10-301/601: Introduction to Machine Learning Lecture 3 – Decision Trees

Hoda Heidari, Henry Chai & Matt Gormley

1/24/24

Front Matter

- Announcements:
  - HW1 released 1/17, due 1/24 (today!) at 11:59 PM
    - Reminder: we will grant (basically) any extension requests for this assignment!
  - HW2 released 1/24 (today!), due 2/5 at 11:59 PM
    - Unlike HW1, you will only have:
      - 1 *graded* submission for the written portion
      - 10 submissions of the programming portion to our autograder

Q & A:

How do these in-class polls work?

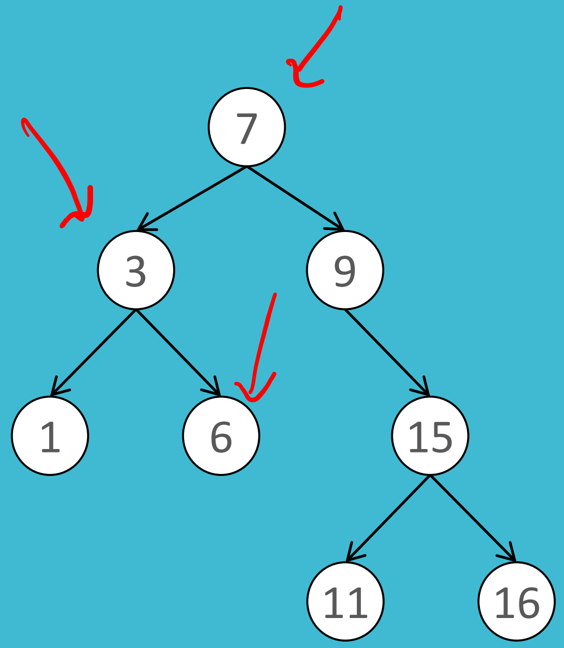
- Open the poll, either by clicking the [Poll] link on the schedule page of our course website or going to http://poll.mlcourse.org
- Sign into Google Forms using your **Andrew email**
- Answer all poll questions **during lecture for full credit** or **within 24 hours for half credit**
- Avoid the **toxic option** (will be clearly specified in lecture) which gives **negative poll points**
- You have 8 free “poll points” for the semester that will excuse you from all polls from a single lecture; you cannot use more than 3 poll points consecutively.

Poll Question 1:

Which of the following did you bring to class today? Select all that apply

A. A smartphone
B. A flip phone
C. A payphone **(TOXIC)**
D. No phone

Background: Recursion



- A **binary search tree** (BST) consists of nodes, where each node:
  - has a value, v
  - up to 2 children, a left descendant and a right descendant
  - 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

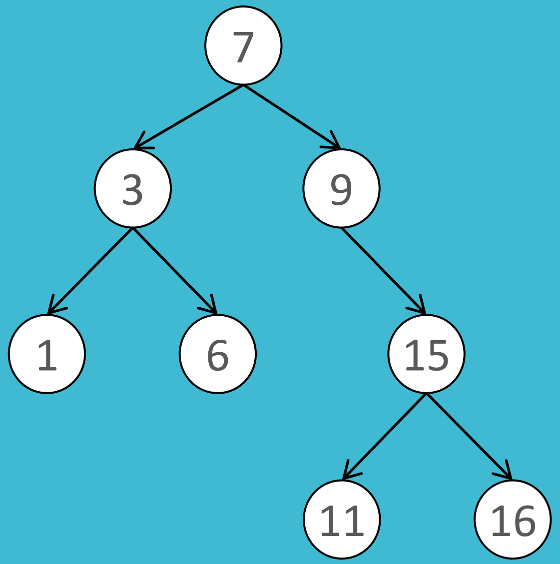
contains(node, 5)

||

n

```
def contains_iterative(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
```

Background: Recursion

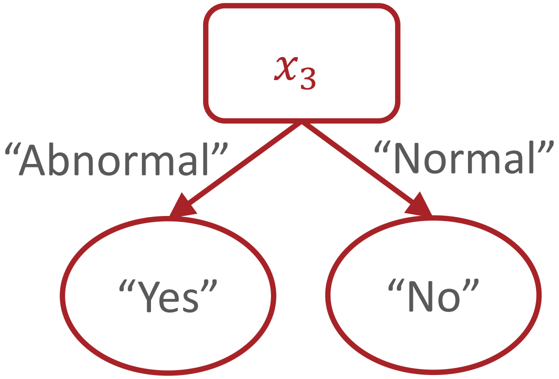


- A **binary search tree** (BST) consists of nodes, where each node:
  - has a value, v
  - up to 2 children, a left descendant and a right descendant
  - all its left descendants have values less than v and its right descendants have values greater than v
- We like BSTs because they permit search in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

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

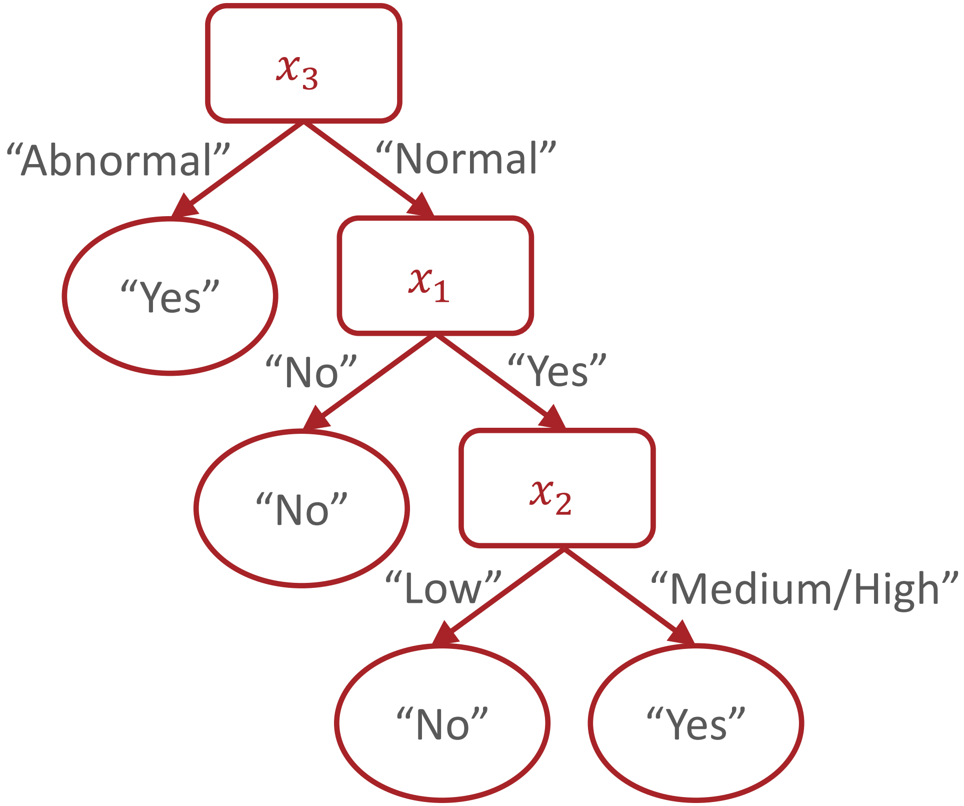
From Decision Stump …

x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes



From Decision Stump to Decision Tree

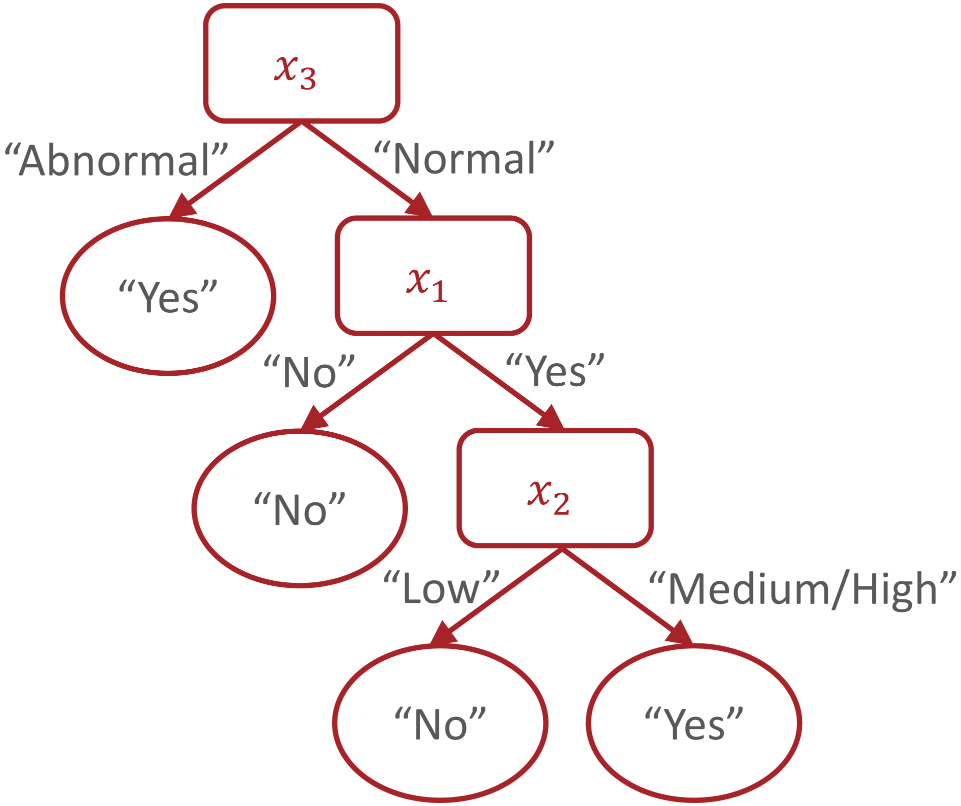
x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes



From Decision Stump to Decision Tree

x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

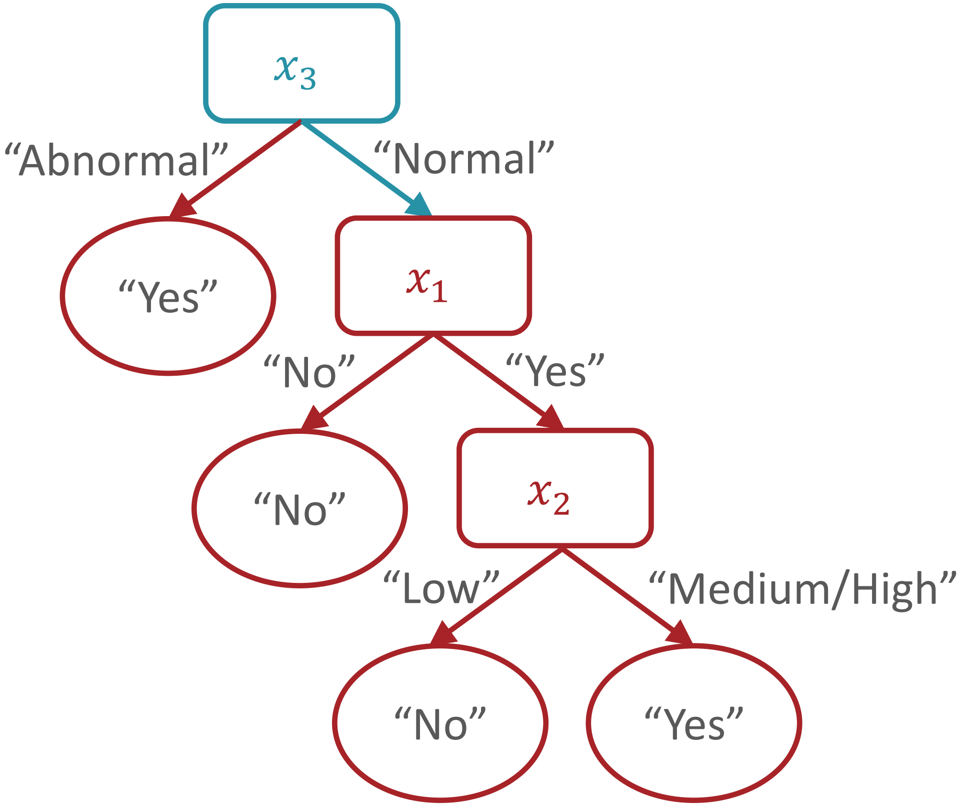
No	High	Normal	No



From Decision Stump to Decision Tree

x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

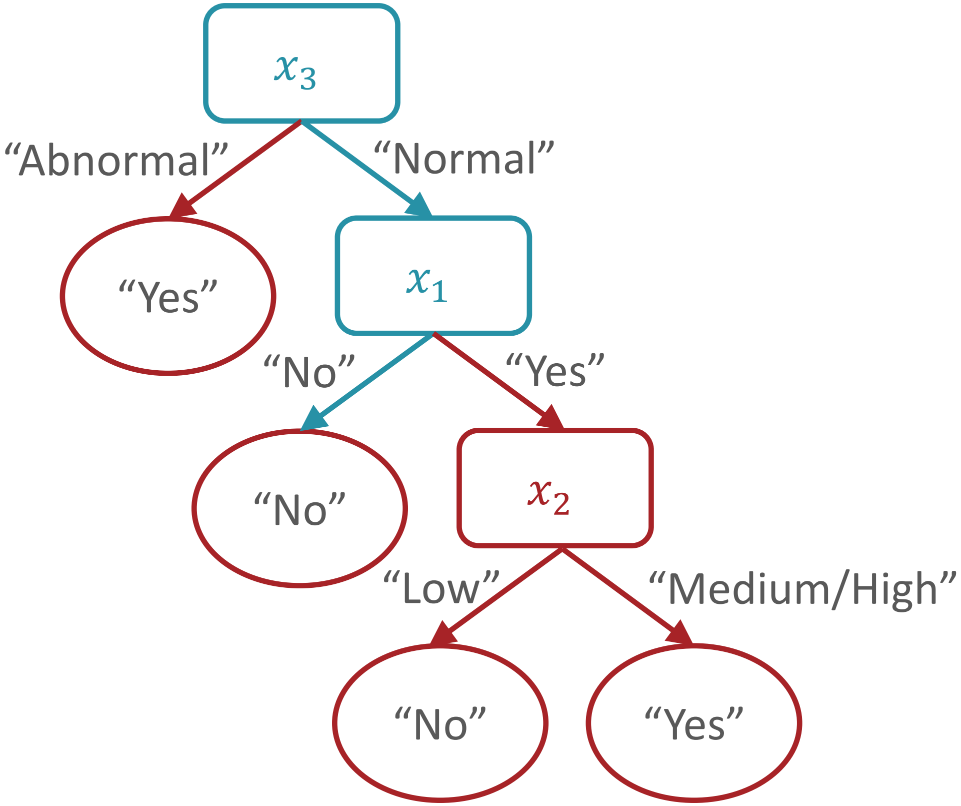
No	High	Normal	No



From Decision Stump to Decision Tree

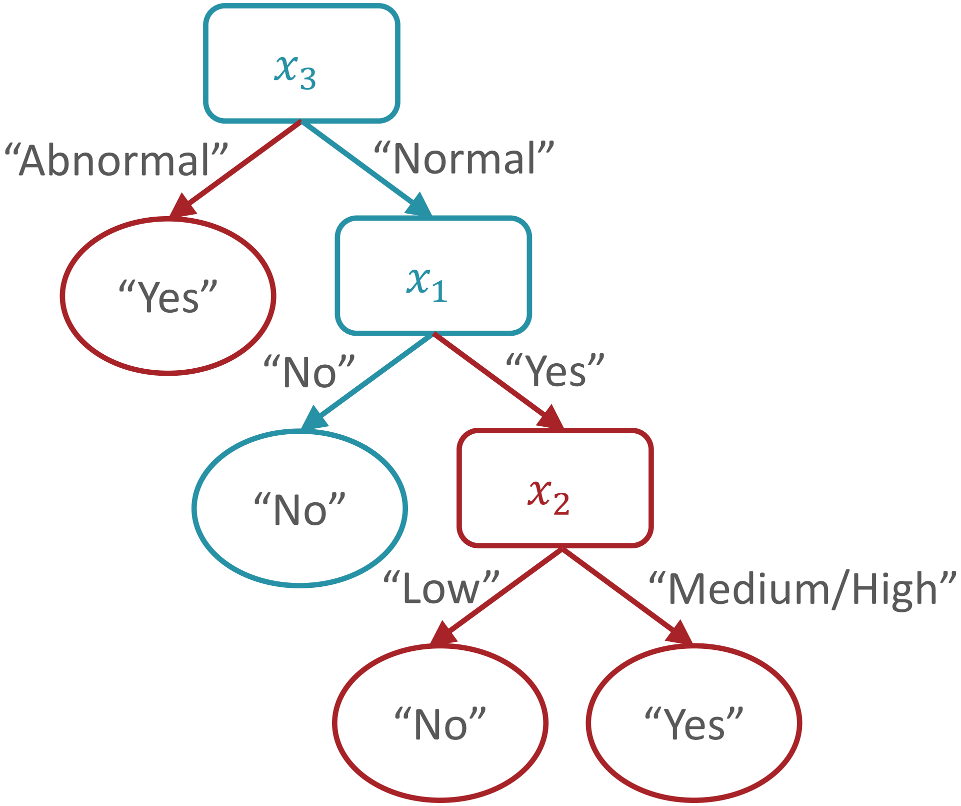
x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes

No	High	Normal	No



From Decision Stump to Decision Tree

x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes
No	High	Normal	No



Decision Tree: Pseudocode

(iteratively)

$[x_1', x_2', \ldots, x_D']$

```
def h($x'$):
    current_node = root
    # Walk from the root to a leaf nodes
    while(true):
        if the current_node is internal:
            check the associated feature, $x_d$
            go down the branch corresponding to $x_d'$
        else (current_node is a leaf):
            return the associated label
```

Decision Tree: Example

Learned from medical records of 1000 women

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-
```

sufficient statistics

Figure courtesy of Tom Mitchell

Decision Stumps: Questions

1. Why stop at just one feature?

   Don't

2. How can we pick which feature to split on?

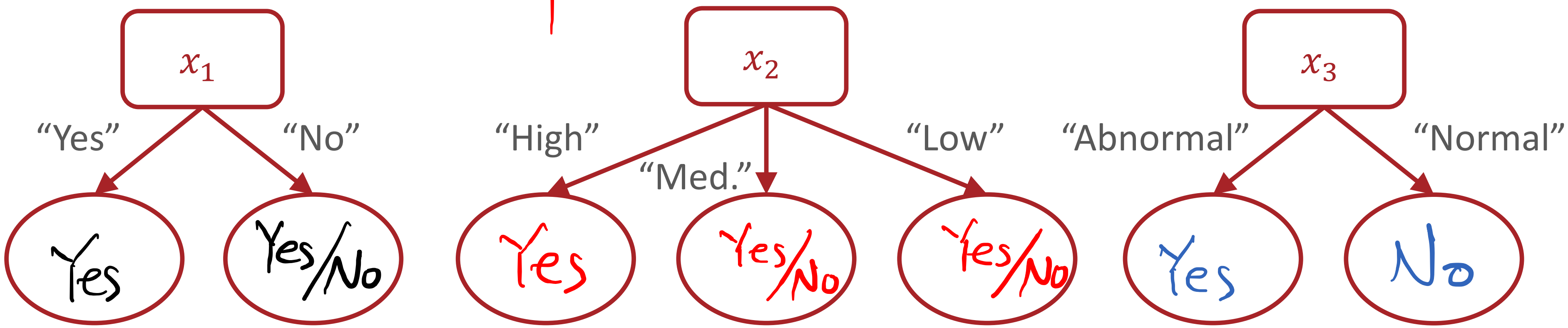
3. How can we pick the order of the splits?

Splitting Criterion

- A **splitting criterion** is a function that measures how good or useful splitting on a particular feature is *for a specified dataset*
- Idea: when deciding which feature to split on, use the one that optimizes the splitting criterion

Training Error Rate as a Splitting Criterion

x_1 Family History	x_2 Resting Blood Pressure	x_3 Cholesterol	y Heart Disease?
Yes	Low	Normal	No
No	Medium	Normal	No
No	Low	Abnormal	Yes
Yes	Medium	Normal	Yes
Yes	High	Abnormal	Yes



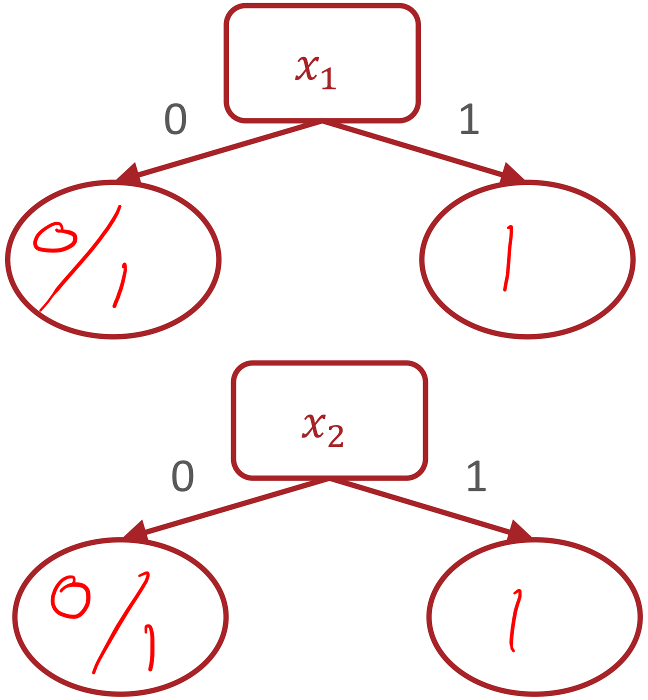
Training error rate: 2/5

Training error rate: 2/5

Training error rate: 1/5

Training Error Rate as a Splitting Criterion?

x_1	x_2	y
1	0	0
1	0	0
1	0	1
1	0	1
1	1	1
1	1	1
1	1	1
1	1	1



Training error rate: $2/8$

Splitting Criterion

- A **splitting criterion** is a function that measures how good or useful splitting on a particular feature is *for a specified dataset*
- Idea: when deciding which feature to split on, use the one that optimizes the splitting criterion
- Potential splitting criteria:
  - Training error rate (minimize)
  - Gini impurity (minimize) → CART algorithm
  - Mutual information (maximize) → ID3 algorithm

Splitting Criterion

- A **splitting criterion** is a function that measures how good or useful splitting on a particular feature is *for a specified dataset*
- Idea: when deciding which feature to split on, use the one that optimizes the splitting criterion
- Potential splitting criteria:
  - Training error rate (minimize)
  - Gini impurity (minimize) → CART algorithm
  - **Mutual information** (maximize) → ID3 algorithm

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.

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

where X is a (discrete) random variable

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

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

$|\cdot| =$ cardinality of a collection or the "size of"

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

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 all the elements in S are the same, then

$$H(S) = -\frac{N}{N}\log_2\frac{N}{N} = -1\log_2 1 = 0$$

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

$$H(S) = -\left(\frac{N/2}{N}\log_2\frac{N/2}{N} + \frac{N/2}{N}\log_2\frac{N/2}{N}\right) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right)$$

$$= -\left(\frac{1}{2}(-1) + \frac{1}{2}(-1)\right) = 1$$

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 $X = v$

Mutual Information

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

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

$$= H(y) - \sum_{v \in V(x_d)} f_v \left(\underline{H\left(Y_{x_d=v}\right)} \right)$$

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 $x_d = v$

Mutual Information: Example

x_d	y
1	1
1	1
0	0
0	0

$$I(y; x_d) = \underline{H(y)} - \frac{2}{4}\underline{H(y_{x_d=0})} - \frac{2}{4}\underline{H(y_{x_d=1})}$$

$$= 1 - \frac{1}{2}(0) - \frac{1}{2}(0)$$

$$= 1$$

Mutual Information: Example

x_d	y
→ 1	1 ☆
0	1
→ 1	0 ☆
0	0

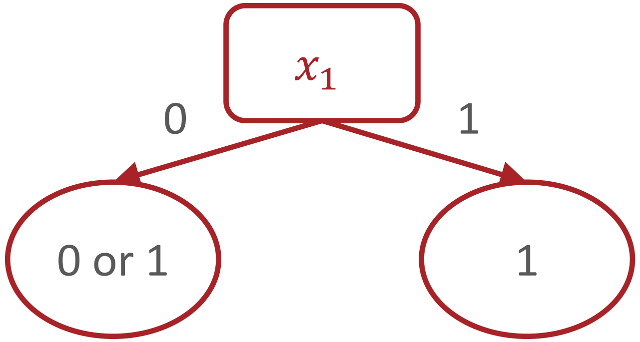
$$I(y ; x_d) = H(y) - \frac{2}{4} H(y_{x_d=0}) - \frac{2}{4} H(y_{x_d=1})$$

$$= 1 - \frac{1}{2}(1) - \frac{1}{2}(1)$$

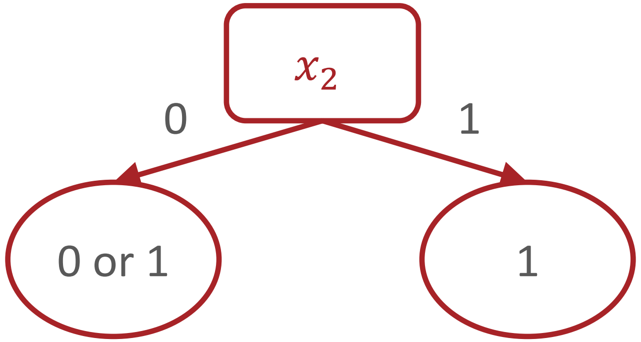
$$= 0$$

Mutual Information as a Splitting Criterion

x_1	x_2	y
1	0	0
1	0	0
1	0	1
1	0	1
1	1	1
1	1	1
1	1	1
1	1	1



Mutual Information: 0

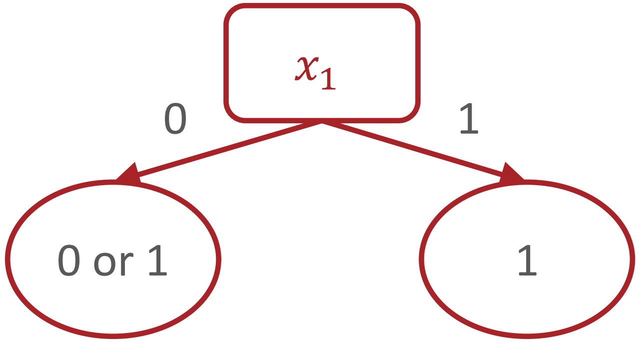


Mutual Information: $H(Y) - \frac{1}{2}\underset{\uparrow}{H(Y_{x_2=0})} - \frac{1}{2}\underset{\uparrow}{H(Y_{x_2=1})}$

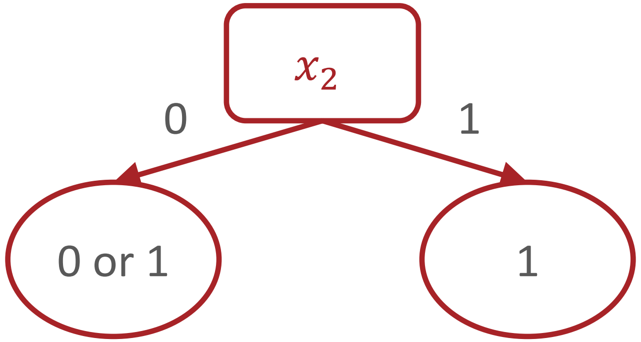
$$H(y) - \frac{8}{8} H(y_{x_1=1}) - \frac{0}{8} H(\underset{\uparrow}{y_{x_1=0}})$$

Mutual Information as a Splitting Criterion

x_1	x_2	y
1	0	0
1	0	0
1	0	1
1	0	1
1	1	1
1	1	1
1	1	1
1	1	1



Mutual Information: 0



Mutual Information: $\left(-\frac{2}{8}\log_2\frac{2}{8} - \frac{6}{8}\log_2\frac{6}{8}\right) - \frac{1}{2}(1) - \frac{1}{2}(0) \approx 0.31$

Decision Stumps: Questions

1. Why stop at just one feature?
2. How can we pick which feature to split on?
3. How can we pick the order of the splits?

Recursion!

Decision Tree: Pseudocode

(recursively)

```
def train($\mathcal{D}_{train}$):
    store root = tree_recurse($\mathcal{D}_{train}$)
def tree_recurse($\mathcal{D}'$):
    q = new node()
    base case – if (SOME CONDITION):
    recursion – else:
        find the best feature to split on, $x_d$
        q.split = $x_d$
        for v in $V(x_d)$, all possible values of $x_d$:
            $D_v = \{(x^{(n)}, y^{(n)}) \in D' \mid x_d^{(n)} = v\}$
            q.children(v) = tree_recurse($D_v$)
    return q
```

Decision Tree: Pseudocode

```
def train($\mathcal{D}_{train}$):
    store root = tree_recurse($\mathcal{D}_{train}$)
def tree_recurse($\mathcal{D}'$):
    q = new node()
    base case – if (the entropy of the labels in $D' < t$,
            $|D'|$ is less than $c$,
            all feature vectors in $D'$ are the same,
            some other stopping criteria):
                q.label = mode(labels in $D'$)

    recursion – else:
    return q
```