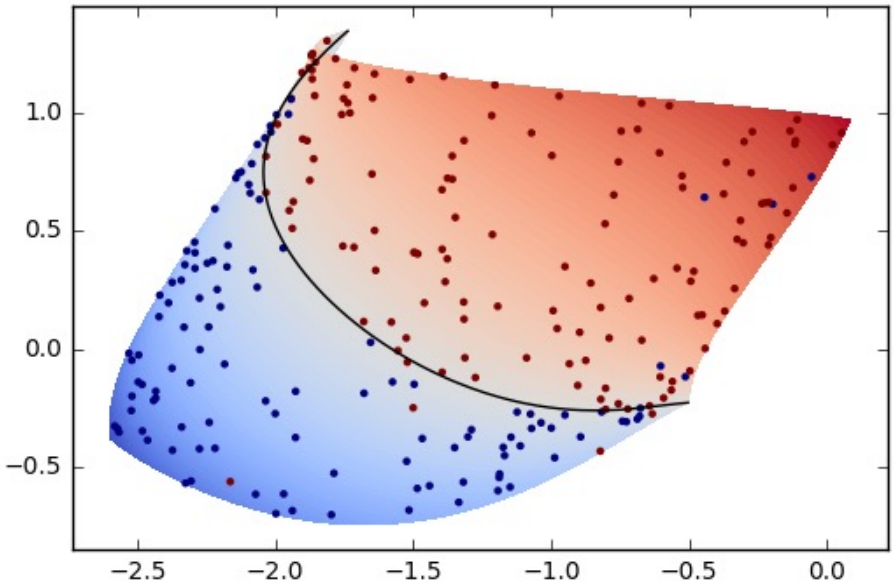
# Lecture 18: Decision Trees




CS178 Spring 2023

Gavin Kerrigan

Some slides adapted from Padhraic Smyth, Alex Ihler

# Announcements/Reminders

- HW3 due tonight at midnight
- Project team formation due in 1 week (Friday 5/19)
  - Start thinking about working on project soon (due 6/12)
- HW4 released by Monday
  - Decision Trees (today's lecture)
- Midterm exam graded
  - Look out for an announcement soon

Decision Tree Classifiers

Real-Valued Features

Generalizations

Learning Decision Trees

# A Decision Tree Classifier

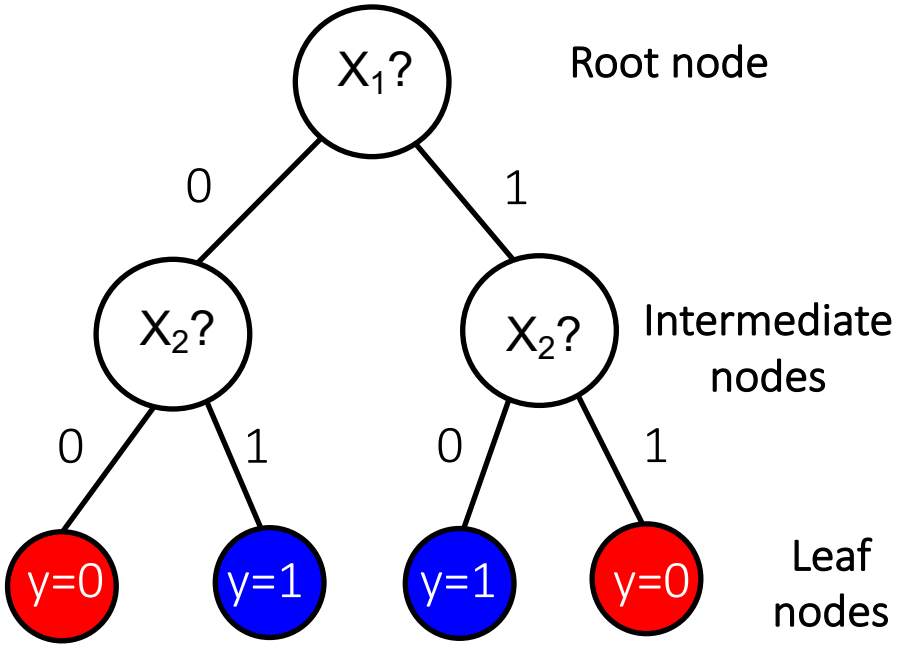
A simple binary classification problem

XOR Dataset

| $x_1$ | $x_2$ | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$y = XOR(x_1, x_2)$

Can represent a Boolean function as a decision tree




The y prediction at each leaf node depends on the path to that node

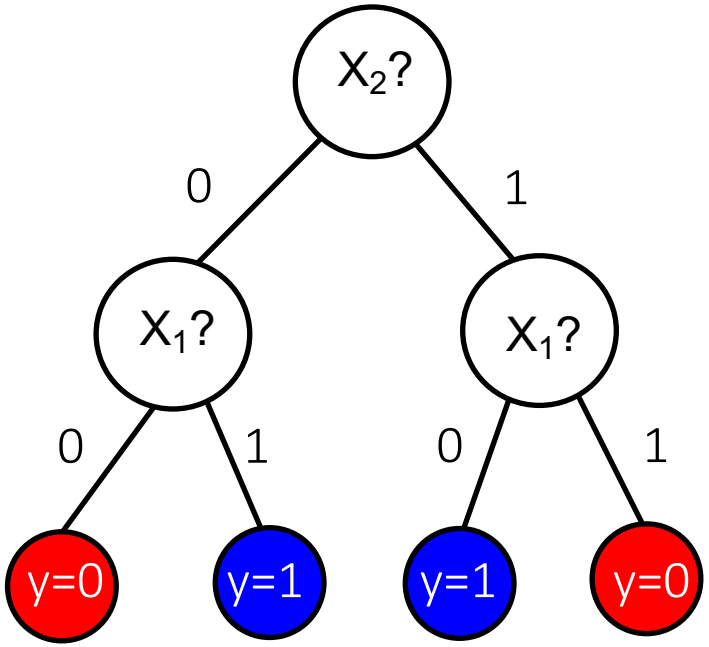
# A Decision Tree Classifier

A different decision tree representation for XOR

XOR Dataset

| $x_1$ | $x_2$ | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$y = XOR(x_1, x_2)$



# Adding More Variables (Features)..

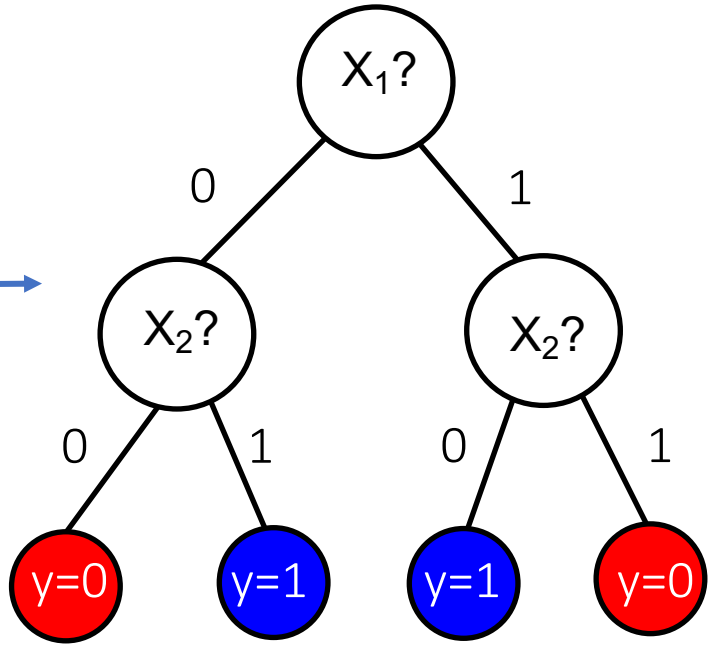
Say we have 3 extra binary features….

XOR Dataset

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | y |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 |

$y = XOR(x_1, x_2)$

Same decision tree



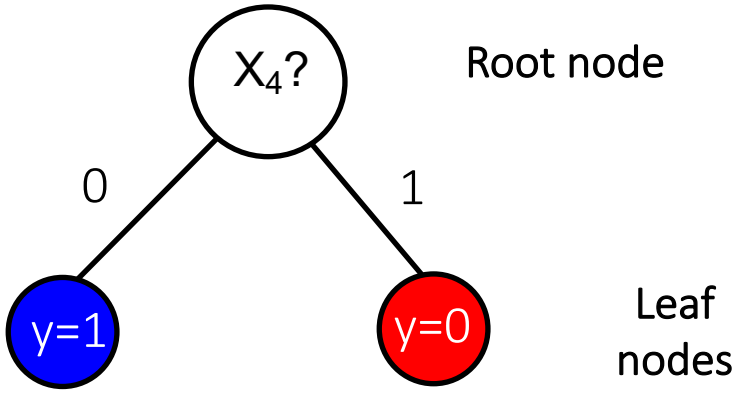

…….in principle we can represent the function compactly if we don't need all features to predict y

# An even Simpler Tree

XOR Dataset

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | y |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 |

Simpler decision tree



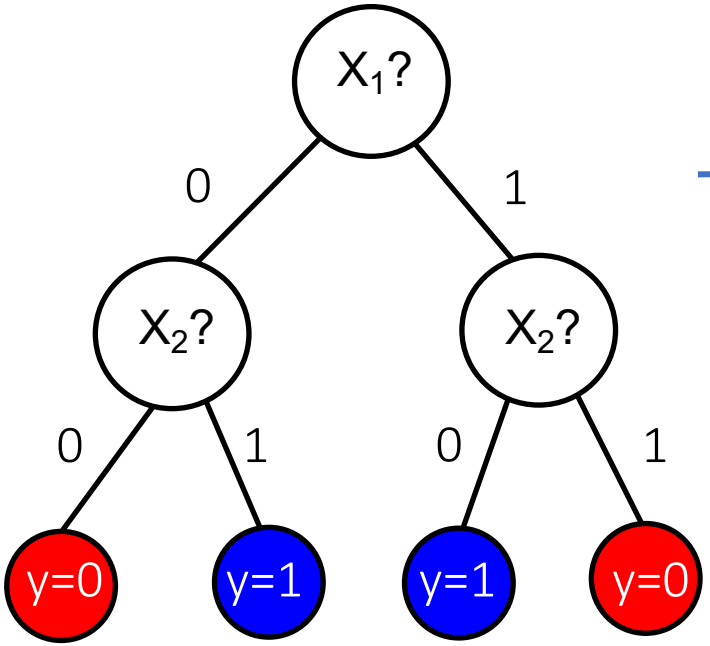

…….a natural question is: can we learn a minimal tree structure, e.g., if we had 1000's of rows and 100's of features?

# Decision Tree representation of Boolean Functions

Equivalent to a Disjunctive Normal Form (DNF) representation:

Each path corresponds to AND statements for propositions along the path

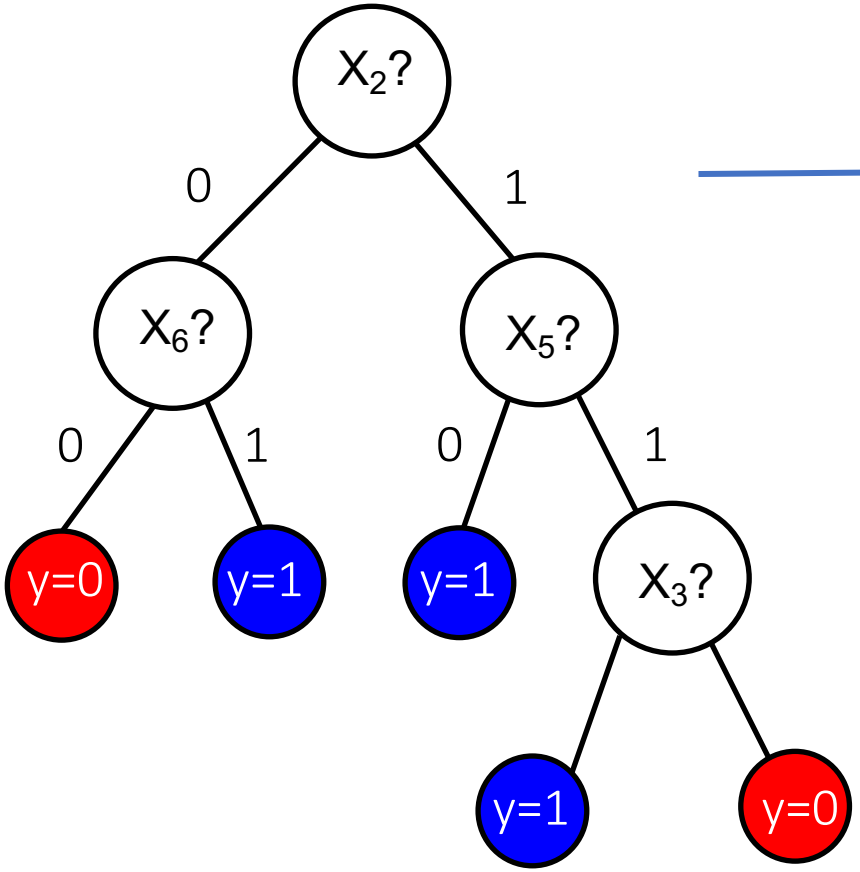
The overall function is the OR of the different positive paths in the tree




( NOT($X_1$) AND $X_2$ )
OR
( $X_1$ AND NOT( $X_2$ ) )

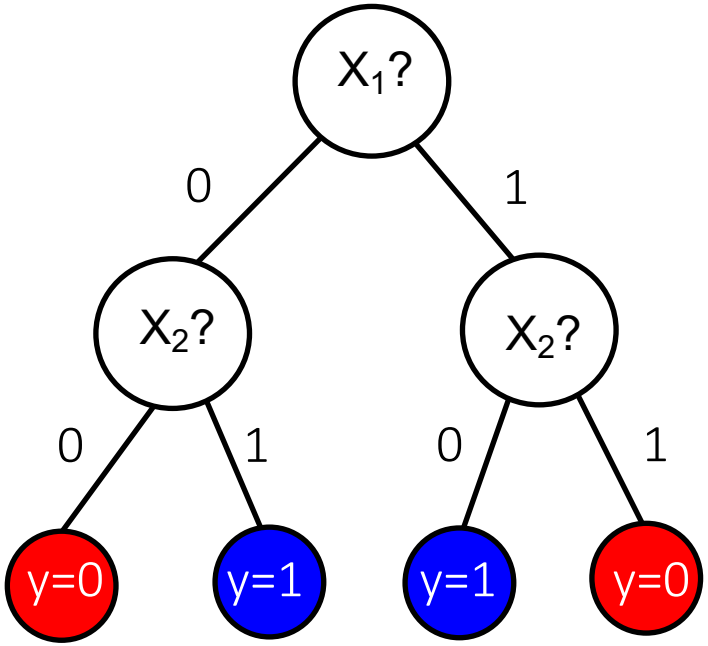
Tree is a disjunction of conjunctions where each path is a conjunction

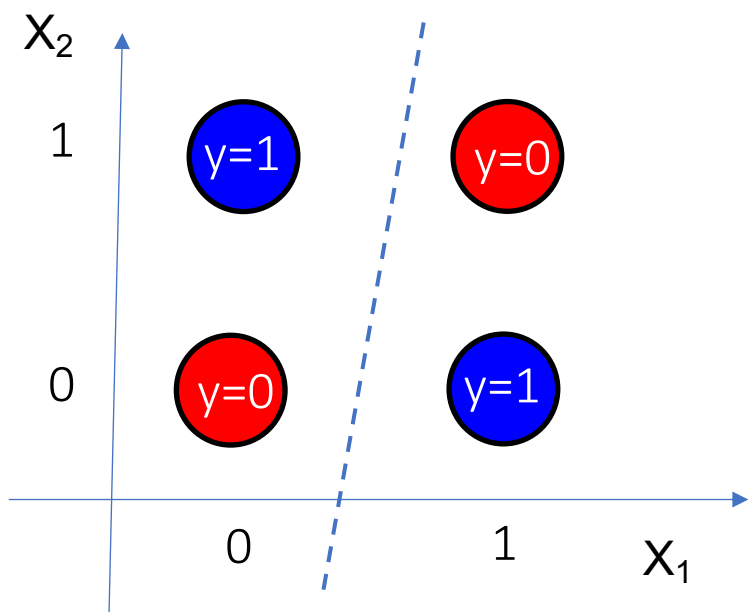
# Trees need not be Balanced




( NOT($X_2$) AND $X_6$ )
OR
( $X_2$ AND NOT($X_5$) )
OR
($X_2$ AND $X_5$ AND NOT($X_3$) )

# Geometric Representation of XOR




No linear function can represent XOR

But a decision tree can represent XOR
=> more flexible than linear functions

Questions?

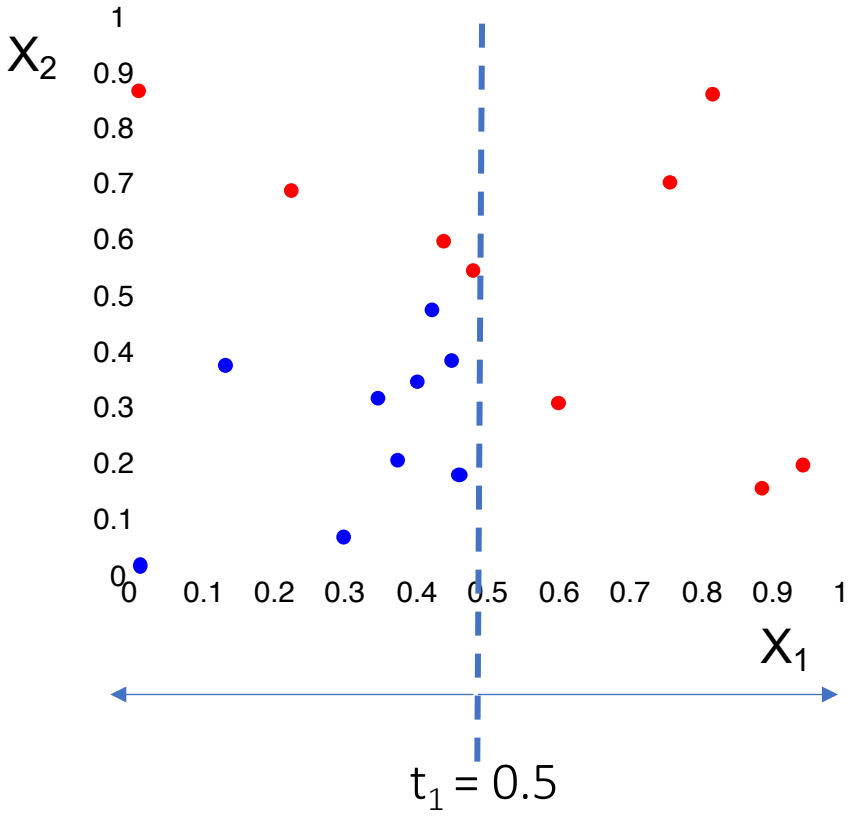
Decision Tree Classifiers

Real-Valued Features

Generalizations

Learning Decision Trees

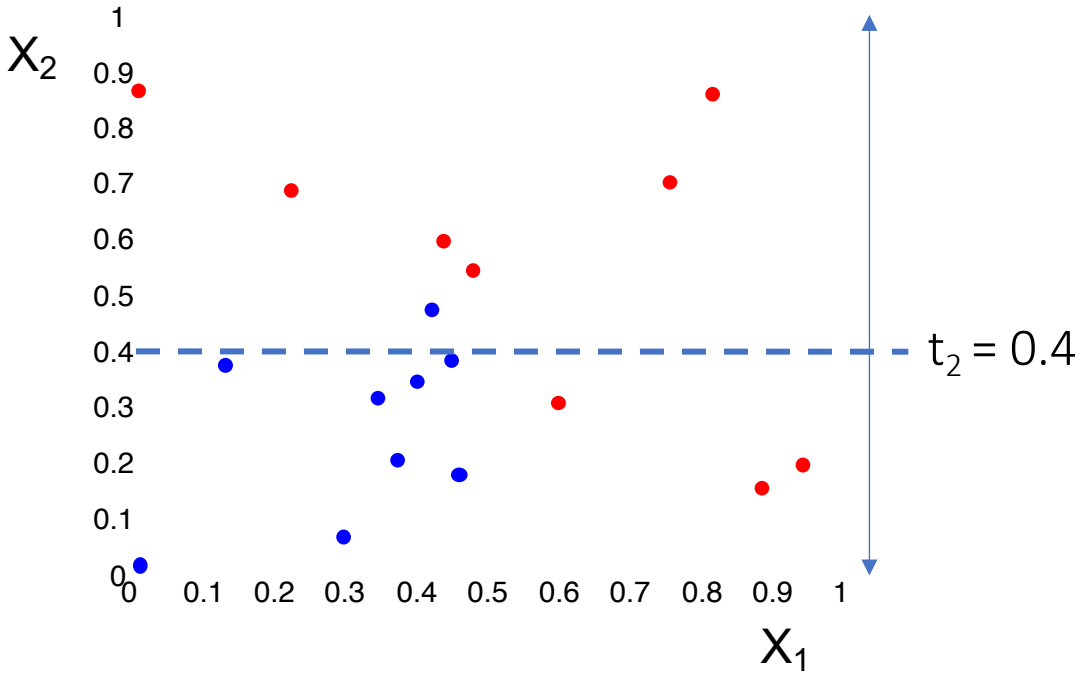
# What about Real-Valued Features?




**Key Idea:**
Convert to binary features, using thresholds

Example:
$X_1 > t_1$ ?

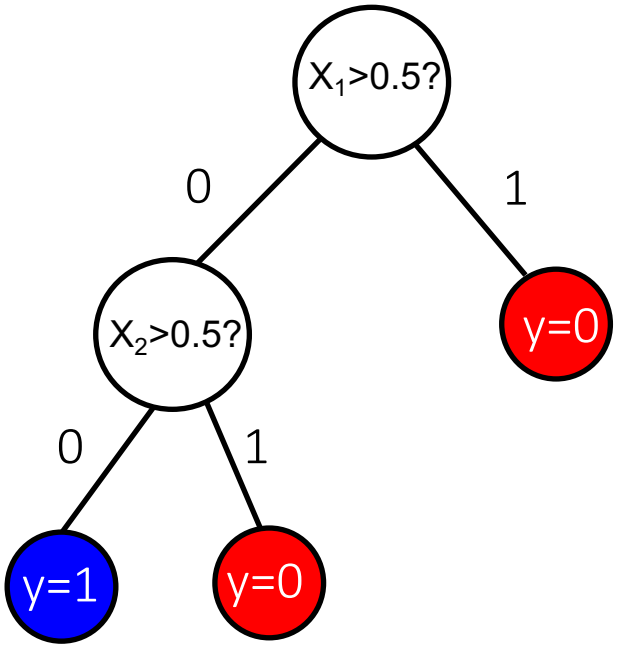
# What about Real-Valued Features?



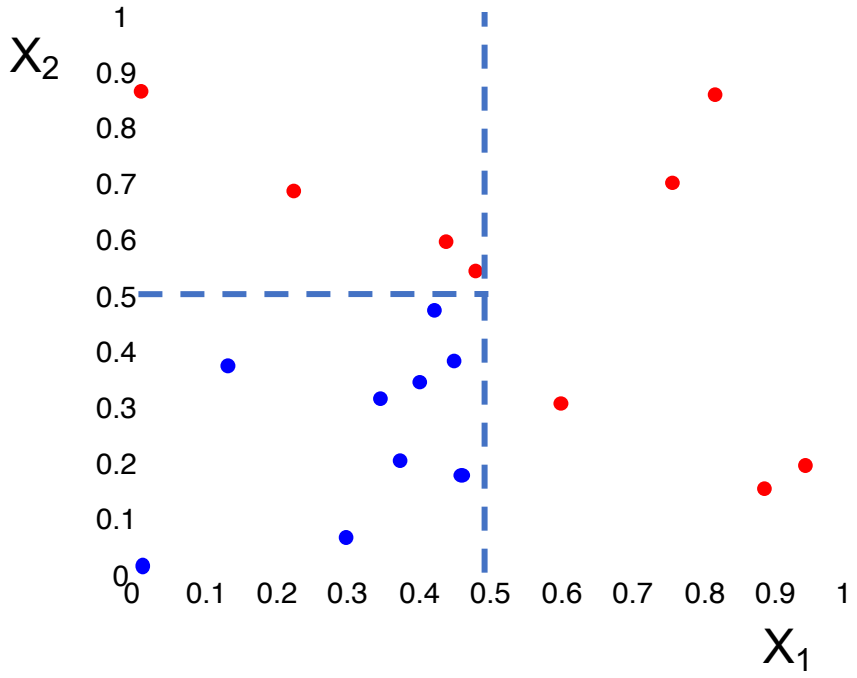

**Key Idea:**
Convert to binary features, using thresholds

Example:
$X_2 > t_2$ ?

# Decision Tree Model in 2 Dimensions

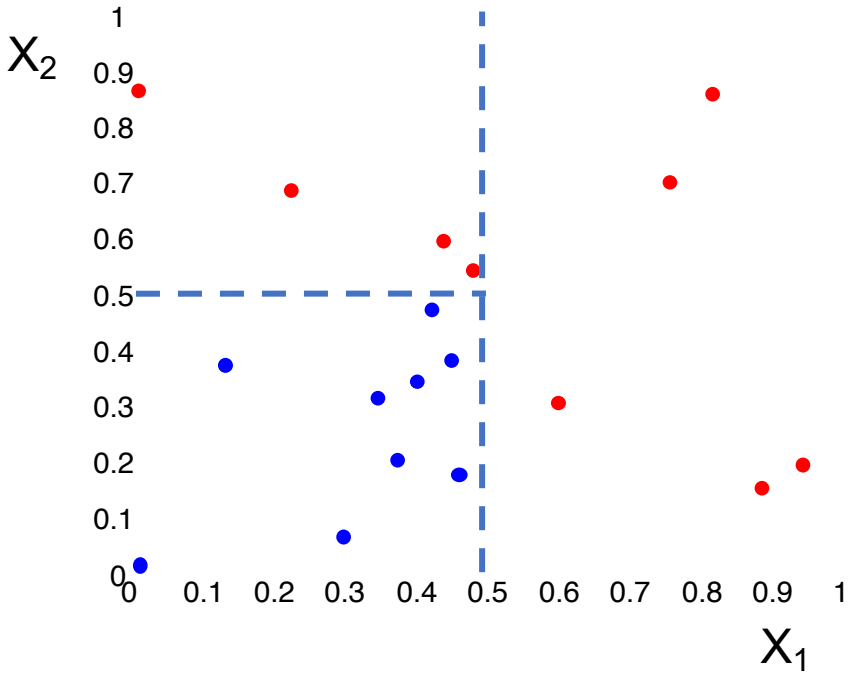


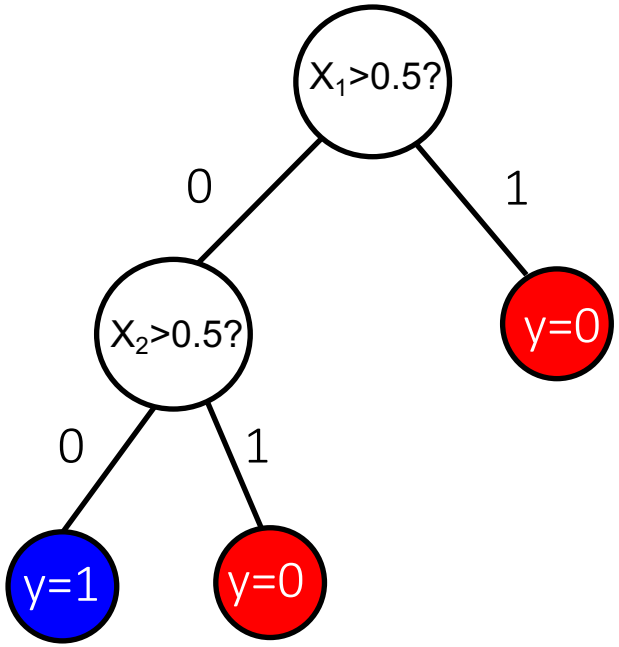

=>



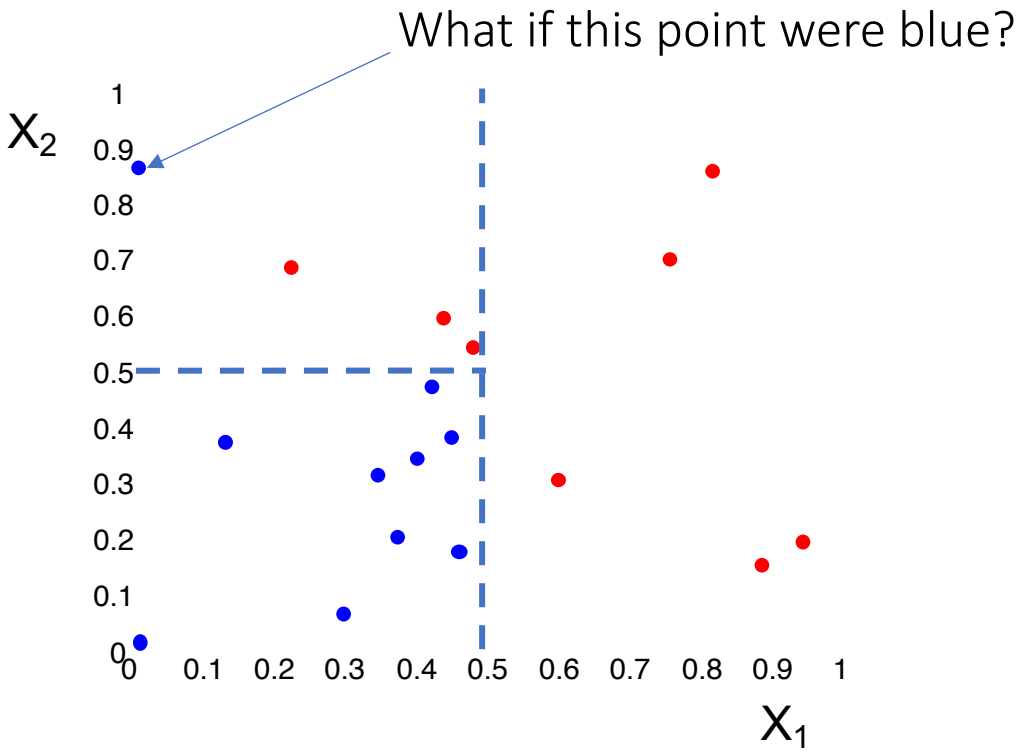

Decision tree defines a hierarchical partitioning of the feature space

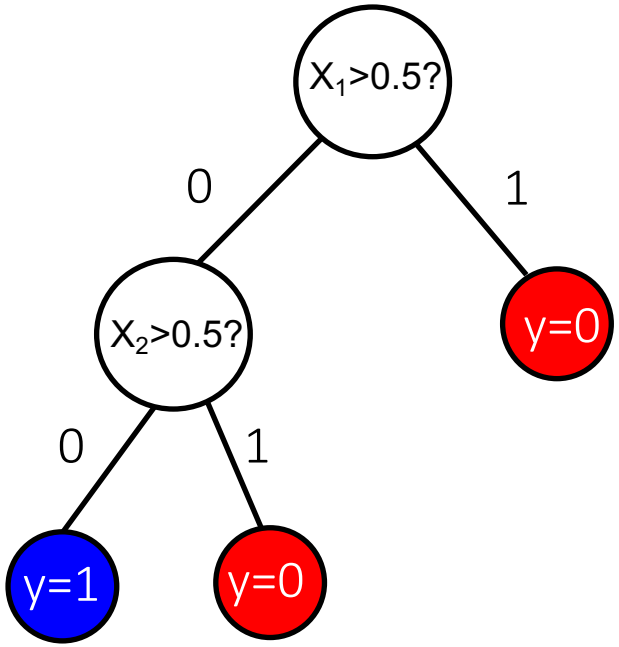
# Decision Tree Model in 2 Dimensions



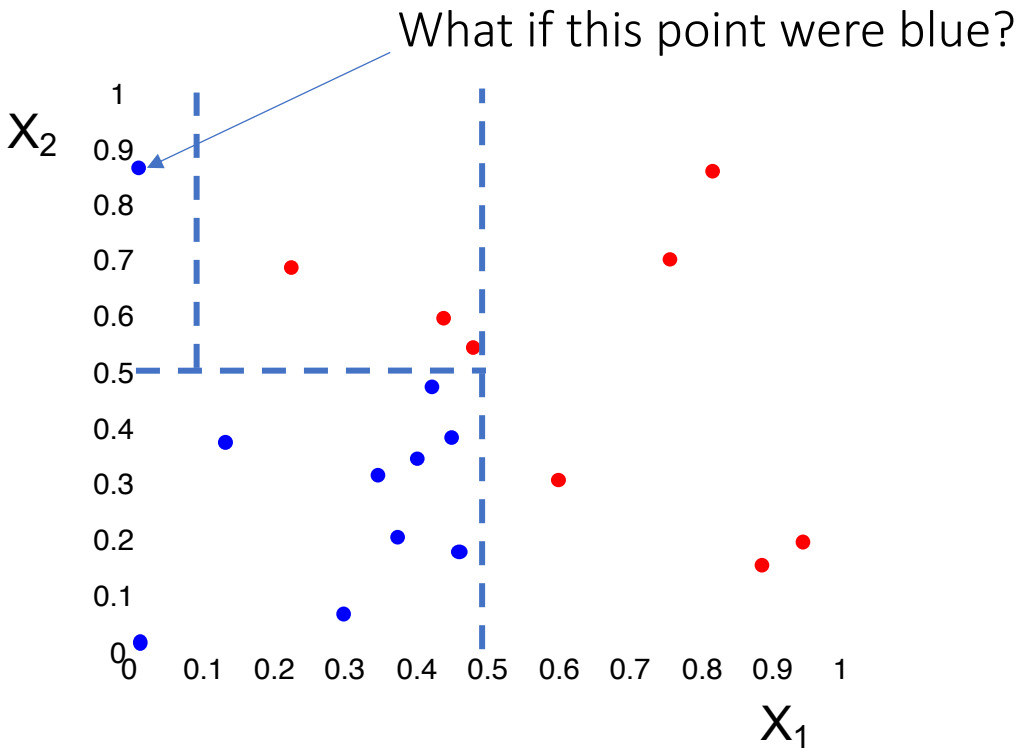



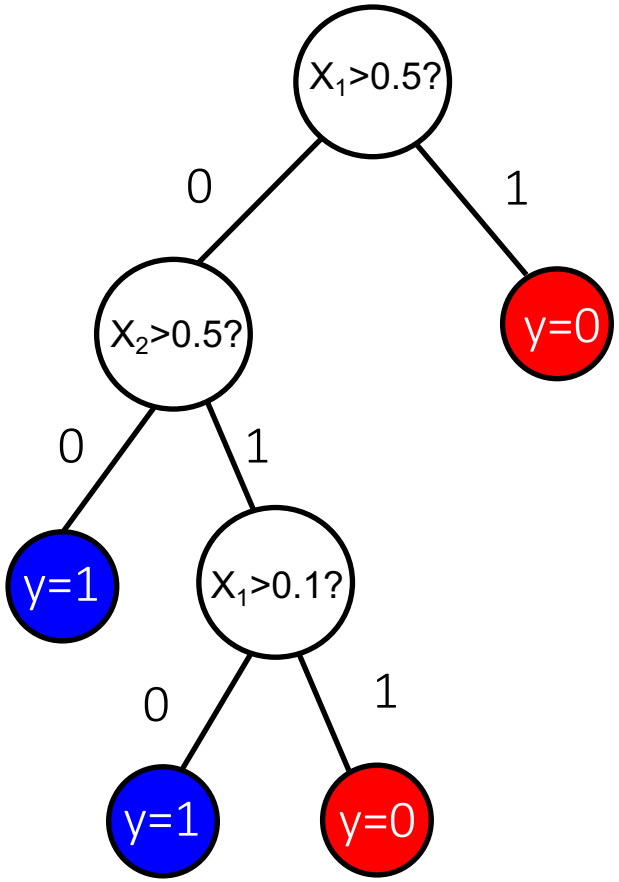
# Decision Tree Model in 2 Dimensions



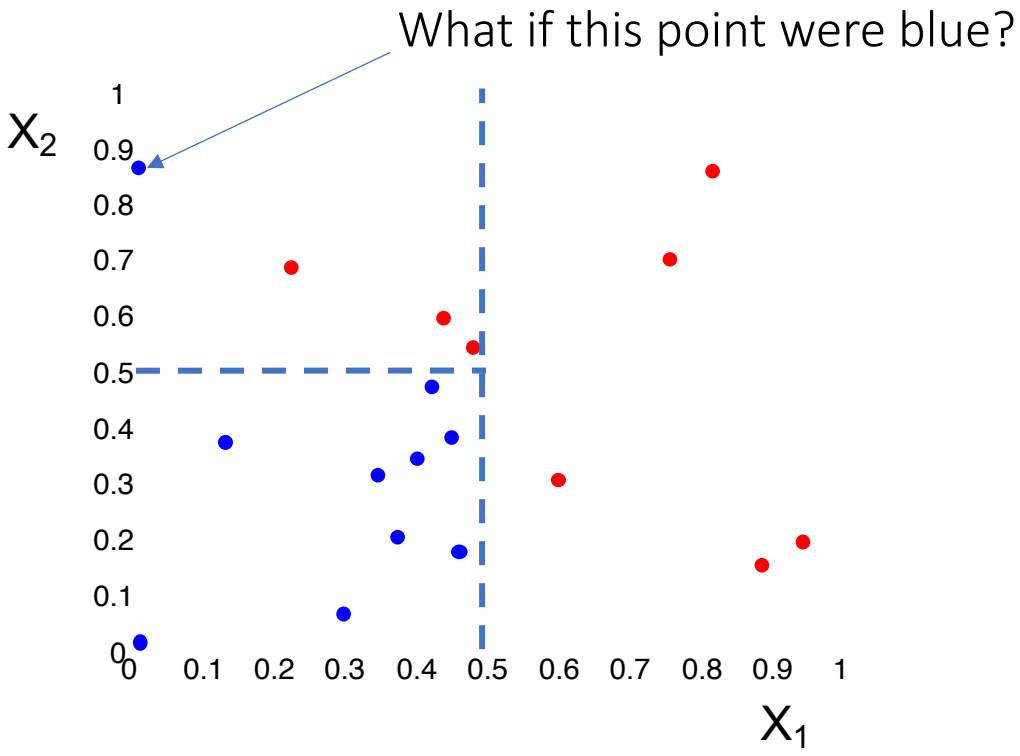
# Decision Tree Model in 2 Dimensions

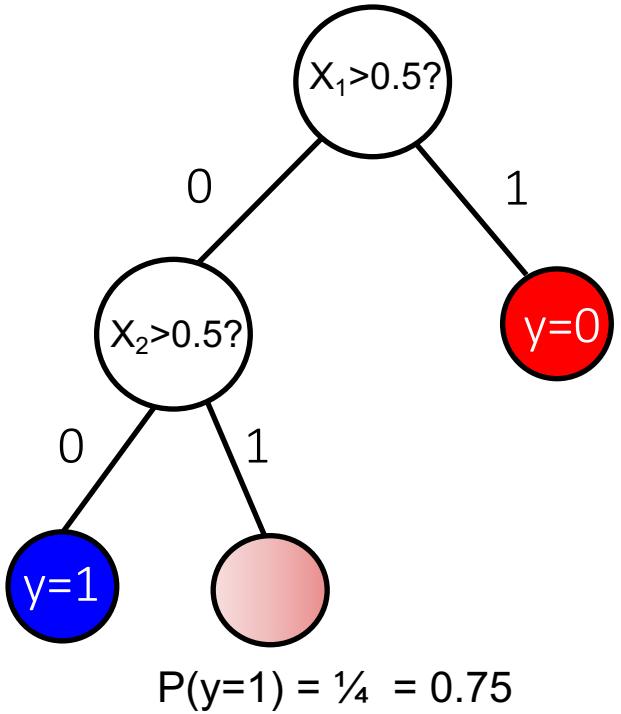



We could extend the tree

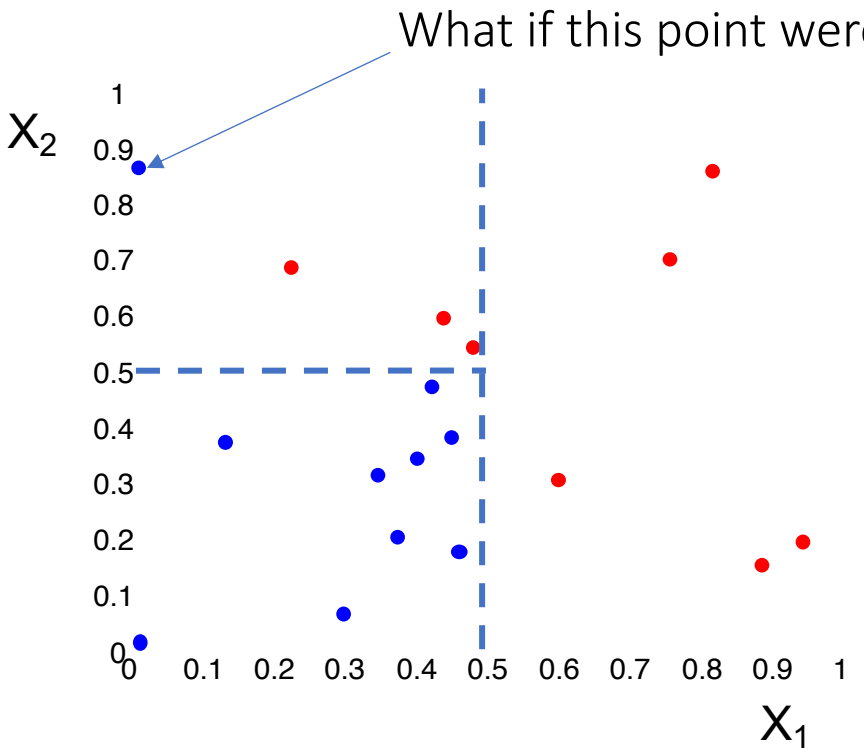
# Decision Tree Model in 2 Dimensions

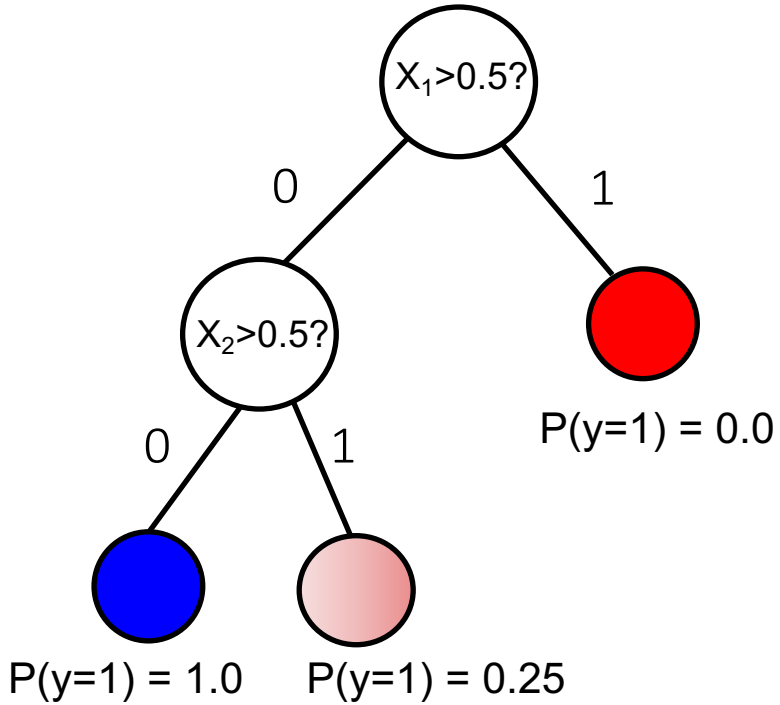



OR…. we could keep the smaller tree and predict P(y=1|path)

# Decision Tree Model in 2 Dimensions




More generally, can predict P(y|path) at any leaf (or node)

Questions?

Decision Tree Classifiers

Real-Valued Features

Generalizations

Learning Decision Trees

# Generalizing….

We can extend the decision tree model to arbitrary number of features d

Each internal node is binary:

$x_j > t$, where $x_j$ is some feature, t is some threshold

At each leaf node we can compute P(y = 1| path)

In principle the tree can grow until all leaf probabilities are 0 or 1,
i.e., a node is "pure" in terms of class membership

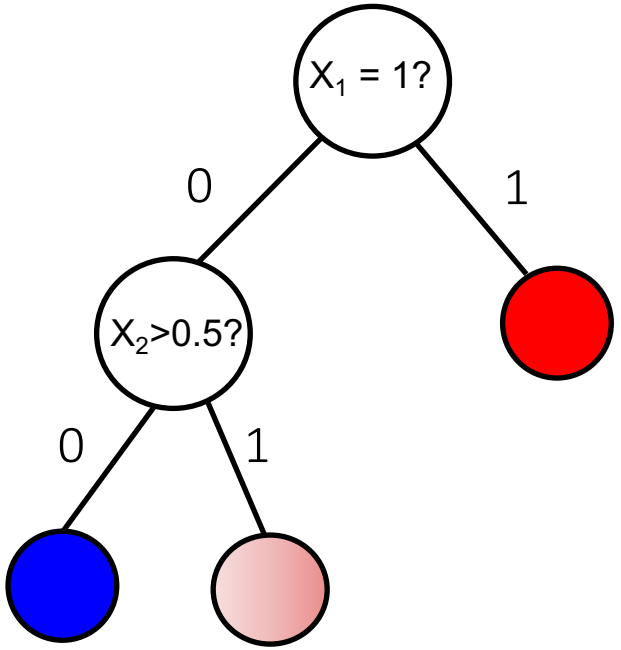
- Complexity of tree can be controlled by maximum allowed depth

Two questions remain:

1. How do we construct the tree from data?
2. How large should the tree be?

# Further Generalizing….

We can mix real-valued and binary features in our decision tree
e.g., $x_1$ is binary, $x_2$ is real-valued

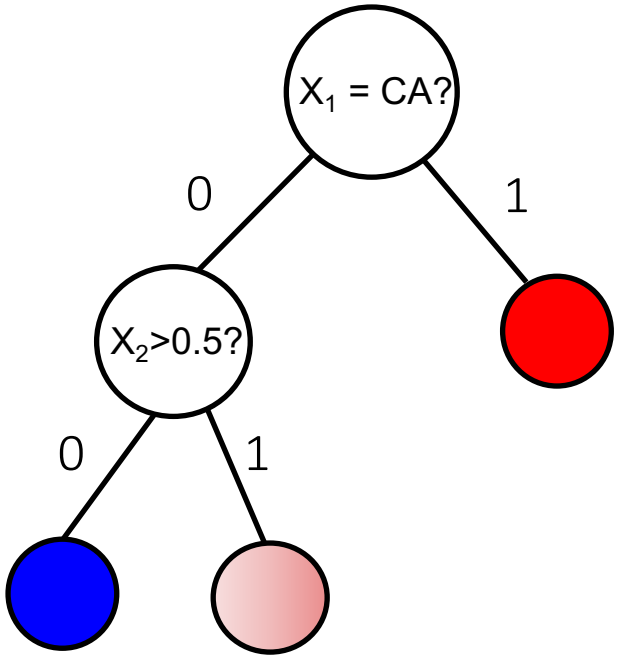


# Further Generalizing….

What about categorical variables?

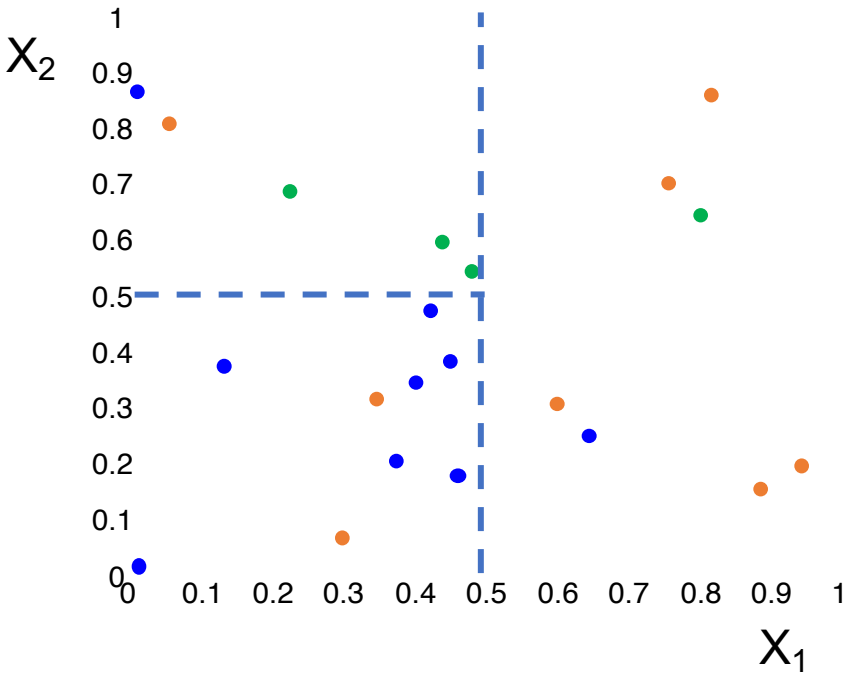
e.g., $X_1$ takes values {AL, AZ, CA, ….}

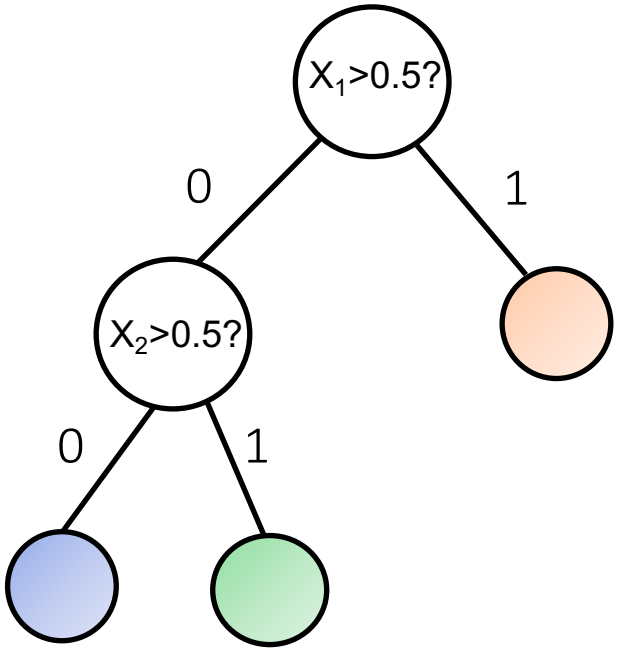
An easy way to handle this is to treat each possible value as its own binary variable, e.g., X=AL? X=AZ? X=CA? ….




**This is a major advantage of decision trees over other models: can easily handle a mix of variables (real, binary, categorical)**

# Generalizing to K>2 Classes

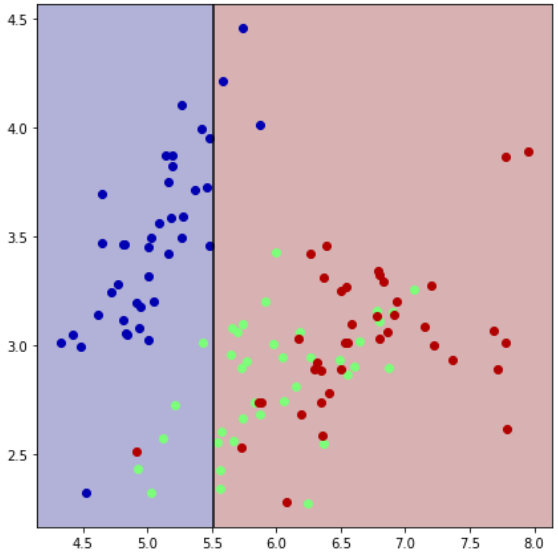


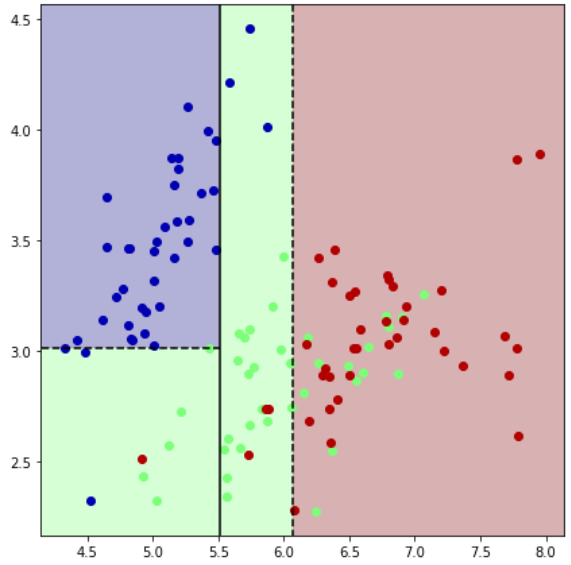




At each leaf node:
vector of probabilities = relative frequencies of class labels
e.g., top left region: P(y=green) = 0.6, P(y=blue) = 0.2, ….

# Examples of Trees of Different Complexities

Relatively Simple (Small) Trees



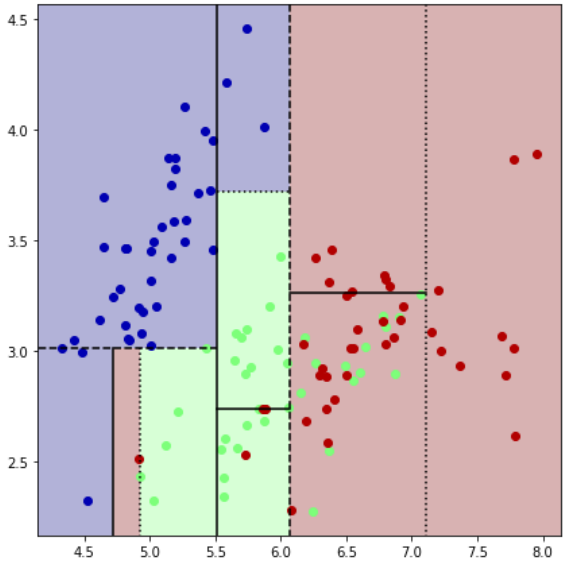



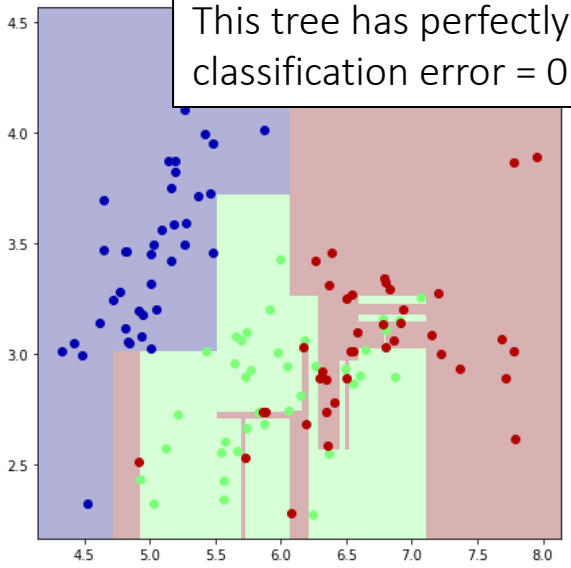

3 classes: shaded color = majority class at each leaf

2 features

# Examples of Trees of Different Complexities

## More Complex (Larger) Trees



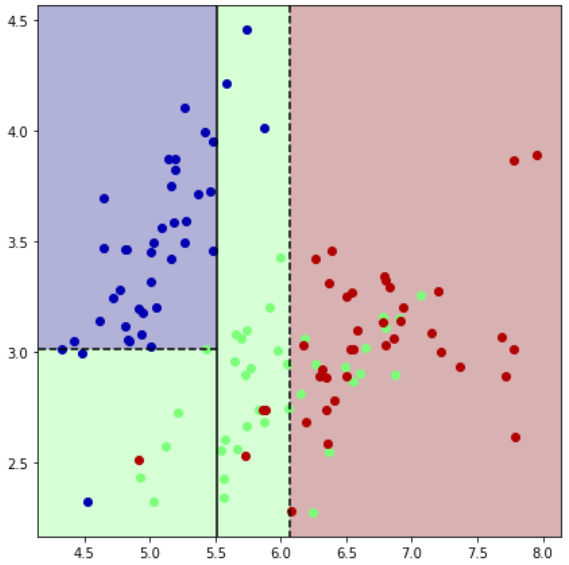

This tree has perfectly fit the data, classification error = 0 on training data

Complexity of tree model is (roughly) size of tree (number of leaves, number of internal nodes)

# Decision Trees are Piecewise Linear

Decision trees produce piecewise linear decision boundaries

Additional constraint: the boundaries are ”axis-parallel” in feature space

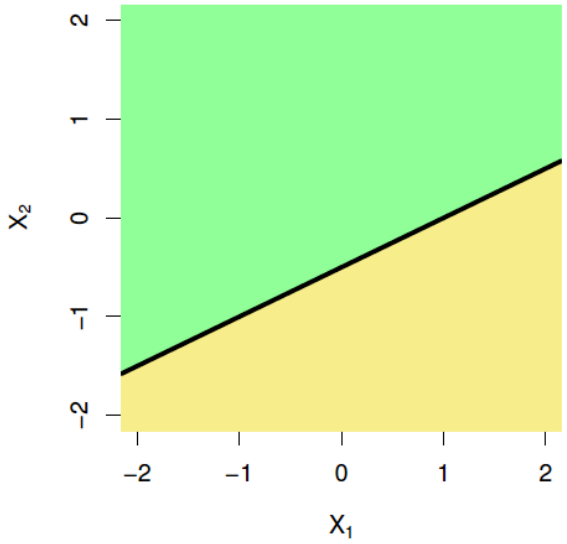


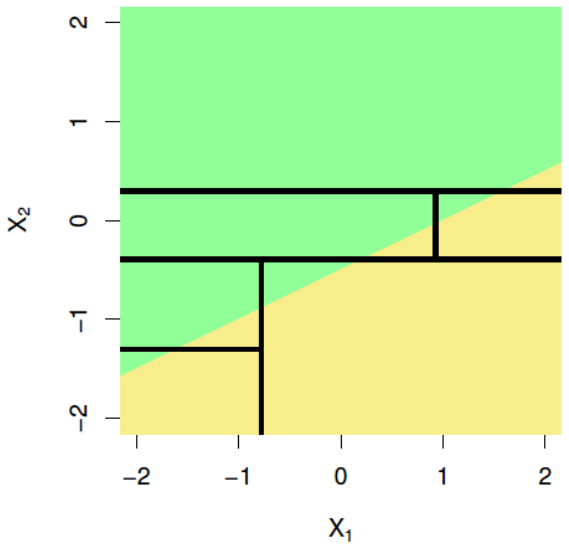

In principle they can approximate curved boundaries
…. but may require a large tree to do this accurately

## Linear Decision Boundaries

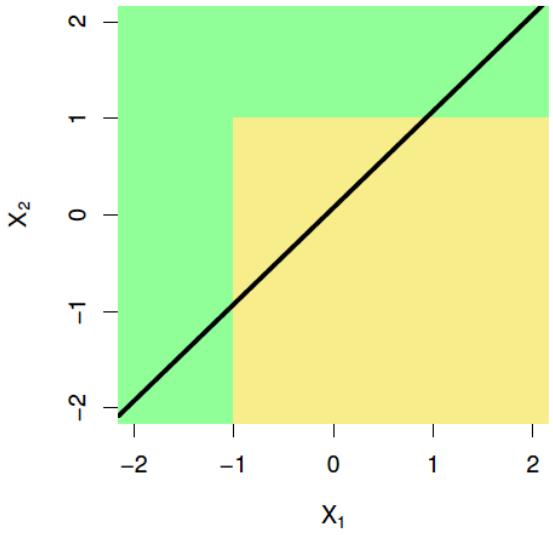
## Decision-Tree Boundaries

Problem 1




Problem 2



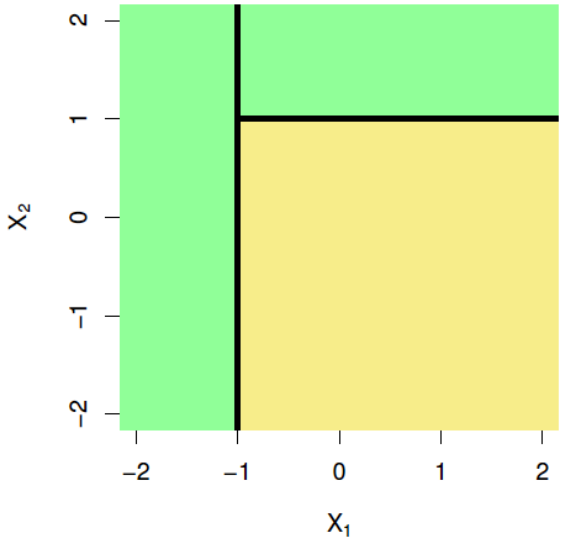




Figure from James et al, An Introduction to Statistical Learning, Chapter 8

# Decision Tree Classifiers as Functions

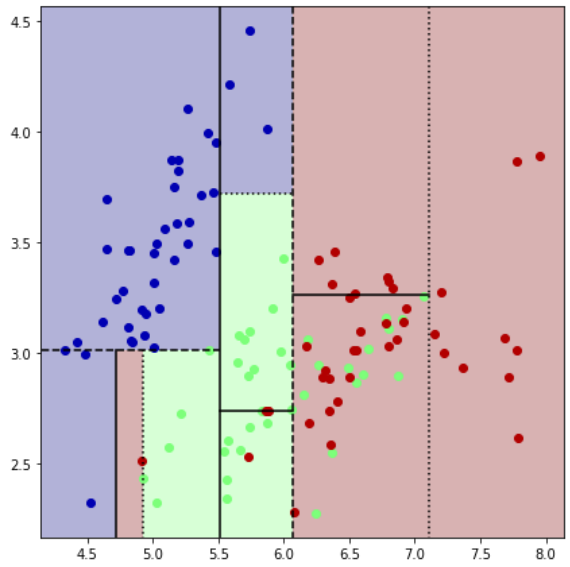
We represented logistic and neural models as functions f( x | $\theta$ )
Can decision trees be represented this way?

Yes, somewhat abstractly:
-> the parameters $\theta$ are

(i) the tree structure (nodes + features)
(ii) threshold values at each internal node
(iii) class probability vectors at leaf nodes

-> the function f( x | $\theta$ ) is represented by the combination of (i), (ii), (iii)

# A Tree Classifier as a Piecewise Constant Function




Example of a Decision Tree Classifier with 2 features, 3 classes

For a classifier, $f_k( x \mid \theta ) = P(y = k \mid x)$

In a tree this is constant at each leaf node (can be different across different leaves)

So, a tree is a **piecewise constant model** of the functions $f_k( x \mid \theta ) = P(\text{class} = k \mid x)$

Questions?

Decision Tree Classifiers

Real-Valued Features

Generalizations

Learning Decision Trees

# How can we Learn Decision Trees?

We have labeled training data $(x_i, y_i)$

Our model is $f(x ; \theta)$ in the form of a tree

Can we use gradient descent?

No: tree structure and thresholds are not differentiable

Instead: use local greedy search to grow the tree

…. and use validation data to select the tree size

# Learning Decision Trees

- Construct (or "grow") trees in a top-down fashion

- Should a node be a leaf node?
  - If so: what should we predict?
  - If not: how should we further split the data?

- Leaf nodes: compute class probabilities, pick majority class

- Non-leaf nodes: pick a feature and a split
  - Greedy: "score" all possible features and splits
  - Score function measures uncertainty about labels after split
    - How much easier is our prediction task after we divide the data?

# Algorithm BuildTree: Greedy training of a decision tree classifier

---

**Input**: Labeled dataset $D = \{ (x_i, y_i) \}$, $i = 1,.. n$
**Output**: A decision tree with parameters $\theta$

---

Compute P = ClassProbabilityVector(D)

**if** LeafCondition(D, P, node) **then**
  node = leaf node  % declare node to be a leaf node

**else**
  $t_j$ = FindBestSplit(D)  % threshold value for best split, for some feature $x_j$
  $D_L = \{ \ (x_i, y_i) \ : \ x_{ij} \leq t \ \}$
  $D_R = \{ \ (x_i, y_i) \ : \ x_{ij} > t \ \}$
  Set left and right children to trees from BuildTree($D_L$) and BuildTree($D_R$)

**end if**

# Algorithm BuildTree: Greedy training of a decision tree classifier

---

**Input**: Labeled dataset $D = \{ (x_i, y_i) \}$, $i = 1,.. n$
**Output**: A decision tree with parameters $\theta$

---

Compute P = ClassProbabilityVector(D)

**if** LeafCondition(D, P, node) **then**
    node = leaf node

**else**
    $t_j$ = FindBestSplit(D)
    $D_L = \{ \ (x_i, y_i) \ : \ x_{ij} \leq t \ \}$
    $D_R = \{ \ (x_i, y_i) \ : \ x_{ij} > t \ \}$
    Set left and right children to trees from BuildTree($D_L$) and BuildTree($D_R$)

**end if**

# ClassProbabilityVector?

The class probability vector at a node
= relative frequencies of the class labels at that node

**Example with 4 classes**:

12 datapoints at a node, with:

Num(class 1) = 6 => P(class 1 | node) = 6/12 = 0.5

Num(class 2) = 4 => P(class 2 | node) = 4/12 = 0.333

Num(class 3) = 2 => P(class 3 | node) = 2/12 = 0.167

Num(class 4) = 0 => P(class 4 | node) = 0/12 = 0

# LeafCondition?

Typically declare a node to be a leaf if any of the following are true

1. All labels at the node belong to the same class
   (no point in splitting any further)

2. The node is at the maxDepth for the tree
   (controls complexity, can prevent the tree from getting too large)

3. The number of examples at a leaf node < $n_{min}$
   (if we split any further we may be fitting noise and overfit)

Hyperparameters: maxDepth, $n_{min}$

# FindBestSplit?

Tree search is a local greedy algorithm

For any node, want a split to best predict the classes in the subtree below…
….. or equivalently, reduces uncertainty (or variance) for the class labels

Several possible approaches for measuring the quality of a split

1. Gini index: average variance (uncertainty) in left/right child nodes

2. Information gain: similar to Gini, reduces class label entropy

3. Classification accuracy? Turns out not to work as well as the other two

# Gini Index

The Gini index is a measure of the variance of a set of class labels

Definition:
Given a set of probabilities of class labels $p_1$, $p_2$, …. $p_K$

Gini index = $\sum_k (p_k)\ (1 - p_k)$

$= \sum_k (p_k) - \sum_k (p_k\ p_k)$

$= 1 - \sum_k (p_k)^2$

Intuition: if one $p_k$=1 and all others = 0, then Gini index = 1 – 1 = 0,
i.e., we have zero uncertainty about the class labels

# Examples of the Gini Index

Say K = 2, two class labels

Example 1: P(class 1) = $p_1 = 0.9$, P(class 2) = $p_2 = 0.1$

Gini index $= 1 - \sum_k (p_k)^2 = 1 - (0.9^2 + 0.1^2) = 1 - (0.81 + 0.01) = 0.18$

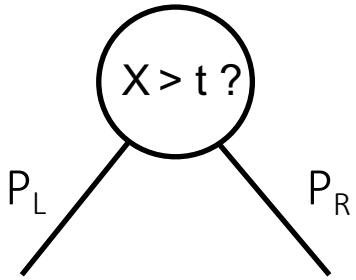
=> Low value of Gini index => low uncertainty

Example 2: P(class 1) = $p_1 = 0.6$, P(class 2) = $p_2 = 0.4$

Gini index $= 1 - \sum_k (p_k)^2 = 1 - (0.36 + 0.16) = 1 - (0.52) = 0.48$

=> Higher value of Gini index => higher uncertainty

# Using the Gini Index at Nodes




Fraction of datapoints that go left and right

$D_L$ = left subset of data

$D_R$ = right subset of data

Class probabilities in $D_L$

Class probabilities in $D_R$

$G_L$ = Gini index of $D_L$

$G_R$ = Gini index of $D_R$

Gini(t) = weighted average = $P_L\, G_L + P_R\, G_R$

We want the feature x and split t that minimizes Gini(t)

# Questions?

# Wrapup

- Decision trees: a new model
  - Implement Boolean functions
  - Internal nodes represent a “splitting condition” on a particular feature
  - Leaf nodes represent predicted class probabilities from following tree

- Learning decision trees
  - Built in a top-down fashion
  - Recursively select best feature to split on until we decide to stop
  - More details & examples next lecture