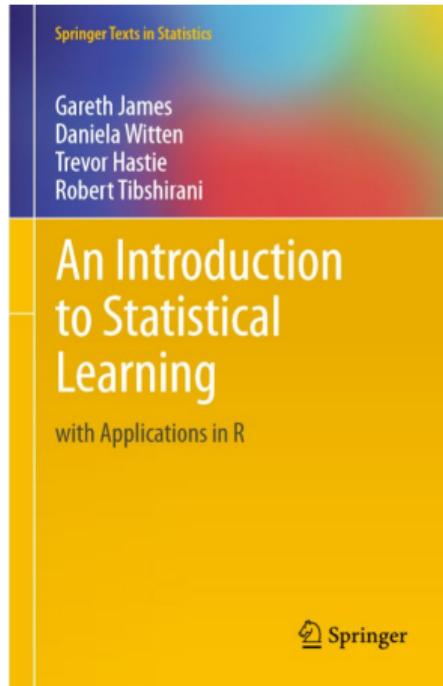
# TEXTBOOK



- **An Introduction to Statistical Learning : with Applications in R**
- 목차:
  - ① Intro
  - ② Statistical Learning
  - ③ Linear Regression
  - ④ Classification
  - ⑤ Resampling Methods
  - ⑥ Linear Model Selection and Regularization
  - ⑦ Moving Beyond Linearity
  - ⑧ Tree-based Methods
  - ⑨ Support Vector Machines
  - ⑩ Unsupervised Learning

# TEXTBOOK

ISL 교재 내용이 많이 빈약하다.. 그래서 다른 책도 참고. 근데 이거 진짜 엄청 어렵다!



**Adaptive Basis Model:**

저번 시간에 배웠던 Step Function, Regression Spline, GAM은 모두 다음과 같은 형태를 취한다.

$$\text{Adaptive Basis Model: } E[y|X] = f(X) = w_0 + \sum_{m=1}^M w_m \phi_m(X)$$

- $X \in \mathcal{R}^{N \times p}$ : N 관측치, p개의 feature (predictor, 설명변수)
- $\phi_m(X)$ :  $X$ 의 m번째 Basis function. 함수 형태를 정해주면 데이터를 통해 학습(fitting)한다. 대개의 경우 학습이란 모수  $V_m$ 의 추정을 의미.

$$\phi_m(X) := \phi(X; V_m)$$

- 모델의 전체 모수  $\theta$  = 상수항 + Basis f마다 모수들 + Basis f의 계수

$$\theta := (w_0, \{w\}_{i=1}^M, \{V_m\}_{i=1}^M)$$

### Adaptive Basis Model:

**Adaptive Basis Model:**  $f(X) = w_0 + \sum_{m=1}^M w_m \phi_m(X)$

- 모델의 전체 모수  $\theta =$  상수항 + Basis f마다 모수들 + Basis f의 계수

$$\theta := (w_0, \{w\}_{i=1}^M, \{V_m\}_{i=1}^M)$$

- 위 식에서 보듯 ABM은 일부의 경우를 제외하고는 "모수에 대해서 선형(linear-in-the-parameters)"이 아니기 때문에 모든 모수 공간에 걸쳐(globally) Likelihood(MLE), 혹은 Likelihood와 Prior를 동시에(MAP) 극대화하는  $\hat{\theta}$ 를 구할 수 없음.
- 오늘 배울 **Decision Tree**도 ABM의 한 종류.

# CLASSIFICATION AND REGRESSION TREES (CART)

## BASICS

Decision Tree (CART Model)은 설명변수의 공간을 반복적으로 분할하여, 각각의 영역마다 일련의 local model을 정의하는 방법. 수식으로 나타내면 다음과 같다.

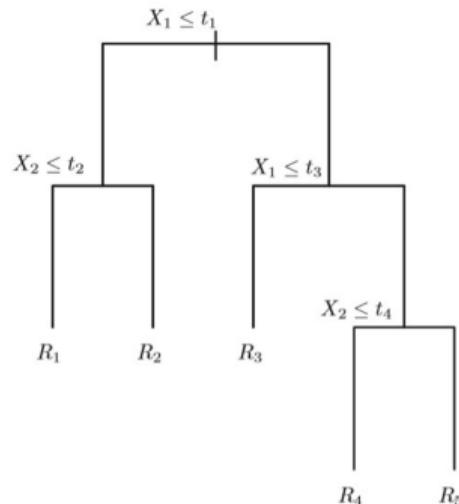
$$E[y|X] = f(X) = \sum_{m=1}^M w_m I(X \in R_m) = \sum_{m=1}^M w_m \phi(X; V_m)$$

- $R_m$ : 분할로 생긴 m번째 영역 (=잎!)
- $w_m$ : m번째 영역에서의 예측값. (Mean response 혹은 최대 범주)
- $V_m$ : m번째 영역(잎)에 이르기까지 각 분기점(위에 있는 잎)에서 쪼개진 변수들과 기준점들의 모음. Basis function의 모수

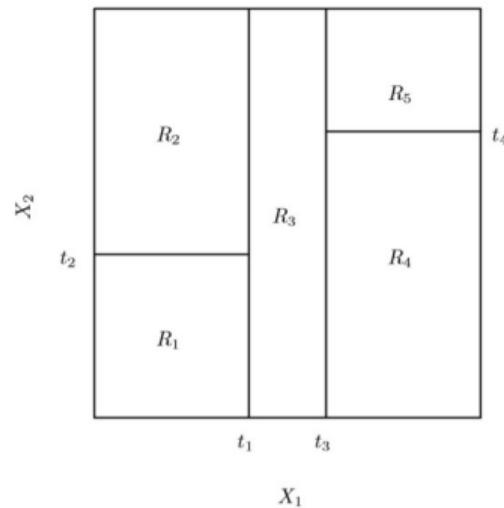
# CLASSIFICATION AND REGRESSION TREES (CART)

## BASICS

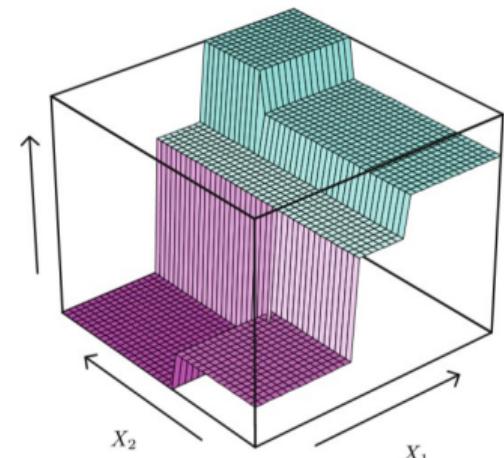
예시: Regression for input space  $(x_1, x_2)$ ,



이 모양으로 Decision Tree를 그리면



Input space를 이렇게 분할하는 것이며



각 영역마다의 예측값을 그리면 이렇게 된다.

# CLASSIFICATION AND REGRESSION TREES (CART)

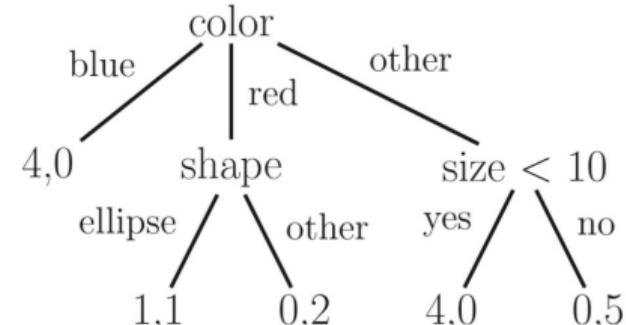
## BASICS

예시: **Classification** for Binary Classification ( $y_i \in \{0, 1\}$ )

- 앞의 (a,b)의 의미는  $y=1|a$ ,  $y=0|b$  가라는 것.
- 이 때 각 영역에서 추정한 Bayes Classifier는

$$p(y = 1|X) = a/(a + b)$$

$$p(y = 0|X) = b/(a + b)$$



# CLASSIFICATION AND REGRESSION TREES (CART)

## GREEDY ALGORITHM

- 최적의 Decision Tree를 짬다는 것은 최적의 영역 분할을 한다는 것. 이는 Best Subset Selection이나 Traveling Salesman과 같은 NP-complete<sup>1</sup>의 문제; 모든 경우의 수를 고려하면 되긴 하지만 그게 현실적으로 불가능하다.
- 때문에 각 노드에서 1) 변수와 2) 기준값을 정할 때 해당 노드에서만 생각 → **greedy algorithm!**
  - ▶ 이게 무슨 말이냐면, 당장은 분할해도 예측값이 별로 좋아지지 않지만 해놓으면 다음 분할 시 예측값이 좋아질 수도 있는 경우를 고려하지 않는다는 것.
- greedy algorithm에서 각 노드에서 최적의 분할 변수  $j^*$ 와 최적의 분할점  $t^*$ 는

$$\begin{aligned}(j^*, t^*) &= \arg \min_{j \in \mathcal{D}} \min_{t \in \mathcal{T}_j} [\ cost(x_i, y_i | x_{ij} \leq t) + cost(x_i, y_i | x_{ij} > t) ] \\ &= \arg \min_{j \in \mathcal{D}} \min_{t \in \mathcal{T}_j} [\ cost(\mathcal{D}_L) + cost(\mathcal{D}_R) ]\end{aligned}$$

( $\mathcal{D}$ 은 그냥 데이터,  $\mathcal{T}_j$ 은  $j$ 변수가 취할 수 있는 모든 분할 기준점을 의미)

<sup>1</sup>NP-complete 개념은 여기 참조: <https://www.mathsisfun.com/sets/np-complete.html>

# CLASSIFICATION AND REGRESSION TREES (CART)

## GREEDY ALGORITHM

- 한 노드에서 다른 노드를 더 팔까 결정은 (**tree의 depth**) 다음과 같이 한다.

---

**Algorithm 16.1:** Recursive procedure to grow a classification/ regression tree

---

```
1 function fitTree(node, D, depth) ;
2   node.prediction = mean( $y_i : i \in D$ ) // or class label distribution ;
3    $(j^*, t^*, D_L, D_R) = \text{split}(D)$ ;
4   if not worthSplitting(depth, cost, DL, DR) then
5     return node
6   else
7     node.test =  $\lambda x. x_{j^*} < t^*$  // anonymous function;
8     node.left = fitTree(node, DL, depth+1);
9     node.right = fitTree(node, DR, depth+1);
10    return node;
```

---

- 참 어렵게도 써놨지만 핵심은

- 1)  $D$ 를  $\text{split}(D)$ 로 쪼개는  $(j^*, t^*)$ 를 구하고,
- 2) 쪼갠  $\text{split}(D)$ 의 depth, cost, 양 쪽 분포  $(D_L, D_R)$  등을 고려해 쪼갤만하면 노드 진출!
- 3) 더이상 안 쪼개도 되면 Decision Tree 그리기 끝!

# CLASSIFICATION AND REGRESSION TREES (CART)

## GREEDY ALGORITHM

- 참 어렵게도 써놨지만 핵심은
  - $\mathcal{D}$ 를  $split(\mathcal{D})$ 로 쪼개는  $(j^*, t^*)$ 를 구하,
  - 쪼갠  $split(\mathcal{D})$ 의 depth, cost, 양 쪽 분포  $(\mathcal{D}_L, \mathcal{D}_R)$  등을 고려해 쪼갤만하면 노드 진출!
  - 더이상 안 쪼개도 되면 Decision Tree 그리기 끝! ← 이건 감으로 결정 (Heuristic)

- ① Cost의 감소가 너무 미미하거나

$$\Delta cost := cost(\mathcal{D}) - \left( \frac{|\mathcal{D}_L|}{|\mathcal{D}|} cost(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} cost(\mathcal{D}_R) \right)$$

- ② Tree의 깊이가 너무 깊거나 (Maximum desired depth)  
③ 분할 기준점 양 쪽( $\mathcal{D}_L, \mathcal{D}_R$ )이 그게 그거거나 (sufficiently homogeneous)  
④ 영역(잎) 안에 데이터가 너무 적거나.

이 cost 함수를 어떻게 정의하냐에 따라 Regression / Classification!

# CLASSIFICATION AND REGRESSION TREES (CART)

## COST FUNCTIONS

### Regression Cost

$$cost(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2$$

$(\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i, \text{ 해당 영역에서의 Mean Response})$

### Classification Cost

Decision Tree에서는 각 영역에서의 Bayes Classifier  $\pi_c = p(y = c|X)$ 를 다음과 같이 추정하며,

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} I(y_i = c)$$

영역 내의 값을 가장  $\hat{\pi}_c$ 가 높은 범주  $c$ 로 분류한다. ( $\hat{y}_{\mathcal{D}} = \arg \max_c \hat{\pi}_c$ )

이 때 쓸 수 있는 Error measures는 1) Misclassification rate, 2) Entropy, 3) Gini index.

# CLASSIFICATION AND REGRESSION TREES (CART)

## COST FUNCTIONS

### Classification Cost (총 C개의 범주가 있을 때)

한 영역(노드)  $\mathcal{D}$ 에서의 Error rate (Node impurity)은 다음과 같이 측정한다.

- Misclassification rate:

$$\frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} [I(y_i \neq \hat{y}_{\mathcal{D}})] = 1 - \arg \max_c \hat{\pi}_c$$

- Entropy, or Deviance:

$$\mathbb{H}(\hat{\pi}) = - \sum_{c=1}^C \hat{\pi}_c \log(\hat{\pi}_c)$$

- Gini index:

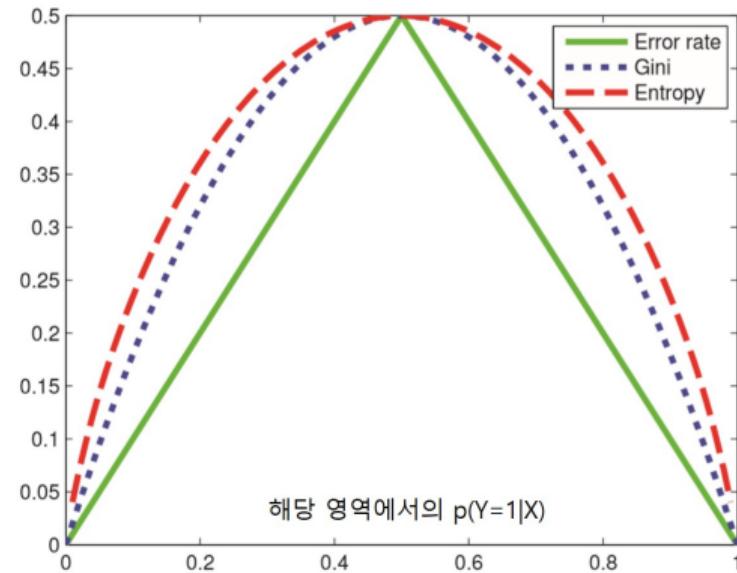
$$\sum_{c=1}^C \hat{\pi}_c (1 - \hat{\pi}_c) = 1 - \sum_{c=1}^C \hat{\pi}_c^2$$

# CLASSIFICATION AND REGRESSION TREES (CART)

## COST FUNCTIONS

### Classification Cost (Binary)

- $1 - \max(p, (1-p)) / H_2(p) / 2p(1-p)$
- Gini와 Entropy가 더욱 'pure'한 노드를 선호
- Ex: 2-case, 400 each.  
한 분할은 (300,100) vs (100,300),  
다른 분할은 (200,400) vs (200,0) 일때,  
Misclassification rate은 0.75로 모두 같으나  
 $(1/2 * 0.25 + 1/2 * 0.25 = 1/3 * 600/800)$   
후자가 더 'pure'하므로 gini/entropy로 계산 시  
후자를 선호.  
(gini:  $3/16 > 1/9$ )



# CLASSIFICATION AND REGRESSION TREES (CART)

## PRUNING

### 예시: iris

```
library('rpart')
irisTree = rpart(Species~Sepal.Length+Sepal.Width, data = iris, method='class', control=rpart.control(cp=0.01, minsplit = 20))

x = seq(min(iris$Sepal.Length), max(iris$Sepal.Length), length=100)
y = seq(min(iris$Sepal.Width), max(iris$Sepal.Width), length=100)
xygrid = expand.grid(x, y)
colnames(xygrid) = c('Sepal.Length', 'Sepal.Width')

irisTreePred = predict(irisTree, xygrid, type='class')
xygrid['Species'] = irisTreePred

windows()
plot(iris$Sepal.Length, iris$Sepal.Width, type='n')
title(main="Species ~ Sepal.Length + Sepal.Width")
points(iris[iris$Species == 'setosa',]$Sepal.Length, iris[iris$Species == 'setosa',]$Sepal.Width, pch=1, col=2)
points(iris[iris$Species == 'versicolor',]$Sepal.Length, iris[iris$Species == 'versicolor',]$Sepal.Width, pch=2, col=3)
points(iris[iris$Species == 'virginica',]$Sepal.Length, iris[iris$Species == 'virginica',]$Sepal.Width, pch=3, col=4)

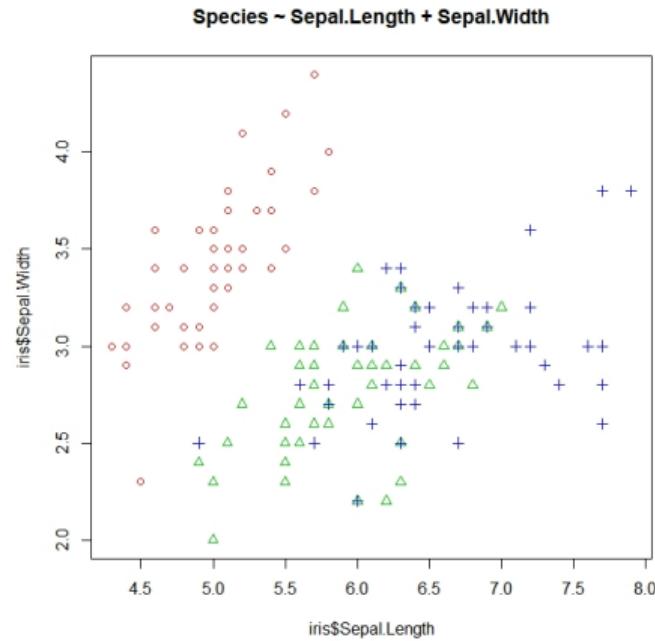
title(main="cp = 0.01, minsplit = 20")
points(xygrid[xygrid$Species == 'setosa',]$Sepal.Length, xygrid[xygrid$Species == 'setosa',]$Sepal.Width, pch=0, col=2)
points(xygrid[xygrid$Species == 'versicolor',]$Sepal.Length, xygrid[xygrid$Species == 'versicolor',]$Sepal.Width, pch=0, col=3)
points(xygrid[xygrid$Species == 'virginica',]$Sepal.Length, xygrid[xygrid$Species == 'virginica',]$Sepal.Width, pch=0, col=4)

windows()
plot(irisTree); title(main="cp = 0.01, minsplit = 20")
text(irisTree,cex = 0.8)
```

# CLASSIFICATION AND REGRESSION TREES (CART)

## PRUNING

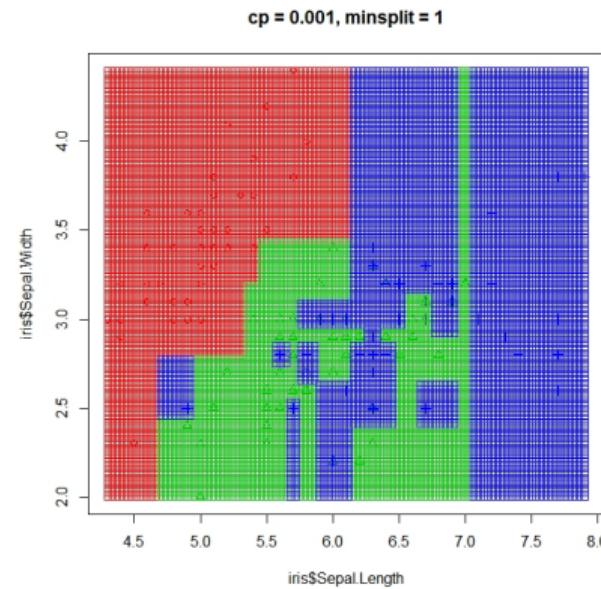
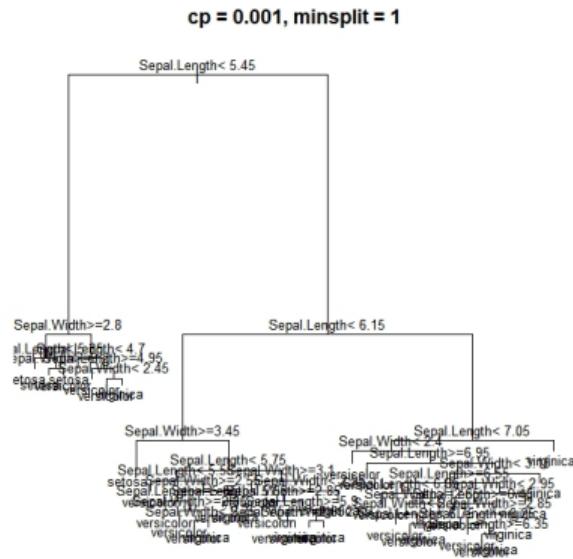
예시: iris



# CLASSIFICATION AND REGRESSION TREES (CART)

## PRUNING

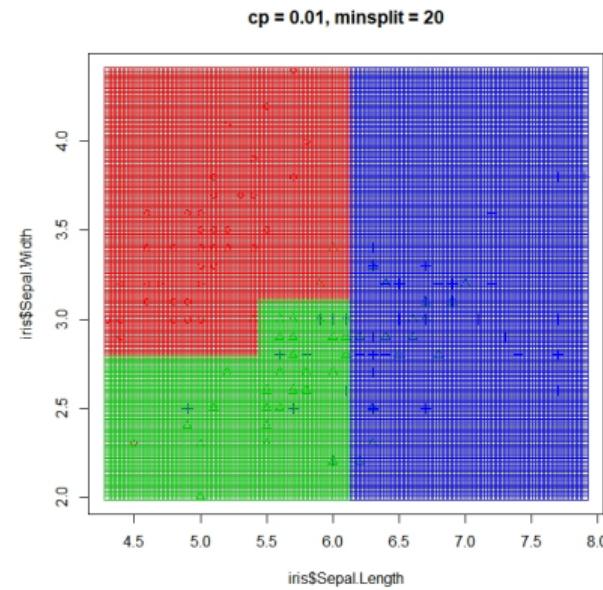
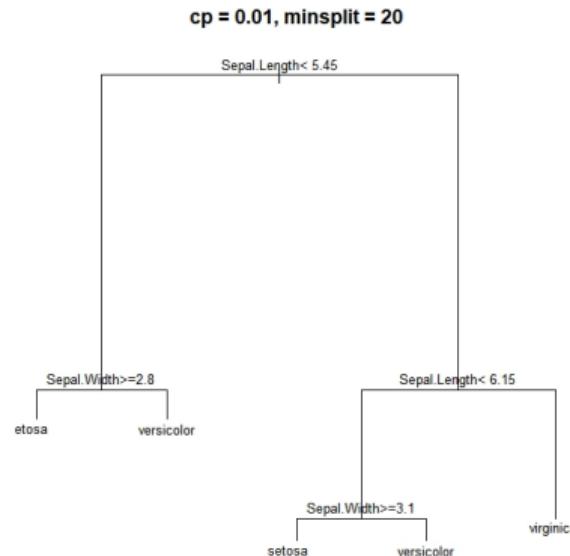
## 예시: iris



# CLASSIFICATION AND REGRESSION TREES (CART)

## PRUNING

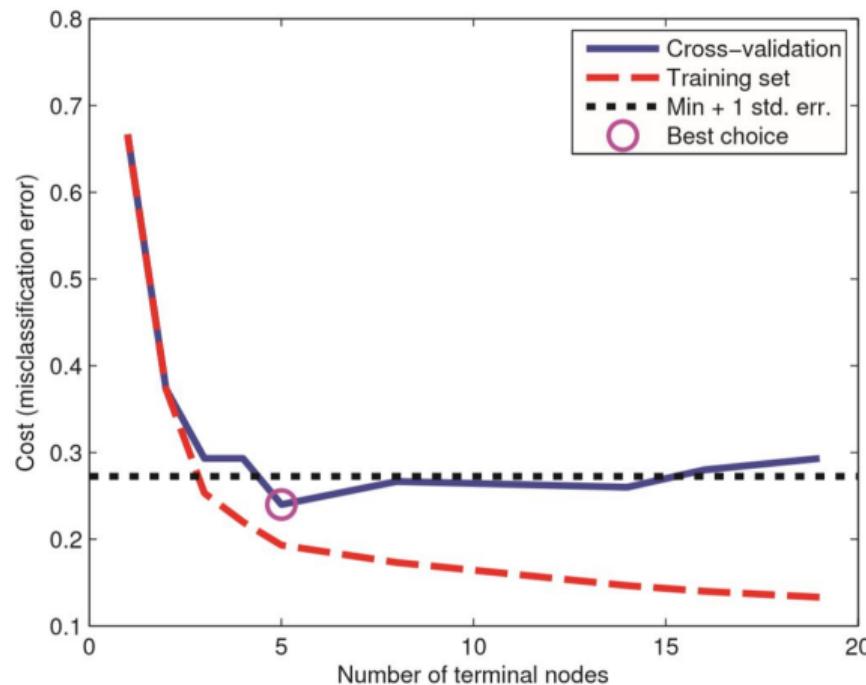
예시: iris



# CLASSIFICATION AND REGRESSION TREES (CART)

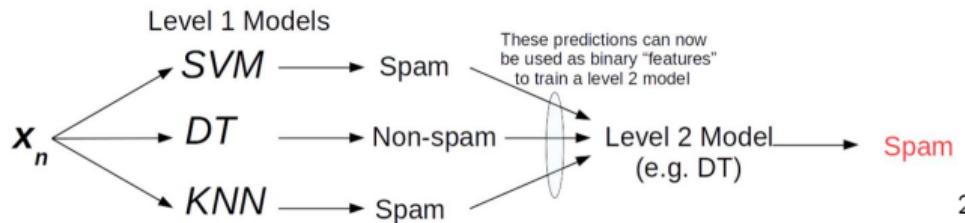
## PRUNING

예시: iris



# ENSEMBLE LEARNING

- Decision Tree는 직관적으로 이해하기 쉬운 장점이 있으나 1) 다른 방법에 비해 예측 정확도가 낮고 2) 데이터에 따라 결과가 많이 바뀌는, 즉 모델의 분산이 크다는 단점이 있다.
- 이러한 단점을 극복하기 위해 여러 개의 Decision Tree Model를 합쳐 하나의 Model로 만들 수 있다. 이처럼 여러 모델을 합쳐 하나의 모델로 만드는 학습 방법을 Ensemble Learning이라고 한다!
- Stacking:** 하나의 데이터에 여러 모델(Lv.1)을 학습해 Lv.2 모델을 만드는 방법.



2

- 반대로 하나의 모델에 여러 데이터를 학습시킬 수도 있다. Decision Tree로 Ensemble Learning을 한다고 할 때, 여러 데이터 셋을 어디서 구하나?

<sup>2</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_autumn16/771A/ec21s/lides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A/ec21s/lides.pdf)

# ENSEMBLE LEARNING

- ① **Bagging:** 주어진 데이터를 Bootstrap해서 데이터의 empirical distribution을 따르는 B개의 새로운 데이터를 만들고, 각각에 Decision Tree 학습
- ② **Random Forest:** Bagging이랑 똑같은데 각 데이터를 학습할 때 predictor 중 랜덤으로 일부는 가려버림!
- ③ **Boosting:** 나무를 1층 그려본다. 그리고 틀린 애들에 가중치를 줘서 다시 그린다. 이걸 계속 반복한 나무들을 다 더한다. (응?)

위 방법들은 Decision Tree 말고도 이론적으로 모든 방법에 적용할 수 있으나, Decision Tree에 쓸 때 가장 성능이 좋다고 한다. 하나씩 알아보자.

# ENSEMBLE LEARNING

## BAGGING

### Bootstrap Aggregation<sup>3</sup>

- ① N개의 관측치로 이루어진 데이터  $\mathcal{D}$ 가 있다고 하자.
- ② Bootstrap 방법으로  $\{\tilde{\mathcal{D}}\}_{m=1}^M$  M개의 sample을 뽑아낸다(각 sample도 N개 관측치). 이럴 경우 각각 데이터  $\tilde{\mathcal{D}}_m$ 는 대략 원래  $\mathcal{D}$ 의 63% 정도만 담고 있으니 reasonably different. (왜?)
  - ▶  $\mathcal{D}$ 의 임의의 관측치가 N번 resampling동안 한 번도 안 뽑힐 확률은  $(1 - \frac{1}{N})^N \rightarrow 0.36$
- ③  $\tilde{\mathcal{D}}_1, \dots, \tilde{\mathcal{D}}_M$ 에다가 decision tree  $h_1, \dots, h_M$ 을 그린다.
- ④ 이걸 다 평균한  $h = \frac{1}{M} \sum h_m$ 을 최종 모델로!  
(Regression이면 단순평균, Classification이면 다수결)

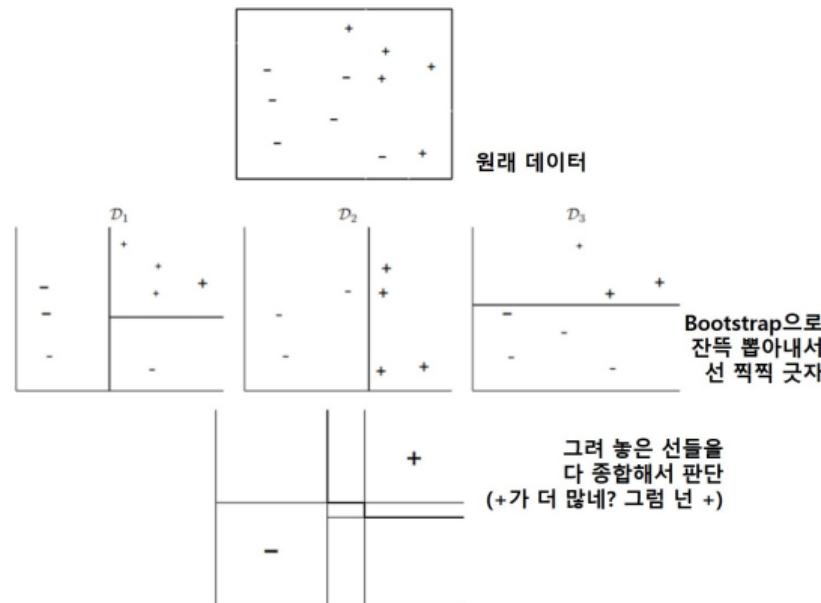
아웃라이어가 많아 분산이 높은 경우에 특효.

<sup>3</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_autumn16/771A/ec21s/lides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A/ec21s/lides.pdf)

# ENSEMBLE LEARNING

## BAGGING

### Bootstrap Aggregation<sup>4</sup>



<sup>4</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_autumn16/771A/ec21s/lides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A/ec21s/lides.pdf)

# ENSEMBLE LEARNING

## RANDOM FOREST

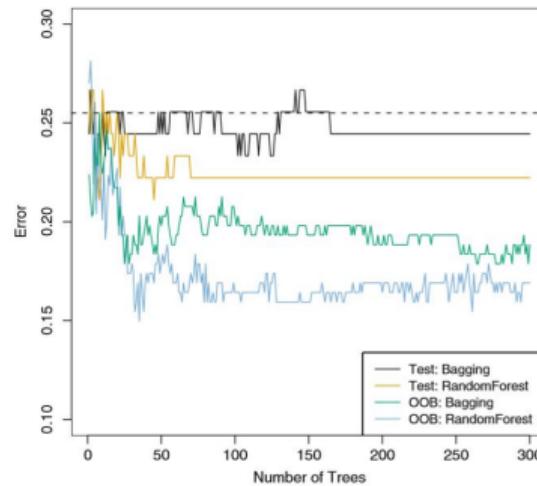
### Random Forest

- ① N개의 관측치로 이루어진 데이터  $\mathcal{D}$ 가 있다고 하자.
- ② Bootstrap 방법으로  $\{\tilde{\mathcal{D}}\}_{m=1}^M$  M개의 sample을 뽑아낸다(각 sample도 N개 관측치). 이럴 경우 각각 데이터  $\tilde{\mathcal{D}}_m$ 는 대략 원래  $\mathcal{D}$ 의 63% 정도만 담고 있으니 reasonably different 하. (왜?)
  - ▶  $\mathcal{D}$ 의 임의의 관측치가 N번 resampling동안 한 번도 안 뽑힐 확률은  $(1 - \frac{1}{N})^N \rightarrow 0.36$
- ③  $\tilde{\mathcal{D}}_1, \dots, \tilde{\mathcal{D}}_M$ 에다가 decision tree  $h_1, \dots, h_M$ 을 그린다.  
이 때 각 샘플마다  $\sqrt{p}$ 개 만큼만 변수를 랜덤으로 뽑아서 그리자. 왜?  
예컨대 하나의 막강한 설명변수가 있다면 Bagging을 해도 나무 모양이 죄다 비슷할 것. 그러면 당연히 모델의 분산이 크다. 때문에 나무 모양을 일부로 (랜덤으로) 다르게 해줘서 나무들을 "de-correlate" 해주는 것!
- ④ 이걸 다 평균한  $h = \frac{1}{M} \sum h_m$ 을 최종 모델로!

# ENSEMBLE LEARNING

## RANDOM FOREST

### Random Forest



**FIGURE 8.8.** Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of  $B$ , the number of bootstrapped training sets used. Random forests were applied with  $m = \sqrt{p}$ . The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is considerably lower.

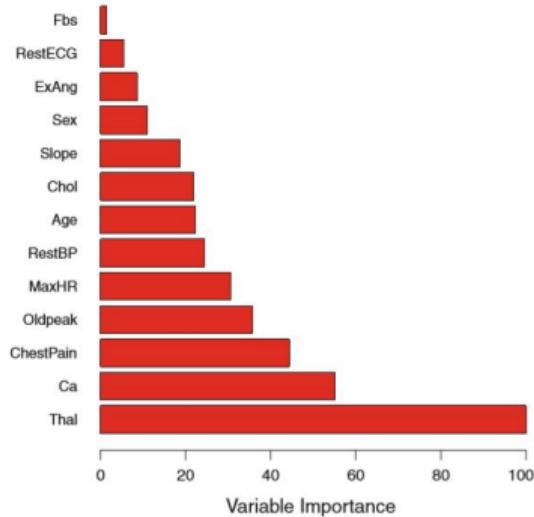
5

<sup>5</sup>OOB: Out-of-Bag Estimates. 각  $\tilde{D}_m$ 마다 여기에 포함 안 된 대략 1/3만큼의 원래 데이터를 test set이라 치고 MSE 계산한 것. test MSE의 추정치로 볼 수 있으며,  $B$ 가 크면 대략 LOOCV와 비슷해진다.

# ENSEMBLE LEARNING

## RANDOM FOREST

### Random Forest



**FIGURE 8.9.** A variable importance plot for the `Heart` data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum. 6

<sup>6</sup>Variance Importance Measure. 특정 변수의 "영향력"을 그 변수에 해당하는 노드들로 인한 RSS (혹은 Error Rate) 변동의 합으로 본 것.

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 직관 얻기<sup>7</sup>

약한 모델들도 계속 쌓이다보면 강해진다!(부스터 온!!)

- 그냥 찍는 거보다 조금 나은 약한 모델(learning algorithm) 아무거나 가져온다.
  - ▶ 예컨대 Binary Classification이면 Random guess는 반반으로 찍는거. 이거보다 조금 나은게  $Y=1$ 의 개수가 좀 더 많으면 그냥 다  $Y=1$ 로 보는 '약한 모델'
- 이 약한 모델이 틀린 부분에 집중해 강하게 키운다.

대부분 Boosting은 이런 식이다.

- ① 약한 모델로 일단 fitting 해보고 이 모델의 전체 에러(곧 모델의 가중치)를 구한다.
- ② 모델이 틀린 부분에 높은 가중치를, 맞춘 부분에는 낮은 가중치를 부여한다.
- ③ 이렇게 가중치가 부여된 관측치에다가 다시 약한 모델을 학습한다.
- ④ 이 과정을 T번 반복한다.

<sup>7</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_autumn16/771A\\_ec21s/slides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A_ec21s/slides.pdf)

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 예시: Adaboost<sup>8</sup>

- ① 주어진 데이터는  $\{(X_i, y_i)\}_{i=1}^N$ , Binary  $y_i \in \{-1, +1\}$
- ② 각 관측치에 가중치를 매김(처음에는 균등하게).  $(X_i, y_i)$ 의 가중치는  $D_1(i) = \frac{1}{N}$
- ③ for(t in 1:T),
  - ▶ 가중치  $D_t(i)$ 로 매겨진 데이터로  $h_t(X) = +1 \text{ or } -1$ 를 학습
  - ▶  $h_t(X)$  전체의 에러를 관측치의 가중평균으로 계산:

$$\epsilon_t = \sum_{i=1}^N D_t(i) I[h_t(X_i) \neq y_i]$$

(가중치 높은 거 또 틀리면 에러가 쑥쑥 올라가요)

- ▶ 요렇게 구한  $\epsilon_t$  가지고  $h_t(X)$ 의 "importance"를 정의:

$$\alpha_t = \frac{1}{2} \log\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

<sup>8</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_autumn16/771A\\_ec21s/slides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A_ec21s/slides.pdf)

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 예시: Adaboost<sup>8</sup>

- ① 주어진 데이터는  $\{(X_i, y_i)\}_{i=1}^N$ , Binary  $y_i \in \{-1, +1\}$
- ② 각 관측치에 가중치를 매김(처음에는 균등하게).  $(X_i, y_i)$ 의 가중치는  $D_1(i) = \frac{1}{N}$
- ③ for( $t$  in 1:T),
  - ▶ 여기서 끝나는게 아님. 채점결과를 토대로 관측치의 가중치를 update:

$$D_{t+1}(i) \propto \begin{cases} D_t(i) \times \exp(-\alpha_t) & \text{if correct: } h_t(X_i) = y_i \\ D_t(i) \times \exp(\alpha_t) & \text{if wrong: } h_t(X_i) \neq y_i \end{cases} = D_t(i) \exp(-\alpha_t h_t(X_i) y_i)$$

- ▶ Normalize:  $D_{t+1}(i) := \frac{D_{t+1}(i)}{\sum_{j=1}^N D_{t+1}(j)}$
- ④ 이렇게해서 구한 "boosted" final model:

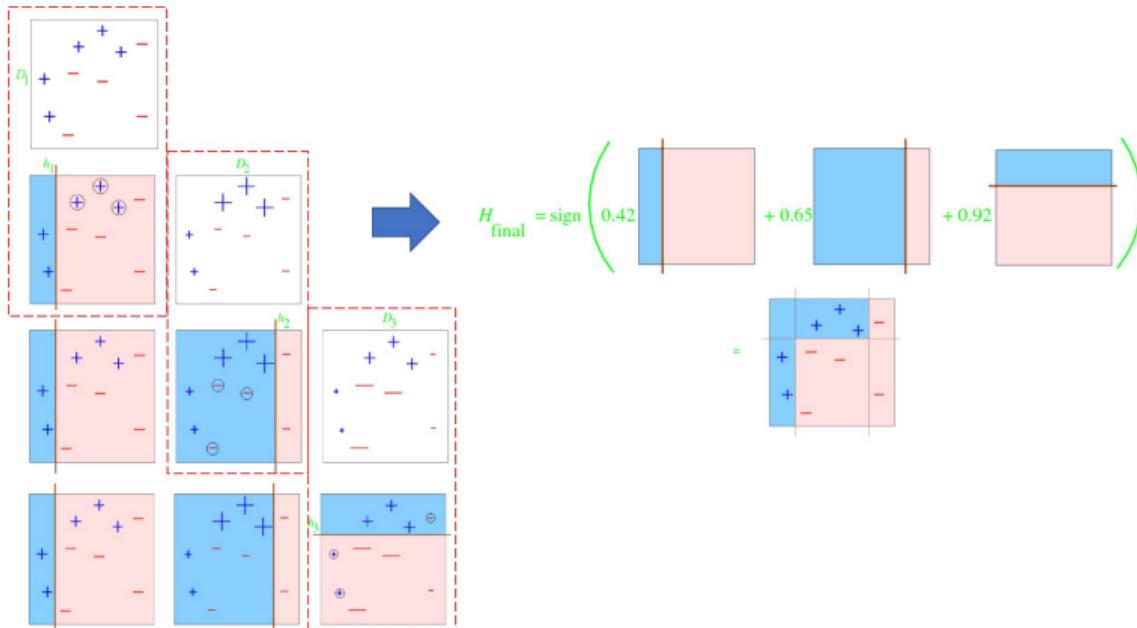
$$H(X) = \operatorname{sgn}\left(\sum_{t=1}^T \alpha_t h_t(X)\right)$$

<sup>8</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_a/utumn16/771A/ec21s/lides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_a/utumn16/771A/ec21s/lides.pdf)

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 예시: Adaboost<sup>9</sup>

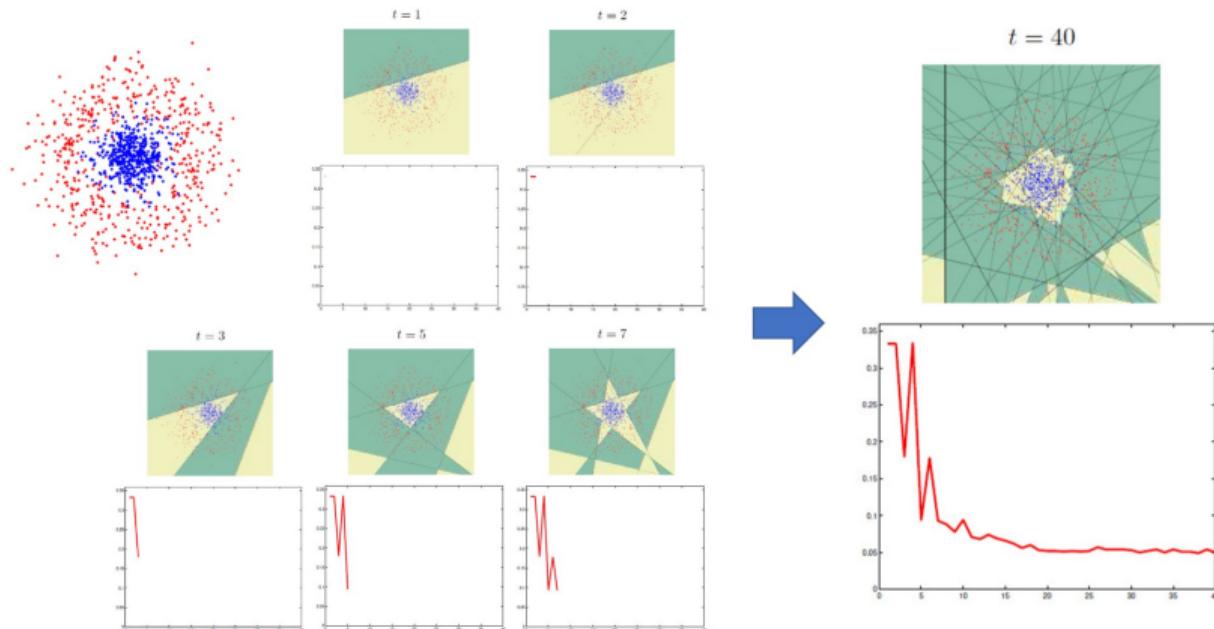


<sup>9</sup><https://cse.iitk.ac.in/users/piyush/courses/mlautumn16/771A/ec21slides.pdf>

# ENSEMBLE LEARNING

## BOOSTING

Boosting 예시: Adaboost<sup>10</sup>



<sup>10</sup>[https://cse.iitk.ac.in/users/piyush/courses/ml\\_autumn16/771A/ec21sides.pdf](https://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A/ec21sides.pdf)

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기

ABM 모델  $f(X_i)$ 을 생각해보자. 우리가 하고싶은 것은

$$\min_f \sum_{i=1}^N L(y_i, f(X_i)) : \text{Find } f \text{ that minimizes loss function } L$$

Name	Loss	Derivative	$f^*$	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y \mathbf{x}_i]$	L2Boosting
Absolute error	$ y_i - f(\mathbf{x}_i) $	$\text{sgn}(y_i - f(\mathbf{x}_i))$	$\text{median}(y \mathbf{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$	$\frac{1}{2} \log \frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2} \log \frac{\pi_i}{1-\pi_i}$	LogitBoost

**Table 16.1** Some commonly used loss functions, their gradients, their population minimizers  $f^*$ , and some algorithms to minimize the loss. For binary classification problems, we assume  $\tilde{y}_i \in \{-1, +1\}$ ,  $y_i \in \{0, 1\}$  and  $\pi_i = \text{sigm}(2f(\mathbf{x}_i))$ . For regression problems, we assume  $y_i \in \mathbb{R}$ . Adapted from (Hastie et al. 2009, p360) and (Buhlmann and Hothorn 2007, p483).

# ENSEMBLE LEARNING

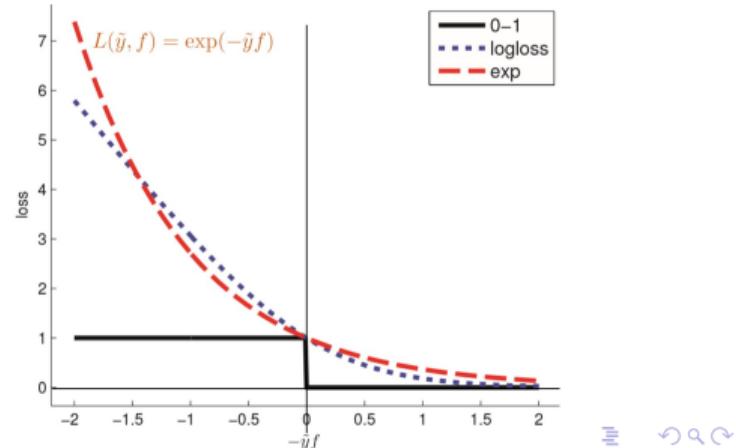
## BOOSTING

### Boosting 자세히 살펴보기

$$\text{목적함수: } \min_f \sum_{i=1}^N L(y_i, f(X_i))$$

- Binary Classification에서 loss function은 당연히 0-1 loss이지만 미분 불가능. 그 대안 중 하나가 **exponential loss**:  $L(\tilde{y}, f) = \exp(-\tilde{y}f)$
- exponential loss로 가정할 경우 목적함수의 해 (population minimizer)는 **log-odds**: (왜?)

$$f^*(X) = \frac{1}{2} \log\left(\frac{p(\tilde{y} = 1|X)}{1 - p(\tilde{y} = 1|X)}\right)$$



# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기

- Population Minimizer는  $E[Loss f|X]$ 를 최소화하는 함수!

$$\begin{aligned}\frac{\partial}{\partial f(\mathbf{x})} \mathbb{E} [e^{-\tilde{y}f(\mathbf{x})} | \mathbf{x}] &= \frac{\partial}{\partial f(\mathbf{x})} [p(\tilde{y} = 1 | \mathbf{x}) e^{-f(\mathbf{x})} + p(\tilde{y} = -1 | \mathbf{x}) e^{f(\mathbf{x})}] \\ &= -p(\tilde{y} = 1 | \mathbf{x}) e^{-f(\mathbf{x})} + p(\tilde{y} = -1 | \mathbf{x}) e^{f(\mathbf{x})} \\ &= 0 \Rightarrow \frac{p(\tilde{y} = 1 | \mathbf{x})}{p(\tilde{y} = -1 | \mathbf{x})} = e^{2f(\mathbf{x})}\end{aligned}$$

$$\rightarrow f^*(\mathbf{x}) = \frac{1}{2} \log \frac{p(\tilde{y} = 1 | \mathbf{x})}{p(\tilde{y} = -1 | \mathbf{x})}$$

- 조건부 분포를 모르니 이걸 직접 찾을 수는 없다. 대신 "sequentially tackle" 하자! → **Boosting!**

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: Forward Stagewise Additive Modeling

$$\text{How to fit ABM: } f(X) = \beta_0 + \sum_{m=1}^M \beta_m \phi(X|\gamma_m)$$

- ① **Initialize:** set  $f_0(X) := \arg \min_{\gamma} \sum_{i=1}^N L(y_i, f(X|\gamma))$

▶ 예컨대 squared loss error이면  $f_0(X) := \bar{y}$ , exponential loss이면  $f_0(X) := \frac{1}{2} \log(\frac{\hat{\pi}}{1-\hat{\pi}})$

- ② **At m's iteration:**

set  $(\beta_m, \gamma_m) := \arg \min_{\gamma, \beta} \sum_{i=1}^N L(y_i, f_{m-1}(X|\gamma) + \beta \phi(X_i|\gamma))$   
set  $f_m(X) := f_{m-1}(X) + \beta_m \phi(X|\gamma_m)$

실제로는 위 식에 shrinkage parameter (learning rate)  $\nu$ 를 추가해(대개 0.1)

$f_m(X) := f_{m-1}(X) + \nu \beta_m \phi(X|\gamma_m)$  // 이 과정을 M번 반복하여 CV 에러의 최저점에서 멈춘다.

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: AdaBoost

Say  $\tilde{y}_i \in \{-1, +1\}$ . At step m, we want to minimize:

$$\begin{aligned} L_m &= \sum_{i=1}^N \exp(-\tilde{y}_i f_m(X)) \\ &= \sum_{i=1}^N \exp(-\tilde{y}_i(f_{m-1}(X) + \beta\phi(X_i))) \\ &= \sum_{i=1}^N w_{i,m} \exp(-\tilde{y}_i \beta\phi(X_i)) \end{aligned}$$

- $w_{i,m} := \exp(-\tilde{y}_i f_{m-1}(X_i))$ , 이전 단계 모델에서 개별 관측치의 loss의 가중치
- $\sum_{i=1}^N w_{i,m} \exp(-\tilde{y}_i \beta\phi(X_i))$ 의 의미는,

가중치가 주어진 관측치에서의 loss를 최소화하는  $\beta\phi(X)$ 를 찾자는 것!

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: AdaBoost

$$L_m = \sum_{i=1}^N w_{i,m} \exp(-\tilde{y}_i \beta \phi(X_i))$$

이때 관측치를 1) m번째 모델이 맞게 예측한 것(correct)과 2) 틀린 것(wrong)으로 나눠보면

$$\exp(-\tilde{y}_i \beta \phi(X_i)) = \begin{cases} \exp(-\beta) & \text{if correct: } \tilde{y}_i = \phi(X_i) \\ \exp(\beta) & \text{if wrong: } \tilde{y}_i \neq \phi(X_i) \end{cases}$$

때문에 목적함수는 다음과 같이 된다.

$$L_m = \sum_{i=1}^N w_{i,m} \exp(-\tilde{y}_i \beta \phi(X_i)) = e^{-\beta} \sum_{\text{correct}} w_{i,m} + e^{\beta} \sum_{\text{wrong}} w_{i,m}$$

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: AdaBoost

$$\begin{aligned}L_m &= \sum_{i=1}^N w_{i,m} \exp(-\tilde{y}_i \beta \phi(X_i)) = e^{-\beta} \sum_{correct} w_{i,m} + e^{\beta} \sum_{wrong} w_{i,m} \\&= e^{-\beta} \left( \sum_{i=1}^N - \sum_{wrong} \right) w_{i,m} + e^{\beta} \sum_{wrong} w_{i,m} \\&= (e^{\beta} - e^{-\beta}) \sum_{wrong} w_{i,m} + e^{-\beta} \sum_{i=1}^N w_{i,m} \\&= (e^{\beta} - e^{-\beta}) \sum_{i=1}^N w_{i,m} I(\tilde{y}_i \neq \phi(X_i)) + e^{-\beta} \sum_{i=1}^N w_{i,m}\end{aligned}$$

## Boosting 자세히 살펴보기: AdaBoost

$$L_m = (e^\beta - e^{-\beta}) \sum_{i=1}^N w_{i,m} I(\tilde{y}_i \neq \phi(X_i)) + e^{-\beta} \sum_{i=1}^N w_{i,m}$$

- 때문에  $\beta$ 가 고정되었을 때 위 식을 최소화하는  $\phi_m(X) = \arg \min_{\phi} w_{i,m} I(\tilde{y}_i \neq \phi(X_i))$
- 즉  $m$  단계에서 이전 단계의 오차에 따라 가중치가 부여된 관측치에서의 Error Rate를 최소화하는 함수를 찾자는 것.
- 그러니까 한 번 틀렸던 거는 가중치를 확확 줘서 또 안 틀리게 하는 거다.

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: AdaBoost

$$L_m = (e^\beta - e^{-\beta}) \sum_{i=1}^N w_{i,m} I(\tilde{y}_i \neq \phi(X_i)) + e^{-\beta} \sum_{i=1}^N w_{i,m}$$

$$\phi_m(X) = \arg \min_{\phi} w_{i,m} I(\tilde{y}_i \neq \phi(X_i))$$

아래 식에서 구한  $\phi_m(X)$ 을 위 식에 대입하면

$$\beta_m = \frac{1}{2} \log\left(\frac{1 - err_m}{err_m}\right)$$

$$0 \text{ 때 } err_m := \frac{\sum_{i=1}^N w_{i,m} I(\tilde{y}_i \neq \phi(X_i))}{\sum_{i=1}^N w_{i,m}}$$

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: AdaBoost

$$\phi_m(X) = \arg \min_{\phi} w_{i,m} I(\tilde{y}_i \neq \phi(X_i))$$

$$\beta_m = \frac{1}{2} \log\left(\frac{1 - err_m}{err_m}\right)$$

$\beta_m, \phi_m$ 이 있으니  $f_m(X)$ (m번째까지 합한 model)와  $w_{i,m+1}$ (Update된 가중치)를 계산할 수 있다.

$$\begin{aligned} f_m(X) &= f_{m-1} + \beta_m \phi_m(X) \\ w_{i,m+1} &= \exp(-\tilde{y}_i f_m(X_i)) \\ &= \exp(-\tilde{y}_i (f_{m-1} - \beta_m \phi_m(X))) \\ &= w_{i,m} \exp(-\tilde{y}_i \beta_m \phi_m(X)) \\ &= w_{i,m} \exp(\beta_m) \text{ if wrong} \end{aligned}$$

# ENSEMBLE LEARNING

## BOOSTING

### Boosting 자세히 살펴보기: AdaBoost

복잡하다... 하지만 알고리즘을 보면 간단하다.

---

#### Algorithm 16.2: Adaboost.M1, for binary classification with exponential loss

---

- 1  $w_i = 1/N;$
  - 2 **for**  $m = 1 : M$  **do**
  - 3     Fit a classifier  $\phi_m(\mathbf{x})$  to the training set using weights  $\mathbf{w}$ ;
  - 4     Compute  $\text{err}_m = \frac{\sum_{i=1}^N w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^N w_{i,m}}$  ;
  - 5     Compute  $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$ ;
  - 6     Set  $w_i \leftarrow w_i \exp[\alpha_m \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))]$ ;
  - 7     Return  $f(\mathbf{x}) = \text{sgn} \left[ \sum_{m=1}^M \alpha_m \phi_m(\mathbf{x}) \right]$ ;
-