CS6375: Machine Learning

Gautam Kunapuli

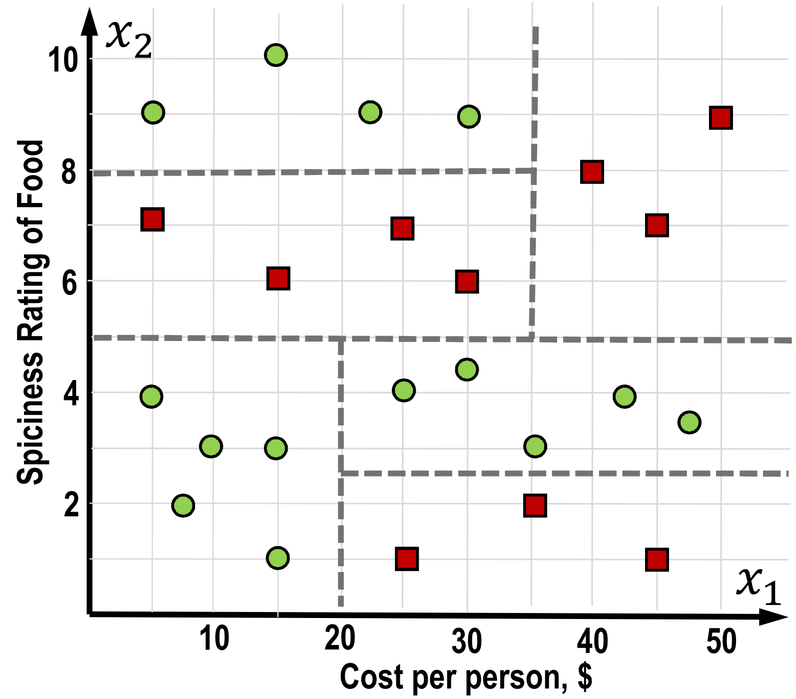
Decision Trees

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Example: Restaurant Recommendation

Example: Develop a model to **recommend restaurants** to users depending on their past dining experiences.

Here, the features are **cost** (x_1) and the user's **spiciness rating** of the food at the restaurant (x_2) and the label is if they liked the food ($y_i =$ green circle) or not ($y_i =$ red square).



A data set is **linearly separable** if there exists a hyperplane that separates positive examples from negative examples.

- Relatively easy to learn (using standard techniques)
- Easy to visualize and interpret

Many **data sets in real world** are **<u>not linearly separable</u>**!
Two options:

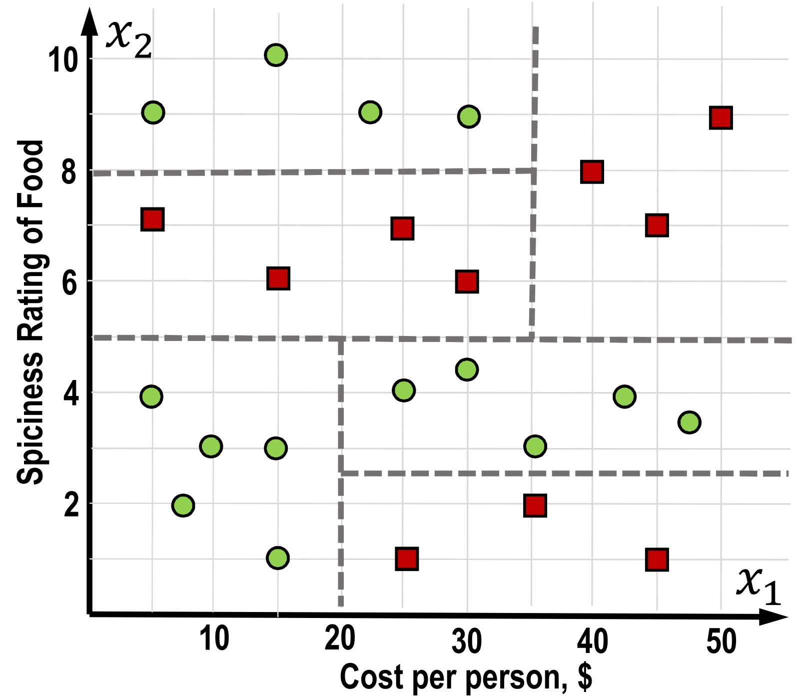
- Use **non-linear features,** and learn a linear classifier in the transformed non-linear feature space
- Use **non-linear classifiers**

Decision Trees can handle nonlinear separable data sets and are one of the **most popular classifiers**

Decision Trees: Introduction

Example: Develop a model to **recommend restaurants** to users depending on their past dining experiences.

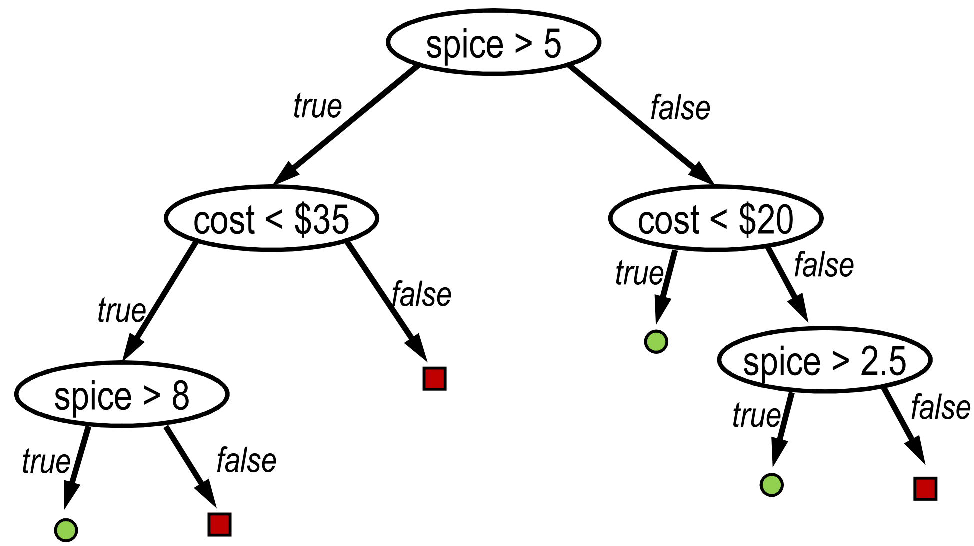
Here, the features are **cost** (x_1) and the user's **spiciness rating** of the food at the restaurant (x_2) and the label is if they liked the food ($y_i =$) or not ($y_i =$).



Decision Trees represent decision-making as a **checklist of questions**, and visualize it using a tree-structure

Decision Tree **representation**:

- Each **non-leaf node tests an attribute**/feature
- Each **branch corresponds to attribute**/feature value, a decision (to choose a path) as a result of the test
- Each **leaf node assigns a classification**

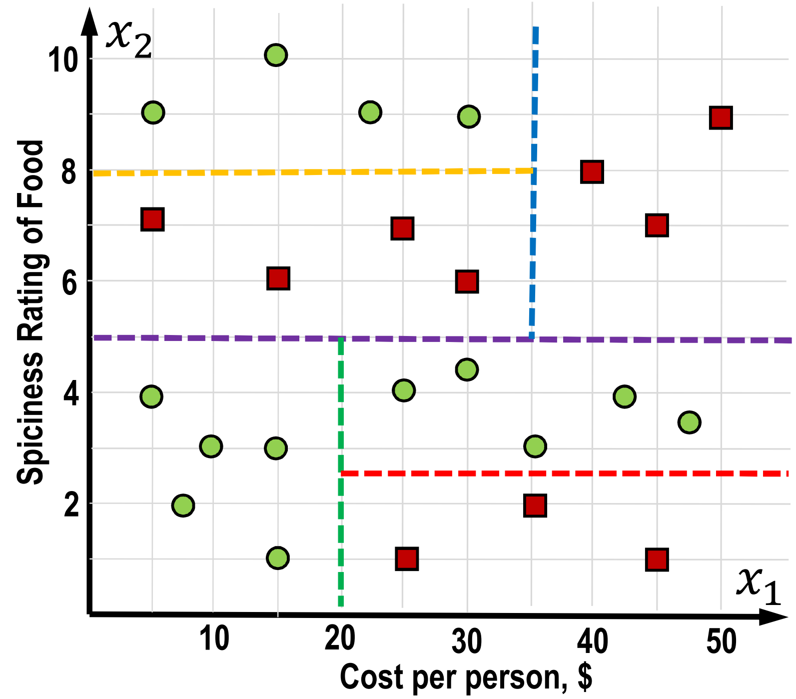


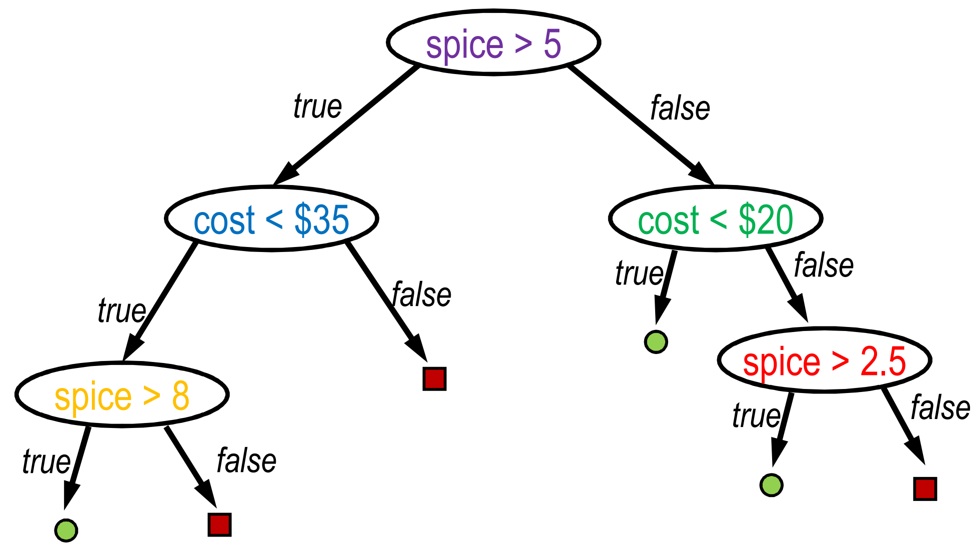
Decision Trees: Introduction

Example: Develop a model to **recommend restaurants** to users depending on their past dining experiences.

Here, the features are **cost** (x_1) and the user's **spiciness rating** of the food at the restaurant (x_2) and the label is if they liked the food ($y_i =$) or not ($y_i =$).

- Decision trees divide the feature space into **axis-parallel rectangles**
- Decision Trees can handle **arbitrarily non-linear representations**, given sufficient tree complexity
- Worst-case scenario: the decision tree has an **exponential number of nodes!** (why?)

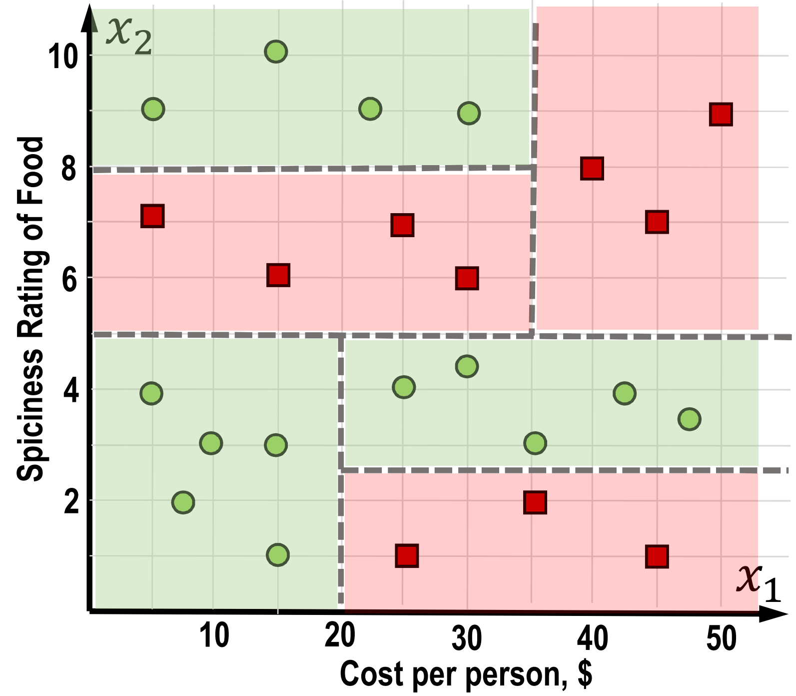




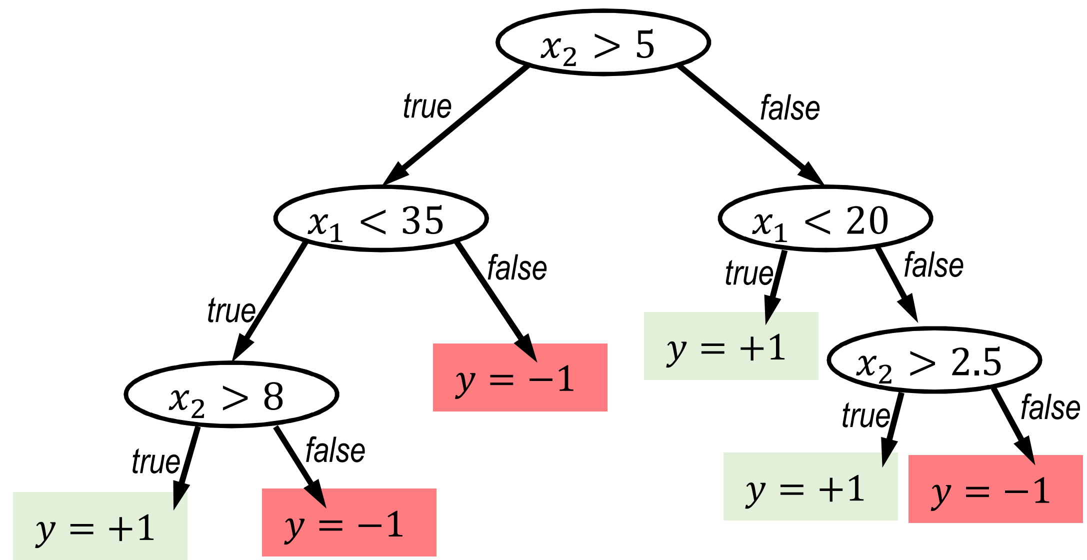
Decision Trees: Introduction

Example: Develop a model to **recommend restaurants** to users depending on their past dining experiences.

Here, the features are **cost** (x_1) and the user's **spiciness rating** of the food at the restaurant (x_2) and the label is if they liked the food ($y = +1$) or not ($y = +1$).



- Decision trees divide the feature space into **axis-parallel rectangles**
- Decision Trees can handle **arbitrarily non-linear representations**, given sufficient tree complexity
- Worst-case scenario: the decision tree has an **exponential number of nodes!**
  - If the target function has n Boolean features, there are 2^n possible inputs
  - In the worst case, there is one leaf node for each input (for example: XOR)

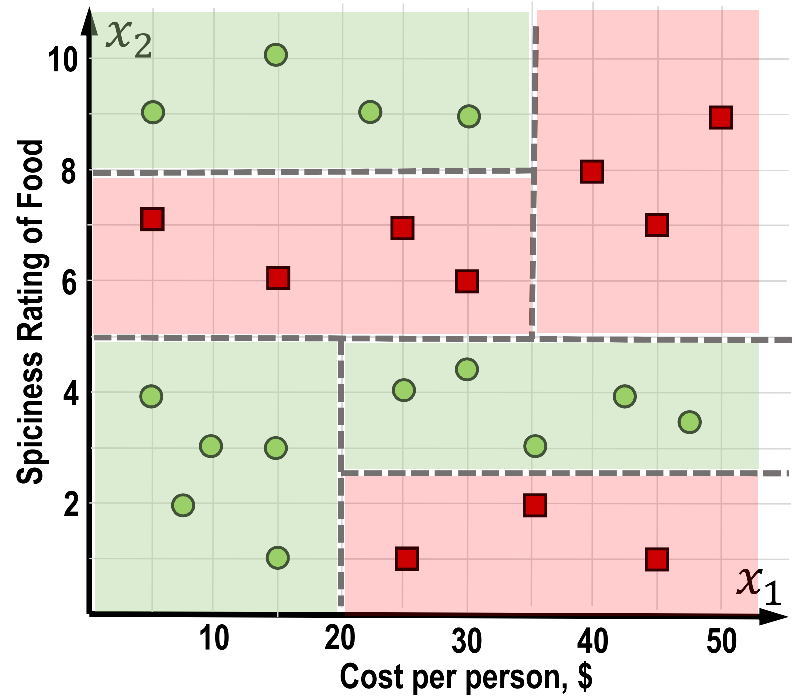


Decision trees are <u>not</u> unique, and many decision trees can represent the same hypothesis!

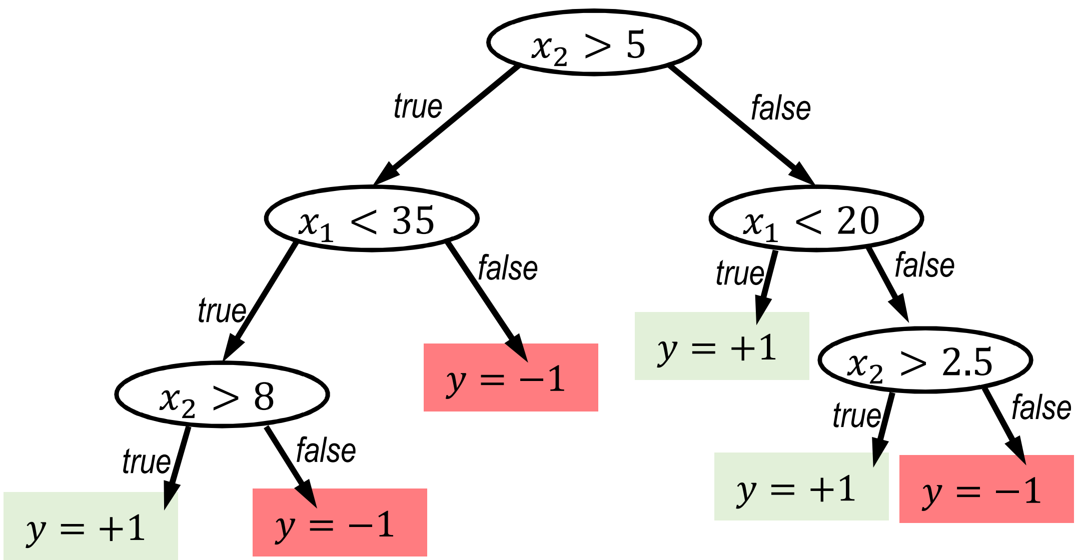
Decision Trees: Introduction

Example: Develop a model to **recommend restaurants** to users depending on their past dining experiences.

Here, the features are **cost** (x_1) and the user's **spiciness rating** of the food at the restaurant (x_2) and the label is if they liked the food ($y = +1$) or not ($y = +1$).



When do you want Decision Trees?
When instances are **describable by attribute-value pairs**:
- target function is **discrete-valued**
- **disjunctive hypothesis** may be required
- need for **interpretable** model



Examples:
- Equipment or medical diagnosis
- Credit risk analysis
- Modeling calendar scheduling preferences

Learning Decision Trees

Problem Formulation: Find a decision tree **h** that achieves minimum misclassification errors on the training data

- **Solution Approach 1 (Naïve solution):** Create a decision tree with one path from root to leaf for each training example. *Such a tree would just memorize the training data, and will **not generalize well to new points**.*

- **Solution Approach 2 (Exact solution):** Find the **smallest** tree that minimizes the classification error. *Finding this solution is **NP-Hard**!*

- **Solution Approach 3 (Heuristic solution)**: Top-down greedy search

Initialize: Choose the best feature f^* for the root of the tree
Function GrowTree(data, f^*)
  [1]Separate data into subsets $\{S_1, S_2, \ldots, S_k\}$, where each subset S_i contains examples that have the **same value for** f^*
  [2] for $S_i \in \{S_1, S_2, \ldots, S_k\}$
    Choose the best feature f_i^* for the next node
    Recursively GrowTree(S_i, f_i^*) until all examples have the same class label

Learning Decision Trees

Problem Formulation: Find a decision tree **h** that achieves minimum misclassification errors on the training data

- **Solution Approach 1 (Naïve solution):** Create a decision tree with one path from root to leaf for each training example. *Such a tree would just memorize the training data, and will **not generalize well to new points**.*

- **Solution Approach 2 (Exact solution):** Find the **smallest** tree that minimizes the classification error. *Finding this solution is **NP-Hard**!*

- **Solution Approach 3 (Heuristic solution)**: Top-down greedy search

Initialize: Choose the best feature f^* for the root of the tree
Function GrowTree(data, f^*)
  [1]Separate data into subsets $\{S_1, S_2, \ldots, S_k\}$, where each subset S_i contains examples that have the **same value for** f^*
  [2] for $S_i \in \{S_1, S_2, \ldots, S_k\}$
    Choose **the best feature** f_i^* for the next node
    Recursively GrowTree(S_i, f_i^*) until all **examples have the same class label**

How do we pick the best feature?

How do we decide when to stop?

Learning Decision Trees

Problem Formulation: Find a decision tree **h** that achieves minimum misclassification errors on the training data

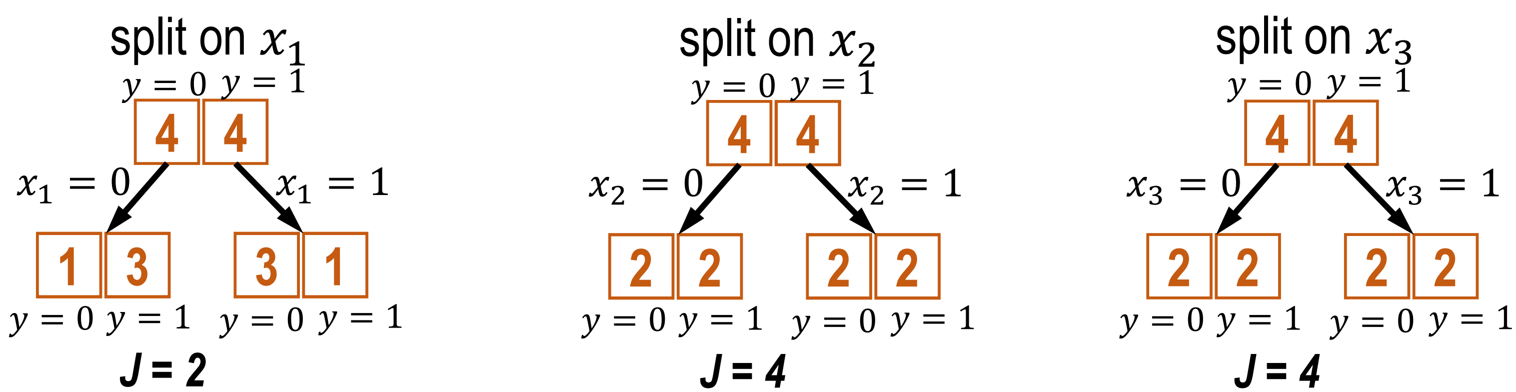
Solution Approach 3 (Heuristic solution): Top-down greedy search
Initialize: Choose the best feature f^* for the root of the tree
Function GrowTree(data, f^*)
[1]Separate data into subsets $\{S_1, S_2, \ldots, S_k\}$, where each subset S_i contains examples that have the **same value for** f^*
[2] for $S_i \in \{S_1, S_2, \ldots, S_k\}$
Choose **the best feature** f_i^* for the next node
Recursively GrowTree(S_i, f_i^*) until all **examples have the same class label**

How do we pick the next best feature to place in a decision tree?

- Random choice
- Largest number of values
- Fewest number of values
- **Lowest classification error**
- Information theoretic measure (Quinlan's approach)

x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Training examples



J is splitting criterion measured for each split, in this case, the classification error

Learning Decision Trees

Problem Formulation: Find a decision tree **h** that achieves minimum misclassification errors on the training data

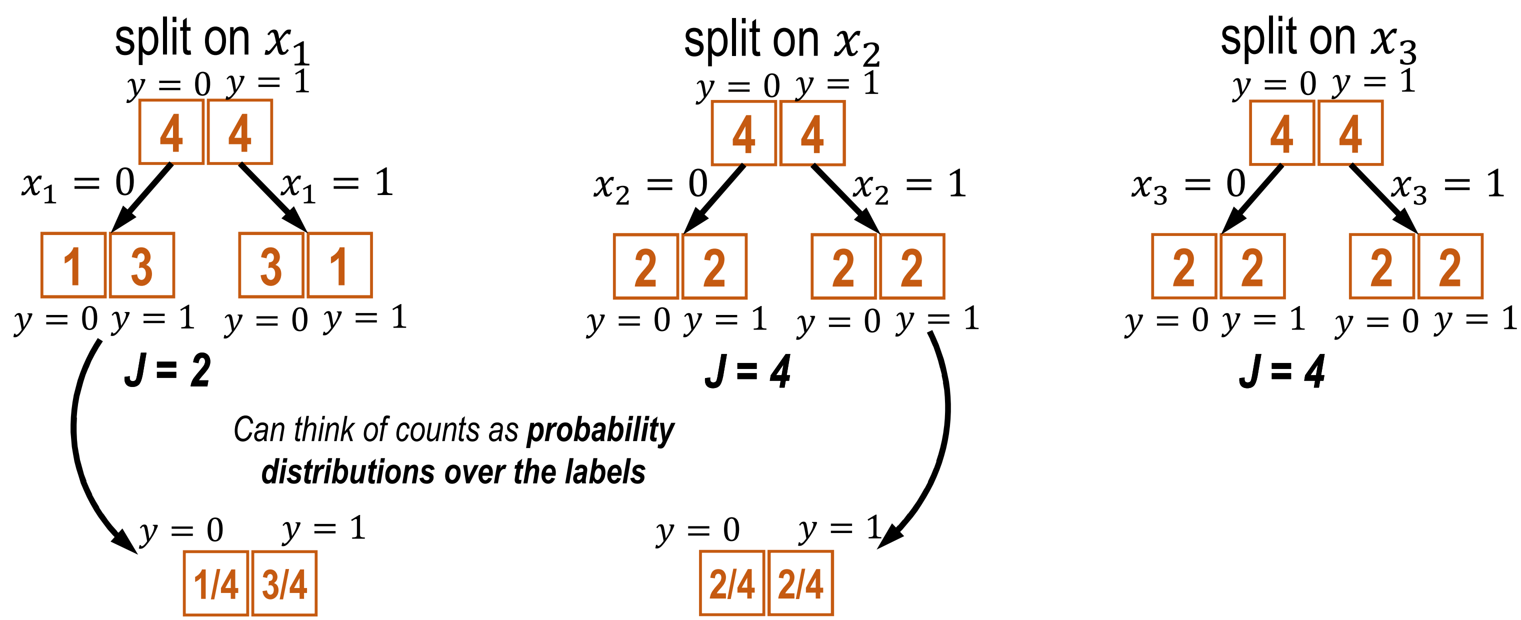
Solution Approach 3 (Heuristic solution): Top-down greedy search
Initialize: Choose the best feature f^* for the root of the tree
Function GrowTree(data, f^*)
[1]Separate data into subsets $\{S_1, S_2, \ldots, S_k\}$, where each subset S_i contains examples that have the **same value for** f^*
[2] for $S_i \in \{S_1, S_2, \ldots, S_k\}$
Choose **the best feature** f_i^* for the next node
Recursively GrowTree(S_i, f_i^*) until all **examples have the same class label**

How do we pick the next best feature to place in a decision tree?

- Random choice
- Largest number of values
- Fewest number of values
- **Lowest classification error**
- Information theoretic measure (Quinlan's approach)

x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

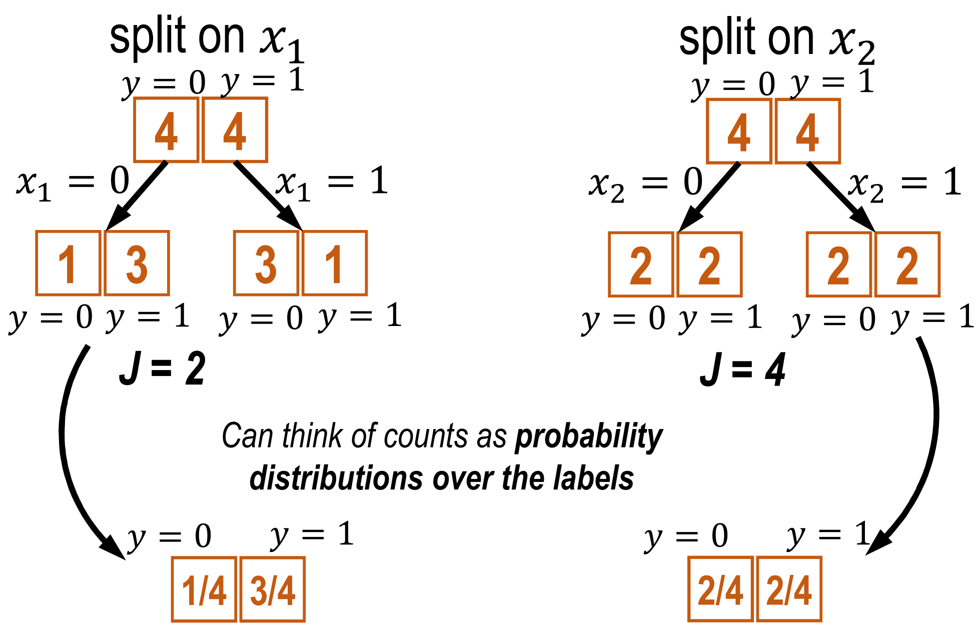
Training examples



Learning Decision Trees

Problem Formulation: Find a decision tree **h** that achieves minimum misclassification errors on the training data

Solution Approach 3 (Heuristic solution): Top-down greedy search
Initialize: Choose the best feature f^* for the root of the tree
Function GrowTree(data, f^*)
[1]Separate data into subsets $\{S_1, S_2, \ldots, S_k\}$, where each subset S_i contains examples that have the **same value for** f^*
[2] for $S_i \in \{S_1, S_2, \ldots, S_k\}$
Choose **the best feature** f_i^* for the next node
Recursively GrowTree(S_i, f_i^*) until all **examples have the same class label**



How do we pick the next best feature to place in a decision tree?
- Random choice
- Largest number of values
- Fewest number of values
- **Lowest classification error**
- Information theoretic measure (Quinlan's approach)

The selected attribute is a **good split** if we are **more "certain"** about the classification after the split (compare with the perceptron)

- If each partition with respect to the chosen attribute has a **distinct class label**, we are **completely certain** about the classification

$y = 0$	$y = 1$
0.0	1.0

- If **class labels are evenly divided** between partitions, we are **very uncertain** about the classification

$y = 0$	$y = 1$
0.5	0.5

Learning Decision Trees

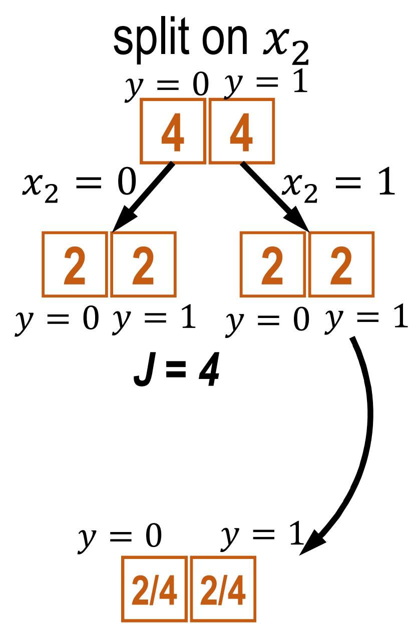
Problem Formulation: Find a decision tree **h** that achieves minimum misclassification errors on the training data

Solution Approach 3 (Heuristic solution): Top-down greedy search
Initialize: Choose the best feature f^* for the root of the tree
Function GrowTree(data, f^*)
  [1]Separate data into subsets $\{S_1, S_2, \ldots, S_k\}$, where each subset S_i contains examples that have the **same value for** f^*
  [2] for $S_i \in \{S_1, S_2, \ldots, S_k\}$
    Choose **the best feature** f_i^* for the next node
    Recursively GrowTree(S_i, f_i^*) until all **examples have the same class label**

How do we pick the next best feature to place in a decision tree?
- Random choice
- Largest number of values
- Fewest number of values
- **Lowest classification error**
- Information theoretic measure (Quinlan's approach)

The selected attribute is a **good split** if we are **more "certain"** about the classification after the split (compare with the perceptron)

We need a better way to resolve the uncertainty!



- If each partition with respect to the chosen attribute has a **distinct class label**, we are **completely certain** about the classification

$y = 0$	$y = 1$
0.0	1.0

- If **class labels are evenly divided** between partitions, we are **very uncertain** about the classification

$y = 0$	$y = 1$
0.5	0.5

Discrete Probability and Information Theory

A **discrete probability distribution** describes the probability of occurrence of each value of a discrete random variable.

The **surprise** or **self-information** of each event of X is defined to be

$$S(X = x) = -\log_2 \text{Prob}(X = x)$$

- An event with **probability 1 has zero surprise**; *this is because when the content of a message is known beforehand with certainty, there is no actual information conveyed*
- The **smaller the probability** of event, the **larger the quantity of self-information** associated with the message that the event occurred
- An event with **probability 0 has infinite surprise**

- The surprise is the **asymptotic number of bits of information** that need to be transmitted to a recipient who knows the probabilities of the results. This is also called the **description length** of X.

***Random Variable**: Number of heads when tossing a coin 3 times*

X	0	1	2	3
Prob(X)	1/8	3/8	3/8	1/8
$-\log_2 \mathrm{P}(X)$	3	1.415	1.415	3
$-\log_e \mathrm{P}(X)$	2.079	0.980	0.980	2.079
$-\log_{10} \mathrm{P}(X)$	0.903	0.426	0.426	0.903

*If the logarithm is base 2, the unit of information is **bits**, base e is **nats** and base 10 **hartleys***

Entropy

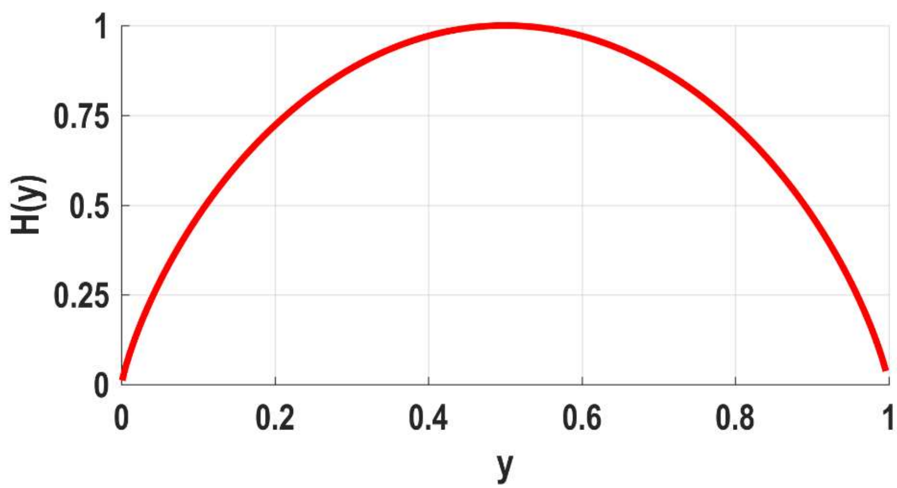
A standard way to measure **uncertainty of a random variable** is to use **entropy**

$$H(Y) = -\sum_{y} P(Y = y)\log_2 P(Y = y)$$

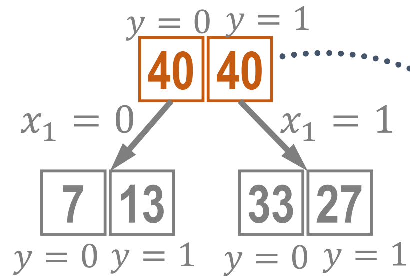
- Note that the entropy is computed by **summing over all the events**/outcomes/ states of the random variable.
- Entropy is **maximized for uniform distributions**, where the probability of all outcomes is equal *(is this what we want?)*
- Entropy is **minimized for distributions that place all their probability on a single outcome** *(or is this what we want?)*

The entropy of (binary) label distributions can be computed as:

$$H(y) = -P(y=0)\log_2 P(y=0) - P(y=1)\log_2 P(y=1)$$



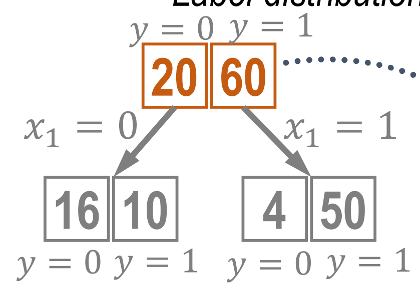
Uniform label distribution, where all outcomes have the same probability



$P(y=0)$ $P(y=1)$: $\frac{40}{80}$ $\frac{40}{80}$

$$H(y) = -\frac{40}{80}\log_2\frac{40}{80} - \frac{40}{80}\log_2\frac{40}{80} = 1$$

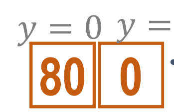
Label distribution in between the two extreme cases above and below



$P(y=0)$ $P(y=1)$: $\frac{20}{80}$ $\frac{60}{80}$

$$H(y) = -\frac{20}{80}\log_2\frac{20}{80} - \frac{60}{80}\log_2\frac{60}{80} = 0.81$$

Label distribution that places all its probability on a single outcome



this will be a leaf node as there isn't anything left to split on

$P(y=0)$ $P(y=1)$: $\frac{80}{80}$ $\frac{0}{80}$

$$H(y) = -\frac{80}{80}\log_2\frac{80}{80} - \frac{0}{80}\log_2\frac{0}{80} = 0$$

we use the convention that $0 \cdot \log_2 0 = 0$

Conditional Entropy and Mutual Information

Entropy can also be computed when **conditioned** on another variable:

$$H(Y|X) = -\sum_{x} P(X = x) \sum_{y} P(Y = y \mid X = x)\ \log_2 (Y = y \mid X = x)$$

This is called **conditional entropy** and is the amount of information needed to quantify the random variable Y **given** the random variable X. The **mutual information** or **information gain** between two random variables is

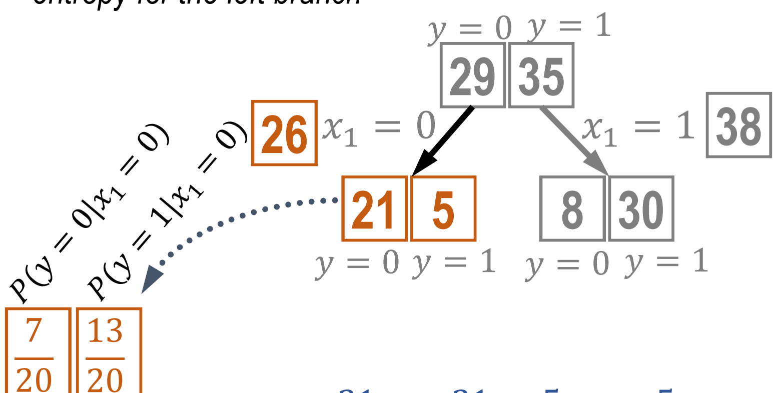
$$I(X, Y) = H(Y) - H(Y|X)$$

This is the amount of information we learn about Y by **knowing the value of** X and vice-versa (it is symmetric).

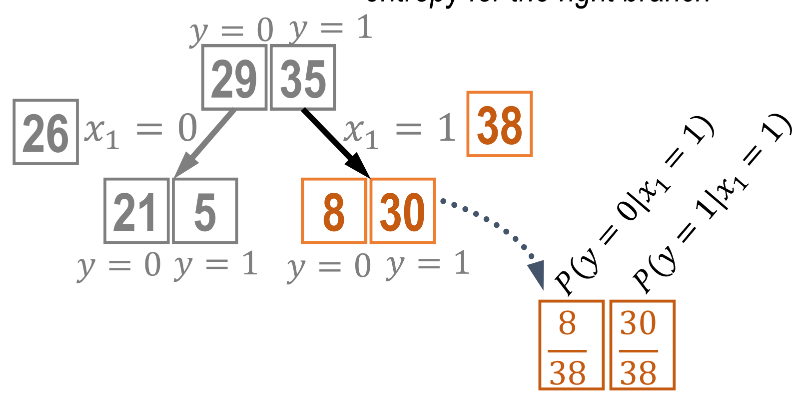
$$H(y) = -\frac{29}{64}\log_2\frac{29}{64} - \frac{35}{64}\log_2\frac{35}{64} = \mathbf{0.99}$$

*entropy **before** knowing the value of x_1*

entropy for the left branch



entropy for the right branch



$$H(y|x_1 = 0) = -\frac{21}{26}\log_2\frac{21}{26} - \frac{5}{26}\log_2\frac{5}{26} = 0.71$$

$$H(y|x_1 = 1) = -\frac{8}{38}\log_2\frac{8}{38} - \frac{30}{38}\log_2\frac{30}{38} = \mathbf{0.74}$$

$$H(y|x_1) = P(x_1 = 0)H(y|x_1 = 0) + P(x_1 = 1)H(y|x_1 = 1) = \frac{26}{64}\cdot(0.71) + \frac{38}{64}\cdot(0.74) = \mathbf{0.73}$$

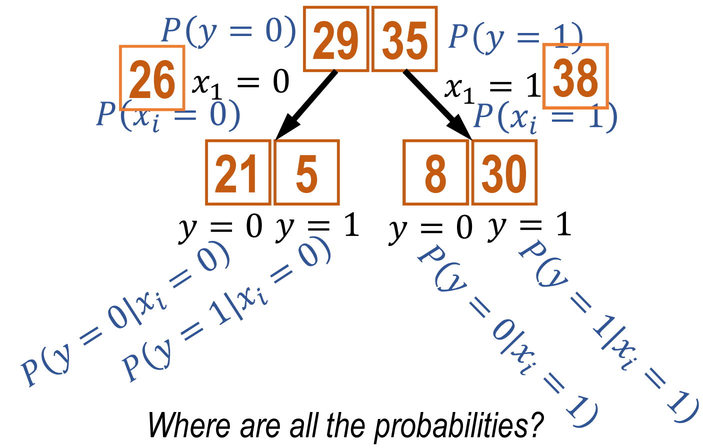
*entropy **after** knowing the value of x_1*

$$I(x_1, y) = H(y) - (y|x_1) = 0.99 - 0.73 = \mathbf{0.26}$$

***information gained** by knowing the value of x_1*
***larger information gain** corresponds to **less uncertainty** about y (labels) given x_1 (feature)*

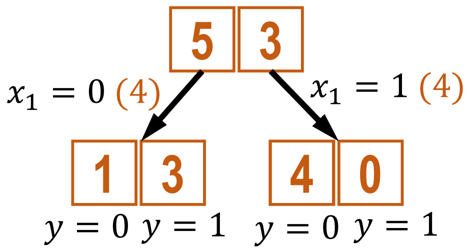
Choosing the Best Feature

split on x_i

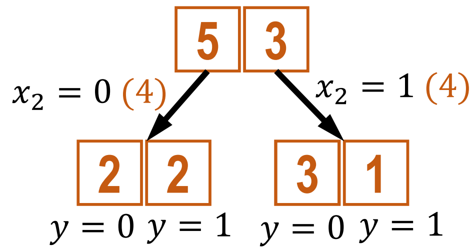


Step 1: Count the various combinations of features and labels

split on x_1



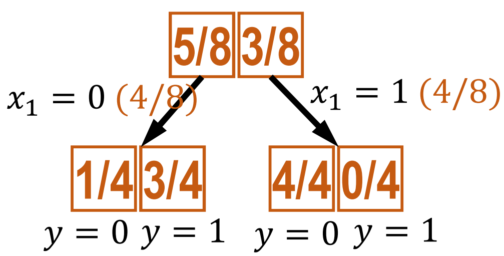
split on x_2



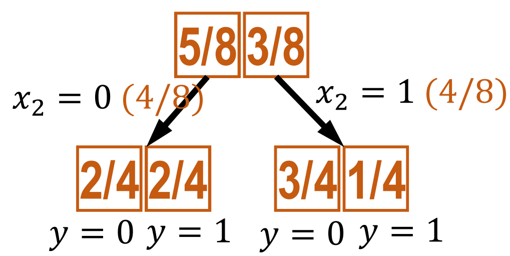
x_1	x_2	y
1	1	0 (+)
1	0	0 (+)
1	1	0 (+)
1	0	0 (+)
0	1	0 (+)
0	0	1 (-)
0	1	1 (-)
0	0	1 (-)

Step 2: Convert to probabilities

split on x_1

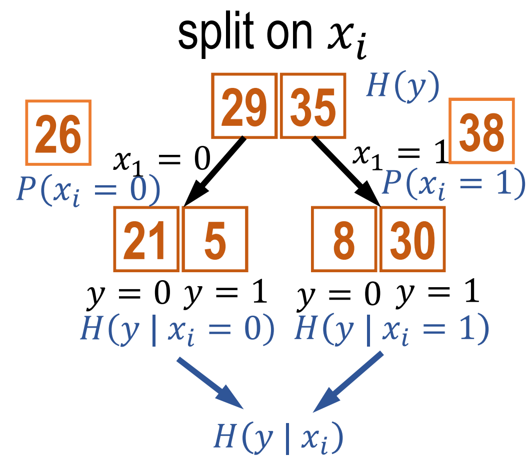


split on x_2



Choosing the Best Feature

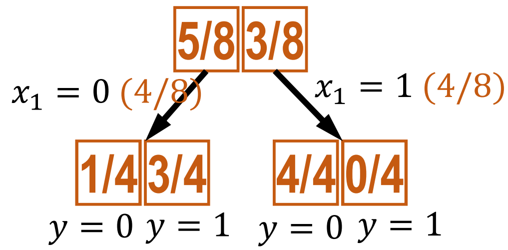
Step 3: Compute information gain for both splits and pick the variable with the biggest gain



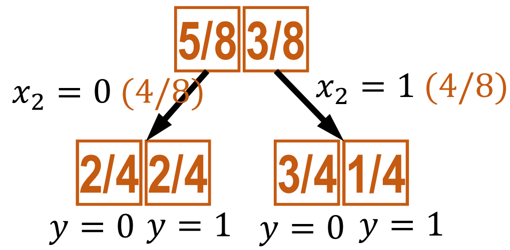
Where are all the entropies?

$$H(y) = -\frac{5}{8}\log\frac{5}{8} - \frac{3}{8}\log\frac{3}{8}$$

split on x_1



split on x_2



$$H(y|x_1 = 0) = -\frac{1}{4}\log\frac{1}{4} - \frac{3}{4}\log\frac{3}{4}$$

$$H(y|x_1 = 1) = -1\log 1 - 0\log 0$$

$$H(y|x_2 = 0) = -\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4}$$

$$H(y|x_2 = 1) = -\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4}$$

$$H(y \mid x_1) = -\frac{4}{8}H(y|x_1 = 0) - \frac{4}{8}H(y|x_1 = 1)$$

$$H(y \mid x_2) = -\frac{4}{8}H(y|x_2 = 0) - \frac{4}{8}H(y|x_2 = 1)$$

$$I(x_1, y) = H(y) - H(y \mid x_1)$$

$$I(x_2, y) = H(y) - H(y \mid x_2)$$

x_1	x_2	y
1	1	0 (+)
1	0	0 (+)
1	1	0 (+)
1	0	0 (+)
0	1	0 (+)
0	0	1 (-)
0	1	1 (-)
0	0	1 (-)

$I(x_1, y) > I(x_2, y) \Rightarrow$ pick feature x_1 next

The ID3 Algorithm

The ID3 (Iterative Dichotomizer) and its successor, C4.5 were developed by Ross Quinlan in the early to mid 1980s and are widely considered to be a landmark machine learning algorithms, and until at least 2008, were the #1 data mining tool.

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
    - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
    - The decision attribute for *Root* $\leftarrow A$
    - For each possible value, v_i, of A,
        - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
        - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
        - If $Examples_{v_i}$ is empty
            - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
            - Else below this new branch add the subtree
              ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* $- \{A\}$))
- End
- Return *Root*

Some Final Details

When do we terminate?

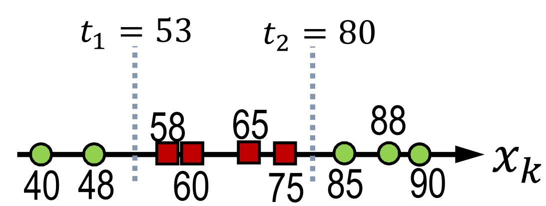
- If the current set is **"pure"** (i.e., has a single label in the output), stop
- If you **run out of attributes to recurse on**, even if the current data set isn't pure, stop and use a majority vote
- If a partition contains no data points, use the majority vote at its parent in the tree
- If a partition contains no data items, nothing to recurse on
- For fixed depth decision trees, the **final label is determined by majority vote**

How do we handle real-valued features?

- For continuous attributes, use threshold splits
- Split the tree into $x_k < t$ and $x_k \geq t$
- Can split on the same attribute multiple times on the same path down the tree

How do we select the splitting threshold?

- Sort the values of feature x_k
- Identify a finite number of feature transitions
- Calculate thresholds in between transitions
- How do we select which split to insert as a node?

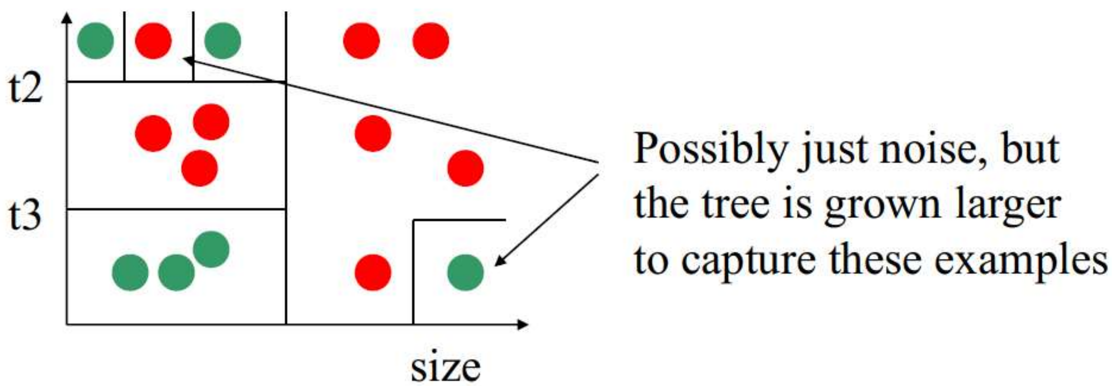


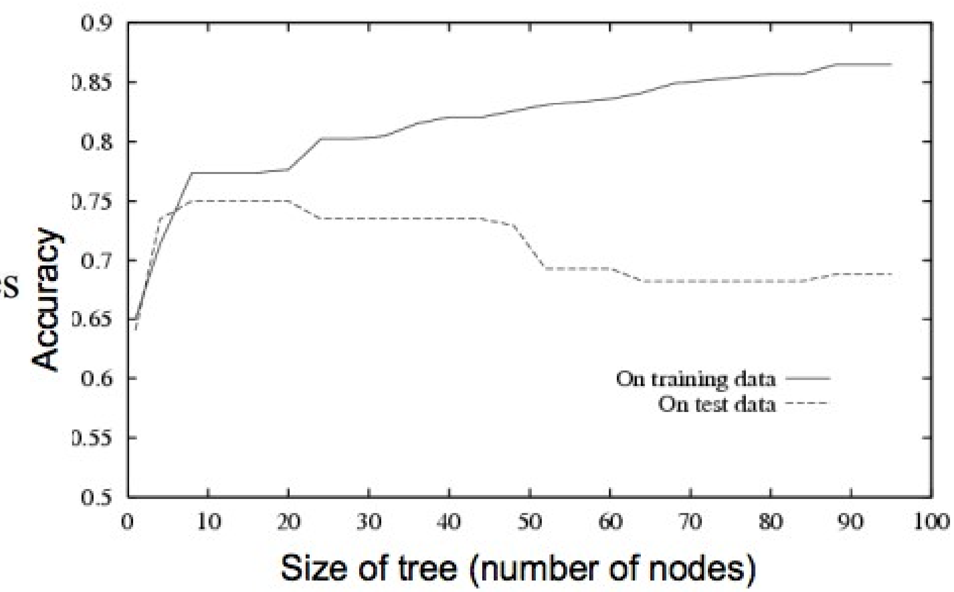
Overfitting in Decision Trees

Hypothesis space is complete! *Target function is surely in there; but ID3 search is incomplete*
No back tracking; *Greedy thus local minima*
Statistics-based search choices; *Robust to noisy data*
Inductive bias: heuristically prefers shorter trees, trees that place attributes with highest information gain closest to the root are preferred

Decision trees will always overfit!

- It is always possible to obtain zero training error on the input data with a deep enough tree (if there is no noise in the labels)
- Random noise in the training examples also leads to overfitting





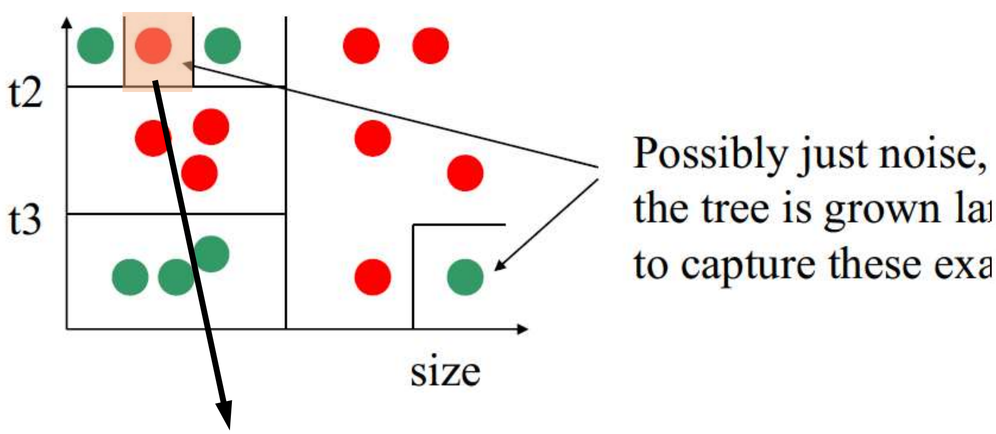
Avoiding Overfitting in Decision Trees

Pre-pruning/early stopping before overfitting

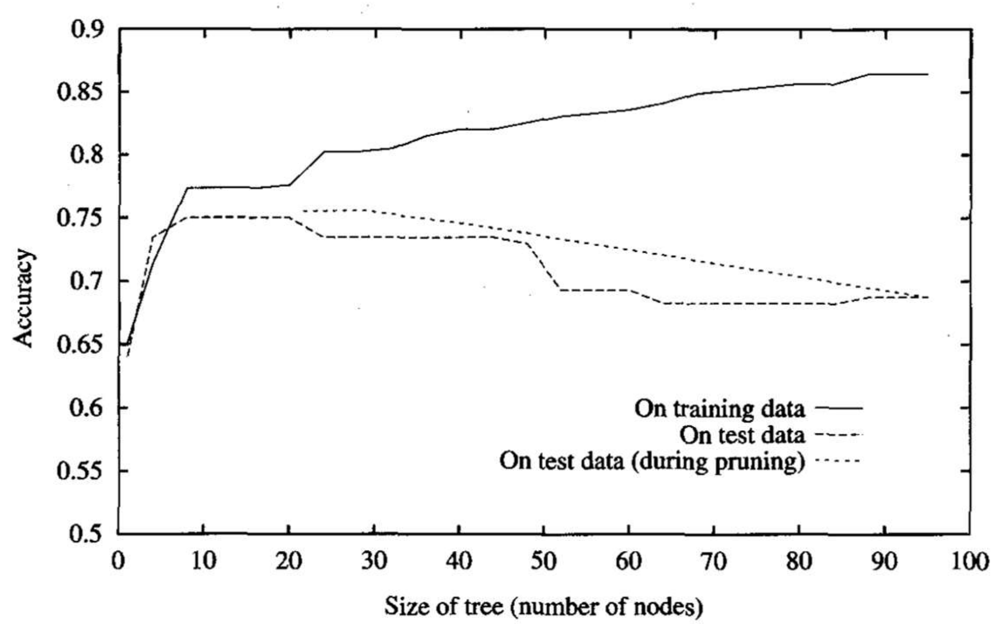
- Typical stopping criterion
  - No error (if all instances belong to same class)
  - If all the attribute values are same
- More restrictive conditions
  - Stop growing when **data split is not statistically significant** (example using chi-square test)
  - Stop if the **number of instances** is small
  - Stop if expanding does **not significantly improve measures** (information gain)
- Hard to determine if we are actually overfitting

Post-pruning after allowing a tree to overfit

- Separate data into training and validation sets
- Evaluate impact on validation set **when a node is "pruned"**
- **Greedily remove** node that improves performance the most
- Produces smallest version of most accurate subtree
- Typically use minimum description length (MDL) for post-pruning
- Highly successful empirically



leaf node added due to noise in the training set most-likely to be pruned;
pruning this node reduces accuracy on training set, increases accuracy on validation set



Some Post-pruning Methods

Reduced-Error Pruning

- Use a **validation set** (**tuning**) to identify errors at every node
- Prune node with highest reduction in error
- Repeat until error no longer reduces
- **Con:** requires a large amount of data to create a validation set

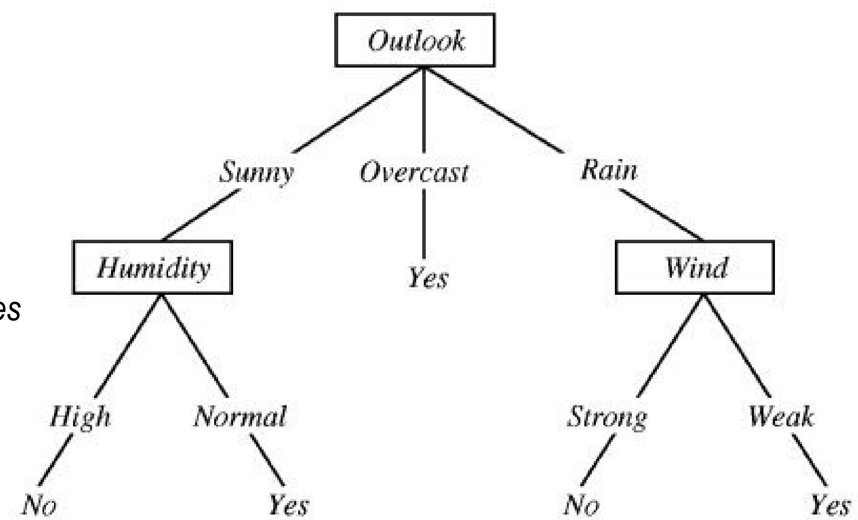
Pessimistic Pruning

- No validation set, use a training set
- Error estimate at every node is conservative based on training examples
- **Con**: Heuristic estimate, not statistically valid

Rule-post Pruning

- Convert tree to equivalent set of rules
- Prune **each rule** independently of others by **removing pre-conditions that improve rule accuracy**
- Sort final rules into desired sequence

IF *(Outlook = Sunny* ***AND*** *Humidity = High)* ***THEN*** *PlayTennis= No*
IF *(Outlook = Sunny* ***AND*** *Humidity = Normal)* ***THEN*** *PlayTennis= Yes*
IF *(Outlook = Overcast)* ***THEN*** *PlayTennis= Yes*
IF *(Outlook = Rain* ***AND*** *Wind = Strong)* ***THEN*** *PlayTennis= No*
IF *(Outlook = Rain* ***AND*** *Wind = Weak)* ***THEN*** *PlayTennis= Yes*



Decision Trees

- **Decision Trees** – popular and a very efficient hypothesis space
  - Variable size: Any Boolean function can be represented
  - Handles discrete and continuous features
  - Handles classification **and regression**
  - Easy to implement
  - Easy to use
  - Computationally cheap

- Constructive **heuristic** search: built top-down by adding nodes
- **Decision trees will overfit!**
  - zero bias classifier (no mistakes) = large variance
  - must use tricks to find simpler trees
    - early stopping, pruning etc.