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

Henry Chai & Matt Gormley

9/6/23

# Front Matter

- Announcements:
  - HW1 released 9/1, due 9/6 (today!) at 11:59 PM
    - Reminder: we will grant (basically) any extension requests for this assignment!
  - HW2 released 9/6 (today!), due 9/15 at 11:59 PM
    - Unlike HW1, you will only have:
      - 1 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.

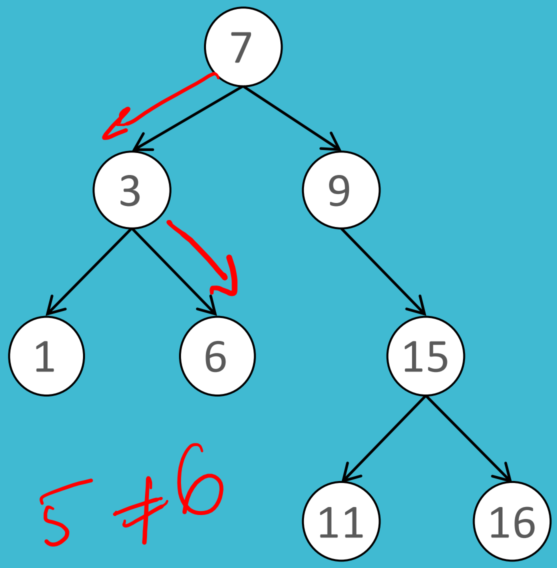
# Poll Question 1:

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

A. A smartphone
B. A flip phone
C. A payphone
D. No phone

# Background: Recursion

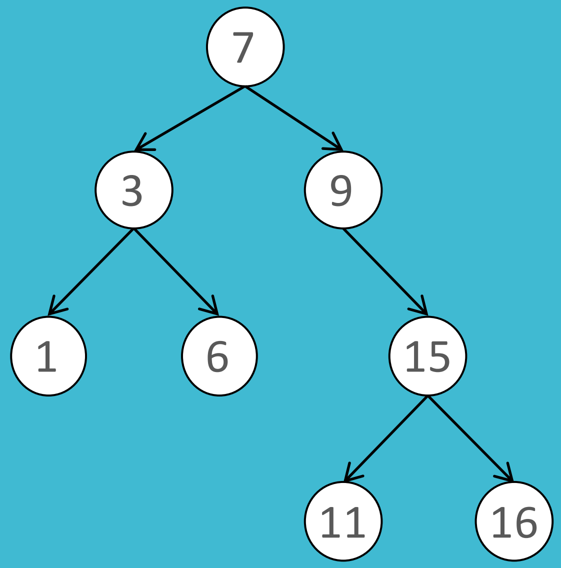
5?




- 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_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)
    else:  (base case)
        return key == node.value
```

# Recall: Decision Stumps

Algorithm 3: 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 $x_d$

| | $y$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|---|
| predictions | 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 |

Nonzero training error, but perhaps still better than the memorizer

Example decision stump:

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

# But why would we only use just one feature?

Algorithm 3: 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 |

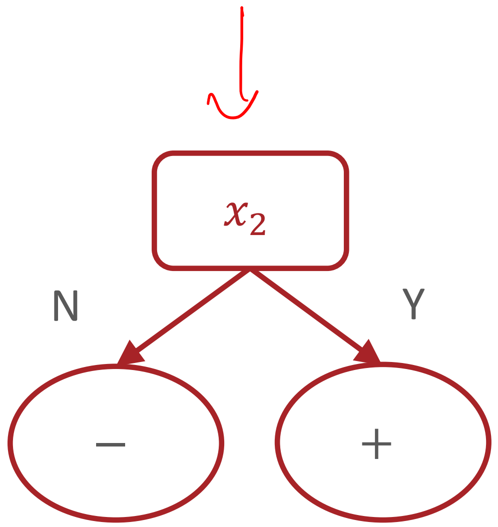
Nonzero training error, but perhaps still better than the memorizer

Example decision stump:

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

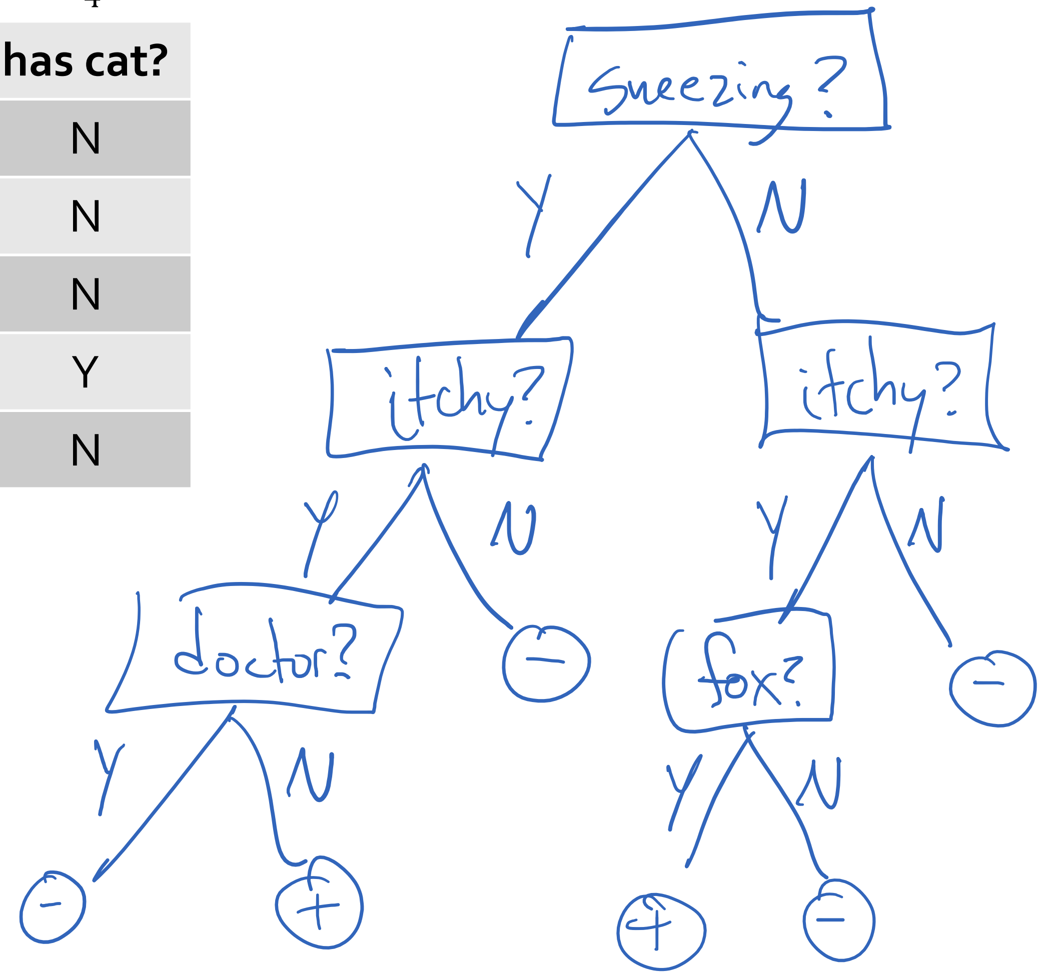
# From Decision Stump …

| $y$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|
| 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

| $y$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|
| 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 |



# Decision Tree: Pseudocode

(iterative)

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

def $h(\boldsymbol{x}')$:

— walk from the root to a leaf node

while (true):

  if (curr_node is internal):

    check the associated feature, $x_d$

    go down the branch according to $x_d'$

  else:

    curr_node is a leaf node

    return the label stored at curr_node

# 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-
```



Figure courtesy of Tom Mitchell

# Decision Tree Questions

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

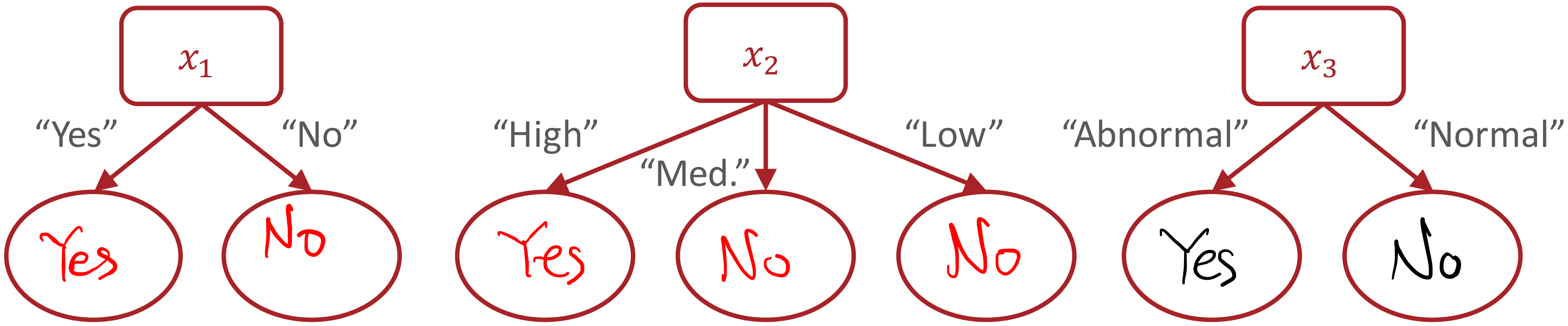
2. How do we pick the order of the splits?

# 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

# Poll Question 2:

Which feature would you split on using training error rate as the 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 |

A. $x_1$

B. $x_2$

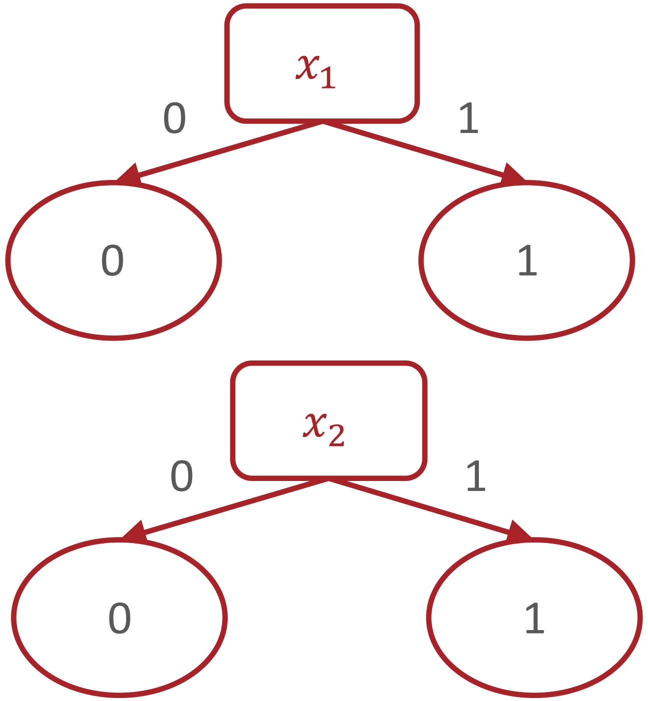
C. Either $x_1$ or $x_2$

D. Neither $x_1$ nor $x_2$ (Toxic)

# Poll Question 2:

Which feature would you split on using training error rate as the 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

$|\cdot| =$ "size of" or cardinality

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

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

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

# 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

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

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

# Mutual Information

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

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

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

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

$$I(y; x_d) = \underline{H(y) - H(y|x_d)}$$

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

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 |

$$I(X_d; Y) = H(Y) - \sum_{v \in V(X_d)} f_v \left(H\left(Y_{X_d = v}\right)\right)$$

$$= 1 - \left(\frac{2}{4} H\left(Y_{X_d=1}\right) + \frac{2}{4} H\left(Y_{X_d=0}\right)\right)$$

$$= 1 - \left(\frac{2}{4}(0) + \frac{2}{4}(0)\right) = 1$$

# Mutual Information: Example

| $x_d$ | $y$ |
|---|---|
| 1 | 1 |
| 0 | 1 |
| 1 | 0 |
| 0 | 0 |

$$I(x_d ; Y) = H(Y) - \frac{2}{4}\left(H(Y_{x_d=1})\right) - \frac{2}{4}\left(H(Y_{x_d=0})\right)$$

$$= 1 - \frac{2}{4}(1) - \frac{2}{4}(1) = 0$$

# Poll Question 3:

# Which feature would you split on using mutual information as the 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 |

A. $x_1$

B. $x_2$

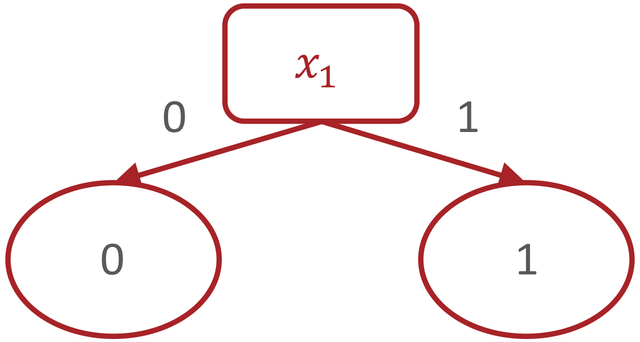
C. Either $x_1$ or $x_2$

D. Neither $x_1$ nor $x_2$ (toxic)

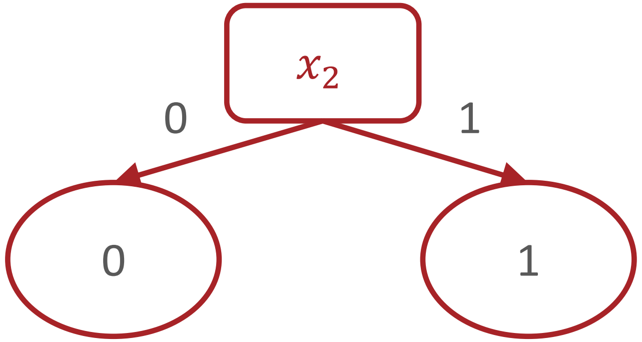
# Poll Question 3:

Which feature would you split on using mutual information as the 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

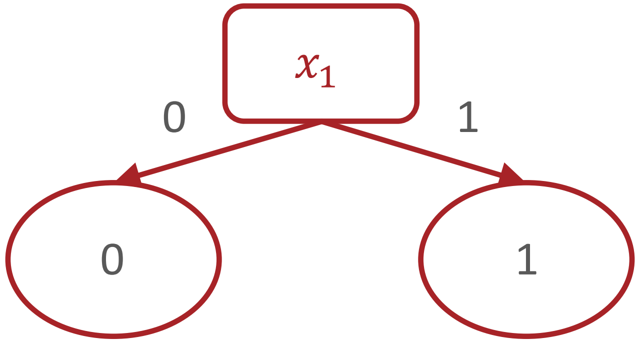



Mutual Information: $H(Y) - \frac{1}{2}H\left(Y_{x_2=0}\right) - \frac{1}{2}H\left(Y_{x_2=1}\right)$

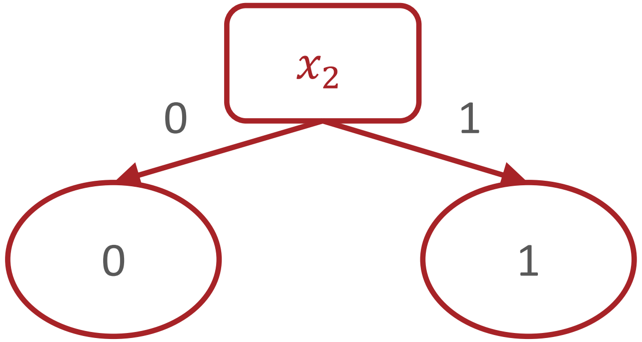
# Poll Question 3:

Which feature would you split on using mutual information as the 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: $-\frac{2}{8}\log_2\frac{2}{8} - \frac{6}{8}\log_2\frac{6}{8} - \frac{1}{2}(1) - \frac{1}{2}(0) \approx 0.31$

# Decision Tree Questions

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

   Use mutual information

2. How do we pick the order of the splits?

   Recursion!

# Decision Tree: Pseudocode

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

def tree_recurse(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):
            q.child(v) = tree_recurse(D_v)
            where D_v is the subset of D′
            w/ all data points x_d = v
    return q
```

# Decision Tree: Pseudocode

```
def train(𝒟):
    store root = tree_recurse(𝒟)
def tree_recurse(𝒟′):
    q = new node()
    base case – if(all labels are the same
              OR all features have already been
                   on
              OR if D′ is empty
              OR some other stopping criteria)
                   q.label = majority_vote(D′)
    recursion – else:
    return q
```