机器学习

主讲教师：刘峤

第5章 Emsemble Methods

Bagging and Boosting

Adaptive Boosting

Bootstrapping as Re-weighting Process

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}$$

$$\overset{bootstrap}{\Longrightarrow} \quad \mathcal{D}_t = \{(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_4, y_4)\}$$

- E_{in} **on** $\mathcal{D}_t$ **:** $\quad E_{in}(h) = \frac{1}{4} \sum_{(\mathbf{x},y)\in\mathcal{D}_t} ind\{y \neq h(\mathbf{x})\}$
- **weighted** E_{in} **on** $\mathcal{D}$

$$E_{in}(\alpha, h) = \frac{1}{4} \sum_{i=1}^{4} \alpha_i \cdot ind\{y_i \neq h(\mathbf{x}_i)\}$$

$$(\mathbf{x}_1, y_1) : \alpha_1 = 2 \quad (\mathbf{x}_2, y_2) : \alpha_2 = 1 \quad (\mathbf{x}_3, y_3) : \alpha_3 = 0 \quad (\mathbf{x}_4, y_4) : \alpha_4 = 0$$

- **each diverse g_t in bagging:**
  - **by minimizing bootstrap-weighted error**

Re-weighting for More Diverse Hypothesis

- improving bagging for binary classification:
  - how to re-weight for more diverse hypotheses?

$$g_t \leftarrow \underset{h \in \mathcal{H}}{argmin} \left(\sum_{i=1}^{N} \alpha_i^t \cdot ind\{y_i \neq h(\mathbf{x}_i)\} \right)$$

$$g_{t+1} \leftarrow \underset{h \in \mathcal{H}}{argmin} \left(\sum_{i=1}^{N} \alpha_i^{t+1} \cdot ind\{y_i \neq h(\mathbf{x}_i)\} \right)$$

  - if g_t not good for α^{t+1}, then:
    - g_t-like hypotheses not returned as g_{t+1}
    - g_{t+1} diverse from g_t
  - idea: construct α^{t+1} to make g_t random-like

$$\sum_{i=1}^{N} \alpha_i^{t+1} \cdot ind\{y_i \neq g_t(\mathbf{x}_i)\} = \frac{1}{2} \sum_{i=1}^{N} \alpha_i^{t+1}$$

Optimal Re-weighting

- **Let :** $e^{t+1} = \sum_{i=1}^{N} \alpha_i^{t+1} \cdot ind\{y_i \neq g_t(\mathbf{x}_i)\}$ **denotes total** α_i^{t+1} **of incorrect**
- **Let :** $r^{t+1} = \sum_{i=1}^{N} \alpha_i^{t+1} \cdot ind\{y_i = g_t(\mathbf{x}_i)\}$ **denotes total** α_i^{t+1} **of correct**
- **Want :** $\sum_{i=1}^{N} \alpha_i^{t+1} \cdot ind\{y_i \neq g_t(\mathbf{x}_i)\} = \frac{1}{2} \sum_{i=1}^{N} \alpha_i^{t+1}$
- one possibility by **re-scaling (multiplying) weights**, if

➔ **total** α_i^t **of incorrect = 1126 ;** **total** α_i^t **of correct = 6211**

➔ **weighted incorrect rate = 1126 / 7337**

➔ **incorrect :** $\alpha_i^{t+1} \leftarrow \alpha_i^t \cdot 6211$ ➔ **correct :** $\alpha_i^{t+1} \leftarrow \alpha_i^t \cdot 1126$

Decision Stump

want: a '**weak**' base learning algorithm $\mathcal{A}$
that minimizes $E_{\text{in}}^{\mathbf{u}}(h) = \frac{1}{N}\sum_{n=1}^{N} u_n \cdot [\![y_n \neq h(\mathbf{x}_n)]\!]$ **a little bit**

a popular choice: decision stump

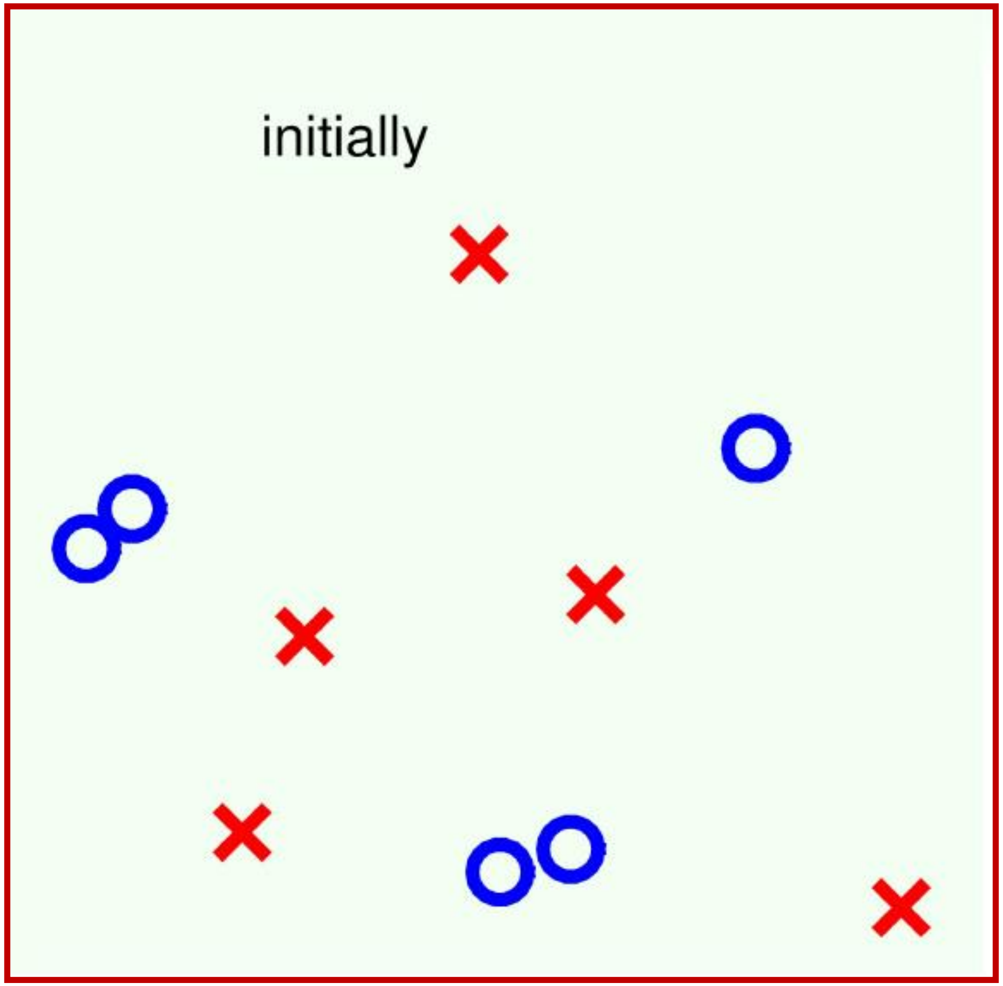
- **in ML Foundations Homework 2, remember? :-)**

$$h_{s,i,\theta}(\mathbf{x}) = s \cdot \text{sign}(x_i - \theta)$$

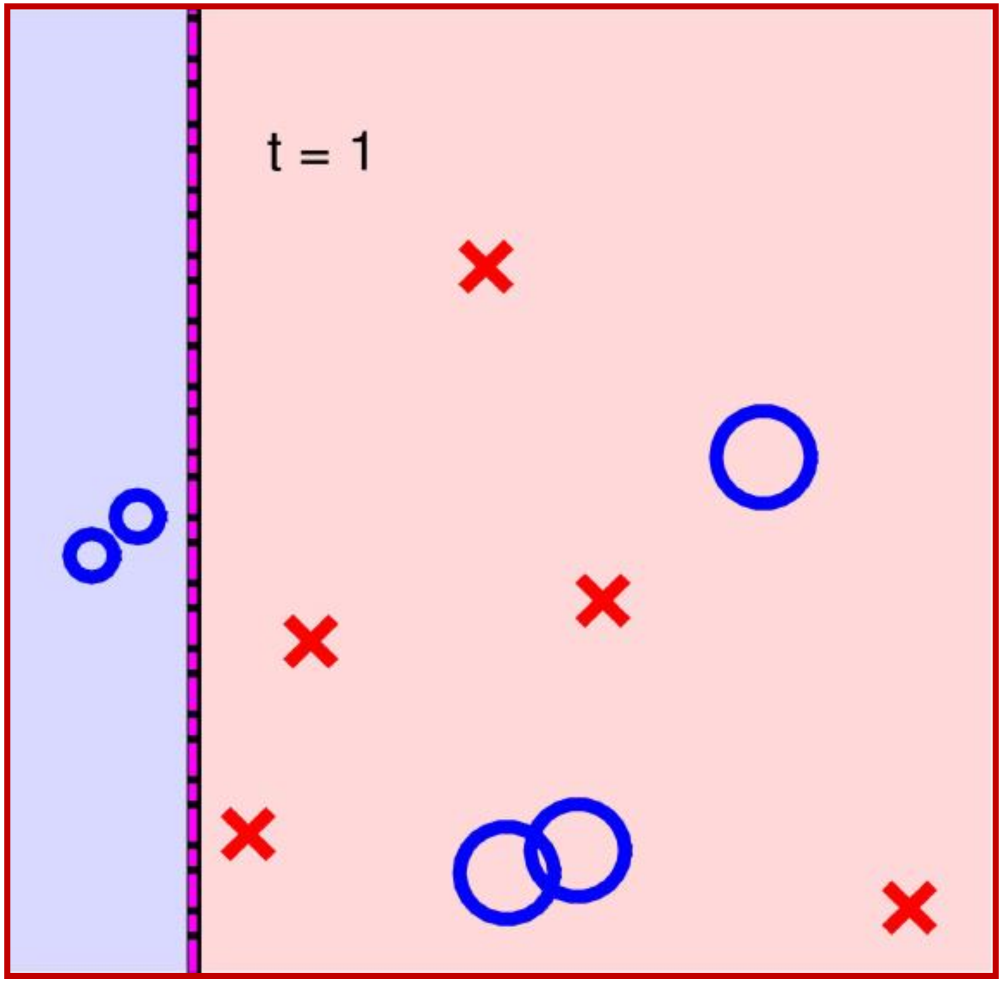
- positive and negative rays on some feature: three parameters (feature i, threshold θ, direction s)
- physical meaning: vertical/horizontal lines in 2D
- efficient to optimize: $O(d \cdot N \log N)$ time

decision stump model:
allows efficient minimization of $E_{\text{in}}^{\mathbf{u}}$
but perhaps **too weak to work by itself**

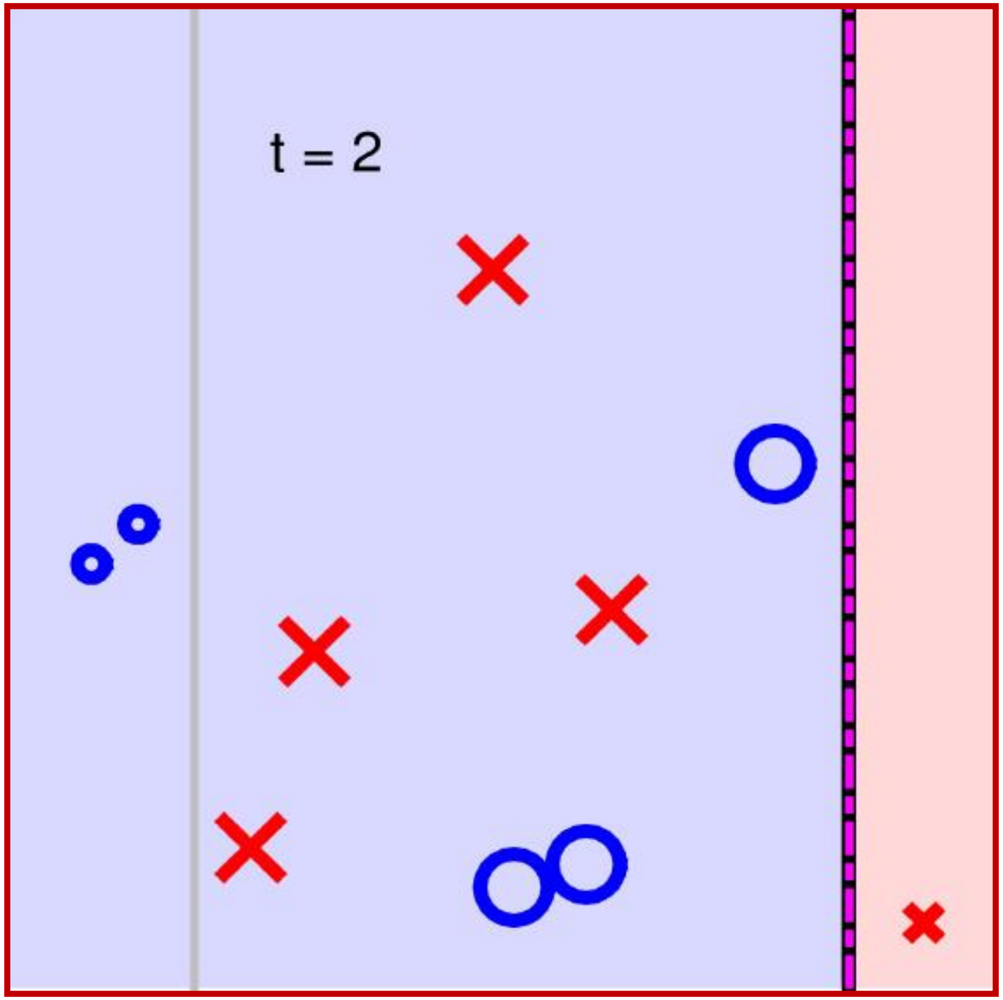
A Simple Data Set



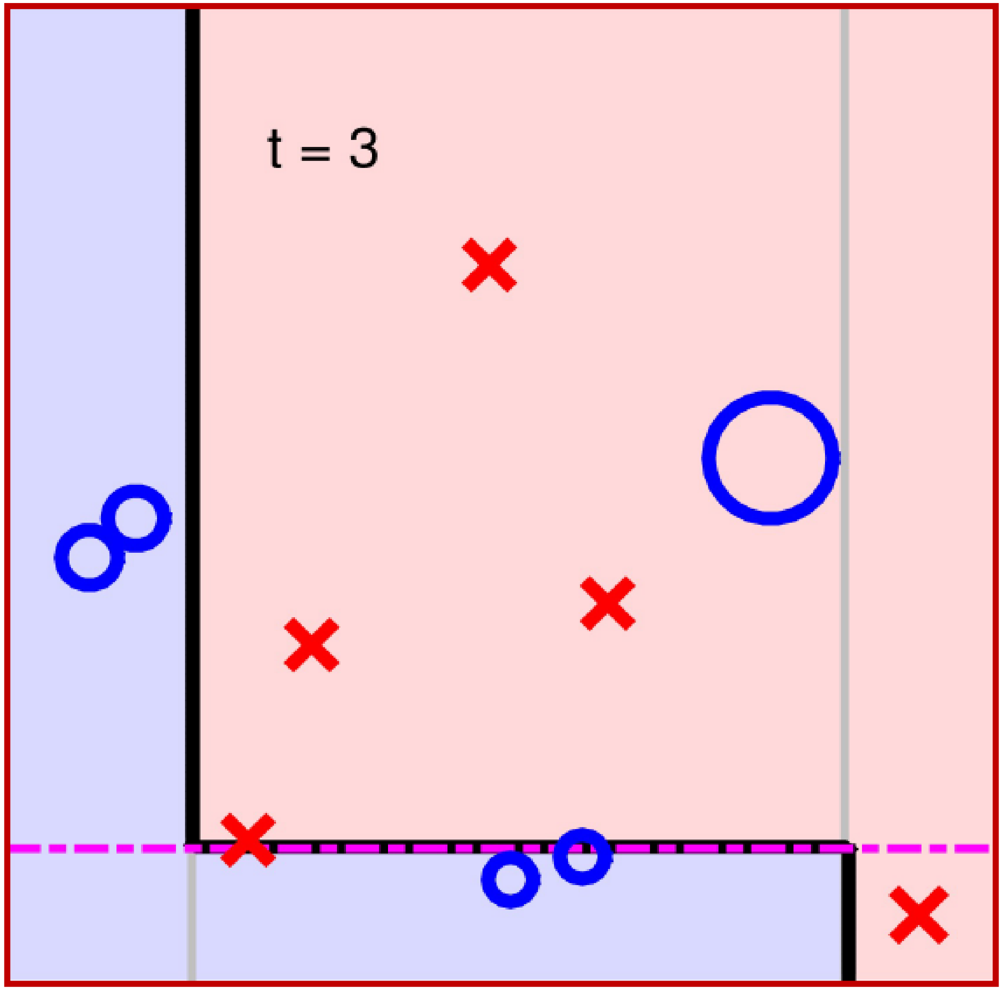
A Simple Data Set



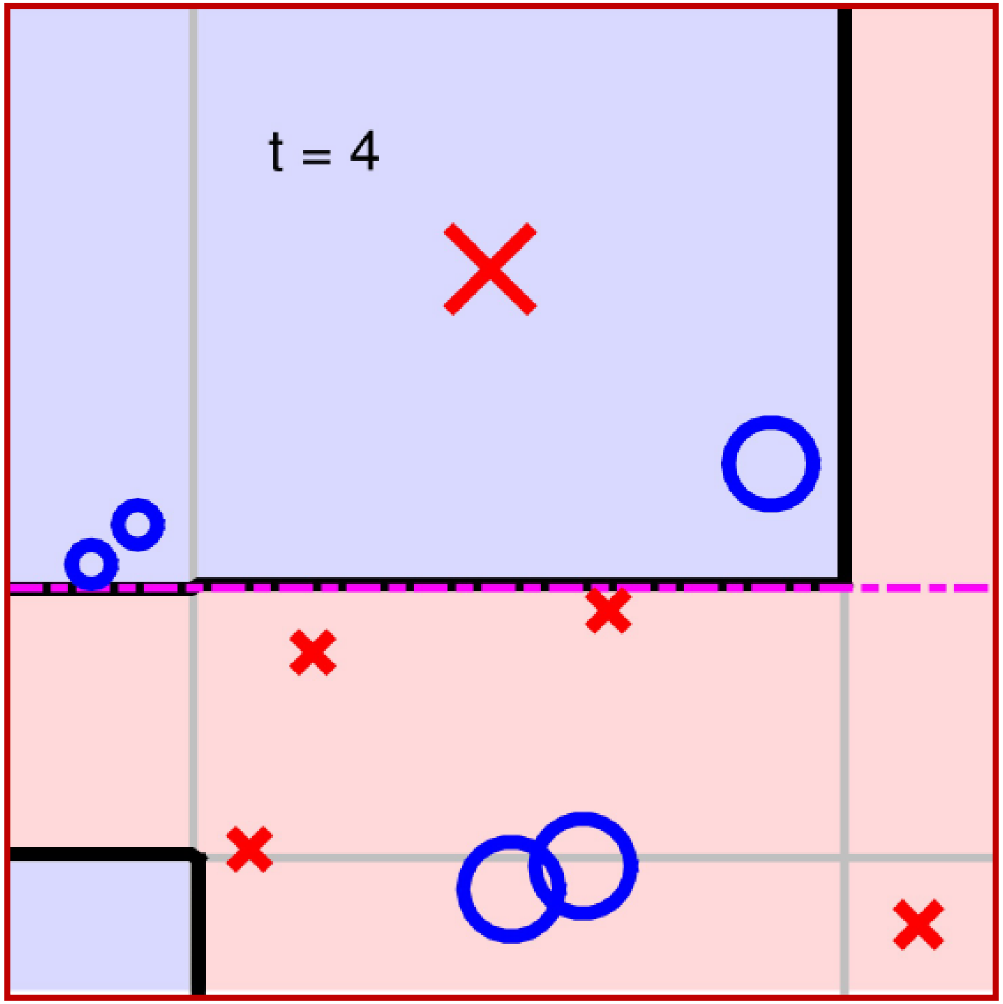
A Simple Data Set



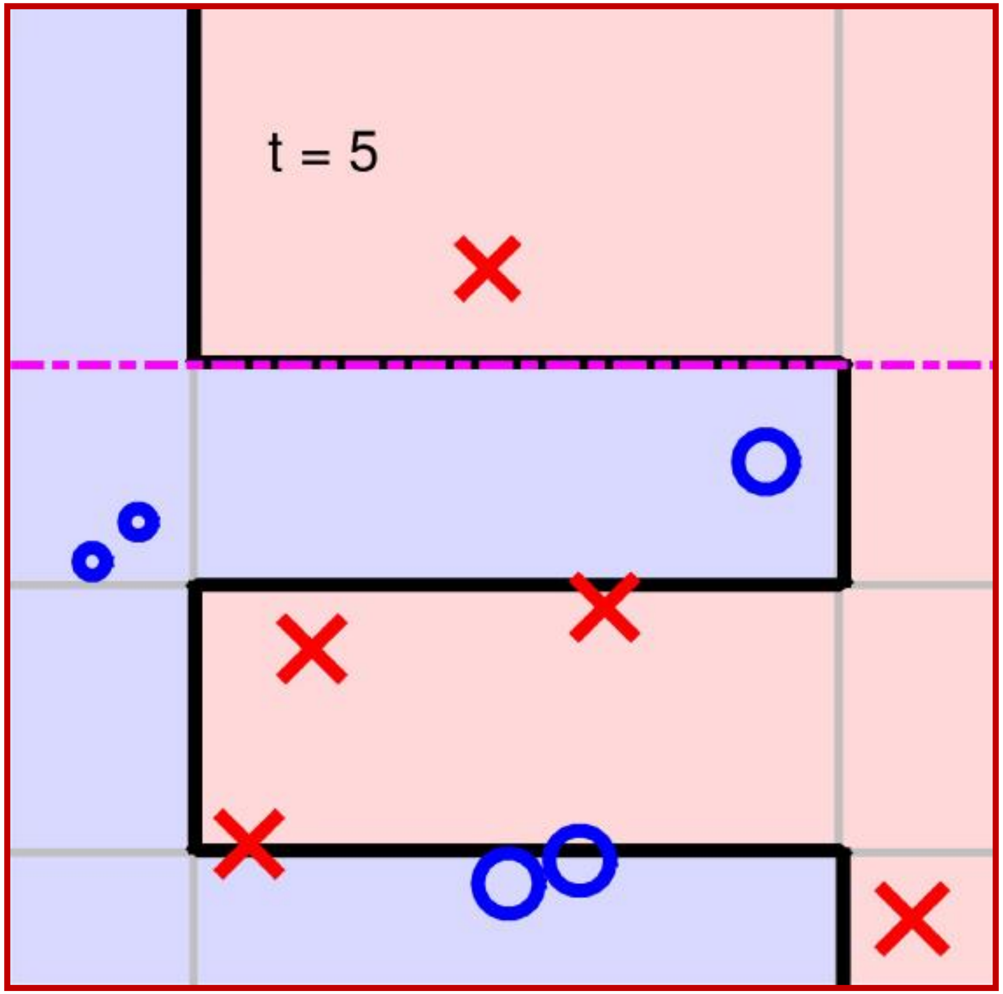
A Simple Data Set



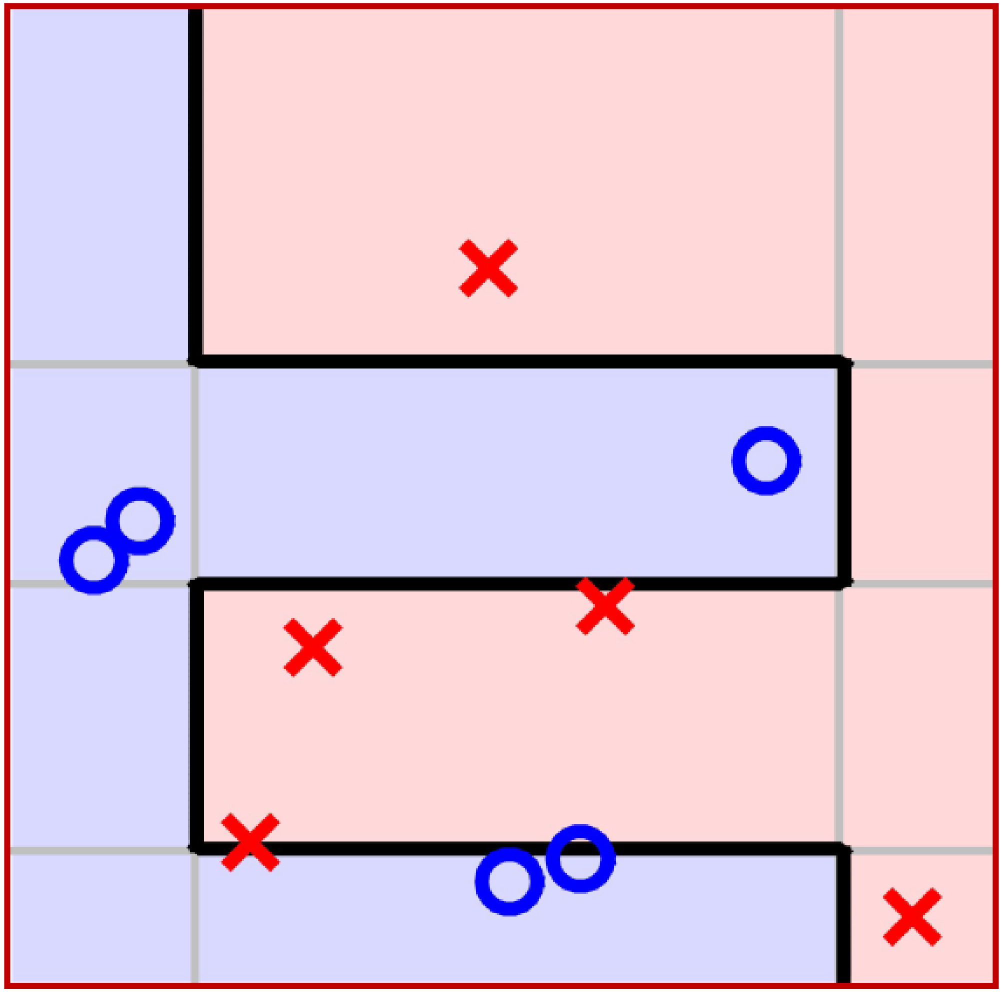
A Simple Data Set



A Simple Data Set

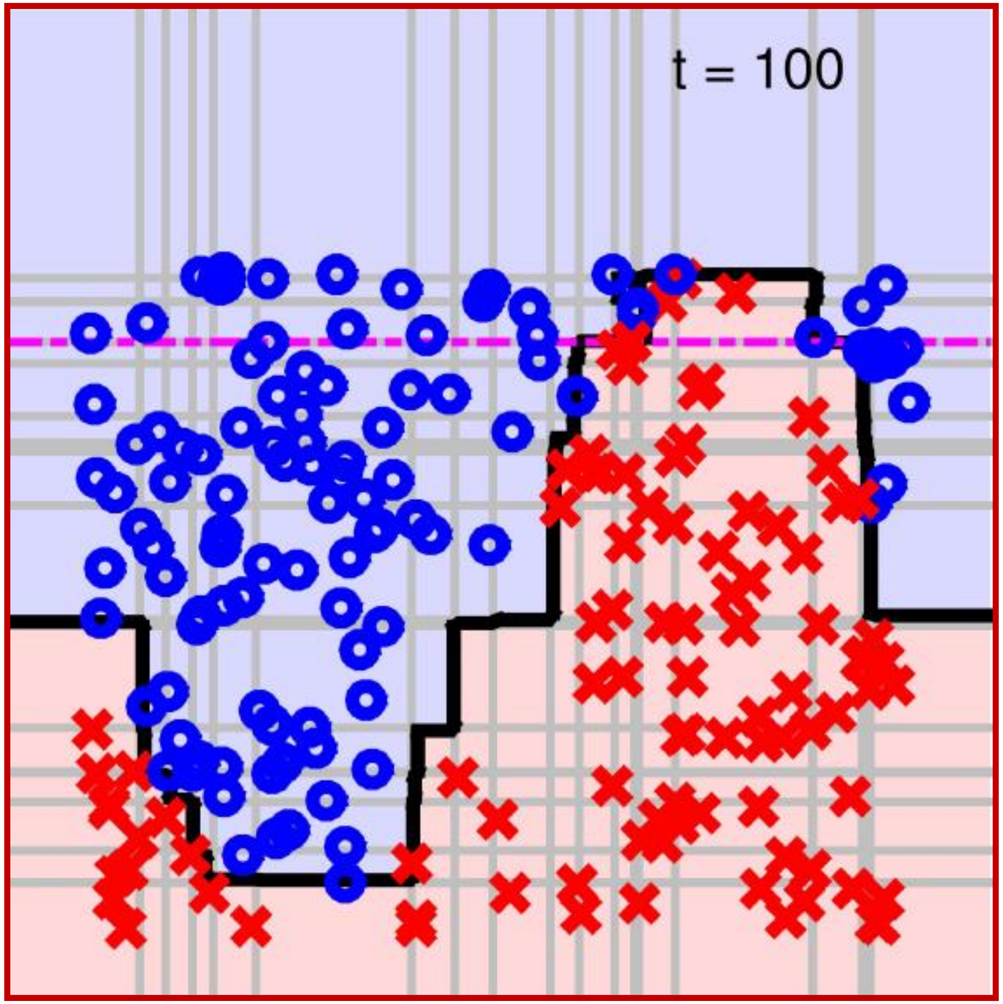


A Simple Data Set



Teacher-like algorithm works!

A Complicated Data Set



AdaBoost-Stump: **non-linear yet efficient**

Quize

For four examples with $\alpha_i^{(1)} = 1/4$ for all examples. If g_1 predicts the first example wrongly but all the other three examples correctly. After the optimal re-weighting, what is $\alpha_1^{(2)}/\alpha_2^{(2)}$

Reference Answer: 2

(1) 4 **(2) 3** **(3) 1/3** **(4) 1/4**

By optimal re-weighting, α_1 is scaled proportional to 3/4 and every other α_i is scaled proportional to 1/4 . So example 1 is now three times more important than any other example.

Bagging : Bootstrap Aggregation

- **optimal re-weighting**: let $\epsilon_t = \sum_{i=1}^{N} \alpha_i^{t+1} \cdot ind\{y_i \neq g_t(\mathbf{x}_i)\} / \sum_{i=1}^{N} \alpha_i^{t+1}$
  - **multiply incorrect** $\propto (1 - \epsilon_t)$ **; multiply correct** $\propto \epsilon_t$

- **define scaling factor:** $\lambda_t = \sqrt{(1 - \epsilon_t)/\epsilon_t}$
  - **incorrect ← incorrect** $\cdot \lambda_t$ **; correct ← correct** $/\lambda_t$
  - **equivalent to optimal re-weighting:** $\lambda_t \geq 1 \text{ iff } \epsilon_t \leq 1/2$
  - **physical meaning: scale up incorrect; scale down correct**

- **scaling-up incorrect examples leads to diverse hypotheses!**

A Preliminary Algorithm

- for $t = 1, 2, \cdots, T$
  1. obtain g_t by $\mathcal{A}(D, \alpha^{(t)})$

     where $\mathcal{A}$ tries to minimize $\alpha^{(t)}$-weighted 0/1 error
  2. update $\alpha^{(t)}$ to $\alpha^{(t+1)}$ by $\lambda_t = \sqrt{(1-\epsilon_t)/\epsilon_t}$

     where ϵ_t = weighted error (incorrect) rate of g_t
- return G(x)

- $\alpha^{(1)}$ =? want g_1 "best" for E_{in}: $\alpha_i^{(1)} = 1/N$
- G(x) = ? uniform? -- but g_2 very bad for E_{in} (why? :-))
  - linear, non-linear? as you wish!

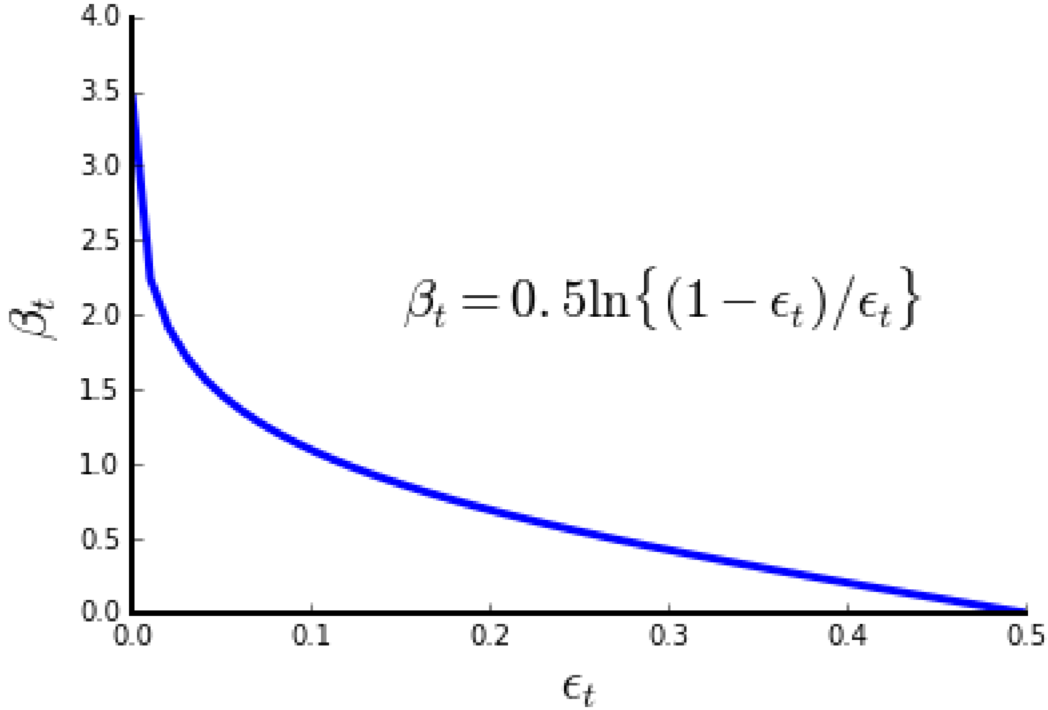
Linear Aggregation on the Fly

- $\alpha^{(1)} = [1/N, 1/N, \ldots, 1/N]$, for $t = 1, 2, \cdots, T$
  1. obtain g_t by $\mathcal{A}(D, \alpha^{(t)})$
  2. update $\alpha^{(t)}$ to $\alpha^{(t+1)}$ by $\lambda_t = \sqrt{(1-\epsilon_t)/\epsilon_t}$
  3. compute $\beta_t = \ln(\lambda_t)$
- return $G(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t g_t(\mathbf{x})\right)$

- wish: large β_t for good g_t $\Rightarrow \beta_t = f(\lambda_t)$ monotonic
- will take $\beta_t = \ln(\lambda_t)$
  - $\epsilon_t = 0.5 \Rightarrow \lambda_t = 1 \Rightarrow \beta_t = 0$ (bad g_t zero weight)
  - $\epsilon_t = 0 \Rightarrow \lambda_t = \infty \Rightarrow \beta_t = \infty$ (super gt superior weight)

Linear Aggregation on the Fly

- wish: large β_t for good $g_t \Rightarrow \beta_t = f(\lambda_t)$ monotonic



Essentially, the weight of the classifier in the ensemble is proportional to the log-odds of it being correct vs making an error

assuming $0 < \epsilon_t < 0.5$

Adaptive Boosting

Adaptive Boosting = weak base learning algorithm $\mathcal{A}$

optimal re-weighting factor λ_t

"magic" linear aggregation β_t

AdaBoost: provable **boosting property**

Theoretical Guarantee of AdaBoost

- From VC bound

$$E_{out}(G) \leq E_{in}(G) + O\left(\sqrt{O(d_{VC}(\mathcal{H}) \cdot T \log T) \cdot \frac{\log N}{N}}\right)$$

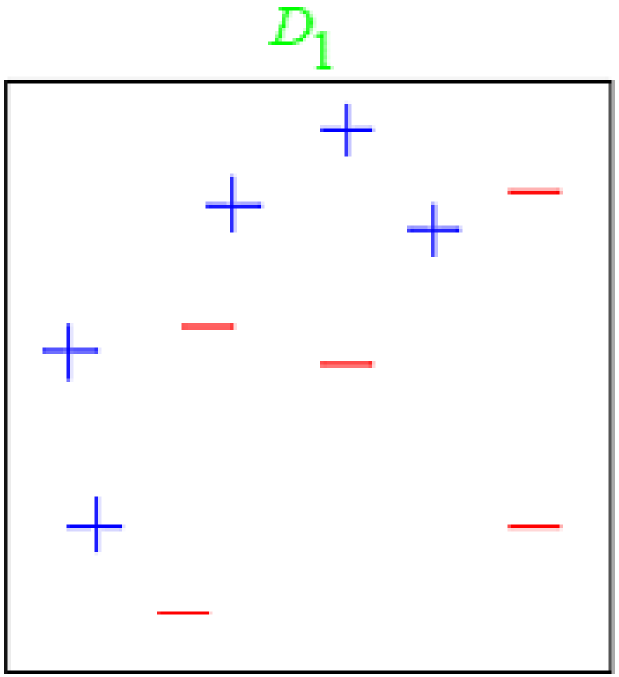
d_{VC} of all possible *G*

- $E_{in}(G)$ can be small:
  - $E_{in}(G) = 0$ after $\mathrm{T} = \mathrm{O}(\log N)$ iterations if: $\epsilon_t \leq \epsilon < 0.5$
- second term can be small:
  - overall d_{VC} grows "slowly" with T

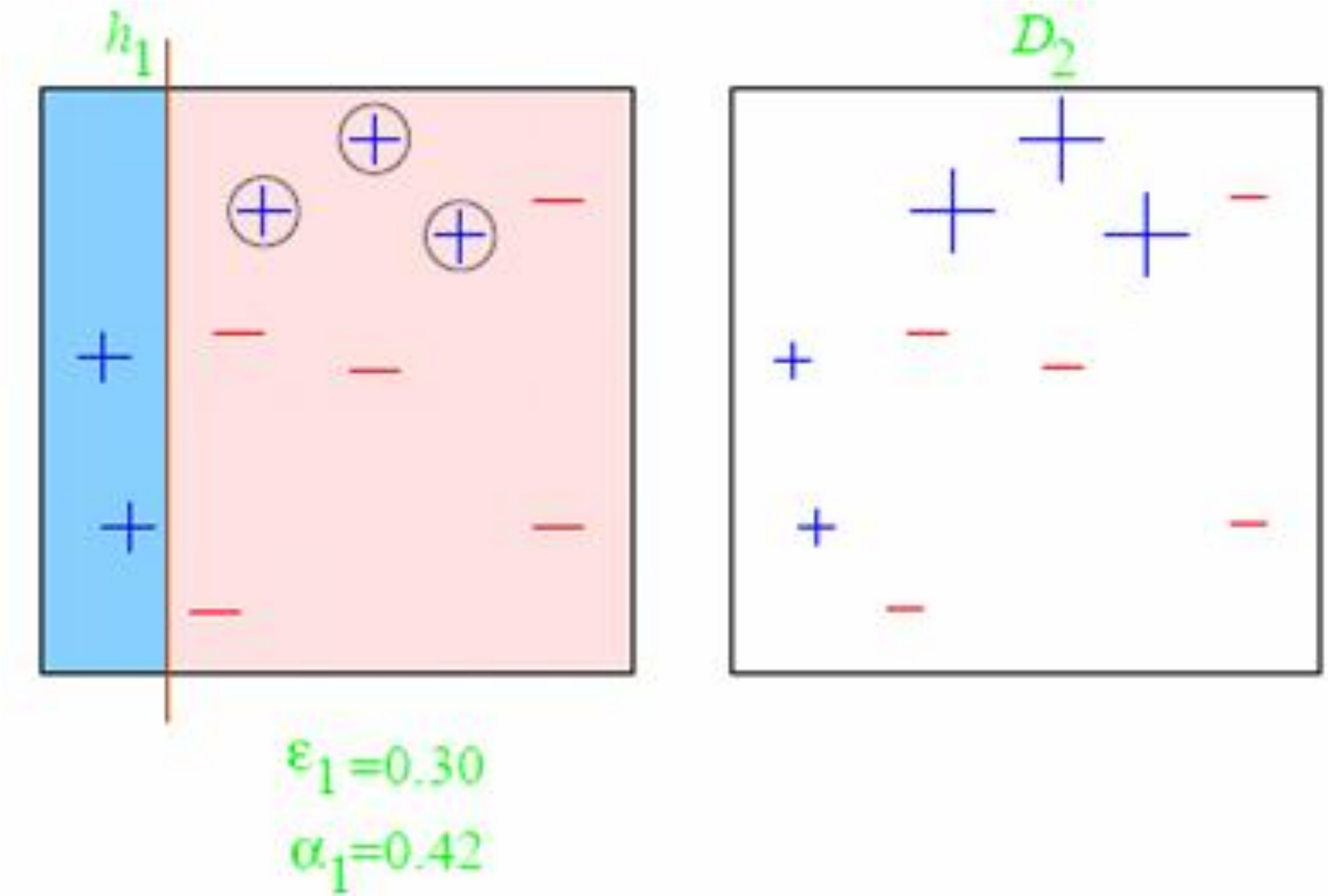
boosting view of AdaBoost:

- if $\mathcal{A}$ is weak but always slightly better than random ($\epsilon_t \leq \epsilon < 0.5$),
- then (AdaBoost+$\mathcal{A}$) can be strong ($E_{in} = 0$ and E_{out} small)

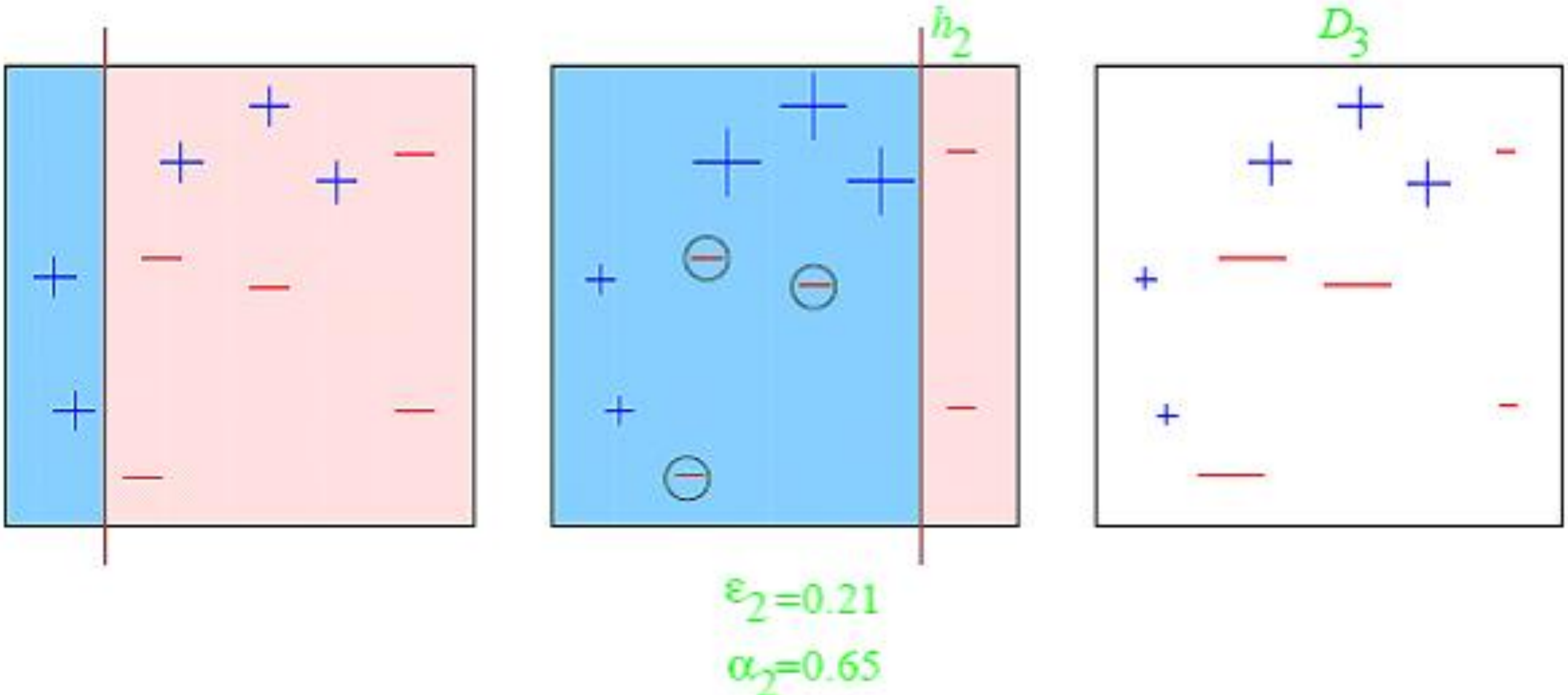
A toy example from Schapire's tutorial



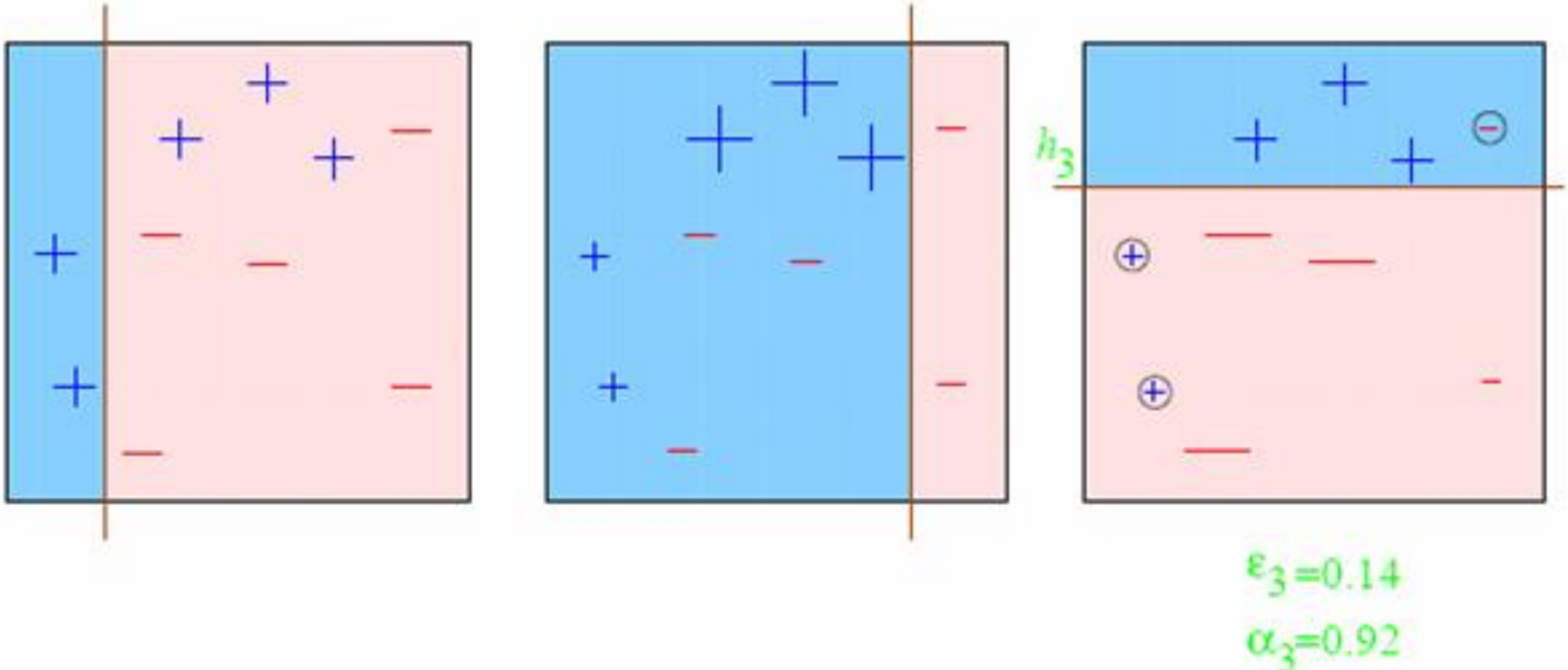
The second round:



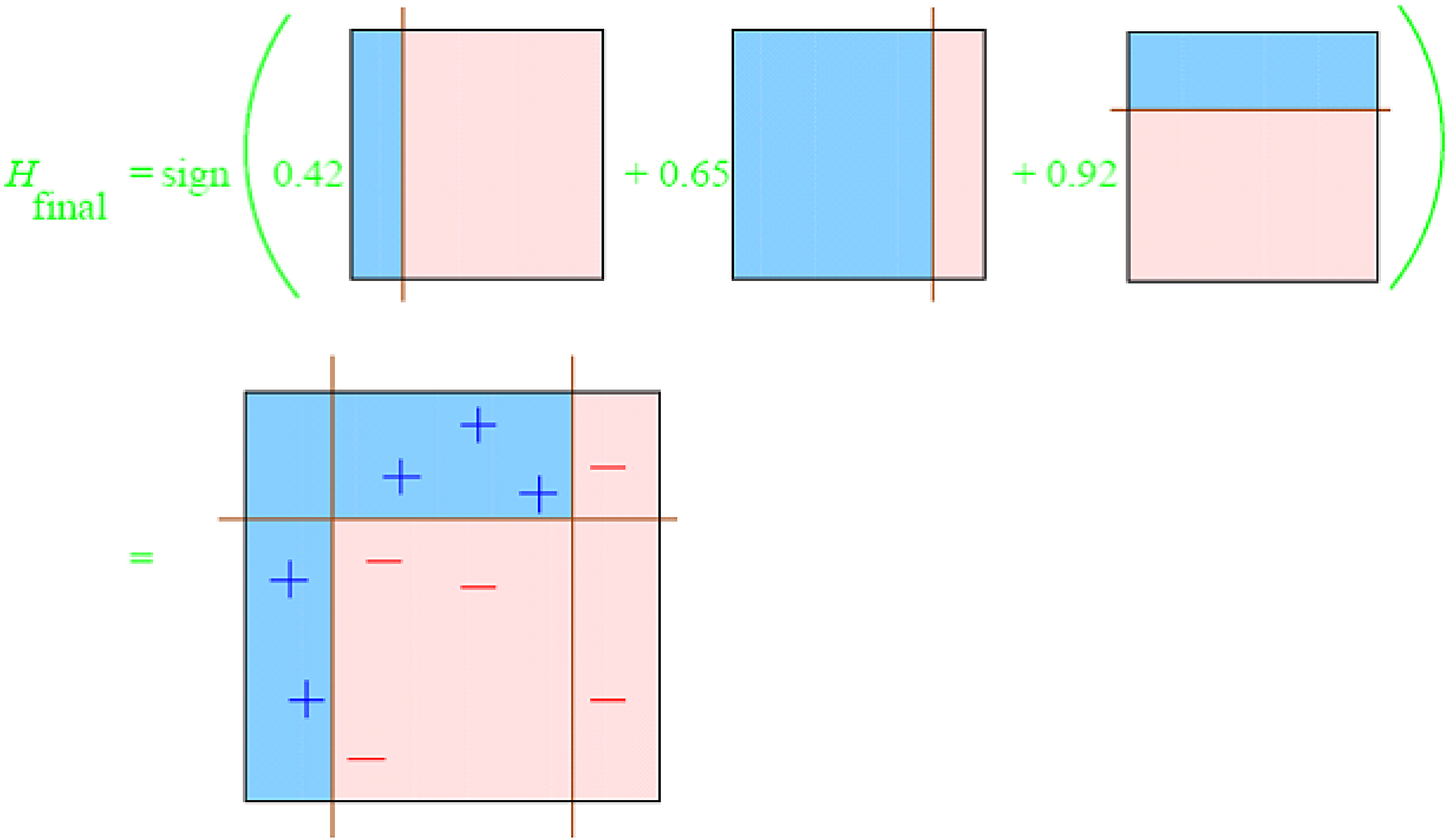
The first round:



The third round:



The final classifier



AdaBoost Algorithm

- $\alpha^{(1)} = [1/N, 1/N, \ldots, 1/N]$, for $t = 1, 2, \cdots, T$
  1. obtain g_t by $\mathcal{A}(D, \alpha^{(t)})$

     where $\mathcal{A}$ tries to minimize $\alpha^{(t)}$ -weighted 0/1 error
  2. update $\alpha^{(t)}$ to $\alpha^{(t+1)}$ by $\lambda_t = \sqrt{(1-\epsilon_t)/\epsilon_t}$

     incorrect examples： $\alpha^{(t+1)} = \alpha^{(t)} \cdot \lambda_t$

     incorrect examples： $\alpha^{(t+1)} = \alpha^{(t)} / \lambda_t$

     where： $\epsilon_t = \sum_{i=1}^{N} \alpha_i^t \cdot ind\{y_i \neq g_t(\mathbf{x}_i)\} / \sum_{i=1}^{N} \alpha_i^t$
  3. compute $\beta_t = \ln(\lambda_t)$
- return $G(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \beta_t g_t(\mathbf{x})\right)$

Quize

According to $\beta_t = \ln(\lambda_t)$ and $\lambda_t = \sqrt{(1-\epsilon_t)/\epsilon_t}$,

when would $\beta_t > 0$?

(1) $\epsilon_t < 0.5$ **(2)** $\epsilon_t > 0.5$ **(3)** $\epsilon_t \neq 1$ **(4)** $\epsilon_t \neq 0$

Reference Answer: 1

The math part should be easy for you, and it is interesting to think about the physical meaning: $\beta_t > 0$ (gt is useful for G) if and only if the weighted error rate of gt is better than random!

Quize

For a data set of size 9876 that contains $\mathbf{x}_n \in \mathbb{R}^{5566}$, after running AdaBoost-Stump for 1126 iterations, what is the number of distinct features within $\mathbf{x}$ that are effectively used by G?

1. $0 \leq \text{number} \leq 1126$
2. $1126 < \text{number} \leq 5566$
3. $5566 < \text{number} \leq 9876$
4. $9876 < \text{number}$

Each decision stump takes only one feature. So 1126 decision stumps need at most 1126 distinct features.

Reference Answer: 1

Decision Tree

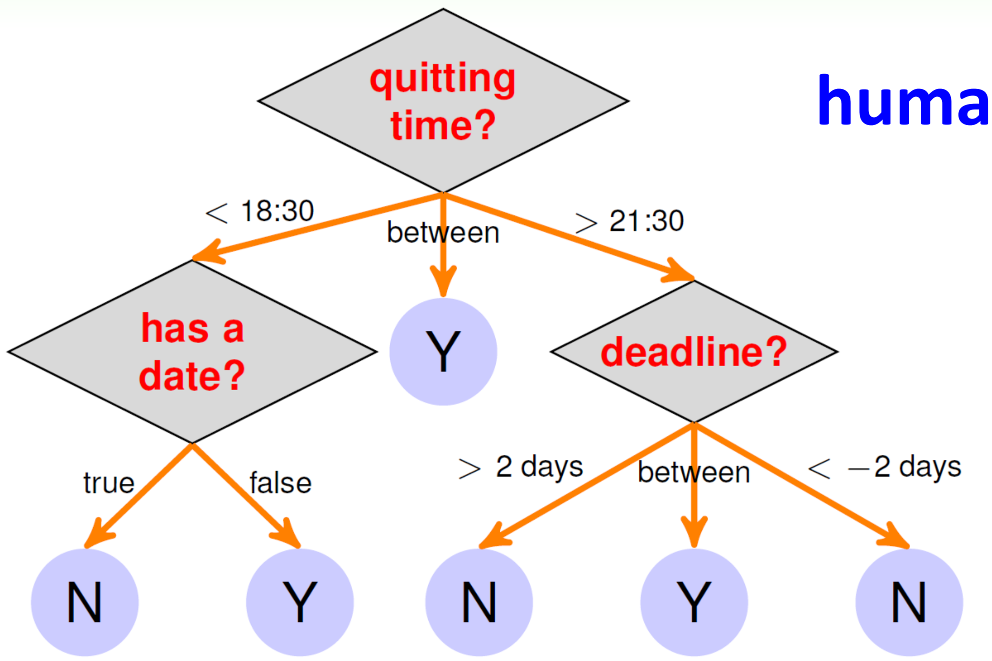
What We Have Done

- blending: aggregate after getting gt ;
- learning: aggregate as well as getting gt

aggregation type	blending	learning
uniform	voting/averaging	Bagging
non-uniform	linear	AdaBoost
conditional	stacking	Decision Tree

decision tree: a traditional learning model that realizes conditional aggregation

Decision Tree : Path View

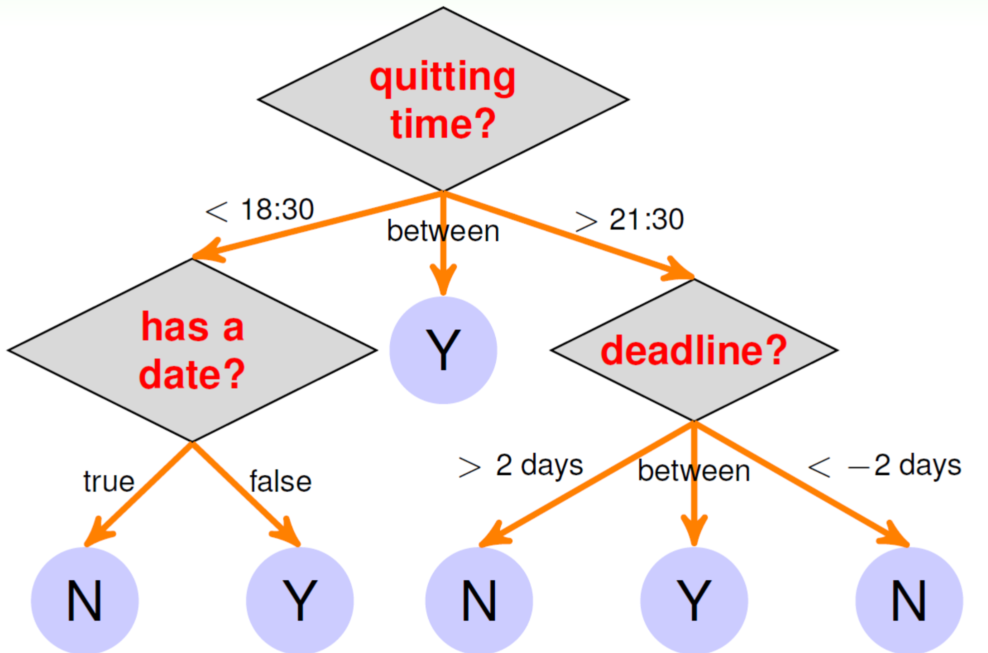


human-mimicking models

$$G(\mathbf{x}) = \sum_{t=1}^{N} q_t(\mathbf{x}) \cdot g_t(\mathbf{x})$$

- **base hypothesis** g_t (x): leaf at end of path t, a **constant** here
- **condition** q_t (x): *ind* (is x on path t?)
- usually with simple **internal nodes**

Decision Tree : Recursive View



$$G(\mathbf{x}) = \sum_{c=1}^{C} ind\{b(\mathbf{x}) = c\} \cdot G_c(\mathbf{x})$$

- G(x): full-tree hypothesis
- b(x): branching criteria
- G_c(x): sub-tree hypothesis at the c-th branch

Disclaimers about Decision Tree

- Usefulness
  - human-explainable: widely used in business/medical data analysis
  - simple: easy to be implemented
  - efficient in prediction and training
- However......
  - heuristic: mostly little theoretical explanations
  - heuristics: ‘heuristics selection’ confusing to beginners
  - arguably no single representative algorithm
- decision tree: mostly heuristic but useful on its own

Quize

The following C-like code can be viewed as a decision tree of three leaves.

```
if (income > 100000) return true;
else {
    if (debt > 50000) return false;
    else return true;
}
```

Reference Answer: 2

What is the output of the tree for (income, debt) = (98765; 56789)?

(1) True (2) False (3) 98765 (4) 56789

- You can simply trace the code.
- The tree expresses a complicated boolean condition
  - income > 100000 or debt <= 50000

A Basic Decision Tree Algorithm

$$G(\mathbf{x}) = \sum_{c=1}^{C} ind\{b(\mathbf{x}) = c\} \cdot G_c(\mathbf{x})$$

- function **DecisionTree** (data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}, i = 1 \cdots N$)
- **if** termination criteria met
  - return base hypothesis $g_t(\mathbf{x})$
- **else**
  1. learn branching criteria $b(\mathbf{x})$
  2. split $\mathcal{D}$ to C parts $\mathcal{D}_c = \{(\mathbf{x}_i, \mathbf{y}_i) : b(\mathbf{x}_i) = c\}$
  3. build sub-tree $G_c \leftarrow$ **DecisionTree**$(\mathcal{D}_c)$
  4. return $G(\mathbf{x}) = \sum_{c=1}^{C} ind\{b(\mathbf{x}) = c\} \cdot G_c(\mathbf{x})$

Classification and Regression Tree (C&RT)

- function **DecisionTree** (data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}, i = 1 \cdots N$)
- if termination criteria met : return base hypothesis $g_t(\mathbf{x})$
- else : (1) learn branching criteria $b(\mathbf{x})$
- (2) split $\mathcal{D}$ to **C** parts $\mathcal{D}_c = \{(\mathbf{x}_i, \mathbf{y}_i) : b(\mathbf{x}_i) = c\}$

- two simple choices

CART™ : California Statistical Software

  - C = 2 : **binary tree**
  - $g_t(\mathbf{x}) = E_{in}\text{-optimal}$ **constant**
    - binary/multiclass classification (0/1 error): majority of $\{y_i\}$
    - regression (squared error): average of $\{y_i\}$

Branching in C&RT: Purifying

- function DecisionTree (data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}, i = 1 \cdots N$)
- if termination criteria met : return base hypothesis $g_t(\mathbf{x})$
- else : (1) learn branching criteria $b(\mathbf{x})$
- (2) split $\mathcal{D}$ to 2 parts $\mathcal{D}_c = \{(\mathbf{x}_i, \mathbf{y}_i) : b(\mathbf{x}_i) = c\}$

- more simple choices
  - simple internal node for C = 2 : {1,2} - output decision stump
  - 'easier' sub-tree: branch by purifying

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{argmin} \sum_{c=1}^{2} |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

C&RT: bi-branching by purifying

Impurity Functions

- by E_{in} of optimal constant
  - regression error:

$$\text{impurity}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2$$

    - **with** $\bar{y} = \text{average of } \{y_i\}$
  - classification error:

$$\text{impurity}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} ind(y_i \neq y^*)$$

    - **with** $y^* = \text{majority of } \{y_i\}$

Impurity Functions

- for classification
  - classification error:

$$1 - \max_{1 \leq k \leq K} \frac{\sum_{i=1}^{N} ind(y_i = k)}{N}$$

    - optimal k = y* only
  - Gini index:

$$1 - \sum_{k=1}^{K} \left(\frac{\sum_{i=1}^{N} ind(y_i = k)}{N} \right)^2$$

    - all k considered together

Quize

For the Gini index $1 - \sum_{k=1}^{K} \left(\frac{\sum_{i=1}^{N} ind(y_i = k)}{N} \right)^2$. Consider K = 2, and let $\mu = \frac{N_1}{N}$, where N_1 is the number of examples with $y_n = 1$. Which of the following formula of μ equals the Gini index in this case?

(1) $2\mu(1-\mu)$ **(2)** $2\mu^2(1-\mu)$ **(3)** $2\mu(1-\mu)^2$ **(4)** $2\mu^2(1-\mu)^2$

Reference Answer: 1

$$1 - (\mu^2 + (1-\mu)^2) = 1 - \mu^2 - 1 + 2\mu - \mu^2 = 2\mu - 2\mu^2$$

Termination in C&RT

- function DecisionTree (data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}, i = 1 \cdots N$)
- if termination criteria met : return base hypothesis $g_t(\mathbf{x})$
- else : (1) learn branching criteria $b(\mathbf{x})$
- (2) split $\mathcal{D}$ to 2 parts $\mathcal{D}_c = \{(\mathbf{x}_i, \mathbf{y}_i) : b(\mathbf{x}_i) = c\}$

- forced to terminate when
  - all y_i the same: $\text{impurity} = 0 \Rightarrow g_t(\mathbf{x}) = y_i$
  - all x_i the same: no decision stumps

C&RT: fully-grown tree with constant leaves that come from bi-branching by purifying

Basic C&RT Algorithm

- function **DecisionTree** (data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}, i = 1 \cdots N$)
- if cannot branch anymore : return $g_t(\mathbf{x}) = E_{in}\text{-optimal}$ **constant**
- else : (1) learn branching criteria $b(\mathbf{x})$

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{argmin} \sum_{c=1}^{2} |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

(2) split $\mathcal{D}$ to **2** parts $\mathcal{D}_c = \{(\mathbf{x}_i, \mathbf{y}_i) : b(\mathbf{x}_i) = c\}$

(3) build sub-tree $G_c \leftarrow$ **DecisionTree** $(\mathcal{D}_c)$

(4) return $G(\mathbf{x}) = \sum_{c=1}^{2} ind\{b(\mathbf{x}) = c\} \cdot G_c(\mathbf{x})$

easily handle binary classification, regression, & multi-class classification

Regularization by Pruning

- **fully-grown tree:** $E_{in}(G) = 0$ if all $\mathbf{x}_i$ different
- **but overfit (large E_{out}) because low-level trees built with small $\mathcal{D}_c$**
- **need a regularizer, say,** $\Omega(G) = \text{NumberOfLeaves}(G)$
- **want regularized decision tree:**

$$\underset{\text{all possible } G}{argmin} \quad E_{in}(G) + \lambda\Omega(G)$$

  - **called pruned decision tree**
- **cannot enumerate all possible G computationally -- often consider**
  - $G^{(0)} = \text{fully-grown tree}$
  - $G^{(i)} = \underset{G}{argmin} E_{in}(G)$ **such that G is one-leaf removed from** $G^{(i-1)}$

Branching on Categorical Features

- **numerical features**
  - Eg. blood pressure: 130, 98, 115, 147, 120
- **branching for numerical features**
  - decision stump $b(\mathbf{x}) = ind(\mathbf{x}_i \leq \theta) + 1$ with $\theta \in \mathbb{R}$
- **categorical features**
  - major symptom: fever, pain, tired, sweaty
- **branching for categorical features**
  - decision subset $b(\mathbf{x}) = ind(\mathbf{x}_i \in S) + 1$ with $S \subset \{1, 2, \ldots, K\}$
- **decision trees usually handle categorical features easily**

Missing Features by Surrogate Branch

$$b(\mathbf{x}) = ind(\text{weight} \leq 50kg)$$

- **if weight missing during prediction:**
  - **what would human do?**
    - **go get weight**
    - **or, use threshold on height instead**
      - ➔ **because threshold on height ≈ threshold on weight**
  - **surrogate branch:**
    - **maintain surrogate branch during training**

$$b_1(\mathbf{x}), b_2(\mathbf{x}), \ldots \approx \text{best branch } b(\mathbf{x})$$

    - **allow missing feature for b(x) during prediction by using surrogate instead**

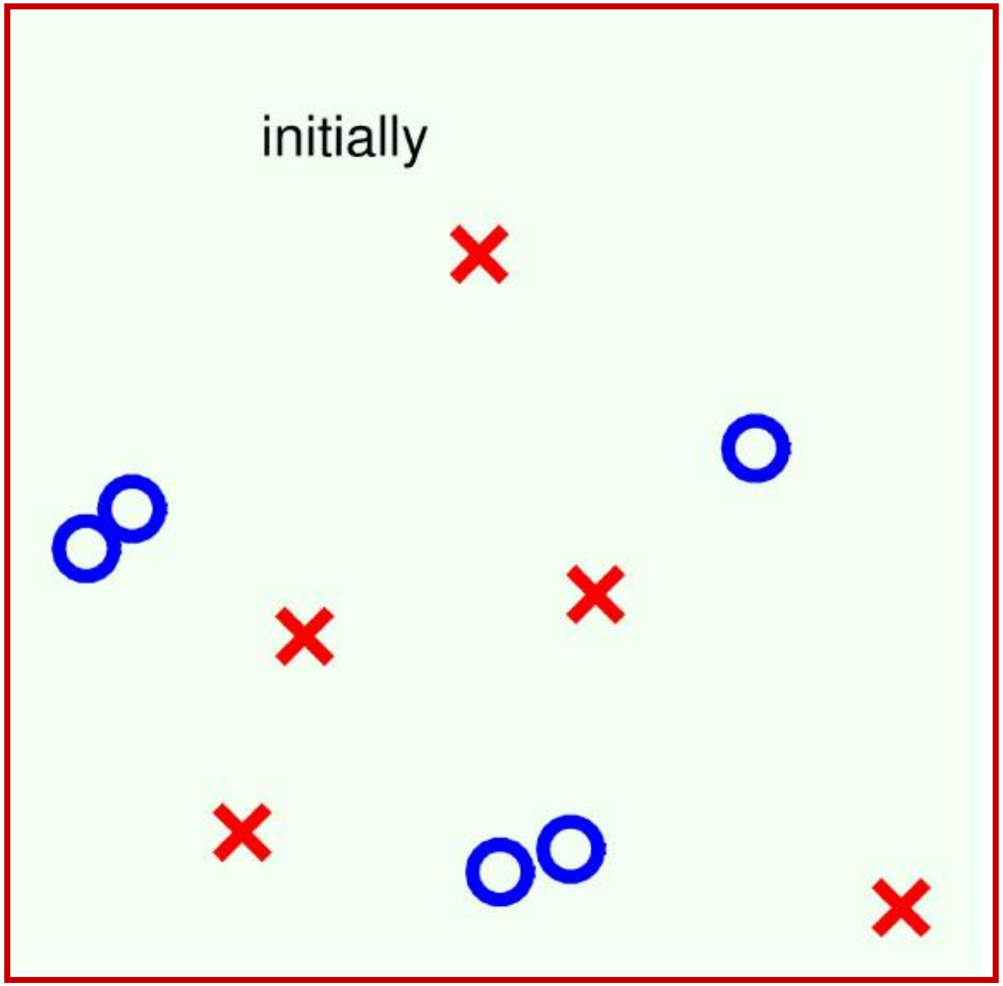
Quize

For a categorical branching criteria $b(\mathbf{x}) = [\![x_i \in S]\!] + 1$ with $S = \{1, 6\}$. Which of the following is the explanation of the criteria?

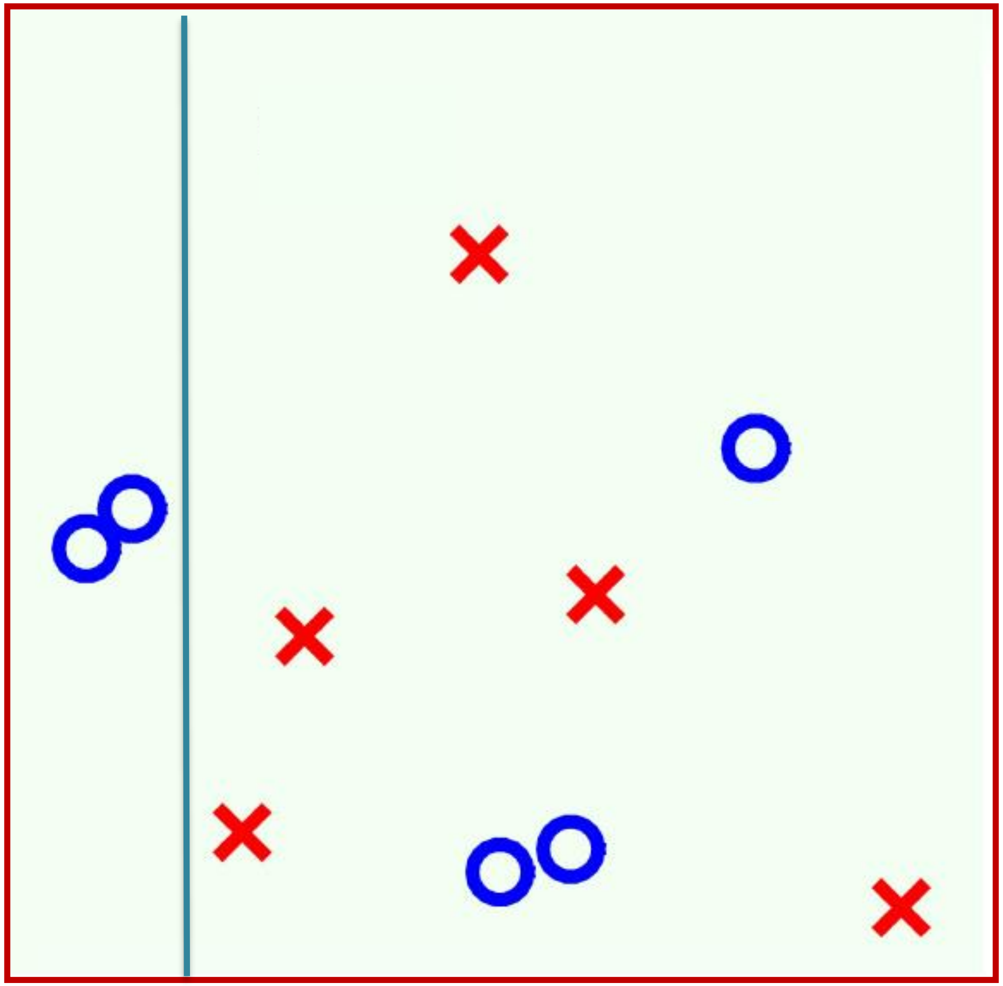
1. if i-th feature is of type 1 or type 6, branch to first sub-tree; else branch to second sub-tree
2. if i-th feature is of type 1 or type 6, branch to second sub-tree; else branch to first sub-tree
3. if i-th feature is of type 1 and type 6, branch to second sub-tree; else branch to first sub-tree
4. if i-th feature is of type 1 and type 6, branch to first sub-tree; else branch to second sub-tree

Reference Answer: 2

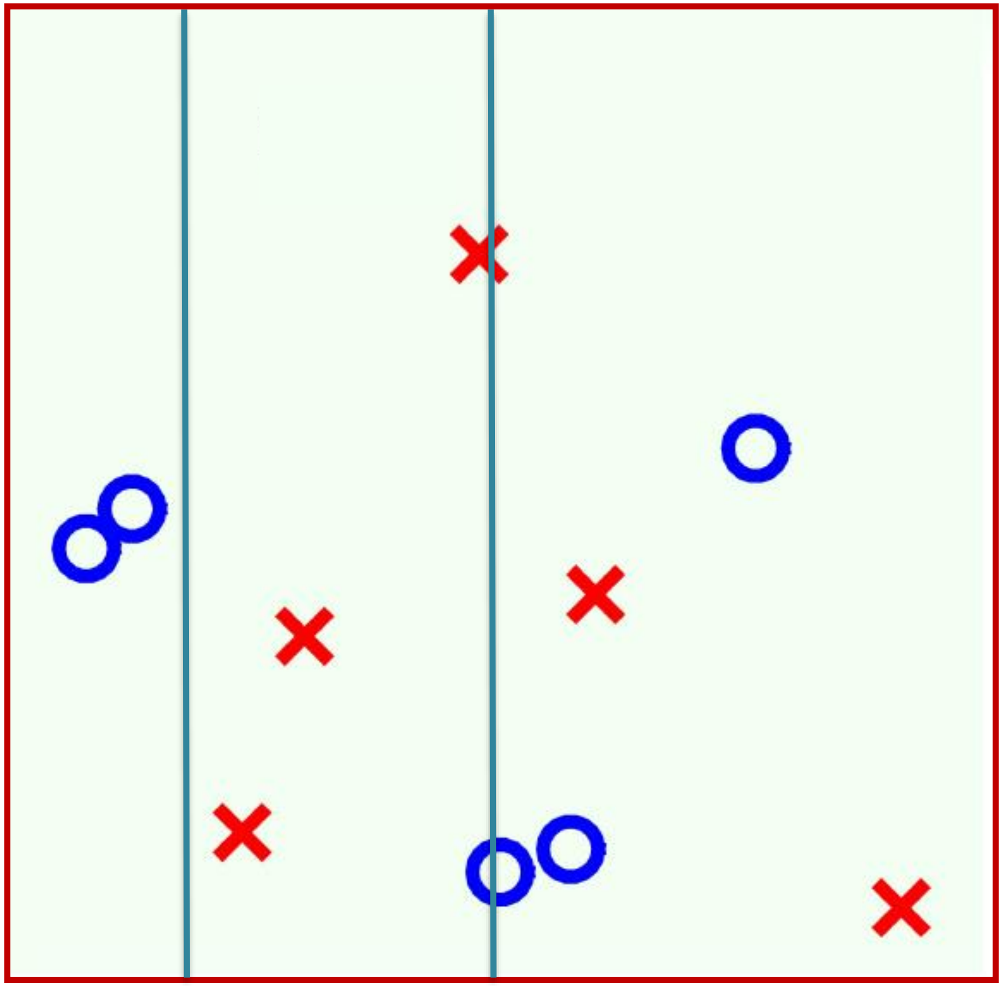
A Simple Data Set



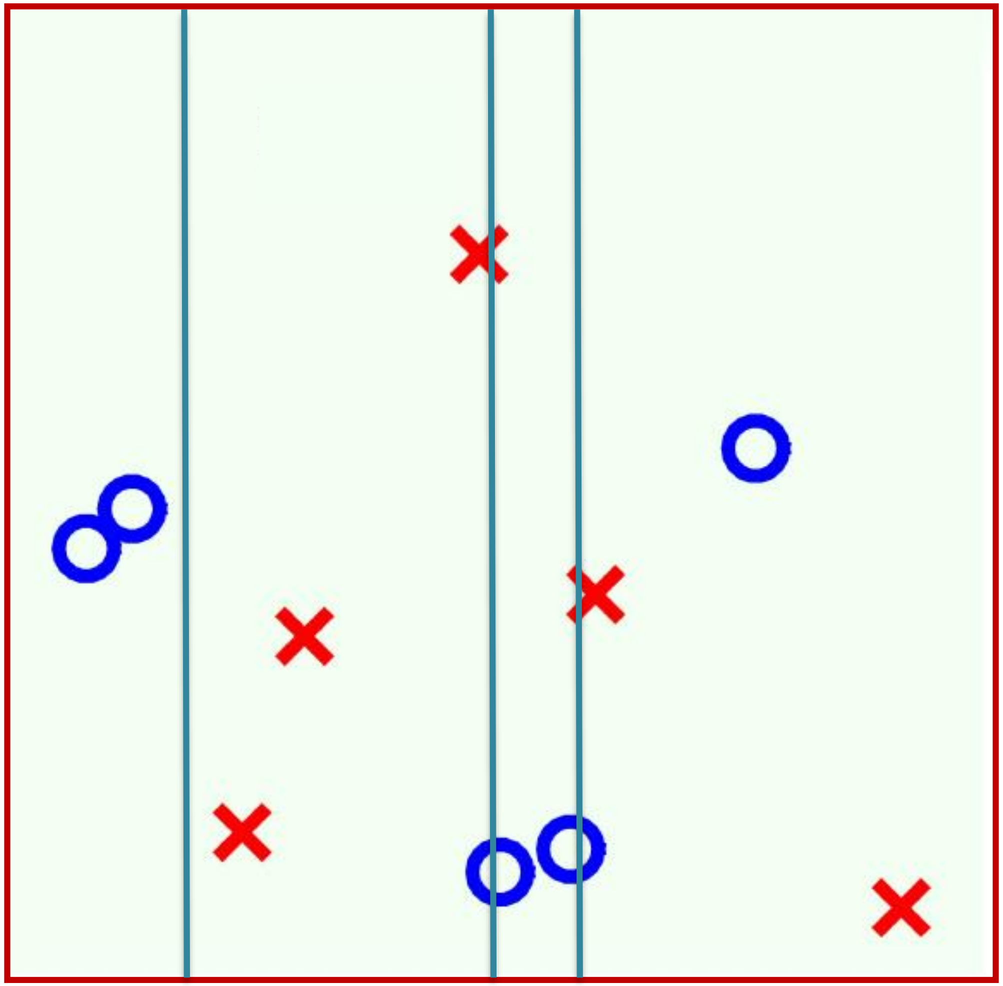
A Simple Data Set



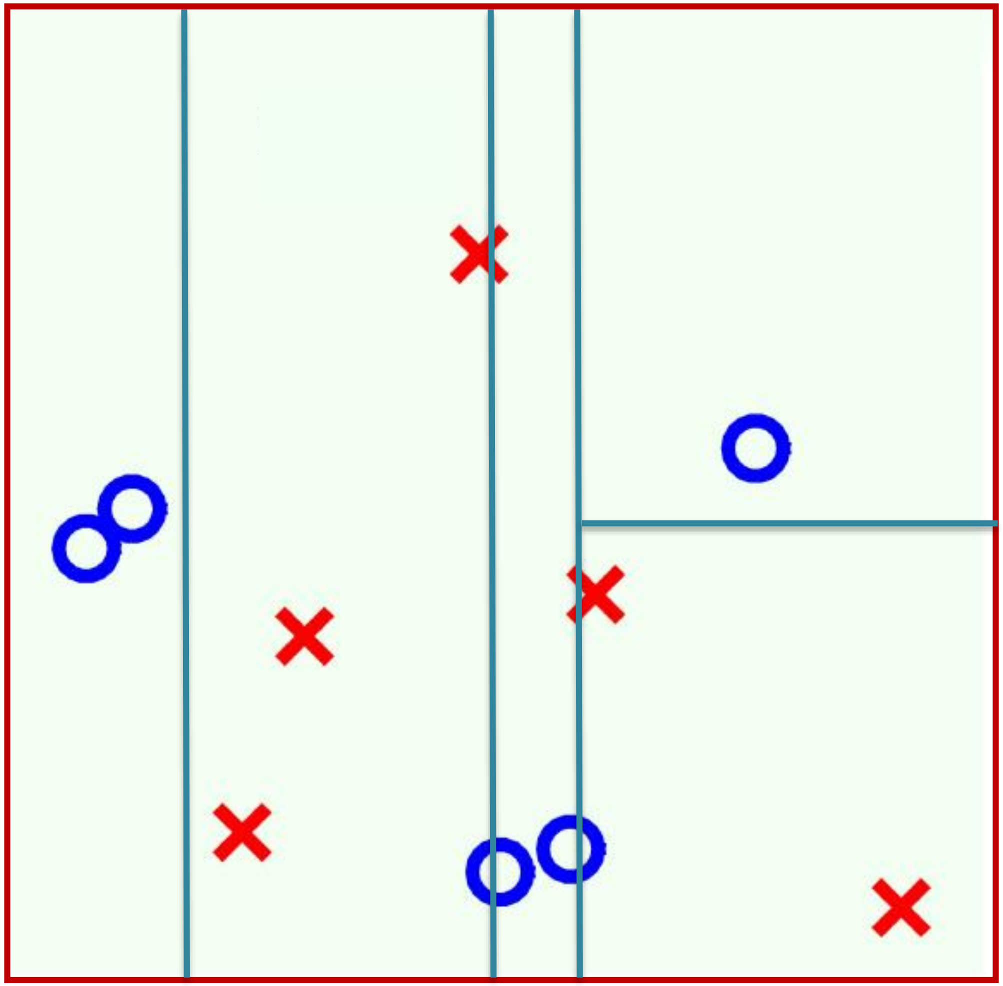
A Simple Data Set



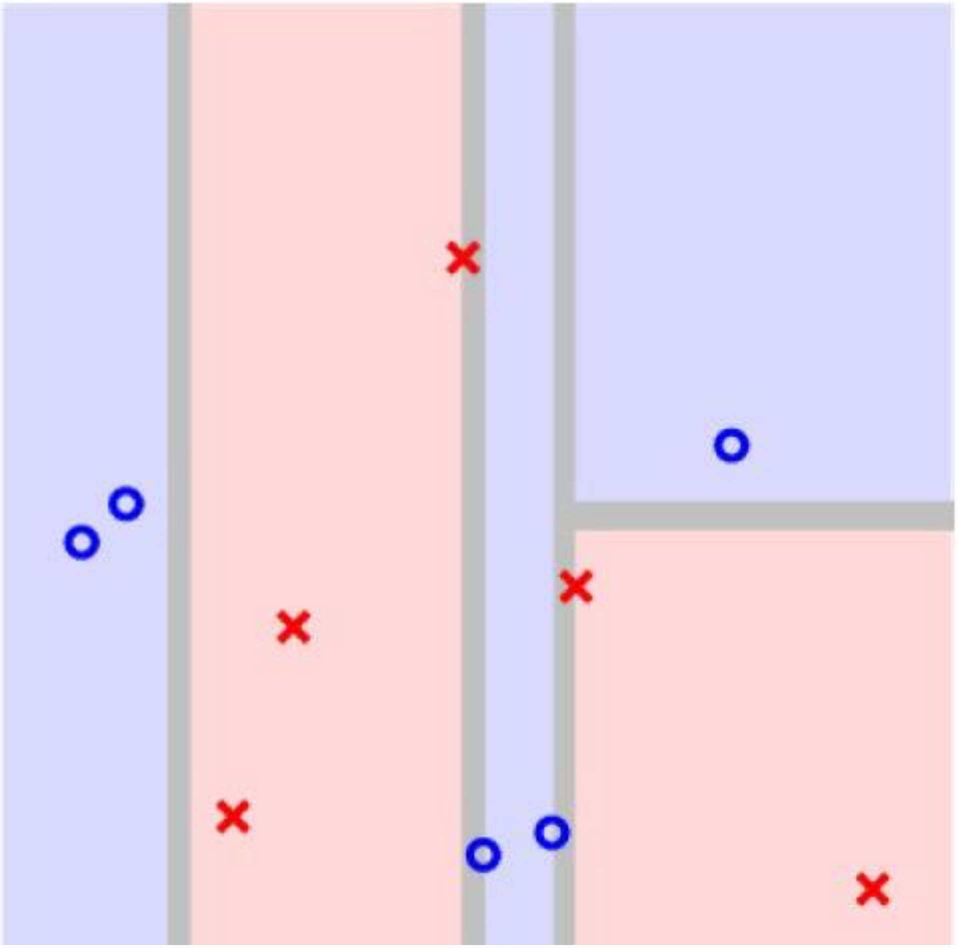
A Simple Data Set



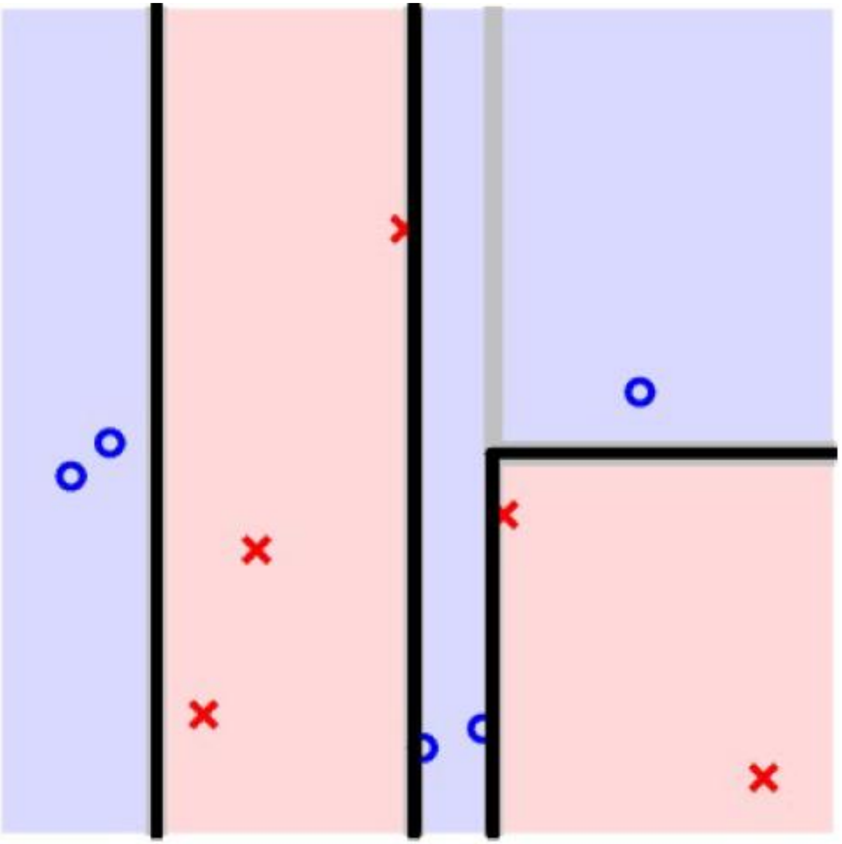
A Simple Data Set



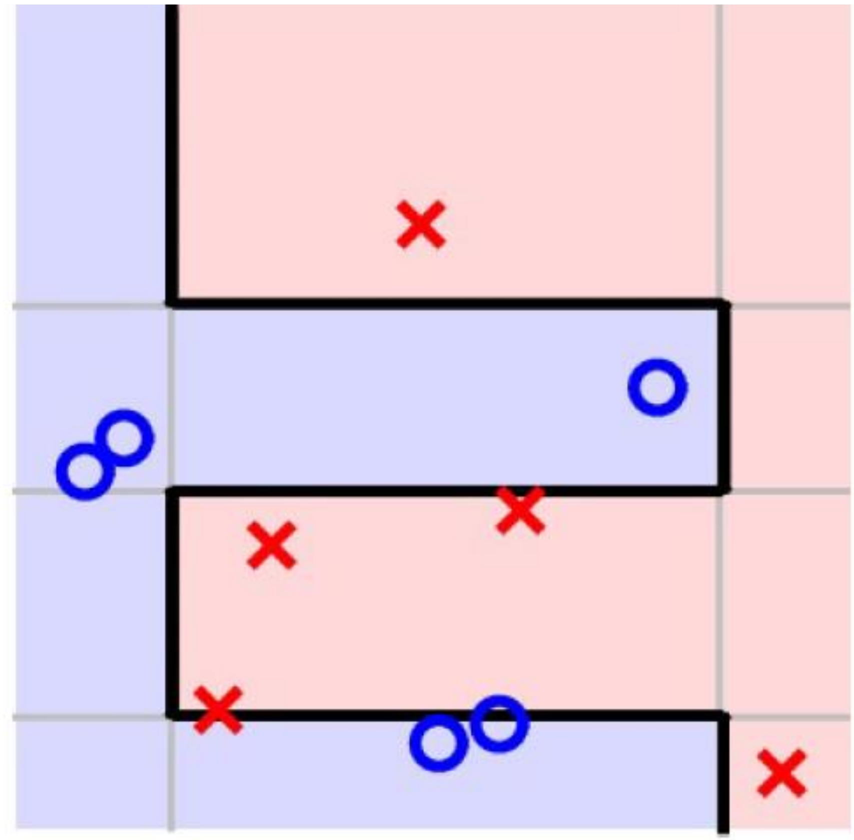
A Simple Data Set



A Simple Data Set



C&RT



AdaBoost-Stump

Practical Specialties of C&RT

- human-explainable
- multiclass easily
- categorical features easily
- missing features easily
- efficient non-linear training (and testing)

almost no other learning model share all such specialties, except for other decision trees

another popular decision tree algorithm: C4.5, with different choices of heuristics

电子科技大学

UeSTC

1956