

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

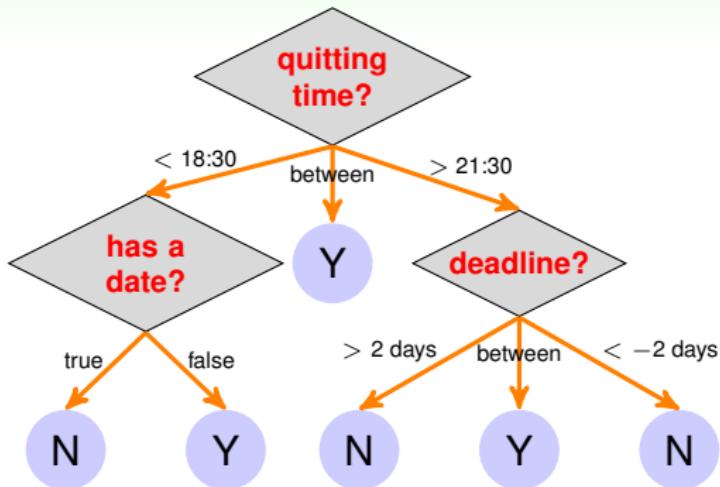
aggregation type	blending	learning
uniform	voting/averaging	Bagging
non-uniform	linear	AdaBoost
conditional	stacking	Decision Tree

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

Decision Tree for Watching MOOC Lectures

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

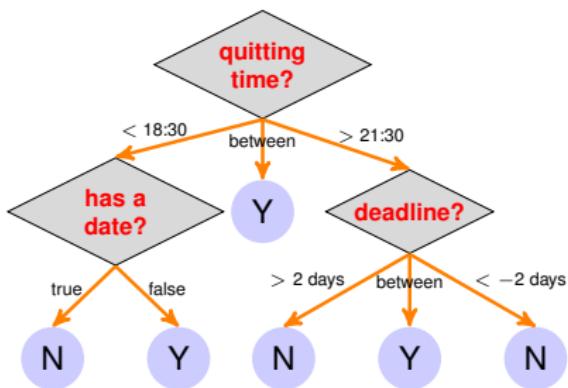
- **base hypothesis $g_t(\mathbf{x})$:** leaf at end of path t , a **constant** here
- **condition $q_t(\mathbf{x})$:** $\llbracket \text{is } \mathbf{x} \text{ on path } t? \rrbracket$
- usually with **simple internal nodes**



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
- heuristics:
'heuristics selection'
confusing to beginners
- arguably no single **representative algorithm**

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

Questions?

A Basic Decision Tree Algorithm

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

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 $G_c \leftarrow \text{DecisionTree}(\mathcal{D}_c)$
- ④ return $G(\mathbf{x}) = \sum_{c=1}^C \llbracket b(\mathbf{x}) = c \rrbracket G_c(\mathbf{x})$

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

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

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:

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

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

for classification

- Gini index:

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

—all k considered together

- classification error:

$$1 - \max_{1 \leq k \leq K} \frac{\sum_{n=1}^N \llbracket y_n = k \rrbracket}{N}$$

—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  $g_t(\mathbf{x}) = E_{in}\text{-optimal constant}$ 
else ...
    ① learn branching criteria

```

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

'forced' to terminate when

- all y_n the same: $\text{impurity} = 0 \implies g_t(\mathbf{x}) = y_n$
- 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 $g_t(\mathbf{x}) = E_{\text{in}}\text{-optimal constant}$

else

① learn **branching criteria**

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

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

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

④ return $G(\mathbf{x}) = \sum_{c=1}^2 [\![\mathbf{b}(\mathbf{x}) = c]\!] G_c(\mathbf{x})$

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 D_c**

- need a **regularizer**, say, $\Omega(G) = \text{NumberOfLeaves}(G)$
- want **regularized** decision tree:

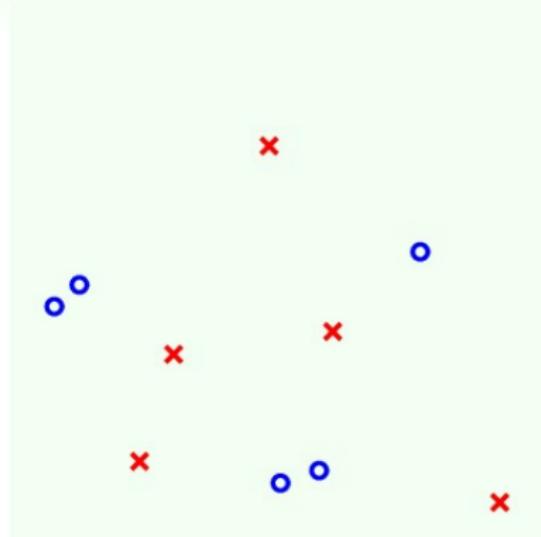
$$\underset{\text{all possible } G}{\operatorname{argmin}} \quad E_{\text{in}}(G) + \lambda \Omega(G)$$

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

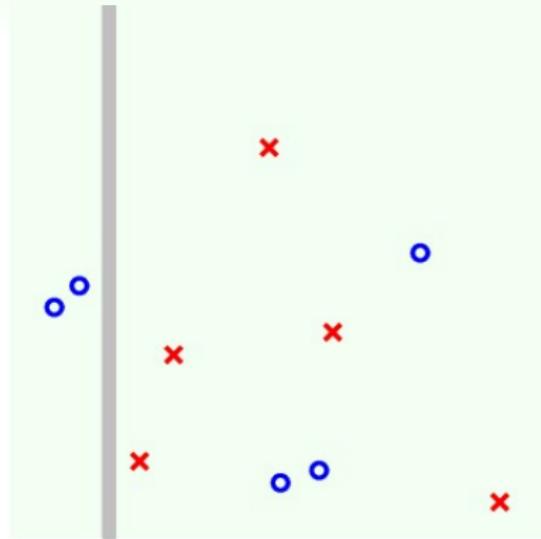
systematic choice of λ ? **validation**

Questions?

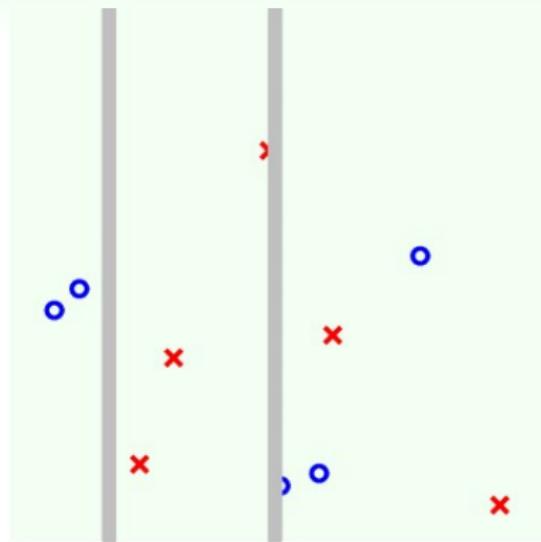
A Simple Data Set



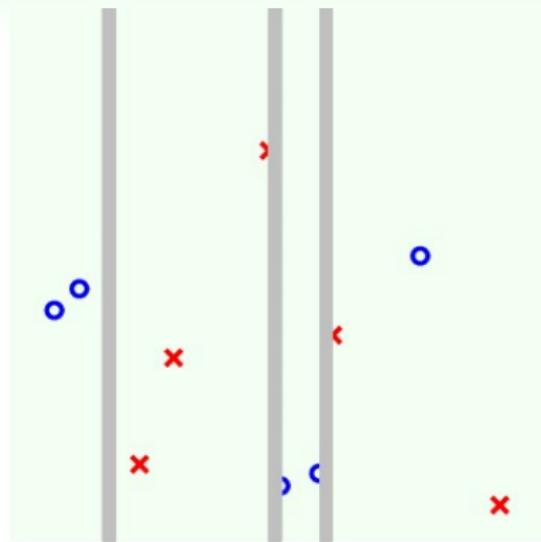
A Simple Data Set



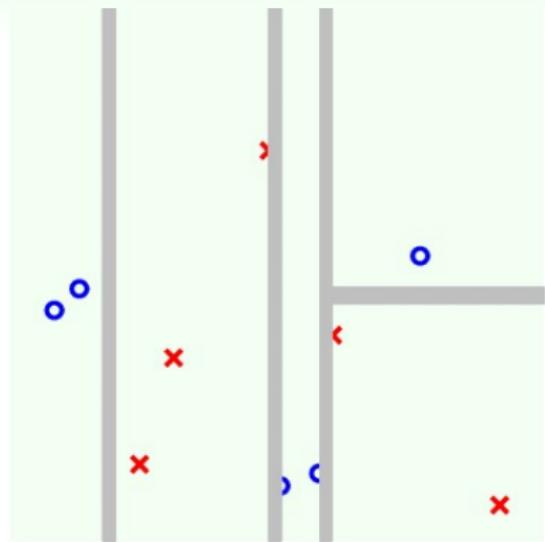
A Simple Data Set



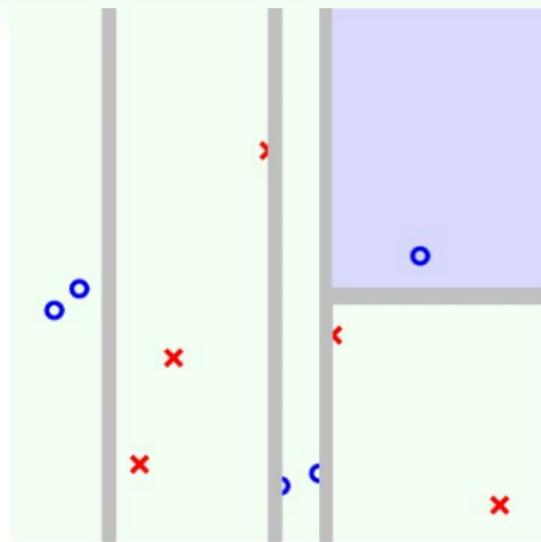
A Simple Data Set



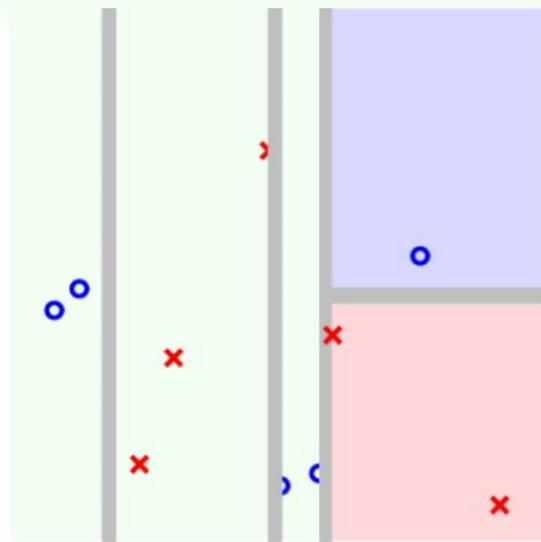
A Simple Data Set



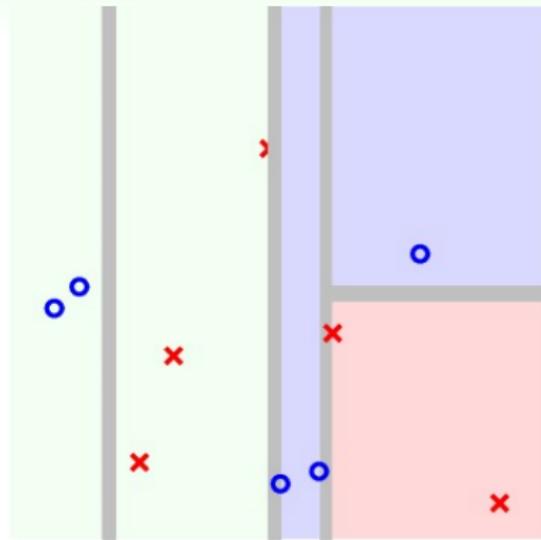
A Simple Data Set



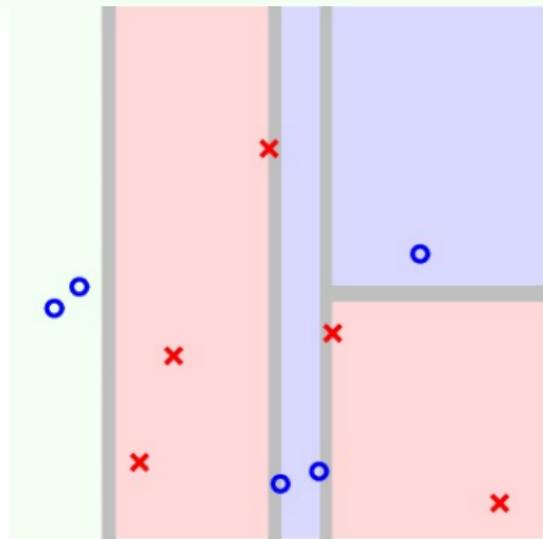
A Simple Data Set



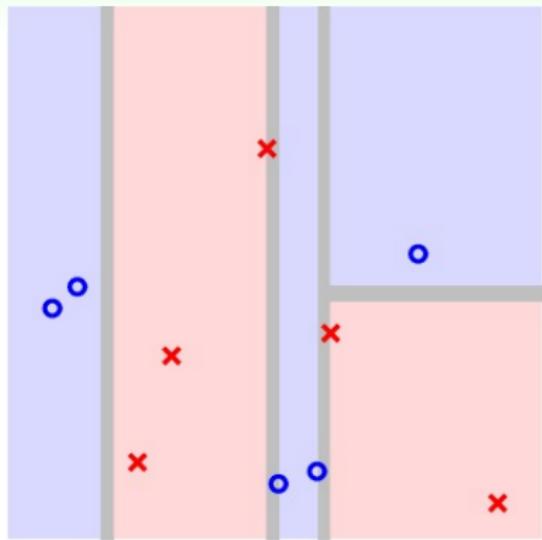
A Simple Data Set



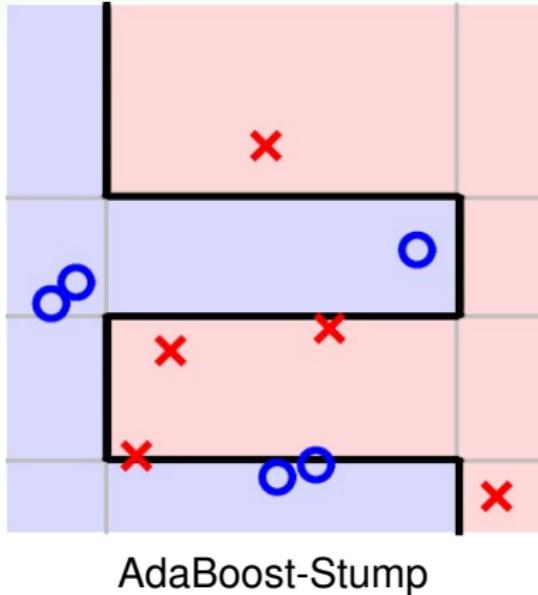
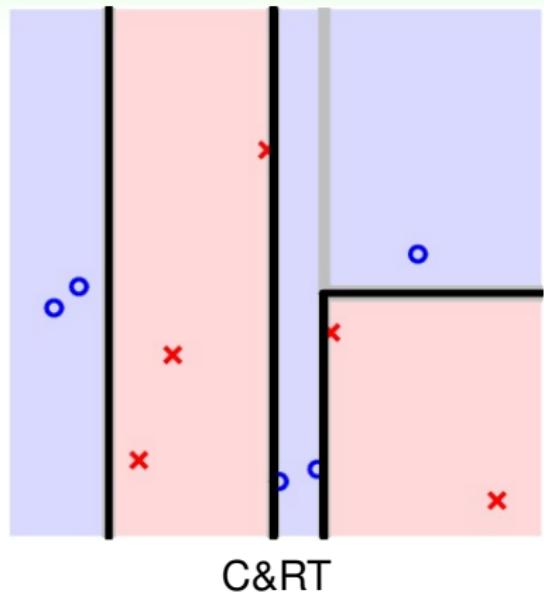
A Simple Data Set



A Simple Data Set

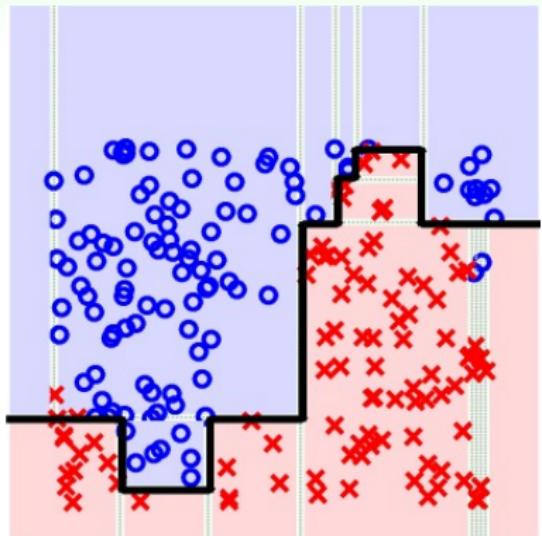


A Simple Data Set

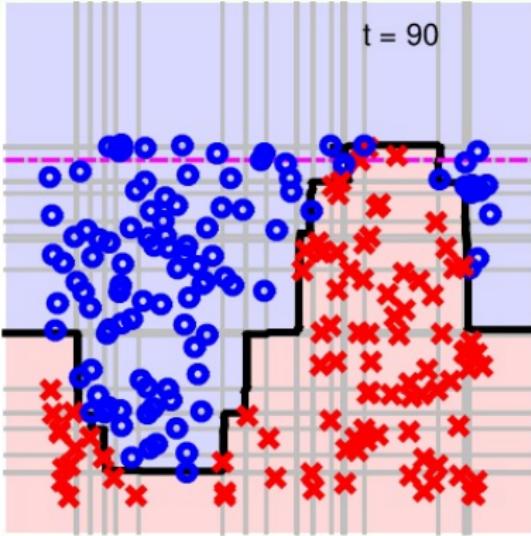


C&RT: 'divide-and-conquer'

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 $\text{Bag}(\mathcal{D}, \mathcal{A})$

For $t = 1, 2, \dots, T$

- 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 $\text{DTree}(\mathcal{D})$

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

—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 $t = 1, 2, \dots, T$

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 $G(\mathbf{x}) =$

$$\sum_{c=1}^C \llbracket b(\mathbf{x}) = c \rrbracket G_c(\mathbf{x})$$

- 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} : $\Phi(\mathbf{x}) = \mathbf{P} \cdot \mathbf{x}$
with row i of \mathbf{P} sampled randomly

- **projection** (combination) with random row \mathbf{p}_i of \mathbf{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 $\text{Bag}(\mathcal{D}, \mathcal{A})$

For $t = 1, 2, \dots, T$

- 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	\dots	g_T
(\mathbf{x}_1, y_1)	$\tilde{\mathcal{D}}_1$	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
(\mathbf{x}_2, y_2)	★	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
(\mathbf{x}_3, y_3)	★	$\tilde{\mathcal{D}}_2$	★		$\tilde{\mathcal{D}}_T$
...					
(\mathbf{x}_N, y_N)	$\tilde{\mathcal{D}}_1$	$\tilde{\mathcal{D}}_2$	★		★

★ 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) \iff not sampled after N' drawings

if $N' = N$

- probability for (\mathbf{x}_n, y_n) to be OOB for g_t : $(1 - \frac{1}{N})^N$
- 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	...	g_T
(\mathbf{x}_1, y_1)	$\tilde{\mathcal{D}}_1$	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
(\mathbf{x}_2, y_2)	★	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
(\mathbf{x}_3, y_3)	★	$\tilde{\mathcal{D}}_2$	★		$\tilde{\mathcal{D}}_T$
...					
(\mathbf{x}_N, y_N)	$\tilde{\mathcal{D}}_1$	★	★		★

Validation

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

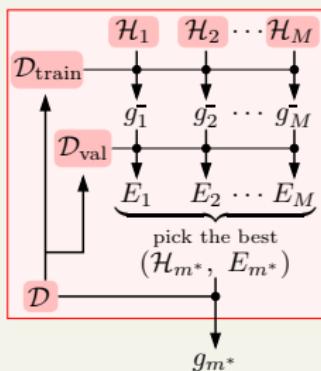
- ★ like \mathcal{D}_{val} : ‘enough’ random examples unused during training
- use ★ to validate g_t ? easy, but **rarely needed**
- use ★ to validate G ? $E_{\text{oob}}(G) = \frac{1}{N} \sum_{n=1}^N \text{err}(y_n, G_n^-(\mathbf{x}_n))$,
with G_n^- contains only trees that \mathbf{x}_n is OOB of,
such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

$$\begin{aligned} g_{m^*} &= \mathcal{A}_{m^*}(\mathcal{D}) \\ m^* &= \operatorname{argmin}_{1 \leq m \leq M} E_m \\ E_m &= E_{\text{val}}(\mathcal{A}_m(\mathcal{D}_{\text{train}})) \end{aligned}$$



RF: by Best E_{oob}

$$\begin{aligned} G_{m^*} &= \text{RF}_{m^*}(\mathcal{D}) \\ m^* &= \operatorname{argmin}_{1 \leq m \leq M} E_m \\ E_m &= E_{\text{oob}}(\text{RF}_m(\mathcal{D})) \end{aligned}$$

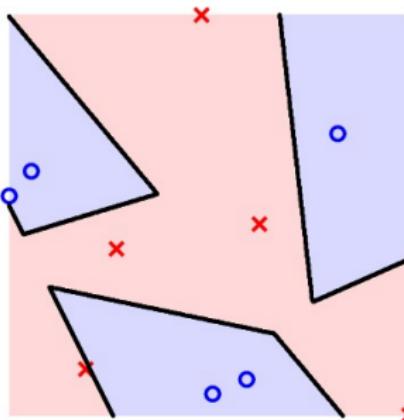
- use E_{oob} for **self-validation** —of RF **parameters** such as d''
- no re-training** needed

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

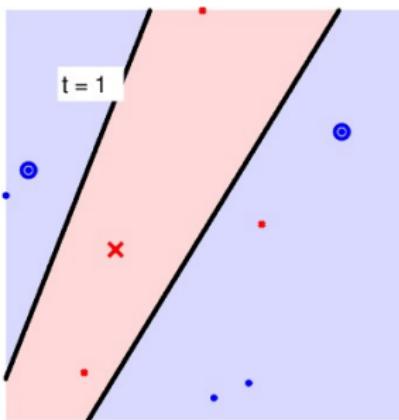
Questions?

A Simple Data Set

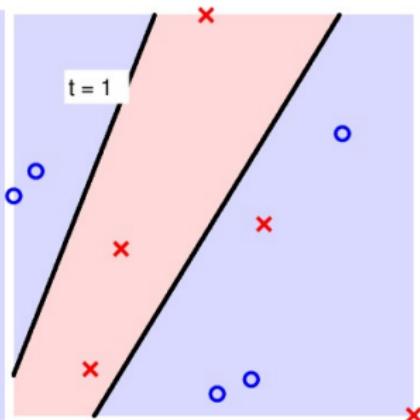
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

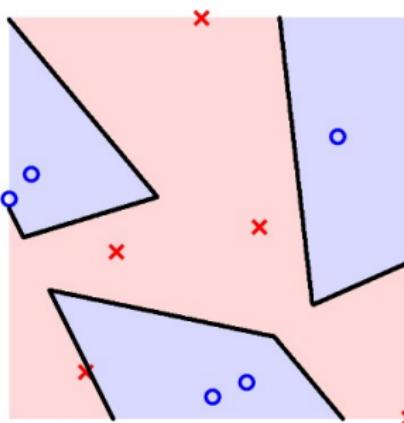


G with first t trees

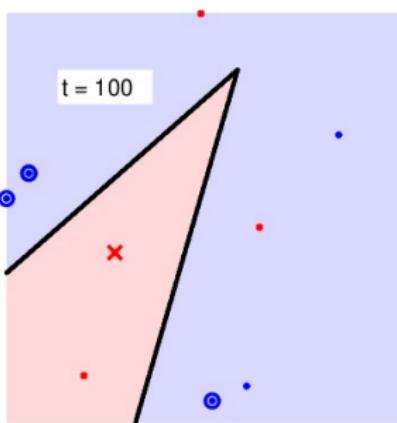


A Simple Data Set

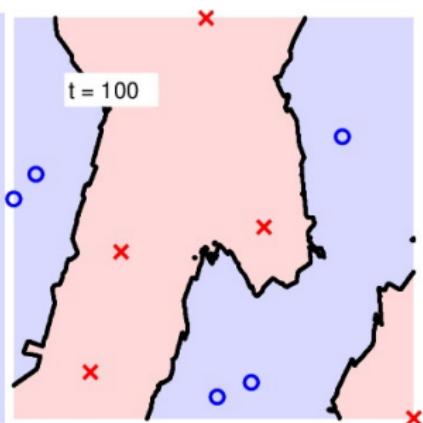
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

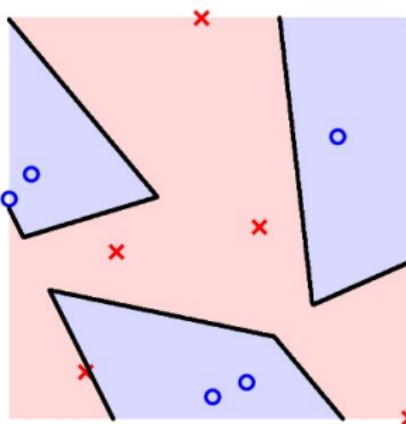


G with first t trees

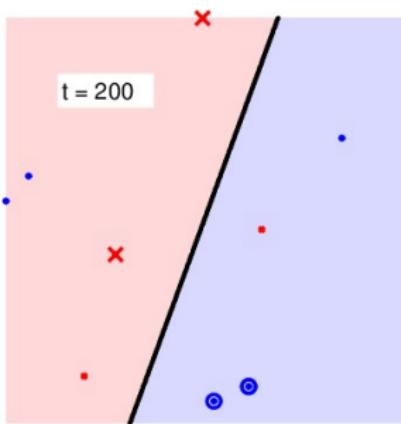


A Simple Data Set

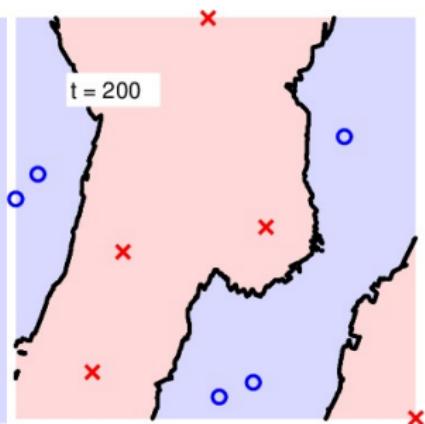
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

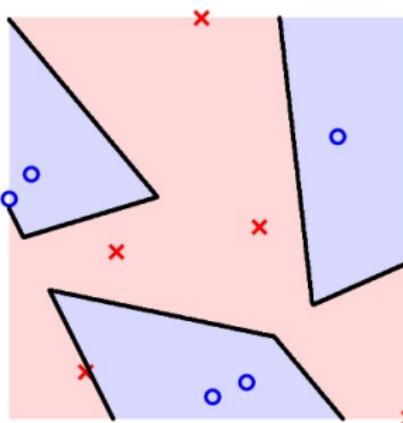


G with first t trees

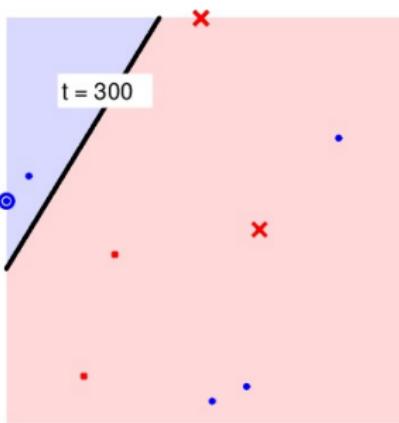


A Simple Data Set

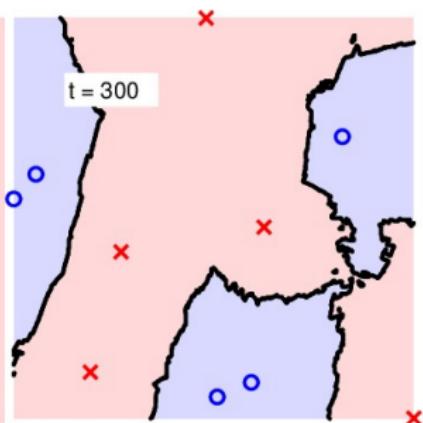
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

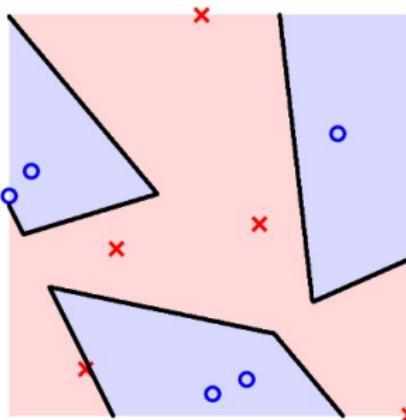


G with first t trees

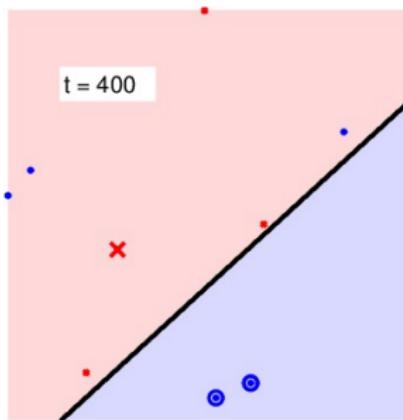


A Simple Data Set

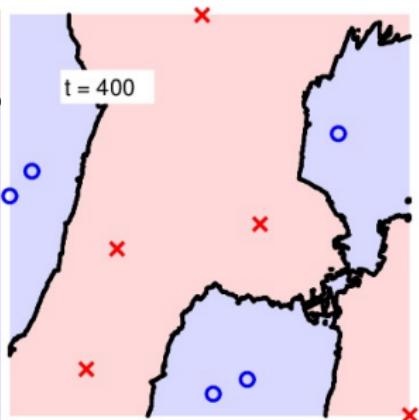
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

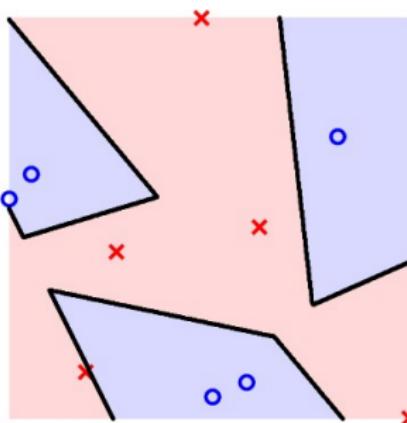


G with first t trees

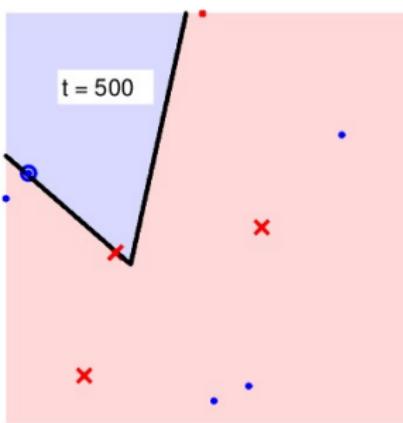


A Simple Data Set

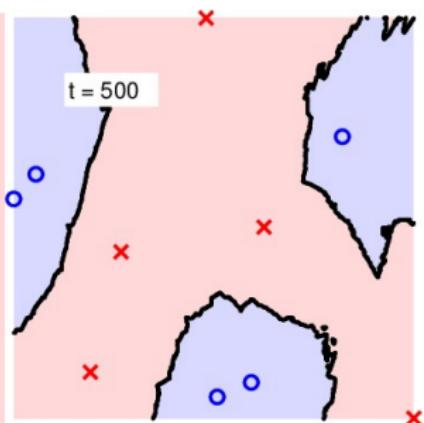
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

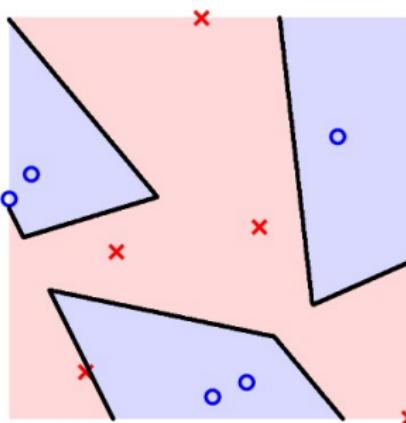


G with first t trees

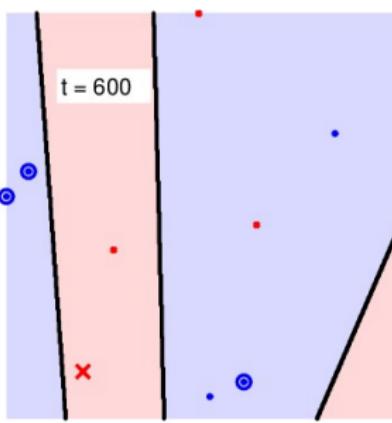


A Simple Data Set

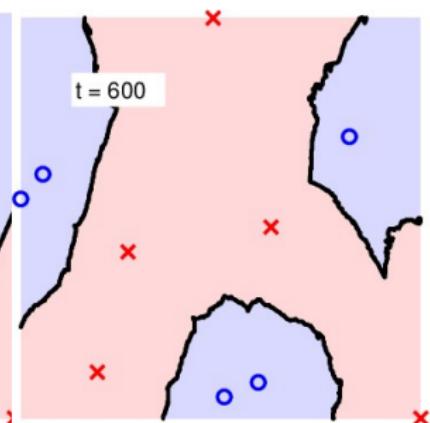
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

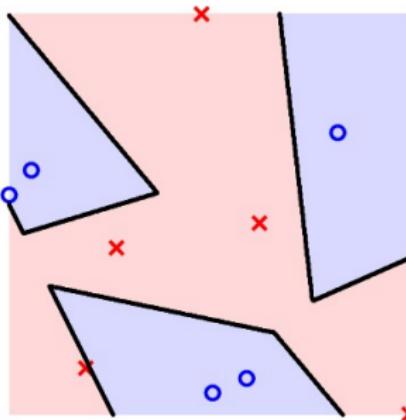


G with first t trees

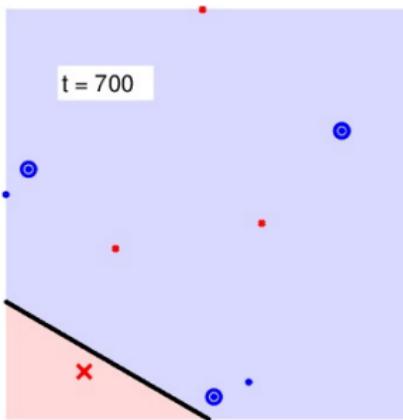


A Simple Data Set

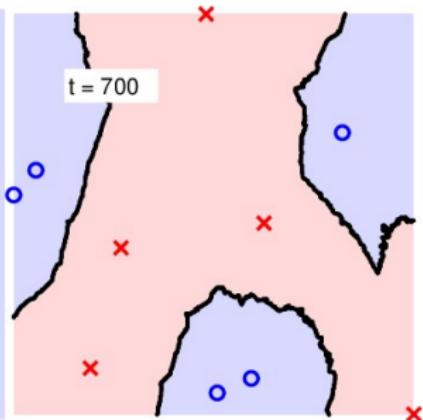
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

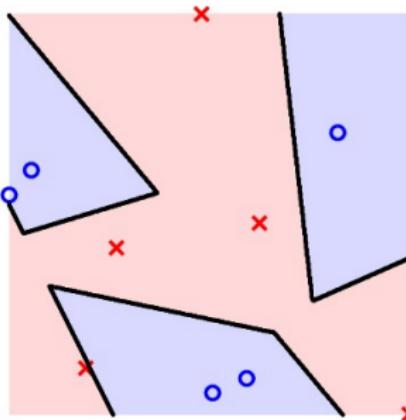


G with first t trees

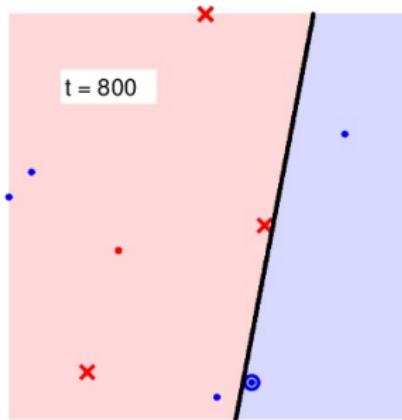


A Simple Data Set

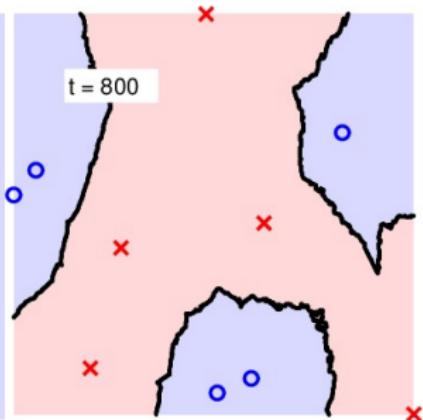
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

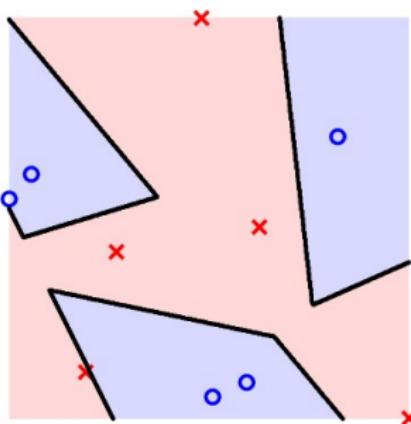


G with first t trees

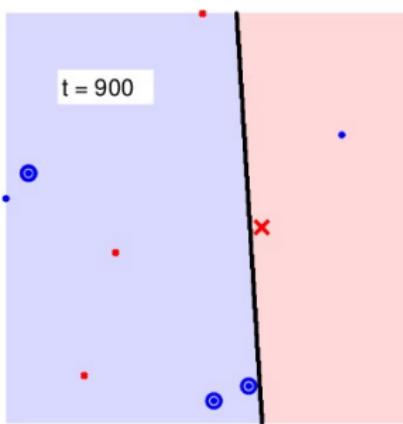


A Simple Data Set

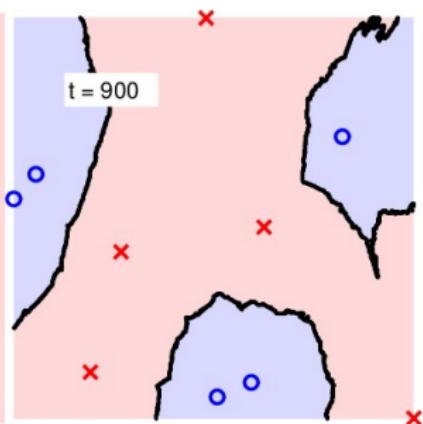
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$

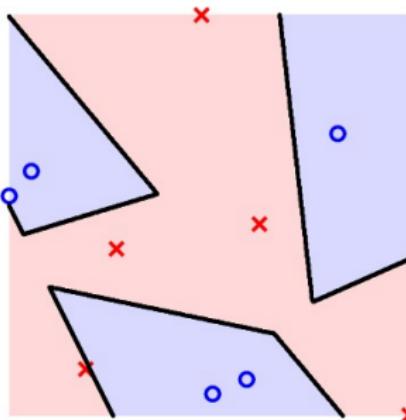


G with first t trees

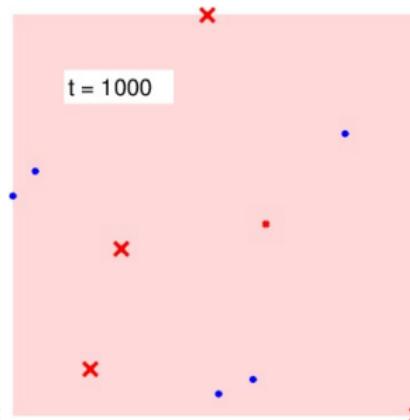


A Simple Data Set

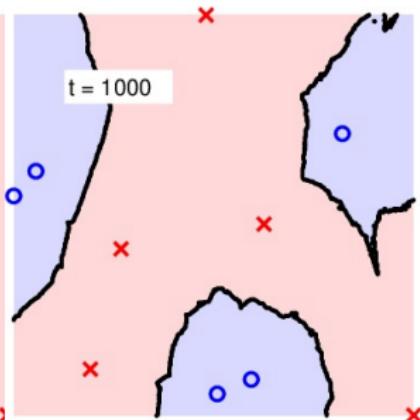
$g_{C\&RT}$
with random combination



$g_t (N' = N/2)$



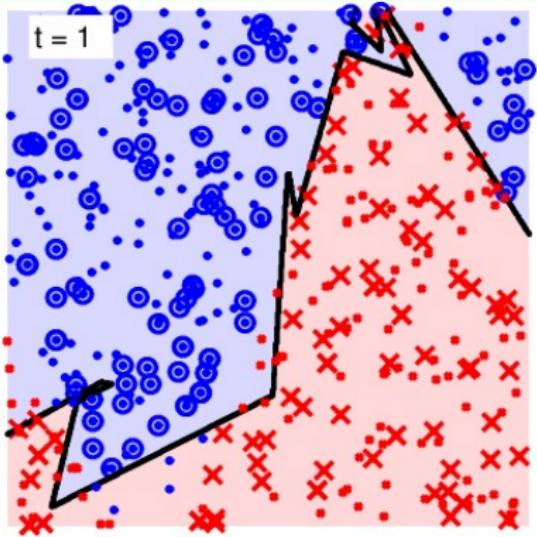
G with first t trees



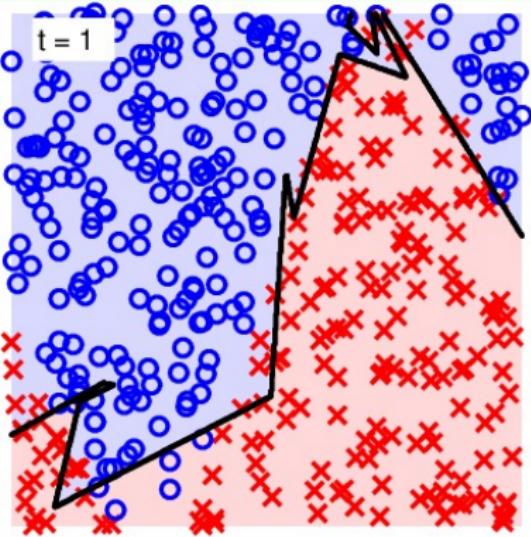
**'smooth' and large-margin-like boundary
with many trees**

A Complicated Data Set

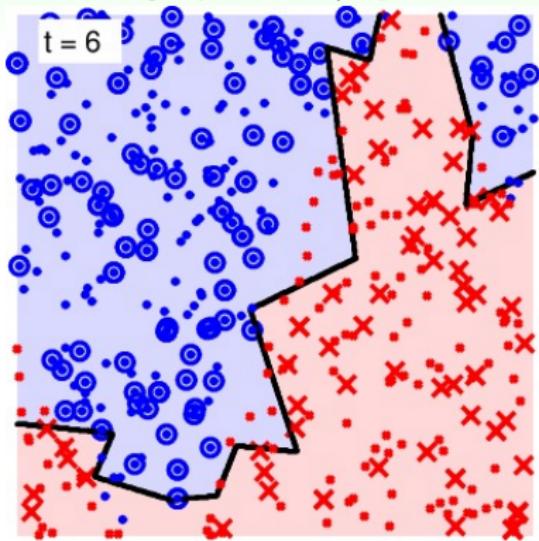
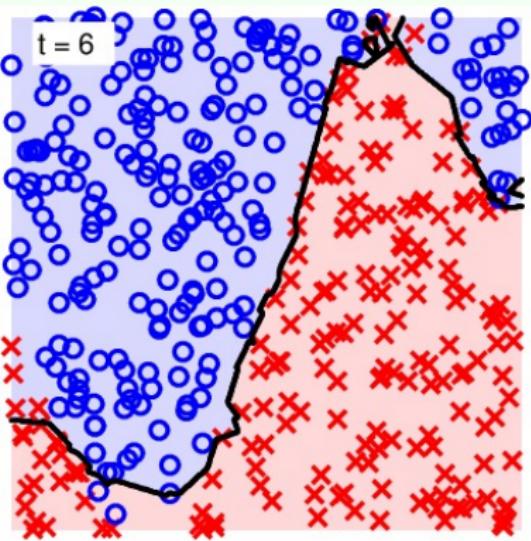
$g_t (N' = N/2)$



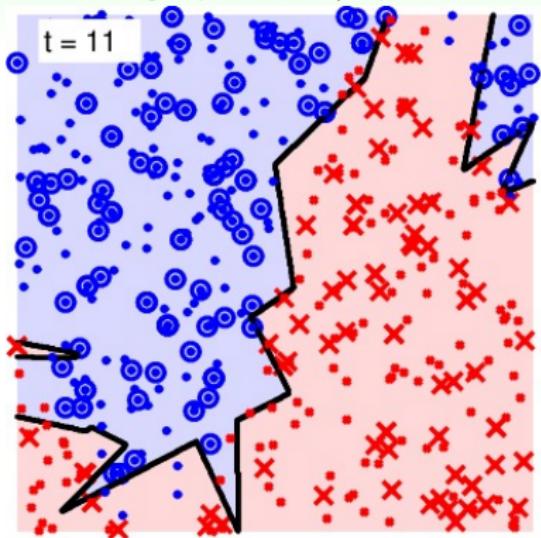
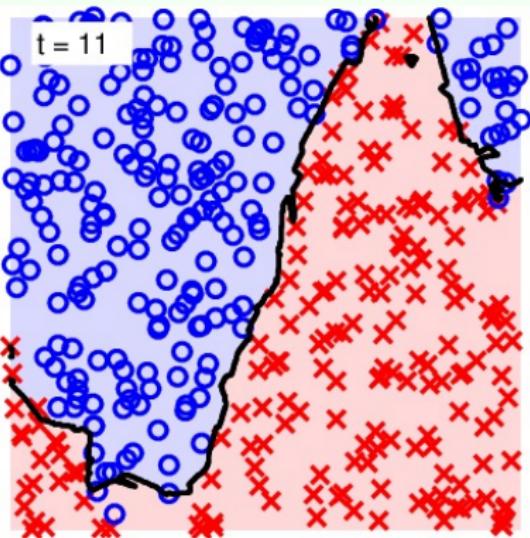
G with first t trees



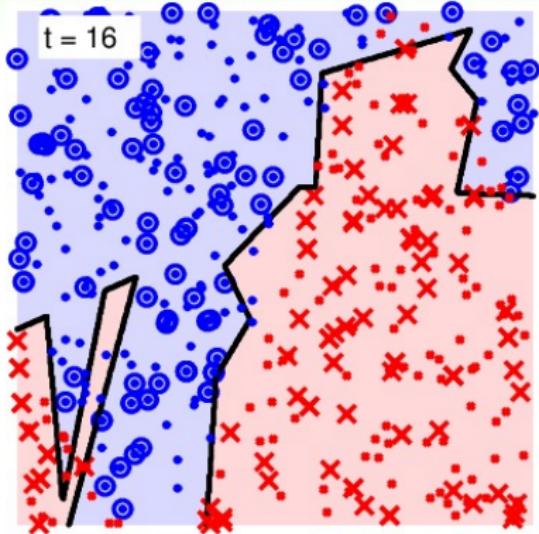
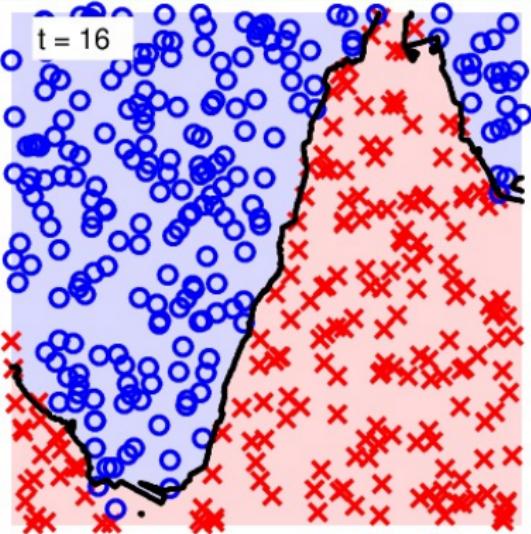
A Complicated Data Set

 $g_t (N' = N/2)$  G with first t trees

A Complicated Data Set

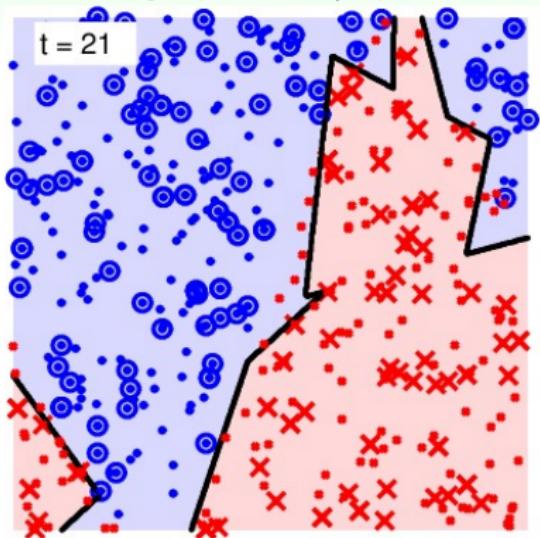
 $g_t (N' = N/2)$  G with first t trees

A Complicated Data Set

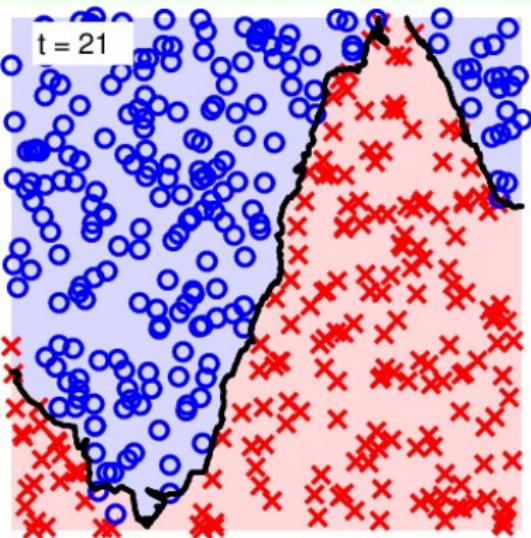
 $g_t (N' = N/2)$  G with first t trees

A Complicated Data Set

$g_t (N' = N/2)$



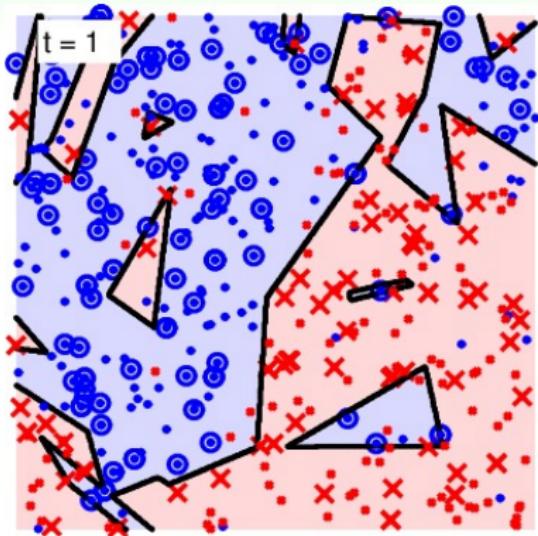
G with first t trees



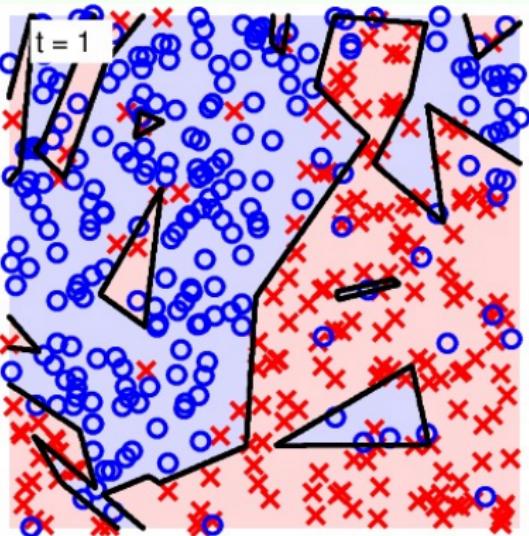
'easy yet robust' nonlinear model

A Complicated and Noisy Data Set

$g_t (N' = N/2)$

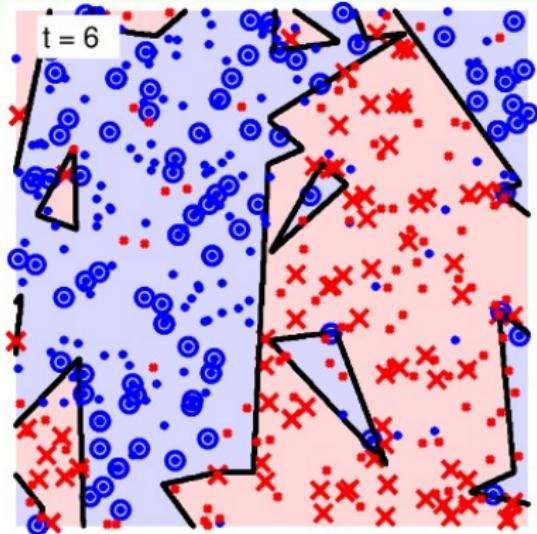


G with first t trees

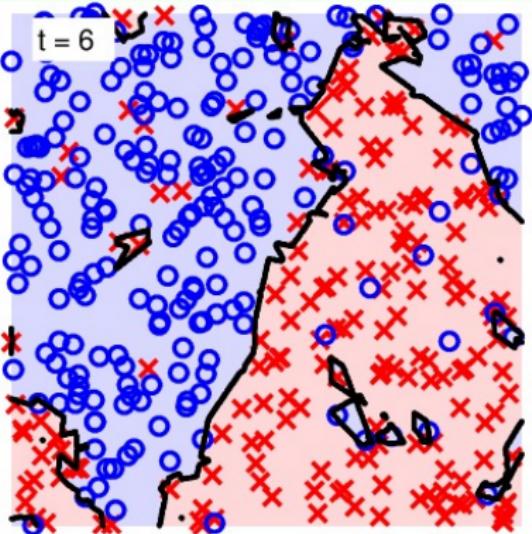


A Complicated and Noisy Data Set

g_t ($N' = N/2$)

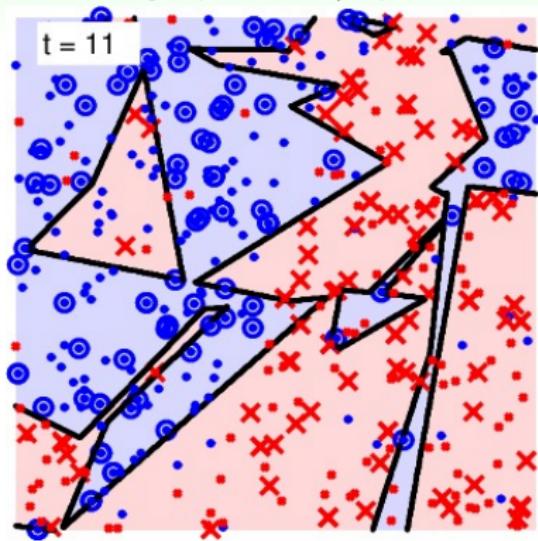


G with first t trees

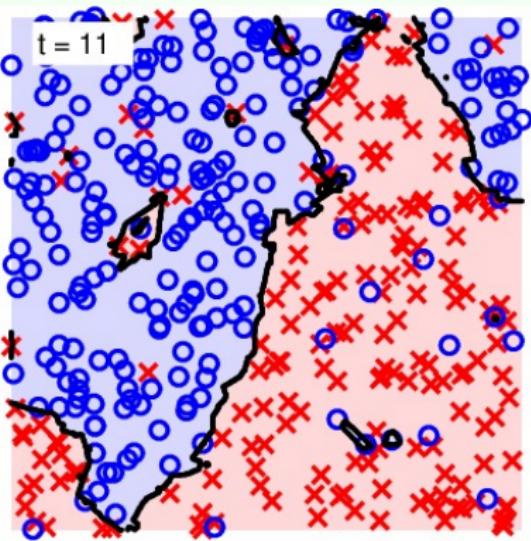


A Complicated and Noisy Data Set

g_t ($N' = N/2$)

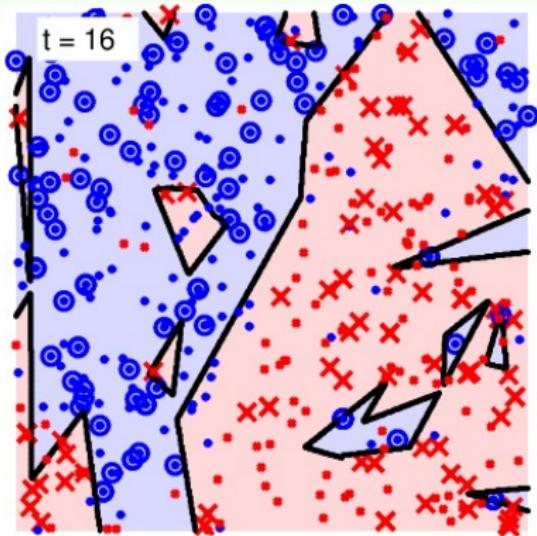


G with first t trees

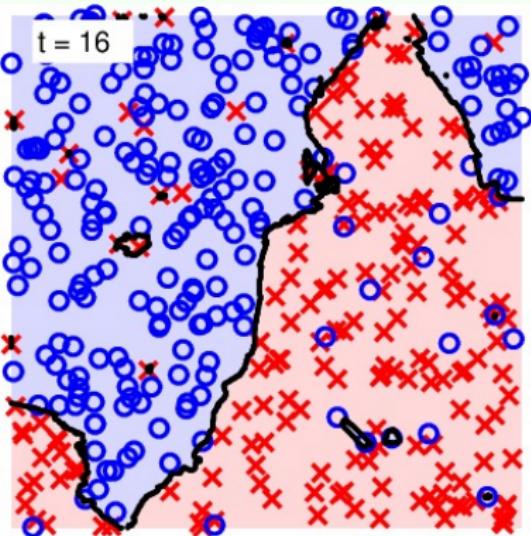


A Complicated and Noisy Data Set

g_t ($N' = N/2$)

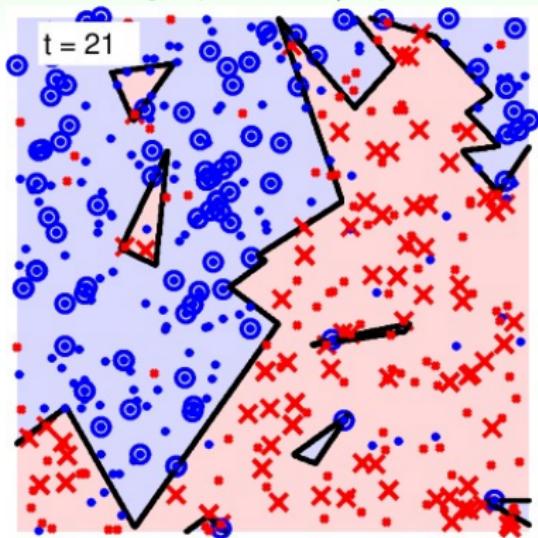


G with first t trees

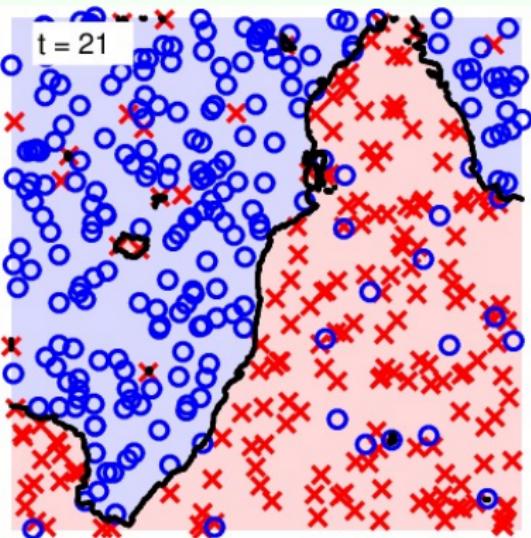


A Complicated and Noisy Data Set

$g_t (N' = N/2)$



G with first t trees



noise corrected by voting

How Many Trees Needed?

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

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

$$\begin{aligned} u_n^{(t+1)} &= \begin{cases} u_n^{(t)} \cdot \diamond_t & \text{if incorrect} \\ u_n^{(t)} / \diamond_t & \text{if correct} \end{cases} \\ &= u_n^{(t)} \cdot \diamond_t^{-y_n g_t(\mathbf{x}_n)} = u_n^{(t)} \cdot \exp(-y_n \alpha_t g_t(\mathbf{x}_n)) \end{aligned}$$

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

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

Voting Score and Margin

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

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

voting score

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

$y_n(\text{voting score})$ = signed & unnormalized **margin**

want $y_n(\text{voting score})$ **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 $s = \sum_{t=1}^T \alpha_t g_t(\mathbf{x}_n)$

- $\text{err}_{0/1}(s, y) = \llbracket ys \leq 0 \rrbracket$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$:
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 $s = \sum_{t=1}^T \alpha_t g_t(\mathbf{x}_n)$

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

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

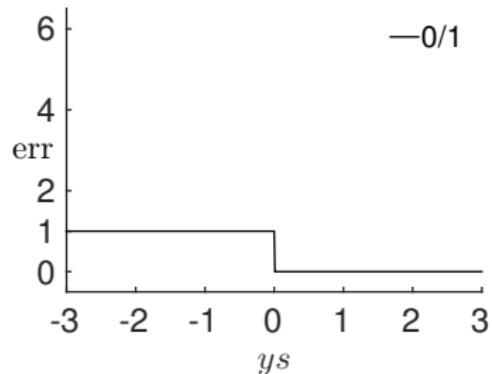
AdaBoost Error Function

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

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



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

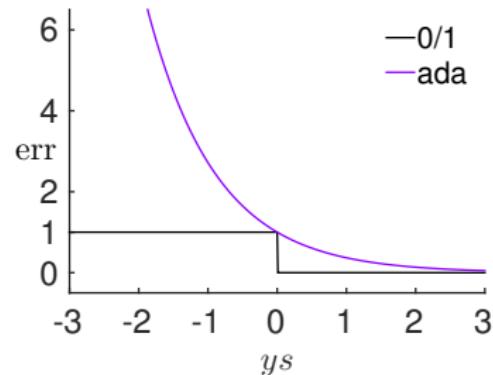
AdaBoost Error Function

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

- $\text{err}_{0/1}(s, y) = \llbracket ys \leq 0 \rrbracket$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$:
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

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

at iteration t , to find \mathbf{g}_t , solve

$$\begin{aligned} \min_{\mathbf{h}} \quad \hat{E}_{\text{ADA}} &= \frac{1}{N} \sum_{n=1}^N \exp \left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n) \right) \right) \\ &= \sum_{n=1}^N u_n^{(t)} \exp (-y_n \eta h(\mathbf{x}_n)) \\ &\stackrel{\text{taylor}}{\approx} \sum_{n=1}^N u_n^{(t)} (1 - y_n \eta h(\mathbf{x}_n)) = \sum_{n=1}^N u_n^{(t)} - \eta \sum_{n=1}^N u_n^{(t)} y_n h(\mathbf{x}_n) \end{aligned}$$

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

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 $\in \{-1, +1\}$:

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

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

A: good $g_t = h$ for ‘gradient descent’

Deciding Blending Weight as Optimization

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

after finding g_t , how about

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

- 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

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

$$\begin{aligned} \min_h \dots & \stackrel{\text{taylor}}{\approx} \min_h \quad \frac{1}{N} \sum_{n=1}^N \underbrace{\text{err}(s_n, y_n)}_{\text{constant}} + \frac{1}{N} \sum_{n=1}^N \eta h(\mathbf{x}_n) \left. \frac{\partial \text{err}(s, y_n)}{\partial s} \right|_{s=s_n} \\ &= \min_h \quad \text{constants} + \frac{\eta}{N} \sum_{n=1}^N h(\mathbf{x}_n) \cdot 2(s_n - y_n) \end{aligned}$$

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

Learning Hypothesis as Optimization

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

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

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

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

GradientBoost for regression:

find $g_t = h$ by regression with **residuals**

Deciding Blending Weight as Optimization

after finding $g_t = h$,

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

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

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

GradientBoost for regression: $\alpha_t = \text{optimal } \eta$
by g_t -transformed linear regression

Putting Everything Together

Gradient Boosted Decision Tree (GBDT)

$$s_1 = s_2 = \dots = s_N = 0$$

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

- 1 obtain g_t by $\mathcal{A}(\{(x_n, y_n - s_n)\})$ where \mathcal{A} is a (squared-error) regression algorithm

—**how about pruned C&RT?**

- 2 compute $\alpha_t = \text{OneVarLinearRegression}(\{(g_t(x_n), y_n - s_n)\})$
- 3 update $s_n \leftarrow s_n + \alpha_t g_t(x_n)$

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

simple

voting/averaging of g_t

non-uniform

linear model on
 g_t -transformed inputs

conditional

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

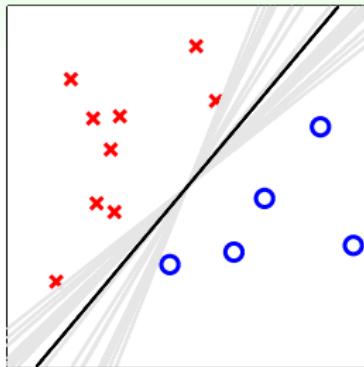
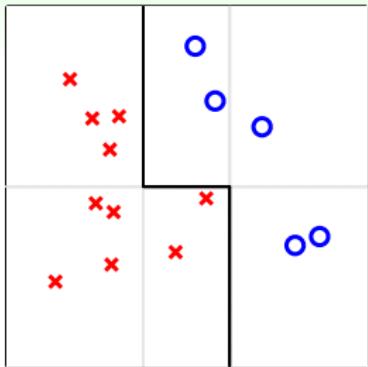
GradientBoost

GBDT

GradientBoost
+ ‘weak’ DTree

all three frequently used in practice

Specialty of Aggregation Models



cure underfitting

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

cure overfitting

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

proper aggregation (a.k.a. 'ensemble')
⇒ **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