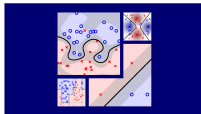


Machine Learning Techniques (機器學習技法)



Lecture 9: Decision Tree

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 Embedding Numerous Features: Kernel Models
- 2 Combining Predictive Features: Aggregation Models

Lecture 8: Adaptive Boosting

optimal re-weighting for diverse hypotheses
and adaptive **linear aggregation** to
boost 'weak' algorithms

Lecture 9: Decision Tree

- Decision Tree Hypothesis
- Decision Tree Algorithm
- Decision Tree Heuristics in C&RT
- Decision Tree in Action

- 3 Distilling Implicit Features: Extraction Models

What We Have Done

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

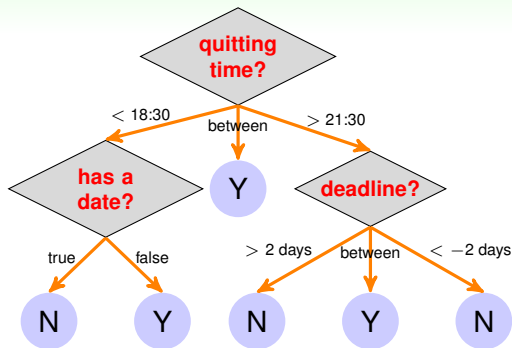
aggregation type	blending	learning
uniform	voting/averaging	Bagging
non-uniform	linear	AdaBoost
conditional	stacking	Decision Tree

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

Decision Tree for Watching MOOC Lectures

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

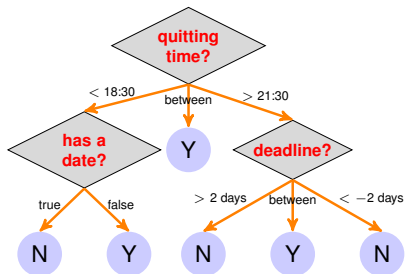
- **base hypothesis** $g_t(\mathbf{x})$:
leaf at end of path t ,
a **constant** here
- **condition** $q_t(\mathbf{x})$:
[[is \mathbf{x} on path t ?]]
- usually with **simple internal nodes**



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

Recursive View of Decision Tree

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



Recursive View

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

- $G(\mathbf{x})$: 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

Fun Time

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;
}
```

What is the output of the tree for $(\text{income}, \text{debt}) = (98765, 56789)$?

① true

② false

③ 98765

④ 56789

Fun Time

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;
}
```

What is the output of the tree for $(\text{income}, \text{debt}) = (98765, 56789)$?

① true

③ 98765

② false

④ 56789

Reference Answer: ②

You can simply trace the code. The tree expresses a complicated boolean condition $[\text{income} > 100000 \text{ or } \text{debt} \leq 50000]$.

A Basic Decision Tree Algorithm

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

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

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 **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**

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

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:

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

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

for classification

- Gini index:

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

—**all** k considered together

- classification error:

$$1 - \max_{1 \leq k \leq K} \frac{\sum_{n=1}^N \mathbb{I}[y_n = k]}{N}$$

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

Fun Time

For the Gini index, $1 - \sum_{k=1}^K \left(\frac{\sum_{n=1}^N \mathbb{I}[y_n=k]}{N} \right)^2$. 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 $2\mu^2(1 - \mu)^2$

Fun Time

For the Gini index, $1 - \sum_{k=1}^K \left(\frac{\sum_{n=1}^N \mathbb{I}[y_n=k]}{N} \right)^2$. 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

Simplify $1 - (\mu^2 + (1 - \mu)^2)$ and the answer should pop up.

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**

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

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 $G(\mathbf{x}) = \sum_{c=1}^2 \mathbb{I}[b(\mathbf{x}) = c] G_c(\mathbf{x})$

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

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

systematic **choice of λ** ? **validation**

Branching on Categorical Features

numerical features

blood pressure:

130, 98, 115, 147, 120

categorical features

major symptom:

fever, pain, tired, sweaty

branching for numerical decision stump

$$b(\mathbf{x}) = \llbracket x_i \leq \theta \rrbracket + 1$$

with $\theta \in \mathbb{R}$

branching for categorical decision subset

$$b(\mathbf{x}) = \llbracket x_i \in S \rrbracket + 1$$

with $S \subset \{1, 2, \dots, K\}$

C&RT (& general decision trees):
handles **categorical features easily**

Missing Features by Surrogate Branch

possible $b(\mathbf{x}) = \llbracket \text{weight} \leq 50\text{kg} \rrbracket$

if **weight** missing during prediction:

- what would human do?
 - go get **weight**
 - or, use **threshold on height** instead, because $\text{threshold on height} \approx \text{threshold on weight}$
- **surrogate branch**:
 - maintain **surrogate branch** $b_1(\mathbf{x}), b_2(\mathbf{x}), \dots \approx \text{best branch } b(\mathbf{x})$ during training
 - allow **missing feature for } b(\mathbf{x}) during prediction by using **surrogate** instead**

C&RT: handles **missing features easily**

Fun Time

For a categorical branching criteria $b(\mathbf{x}) = \mathbb{I}[x_i \in S] + 1$ 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

Fun Time

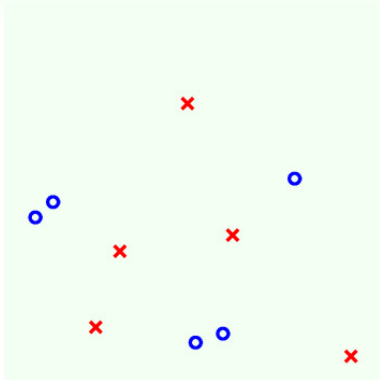
For a categorical branching criteria $b(\mathbf{x}) = \llbracket x_i \in S \rrbracket + 1$ with $S = \{1, 6\}$. Which of the following is the explanation of the criteria?

- 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

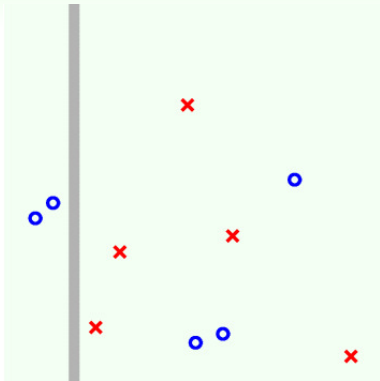
Reference Answer: ②

Note that ' $\in S$ ' is an 'or'-style condition on the elements of S in human language.

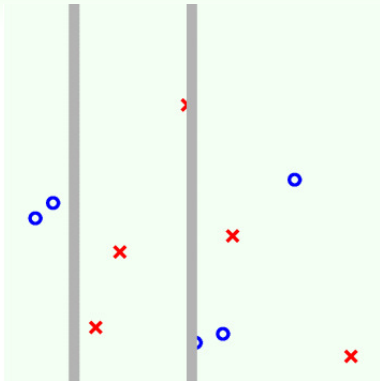
A Simple Data Set



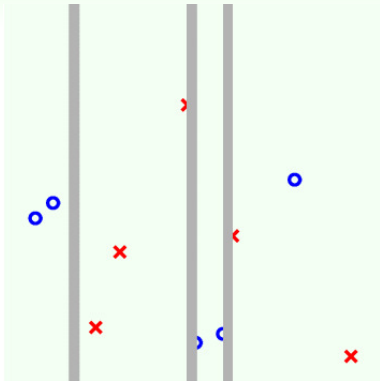
A Simple Data Set



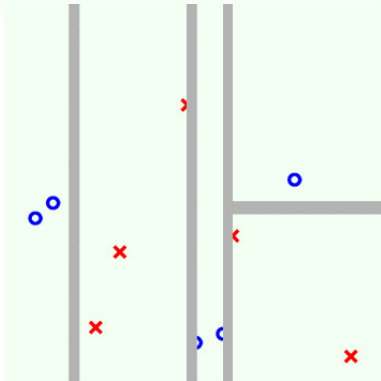
A Simple Data Set



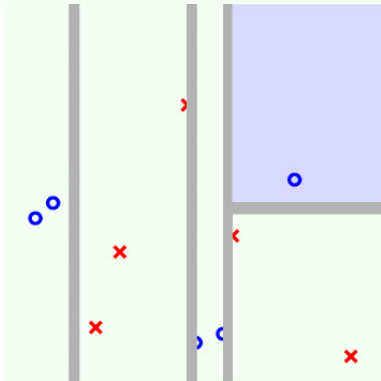
A Simple Data Set



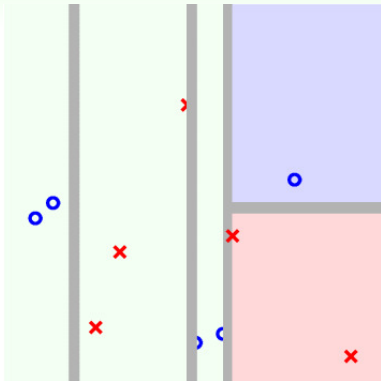
A Simple Data Set



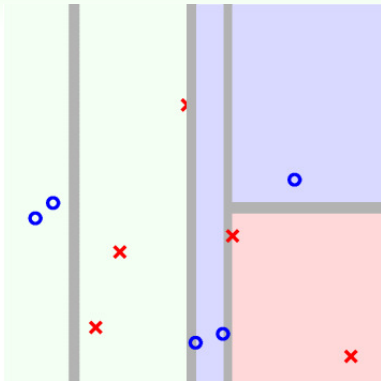
A Simple Data Set



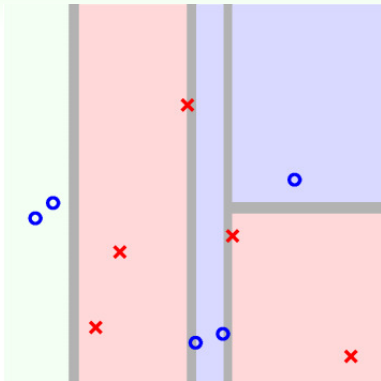
A Simple Data Set



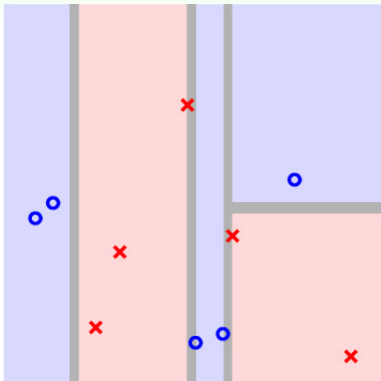
A Simple Data Set



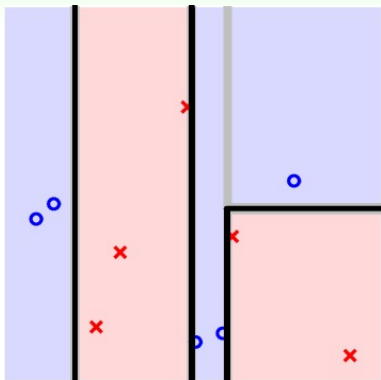
A Simple Data Set



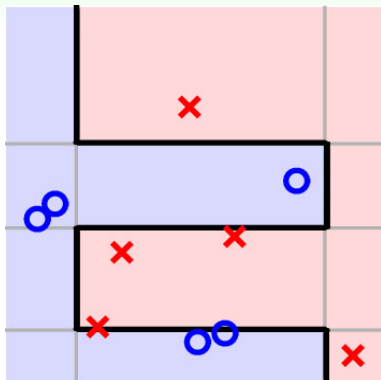
A Simple Data Set



A Simple Data Set



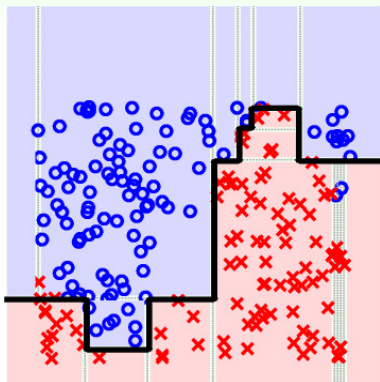
C&RT



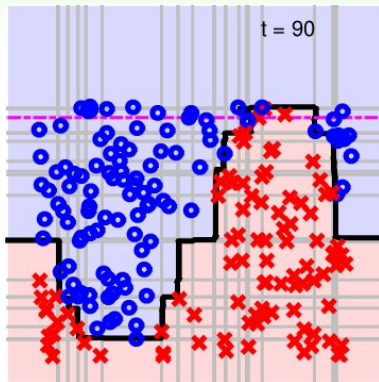
AdaBoost-Stump

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
- **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**

Fun Time

Which of the following is **not** a specialty of C&RT without pruning?

- ① handles missing features easily
- ② produces explainable hypotheses
- ③ achieves low E_{in}
- ④ achieves low E_{out}

Fun Time

Which of the following is **not** a specialty of C&RT without pruning?

- ① handles missing features easily
- ② produces explainable hypotheses
- ③ achieves low E_{in}
- ④ achieves low E_{out}

Reference Answer: ④

The first two choices are easy; the third comes from the fact that fully grown C&RT greedy minimizes E_{in} (almost always to 0). But as you may imagine, overfitting may happen and E_{out} may not always be low.

Summary

- 1 Embedding Numerous Features: Kernel Models
- 2 Combining Predictive Features: Aggregation Models

Lecture 9: Decision Tree

- Decision Tree Hypothesis
express path-conditional aggregation
- Decision Tree Algorithm
recursive branching until termination to base
- Decision Tree Heuristics in C&RT
pruning, categorical branching, surrogate
- Decision Tree in Action
explainable and efficient

- **next: aggregation of aggregation?!**

- 3 Distilling Implicit Features: Extraction Models