Machine Learning
(機器學習)

Lecture 13: Decision Tree Ensembles

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. When Can Machines Learn?
2. Why Can Machines Learn?
3. How Can Machines Learn?
4. How Can Machines Learn Better?
5. Embedding Numerous Features: Kernel Models
6. Combining Predictive Features: Aggregation Models

Lecture 13: Decision Tree Ensembles

- Decision Tree Hypothesis
- Decision Tree Algorithm
- Decision Tree in Action
- Random Forest Algorithm
- Out-Of-Bag Estimate
- Random Forest in Action
- Optimization View of AdaBoost
- Gradient Boosting
- Summary of Aggregation Models

What We Have Done

blending: aggregate after getting g_t;
learning: aggregate as well as getting g_t

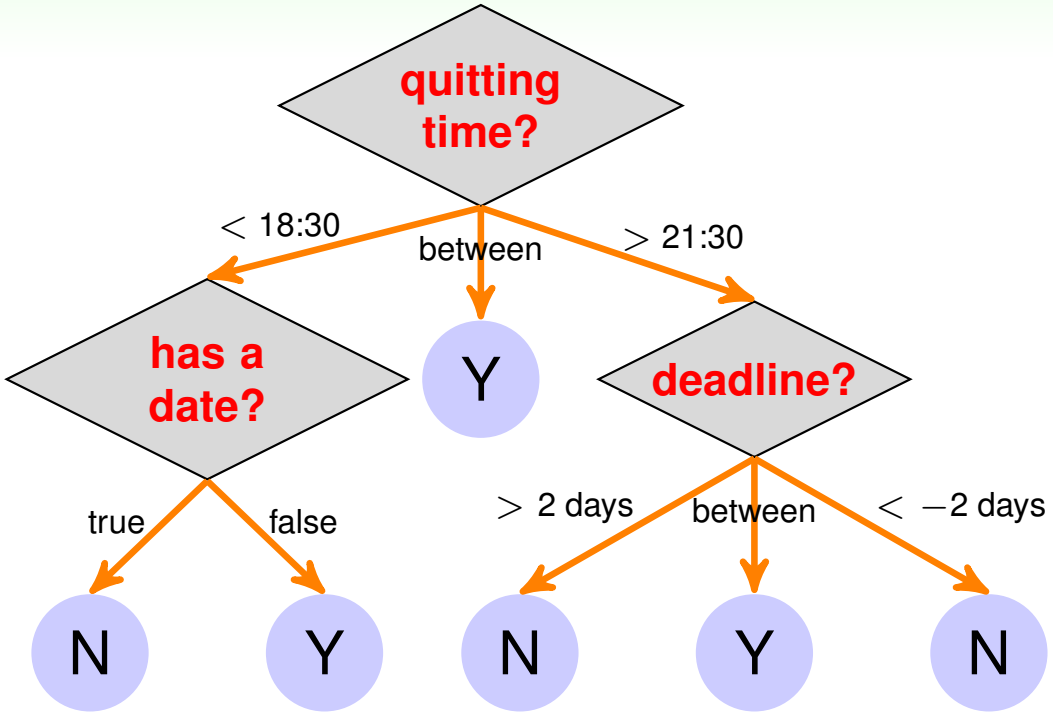
aggregation type	blending	learning
uniform	voting/averaging	Bagging
non-uniform	linear	AdaBoost
conditional	stacking	**Decision Tree**

decision tree: a traditional learning model that realizes **conditional aggregation**

Decision Tree for Watching MOOC Lectures

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

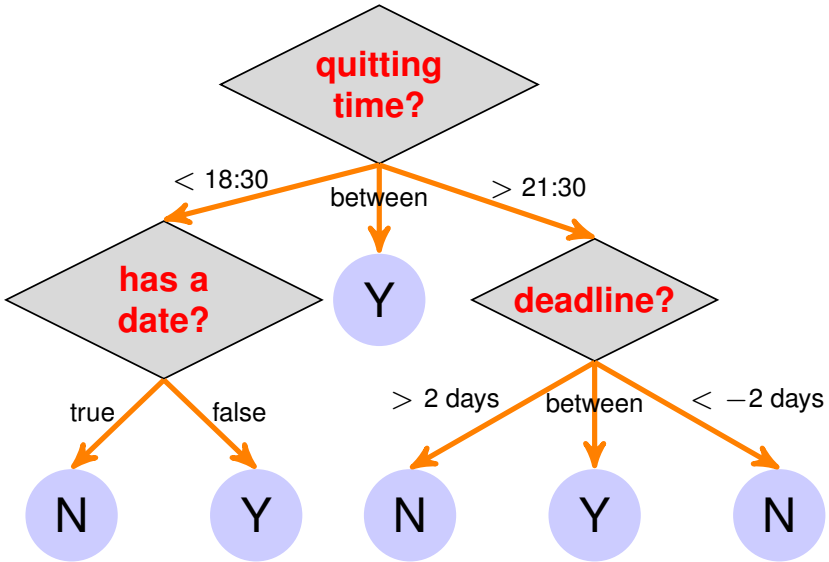
- **base hypothesis** $g_t(\mathbf{x})$: leaf at end of path t, a **constant** here
- **condition** $q_t(\mathbf{x})$: ⟦is $\mathbf{x}$ on path t?⟧
- usually with **simple internal nodes**



decision tree: arguably one of the most **human-mimicking models**

Recursive View of Decision Tree

Path View: $G(\mathbf{x}) = \sum_{t=1}^{T} [\![\mathbf{x} \text{ on path } t]\!] \cdot \text{leaf}_t(\mathbf{x})$



Recursive View

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

- $G(\mathbf{x})$: full-tree hypothesis
- $b(\mathbf{x})$: branching criteria
- $G_c(\mathbf{x})$: sub-tree hypothesis at the c-th branch

tree = (root, sub-trees), just like what **your data structure instructor would say :-)**

Disclaimers about Decision Tree

Usefulness

- human-explainable: **widely used** in business/medical data analysis
- simple: **even freshmen can implement one :-)**
- efficient in prediction and **training**

However.....

- heuristic: mostly **little theoretical** explanations
- heuristic**s**: 'heuristic**s** selection' confusing to beginners
- arguably no single **representative algorithm**

decision tree: mostly **heuristic** **but useful** on its own

Questions?

A Basic Decision Tree Algorithm

$$G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$$

function DecisionTree(data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)

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 $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$
3. build sub-tree $G_c \leftarrow$ DecisionTree($\mathcal{D}_c$)
4. return $G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

four choices: number of branches, branching criteria, termination criteria, & base hypothesis

Classification and Regression Tree (C&RT)

function DecisionTree(data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)
if termination criteria met
    return base hypothesis $g_t(\mathbf{x})$
else ...
    2 split $\mathcal{D}$ to C parts $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$

two simple choices

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

disclaimer:
C&RT here is based on **selected components** of **CART^{TM} of California Statistical Software**

Branching in C&RT: Purifying

function DecisionTree(data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)
if termination criteria met
    return base hypothesis $g_t(\mathbf{x})$ = E_{in}-optimal constant
else ...
1. learn branching criteria $b(\mathbf{x})$
2. split $\mathcal{D}$ to 2 parts $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$

more simple choices

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

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

C&RT: **bi-branching** by **purifying**

Impurity Functions

by E_{in} of optimal constant

- regression error:

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

with $\bar{y}$ = average of $\{y_n\}$

- classification error:

$$\text{impurity}(\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}[\![y_n \neq y^*]\!]$$

with y^* = majority of $\{y_n\}$

for classification

- Gini index:

$$1 - \sum_{k=1}^{K}\left(\frac{\sum_{n=1}^{N}[\![y_n = k]\!]}{N}\right)^2$$

—all k considered together

- classification error:

$$1 - \max_{1\le k\le K}\frac{\sum_{n=1}^{N}[\![y_n = k]\!]}{N}$$

—optimal $k = y^*$ only

popular choices: **Gini** for classification, **regression error** for regression

Termination in C&RT

function DecisionTree(data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)
if termination criteria met
    return base hypothesis $g_t(\mathbf{x})$ = E_{in}-optimal constant
else ...
1. learn branching criteria

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

'forced' to terminate when

- all y_n the same: impurity = 0 $\Longrightarrow$ $g_t(\mathbf{x}) = y_n$
- all $\mathbf{x}_n$ 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 $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)

if cannot branch anymore

return $g_t(\mathbf{x}) = E_{\text{in}}$-optimal constant

else

1. learn branching criteria

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

2. split $\mathcal{D}$ to 2 parts $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$
3. build sub-tree $G_c \leftarrow \text{DecisionTree}(\mathcal{D}_c)$
4. return $G(\mathbf{x}) = \sum_{c=1}^{2} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

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

Regularization by Pruning

fully-grown tree: $E_{\text{in}}(G) = 0$ if all $\mathbf{x}_n$ 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:

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

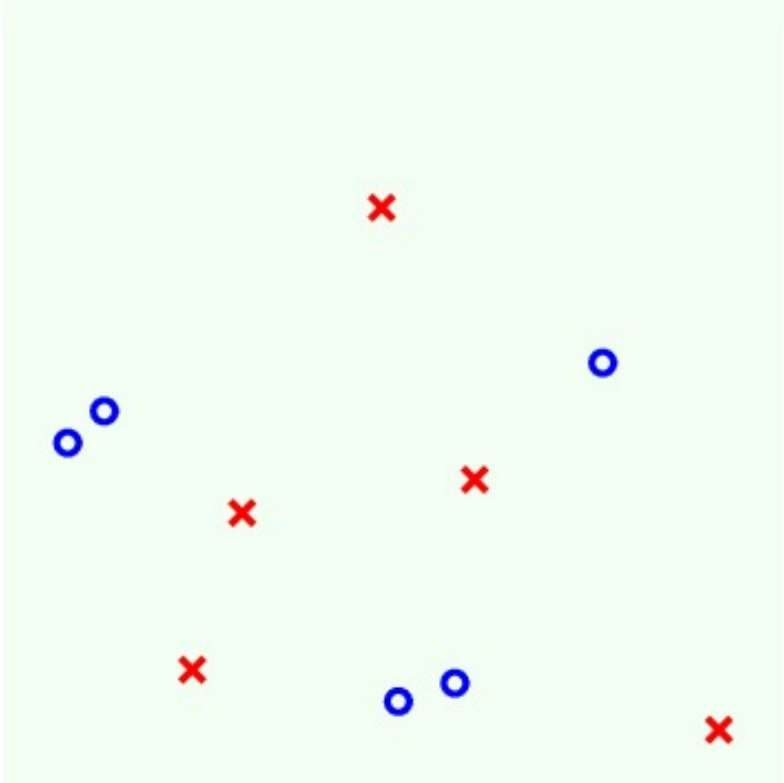
—called **pruned** decision tree

- heuristic mature & prevalent in many decision tree algorithms (details omitted)

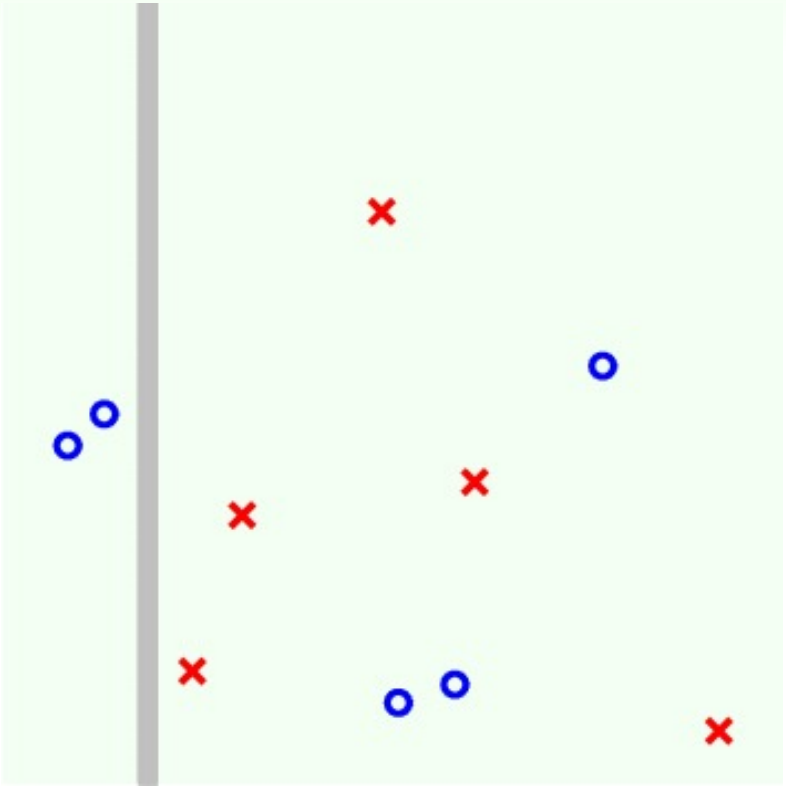
systematic choice of λ? **validation**

Questions?

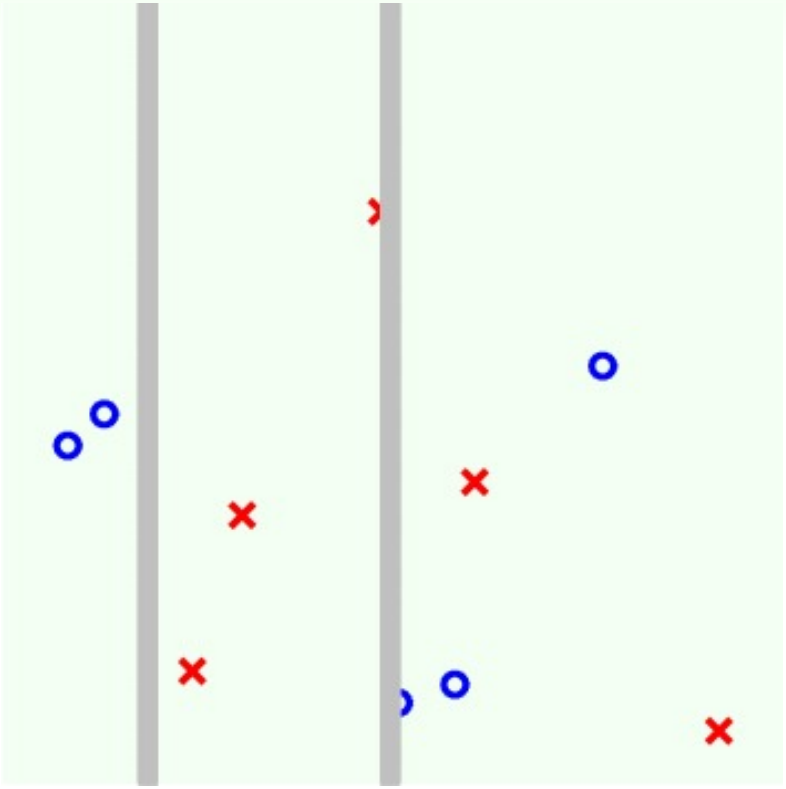
A Simple Data Set



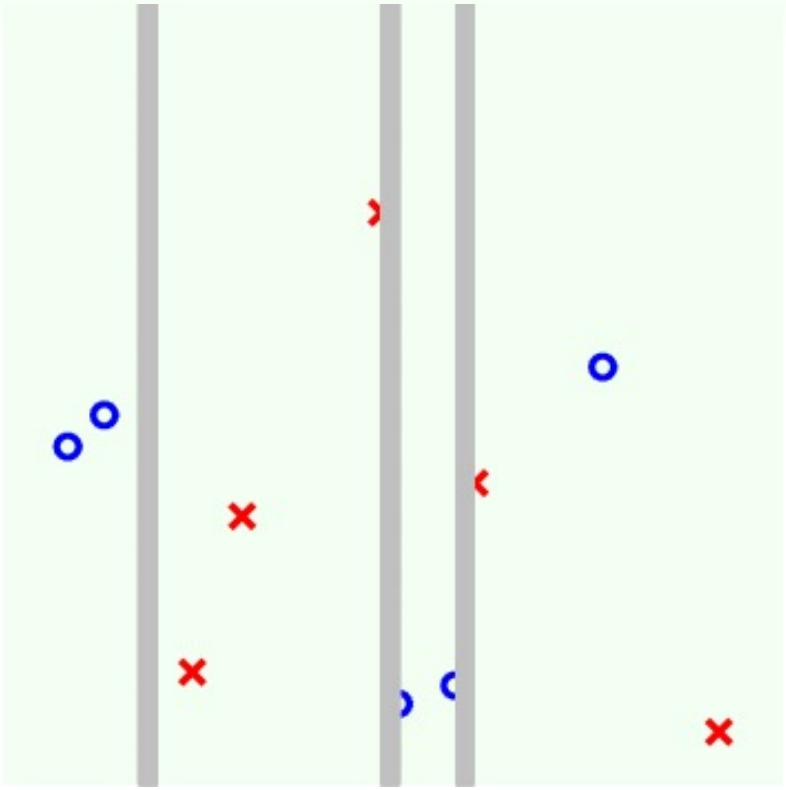
A Simple Data Set



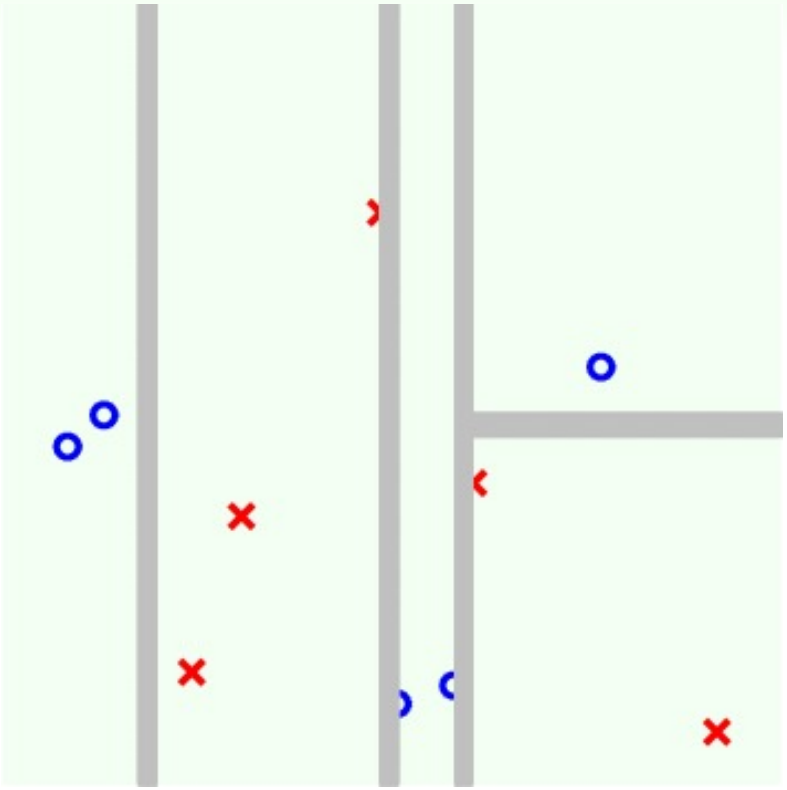
A Simple Data Set



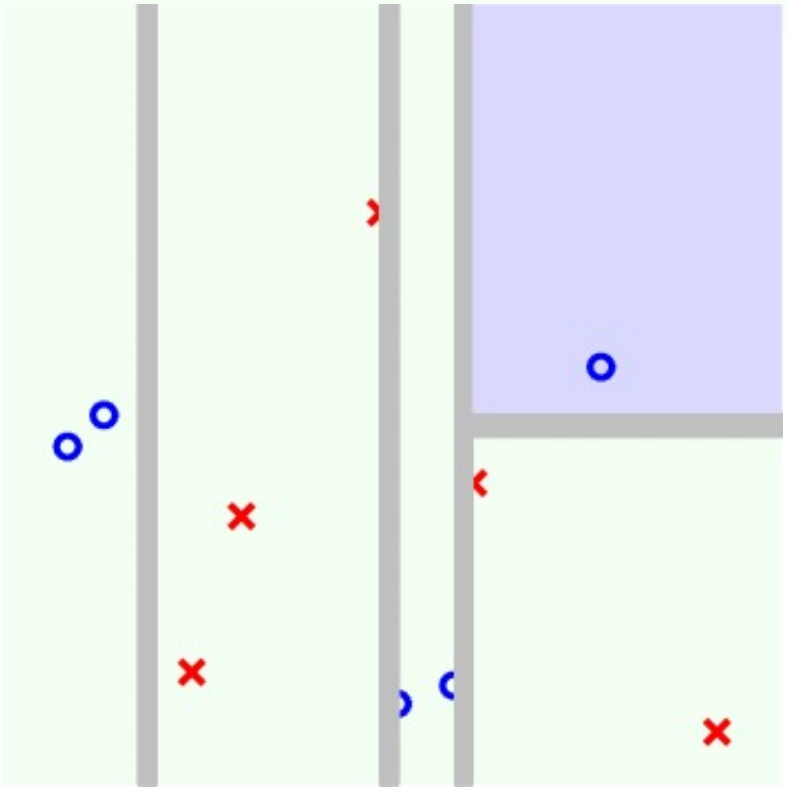
A Simple Data Set



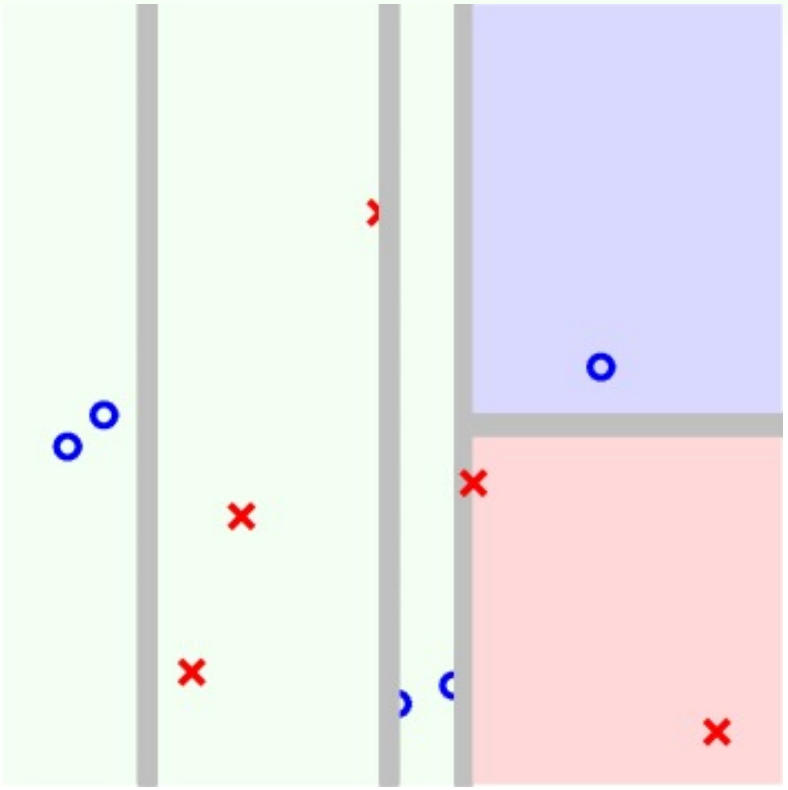
A Simple Data Set



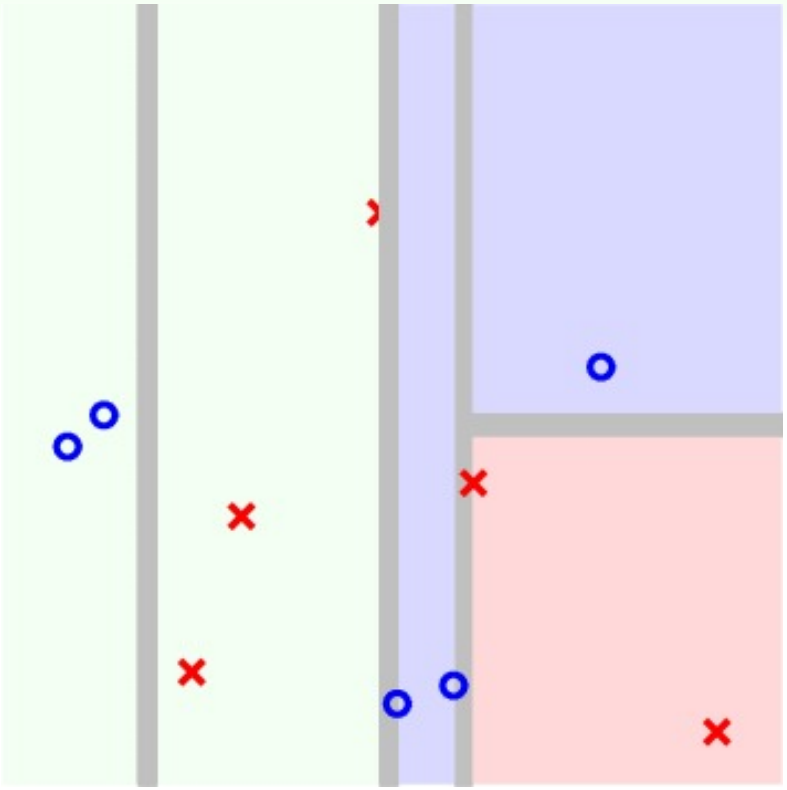
A Simple Data Set



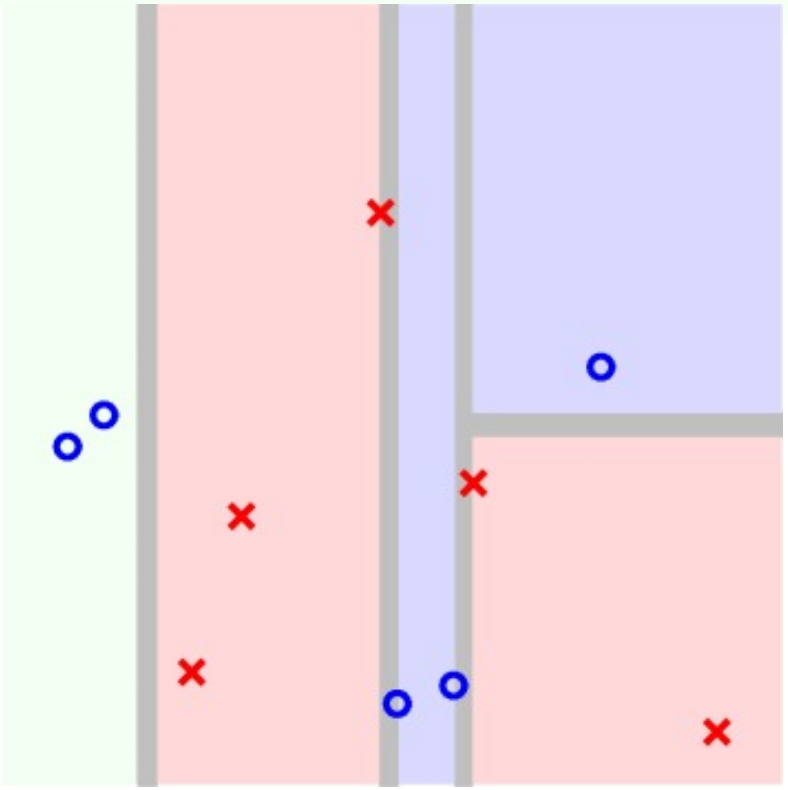
A Simple Data Set



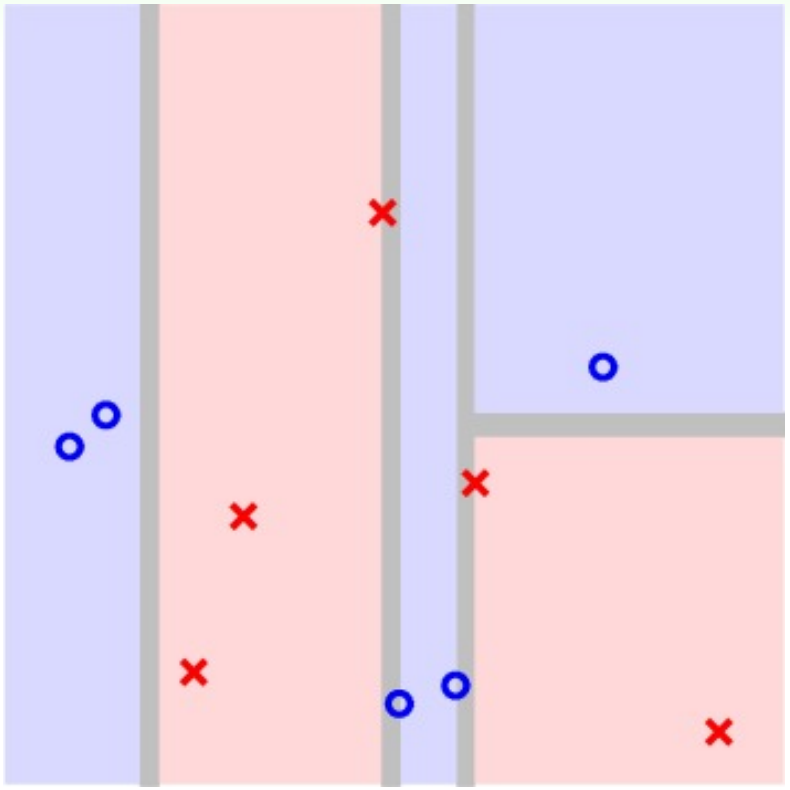
A Simple Data Set



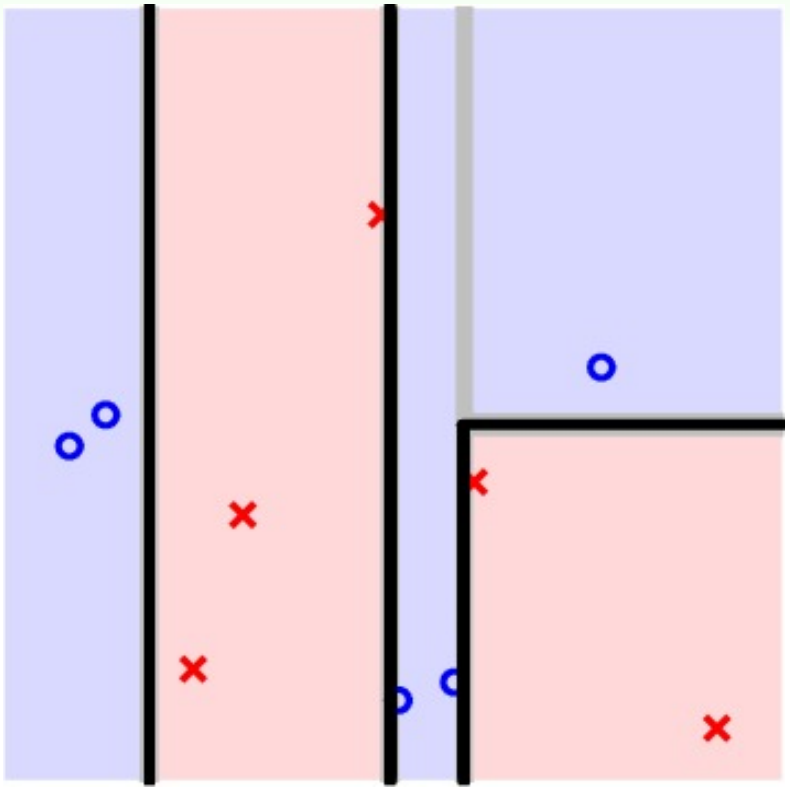
A Simple Data Set



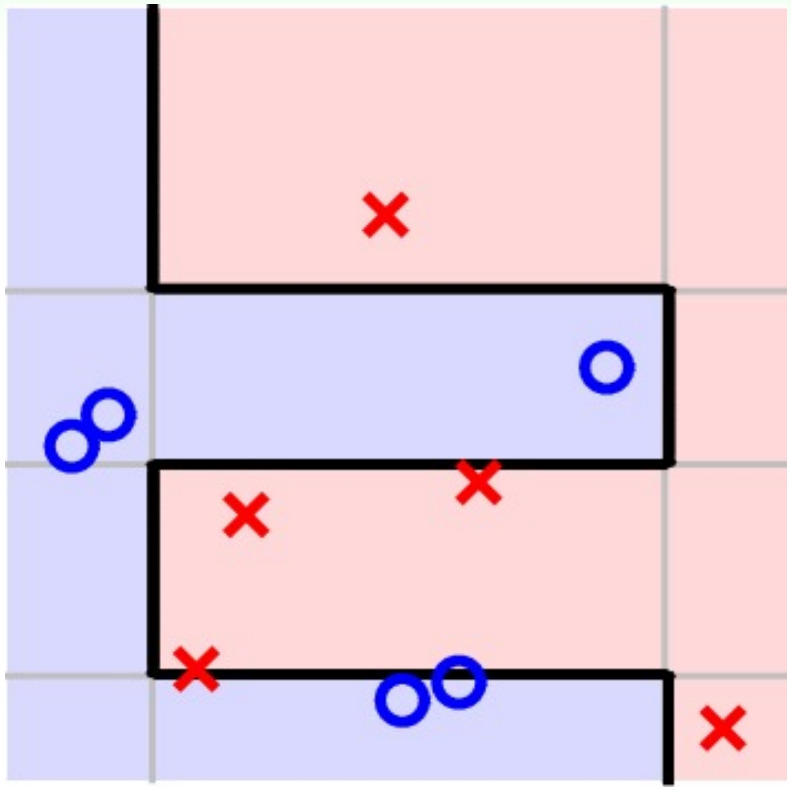
A Simple Data Set



A Simple Data Set



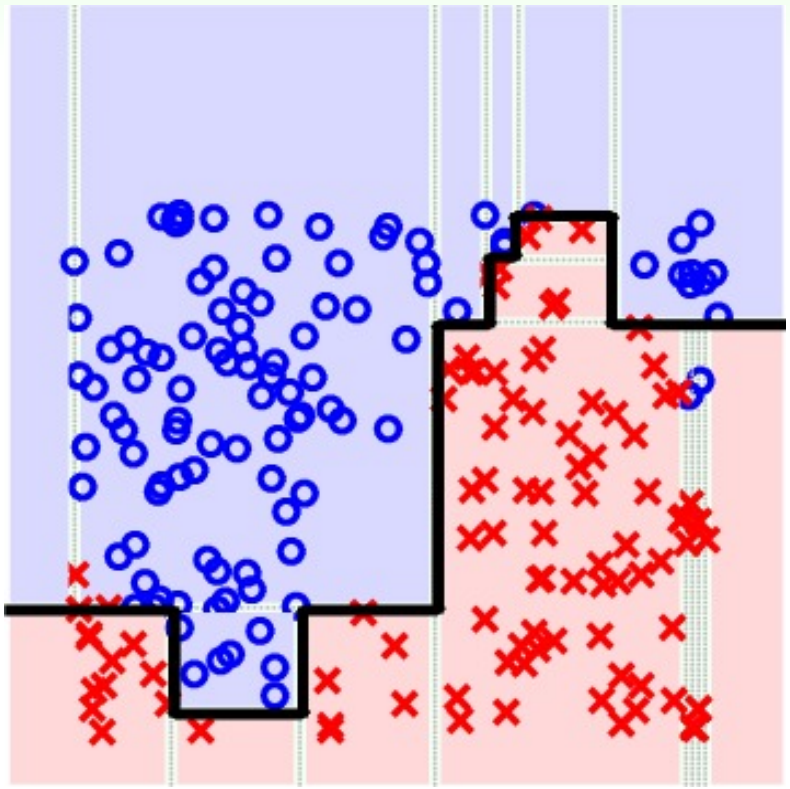
C&RT



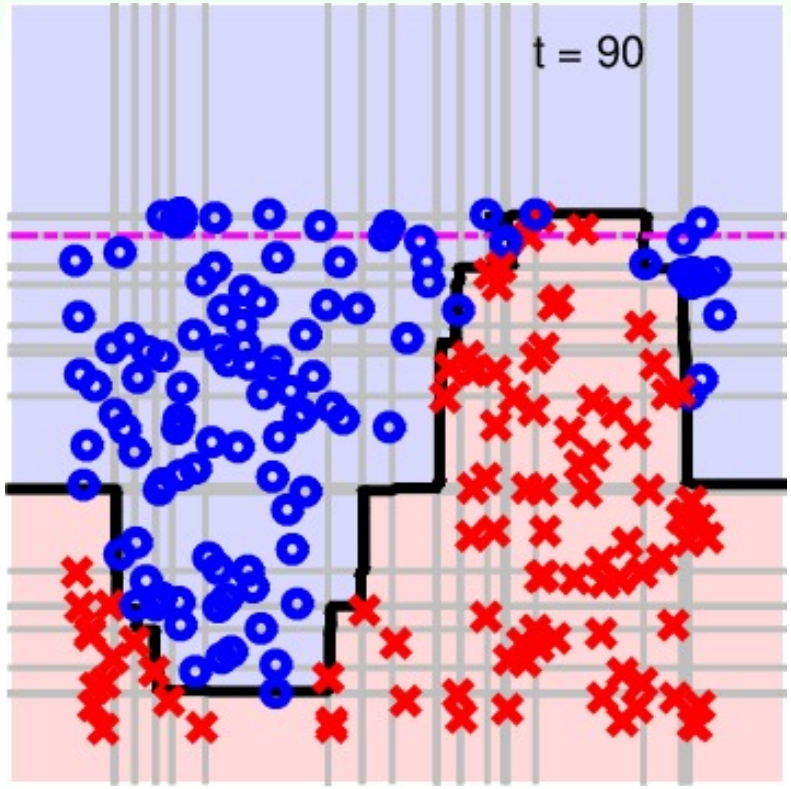
AdaBoost-Stump

C&RT: 'divide-and-conquer'

A Complicated Data Set



C&RT



AdaBoost-Stump

C&RT: even more efficient than AdaBoost-Stump

Practical Specialties of C&RT

- human-explainable
- multiclass easily
- categorical features easily (details omitted)
- missing features easily (details omitted)
- 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**

Questions?

Recall: Bagging and Decision Tree

Bagging

function Bag($\mathcal{D}, \mathcal{A}$)
For $t = 1, 2, \ldots, T$

1. request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$
2. obtain base g_t by $\mathcal{A}(\tilde{\mathcal{D}}_t)$

return $G = \text{Uniform}(\{g_t\})$

—reduces variance
by voting/averaging

Decision Tree

function DTree($\mathcal{D}$)
if termination return base g_t
else

1. learn $b(\mathbf{x})$ and split $\mathcal{D}$ to $\mathcal{D}_c$ by $b(\mathbf{x})$
2. build $G_c \leftarrow \text{DTree}(\mathcal{D}_c)$
3. return $G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

—large variance
especially if fully-grown

putting them together?
(i.e. aggregation of aggregation :-))

Random Forest (RF)

random forest (RF) = bagging + fully-grown C&RT decision tree

function RandomForest($\mathcal{D}$)
For $t = 1, 2, \ldots, T$

1. request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$
2. obtain tree g_t by DTree($\tilde{\mathcal{D}}_t$)

return $G = \text{Uniform}(\{g_t\})$

function DTree($\mathcal{D}$)
if termination return base g_t
else

1. learn $b(\mathbf{x})$ and split $\mathcal{D}$ to $\mathcal{D}_c$ by $b(\mathbf{x})$
2. build $G_c \leftarrow \text{DTree}(\mathcal{D}_c)$
3. return $G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

- highly **parallel**/**efficient** to learn
- **inherit pros** of C&RT
- **eliminate cons** of fully-grown tree

Diversifying by Feature Expansion

randomly **sample** d' **features** from $\mathbf{x}$: $\Phi(\mathbf{x}) = \mathrm{P} \cdot \mathbf{x}$
with row i of P sampled randomly

- **projection** (combination) with random row $\mathbf{p}_i$ of P: $\phi_i(\mathbf{x}) = \mathbf{p}_i^T \mathbf{x}$
- often consider **low-dimensional** projection:
  only d'' **non-zero** components in $\mathbf{p}_i$
- original RF consider d' random **low-dimensional projections for each** $b(\mathbf{x})$ in C&RT

RF = bagging + random-**combination** C&RT
—randomness everywhere!

Questions?

Bagging Revisited

Bagging

function Bag($\mathcal{D}, \mathcal{A}$)
For $t = 1, 2, \ldots, T$

1. request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$
2. obtain base g_t by $\mathcal{A}(\tilde{\mathcal{D}}_t)$

return $G = \text{Uniform}(\{g_t\})$

	g_1	g_2	g_3	$\cdots$	g_T
$(\mathbf{x}_1, y_1)$	$\tilde{\mathcal{D}}_1$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_2, y_2)$	$\star$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_3, y_3)$	$\star$	$\tilde{\mathcal{D}}_2$	$\star$		$\tilde{\mathcal{D}}_T$
$\cdots$					
$(\mathbf{x}_N, y_N)$	$\tilde{\mathcal{D}}_1$	$\tilde{\mathcal{D}}_2$	$\star$		$\star$

$\star$ in t-th column: not used for obtaining g_t
—called **out-of-bag (OOB) examples** of g_t

Number of OOB Examples

OOB (in $\star$) $\Longleftrightarrow$ not sampled after N' drawings

if $N' = N$

- probability for $(\mathbf{x}_n, y_n)$ to be OOB for g_t: $\left(1 - \frac{1}{N}\right)^N$
- if N large:

$$\left(1 - \frac{1}{N}\right)^N = \frac{1}{\left(\frac{N}{N-1}\right)^N} = \frac{1}{\left(1 + \frac{1}{N-1}\right)^N} \approx \frac{1}{e}$$

OOB size per $g_t \approx \frac{1}{e}N$

OOB versus Validation

OOB

	g_1	g_2	g_3	$\cdots$	g_T
$(\mathbf{x}_1, y_1)$	$\tilde{\mathcal{D}}_1$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_2, y_2)$	$\star$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_3, y_3)$	$\star$	$\tilde{\mathcal{D}}_2$	$\star$		$\tilde{\mathcal{D}}_T$
$\cdots$					
$(\mathbf{x}_N, y_N)$	$\tilde{\mathcal{D}}_1$	$\star$	$\star$		$\star$

Validation

g_1^-	g_2^-	$\cdots$	g_M^-
$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{val}}$	$\mathcal{D}_{\text{val}}$		$\mathcal{D}_{\text{val}}$
$\mathcal{D}_{\text{val}}$	$\mathcal{D}_{\text{val}}$		$\mathcal{D}_{\text{val}}$
$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{train}}$

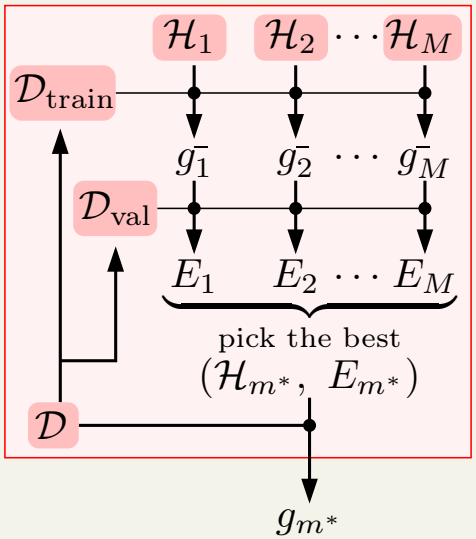
- $\star$ like $\mathcal{D}_{\text{val}}$: 'enough' random examples unused during training
- use $\star$ to validate g_t? easy, but **rarely needed**
- use $\star$ to validate G? $E_{\text{oob}}(G) = \frac{1}{N}\sum_{n=1}^{N} \text{err}(y_n, G_n^-(\mathbf{x}_n))$, with G_n^- contains only trees that $\mathbf{x}_n$ is OOB of,
  such as $G_N^-(\mathbf{x}) = \text{average}(g_2, g_3, g_T)$

E_{oob}: self-validation of bagging/RF

Model Selection by OOB Error

Previously: by Best E_{val}

$$g_{m^*} = \mathcal{A}_{m^*}(\mathcal{D})$$
$$m^* = \underset{1 \le m \le M}{\text{argmin}}\ E_m$$
$$E_m = E_{\text{val}}(\mathcal{A}_m(\mathcal{D}_{\text{train}}))$$



RF: by Best E_{oob}

$$G_{m^*} = \text{RF}_{m^*}(\mathcal{D})$$
$$m^* = \underset{1 \le m \le M}{\text{argmin}}\ E_m$$
$$E_m = E_{\text{oob}}(\text{RF}_m(\mathcal{D}))$$

- use E_{oob} for self-validation —of RF **parameters** such as d''
- no re-training needed

E_{oob} often **accurate** in practice

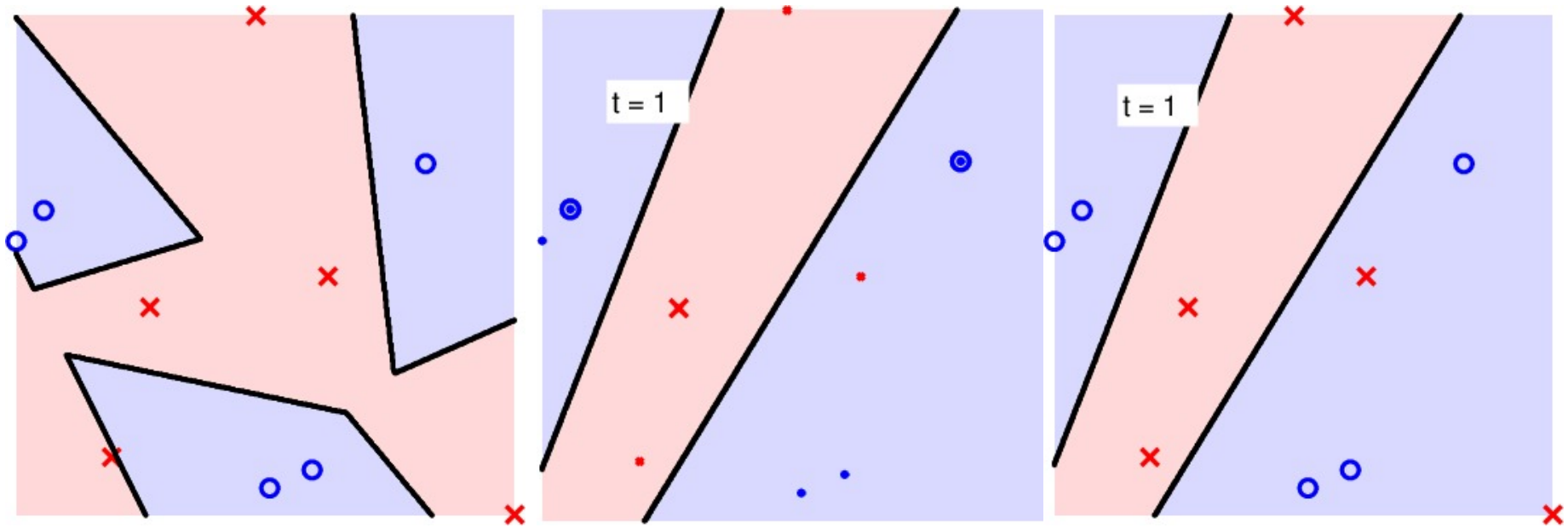
Questions?

A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

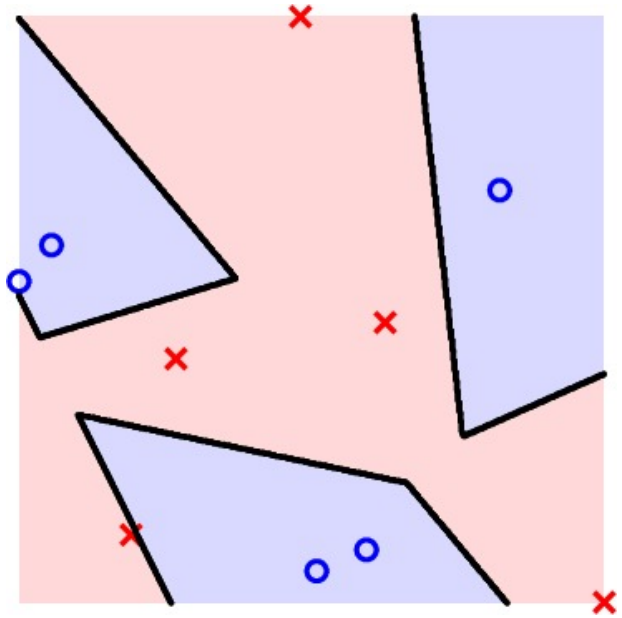
g_t $(N' = N/2)$

G with first t trees

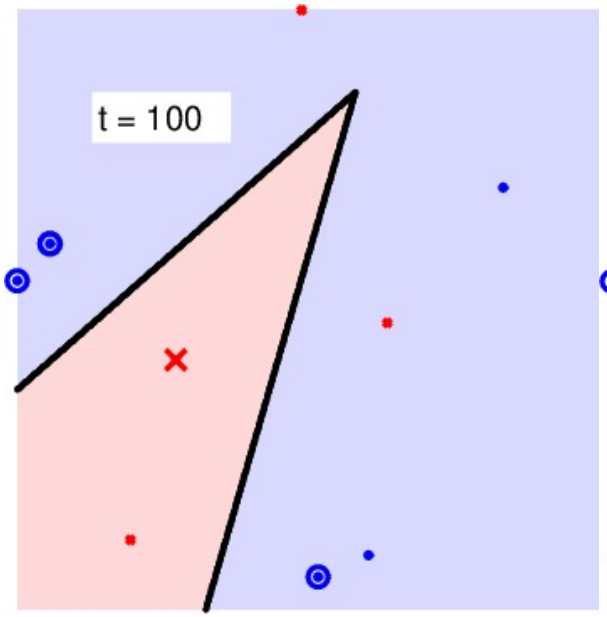


A Simple Data Set

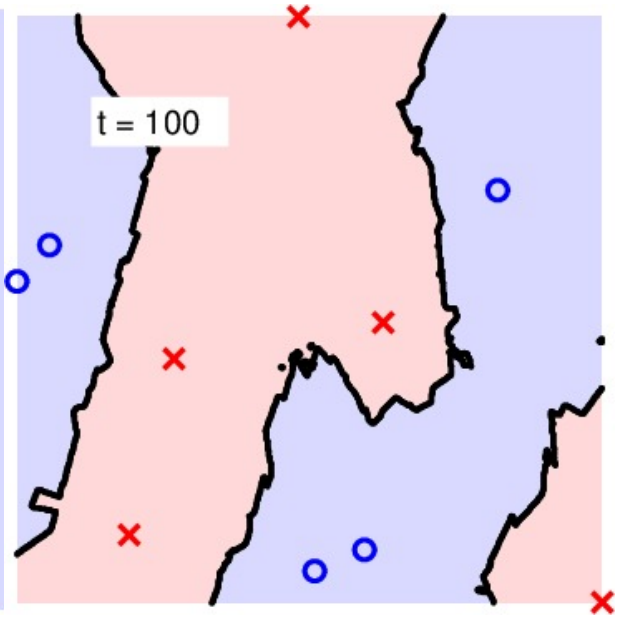
$g_{\text{C\&RT}}$ with random combination



g_t $(N' = N/2)$



G with first t trees

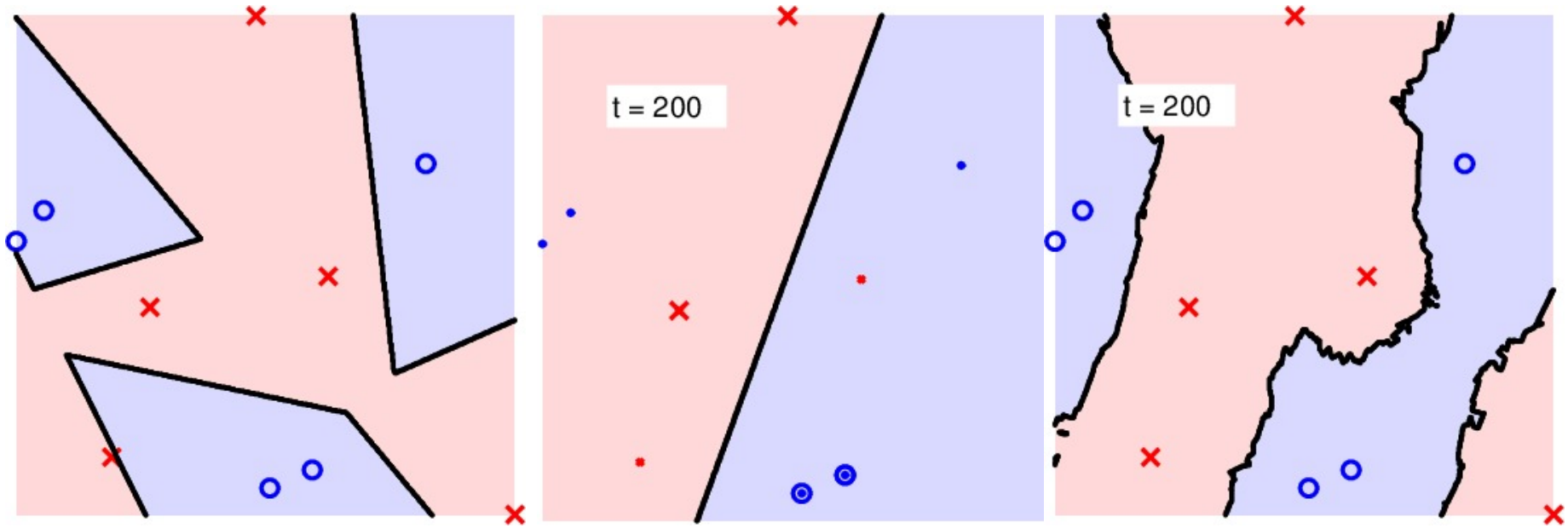


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

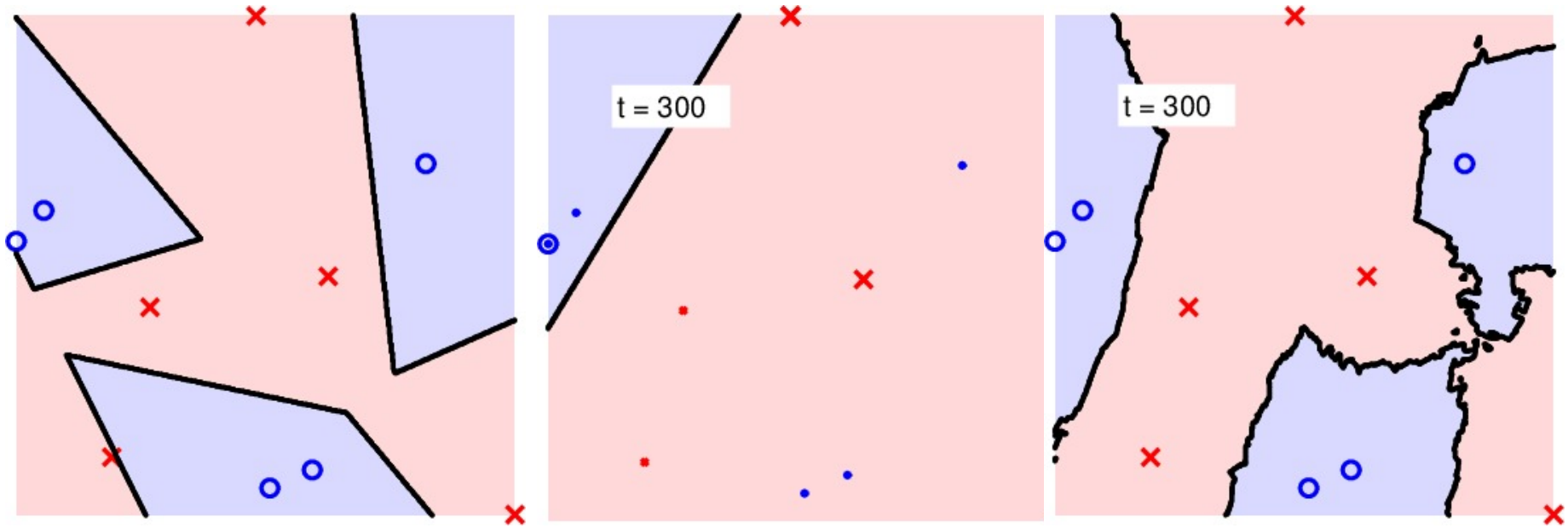


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

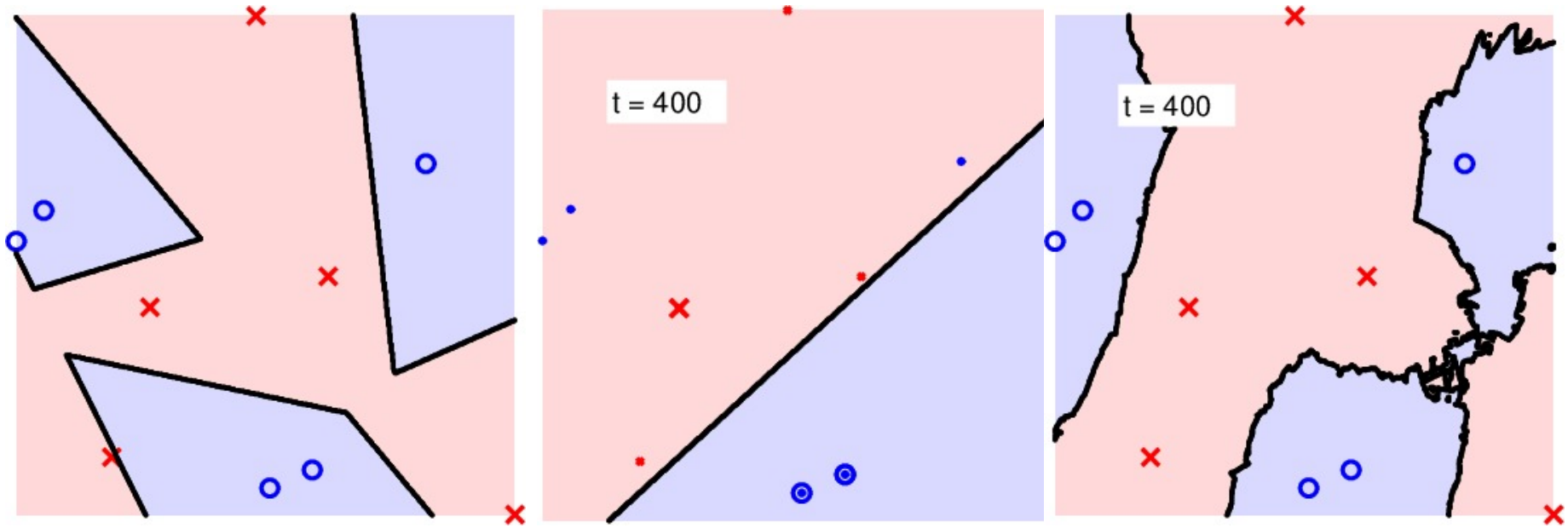


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

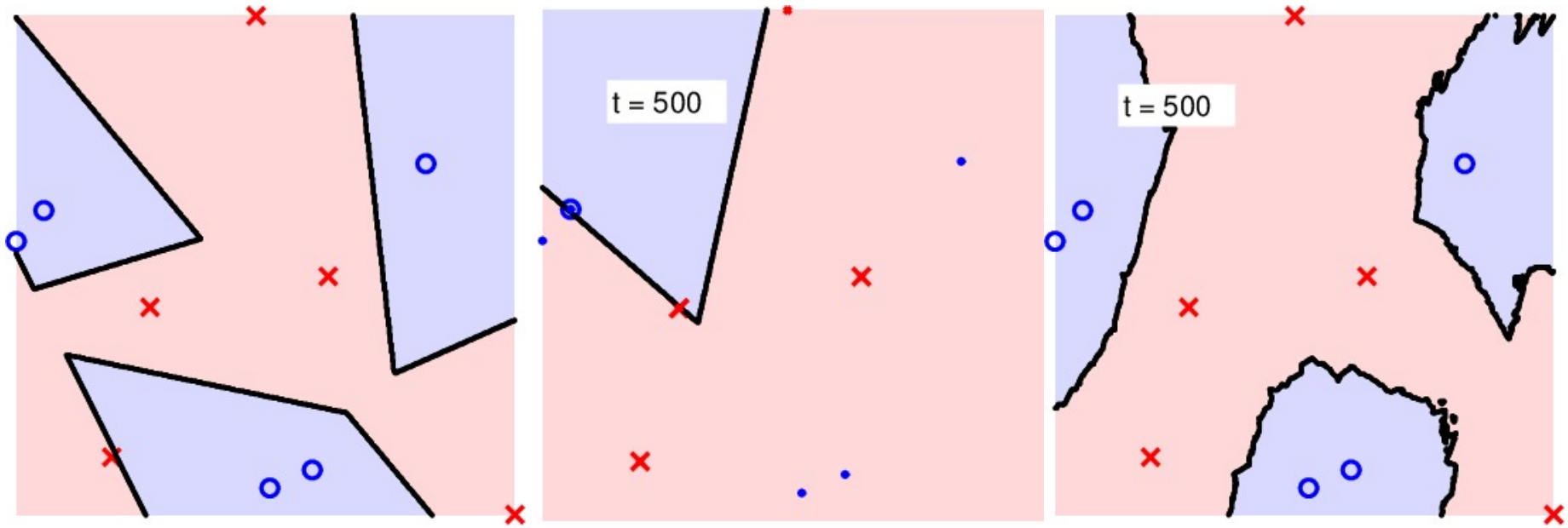


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

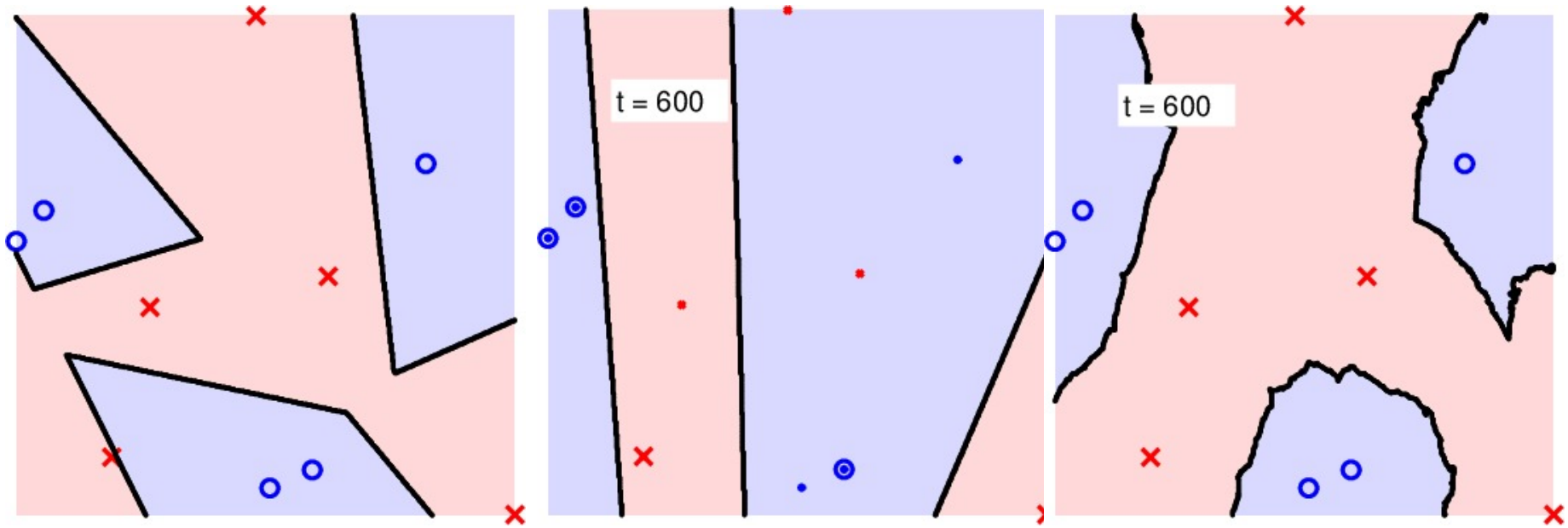


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

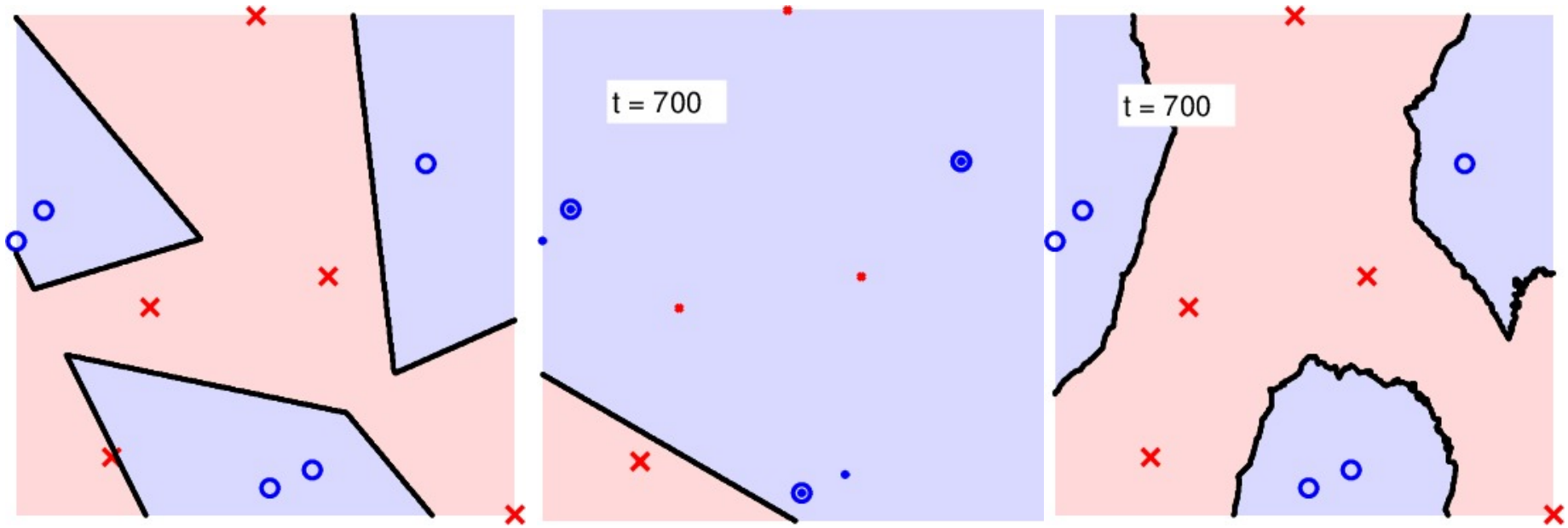


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

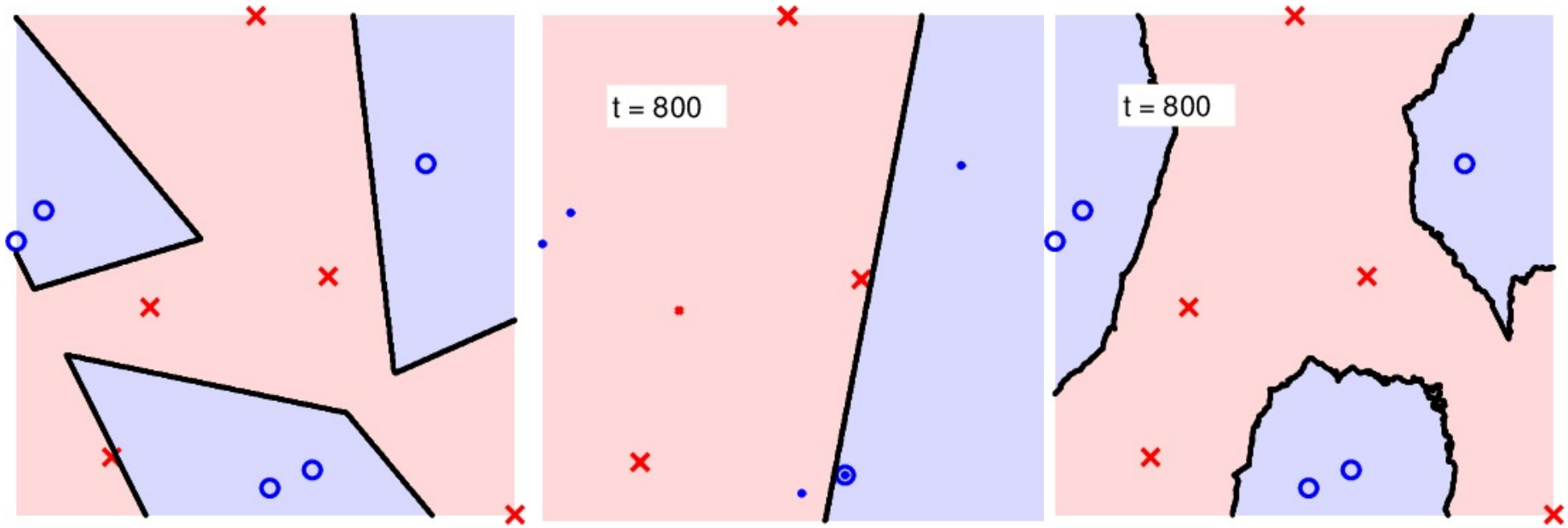


A Simple Data Set

$g_{C\&RT}$ with random combination

g_t $(N' = N/2)$

G with first t trees

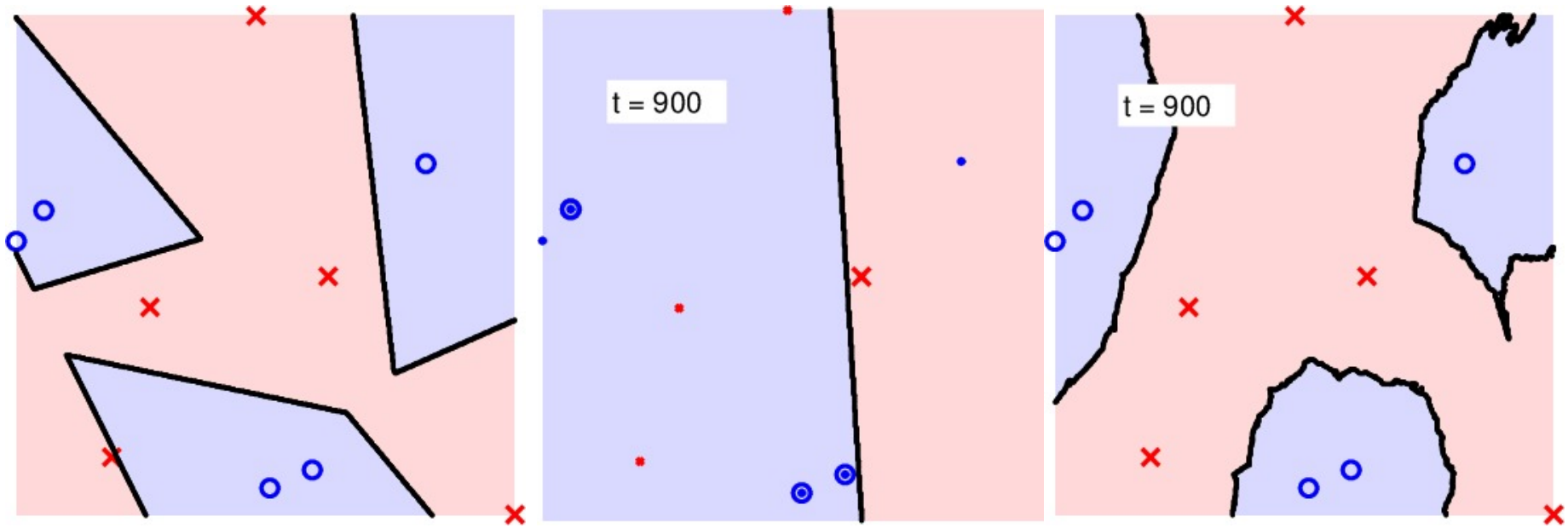


A Simple Data Set

$g_{\text{C\&RT}}$ with random combination

g_t $(N' = N/2)$

G with first t trees

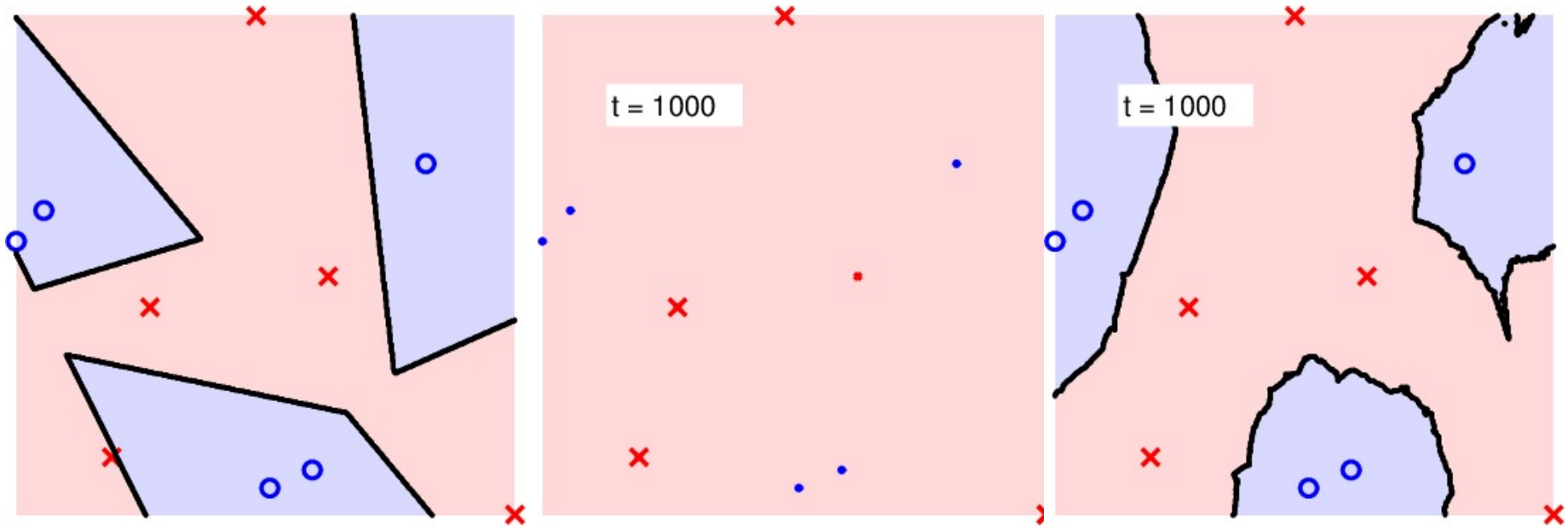


A Simple Data Set

$g_{C\&RT}$ with random combination

g_t $(N' = N/2)$

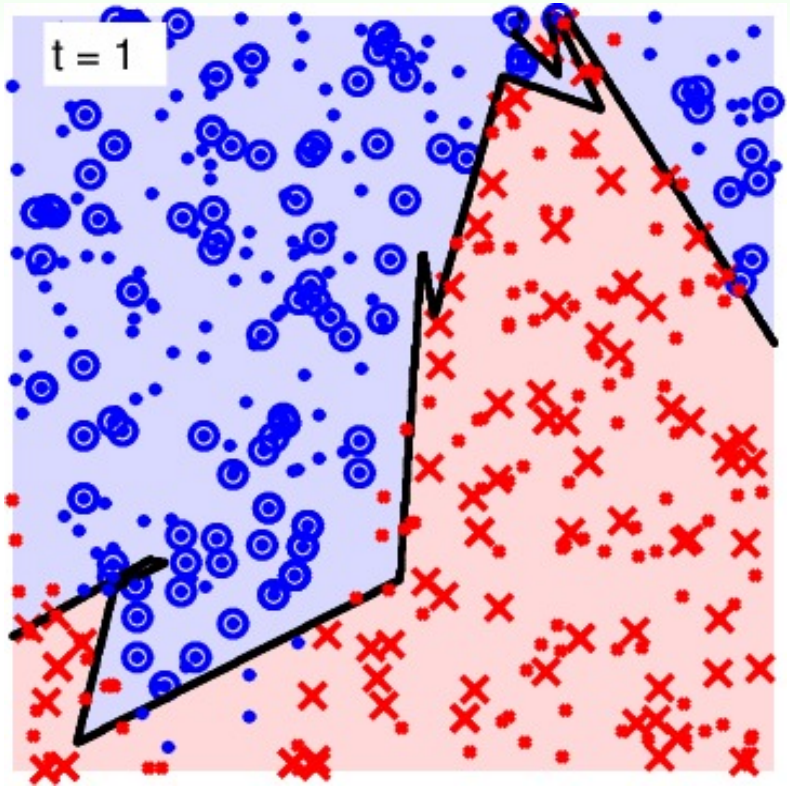
G with first t trees



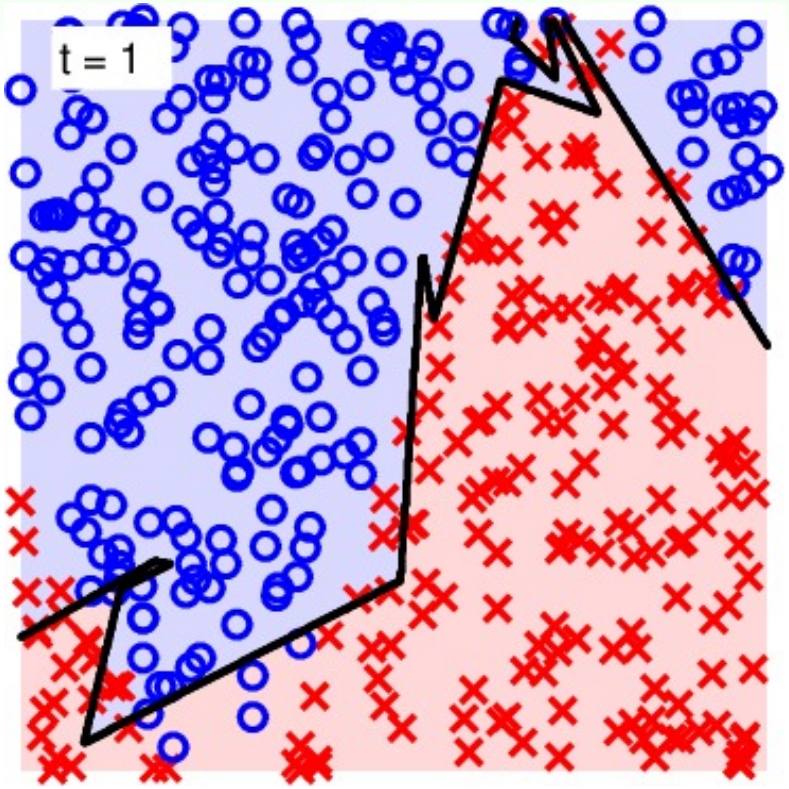
'smooth' and large-margin-like boundary with many trees

A Complicated Data Set

g_t $(N' = N/2)$

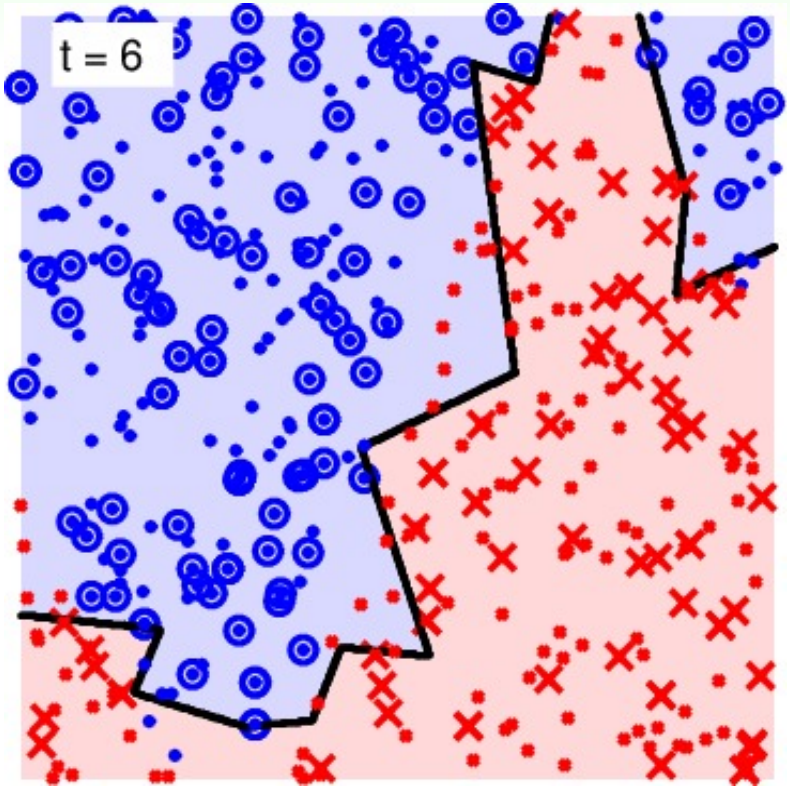


G with first t trees

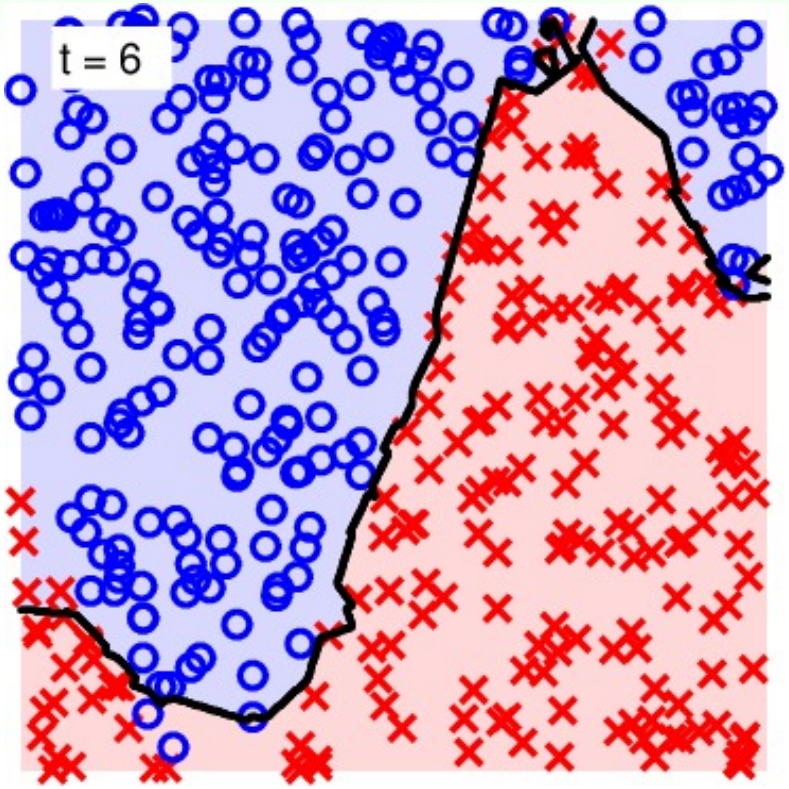


A Complicated Data Set

g_t $(N' = N/2)$

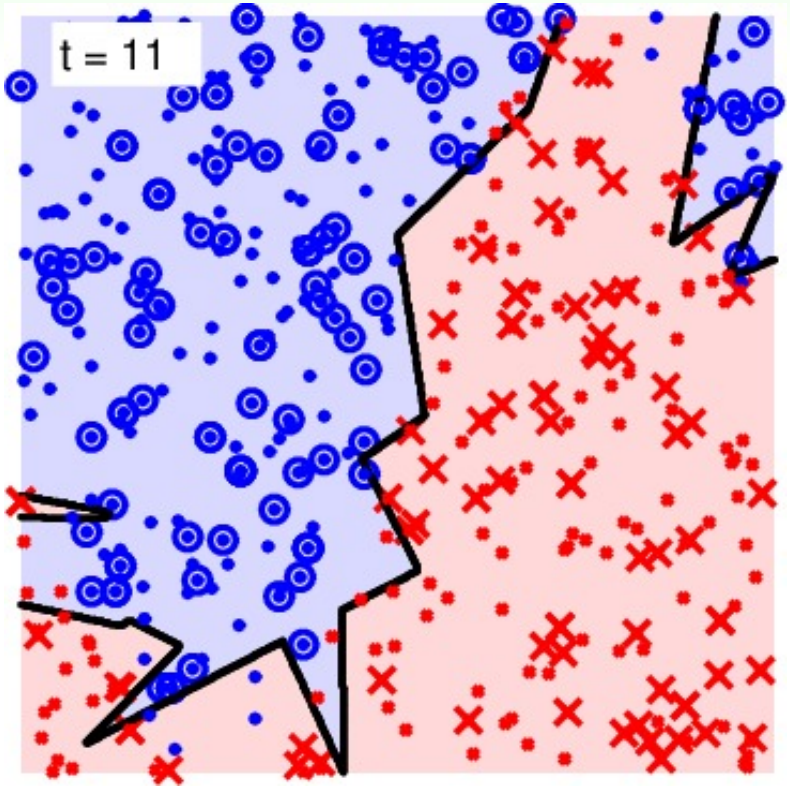


G with first t trees

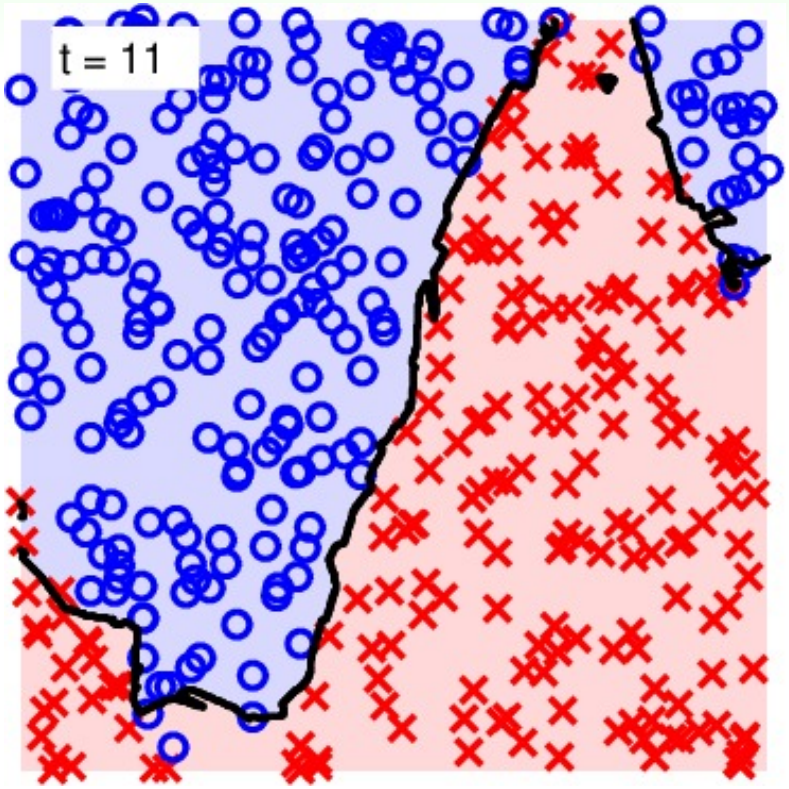


A Complicated Data Set

g_t ($N' = N/2$)

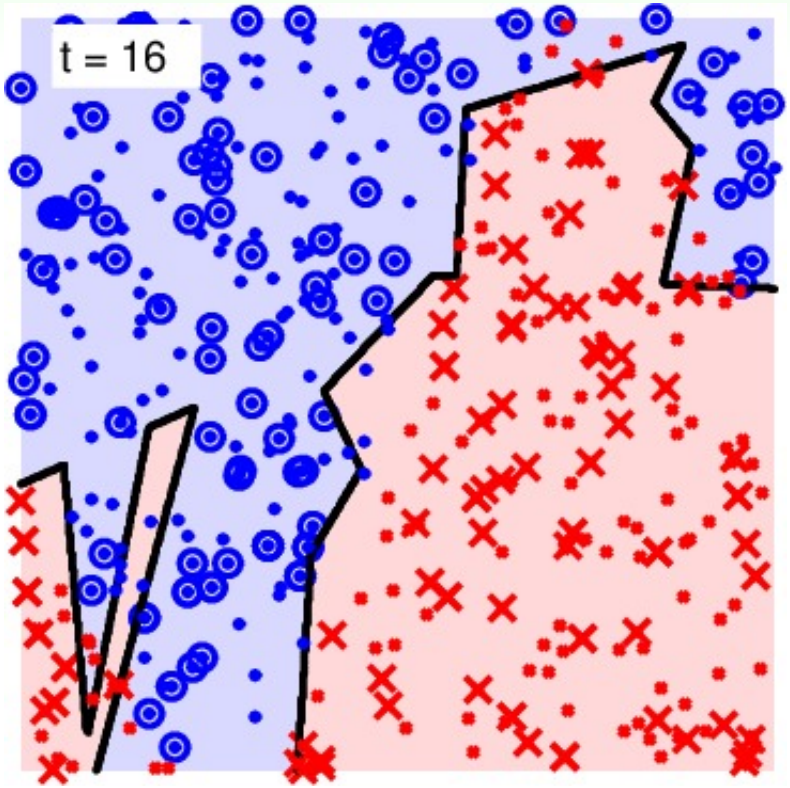


G with first t trees

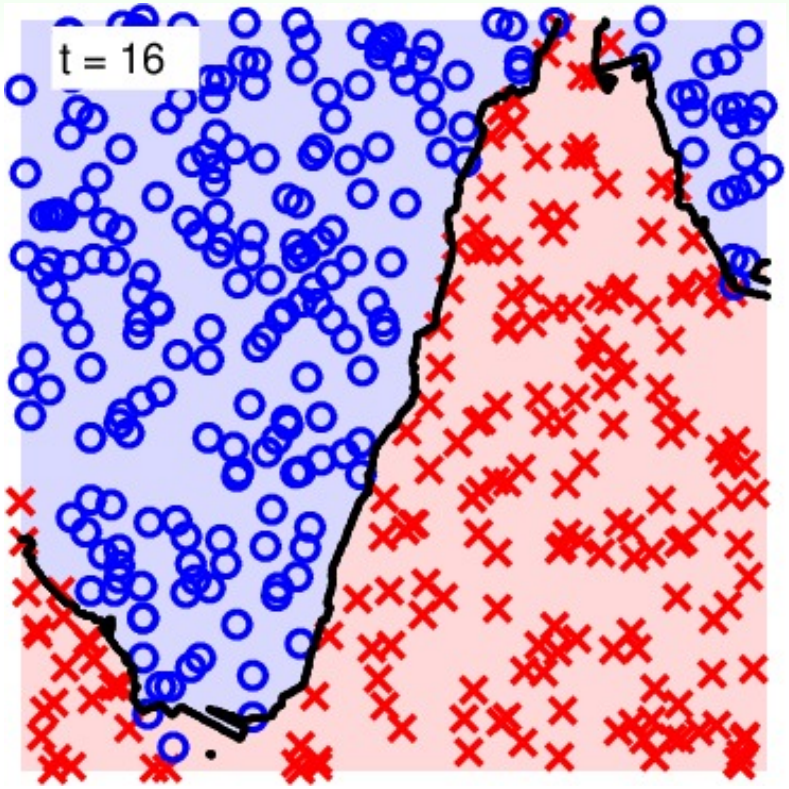


A Complicated Data Set

g_t ($N' = N/2$)

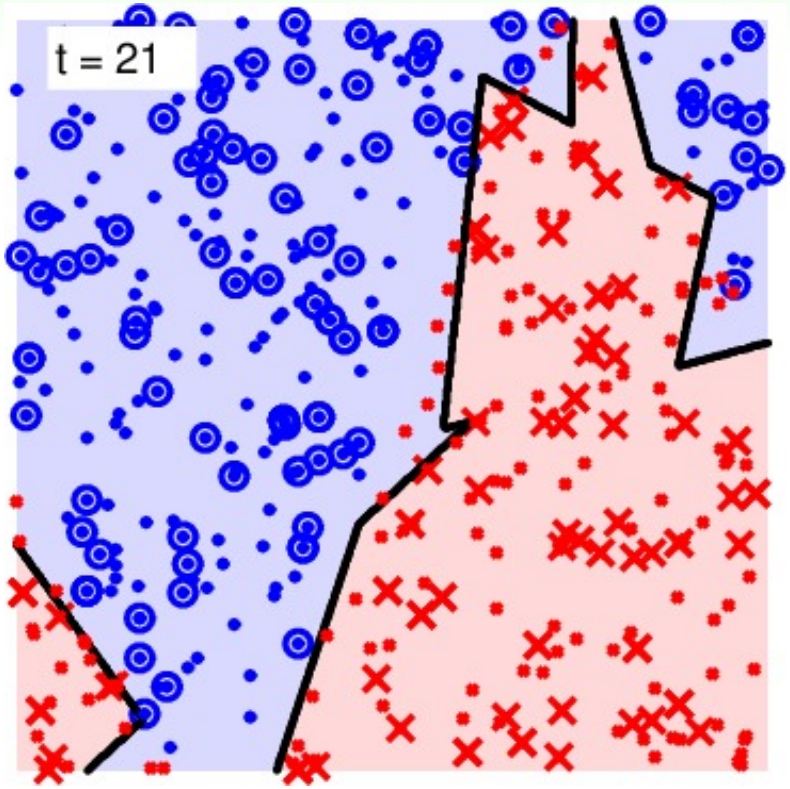


G with first t trees

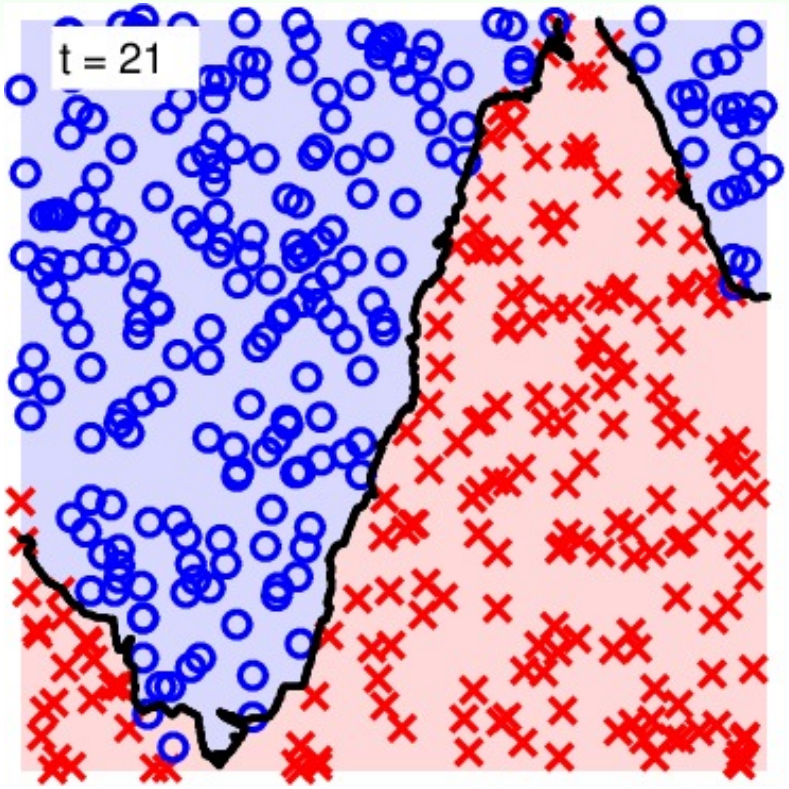


A Complicated Data Set

g_t $(N' = N/2)$



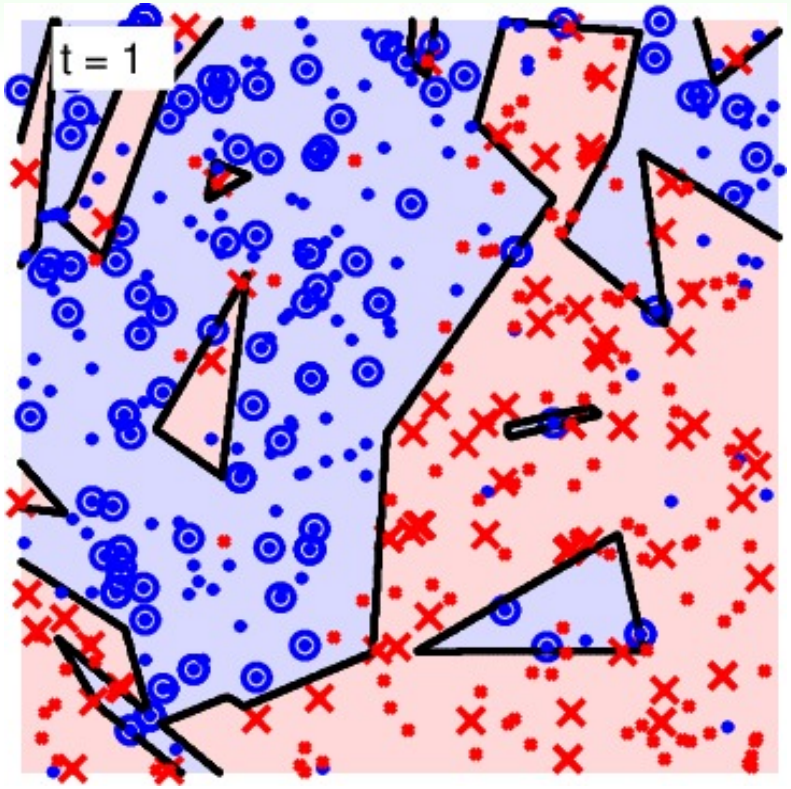
G with first t trees



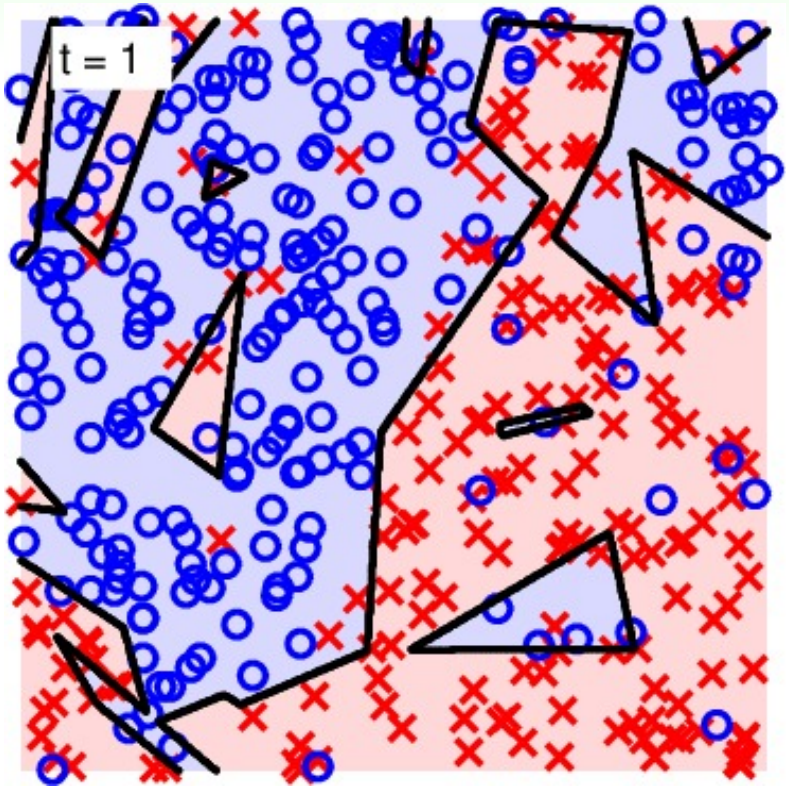
'easy yet robust' nonlinear model

A Complicated and Noisy Data Set

g_t ($N' = N/2$)

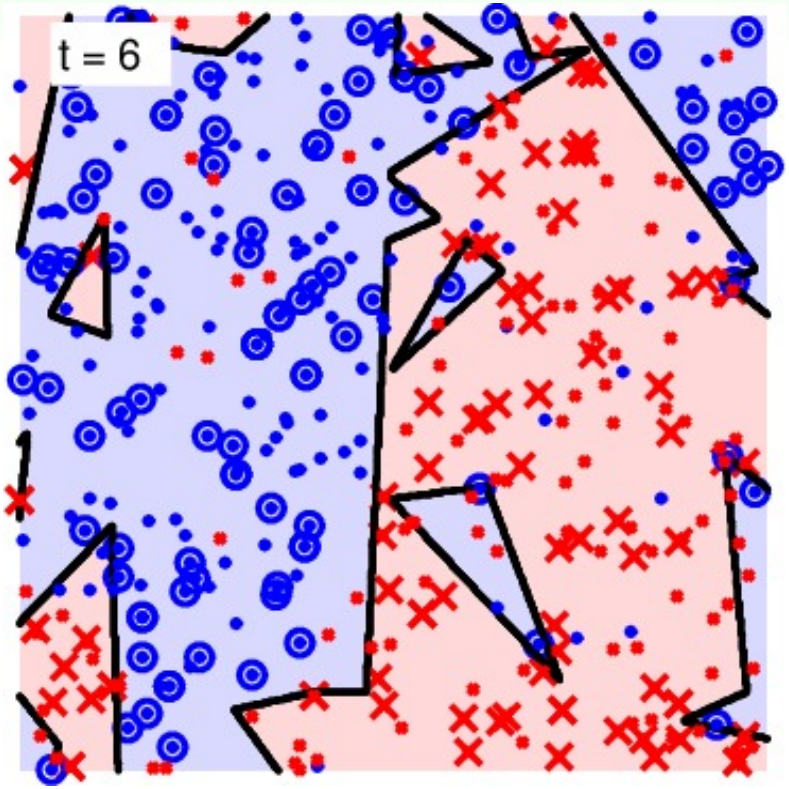


G with first t trees

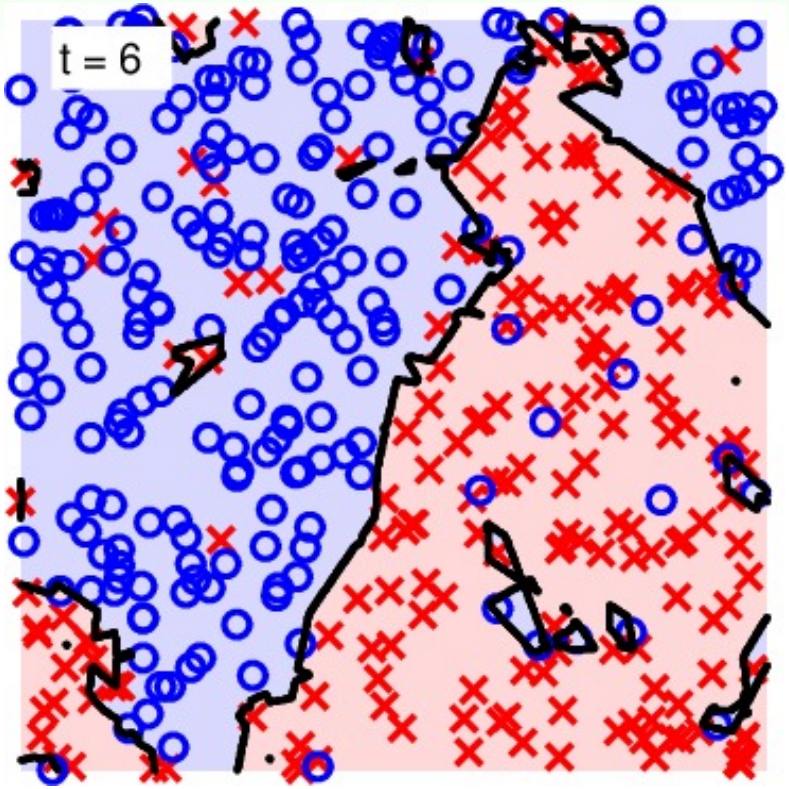


A Complicated and Noisy Data Set

g_t ($N' = N/2$)

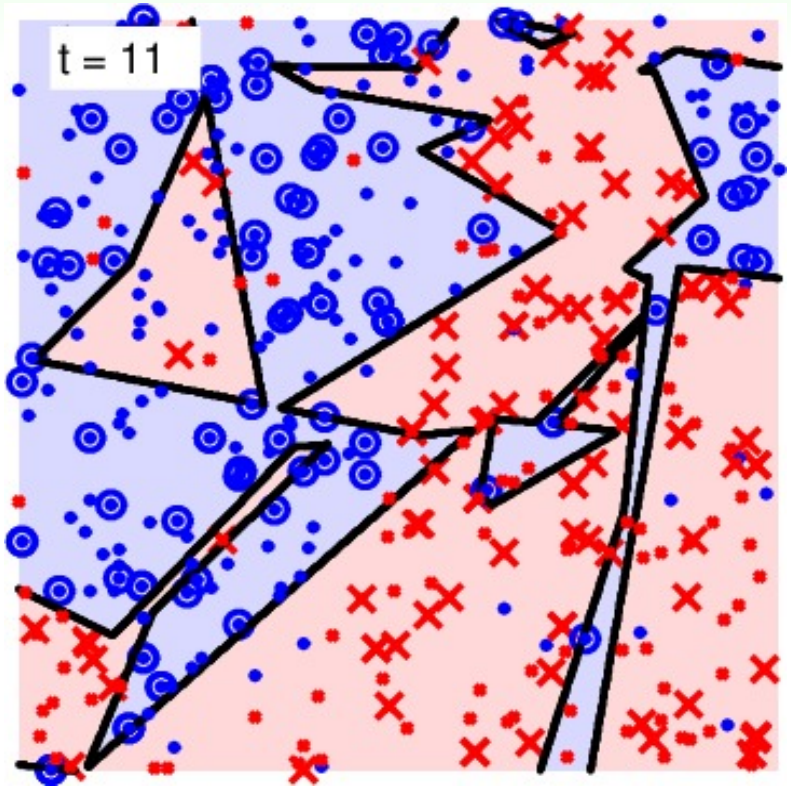


G with first t trees

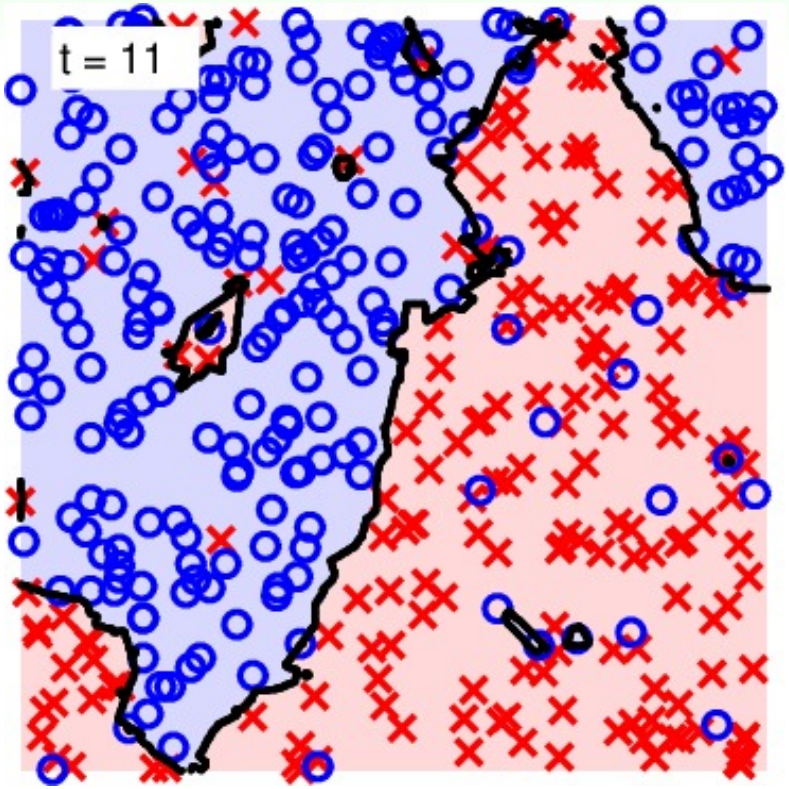


A Complicated and Noisy Data Set

g_t ($N' = N/2$)

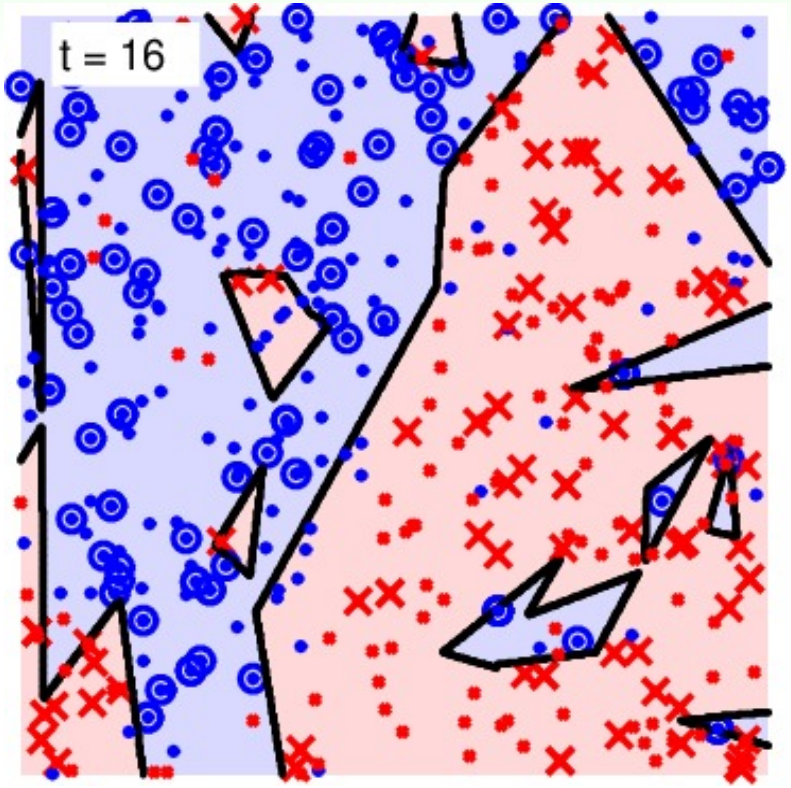


G with first t trees

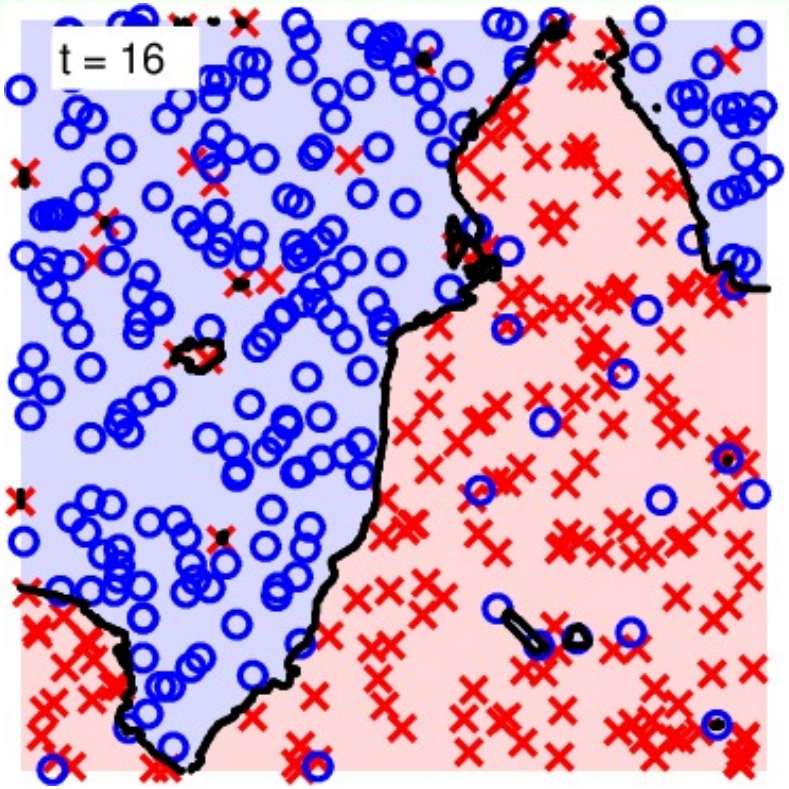


A Complicated and Noisy Data Set

g_t ($N' = N/2$)

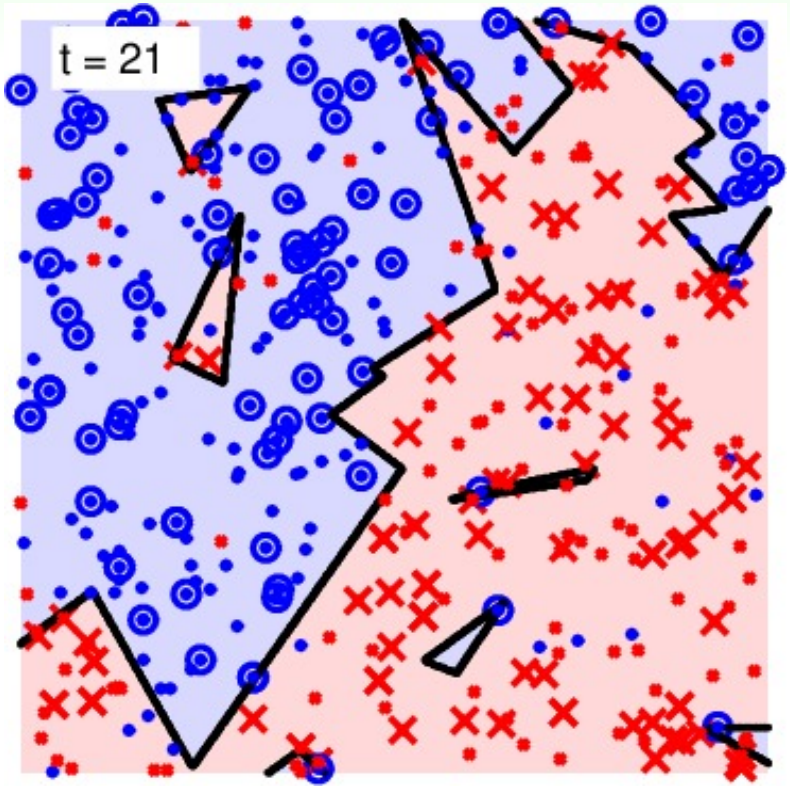


G with first t trees

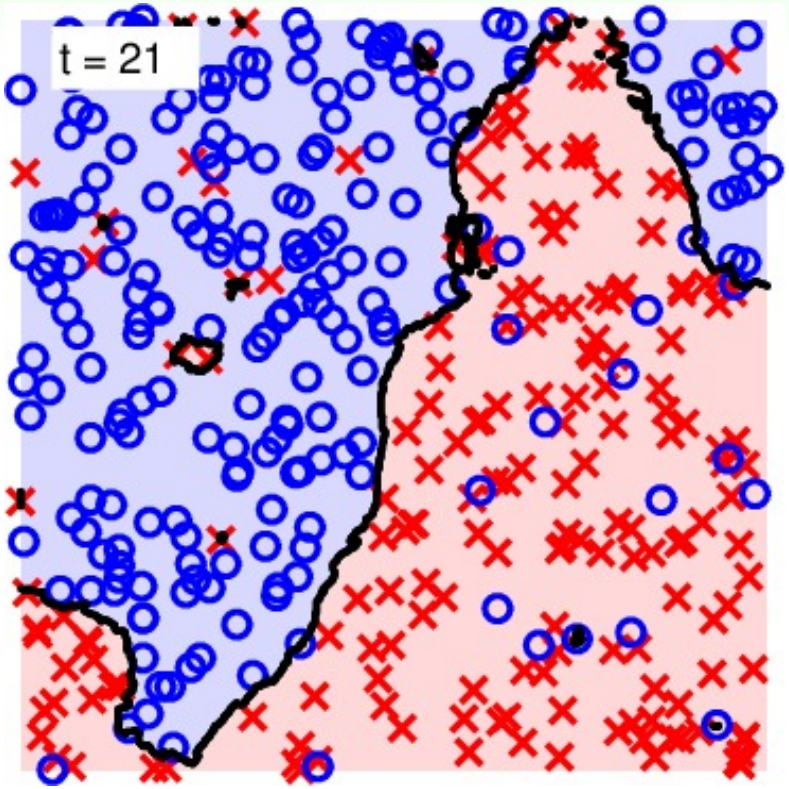


A Complicated and Noisy Data Set

g_t ($N' = N/2$)



G with first t trees



noise corrected by voting

How Many Trees Needed?

almost every theory: the more, **the 'better'**
assuming **good** $\bar{g} = \lim_{T \to \infty} G$

Our NTU Experience

- KDDCup 2013 Track 1 (**yes, NTU is world champion again! :-)**): predicting author-paper relation
- E_{val} of **thousands** of trees: $[0.015, 0.019]$ depending **on seed**; E_{out} of top 20 teams: $[0.014, 0.019]$
- decision: take 12000 **trees** with **seed** 1

cons of RF: may need lots of trees **if the whole random process too unstable**
—should double-check **stability of** G
to ensure **enough trees**

Questions?

AdaBoost Revisited: Example Weights

$$u_n^{(t+1)} = \begin{cases} u_n^{(t)} \cdot \Diamond_t & \textbf{if incorrect} \\ u_n^{(t)} / \Diamond_t & \textbf{if correct} \end{cases}$$

$$= u_n^{(t)} \cdot \Diamond_t^{-y_n g_t(\mathbf{x}_n)} = u_n^{(t)} \cdot \exp\left(-y_n \alpha_t g_t(\mathbf{x}_n)\right)$$

$$u_n^{(T+1)} = u_n^{(1)} \cdot \prod_{t=1}^{T} \exp\left(-y_n \alpha_t g_t(\mathbf{x}_n)\right) = \frac{1}{N} \cdot \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

- recall: $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$
- $\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})$: **voting score** of $\{g_t\}$ on $\mathbf{x}$

AdaBoost: $u_n^{(T+1)} \propto \exp\left(-y_n(\text{ voting score on } \mathbf{x}_n)\right)$

Voting Score and Margin

linear blending = LinModel + hypotheses as transform + ~~constraints~~

$$G(\mathbf{x}_n) = \text{sign}\left(\overbrace{\sum_{t=1}^{T} \underbrace{\alpha_t}_{w_i} \underbrace{g_t(\mathbf{x}_n)}_{\phi_i(\mathbf{x}_n)}}^{\text{voting score}}\right)$$

and hard-margin SVM **margin** = $\frac{y_n \cdot (\mathbf{w}^T \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$, **remember? :-)**

y_n(voting score) = signed & unnormalized **margin**

want y_n(voting score) **positive & large**

$\Leftrightarrow$ $\exp(-y_n(\text{voting score}))$ **small**

$\Leftrightarrow$ $u_n^{(T+1)}$ **small**

claim: AdaBoost **decreases** $\sum_{n=1}^{N} u_n^{(t)}$

AdaBoost Error Function

claim: AdaBoost decreases $\sum_{n=1}^{N} u_n^{(t)}$ and thus somewhat **minimizes**

$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

linear score $s = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)$

- $\text{err}_{0/1}(s, y) = [\![ys \leq 0]\!]$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$: upper bound of $\text{err}_{0/1}$ —called **exponential error measure**

$\widehat{\text{err}}_{\text{ADA}}$: **algorithmic error measure** by **convex upper bound** of $\text{err}_{0/1}$

AdaBoost Error Function

claim: AdaBoost decreases $\sum_{n=1}^{N} u_n^{(t)}$ and thus somewhat **minimizes**

$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

linear score $s = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)$

- $\text{err}_{0/1}(s, y) = [\![ys \leq 0]\!]$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$: upper bound of $\text{err}_{0/1}$ —called **exponential error measure**

$\widehat{\text{err}}_{\text{ADA}}$: **algorithmic error measure** by **convex upper bound** of $\text{err}_{0/1}$

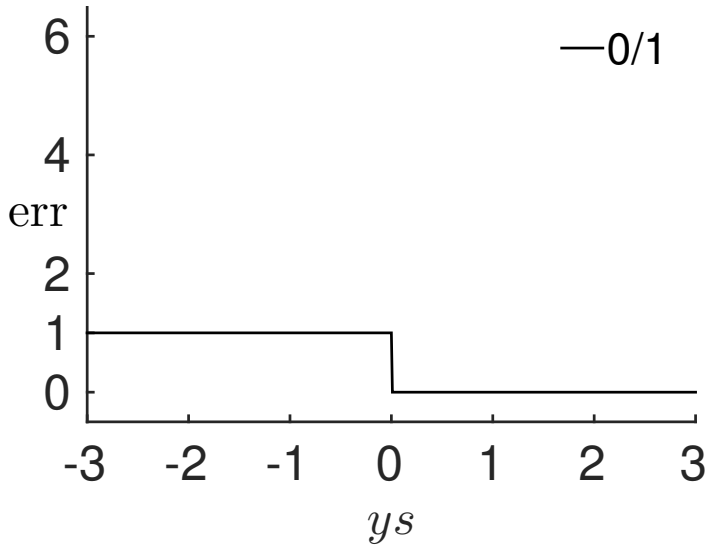
AdaBoost Error Function

claim: AdaBoost decreases $\sum_{n=1}^{N} u_n^{(t)}$ and thus somewhat **minimizes**

$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

linear score $s = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)$

- $\text{err}_{0/1}(s, y) = [\![ys \le 0]\!]$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$: upper bound of $\text{err}_{0/1}$ —called **exponential error measure**



$\widehat{\text{err}}_{\text{ADA}}$: **algorithmic error measure** by **convex upper bound** of $\text{err}_{0/1}$

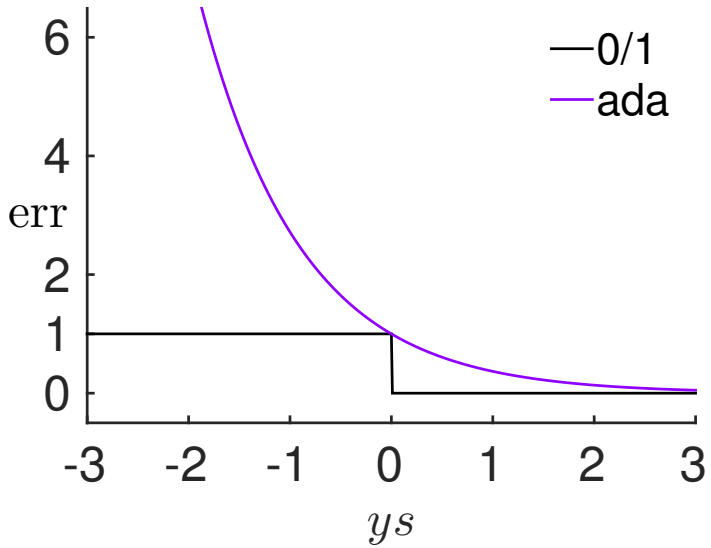
AdaBoost Error Function

claim: AdaBoost decreases $\sum_{n=1}^{N} u_n^{(t)}$ and thus somewhat **minimizes**

$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

linear score $s = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)$

- $\text{err}_{0/1}(s, y) = [\![ys \le 0]\!]$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$: upper bound of $\text{err}_{0/1}$ —called **exponential error measure**



$\widehat{\text{err}}_{\text{ADA}}$: **algorithmic error measure** by **convex upper bound** of $\text{err}_{0/1}$

Gradient Descent on AdaBoost Error Function

recall: gradient descent (**remember? :-)**), at iteration t

$$\min_{\|\mathbf{v}\|=1} E_{\text{in}}(\mathbf{w}_t + \eta\mathbf{v}) \approx \underbrace{E_{\text{in}}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \mathbf{v}^T \underbrace{\nabla E_{\text{in}}(\mathbf{w}_t)}_{\text{known}}$$

at iteration t, to find g_t, solve

$$\min_h \quad \widehat{E}_{\text{ADA}} = \frac{1}{N}\sum_{n=1}^{N} \exp\left(-y_n\left(\sum_{\tau=1}^{t-1}\alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right)$$

$$= \sum_{n=1}^{N} u_n^{(t)} \exp\left(-y_n \eta h(\mathbf{x}_n)\right)$$

$$\overset{\text{taylor}}{\approx} \sum_{n=1}^{N} u_n^{(t)}\left(1 - y_n\eta h(\mathbf{x}_n)\right) = \sum_{n=1}^{N} u_n^{(t)} - \eta\sum_{n=1}^{N} u_n^{(t)} y_n h(\mathbf{x}_n)$$

good h: minimize $\sum_{n=1}^{N} u_n^{(t)}\left(-y_n h(\mathbf{x}_n)\right)$

Learning Hypothesis as Optimization

finding good h (function direction) $\Leftrightarrow$ minimize $\sum_{n=1}^{N} u_n^{(t)} (-y_n h(\mathbf{x}_n))$

for binary classification, where y_n and $h(\mathbf{x}_n)$ both $\in \{-1, +1\}$:

$$\begin{aligned}
\sum_{n=1}^{N} u_n^{(t)} (-y_n h(\mathbf{x}_n)) &= \sum_{n=1}^{N} u_n^{(t)} \begin{cases} -1 & \text{if } y_n = h(\mathbf{x}_n) \\ +1 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\
&= -\sum_{n=1}^{N} u_n^{(t)} + \sum_{n=1}^{N} u_n^{(t)} \begin{cases} 0 & \text{if } y_n = h(\mathbf{x}_n) \\ 2 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\
&= -\sum_{n=1}^{N} u_n^{(t)} + 2 E_{\text{in}}^{\mathbf{u}^{(t)}}(h) \cdot N
\end{aligned}$$

—who minimizes $E_{\text{in}}^{\mathbf{u}^{(t)}}(h)$? $\mathcal{A}$ **in AdaBoost! :-)**

$\mathcal{A}$: **good** $g_t = h$ for 'gradient descent'

Deciding Blending Weight as Optimization

AdaBoost finds g_t by approximately $\min_h \widehat{E}_{\text{ADA}} = \sum_{n=1}^{N} u_n^{(t)} \exp\left(-y_n \eta h(\mathbf{x}_n)\right)$

after finding g_t, how about $\min_\eta \widehat{E}_{\text{ADA}} = \sum_{n=1}^{N} u_n^{(t)} \exp\left(-y_n \eta g_t(\mathbf{x}_n)\right)$

- optimal η_t somewhat **'greedily faster'** than fixed (small) η
  —called **steepest** descent for optimization
- two cases inside summation:
  - $y_n = g_t(\mathbf{x}_n)$: $u_n^{(t)} \exp(-\eta)$ (correct)
  - $y_n \neq g_t(\mathbf{x}_n)$: $u_n^{(t)} \exp(+\eta)$ (incorrect)
- $\widehat{E}_{\text{ADA}} = \left(\sum_{n=1}^{N} u_n^{(t)}\right) \cdot \left((1-\epsilon_t)\exp(-\eta) + \epsilon_t \exp(+\eta)\right)$

by solving $\frac{\partial \widehat{E}_{\text{ADA}}}{\partial \eta} = 0$, **steepest** $\eta_t = \ln\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \alpha_t$, **remember? :-)**
—AdaBoost: steepest descent with approximate functional gradient

Questions?

Gradient Boosting for Arbitrary Error Function

AdaBoost

$$\min_{\eta} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right)$$

with binary-output hypothesis h

GradientBoost

$$\min_{\eta} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \text{err}\left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n), y_n\right)$$

with any hypothesis h (usually real-output hypothesis)

GradientBoost: allows **extension to different** err for regression/soft classification/etc.

GradientBoost for Regression

$$\min_{\eta} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \text{err}\Big(\underbrace{\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n)}_{s_n} + \eta h(\mathbf{x}_n), y_n \Big) \quad \text{with } \text{err}(s, y) = (s - y)^2$$

$$\min_{h} \ldots \overset{\text{taylor}}{\approx} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \underbrace{\text{err}(s_n, y_n)}_{\text{constant}} + \frac{1}{N} \sum_{n=1}^{N} \eta h(\mathbf{x}_n) \left. \frac{\partial \text{err}(s, y_n)}{\partial s} \right|_{s=s_n}$$

$$= \min_{h} \text{constants} + \frac{\eta}{N} \sum_{n=1}^{N} h(\mathbf{x}_n) \cdot 2(s_n - y_n)$$

naïve solution $h(\mathbf{x}_n) = -\infty \cdot (s_n - y_n)$
if no constraint on h

Learning Hypothesis as Optimization

$$\min_{h} \quad \text{constants} + \frac{\eta}{N} \sum_{n=1}^{N} 2h(\mathbf{x}_n)(s_n - y_n)$$

- magnitude of h does not matter: because η will be optimized next
- penalize large magnitude to avoid naïve solution

$$\min_{h} \quad \text{constants} + \frac{\eta}{N} \sum_{n=1}^{N} \left(2h(\mathbf{x}_n)(s_n - y_n) + (h(\mathbf{x}_n))^2\right)$$

$$= \quad \text{constants} + \frac{\eta}{N} \sum_{n=1}^{N} \left(\text{constant} + \left(h(\mathbf{x}_n) - (y_n - s_n)\right)^2\right)$$

- solution of penalized approximate functional gradient: squared-error regression on $\{(\mathbf{x}_n, \underbrace{y_n - s_n}_{\text{residual}})\}$

GradientBoost for regression:
find $g_t = h$ by regression with **residuals**

Deciding Blending Weight as Optimization

after finding $g_t = h$,

$$\min_{\eta} \cancel{\min_{h}} \frac{1}{N} \sum_{n=1}^{N} \text{err}\Big(\underbrace{\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n)}_{s_n} + \eta g_t(\mathbf{x}_n), y_n \Big) \text{ with } \text{err}(s, y) = (s - y)^2$$

$$\min_{\eta} \quad \frac{1}{N} \sum_{n=1}^{N} (s_n + \eta g_t(\mathbf{x}_n) - y_n)^2 = \frac{1}{N} \sum_{n=1}^{N} ((y_n - s_n) - \eta g_t(\mathbf{x}_n))^2$$

—one-variable linear regression on $\{(g_t\text{-transformed input}, \textbf{residual})\}$

GradientBoost for regression: $\alpha_t =$ optimal η
by g_t-transformed linear regression

Putting Everything Together

Gradient Boosted Decision Tree (GBDT)

$s_1 = s_2 = \ldots = s_N = 0$

for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\{(\mathbf{x}_n, y_n - s_n)\})$ where $\mathcal{A}$ is a (squared-error) regression algorithm
   —**how about pruned C&RT?**
2. compute $\alpha_t = \text{OneVarLinearRegression}(\{(g_t(\mathbf{x}_n), y_n - s_n)\})$
3. update $s_n \leftarrow s_n + \alpha_t g_t(\mathbf{x}_n)$

return $G(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})$

GBDT: 'regression sibling' of AdaBoost + Decision Tree
—**popular in practice**

Questions?

Map of Blending Models

blending: aggregate after getting **diverse** g_t

uniform	non-uniform	conditional
simple voting/averaging of g_t	linear model on g_t-transformed inputs	nonlinear model on g_t-transformed inputs

uniform for 'stability';
non-uniform/conditional **carefully** for 'complexity'

Map of Aggregation-Learning Models

learning: aggregate as well as getting **diverse** g_t

Bagging

diverse g_t by
bootstrapping;
uniform vote
by nothing :-)

AdaBoost

diverse g_t
by reweighting;
linear vote
by steepest search

Decision Tree

diverse g_t
by data splitting;
conditional vote
by branching

GradientBoost

diverse g_t
by residual fitting;
linear vote
by steepest search

boosting-like algorithms most popular

Map of Aggregation of Aggregation Models

Bagging

AdaBoost

Decision Tree

Random Forest

randomized bagging
+ 'strong' DTree

AdaBoost-DTree

AdaBoost
+ 'weak' DTree

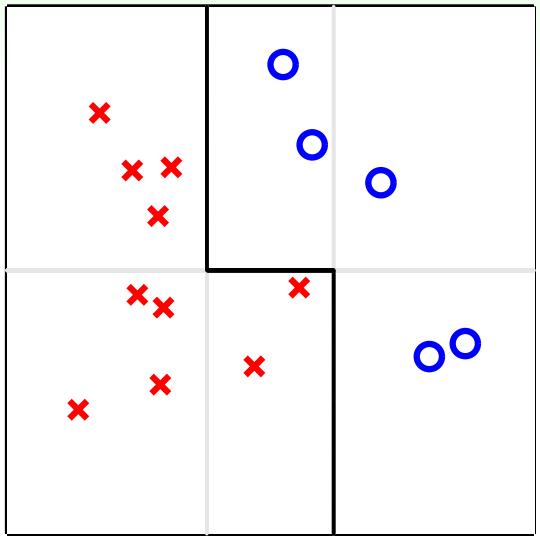
GradientBoost

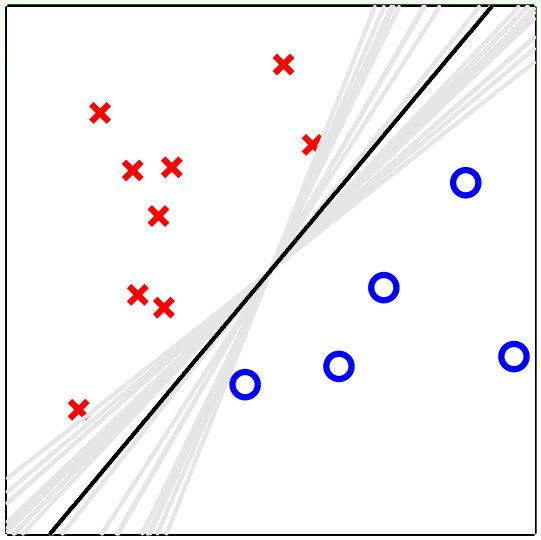
GBDT

GradientBoost
+ 'weak' DTree

all three frequently used in practice

Specialty of Aggregation Models





cure underfitting

- $G(\mathbf{x})$ 'strong'
- aggregation
  $\Longrightarrow$ **feature transform**

cure overfitting

- $G(\mathbf{x})$ 'moderate'
- aggregation
  $\Longrightarrow$ **regularization**

proper aggregation (a.k.a. 'ensemble')
$\Longrightarrow$ **better performance**

Questions?

Summary

1. Combining Predictive Features: Aggregation Models

Lecture 13: Decision Tree Ensembles

- Decision Tree Hypothesis
  express path-conditional aggregation
- Decision Tree Algorithm
  recursive branching until termination to base
- Decision Tree in Action
  explainable and efficient
- Random Forest Algorithm
  bag of trees on randomly projected subspaces
- Out-Of-Bag Estimate
  self-validation with OOB examples
- Random Forest in Action
  'smooth' boundary with many trees
- Optimization View of AdaBoost
  functional gradient descent on exponential error
- Gradient Boosting
  iterative steepest residual fitting
- Summary of Aggregation Models
  some cure underfitting; some cure overfitting