

# *Gradient, XG, Ada Boosting*

## Chapter 11. 모델 성능 개선

이지현 고지형 정준모 이민지

# 목차

10%

Review  
부스팅(Boosting)

Gradient Boosting  
이론  
실습

30%

30%

XG Boosting  
이론  
실습

Ada Boosting  
이론  
실습

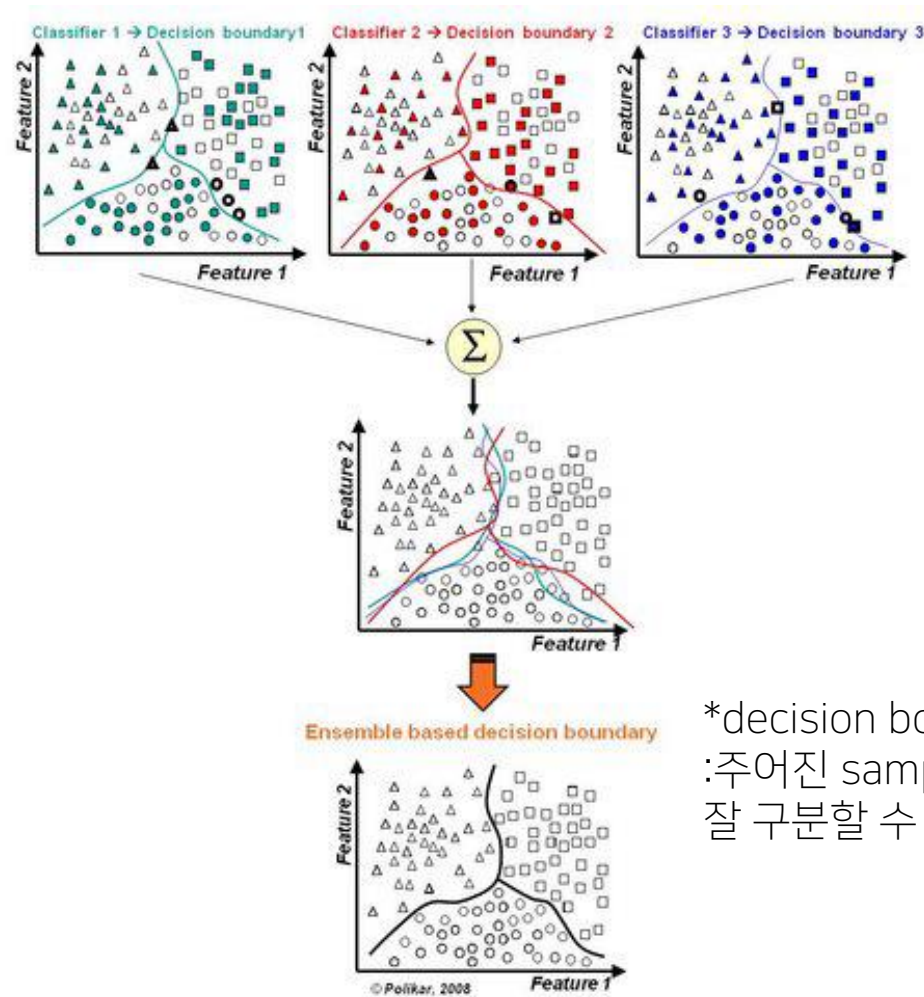
30%

# REVIEW

**지난 개념 훑고 갑시다.**

# Review 앙상블(Ensemble)이란?

머신러닝에서 앙상블이란 단일 모델이 아닌 여러 모델을 혼합하여 의사결정을 내리는 방법



\*decision boundary  
:주어진 sample의 종류를  
잘 구분할 수 있는 경계

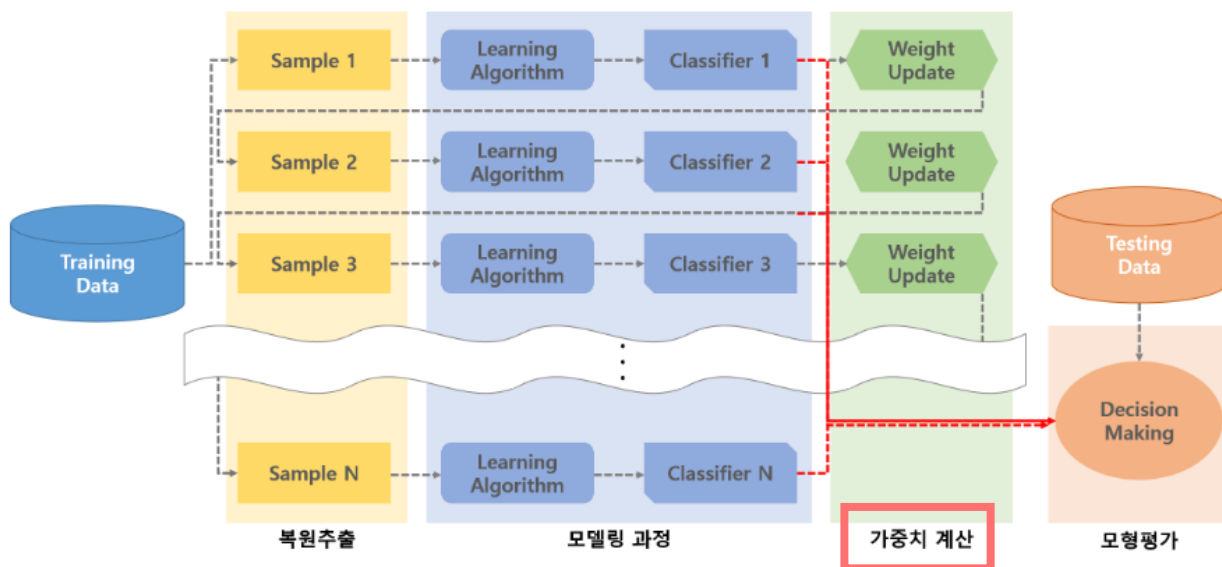
# Review 앙상블(Ensemble)이란?

여러 가지 유형의 앙상블 방법이 존재함

- 단순/가중평균(Simple/Weighted average)
- 배깅(Bagging: Bootstrap aggregating)
- 부스팅(Boosting)
- 스택킹(Stacking)
- 메타 학습(Meta-learning)
- ...

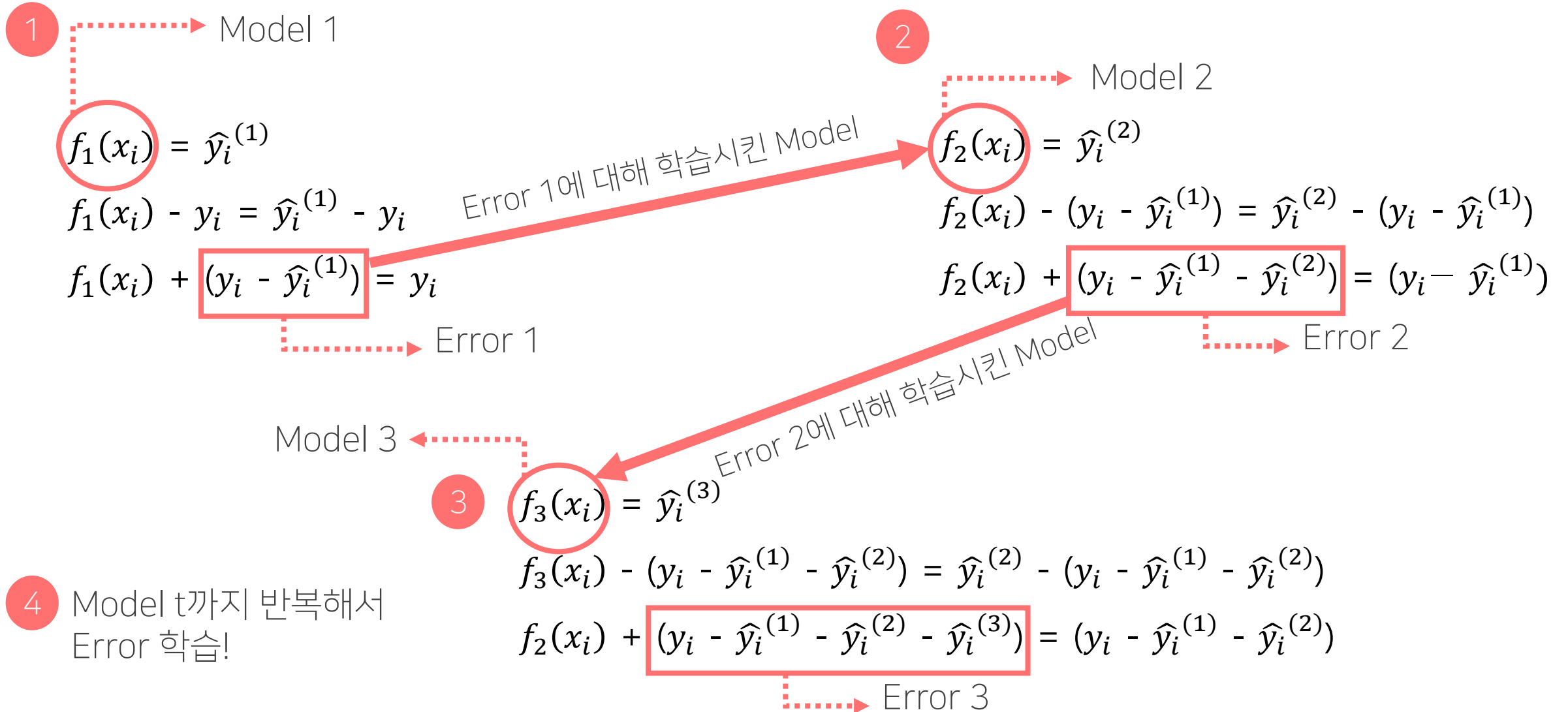
# Review 부스팅(Boosting)이란?

- 배깅(Bagging)과 유사하게 초기 샘플 데이터를 조작하여 다수의 분류기를 생성하는 기법 중 하나
- 가중치를 이용**하여 약검출기(weak classifier)들을 여러 개 모아 강검출기(strong classifier)를 생성하는 방법을 말하고 주로 약검출기로 의사결정나무(decision tree) 모형을 사용함



의사결정나무가 다른 통계모형과 다르게 가정이 적고, 범주형이든 연속형이든 제약없이 쉽게 만들 수 있는 모델이라 주로 사용!

# Review 부스팅의 알고리즘



# Review

## 부스팅의 알고리즘

$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= \hat{y}_i^{(0)} + f_1(x_i) = F_1(x_i) \\ \hat{y}_i^{(2)} &= \hat{y}_i^{(1)} + f_2(x_i) = f_1(x_i) + f_2(x_i) = F_1(x_i) + f_2(x_i) = F_2(x_i) \\ &\dots \\ \hat{y}_i^{(t)} &= \hat{y}_i^{(t-1)} + f_t(x_i) = \sum_{k=1}^{t-1} f_k(x_i) + f_t(x_i) = F_{t-1}(x_i) + f_t(x_i) = F_t(x_i)\end{aligned}$$

Model 1

Model 2

Model t

Model 1부터 t를 combine하면 강력한 모델이 나온다!



# Review

## 부스팅의 절차

1. bootstrap sampling 기법을 이용해 sample1을 추출하고 특정한 학습알고리즘에 적용하여 분류기를 생성
2. 생성된 분류기의 분류결과를 통해 잘못 분류한 데이터와 추출되지 않은 데이터(오분류 데이터)에는 가중치를 부여하여 다음 학습에 이용 → boosting round
3. N번의 boosting round를 거쳐서 완성된 모형들을 이용해 최종적인 분류모형 생성  
※마지막에 학습한 분류기만을 의사결정에 이용하는 것이 아니라 학습에 이용되었던 모든 분류기들의 앙상블 이용!
4. Testing Data를 생성된 분류기를 활용하여 모형을 평가

# Review

## 부스팅의 목적

- ✓ 일반적인 분류문제는 잘못 분류된 개체들을 더 잘 분류하는 것이 목적이므로 부스팅은 잘못 분류된 개체들에 집중하여 새로운 분류규칙을 만드는 것을 목적으로 함.
- ✓ 배깅이 일반적인 모델을 만드는데 집중되어 있다면, 부스팅은 맞추기 어려운 문제를 맞추는데 초점이 맞춰져 있음.
  - 새로운 학습자를 평가할 때 학습모델의 에러들을 계속해서 추적하는 경향이 있음.

# Review

## 부스팅의 특징

- ✓ 전체 데이터에서 여러 샘플링 데이터를 추출하여 순차적으로 이전 학습분류기의 결과를 토대로 다음 학습데이터의 샘플가중치를 조정하면서 학습을 진행.
- ✓ 다음 단계의 weak classifier가 이전 단계의 weak classifier의 영향을 받음. 즉, 이전의 classifier의 양상을 보고 보다 잘 맞출 수 있는 방향으로 다음 단계를 진행하고 각 classifier의 weight를 업데이트함.
- ✓ 최종적으로 서로 영향을 받아 만들어진 여러 weak classifier와 서로 다른 weight를 통해 strong classifier를 생성하게 됨.
- ✓ 단점
  - n차 분류기에 들어가는 데이터는 기존 데이터의 일부만 적용되므로 train data의 규모가 커야 함.
  - 반복수 N을 높이면 성능은 높아지지만 과적합이 일어남.

# Gradient Boosting

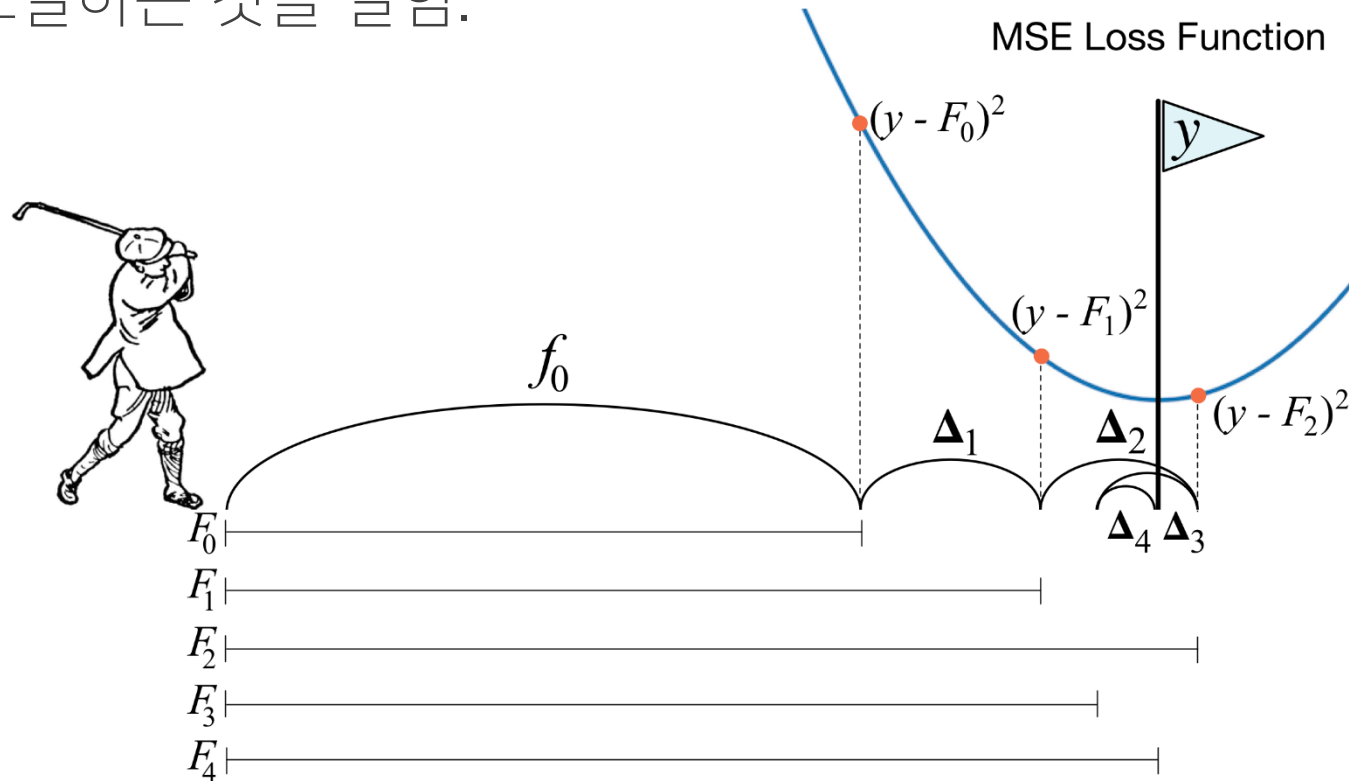
# Gradient Boosting

Gradient Boosting이란?

## = Gradient Descent + Boosting

: 가중치 계산방식에서 Gradient Descent를 이용하는 부스팅

\* Gradient Descent란 손실함수(loss function)를 parameter로 미분해서 기울기 (gradient)를 구하고, 값이 작아지는 방향으로 parameter를 움직이다 보면 손실함수가 최소화되는 지점에 도달하는 것을 말함.



# Gradient Boosting

## Gradient Boosting의 특징

- ✓ 지도학습에서 가장 강력하고 널리 사용하는 모델 중 하나
- ✓ 단점
  - 매개변수를 잘 조정해야 함
  - 훈련시간이 김
- ✓ 다른 tree 기반 모델처럼 feature의 scale을 조정하지 않아도 되고 이진 특성이나 연속적인 특성에서도 잘 동작
- ✓ 하지만 tree 기반 모델의 특성상 희소한 고차원 데이터에는 잘 작동하지 않음

# Gradient Boosting

Gradient Boosting의 알고리즘

$$\hat{F}(x) = \operatorname{argmin}_F \mathbb{E}_{x,y}[L(y, F(x))] \xrightarrow{\text{Risk}}$$

$$F_t(x) = F_{t-1}(x) + \operatorname{argmin}_f \sum_{i=1}^n L(y_i, F_{t-1}(x_i) + f(x_i))$$

$$= F_{t-1}(x) - \gamma_t \sum_{i=1}^n \nabla_{F_{t-1}} L(y_i, F_{t-1}(x_i)) \quad \leftarrow \text{계산 구하기가 힘들어서 근사하여 사용}$$

$$\gamma_t = \operatorname{argmin}_\gamma \sum_{i=1}^n L\left(y_i, F_{t-1}(x_i) - \gamma \frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}\right)$$

이동할 거리를 조절하는 매개변수

Gradient Descent를 수식적으로 설명하자면  $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), n \geq 0.$

# Gradient Boosting 실습

## 1. 데이터 수집

### ✓ 목적

보스턴 주택 가격을 Gradient boosting으로 예측

### ✓ 데이터명

MASS::Boston (House Prices in Boston, 보스턴 주택 가격 데이터)

### ✓ 데이터 설명

보스턴 시의 주택 가격에 대한 데이터

506개의 관측치와 14개의 변수를 가지고 있음

주택의 여러가지 요건들과 주택의 가격 정보가 포함되어 있음

**require(MASS)** Boston 데이터셋을 사용하기 위해 MASS(Modern Applied Statistics with S) 패키지 사용

```
Boston <- MASS::Boston
```



# Gradient Boosting 실습

## 2-1. 데이터 탐색

```
> str(Boston)
```

```
'data.frame':  506 obs. of  14 variables:
 $ crim      : num  0.00632 0.02731 0.02729 0.03237 0.06905 ... 자치시(town)별 1인당 범죄율
 $ zn        : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ... 25,000 평방피트를 초과하는 거주지역의 비율
 $ indus     : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ... 비소매 상업지역이 점유하고 있는 토지의 비율
 $ chas      : int   0 0 0 0 0 0 0 0 0 0 ... 찰스강 더미 변수(강의 경계에 위치한 경우는 1, 아니면 0)
 $ nox       : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ... 질소산화물(PPM: 1,000만분의 1)
 $ rm        : num  6.58 6.42 7.18 7 7.15 ... 1가구 주택당 평균 방의 개수
 $ age       : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ... 1940년 이전에 건축된 소유주택의 비율
 $ dis       : num  4.09 4.97 4.97 6.06 6.06 ... 5개의 보스턴 직업 센터까지의 가중평균거리
 $ rad       : int   1 2 2 3 3 3 5 5 5 5 ... 방사형 도로까지의 접근성 지수
 $ tax       : num  296 242 242 222 222 222 311 311 311 311 ... 10,000달러당 재산세율
 $ ptratio   : num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ... 자치시별 학생/교사 비율
 $ black     : num  397 397 393 395 397 ...  $1000(Bk-0.63)^2$ , Bk는 자치시별 흑인의 비율
 $ lstat     : num  4.98 9.14 4.03 2.94 5.33 ... 모집단의 하위계층의 비율(%)
 $ medv      : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ... 본인소유 주택가격(중앙값) (단위: 천달러)
```

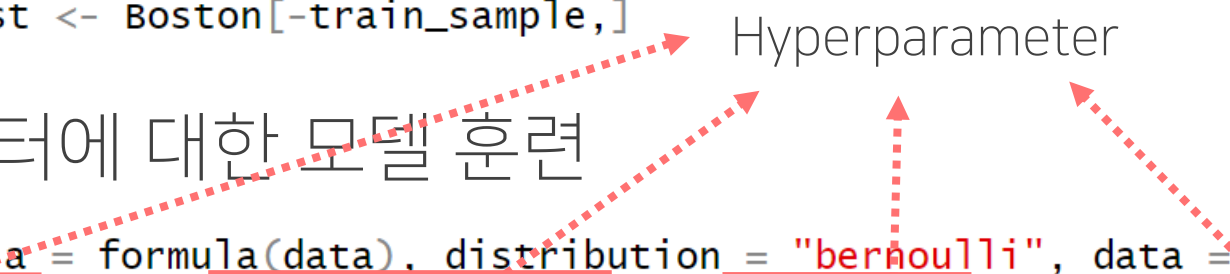
# Gradient Boosting 실습

## 2-2. 데이터 준비

```
train_sample=sample(1:506,size=374) 374/506 = 0.74로 전체 Boston 데이터셋의 74%를 train 데이터셋으로 추출할 예정
Boston_train <- Boston[train_sample,]
Boston_test <- Boston[-train_sample,]
```

## 3. 데이터에 대한 모델 훈련

```
gbm(formula = formula(data), distribution = "bernoulli", data = list(),
     n.trees = 100, interaction.depth = 1, shrinkage = 0.1, bag.fraction = 0.5)
```



**distribution:** 사용할 분포의 이름 (the name of the distribution to use)

"gaussian" (squared error), "laplace" (absolute loss), "huberized" (huberized hinge loss for 0-1 outcomes), "classes" 등이 있음

**n.trees:** tree의 전체 개수 (the total number of trees to fit)

**interaction.depth:** 각 tree의 최대 깊이 (the maximum depth of each tree)

**shrinkage:** 학습률 (the shrinkage or learning rate)

**bag.fraction:** 서브샘플링 비율 (percent of training data to sample for each tree)

# Gradient Boosting 실습

```
install.packages('gbm') # gradient boosting을 사용하기 위해 GBM(Generalized Boosted Models) 패키지 설치
require(gbm)
Boston_model=gbm(medv ~ . ,data = Boston_train,distribution = "gaussian",n.trees = 10000,
                 shrinkage = 0.01, interaction.depth = 4)
```

```
> Boston_model
```

```
gbm(formula = medv ~ ., distribution = "gaussian", data = Boston_train,
     n.trees = 10000, interaction.depth = 4, shrinkage = 0.01)
```

A gradient boosted model with gaussian loss function.

10000 iterations were performed.

There were 13 predictors of which 13 had non-zero influence.

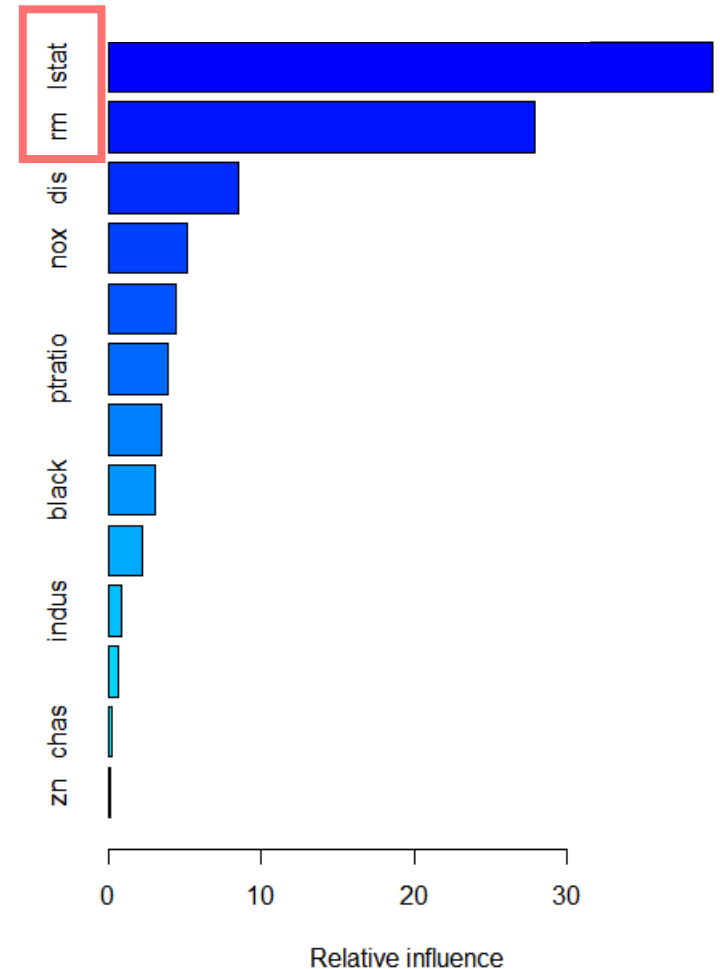
```
> summary(Boston_model)
```

Boston\_model을 summary() 하면 변수 중요도를 표와 그림으로 제공

var	rel.inf	relative influence
-----	---------	--------------------

lstat	lstat	39.5408249
rm	rm	27.8974974
dis	dis	8.5020143
nox	nox	5.0864397
crim	crim	4.4396504
ptratio	ptratio	3.8589211
age	age	3.4467880
black	black	3.0801528
tax	tax	2.2508876
indus	indus	0.8771187
rad	rad	0.6768587
chas	chas	0.2206739
zn	zn	0.1221725

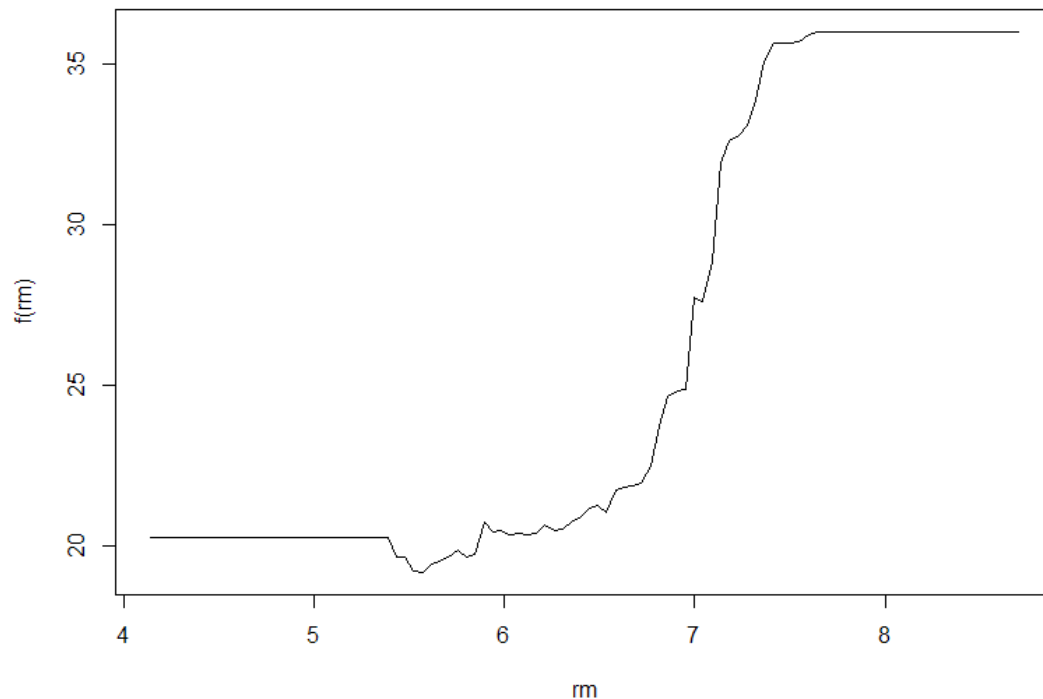
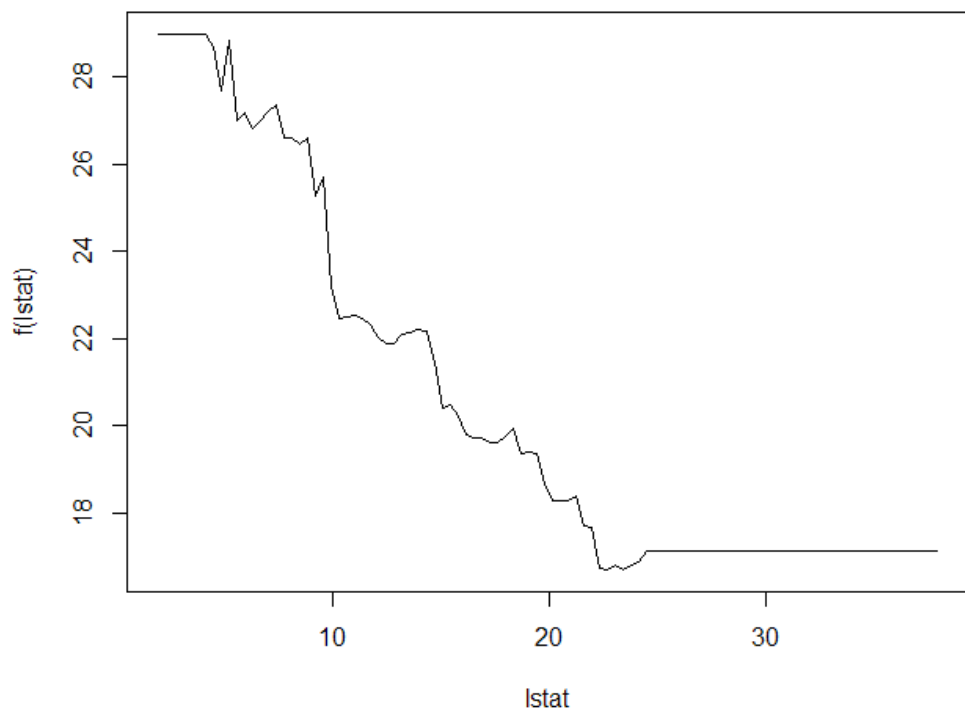
가장 중요한 2개의 변수인 lstat과 rm이  
Boston\_train 데이터셋의 최대 분산을 설명해 줌



# Gradient Boosting 실습

```
plot(Boston.boost,i="lstat")  
plot(Boston.boost,i="rm")
```

부분 의존도 그림(Partial Dependence Plot)은 lstat, rm과 medv의 관계를 보여줌



```
> cor(Boston$lstat,Boston$medv)#negative correlation coeff-r  
[1] -0.7376627  
> cor(Boston$rm,Boston$medv)#positive correlation coeff-r  
[1] 0.6953599
```

# Gradient Boosting 실습

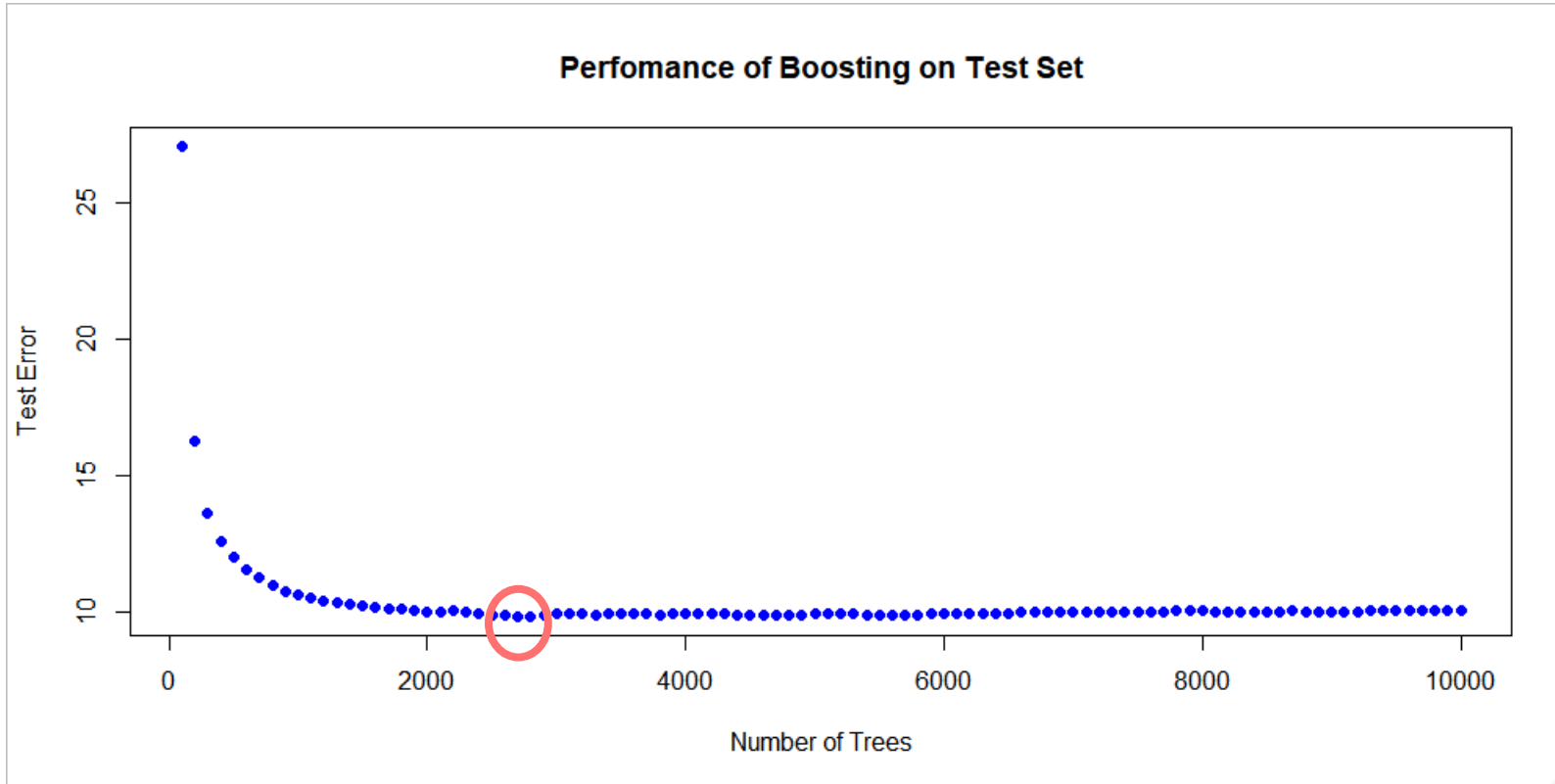
## 4. 모델 성능 평가

```
> Boston_pred <- predict(Boston_model, Boston_test, n.trees = 10000)
> test.error <- mean((Boston_pred-Boston_test$medv)^2)
> test.error
[1] 10.07382
```

## 5. 모델 성능 개선

```
> n.trees = seq(from=100 ,to=10000, by=100)
> predmatrix<-predict(Boston_model,Boston_test,n.trees = n.trees)
> dim(predmatrix)
[1] 132 100
> test.error<-with(Boston_test,apply( (predmatrix-medv)^2,2,mean))
> head(test.error)
      100      200      300      400      500      600
27.06326 16.25664 13.64711 12.57527 12.01219 11.57782
> plot(n.trees , test.error , pch=19,col="blue",xlab="Number of Trees",ylab="Test Error",
      main = "Perfomance of Boosting on Test Set")
```

# Gradient Boosting 실습



```
> which(test.error == min(test.error), arr.ind = TRUE)
```

```
2800
```

```
28
```

```
> Boston_pred_tune <- predict(Boston_model, Boston_test, n.trees = 2800)
```

```
> test.error <- mean((Boston_pred_tune - Boston_test$medv)^2)
```

```
> test.error
```

```
[1] 9.854656
```

다른 Hyperparameter를 조정하는 건 여기를 참고하세요

[http://uc-r.github.io/gbm\\_regression](http://uc-r.github.io/gbm_regression)



# XGBoost

# XGBoost

- Gradient Boosting의 단점: 학습 성능은 좋지만 느립니다.
- 바로 이 속도를 개선한 것이 바로 XGBoost입니다.
- eXtra Gradient Boosting: 병렬처리를 활용한 빠른 속도 + Gradient Boosting
- 다양한 평가함수(evaluation function)를 제공하며 커스텀 또한 가능합니다.
- Object 함수에 규제(Regularization) 항을 추가하여 과적합이 잘 발생하지 않습니다.
- (정보)캐글 우승자들 십중팔구 다 이거 씁니다.



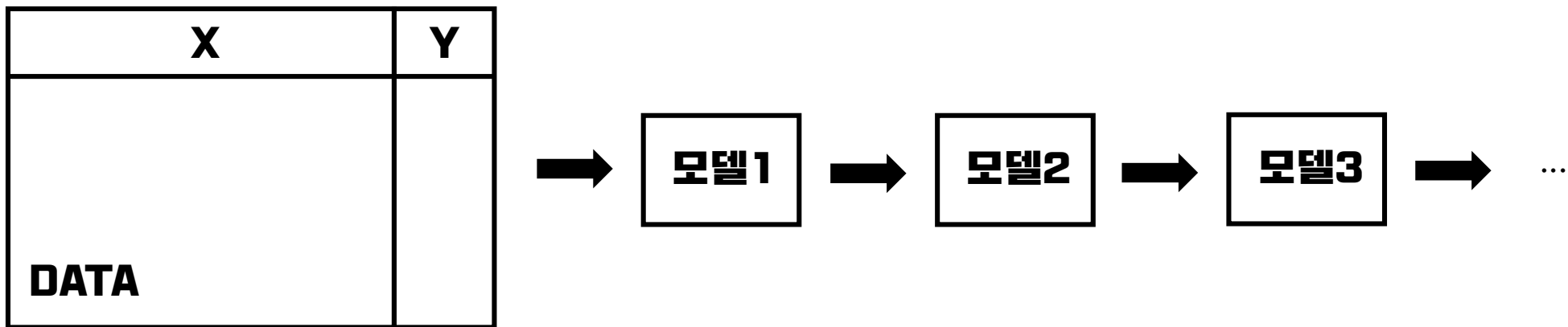
# 학습원리

**조약돌에 묻은 진흙을 털어가는 과정입니다.**

## 기본적인 컨셉만 알고감시다

자세한 설명: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

- 기본적으로 부스팅 모델은 여러 일련의 모델을 순차적으로 생성시키는 과정을 밟습니다.
- 이는 앞의 모델에서 발생한 오차를 예측하는 모델을 이어서 학습시키기 위함입니다.



# 학습원리

- 아래 그림처럼 첫번째 예측 모델인 모델1로부터 예측을 진행하면 다음과 같은 식을 생각할 수 있겠죠?

X	Y
DATA	



**모델1**



$$Y = M_1(X) + error_1$$

$M_1(X)$  : 모델1을 활용한 예측값

$error_1$  : 모델1을 활용한 예측값과 실제 데이터값 간의 오차

- 순차적으로 형성되는 다음 모델들은 각 앞의 모델로부터 발생한 오차를 예측하는 모델이 됩니다.

X	err <sub>1</sub>
DATA	



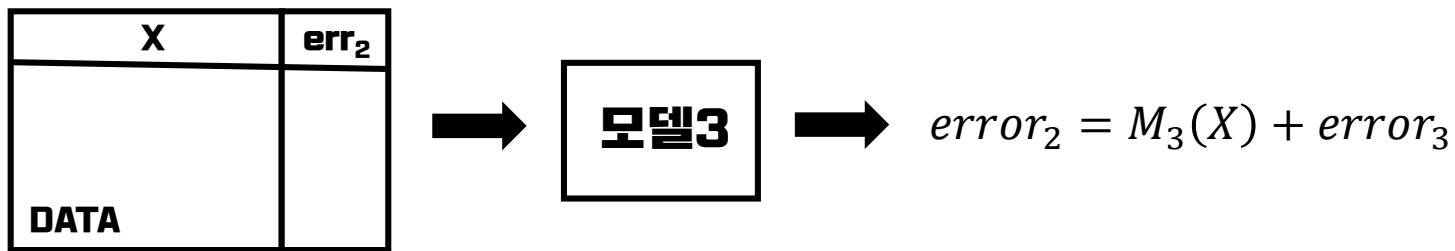
**모델2**



$$error_1 = M_2(X) + error_2$$

# 학습원리

- 이러한 방법으로 계속 모델을 구성했다고 생각합시다.



- 그러면 다음과 같은 식을 얻을 수 있겠죠

$$Y = M_1(X) + error_1$$

$$error_1 = M_2(X) + error_2$$

$$error_2 = M_3(X) + error_3$$



$$error_1 = M_2(X) + M_3(X) + error_3$$

즉,

$$Y = M_1(X) + M_2(X) + M_3(X) + error_3$$

이 성립하며 오차를 점점 줄여 나가게 된다.

# 학습원리

- 이와 더불어, 학습이 '잘' 되도록 하는데 꼭 필요한 **손실함수(loss function)**가 있죠.
- 이부분이 Gradient Boosting과의 차이가 나는 부분입니다.
- XGBoost는 이러한 손실함수에 적절한 **규제(Regularization)**와 관련한 항을 추가합니다. 이를 통해 과적합이 방지됩니다. 아래의 붉은 표시가 되어있는 항이 규제 항이고, 해당 함수를 최소화하는 방향으로 학습을 진행합니다.

이렇게 두 항을 합쳐서 **Object 함수**라고 부르고 결과적으로 XGBoost 알고리즘은 이 함수의 함숫값을 최소화 시키도록 학습합니다.

$$\begin{aligned}\text{obj}^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}\end{aligned}$$

(해당링크를 참고하시면, 최소화를 하기 위한 파라미터들을 수식을 통해 확인해볼 수 있습니다.)  
<https://xgboost.readthedocs.io/en/latest/tutorials/model.html>



# 실습: ADVANCED

## 1. 설치

```
4 # 설치
5 install.packages('DiagrammeR')
6 install.packages("drat", repos="https://cran.rstudio.com")
7 drat::addRepo("dmlc")
8 install.packages("xgboost", repos="http://dmlc.ml/drat/", type = "source")
```

최신 버전을 받아줍니다.

```
12 library(xgboost)
13 data(agaricus.train, package='xgboost')
14 data(agaricus.test, package='xgboost')
15 train <- agaricus.train
16 test <- agaricus.test
```

Xgboost 패키지를 불러오고 xgboost 내장 데이터 Agaricus의 학습데이터와 테스트데이터를 불러옵니다. 해당 데이터는 아가리쿠스버섯에 대한 데이터로, 해당 버섯인가 아닌가에 대한 이진 분류 데이터셋입니다.

# 실습: ADVANCED

## 2. 특이한 데이터셋

```
20 # 타입 확인
21 class(train) #list
22 class(train$data) # dgCMatrix
23 class(train$label) # numeric
```

```
> # 타입 확인
> class(train) #list
[1] "list"
> class(train$data) # dgCMatrix
[1] "dgCMatrix"
attr(,"package")
[1] "Matrix"
> class(train$label) # numeric
[1] "numeric"
```

XGBoost는 다른 모델들과 다르게 특이한 데이터셋을 활용합니다.  
Train데이터의 class가 list임을 확인할 수 있고, train데이터에는 data와 label이 포함되어 있습니다.



# 실습: ADVANCED

## 2. 특이한 데이터셋

```
> train$data
6513 x 126 sparse Matrix of class "dgCMatrix"
  [[ suppressing 44 column names 'cap-shape=bell', 'cap-shape=conical', 'cap-shape=convex' ... ]]
  [[ suppressing 44 column names 'cap-shape=bell', 'cap-shape=conical', 'cap-shape=convex' ... ]]

[1,] . . 1 . . . . . 1 1 . . . . . 1 . . . . . 1 . . . . . 1 . 1 . . . . . 1 1 . . . . .
[2,] . . 1 . . . . . 1 . . . . . 1 1 . 1 . . . . . 1 . 1 . . 1 . 1 . . . . .
[3,] 1 . . . . . 1 . . . . . 1 . 1 . 1 . . . . . 1 . 1 . . 1 . . 1 . . . . .
[4,] . . 1 . . . . . 1 . . . . . 1 . 1 . . . . . 1 . . 1 . 1 . . 1 . 1 . . . . .
[5,] . . 1 . . . . . 1 . . . . . 1 . . . . . 1 . . . . . 1 . . 1 . 1 . 1 . . . . .
[6,] . . 1 . . . . . 1 . . . . . 1 1 . 1 . . . . . 1 . 1 . 1 . 1 . 1 . . . . .
[7,] 1 . . . . . 1 . . . . . 1 . 1 . 1 . . . . . 1 . 1 . 1 . . . . .
[8,] . . 1 . . . . . 1 . . . . . 1 . 1 . . . . . 1 . . 1 . 1 . . 1 . . . . .
[9,] 1 . . . . . 1 . . . . . 1 1 . 1 . . . . . 1 . 1 . 1 . . . . .
[10,] . . 1 . . . . . 1 . . . . . 1 1 . 1 . . . . . 1 . 1 . 1 . . . . .
[11,] . . 1 . . . . . 1 . . . . . 1 1 . 1 . . . . . 1 . 1 . 1 . . 1 . . . . .
```

예제 데이터의 feature데이터는 왼쪽과 같은 dgCMatrix라는 Sparse Matrix(희소행렬)의 XGBoost 자체 자료형으로 구성되어있으나,

# 실습: ADVANCED

## 2. 특이한 데이터셋

```
> as.matrix(train$data)
      cap-shape=bell cap-shape=conical cap-shape=convex cap-shape=flat cap-shape=knobbed cap-shape=s
[1,]           0           0           1           0           0
[2,]           0           0           1           0           0
[3,]           1           0           0           0           0
[4,]           0           0           1           0           0
[5,]           0           0           1           0           0
[6,]           0           0           1           0           0
[7,]           1           0           0           0           0
      cap-surface=grooves cap-surface=scaly cap-surface=smooth cap-color=brown cap-color=buff cap-co
[1,]           0           0           1           1           0
[2,]           0           0           1           0           0
[3,]           0           0           1           0           0
[4,]           0           1           0           0           0
[5,]           0           0           1           0           0
[6,]           0           1           0           0           0
[7,]           0           0           1           0           0
      cap-color=pink cap-color=purple cap-color=red cap-color=white cap-color=yellow bruises?=bruise
[1,]           0           0           0           0           0
[2,]           0           0           0           0           1
```

뒤에서 학습을 해볼테지만, 이렇게 R의 기본 자료형인 Matrix자료형을 활용해서 학습을 하여도 상관없습니다. 저희는 xgboost의 편리한 **DMatrix** 자료형을 사용할 거거든요.

## 2-1. DMatrix

앞의 두 자료형으로도 학습을 할 수 있지만, XGBoost에서는 Dmatrix라는 조금은 특이한 자료형을 활용하여 학습하는 것이 권장됩니다. 다른 자료형에 비해 XGBoost의 학습에 더 최적화 되어있고, 학습 속도가 더 빠르다고 하네요. 다음과 같이 하나의 Dmatrix 객체에 **feature데이터와 label데이터를 함께** 넣어주며 생성할 수 있습니다.

```
47 # DMatrix형태 데이터 생성
48 dtrain <- xgb.DMatrix(data = train$data, label = train$label)
49 dtest <- xgb.DMatrix(data = test$data, label = test$label)
```

```
51 # matrix 자료형을 넣어줘도 상관없어요
52 dtrain <- xgb.DMatrix(data = as.matrix(train$data), label = train$label)
53 dtest <- xgb.DMatrix(data = as.matrix(test$data), label = test$label)
```

이렇게 matrix자료형을 넣어줘도 괜찮습니다.

또한 Dmatrix 자료형은 저장 기능을 제공해서 이렇게 변환한 데이터를 저장할 수 있습니다.

```
64 # DMatrix는 저장가능!
65 xgb.DMatrix.save(dtrain, '저장파일명')
```



# Train: Parameter

XGBoost는 학습을 하는데 만져줘야 하는 파라미터들이 다양합니다. 파라미터는 크게 **General**, **Learning Task**, **Command Line** 의 세 부류로 나뉘게 되고(외울 필요 없음), 주요한 파라미터들은 다음과 같습니다.

## ➤ General Parameter:

- **booster**: gbtrees - 의사결정나무 기반 모델 / gblinear - 선형함수 기반 모델 (일반적으로 gbtrees가 성능이 좋습니다)  
[gbtrees모델에서 활용하는 파라미터]
  - **eta**: [0,1], learning rate를 의미합니다. **작을수록 보수적**으로 학습합니다(=조밀하게 학습)
  - **gamma**: [0, ∞], [default:0]규제(Regularization)항과 관련한 계수 값입니다. **클수록 보수적**으로 학습합니다.
  - **max.depth**: [integer], Tree의 최대 깊이를 결정합니다.
  - **min.child.weight**: [0, ∞], [default: 1]각 단계의 노드들의 가중치들의 합의 최소값을 지정할 수 있습니다. 가중치의 합이 이것보다 작다면 더 이상 뿌리를 깊게 내리지 않습니다. 이 값이 너무 크면 under-fitting, 너무 작으면 over-fitting의 위험이 있습니다.



# Train: Parameter

## ➤ Learning Task Parameter

- *objective*: 자신의 목적에 맞는 object를 설정합니다.
  - 'reg:logistic': 로지스틱 회귀
  - 'reg:squarederror': MSE를 손실함수로 갖는 회귀
  - 'binary:logistic': 이진분류, 확률값을 반환함
- *eval.metric*: 평가기준을 설정합니다.
  - 'rmse': RMSE
  - 'mae': MAE
  - 'error': 에러율
- *seed*: 시드값을 정할 수 있습니다.

## ➤ Command Line Parameter

- *nrounds*: 부스팅을 반복할 횟수를 결정합니다. (=순차적인 모델을 몇 개 만들어 나갈지)



# Train

간단히 학습을 해보겠습니다. 아래와 같이 앞의 파라미터를 참고한 xgb.train() 함수를 활용하여 모델을 만들었습니다.

```
83 watchlist <- list(train = dtrain, test = dtest)
84 bst <- xgb.train(data=dtrain, booster = "gbtree", max.depth=2, nthread = 2,
85                 nrounds=2, watchlist=watchlist, eval.metric = "error",
86                 eval.metric = "logloss", objective = "binary:logistic")
```

\*nthread: 병렬처리에 참여할 CPU개수(Default: 가능한 많이)

\*watchlist: 생성한 리스트로부터 evaluation을 진행합니다.

학습을 진행하면 아래와 같이 학습 round마다(nround=2이므로 2개의 순차적 모델을 생성하여 학습합니다)의 평가척도를 확인할 수 있습니다. 특이한 점이 있다면 train할 시에 **eval.metric의 파라미터가 2개**가 포함되어 있죠? 두 개의 척도를 넣어줘도 아래와 같이 한꺼번에 확인해볼 수 있습니다.

```
> bst <- xgb.train(data=dtrain, booster = "gbtree", max.depth=2, nthread = 2,
+                 nrounds=2, watchlist=watchlist, eval.metric = "error",
+                 eval.metric = "logloss", objective = "binary:logistic")
[1]   train-error:0.046522   train-logloss:0.482554   test-error:0.042831   test-logloss:0.480384
[2]   train-error:0.046522   train-logloss:0.359534   test-error:0.042831   test-logloss:0.357757
```

또한 다음과 같이 모델을 저장하거나 불러올 수도 있습니다.

```
93 xgb.save(bst, 'model')
94 model <- xgb.load('model')
```



# Train

그리고 이렇게 구성한 모델로 predict(모델, 학습데이터-Dmatrix형태)함수를 넣어주면 예측 결과를 확인할 수 있습니다.

```
89 watchlist <- list(train = dtrain, test = dtest)
90 bst <- xgb.train(data=dtrain, booster = "gbtree", max.depth=2, nthread = 2,
91                 nrounds=2, watchlist=watchlist, eval.metric = "error",
92                 eval.metric = "logloss", objective = "binary:logistic")
93 pred <- predict(bst, dtest)
94 pred
```

```
C:/Users/KO_JIHYEONG/Desktop/Bitamin/XGB/xgb_R/ ➡
[745] 0.92392391 0.92392391 0.05169873 0.05169873 0.05169873 0.05169873 0.05169873 0.05169873 0.05169873
[757] 0.92392391 0.92392391 0.92392391 0.92392391 0.05169873 0.92392391 0.05169873 0.92392391 0.92392391
[769] 0.05169873 0.92392391 0.05169873 0.92392391 0.92392391 0.05169873 0.92392391 0.92392391 0.92392391
[781] 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391
[793] 0.05169873 0.92392391 0.92392391 0.92392391 0.05169873 0.92392391 0.05169873 0.05169873 0.05169873
[805] 0.05169873 0.92392391 0.92392391 0.05169873 0.05169873 0.92392391 0.92392391 0.92392391 0.92392391
[817] 0.92392391 0.92392391 0.92392391 0.05169873 0.05169873 0.05169873 0.05169873 0.05169873 0.92392391
[829] 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.05169873 0.92392391
[841] 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391
[853] 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391
[865] 0.92392391 0.92392391 0.92392391 0.05169873 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391
[877] 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391 0.92392391
```

```
77 # Label 값에 접근
78 label ← getinfo(dtest, "label") #DMatrix 형태의 데이터로부터 가져오려 할 때
79 label
```

이렇게 접근한 label값을 예측값(pred)과 비교하여 에러율을 측정해볼 수 있습니다.

```
77 # Label 값에 접근
78 label ← getinfo(dtest, "label") #DMatrix 형태의 데이터로부터 가져오려 할 때
79 label
80 pred ← predict(bst, dtest)
81 err ← as.numeric(sum(as.integer(pred > 0.5) ≠ label))/length(label)
82 print(paste("test-error=", err))
```





# Train: Importance

XGBoost 패키지도 랜덤포레스트와 마찬가지로 `xgb.importance()`를 통해 중요 독립변수들을 확인해볼 수 있습니다. 앞서 간단히 학습해본 모델들로부터 중요변수들을 확인해봅시다.

```
85 importance_matrix ← xgb.importance(model = bst)
86 print(importance_matrix)
```

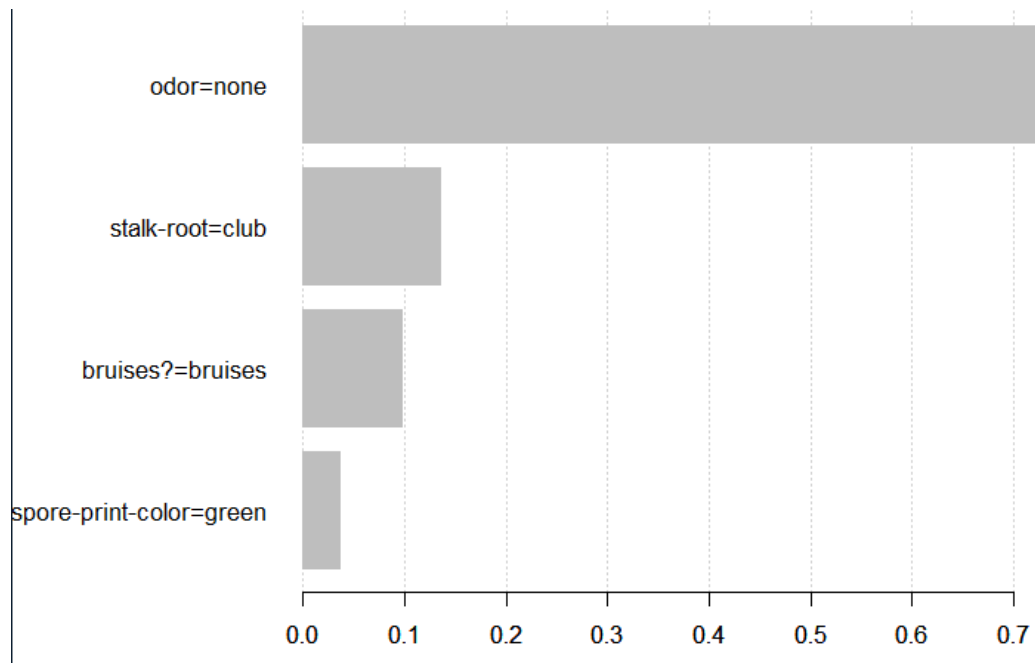
```
> importance_matrix ← xgb.importance(model = bst)
> print(importance_matrix)
      Feature      Gain      Cover Frequency
1:      odor=none 0.72770687 0.5000000 0.3333333
2:    stalk-root=club 0.13678444 0.1471264 0.1666667
3:    bruises?=bruises 0.09899859 0.1378481 0.1666667
4: spore-print-color=green 0.03651011 0.2150256 0.3333333
```

이렇게 수치적으로 확인해볼 수도 있고,



# Train: Importance

```
88 xgb.plot.importance(importance_matrix = importance_matrix)
```



이렇게 xgb.plot.importance() 함수로 시각화를 해볼 수도 있습니다.



# Train: Cross Validation

XGBoost 패키지는 Cross Validation을 자체적으로 제공합니다. 다음과 같이 편리하게 Dmatrix형태로 데이터를 넣어주고, 필요한 파라미터들과 fold수를 지정하여 실행할 수 있습니다. 또한 한번에 여러 평가척도들을 확인할 수 있습니다.

```
105 cv ← xgb.cv(data = dtrain, nrounds = 3, nthread = 2, nfold = 5,  
106             metrics = list("rmse", "auc", 'mae'), max_depth = 3, eta = 1,  
107             objective = "binary:logistic")
```

```
> cv  
##### xgb.cv 5-folds  
iter train_rmse_mean train_rmse_std train_auc_mean train_auc_std train_mae_mean train_mae_std  
  1      0.1624320    0.0015722329    0.9871120  5.667063e-04    0.1369324  0.0007226669  
  2      0.0778636    0.0006376258    0.9999362  7.194453e-06    0.0605298  0.0003091300  
  3      0.0445438    0.0038424843    0.9999504  2.557029e-05    0.0242378  0.0005582495  
test_rmse_mean test_rmse_std test_auc_mean test_auc_std test_mae_mean test_mae_std  
  0.1623910    0.005404889    0.9871140  2.243418e-03    0.1369584  0.001627378  
  0.0779738    0.002838491    0.9999344  2.820355e-05    0.0605972  0.001314111  
  0.0477074    0.007969356    0.9999420  3.802631e-05    0.0246712  0.001335748
```

순차적 모델을 3개로 구성하였으니 3번의 학습이 일어나게 됩니다.

# ADA Boosting

# ADA boosting 이론

## Ada Boost

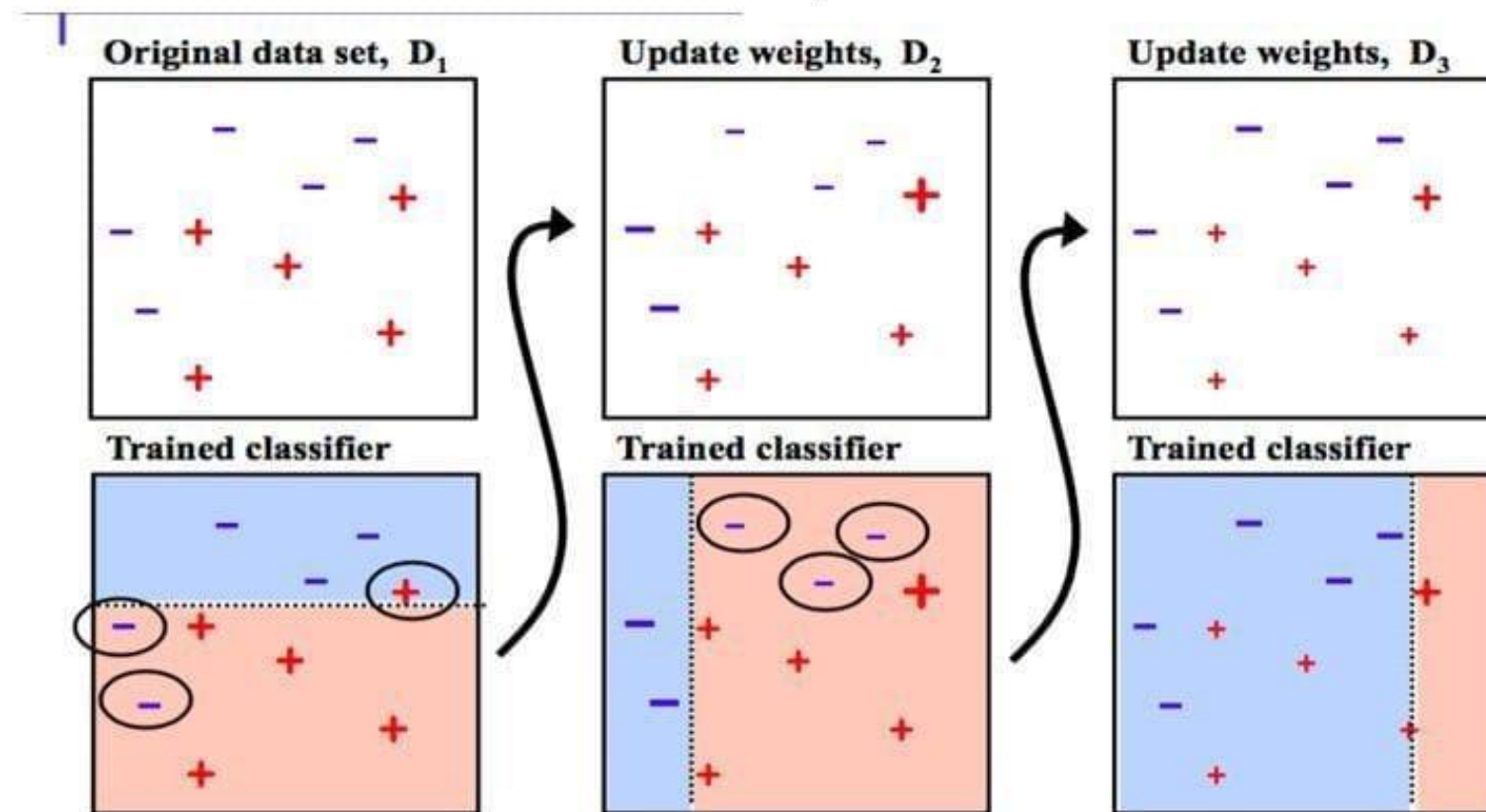
- Adaptive + Boosting => “학습모형들에 대한 가중치를 다르게 함”
- 간단한 약분류기 (weak classifier)들이 상호보완 하도록 단계적으로 학습, 이들을 조합하여 최종 강분류기 (strong classifier)의 성능을 증폭시킨다.

## 훈련

- 약 분류기로 잘못 분류된 경우와 올바르게 분류된 경우를 추출
- 올바르게 분류된 예제들은 가중치를 낮게 잘못 분류된 경우 가중치를 높여줌
- 오류가 0이 되거나 약한 분류기 수 최대치에 도달까지 반복
- 오류를 초점을 맞춘 모델학습!

# ADA boosting 이론

## Algorithm Adaboost - Example

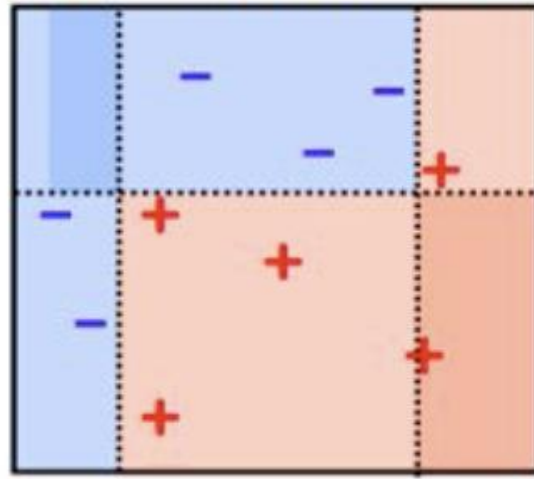


# ADA boosting 이론

Weight each classifier and combine them:

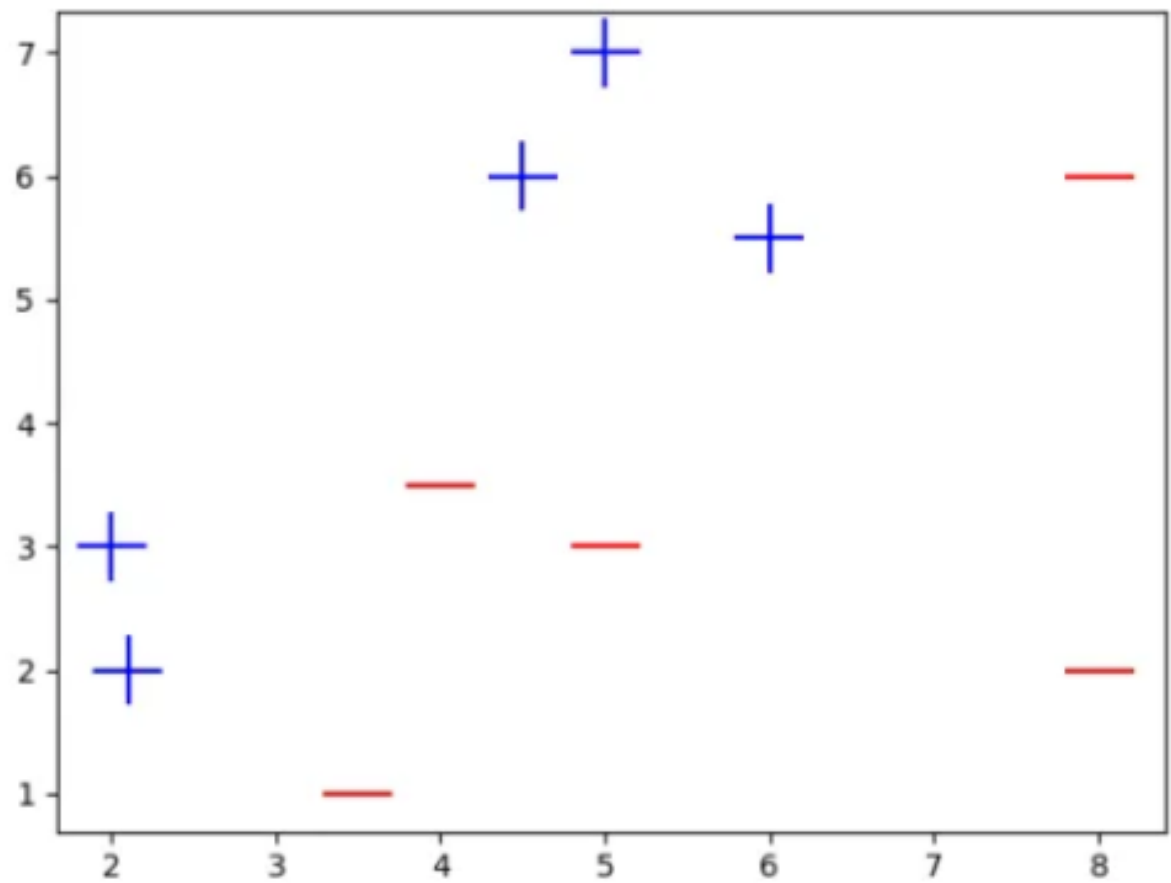
$$.33 * \begin{array}{|c|} \hline \text{blue} \\ \hline \text{orange} \\ \hline \end{array} + .57 * \begin{array}{|c|} \hline \text{blue} \\ \hline \text{orange} \\ \hline \end{array} + .42 * \begin{array}{|c|} \hline \text{blue} \\ \hline \text{orange} \\ \hline \end{array} \approx 0$$

Combined classifier



1-node decision trees  
"decision stumps"  
*very simple classifiers*

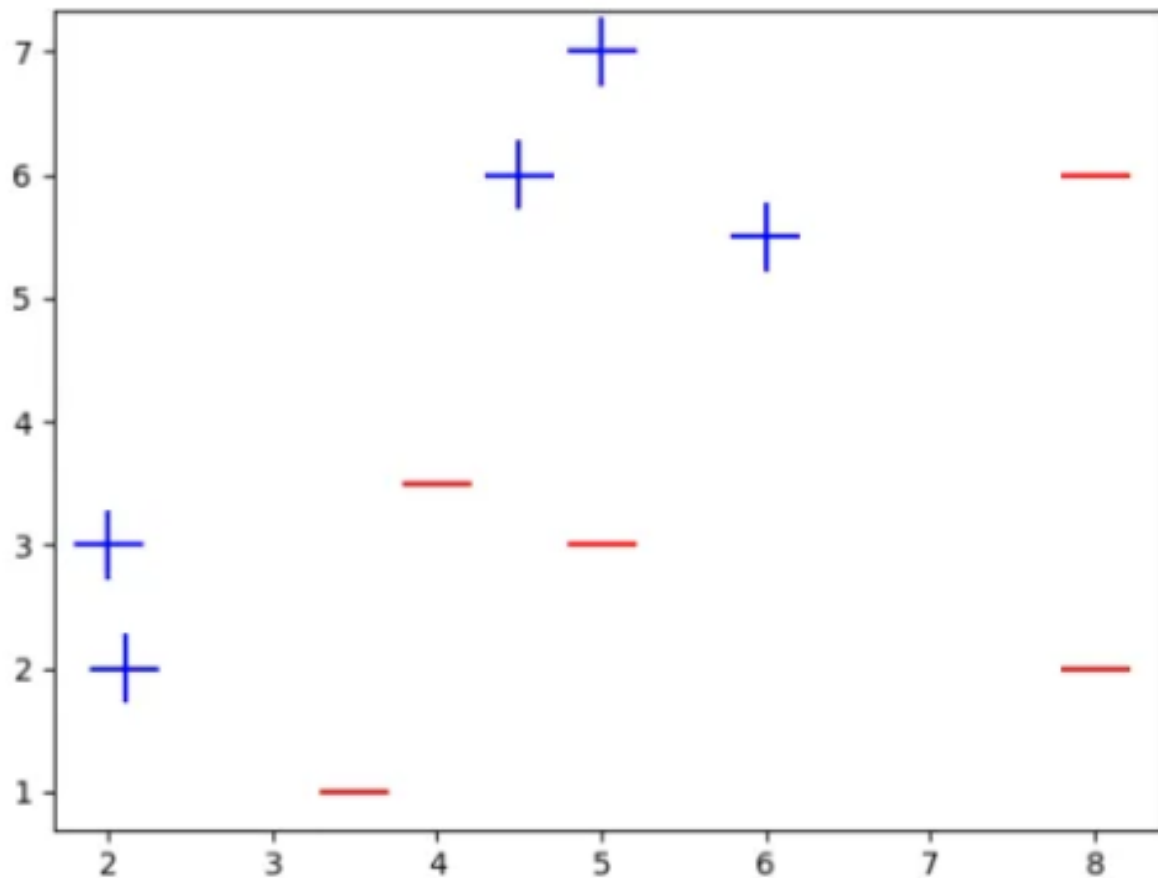
# ADA boosting 이론



X	Y	Decision
2	3	T
2.1	2	T
4.5	6	T
4	3.5	F
3.5	1	F
5	7	T
5	3	F
6	5.5	T
8	6	F
8	2	F



# ADA boosting 이론



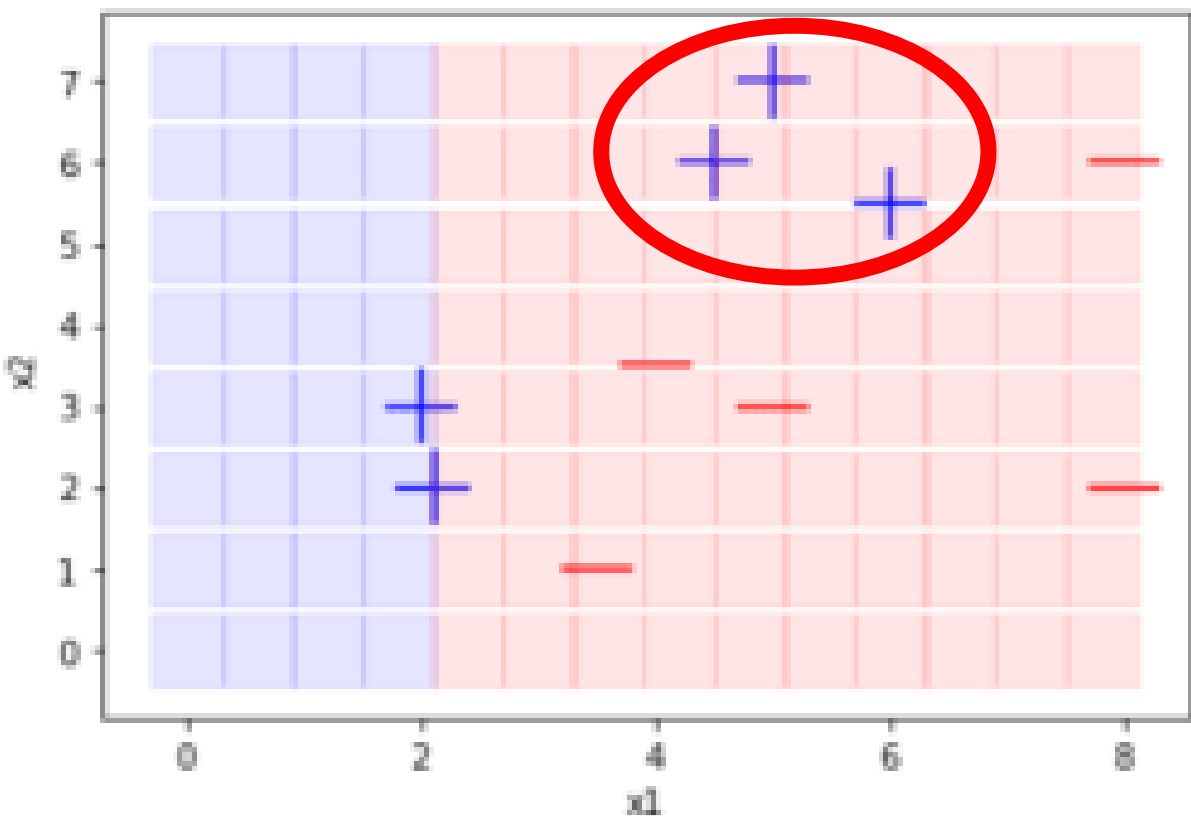
$$D_1(i) = 1/m = 1/10$$

# ADA boosting 이론

## Round 1

0.4236489301936017

x



$$\varepsilon = \frac{\text{잘못 분류된 예제의 수}}{\text{전체 예제의 수}} = 0.3$$

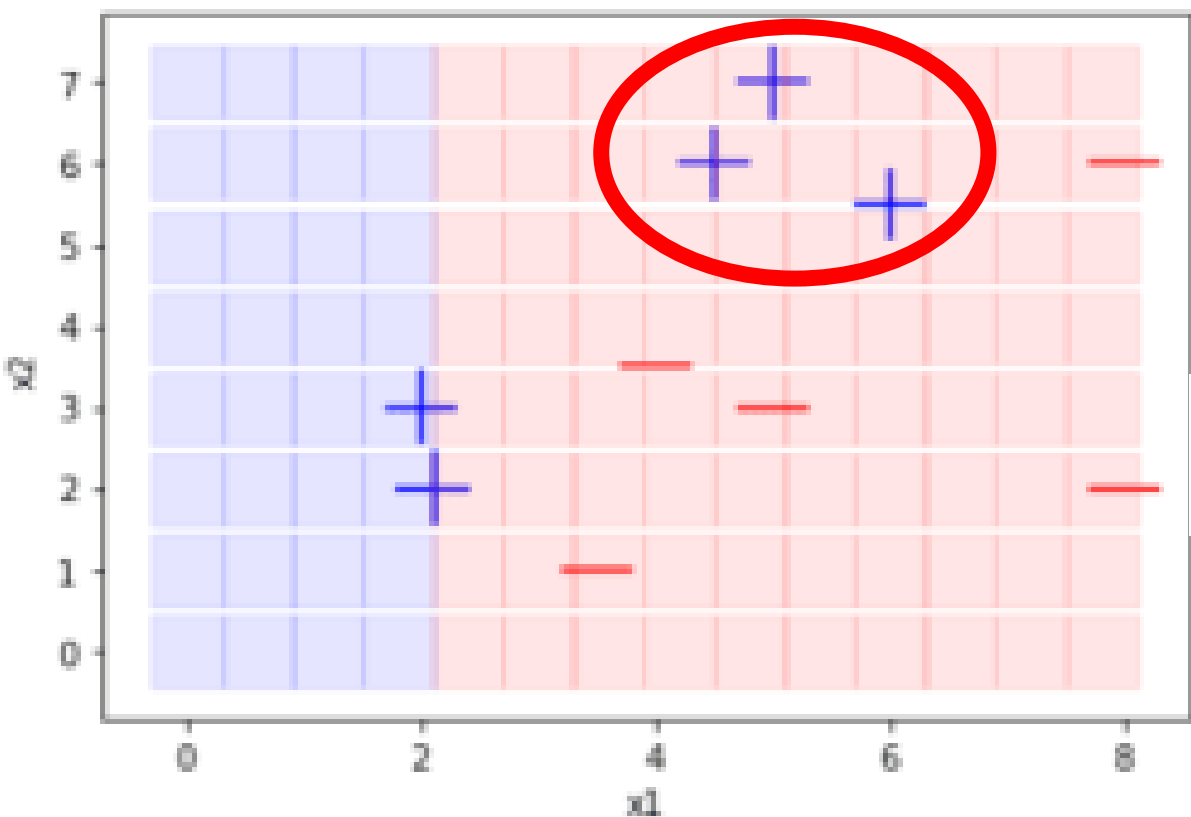
$$\alpha = \frac{1}{2} \ln \left( \frac{1 - \varepsilon}{\varepsilon} \right) = \frac{1}{2} \ln \left( \frac{\text{올바르게 분류된 확률값}}{\text{잘못분류된 확률값}} \right)$$

$$= 0.42$$

# ADA boosting 이론

0.4236489301936017

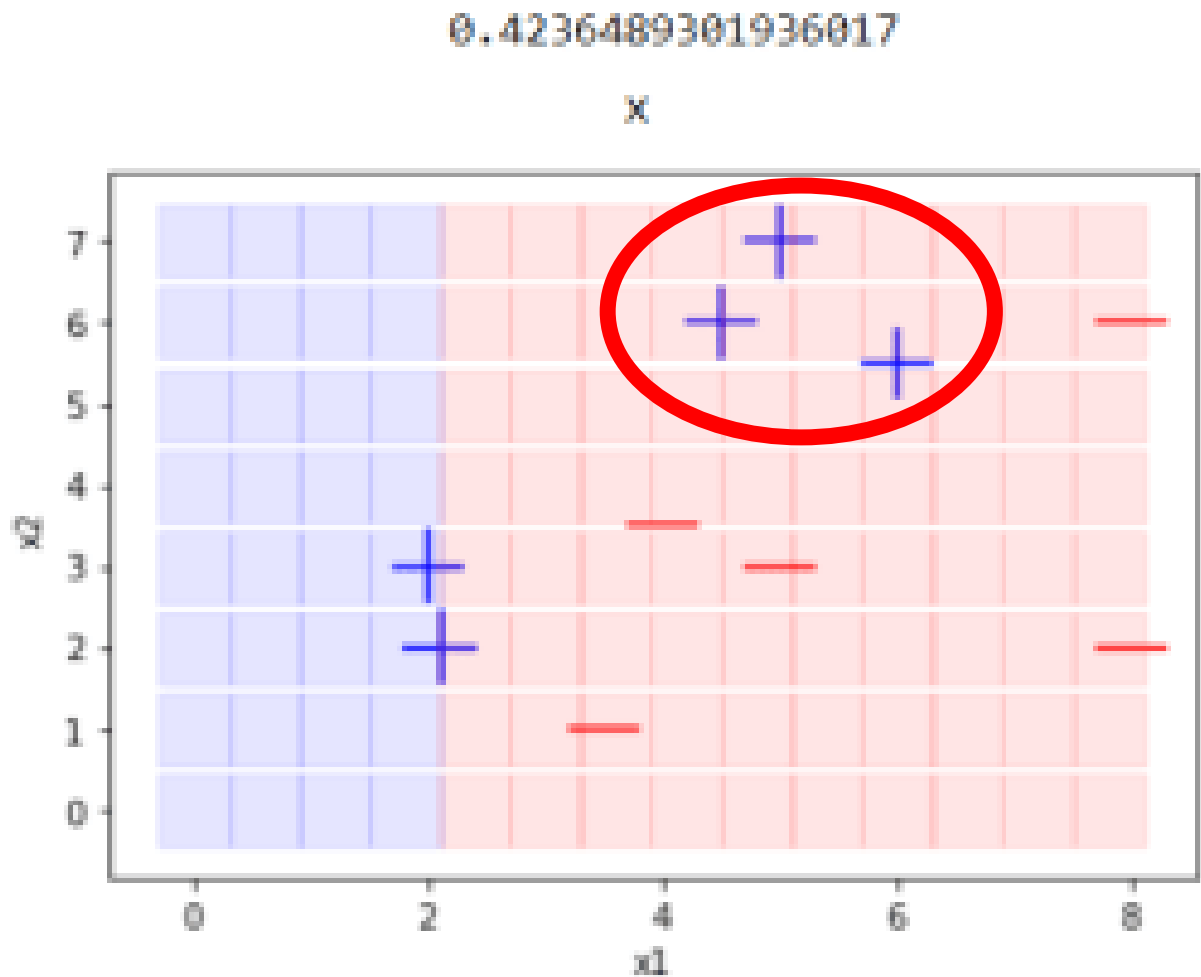
x



$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{\text{Sum}(D)} = (\text{확실하게 예측한 경우})$$

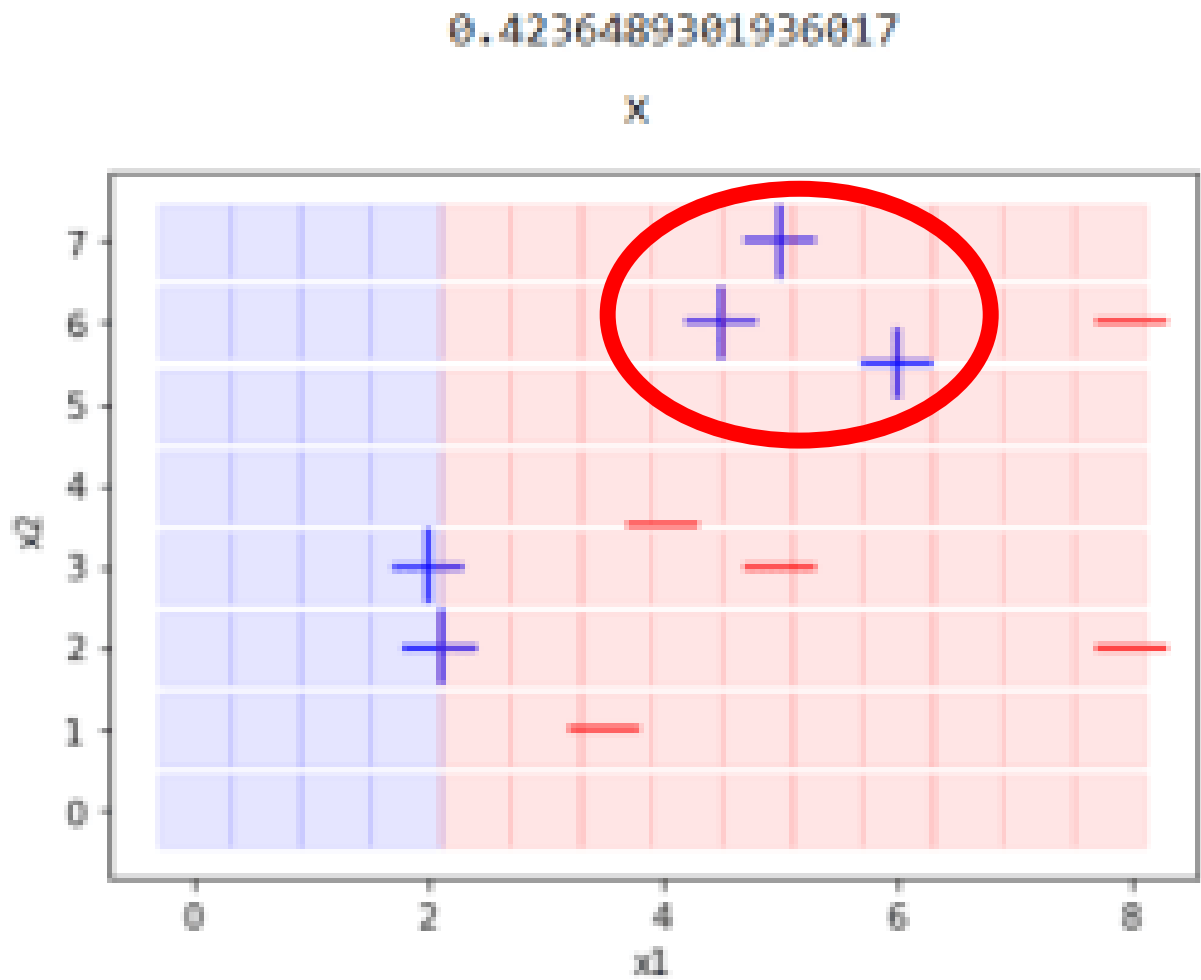
$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{\text{Sum}(D)} = (\text{확실하게 예측하지 못한 경우})$$

# ADA boosting 이론



actual	Pred	w	Norm(w)
T	1	0.065	0.071
T	1	0.065	0.071
T	-1	0.153	0.167
F	-1	0.065	0.071
F	-1	0.065	0.071
T	-1	0.153	0.167
F	-1	0.065	0.071
T	-1	0.153	0.167
F	-1	0.065	0.071
F	-1	0.065	0.071

# ADA boosting 이론

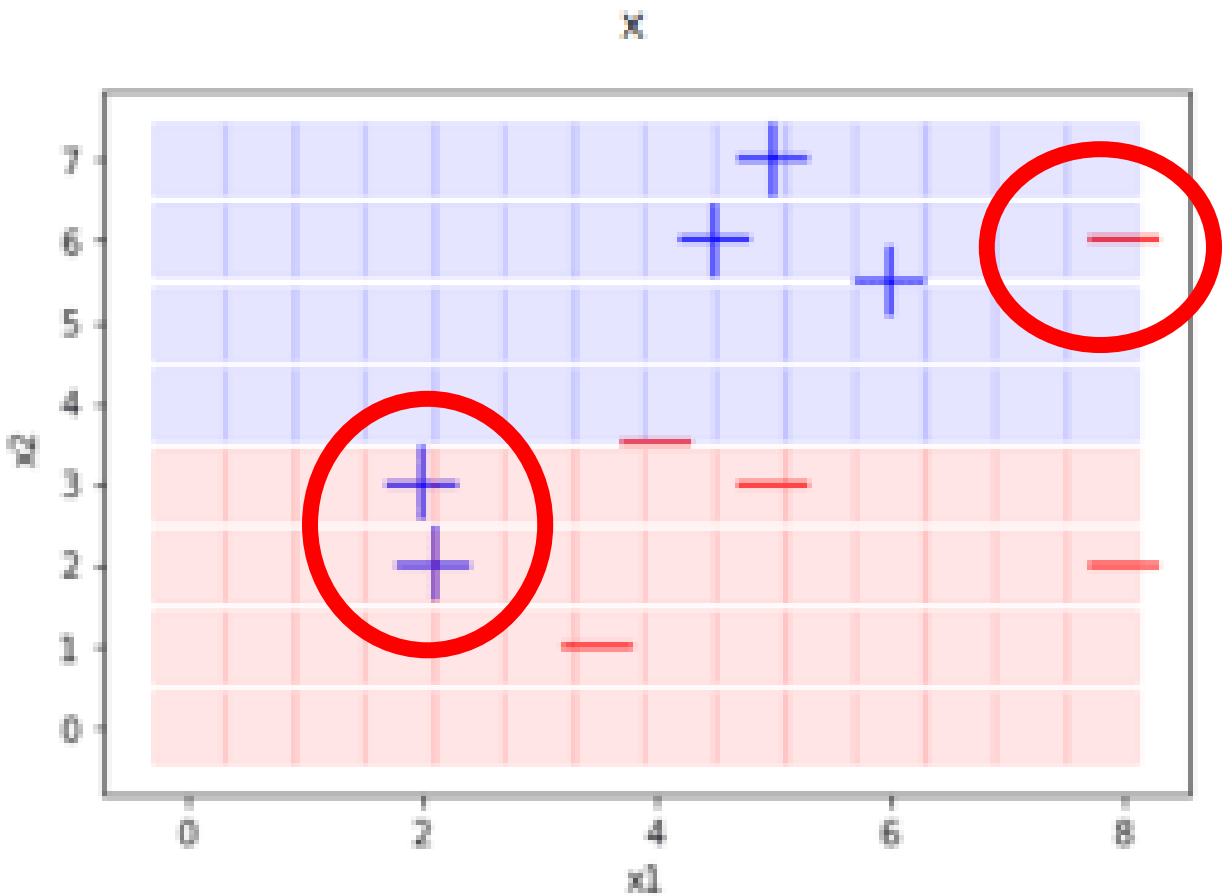


actual	Pred	w	Norm(w)
T	1	0.065	0.071
T	1	0.065	0.071
T	-1	0.153	0.167
F	-1	0.065	0.071
F	-1	0.065	0.071
T	-1	0.153	0.167
F	-1	0.065	0.071
T	-1	0.153	0.167
F	-1	0.065	0.071
F	-1	0.065	0.071

ADA boosting 이론

Round 2

0.6496414920651304

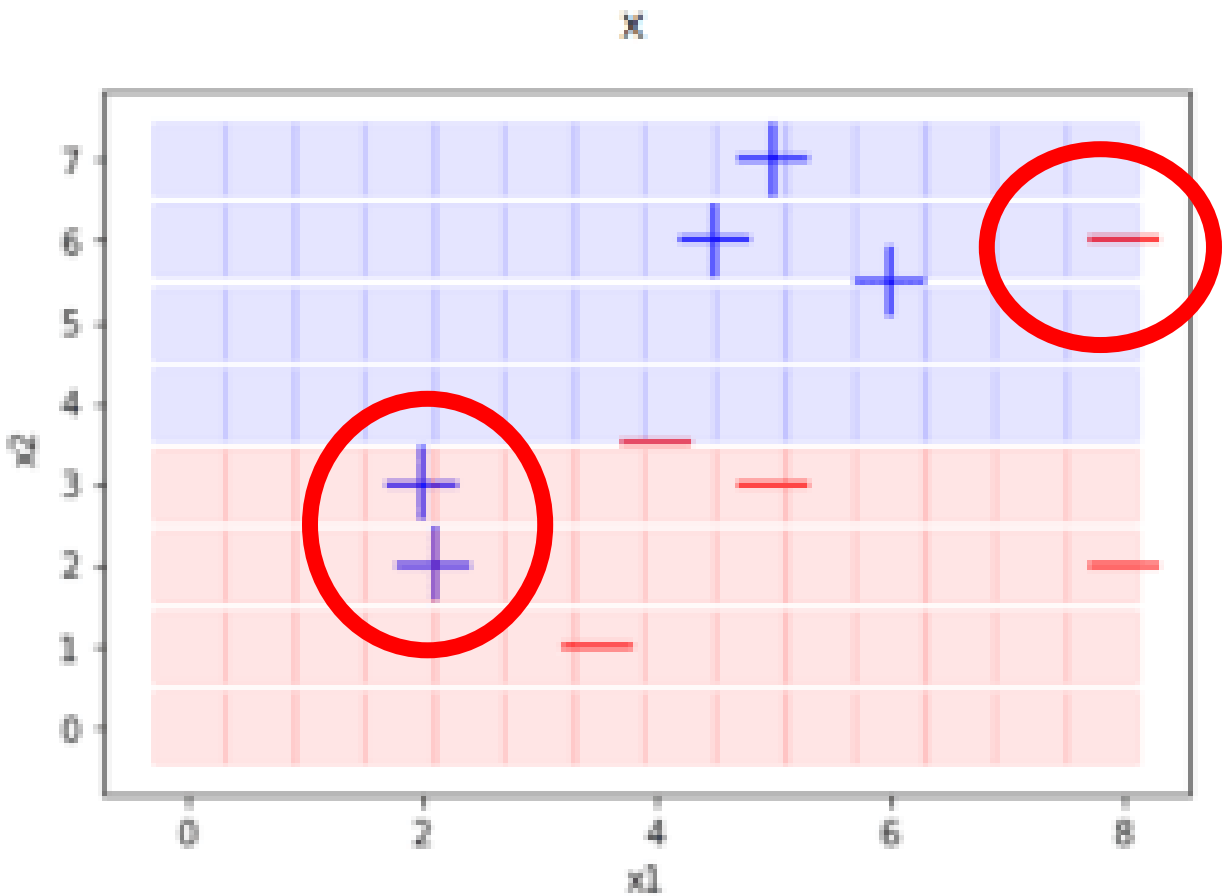


actual	w	Actual w
T	0.071	0.071
T	0.071	0.071
T	0.167	0.167
F	0.071	-0.071
F	0.071	-0.071
T	0.167	0.167
F	0.071	-0.071
T	0.167	0.167
F	0.071	-0.071
F	0.071	-0.071

ADA boosting 이론

Round 2

0.6496414920651304



actual	w	Pred
T	0.071	-1
T	0.071	-1
T	0.167	1
F	0.071	-1
F	0.071	-1
T	0.167	1
F	0.071	-1
T	0.167	1
F	0.071	1
F	0.071	-1

# ADA boosting 이론

actual	w	Pred
T	0.071	-1
T	0.071	-1
T	0.167	1
F	0.071	-1
F	0.071	-1
T	0.167	1
F	0.071	-1
T	0.167	1
F	0.071	1
F	0.071	-1

$$\varepsilon = \frac{\text{잘못 분류된 예제의 수}}{\text{전체 예제의 수}} = 0.213$$

$$\alpha = \frac{1}{2} \ln \left( \frac{1 - \varepsilon}{\varepsilon} \right) = \frac{1}{2} \ln \left( \frac{\text{올바르게 분류된 확률값}}{\text{잘못분류된 확률값}} \right)$$

= 0.65



# ADA boosting 이론

actual	w	Pred	w	Norm(w)
T	0.071	-1	0.137	0.167
T	0.071	-1	0.137	0.167
T	0.167	1	0.087	0.106
F	0.071	-1	0.037	0.045
F	0.071	-1	0.037	0.045
T	0.167	1	0.087	0.106
F	0.071	-1	0.037	0.045
T	0.167	1	0.087	0.106
F	0.071	-1	0.037	0.045
F	0.071	-1	0.037	0.045

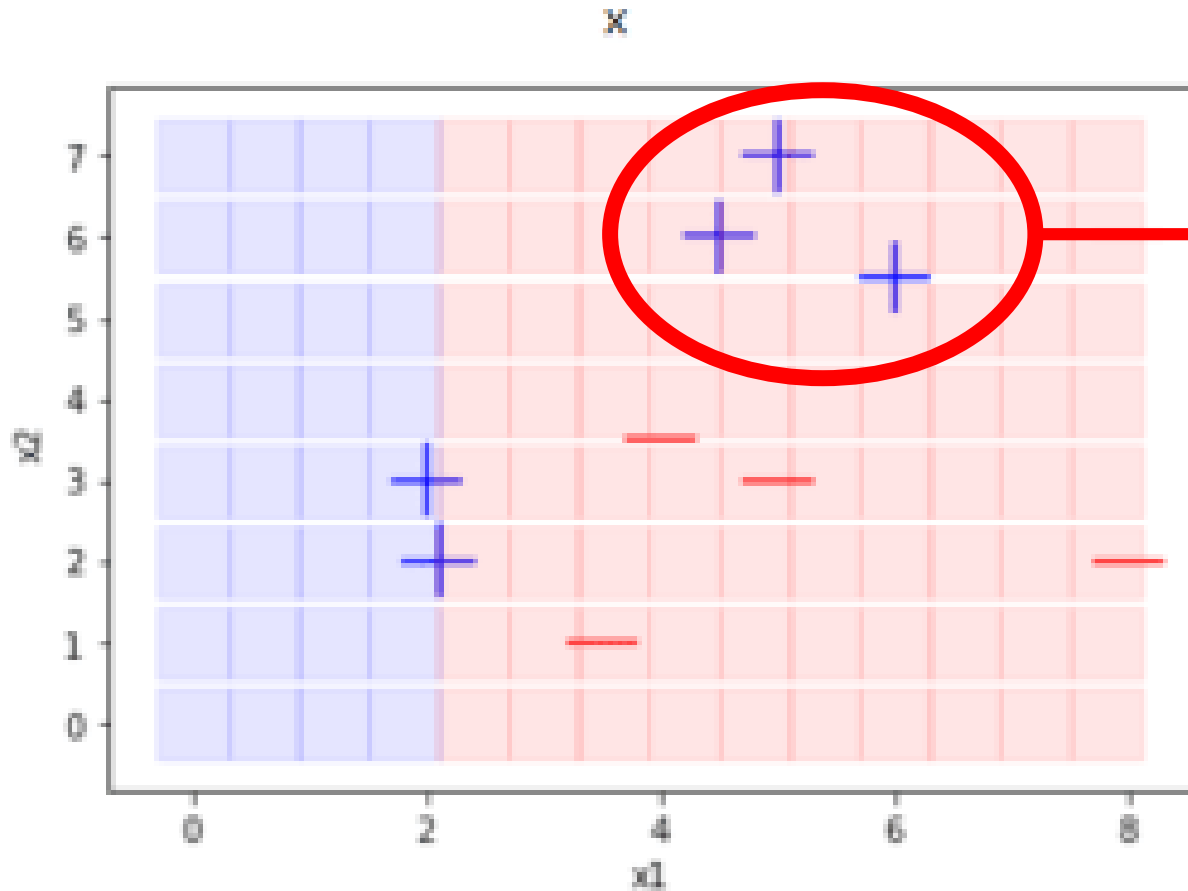
$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{\text{Sum}(D)} = (\text{확실하게 예측한 경우})$$

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{\text{Sum}(D)} = (\text{확실하게 예측하지 못한 경우})$$

# ADA boosting 이론

## Round 3

0.38107002602344847



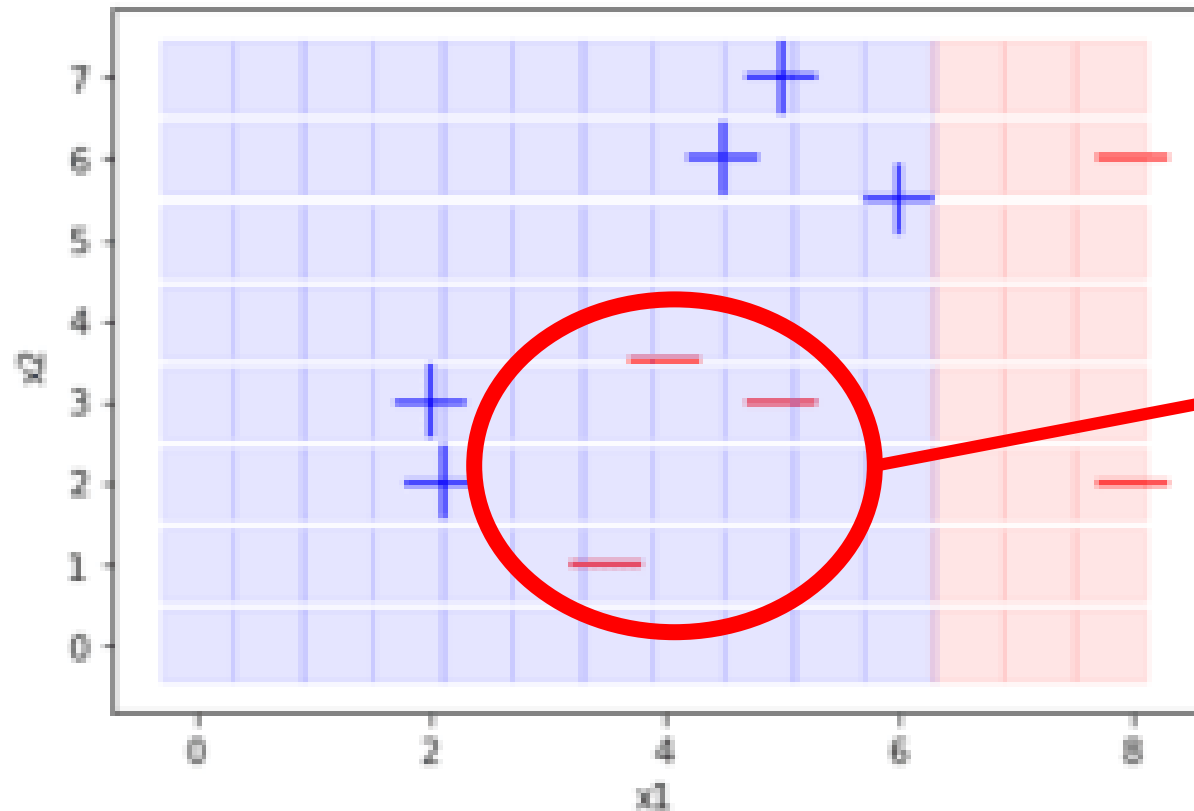
가중치 ↑

# ADA boosting 이론

## Round 4

1.0986122886681096

x



가중치 ↑

# ADA boosting 이론

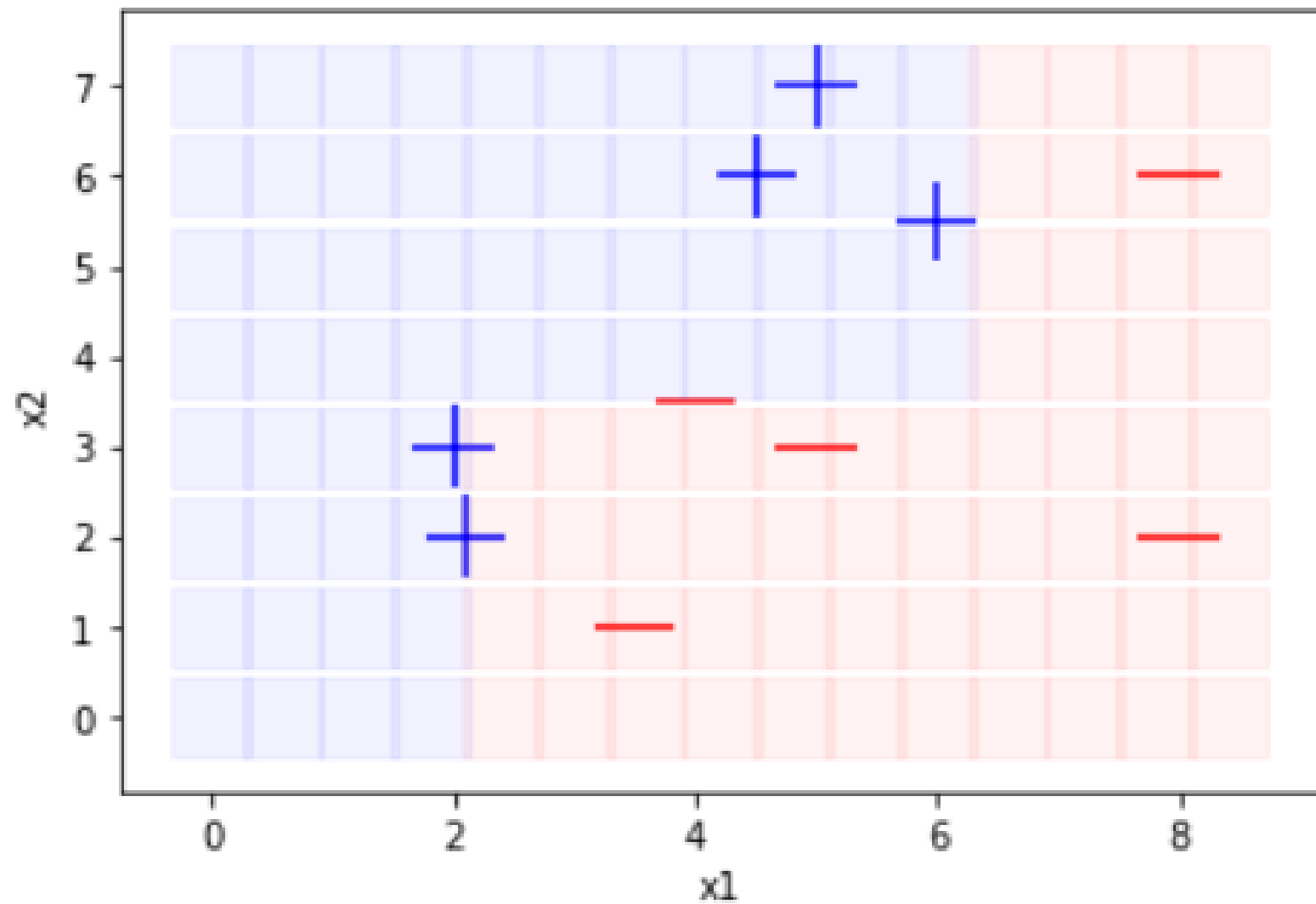
round 1 alpha	round 2 alpha	round 3 alpha	round 4 alpha
0.42	0.65	0.38	1.1
round 1 prediction	round 2 prediction	round 3 prediction	round 4 prediction
1	-1	1	1
1	-1	1	1
-1	1	-1	1
-1	-1	-1	1
-1	-1	-1	1
-1	1	-1	1
-1	-1	-1	1
-1	1	-1	1
-1	1	-1	-1
-1	-1	-1	-1

$$H_{final} = \text{sign}(\sum_t \alpha_t h_t(x))$$

$$\text{Prediction} = 0.42 \times 1 + 0.65 \times (-1) + 0.38 \times 1 + 1.1 \times 1 = 1.25$$

$$\text{Sign}(1.25) = +1 = \text{True}$$

# ADA boosting 이론



# ADA boosting 실습

Adabag 패키지의 boosting() 함수 사용

```
install.packages("adabag")  
library(adabag)
```

```
boosting(formula, data, boos = TRUE, mfinal = 100, coeflearn = 'Breiman',  
control,...)
```

- boos: 부트스트랩 허용 여부
- mfinal: 분류기 수

boosting을 실행하거나 사용할 트리의 수를 위한 반복 횟수 설정 (기본 mfinal = 100) .

- coeflear: alpha값 조정

**coeflearn = "Breiman" :  $\alpha = 1 / 2 \ln((1 - e_b) / e_b)$**

**coeflearn = "Freund" :  $\alpha = \ln((1 - e_b) / e_b)$**

**coeflearn = "Zhu" :  $\alpha = \ln((1 - e_b) / e_b) + \ln(k - 1)$**

- control: 앙상블에서 트리의 크기를 특별하게 제한하기 위하여 **boosting**에 전달

# ADA boosting 실습

```
> set.seed(300)
> m_adaboost <- boosting(default ~ ., data = credit)
> p_adaboost <- predict(m_adaboost, credit)
> head(p_adaboost$class)
[1] "no" "yes" "no" "no" "yes" "no"
```

- 예측값은 class 라는 서브객체에 저장.

```
> p_adaboost$confusion
      Observed class
Predicted class no yes
      no    700   0
      yes     0 300
```

- 과적합 발생.

# ADA boosting 실습

```
set.seed(300)
adaboost_cv <- boosting.cv(default ~ ., data = credit)
> adaboost_cv$confusion
```

	observed class	
Predicted class	no	yes
no	594	151
yes	106	149

- 교차 검증을 통해 과적합 방지.



페이지 6개 남았어요 뒤풀이는 달팽이 예상 ~



## ADA boosting 실습2

```
> library(adabag)
> data("iris")
> train <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
> iris.adaboost <- boosting(Species ~ ., data = iris[train, ], mfinal = 10,
control = rpart.control(maxdepth = 1))
```

- 최대 깊이 1 (stumps)의 10 개 트리의 boosting 앙상블을 가지고 분류

**결과는 매번 다릅니다!**

# ADA boosting 실습2

```
> iris.adaboost
$formula
Species ~ .

$trees
$trees[[1]]
n= 75

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 75 44 versicolor (0.2400000 0.4133333 0.3466667)
   2) Petal.Length < 4.85 49 18 versicolor (0.3673469 0.6326531 0.0000000) *
   3) Petal.Length >= 4.85 26 0 virginica (0.0000000 0.0000000 1.0000000) *
```

- 데이터는 랜덤으로 training, testing 으로 2가지로 나누어져 실행된다.

## ADA boosting 실습2

```
$weights  
[1] 0.3465736 0.1732868 0.4932428 0.4776335 0.3116010 0.3988886  
[7] 0.4133811 0.3343347 0.4243305 0.2873907
```

- 10 개의 작은 트리

트리의 에러가 낮을수록 가중치(weight)가 크다.

즉 에러가 낮을수록 최종결과에 미치는 영향이 크다.

# ADA boosting 실습2

setosa

\$votes	\$prob		
	[,1]	[,2]	[,3]
[1,]	2.0200181	0.923717	0.0000000
[2,]	2.0200181	0.923717	0.0000000
...			
[25,]	2.0200181	0.923717	0.0000000
[26,]	0.6337238	1.963438	0.3465736
[27,]	0.0000000	1.732765	1.2109701
...			
[50,]	0.6337238	1.788337	0.5216748
[51,]	0.0000000	1.039721	1.9040144
[52,]	0.0000000	1.039721	1.9040144
...			
[74,]	0.0000000	1.039721	1.9040144
[75,]	0.0000000	1.039721	1.9040144

- **votes** 와 **prob**는 각 데이터가 각 클래스 (setosa, versicolor, virginica) 로 부터 받은 투표와 소속할 확률을 나타낸다.

## ADA boosting 실습2

```
$class
[1] "setosa" "setosa" "setosa" "setosa" "setosa"
[6] "setosa" "setosa" "setosa" "setosa" "setosa"
[11] "setosa" "setosa" "setosa" "setosa" "setosa"
[16] "setosa" "setosa" "setosa" "setosa" "setosa"
[21] "setosa" "setosa" "setosa" "setosa" "setosa"
[26] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[31] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[36] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[41] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[46] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[51] "virginica" "virginica" "virginica" "virginica" "virginica"
[56] "virginica" "virginica" "virginica" "virginica" "virginica"
[61] "virginica" "virginica" "virginica" "virginica" "virginica"
[66] "virginica" "virginica" "virginica" "virginica" "virginica"
[71] "virginica" "virginica" "virginica" "virginica" "virginica"
```

class

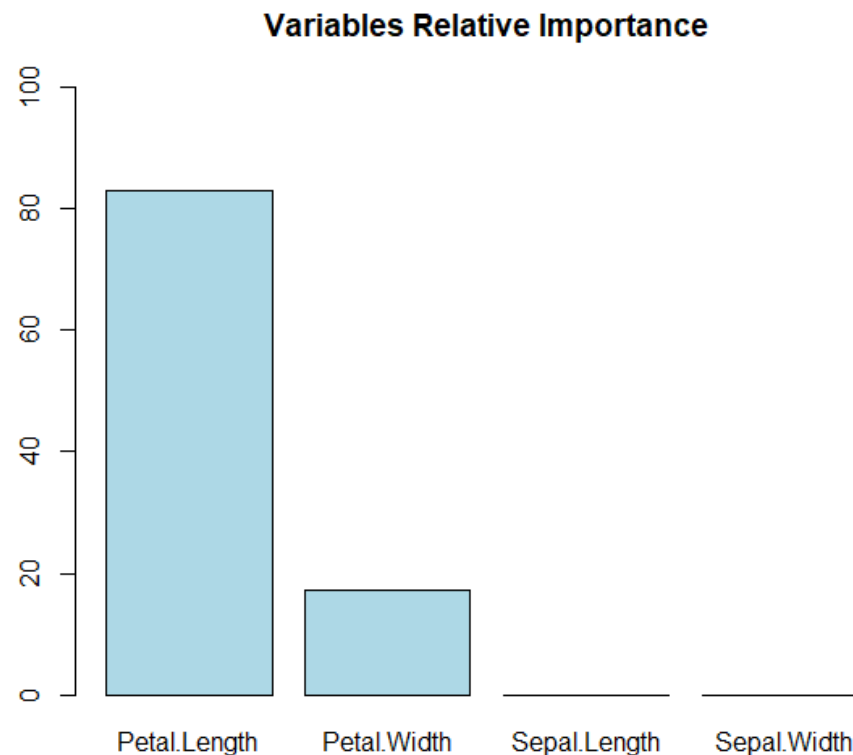
- 각 데이터가 분류된 결과를 볼수 있다.

# ADA boosting 실습2

```
$importance
Petal.Length Petal.Width
82.90007     17.09993
Sepal.Length Sepal.Width
0.00000     0.00000
```

- **importance** 벡터에서 **Petal.Length**를 가장 중요한 변수로 찾을 수 있다. **(82%의 정보 기여)**

```
barplot(iris.adaboost$imp[order(iris.adaboost$imp, decreasing = TRUE)], ylim = c(0, 100), main = "Variables Relative Importance", col = "lightblue")
```



## ADA boosting 실습2

```
R> table(iris.adaboost$class, iris$Species[train],  
+        dnn = c("Predicted Class", "Observed Class"))
```

	Observed Class		
Predicted Class	setosa	versicolor	virginica
setosa	25	0	0
versicolor	0	25	4
virginica	0	0	21

```
R> 1 - sum(iris.adaboost$class == iris$Species[train]) /  
+        length(iris$Species[train])
```

```
[1] 0.05333333
```

- iris.adaboost 객체로 부터 훈련 집합의 혼동 행렬(confusion matrix)을 만들고 에러 계산
- irginica로 부터 4개의 꽃이 versicolor로 구분된다, 그래서 도달한 에러는 5.33% 이다.