10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Decision Trees

Matt Gormley & Geoff Gordon

Lecture 3
Sep. 3, 2025

Q&A

Q: How do these In-Class Polls work?

A:

- Sign into **Google Form (**click [Poll] link on Schedule page http://mlcourse.org/schedule.html) using **Andrew Email**
- Answer **during lecture for full credit,** or within 24 hours for half credit
- Avoid the **toxic option** which gives negative points!
- 8 "free poll points" but can't use more than 3 free polls consecutively. All the questions for one lecture are worth 1 point total.

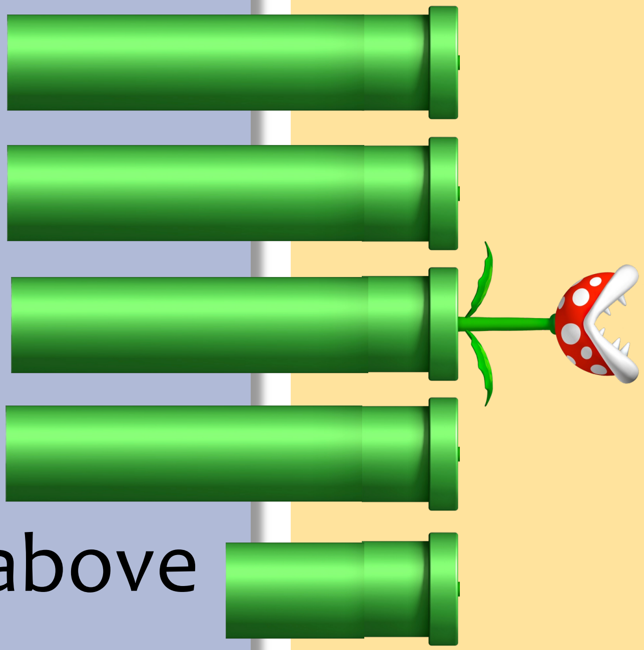
broken

Latest Poll link: http://poll.mlcourse.org

First In-Class Poll

Question: *Which of the following did you bring to class today? Select all that apply.*

A. Smartphone
B. Flip phone
TOXIC C. Pay phone
D. No phone
E. None of the above



do not pick

Answer:

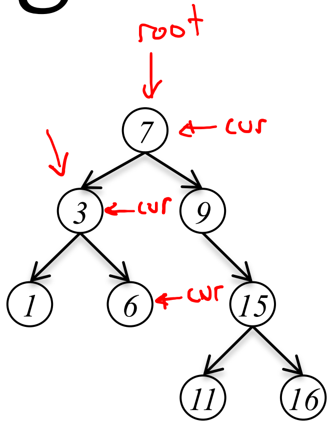
Reminders

- Homework 1: Background
  - Out: Mon, Aug 25
  - Due: Wed, Sep 3 at 11:59pm
  - unique policy for this assignment: we will grant (essentially) any and all extension requests
- Homework 2: Decision Trees
  - Out: Wed, Sep. 3
  - Due: Mon, Sep. 15 at 11:59pm

MAKING PREDICTIONS WITH A DECISION TREES

Background: Recursion

- *Def*: a **binary search tree** (BST) consists of nodes, where each node:
  - has a value, v
  - up to 2 children
  - all its left descendants have values less than v, and its right descendants have values greater than v
- We like BSTs because they permit search in O(log(n)) time, assuming n nodes in the tree



Node Data Structure

```
class Node:
        int value
        Node left
        Node right
```

contains (root, 6) = true

Iterative Search

```
def contains(node, key):
        cur = node
        while true:
                if key < cur.value & cur.left != null:
                        cur = cur.left
                else if cur.value < key & cur.right != null:
                        cur = cur.right
                else:
                        break
        return key == cur.value
```

contains (root, 6) → contains (Node(3), 6) → contains (Node(6), 6) → return true

Recursive Search

```
def contains(node, key):
        if key < node.value & node.left != null:
                return contains(node.left, key)
        else if node.value < key & node.right != null:
                return contains(node.right, key)
        else:
                return key == node.value
```

Algorithms for Classification

Algorithm 3 **decision stump**: based on a single feature, x_d, predict the most common label in the training dataset among all data points that have the same value for x_d

predictions	y **allergic?**	x_1 **hives?**	x_2 **sneezing?**	x_3 **red eye?**	x_4 **has cat?**
-	-	Y	N	N	N
+	-	N	Y	N	N
+	+	Y	Y	N	N
-	-	Y	N	Y	Y
+	+	N	Y	Y	N

But why use just one feature…

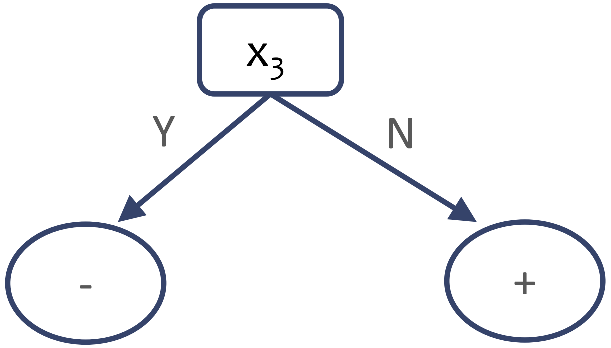
Nonzero training error, but perhaps still better than the memorizer

Example decision stump:

$$h(\mathbf{x}) = \begin{cases} + \text{ if sneezing} = \text{Y} \\ - \text{ otherwise} \end{cases}$$

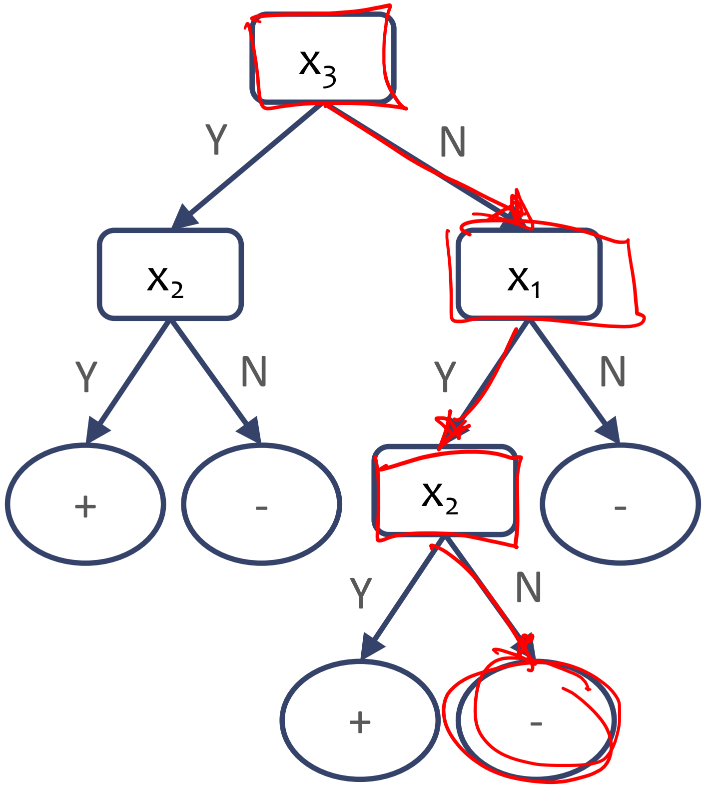
From Decision Stump to Decision Tree

	y	x_1	x_2	x_3	x_4
predic-tions	**allergic?**	**hives?**	**sneezing?**	**red eye?**	**has cat?**
-	-	Y	N	N	N
-	-	N	Y	N	N
+	+	Y	Y	N	N
-	-	Y	N	Y	Y
+	+	N	Y	Y	N



From Decision Stump to Decision Tree

predic-tions	y allergic?	x_1 hives?	x_2 sneezing?	x_3 red eye?	x_4 has cat?
-	-	Y	N	N	N
-	-	N	Y	N	N
+	+	Y	Y	N	N
-	-	Y	N	Y	Y
+	+	N	Y	Y	N



Decision Tree: In-Class Activity

1. Group 1: Answer the questions to determine which leaf node corresponds to your feature values
2. Group 2: (part 1)
    a) Take a blue sticky note if you prefer dogs to cats; otherwise, take a red sticky note
    b) Answer the questions to determine which leaf node corresponds to your feature values and place your sticky note there
3. Group 2: (part 2)
    a) Answer the new question to determine which new leaf node to move your sticky note to

Decision Tree: Prediction

branches = { "cold" : Node( )
"hot" : Node( )}

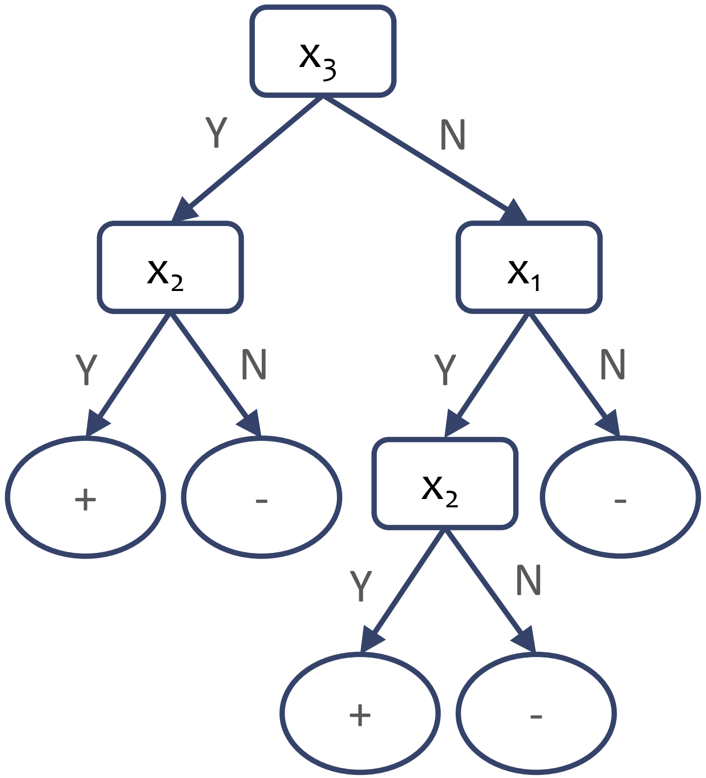
```
def h(x'):
    Let current node = root
    while(true):
        if current node is internal (non-leaf):
            Let m = attribute associated with current node
            Go down branch labeled with value x'_m
        if current node is a leaf:
            return label y stored at that leaf
```

```
class Node:
    str type    // "leaf" or "internal"
    𝒴 vote      // label for leaf node
    {} branches // map from feature
                // values to Node objects
    int m // feature for internal node
```

Decision Tree: Prediction

<u>Algorithm 4</u> **decision tree:** recursively walk from root to a leaf, following the attribute values labeled on the branches, and return the label at the leaf

	y	x_1	x_2	x_3	x_4
predic-tions	**allergic?**	**hives?**	**sneezing?**	**red eye?**	**has cat?**
-	-	Y	N	N	N
-	-	N	Y	N	N
+	+	Y	Y	N	N
-	-	Y	N	Y	Y
+	+	N	Y	Y	N



Zero training error!

Decision Tree: Prediction (Iterative)

def h(**x**'):
    Let *current node* = root
    while(true):
        if *current node* is internal (non-leaf):
            Let m = attribute associated with current node
            Go down branch labeled with value x'_m
        if *current node* is a leaf:
            return label y stored at that leaf

Question: The original h(x') pseudocode is an iterative implementation. Can you implement h(x') recursively?

class Node:
    str type   // "leaf" or "internal"
    $\mathcal{Y}$ vote   // label for leaf node
    {} branches // map from feature
                // values to Node objects
    int m // feature for internal node

Decision Tree: Prediction (Recursive)

```
def h(x⃗'):
    return h_recurse(root, x⃗)
def h_recurse(node, x⃗):
    if node.type == "leaf":
        return node.vote
    else:
        m = node.m
        next = branches[x_m]
        return h_recurse(next, x⃗)
```

Question: The original h(x’) pseudocode is an iterative implementation. Can you implement h(x’) recursively?

```
class Node:
    str type   // "leaf" or "internal"
    𝒴 vote   // label for leaf node
    {} branches // map from feature
                // values to Node objects
    int m // feature for internal node
```

Decision Tree Example

Learned from medical records of 1000 women (Sims et al., 2000)

Negative examples are C-sections

```
[833+,167-] .83+ .17-
Fetal_Presentation = 1: [822+,116-] .88+ .12-
| Previous_Csection = 0: [767+,81-] .90+ .10-
| | Primiparous = 0: [399+,13-] .97+ .03-
| | Primiparous = 1: [368+,68-] .84+ .16-
| | | Fetal_Distress = 0: [334+,47-] .88+ .12-
| | | Fetal_Distress = 1: [34+,21-] .62+ .38-
| Previous_Csection = 1: [55+,35-] .61+ .39-
Fetal_Presentation = 2: [3+,29-] .11+ .89-
Fetal_Presentation = 3: [8+,22-] .27+ .73-
```

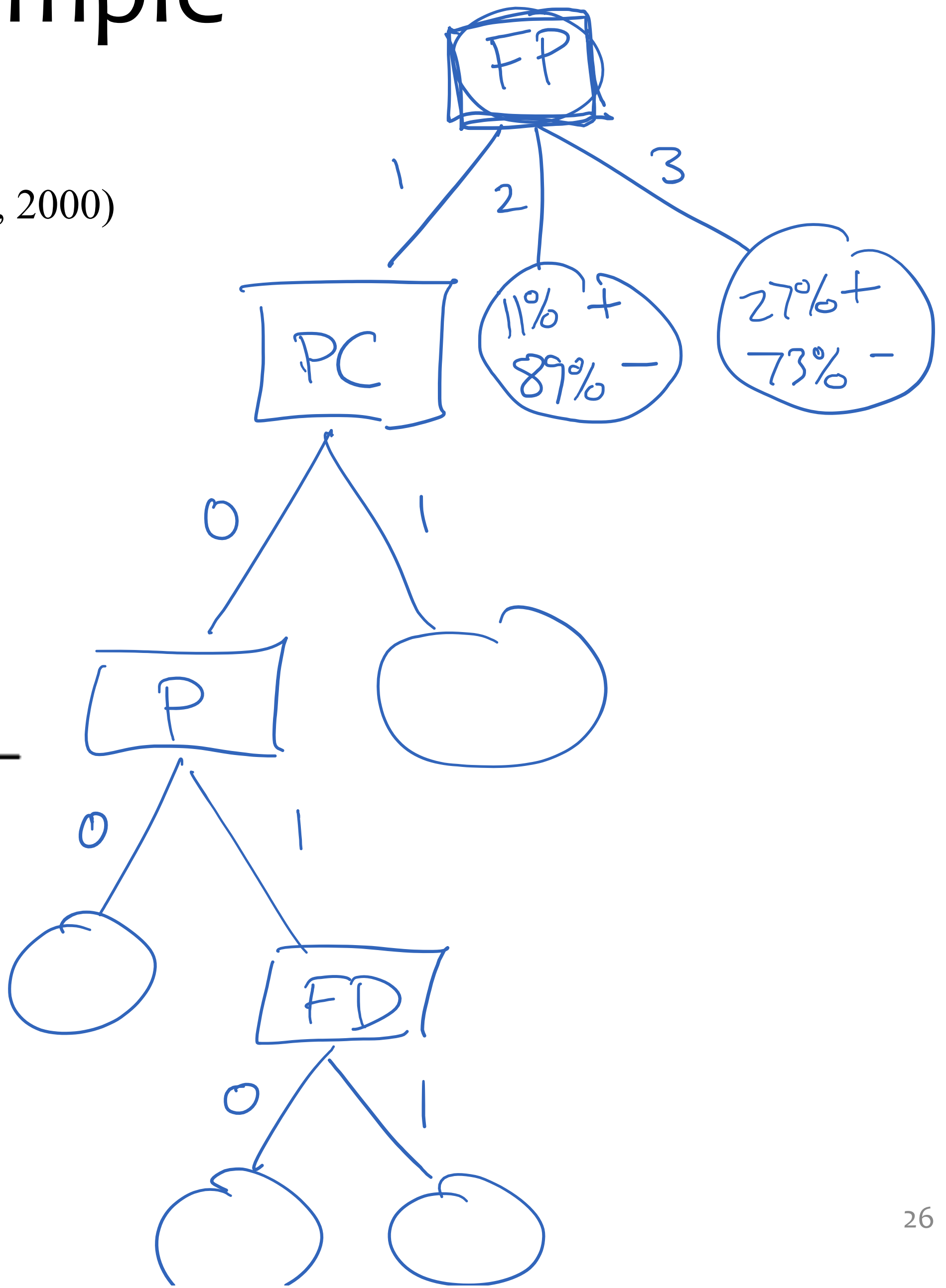


Figure from Tom Mitchell

LEARNING A DECISION TREE

Decision Tree Learning

- *Definition*: a **splitting criterion** is a function that measures the effectiveness of splitting on a particular attribute
- Our decision tree learner **selects the "best" attribute** as the one that maximizes the splitting criterion
- Lots of options for a splitting criterion:
  – error rate (minimize)
  – accuracy = 1 - error rate (maximize)
  – Mutual information (maximize)
  – Gini gain (maximize)

Decision Tree Learning

```
def train(D):
    root = train_recurse(D)
    store root
```

```
def train_recurse(D'):
    Let p = new Node()
    [Base Case] If (a) D' is empty
                   (b) D' has all labels identical
                   (c) for each feature, all values in D' are identical
                   (d) depth of node ≥ max_depth
        p.type = "leaf"
        p.vote = majority_vote(D')
        return p
    [Recursive Case] Else:
        p.type = "internal"
        p.m = best feature according to splitting criterion on D'
            = argmax_{m ∈ {1,...,M}} splitting_criterion(D', m)
        for each value v of feature p.m:
            D_{x_m=v} = {(x⃗, y) ∈ D' : x_m = v}   ← partition of D'
            child_v = train_recurse(D_{x_m=v})     ← recursion
            p.branches[v] = child_v                ← add a branch w/ label v
        return p
```

Decision Tree Learning Example

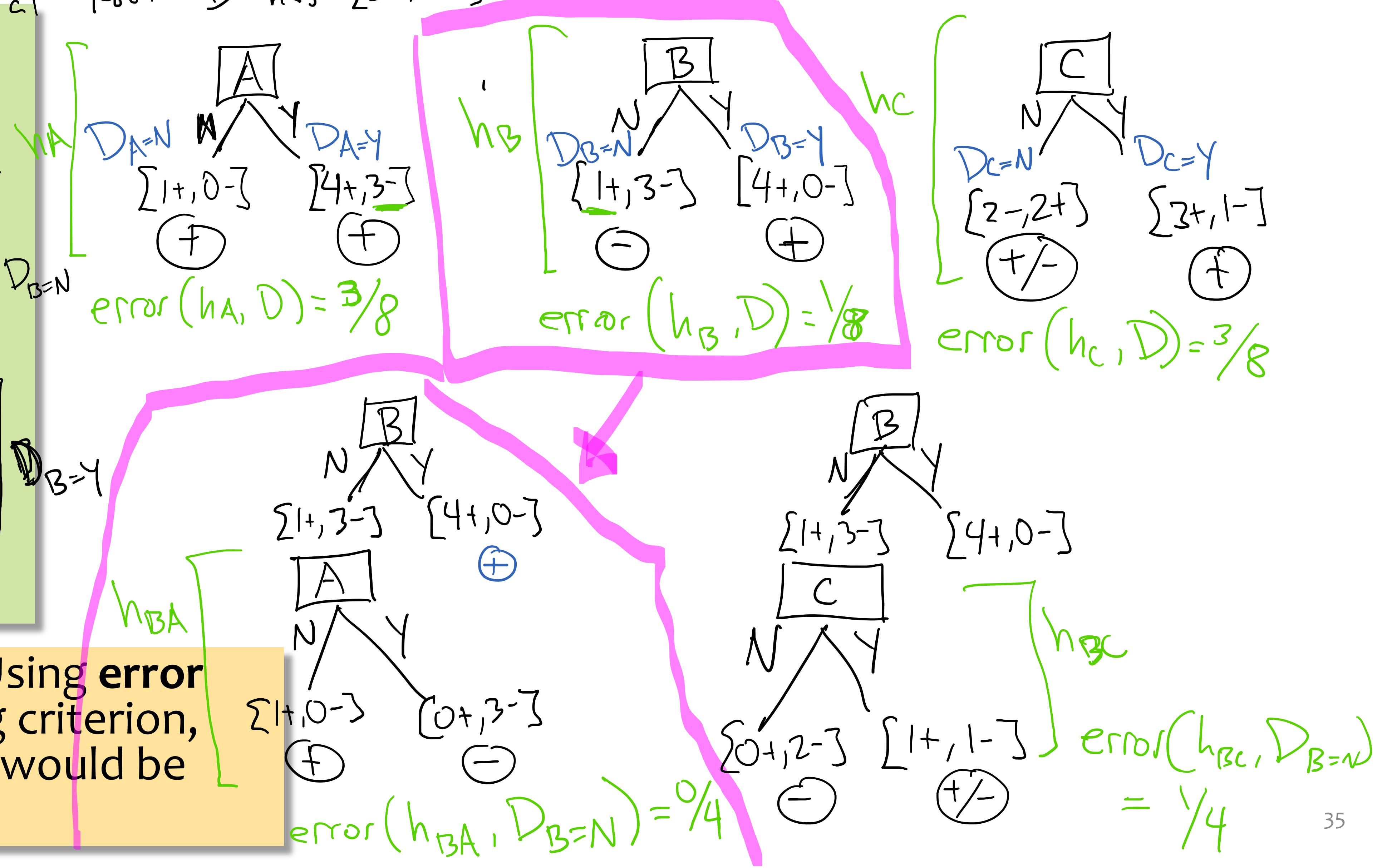
at root D has [5+, 3-]

Dataset:

y	~~x_1~~ A	~~x_2~~ B	~~x_3~~ C
-	Y	N	N
-	Y	N	Y
-	Y	N	N
+	N	N	Y
+	Y	Y	N
+	Y	Y	Y
+	Y	Y	N
+	Y	Y	Y

$D_{B=N}$ (first four rows)

$D_{B=Y}$ (last four rows)



In-Class Exercise: Using **error rate** as the splitting criterion, what decision tree would be learned?

Recursive Training for Decision Trees

- def train(dataset D'):
  - Let p = new Node()
  - ***Base Case:*** If (1) all labels $y^{(i)}$ in D' are identical (2) D' is empty (3) for each attribute, all values are identical
    - p.type = Leaf // The node p is a leaf node
    - p.label = majority_vote(D') // Store the label
    - return p
  - ***Recursive Step:*** Otherwise
    - Make an internal node
      - p.type = Internal // The node p is an internal node
    - Pick the *best* attribute X_m according to splitting criterion
      - p.attr = argmax_m splitting_criterion(D', X_m) // Store the attribute on which to split
    - For each value v of attribute X_m:
      - $D_{X_m = v}$ = {($\mathbf{x}$,y) in D' : x_m = v} // Select a partition of the data
      - child_v = train($D_{X_m = v}$) // Recursively build the child
      - p.branches[v] = child_v // Create a branch with label v
      - return p

SPLITTING CRITERION: ERROR RATE

Decision Tree Learning Example

Dataset:

Label Y, Features A and B

Y	A	B
-	1	0
-	1	0
+	1	0
+	1	0
+	1	1
+	1	1
+	1	1
+	1	1

Poll Question 2

Which attribute would **error rate** select for the next split?

1. A 12%
2. B 8%
3. A or B (tie) 79%
4. Neither ← toxic

Decision Tree Learning Example

Dataset:

Label Y, Features A and B

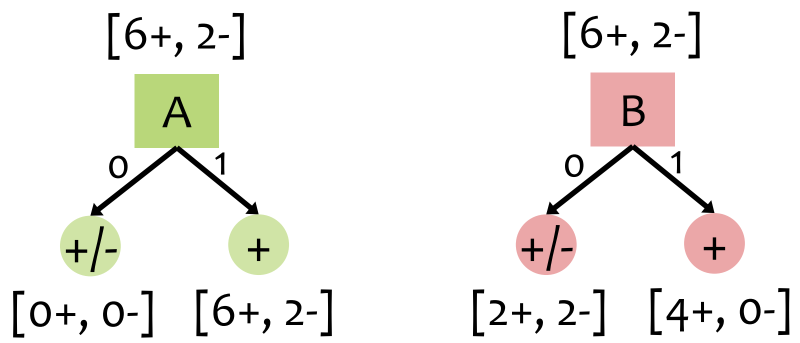
Y	A	B
-	1	0
-	1	0
+	1	0
+	1	0
+	1	1
+	1	1
+	1	1
+	1	1

Decision Tree Learning Example

Dataset:

Label Y, Features A and B

Y	A	B
-	1	0
-	1	0
+	1	0
+	1	0
+	1	1
+	1	1
+	1	1
+	1	1



Error Rate

$\text{error}(h_A, D) = 2/8$

$\text{error}(h_B, D) = 2/8$

error rate treats attributes A and B as equally good

Decision Tree Learning

- *Definition*: a **splitting criterion** is a function that measures the effectiveness of splitting on a particular attribute
- Our decision tree learner **selects the “best” attribute** as the one that maximizes the splitting criterion
- Lots of options for a splitting criterion:
  - error rate (minimize)
  - accuracy = 1 - error rate (maximize)
  - Mutual information (maximize)
  - Gini gain (maximize)

SPLITTING CRITERION: MUTUAL INFORMATION

r.v. over say labels

Entropy

- The **entropy** of a *random variable* describes the uncertainty of its outcome: the higher the entropy, the less certain we are about what the outcome will be.

$$H(X) = -\sum_{v \in V(X)} P(X = v) \log_2\big(P(X = v)\big)$$

where X is a (discrete) random variable

$V(X)$ is the set of possible values X can take on

Entropy

S is a set of labels

- The **entropy** of a *set* describes how uniform or pure it is: the higher the entropy, the more impure or "mixed-up" the set is

P(X=v)

$$H(S) = -\sum_{v \in V(S)} \frac{|S_v|}{|S|} \log_2\left(\frac{|S_v|}{|S|}\right)$$

\# elements in S_v

\# elements in S

where S is a collection of values,

$V(S)$ is the set of unique values in S

S_v is the collection of elements in S with value v

- If all the elements in S are the same, two possible values then

$$H(S) = -\left(\frac{9}{9}\log_2\frac{9}{9} + \frac{0}{9}\log_2\frac{0}{9}\right) = 0$$

$$0 + 0$$

Entropy

- The **entropy** of a *set* describes how uniform or pure it is: the higher the entropy, the more impure or "mixed-up" the set is

$$H(S) = -\sum_{v \in V(S)} \frac{|S_v|}{|S|} \log_2 \left(\frac{|S_v|}{|S|} \right)$$

where S is a collection of values,

$V(S)$ is the set of unique values in S

S_v is the collection of elements in S with value v

- If S is split fifty-fifty between two values, then

$$H(S) = -\left(\frac{5}{10} \log \frac{5}{10} + \frac{5}{10} \log \frac{5}{10} \right) = 1$$

Y label

X feature candidate

Mutual Information

- The **mutual information** between *two random variables* describes how much clarity knowing the value of one random variables provides about the other

$$I(Y;X) = H(Y) - H(Y|X)$$

$$= H(Y) - \sum_{v \in V(X)} P(X = v)H(Y|X = v)$$

where X and Y are random variables

$V(X)$ is the set of possible values X can take on

$H(Y|X = v)$ is the conditional entropy of Y given $X = v$

Slide from Henry Chai

Mutual Information

- The **mutual information** between *a feature and the label* describes how much clarity knowing the feature provides about the label

weighted avg of entropies at children

$$I(y; x_d) = H(y) - H(y|x_d)$$

entropy at parent

$$= H(y) - \overbrace{\sum_{v \in V(x_d)} f_v * \underbrace{H(Y_{x_d=v})}_{\text{entropy of each child}}}$$

where x_d is a feature and y is the set of all labels

$V(x_d)$ is the set of possible values x_d can take on

f_v is the fraction of data points where $x_d = v$

$Y_{x_d=v}$ is the set of all labels where $x_d = v$

Decision Tree Learning Example

Dataset:

Label Y, Features A and B

Y	A	B
-	1	0
-	1	0
+	1	0
+	1	0
+	1	1
+	1	1
+	1	1
+	1	1

Poll Question 3 SKIP

Which feature would **mutual information** select for the next split?

1. A
2. B
3. A or B (tie)
4. Neither

Decision Tree Learning Example

Dataset:

Label Y, Features A and B

Y	A	B
-	1	0
-	1	0
+	1	0
+	1	0
+	1	1
+	1	1
+	1	1
+	1	1

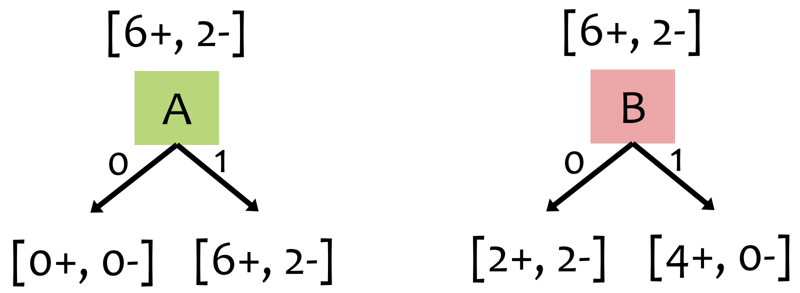
=1

Decision Tree Learning Example

Dataset:

Label Y, Features A and B

Y	A	B
-	1	0
-	1	0
+	1	0
+	1	0
+	1	1
+	1	1
+	1	1
+	1	1



Mutual Information

H(Y) = – 2/8 log(2/8) – 6/8 log(6/8)

H(Y|A=0) = "undefined"
H(Y|A=1) = – 2/8 log(2/8) – 6/8 log(6/8)
= H(Y)
H(Y|A) = P(A=0)H(Y|A=0) + P(A=1)H(Y|A=1)
= 0 + H(Y|A=1) = H(Y)
I(Y; A) = H(Y) - H(Y|A=1) = 0

H(Y|B=0) = -2/4 log(2/4) – 2/4 log(2/4)
H(Y|B=1) = -0 log(0) – 1 log(1) = 0
H(Y|B) = 4/8(0) + 4/8(H(Y|B=0))
I(Y; B) = H(Y) – 4/8 H(Y|B=0) > 0