Introduction to Machine Learning

CS307 --- Fall 2022

Decision Tree Learning

Reading:

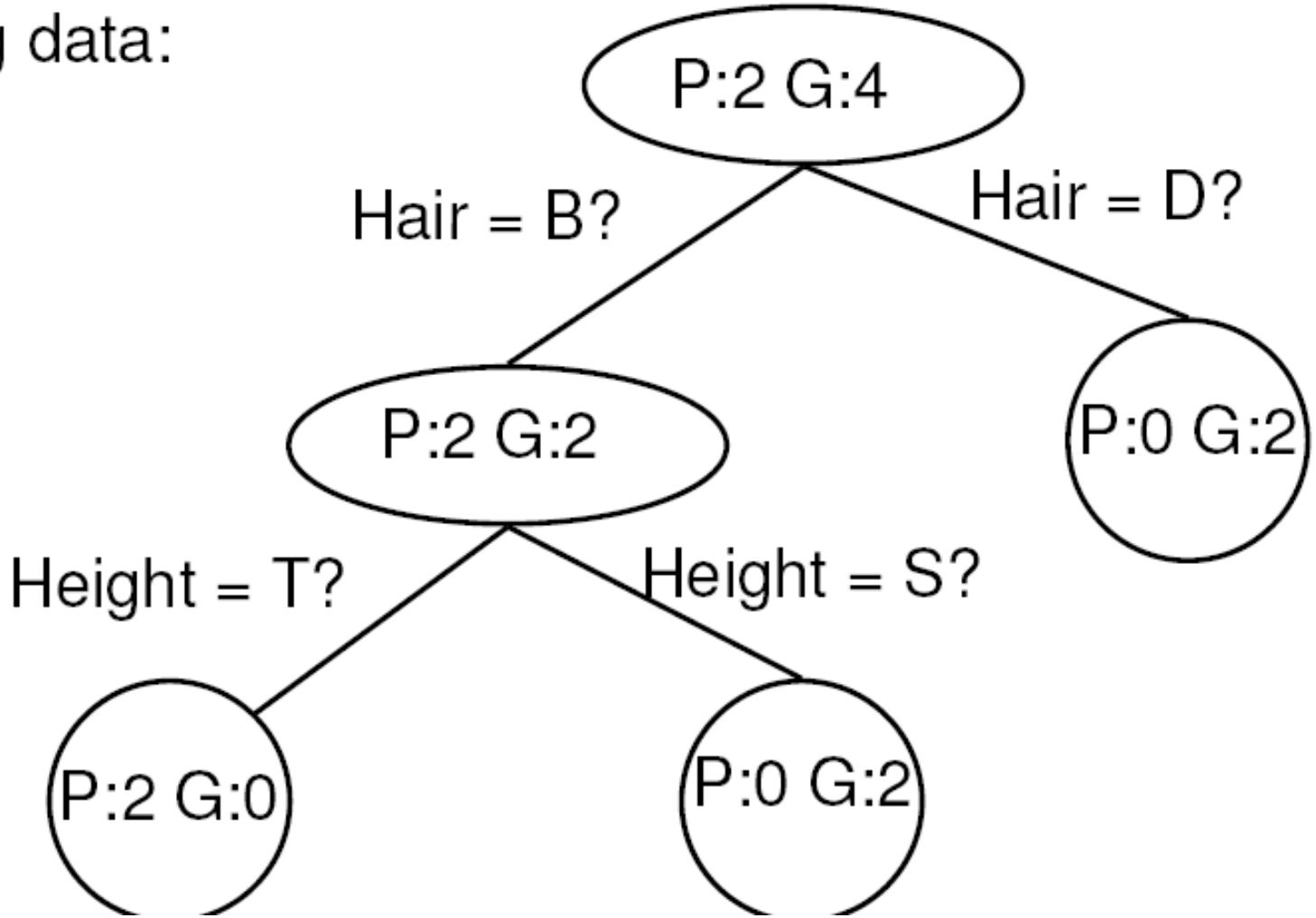
Sections 18.2-18.3, R&N

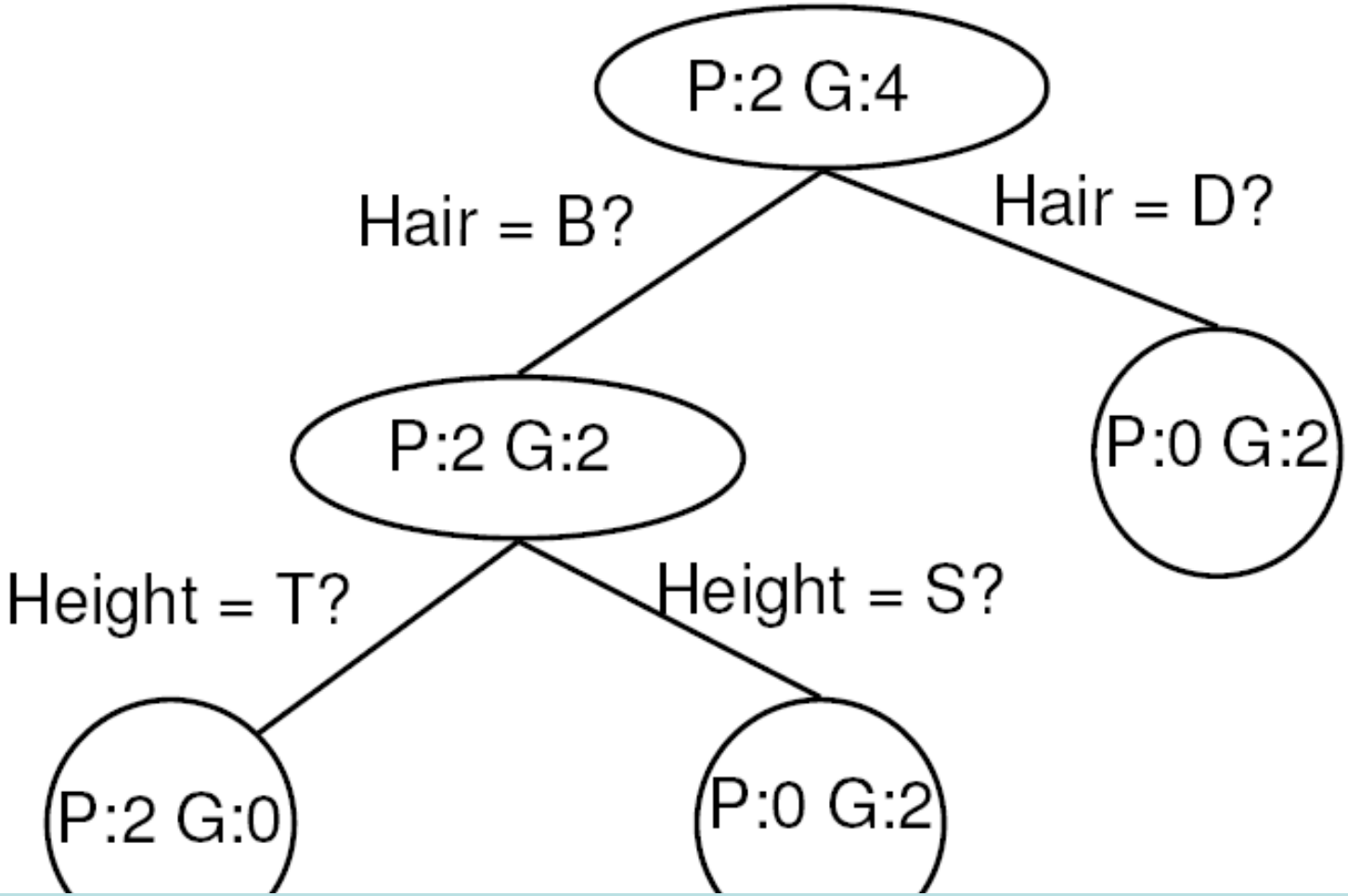
Sections 3.1-3.4, Mitchell

Decision Tree Example

- Three variables:
  - Attribute 1: Hair = {blond, dark}
  - Attribute 2: Height = {tall, short}
  - Class: Country = {Gromland, Polvia}

Training data:
(B,T,P)
(B,T,P)
(B,S,G)
(D,S,G)
(D,T,G)
(B,S,G)





The class of a new input can be classified by following the tree all the way down to a leaf and by reporting the output of the leaf. For example:
(B,T) is classified as
(D,S) is classified as

Decision Trees

Decision Trees are classifiers for instances represented as feature vectors.

Nodes are (equality and inequality) tests for feature values

There is one branch for each value of the feature

Leaves specify the categories (labels)

Can categorize instances into multiple disjoint categories

General Case (Discrete Attributes)

- We have R observations from training data
  - Each observation has M attributes $X_1,..,X_M$
  - Each X_i can take N distinct discrete values
  - Each observation has a class attribute Y with C distinct (discrete) values
  - Problem: Construct a sequence of tests on the attributes such that, given a new input $(x'_1,..,x'_M)$, the class attribute y is correctly predicted

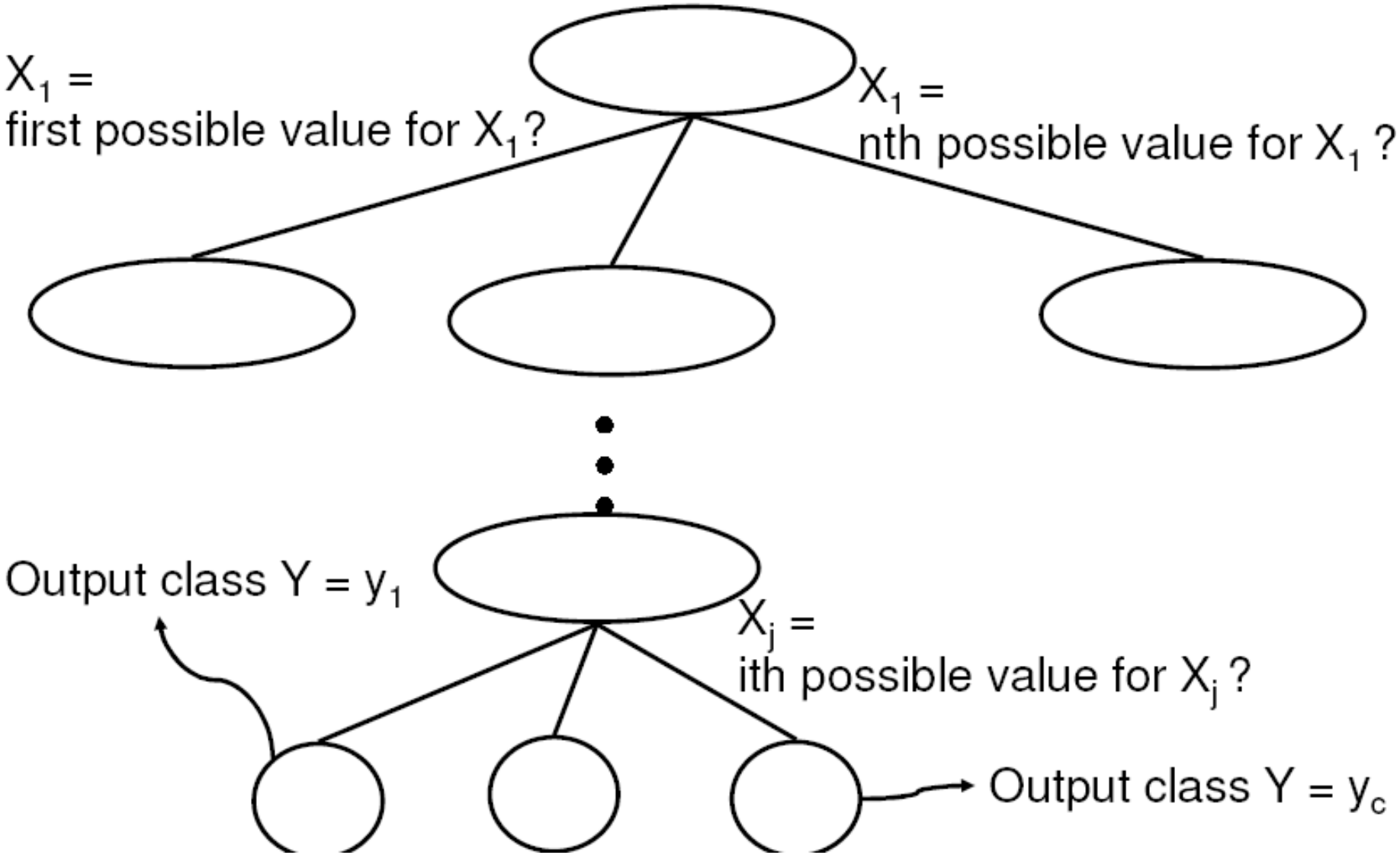
	X_1	.....	X_M	Y
Input data	x'_1	.....	x'_M	???

	X_1	.....	X_M	Y
Data 1	x_1	.....	x_M	y
Data 2				
.....				
Data R				

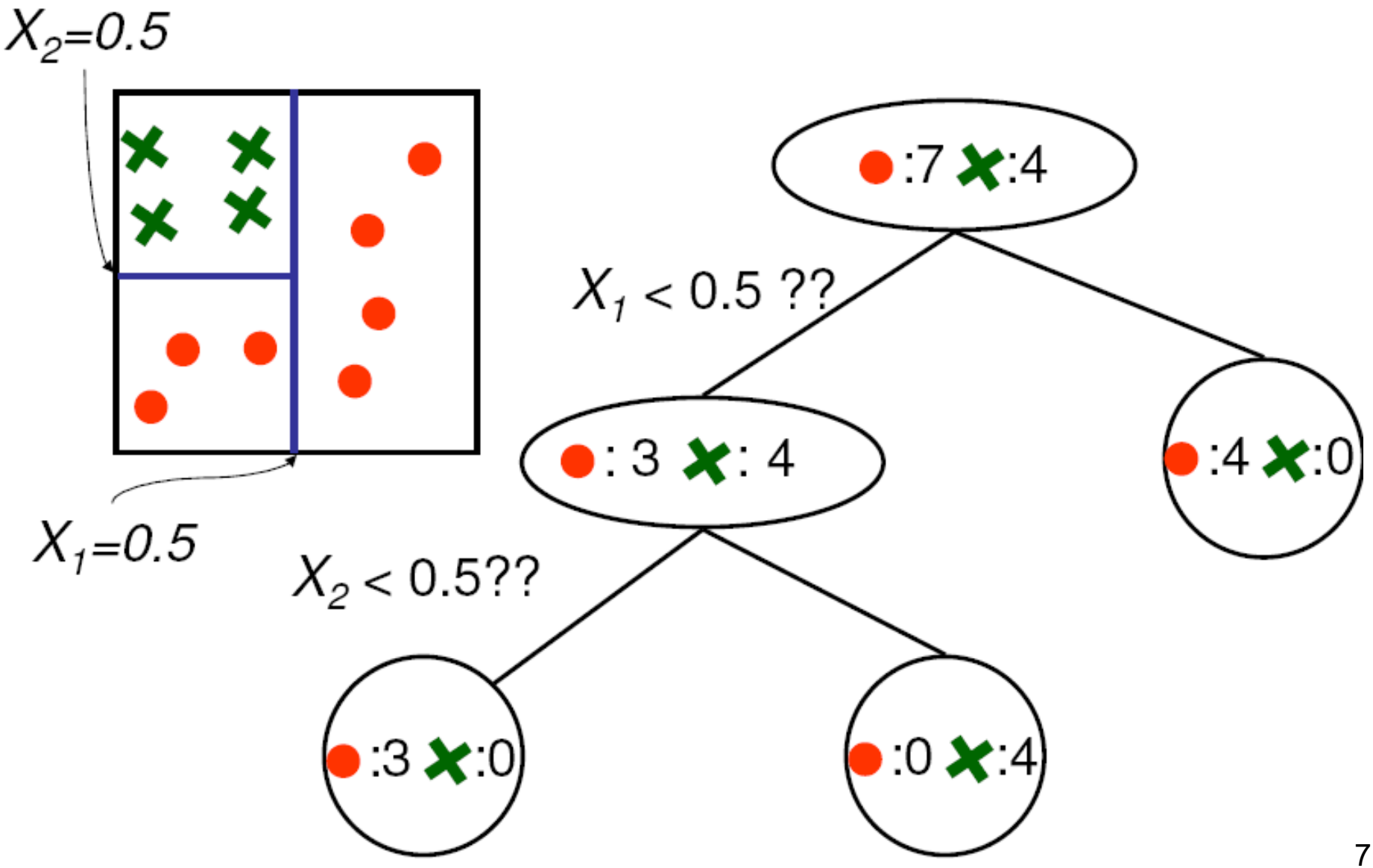
Training Data

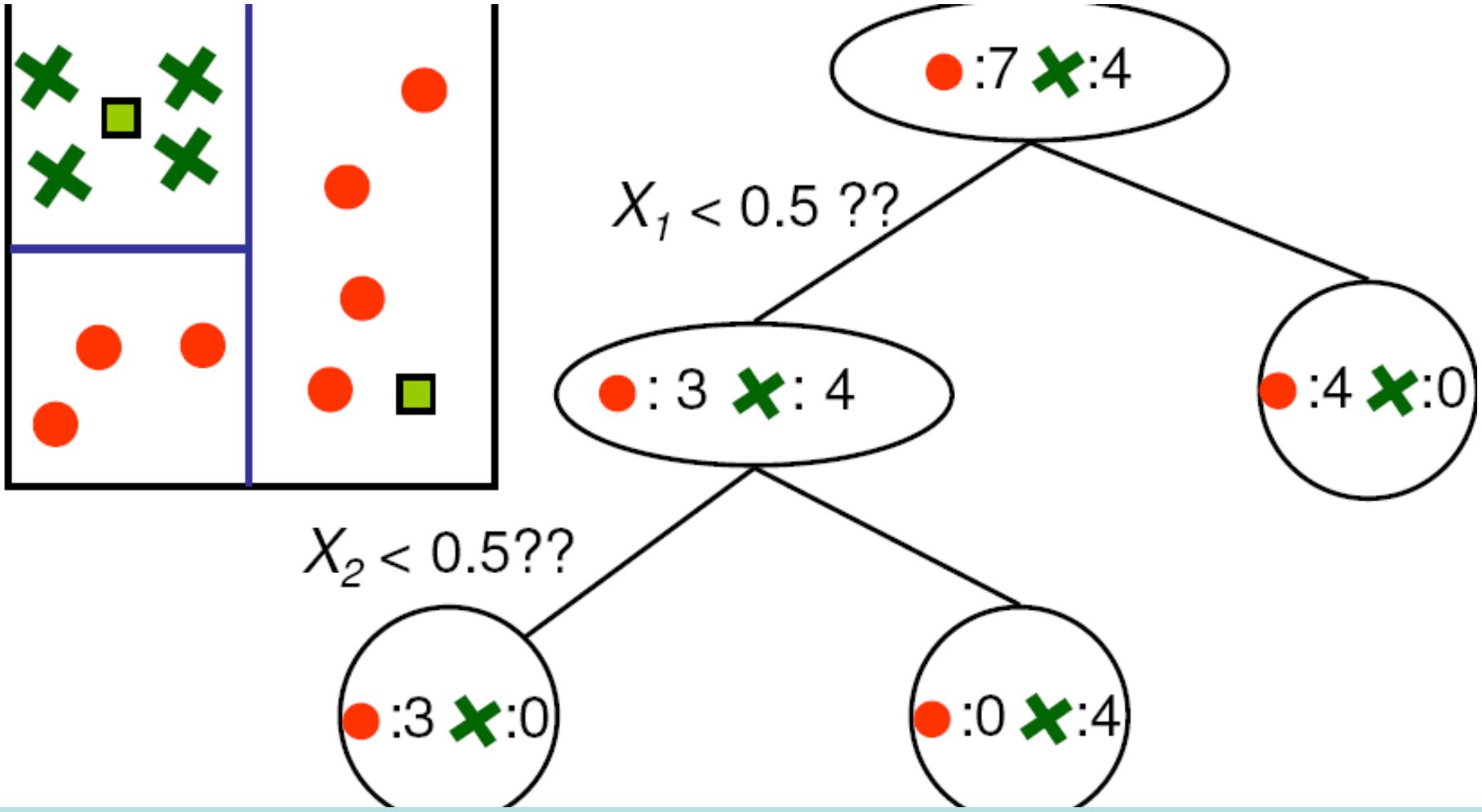
X = attributes of training data (RxM) Y = Class of training data (R)

General Decision Tree (Discrete Attributes)



Decision Tree Example





The class of a new input can be classified by following the tree all the way down to a leaf and by reporting the output of the leaf. For example:
(0.2,0.8) is classified as
(0.8,0.2) is classified as

General Case (Continuous Attributes)

- We have R observations from training data
  - Each observation has M attributes $X_1,..,X_M$
  - Each X_i can take N continuous values
  - Each observation has a class attribute Y with C distinct (discrete) values
  - Problem: Construct a sequence of tests of the form $X_i < t_i$? on the attributes such that, given a new input $(x'_1,..,x'_M)$, the class attribute y is correctly predicted

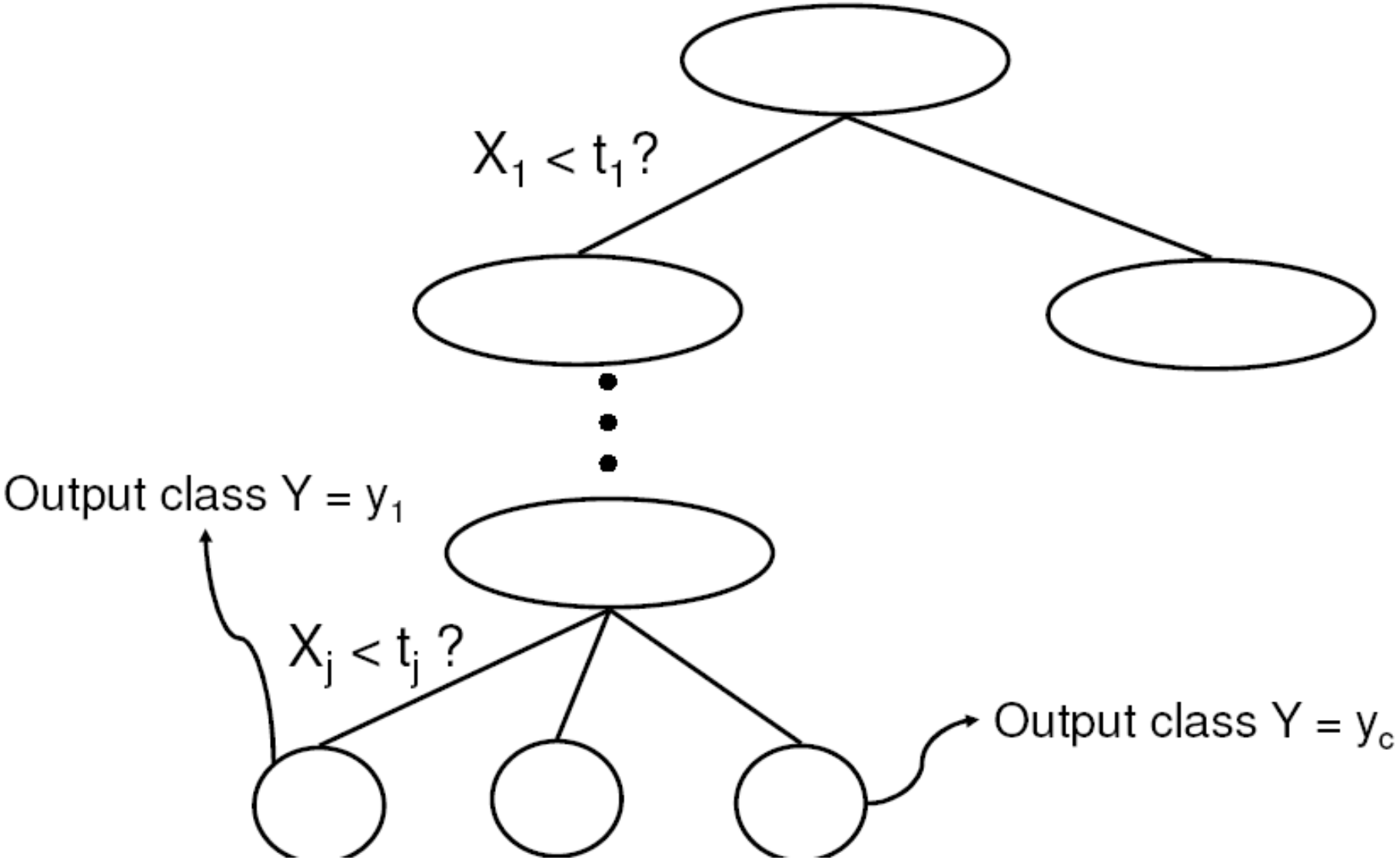
	X_1	.....	X_M	Y
Input data	x'_1	.....	x'_M	???

	X_1	.....	X_M	Y
Data 1	x_1	.....	x_M	y
Data 2				
.....				
Data R				

Training Data

X = attributes of training data (RxM) Y = Class of training data (R)

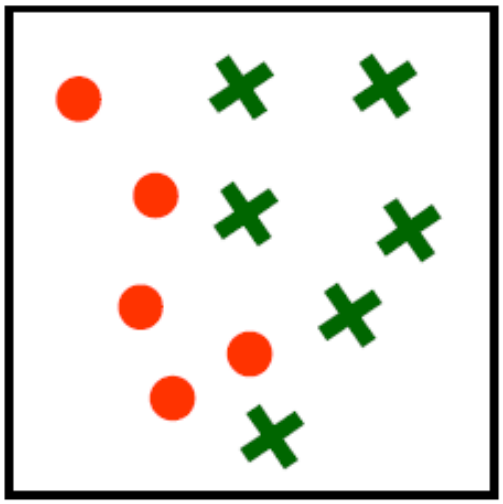
General Decision Tree (Continuous Attributes)



Basic Questions

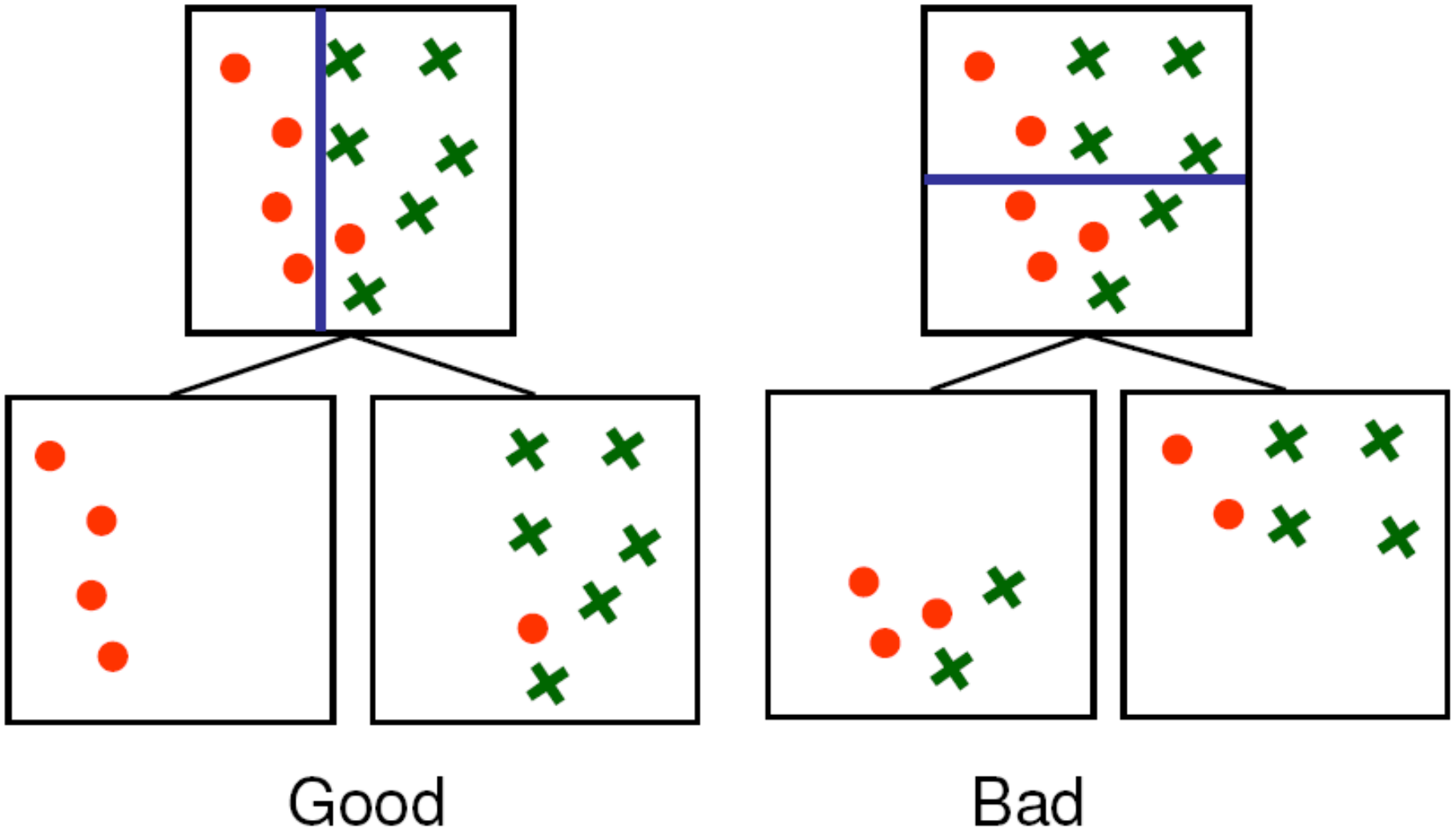
- How to choose the attribute/value to split on at each level of the tree?
- When to stop splitting? When should a node be declared a leaf?
- If a leaf node is impure, how should the class label be assigned?

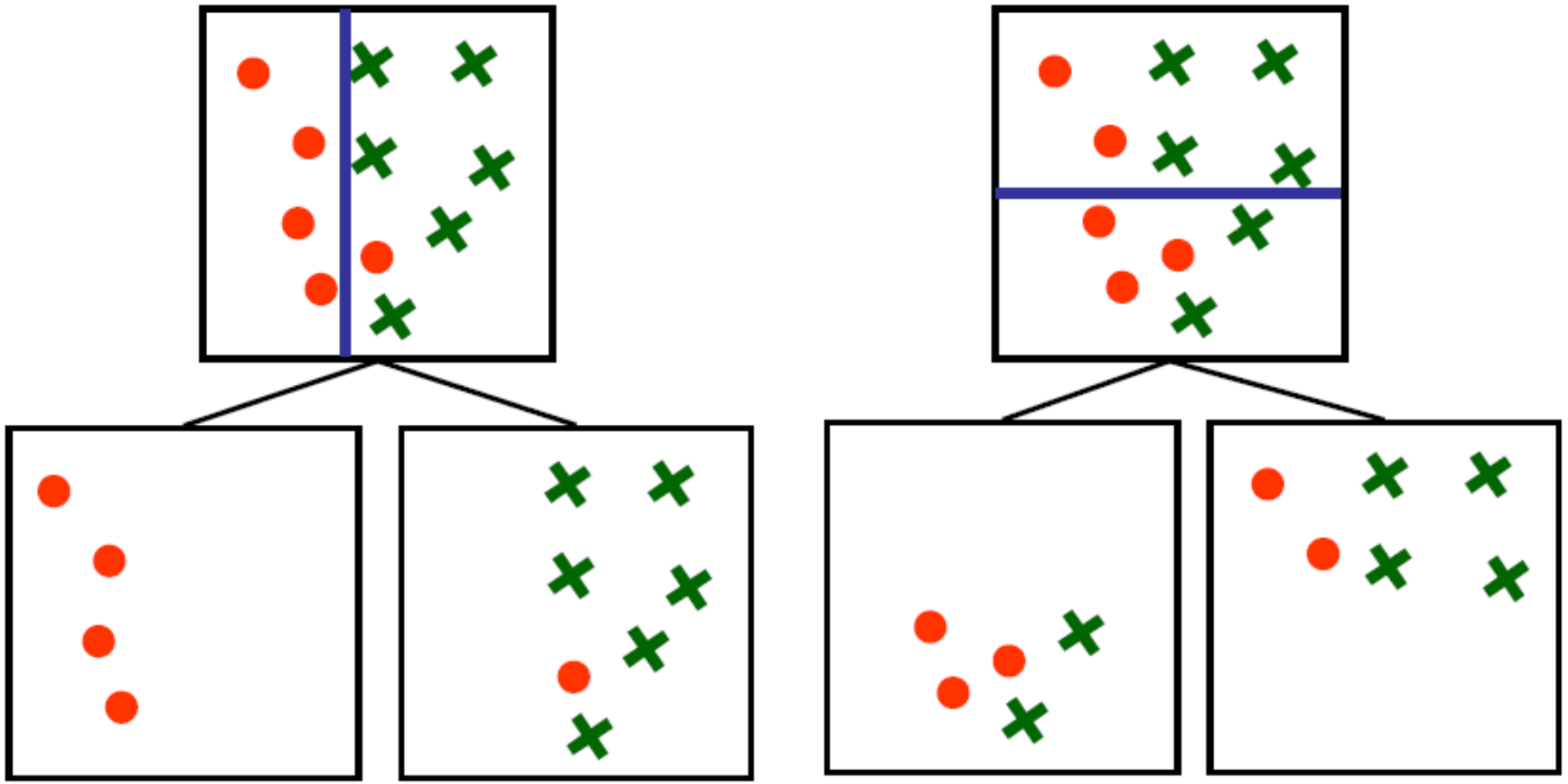
How to choose the attribute/value to split on at each level of the tree?



- Two classes (red circles/green crosses)
- Two attributes: X_1 and X_2
- 11 points in training data
- Goal: Construct a decision tree such that the leaf nodes predict correctly the class for all the training examples

How to choose the attribute/value to split on at each level of the tree?

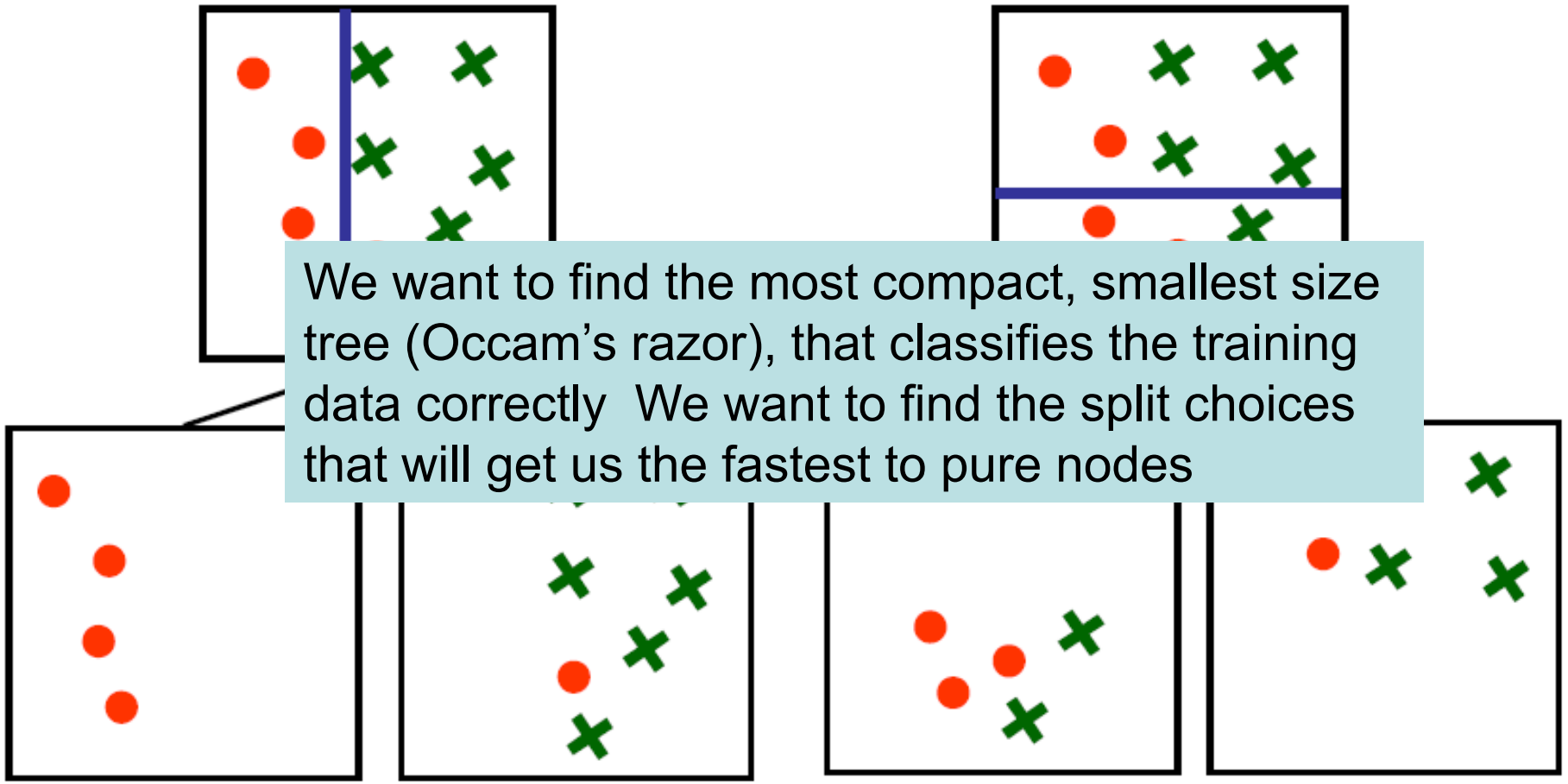




This node is "pure" because there is only one class left → No ambiguity in the class label

This node is almost "pure" → Little ambiguity in the class label

These nodes contain a mixture of classes
Do not disambiguate between the classes



We want to find the most compact, smallest size tree (Occam's razor), that classifies the training data correctly We want to find the split choices that will get us the fastest to pure nodes

This node is "pure" because there is only one class left → No ambiguity in the class label

This node is almost "pure" → Little ambiguity in the class label

These nodes contain a mixture of classes → Do not disambiguate between the classes

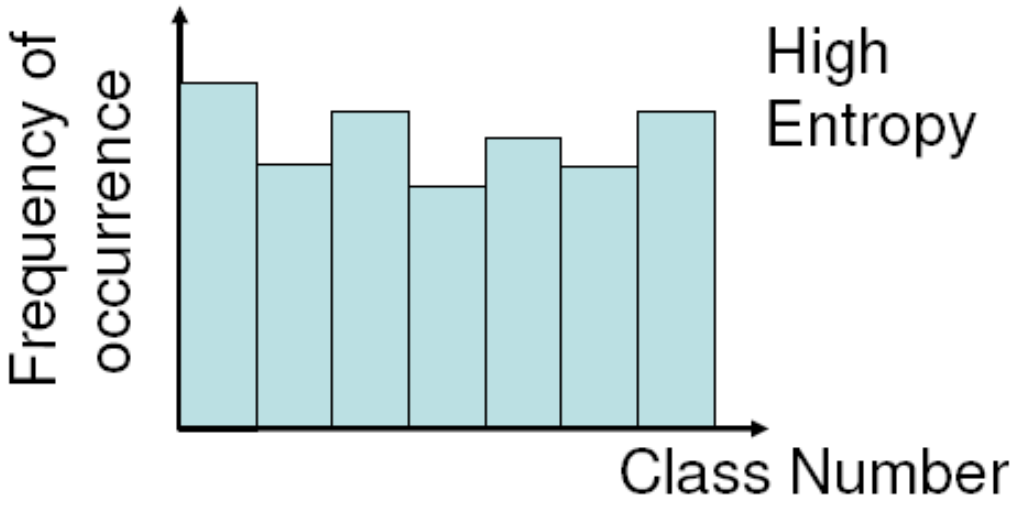
Entropy

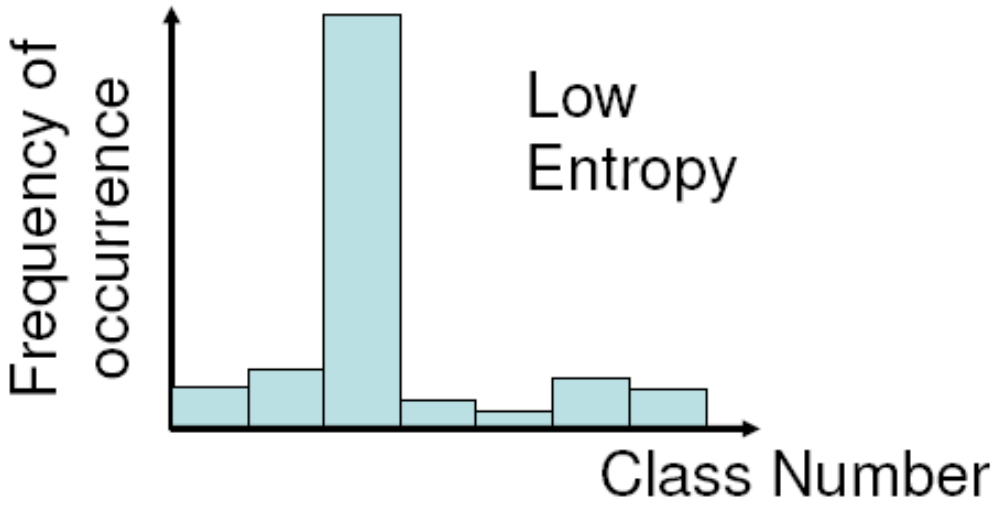
- Entropy is a measure of the impurity of a distribution, defined as:

$$H = \sum_{i=1}^{n} -P_i \log_2 P_i$$

- P_i = probability of occurrence of value i
  - High entropy → All the classes are (nearly) equally likely
  - Low entropy → A few classes are likely; most of the classes are rarely observed
  - assume $0 \log_2 0 = 0$

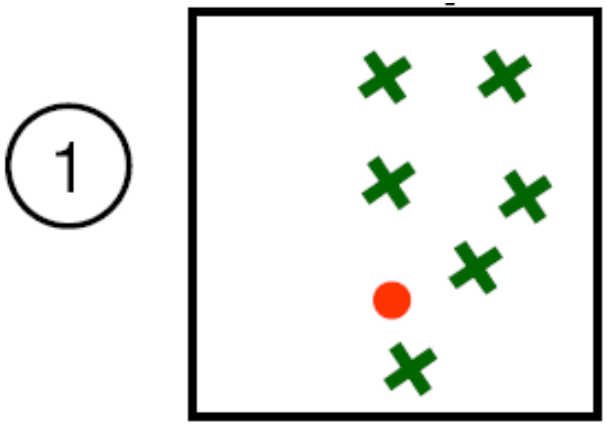
Entropy





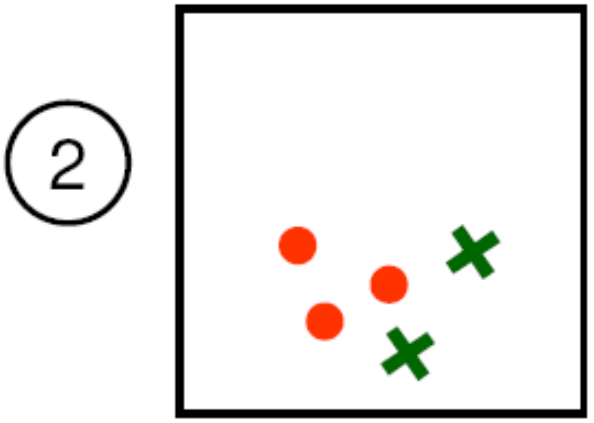
The entropy captures the degree of “purity” of the distribution

Example Entropy Calculation



$N_A = 1$
$N_B = 6$
$p_A = N_A/(N_A+N_B) = 1/7$
$p_B = N_B/(N_A+N_B) = 6/7$

$H_1 = -p_A \log_2 p_A - p_B \log_2 p_B$
$= 0.59$



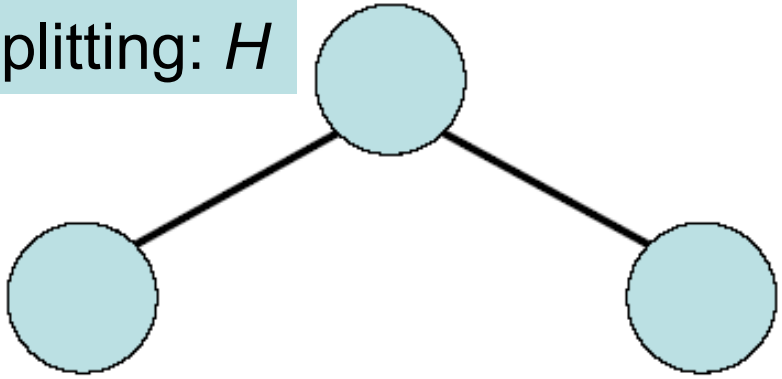
$N_A = 3$
$N_B = 2$
$p_A = N_A/(N_A+N_B) = 3/5$
$p_B = N_B/(N_A+N_B) = 2/5$

$H_2 = -p_A \log_2 p_A - p_B \log_2 p_B$
$= 0.97$

$H_1 < H_2 \Rightarrow$ (2) less pure than (1)

Conditional Entropy

Entropy before splitting: H



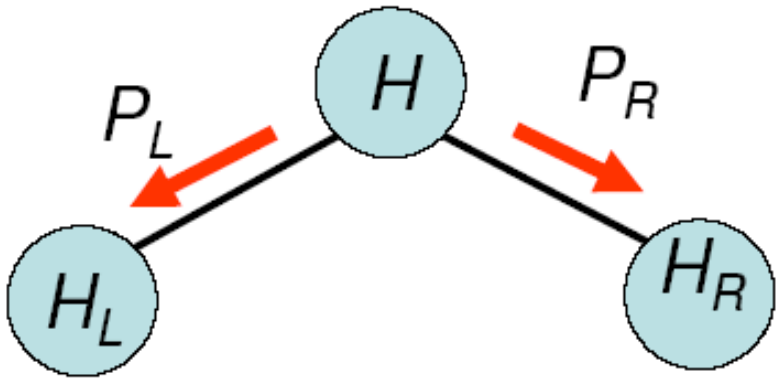
After splitting, a fraction P_L of the data goes to the left node, which has entropy H_L

After splitting, a fraction P_R of the data goes to the right node, which has entropy H_R

The average entropy (or "conditional entropy") after splitting

$$H_L \times P_L + H_R \times P_R$$

Information Gain



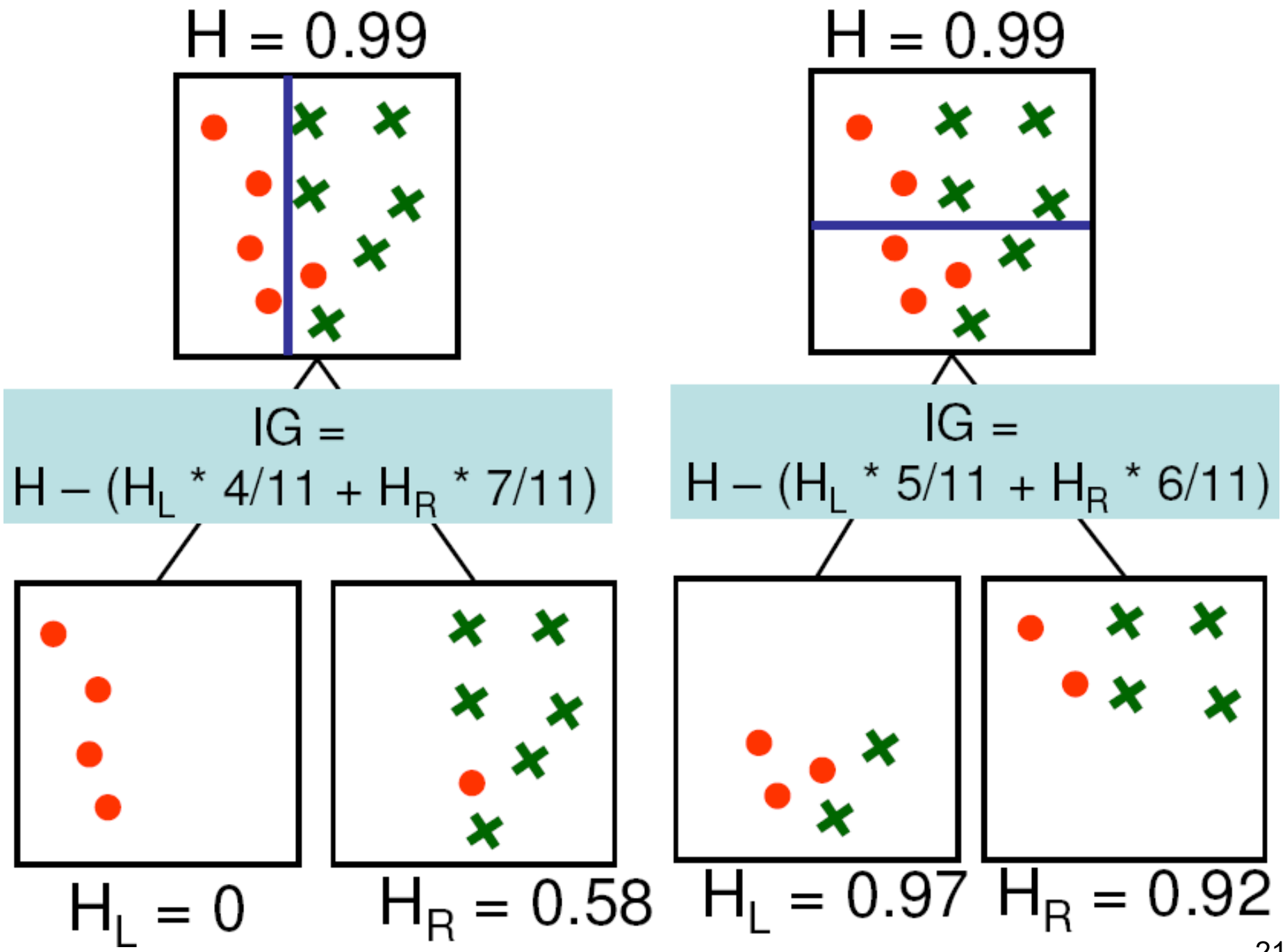
We want nodes as pure as possible

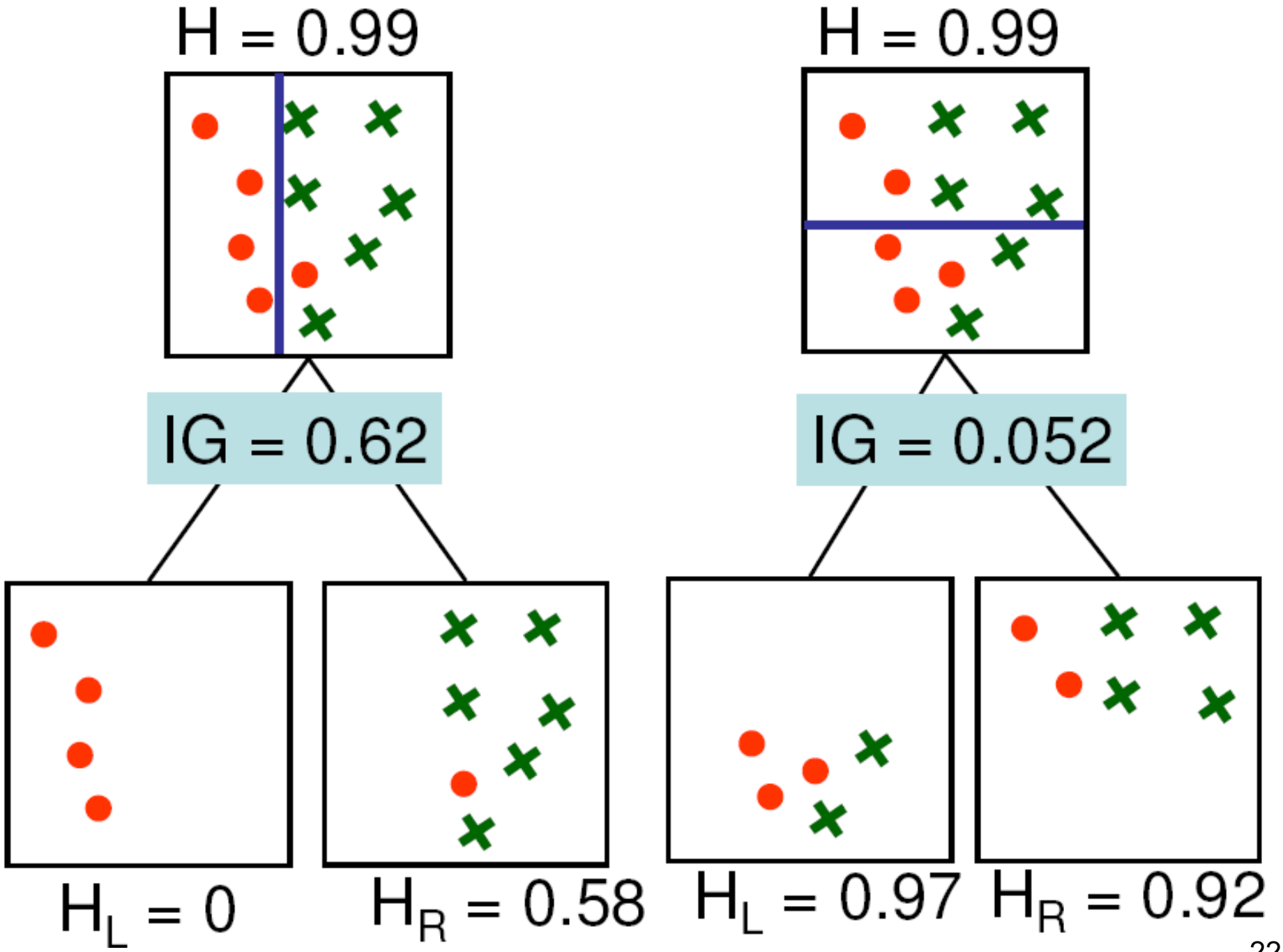
→ We want to reduce the entropy as much as possible

→ We want to maximize the difference between the entropy of the parent node and the expected entropy of the children

Maximize:

$$IG = H - (H_L \times P_L + H_R \times P_R)$$


H = 0.99
IG =
H – (H_L * 4/11 + H_R * 7/11)
H_L = 0
H_R = 0.58
H = 0.99
IG =
H – (H_L * 5/11 + H_R * 6/11)
H_L = 0.97
H_R = 0.92


H = 0.99
IG = 0.62
H = 0.99
IG = 0.052
$H_L = 0$
$H_R = 0.58$
$H_L = 0.97$
$H_R = 0.92$

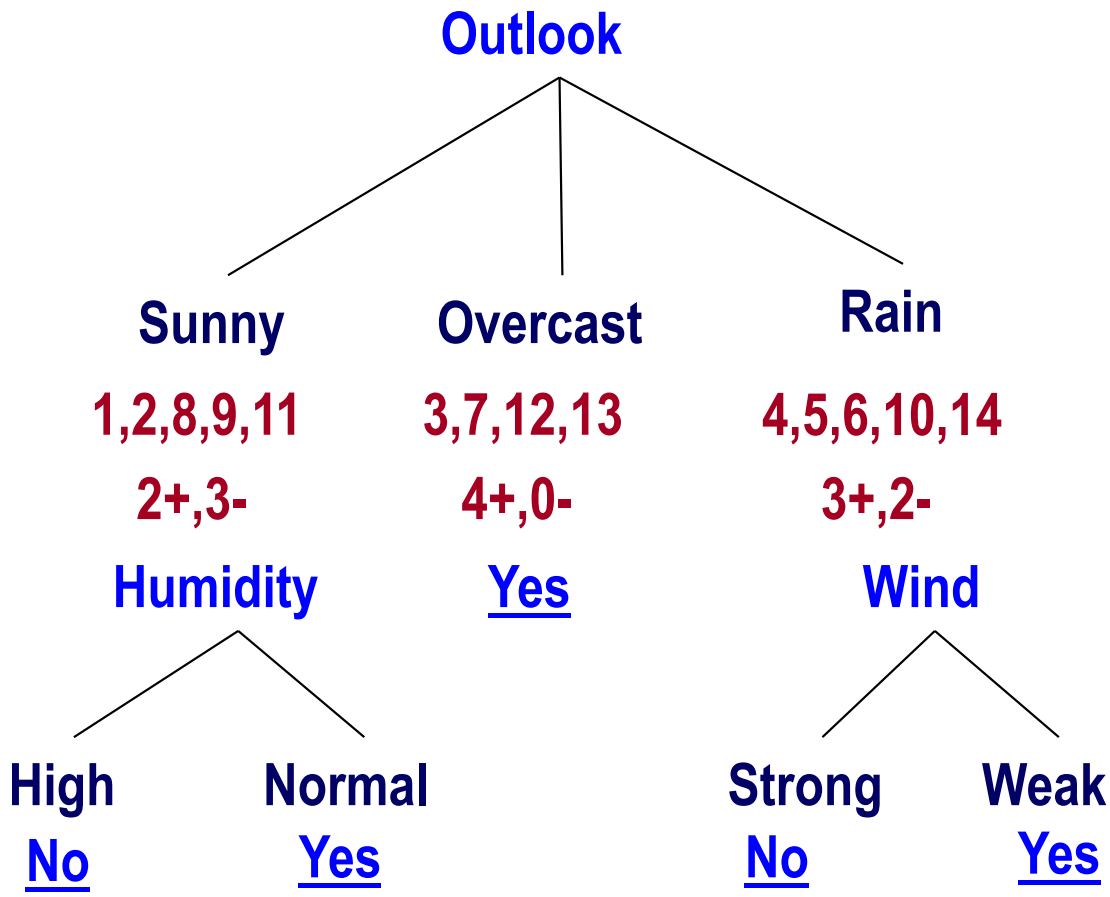
Notations

- Entropy: $H(Y)$ = Entropy of the distribution of classes at a node
- Conditional Entropy:
  - *Discrete*: $H(Y|X_j)$ = Entropy after splitting with respect to variable j
  - *Continuous*: $H(Y|X_j,t)$ = Entropy after splitting with respect to variable j with threshold t
- Information gain:
  - *Discrete*: $IG(Y|X_j) = H(Y) - H(Y|X_j)$
  - *Continuous*: $IG(Y|X_j,t) = H(Y) - H(Y|X_j,t)$

Another Illustrative Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

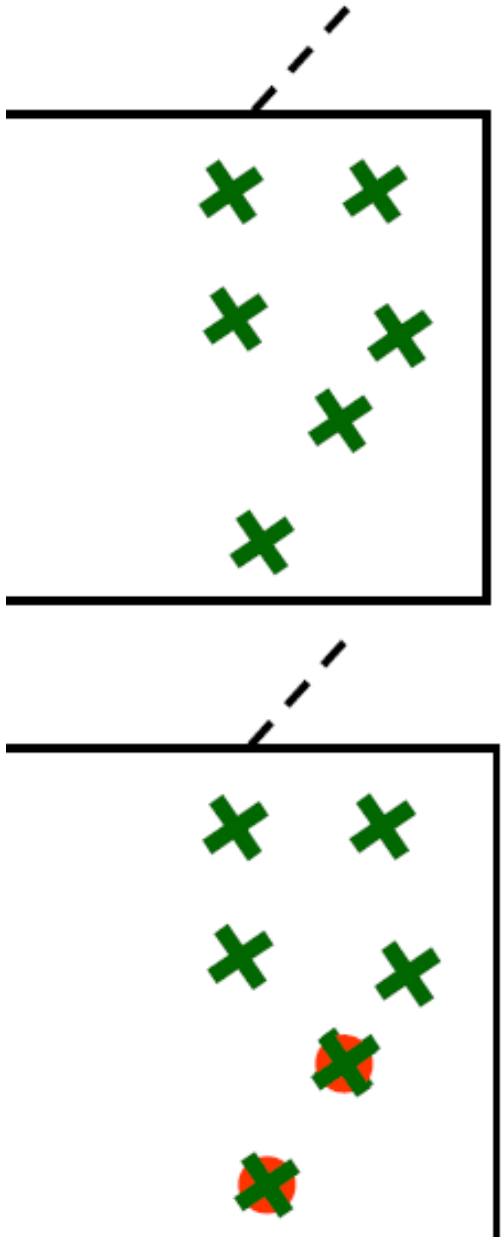
Another Illustrative Example



Basic Questions

- How to choose the attribute/value to split on at each level of the tree?
- When to stop splitting? When should a node be declared a leaf?
- If a leaf node is impure, how should the class label be assigned?
- If the tree is too large, how can it be pruned?

Pure and Impure Leaves and When to Stop Splitting



All the data in the node comes from a single class. We declare the node to be a leaf and stop splitting. This leaf will output the class of the data it contains

Several data points have exactly the same attributes even though they are not from the same class. We cannot split any further. We still declare the node to be a leaf, but it will output the class that is the majority of the classes in the node (in this example, 'B')

Decision Tree Algorithm (Discrete Attributes)

- LearnTree(X, Y)
  - Input:
    - Set X of R training vectors, each containing the values ($x_1,..,x_M$) *of* M attributes ($X_1,..,X_M$)
    - A vector Y of R elements, where y_j = class of the jth datapoint
  - If all the datapoints in X have the same class value y
    - Return a leaf node that predicts y as output
  - If all the datapoints in X have the same attribute value ($x_1,..,x_M$)
    - Return a leaf node that predicts the majority of the class values in Y as output
  - Try all the possible attributes X_j and choose the one, j^*, for which IG($Y|X_j$) is maximum
    - For every possible value v of X_{j^*}:
      - X_v, Y_v= set of datapoints for which $x_{j^*} = v$ and corresponding classes
      - Child$_v$ ← LearnTree(X_v, Y_v)

Decision Tree Algorithm (Continuous Attributes)

- LearnTree(X, Y)
  - Input:
    - Set X of R training vectors, each containing the values ($x_1,..,x_M$) *of* M attributes ($X_1,..,X_M$)
    - A vector Y of R elements, where y_j = class of the j^{th} datapoint
  - If all the datapoints in X have the same class value y
    - Return a leaf node that predicts y as output
  - If all the datapoints in X have the same attribute value ($x_1,..,x_M$)
    - Return a leaf node that predicts the majority of the class values in Y as output
  - Try all the possible attributes X_j and threshold t and choose the one, j^*, for which IG($Y|X_j,t$) is maximum
  - X_L, Y_L = set of datapoints for which $x_j^* < t$ and corresponding classes
  - X_H, Y_H = set of datapoints for which $x_j^* >= t$ and corresponding classes
  - Left Child ← LearnTree(X_L, Y_L)
  - Right Child ← LearnTree(X_H, Y_H)

Expressiveness of Decision Trees

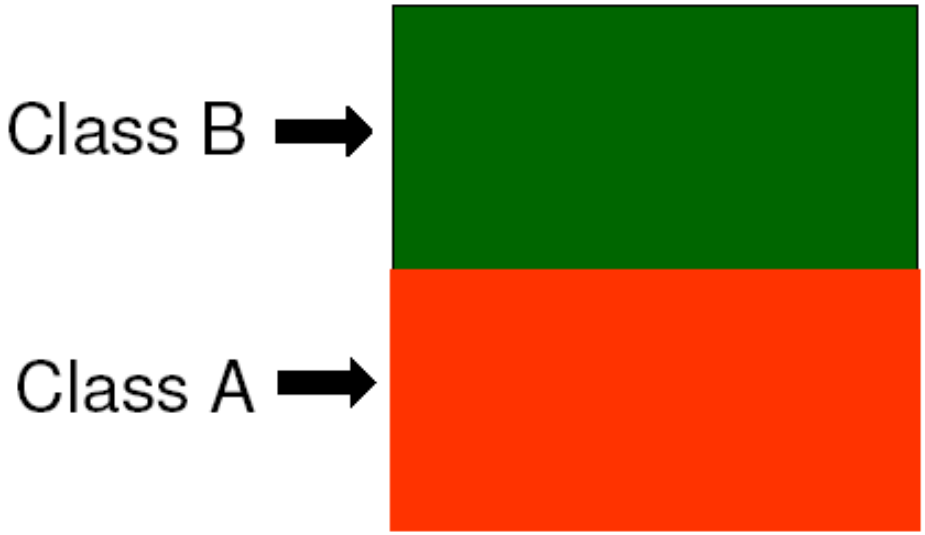
Can represent any Boolean function.

Can be rewritten as rules in Disjunctive Normal Form (DNF)

Decision Trees So Far

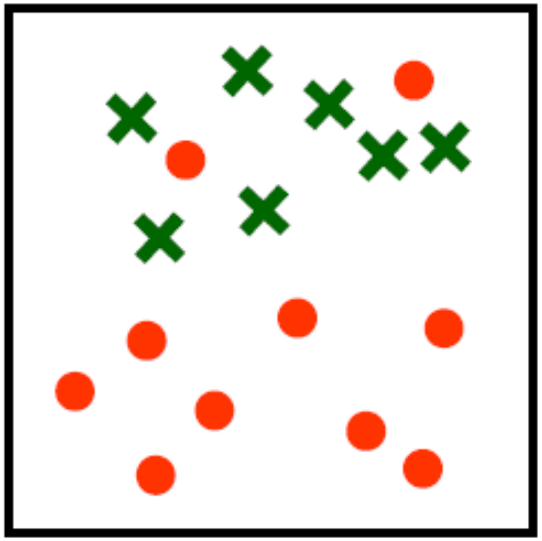
- Given *R* observations from training data, each with *M* attributes *X* and a class attribute *Y*, construct a sequence of tests (decision tree) to predict the class attribute *Y* from the attributes *X*
- Basic strategy for defining the tests ("when to split") → maximize the information gain on the training data set at each node of the tree
- Problem (next):
  – Evaluating the tree on training data is dangerous → *overfitting*

The Overfitting Problem (Example)



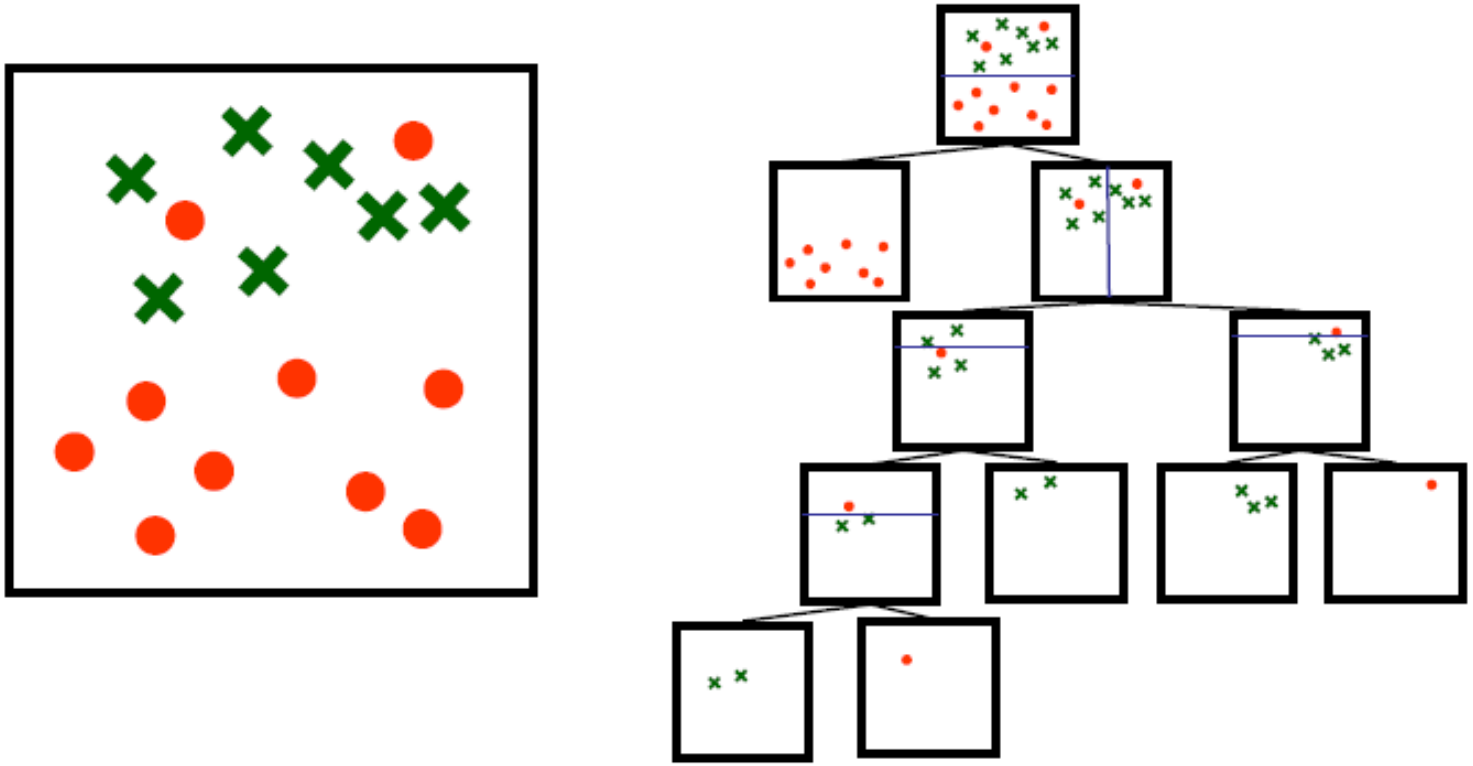
- Suppose that, in an ideal world, class B is everything such that X_2 >= 0.5 and class A is everything with X_2 < 0.5
- Note that attribute X_1 is irrelevant
- Seems like generating a decision tree would be trivial

The Overfitting Problem (Example)



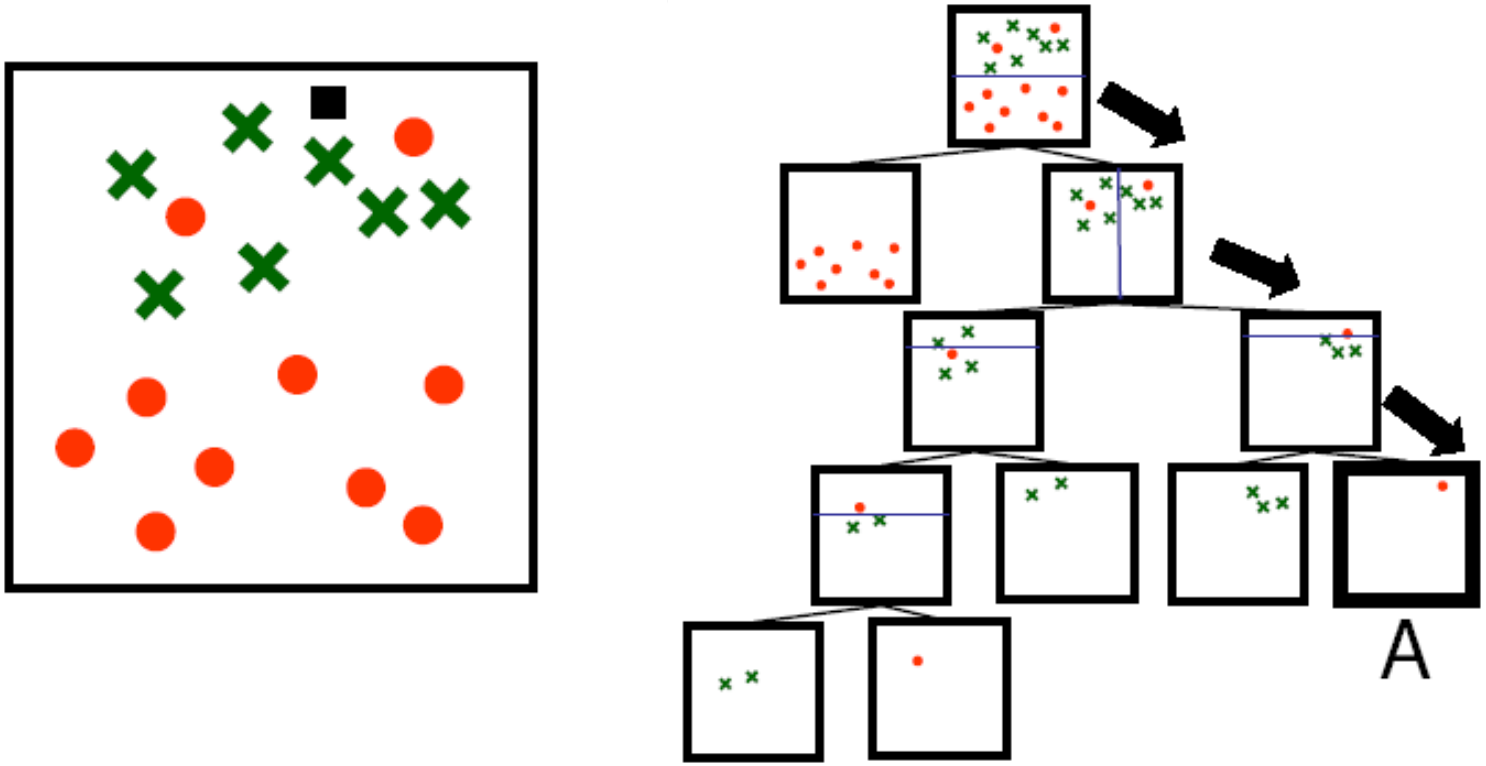
- However, we collect training examples from the perfect world through some imperfect observation device
- As a result, the training data is corrupted by *noise*.

The Overfitting Problem: Example



- Because of the noise, the resulting decision tree is far more complicated than it should be
- This is because the learning algorithm tries to classify *all of the training set perfectly* This is a fundamental problem in learning: *overfitting*

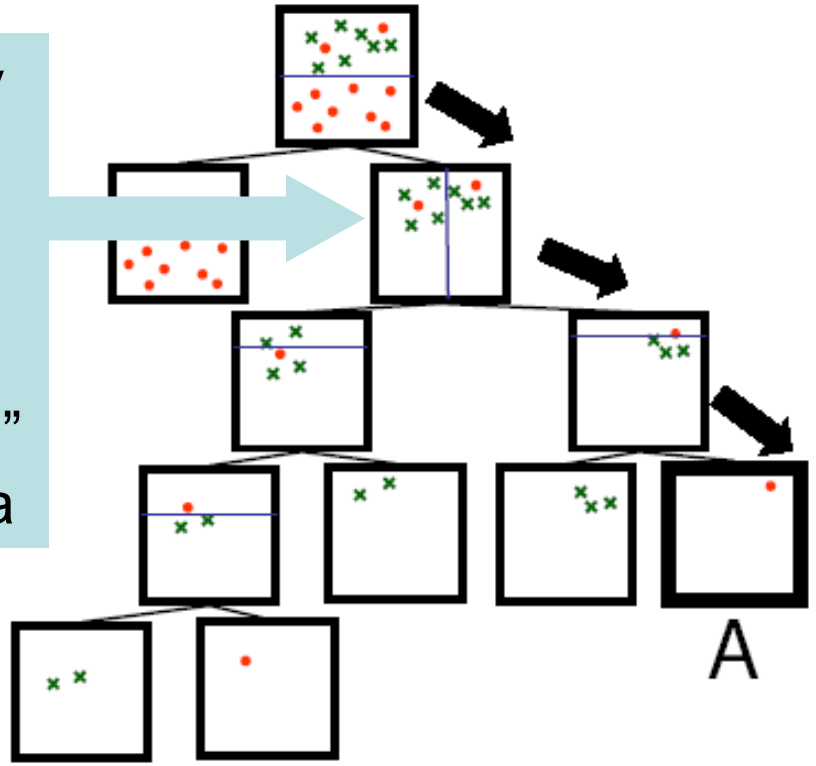
The Overfitting Problem: Example



- The effect of overfitting is that the tree is guaranteed to classify the training data perfectly, but it may do a terrible job at classifying new test data.
- Example: (0.6,0.9) is classified as ‘A’

The Overfitting Problem: Example

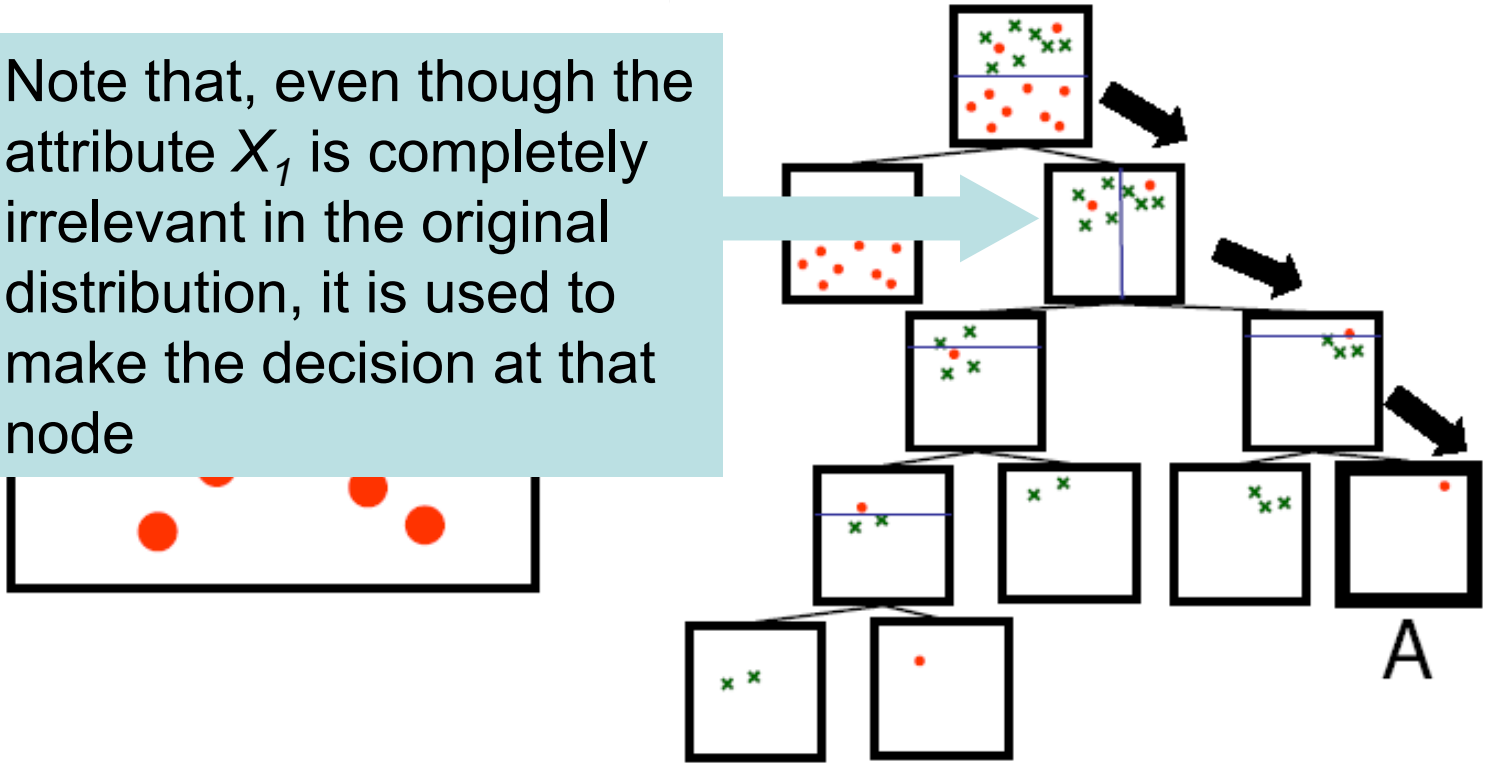
It would be nice to identify automatically that splitting this node is stupid.
Possible criterion: figure out that splitting this node will lead to a “complicated” tree suggesting noisy data



- The effect of overfitting is that the tree is guaranteed to classify the training data perfectly, but it may do a terrible job at classifying new test data.
- Example: (0.6,0.9) is classified as ‘A’

The Overfitting Problem: Example

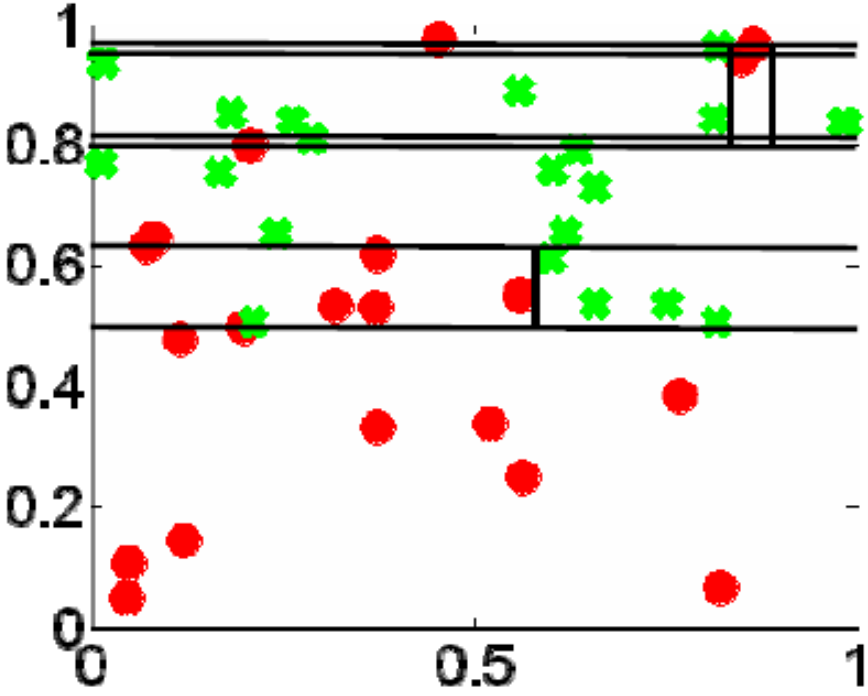
Note that, even though the attribute X_1 is completely irrelevant in the original distribution, it is used to make the decision at that node



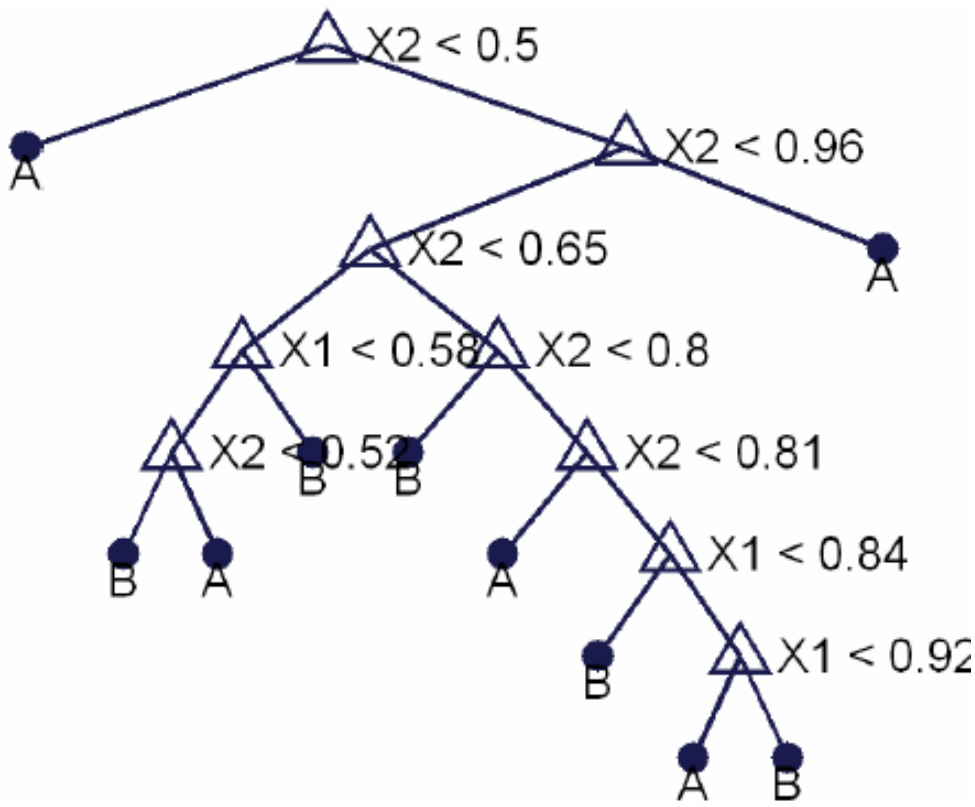
- The effect of overfitting is that the tree is guaranteed to classify the training data perfectly, but it may do a terrible job at classifying new test data.
- Example: (0.6,0.9) is classified as 'A'

Possible Overfitting Solutions

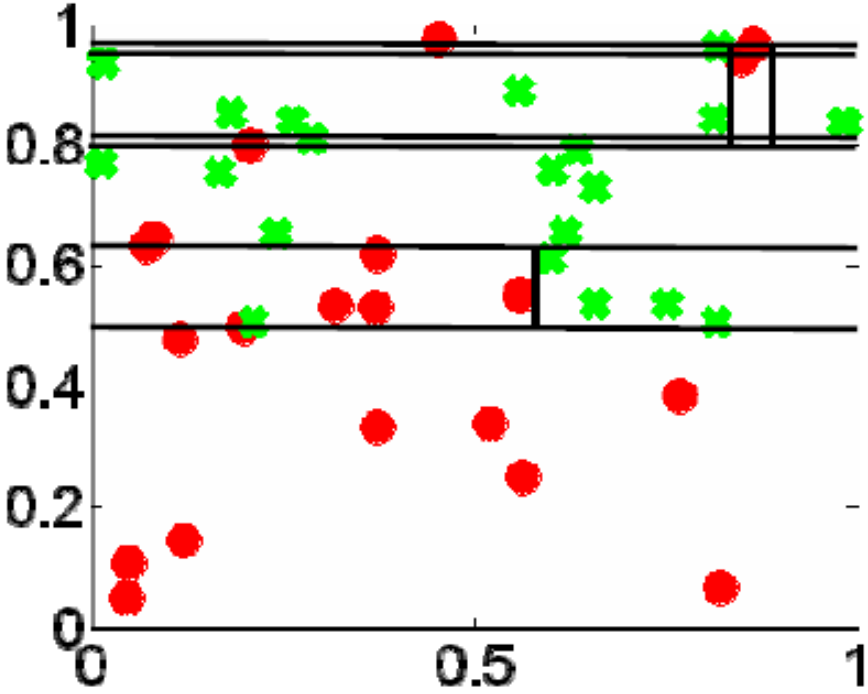
- Grow tree based on training data (unpruned tree)
- Prune the tree by removing useless nodes based on:
  - Additional test data (also known as validation data)



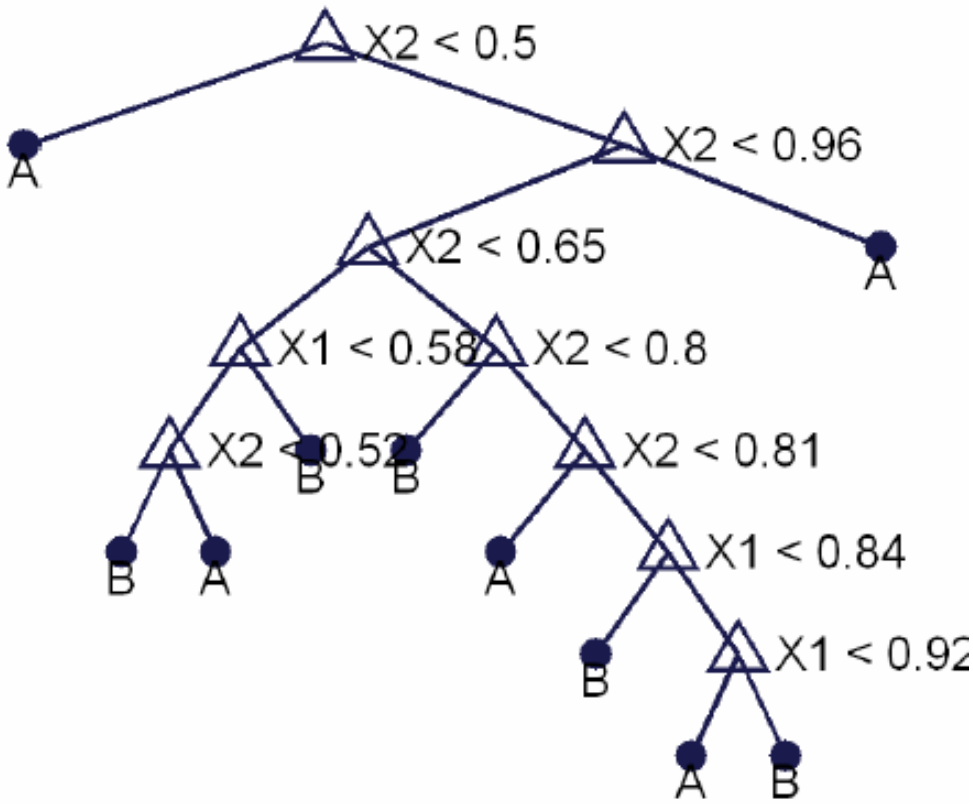
Training data



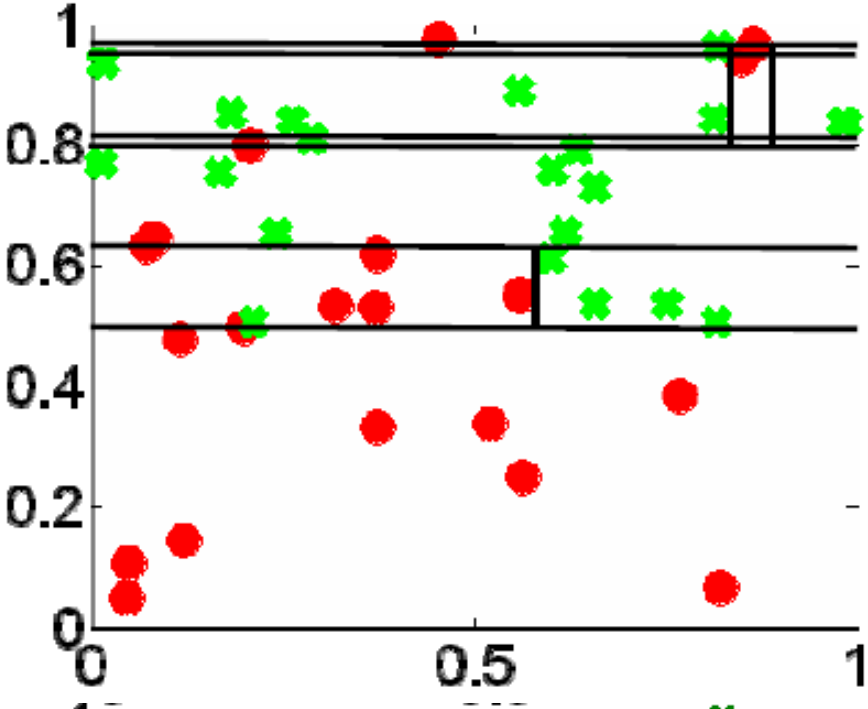
Unpruned decision tree from training data

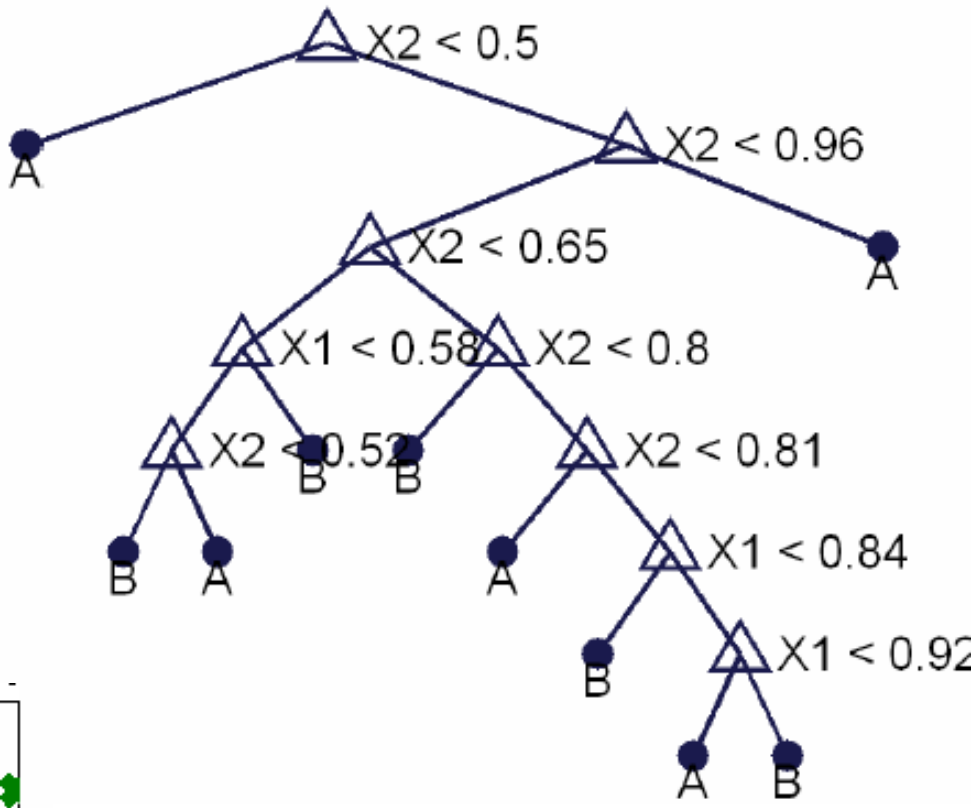


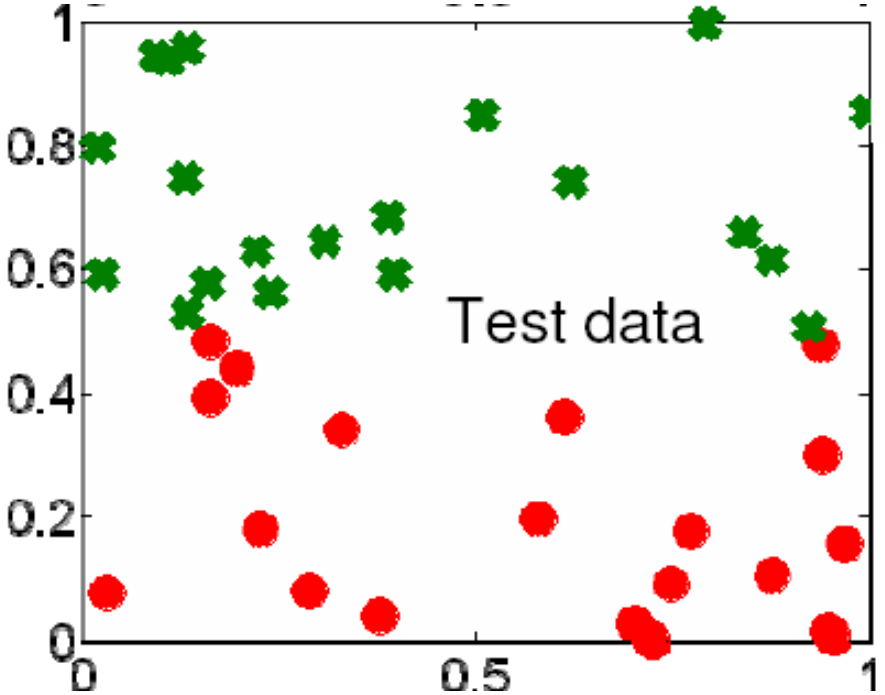
Training data
with the partitions induced by the decision tree
(Notice the tiny regions at the top necessary to correctly classify the ‘A’ outliers!)



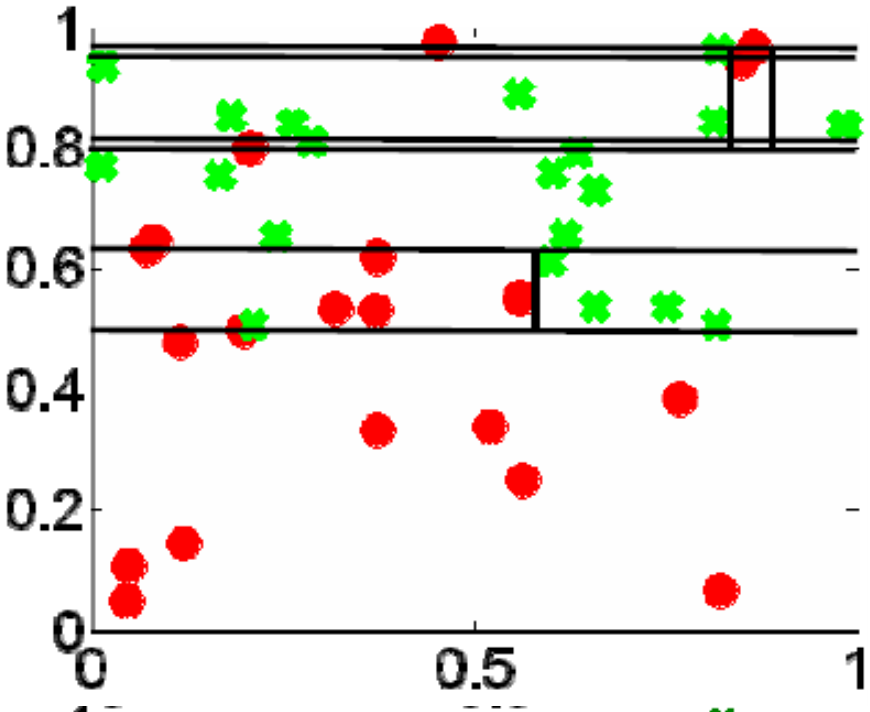
Unpruned decision tree from training data

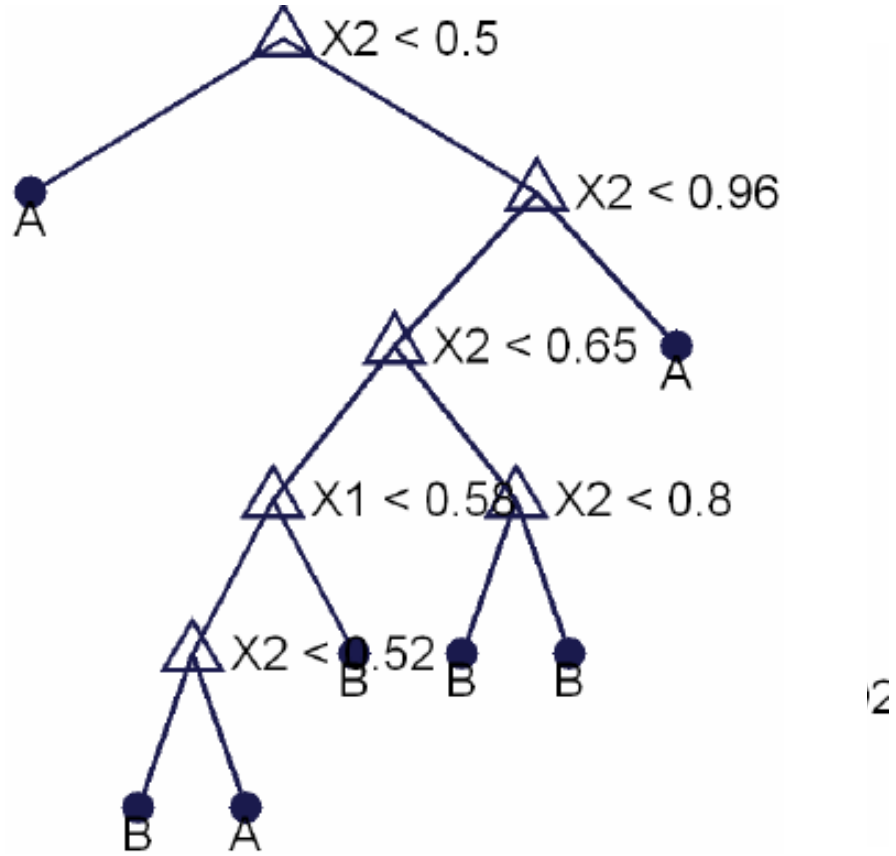


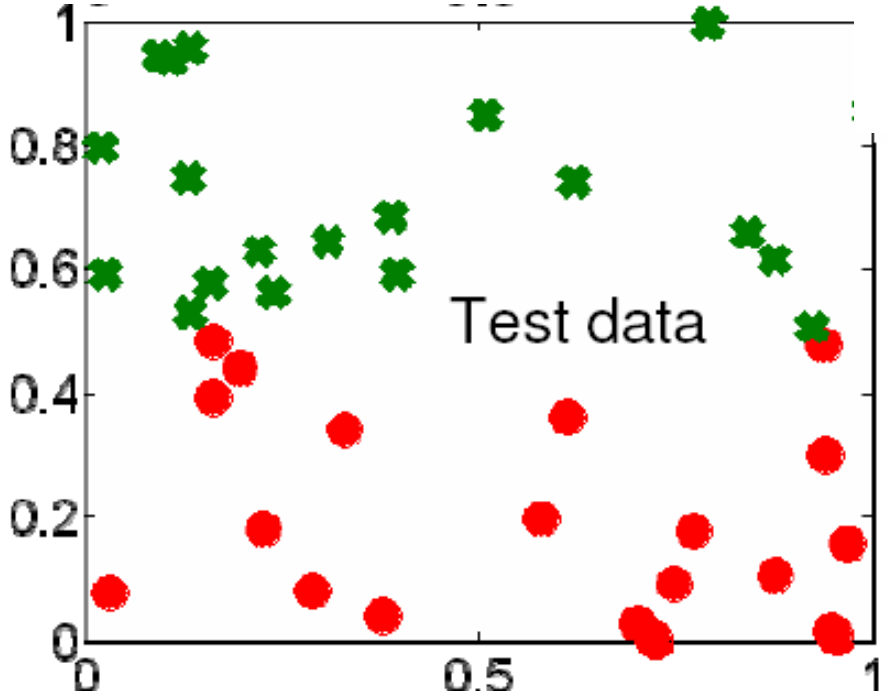




Unpruned decision tree from training data
Performance (% correctly classified)
Training: 100%
Test: 77.5%





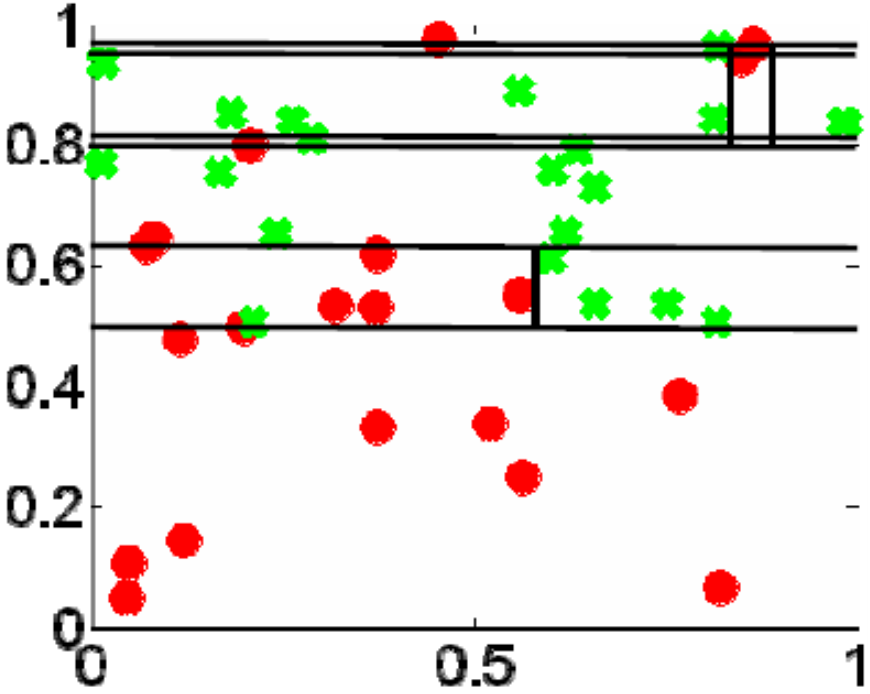


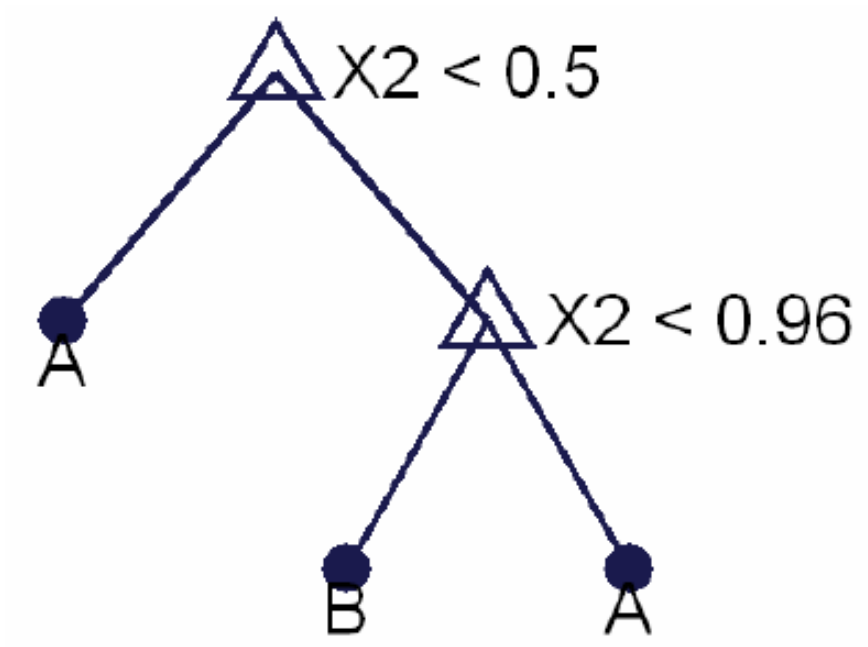
Pruned decision tree from training data

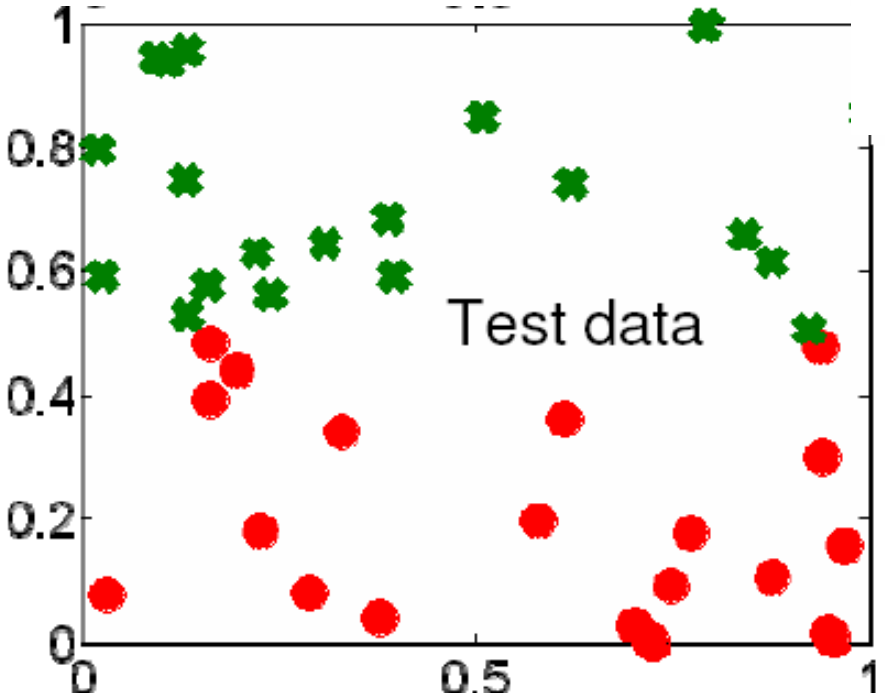
Performance (% correctly classified)

Training: 95%

Test: 80%





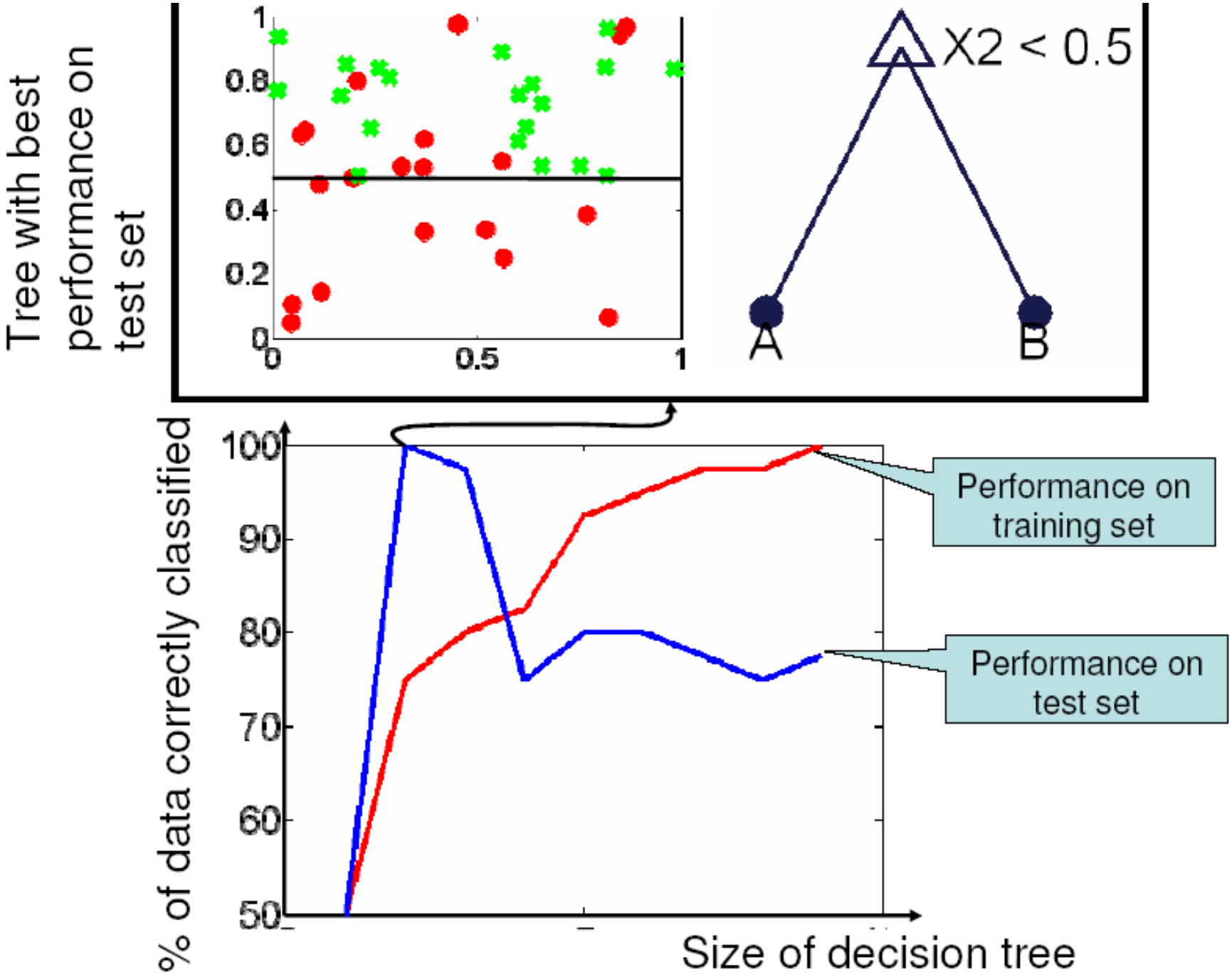


Pruned decision tree from training data

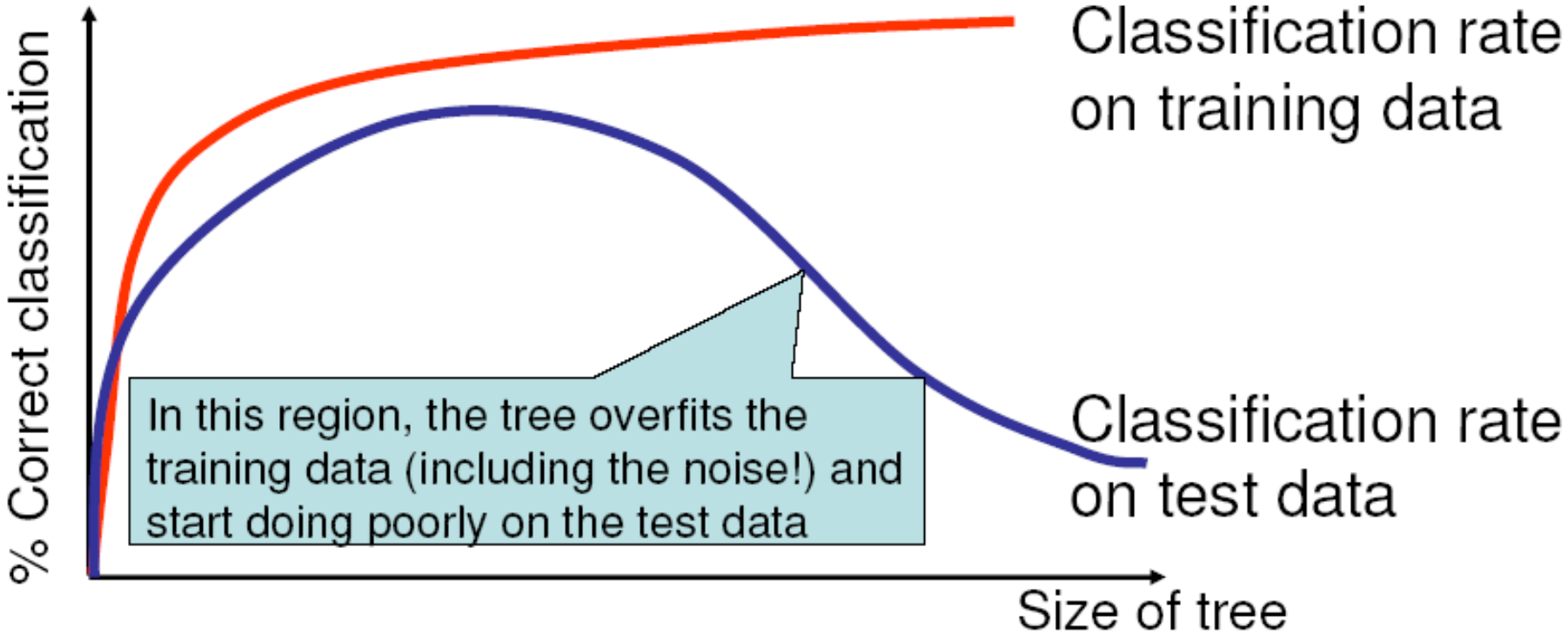
Performance (% correctly classified)

Training: 80%

Test: 97.5%


Tree with best performance on test set
1
0.8
0.6
0.4
0.2
0
0
0.5
1
X2 < 0.5
A
B
% of data correctly classified
100
90
80
70
60
50
Size of decision tree
Performance on training set
Performance on test set

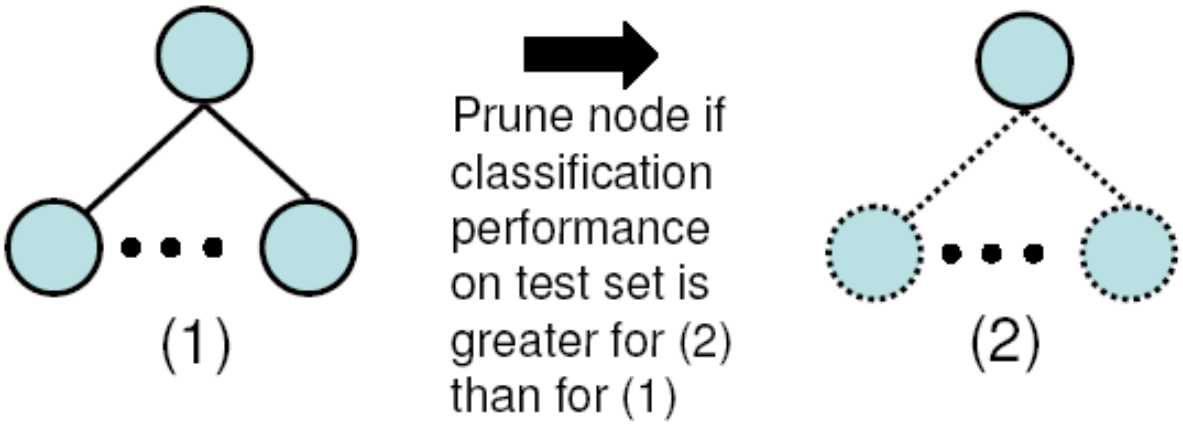
Locating the Overfitting Point



- General principle: As the complexity of the classifier increases (depth of the decision tree), the performance on the training data increases and the performance on the test data decreases when the classifier overfits the training data.

Decision Tree Pruning

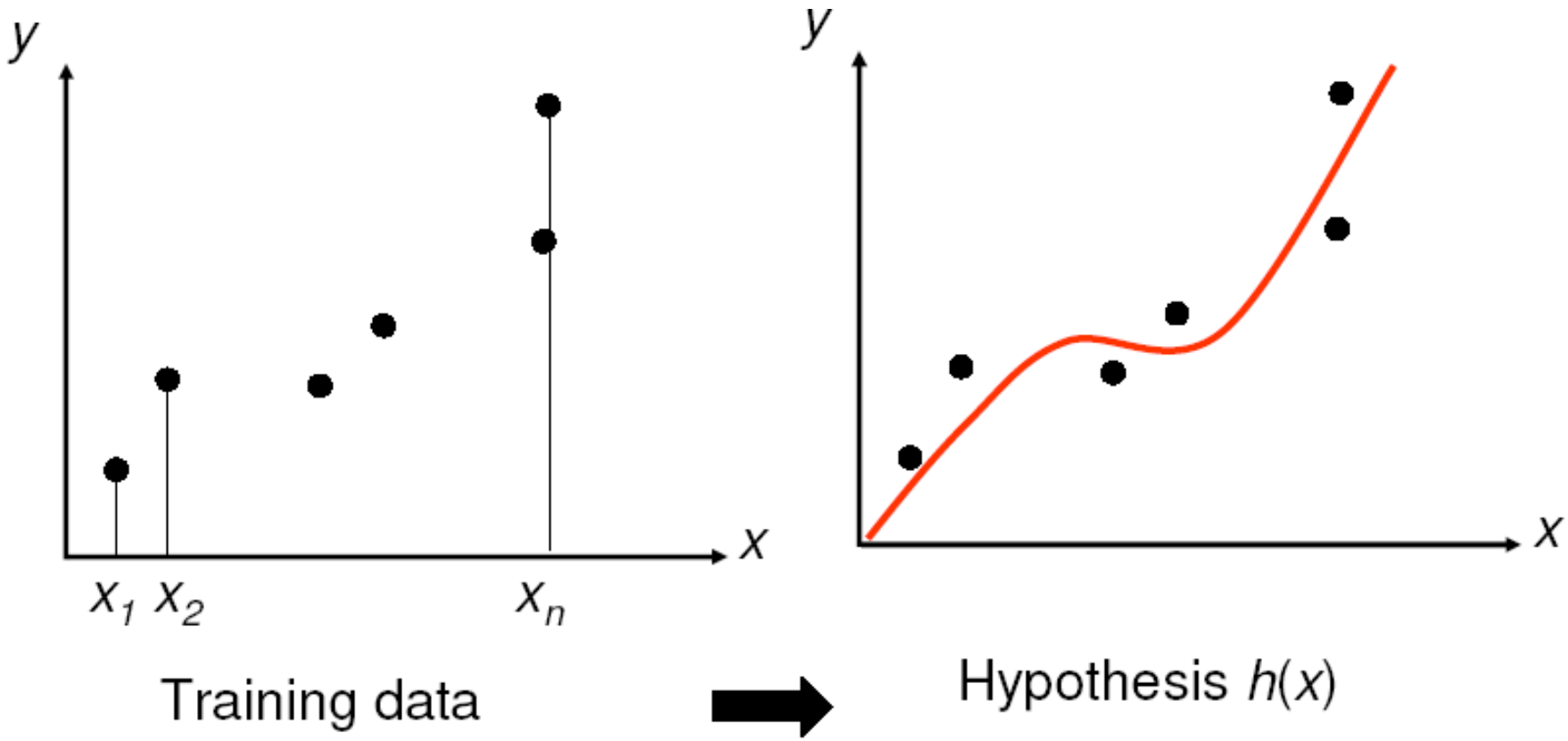
- Construct the entire tree as before
- Starting at the leaves, recursively eliminate splits:
  - Evaluate performance of the tree on additional test data (also known as validation data)
  - Prune the tree if the classification performance increases by removing the split



Inductive Learning

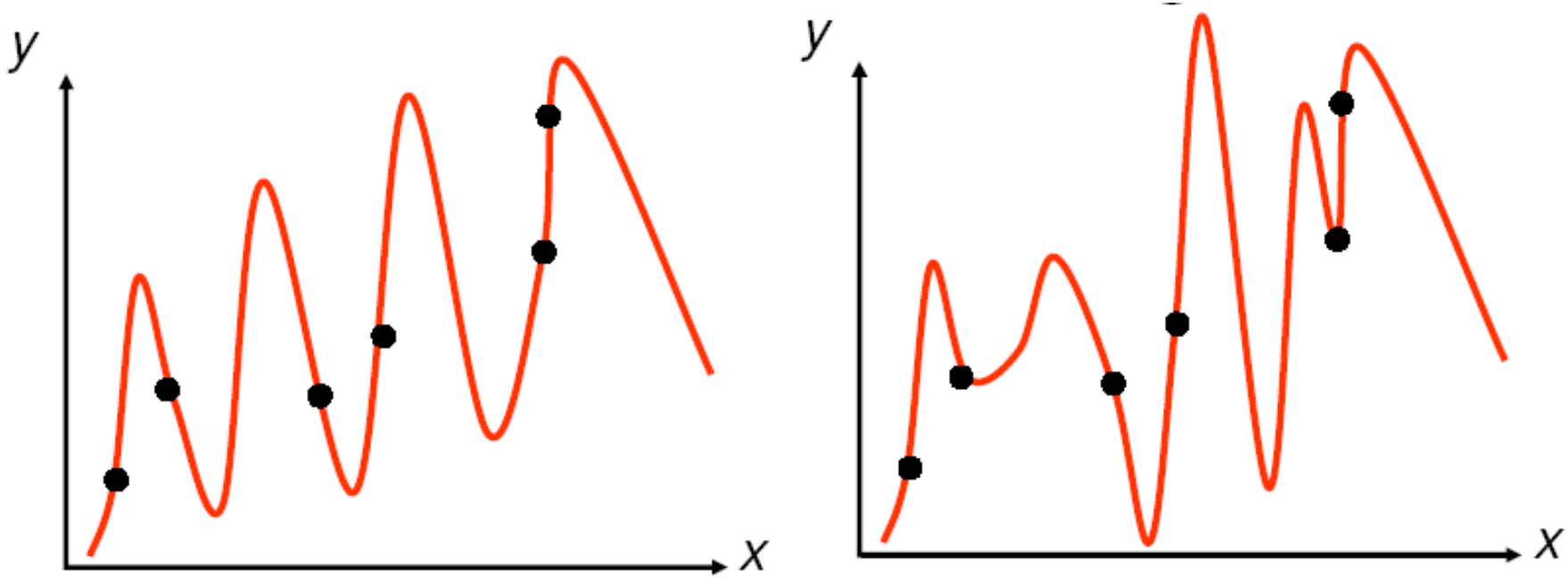
- The decision tree approach is one example of an inductive learning technique:
- Suppose that data x is related to output y by a unknown function $y = f(x)$
- Suppose that we have observed training examples $\{(x_1,y_1),..,(x_n,y_n)\}$
- *Inductive learning problem*: Recover a function h (the "hypothesis") such that $h(x) \approx f(x)$
- $y = h(x)$ predicts y from the input data x
- *The challenge*: The hypothesis space (the space of all hypothesis h of a given form; for example the space of all of the possible decision trees for a set of M attributes) is huge + many different hypotheses may agree with the training data.

Inductive Learning



- What property should *h* have?
- It should agree with the training data…

Inductive Learning

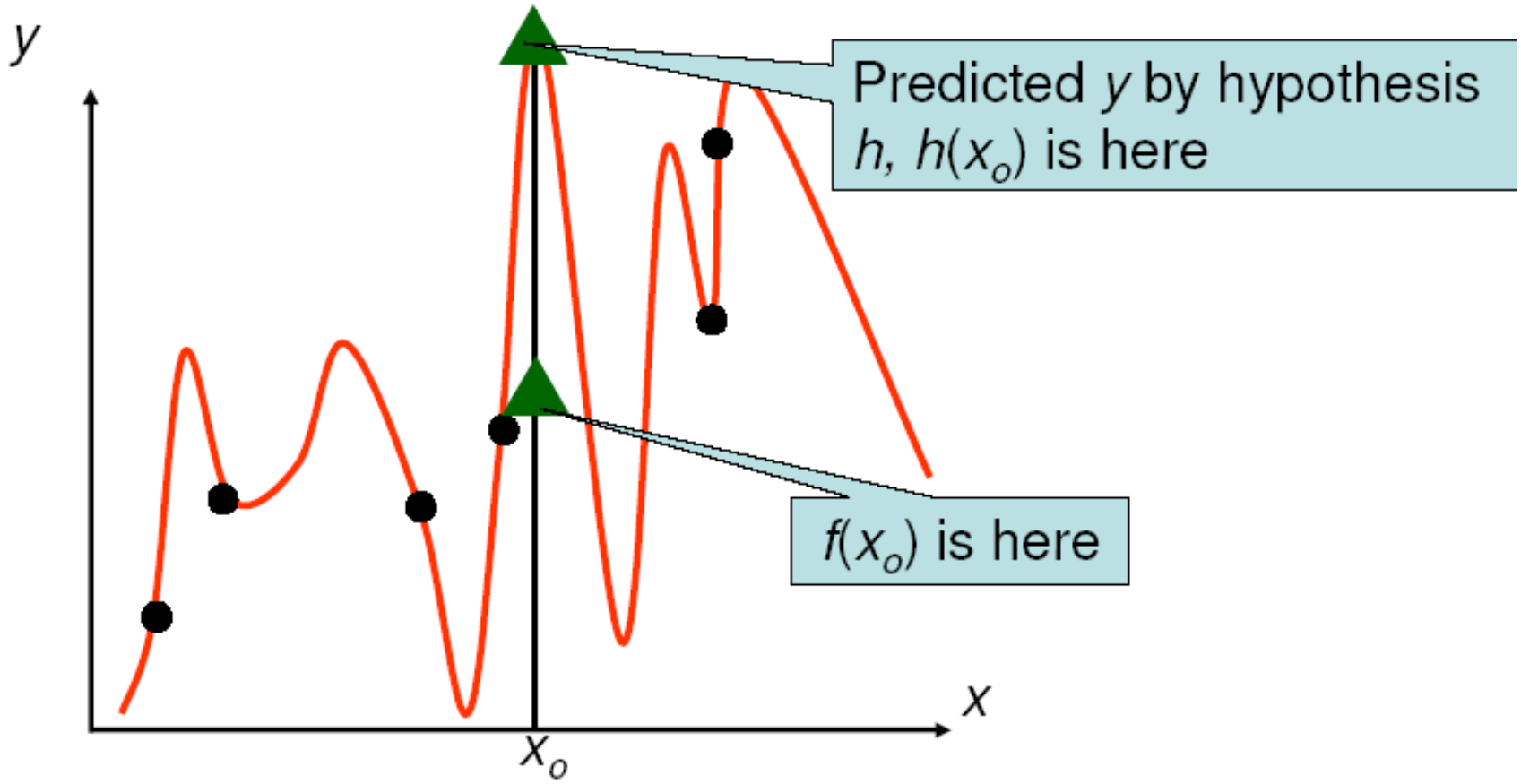


Two stupid hypotheses that fit the training data perfectly

- What property should *h* have?
- It should agree with the training data…
- But that can lead to arbitrarily complex hypotheses

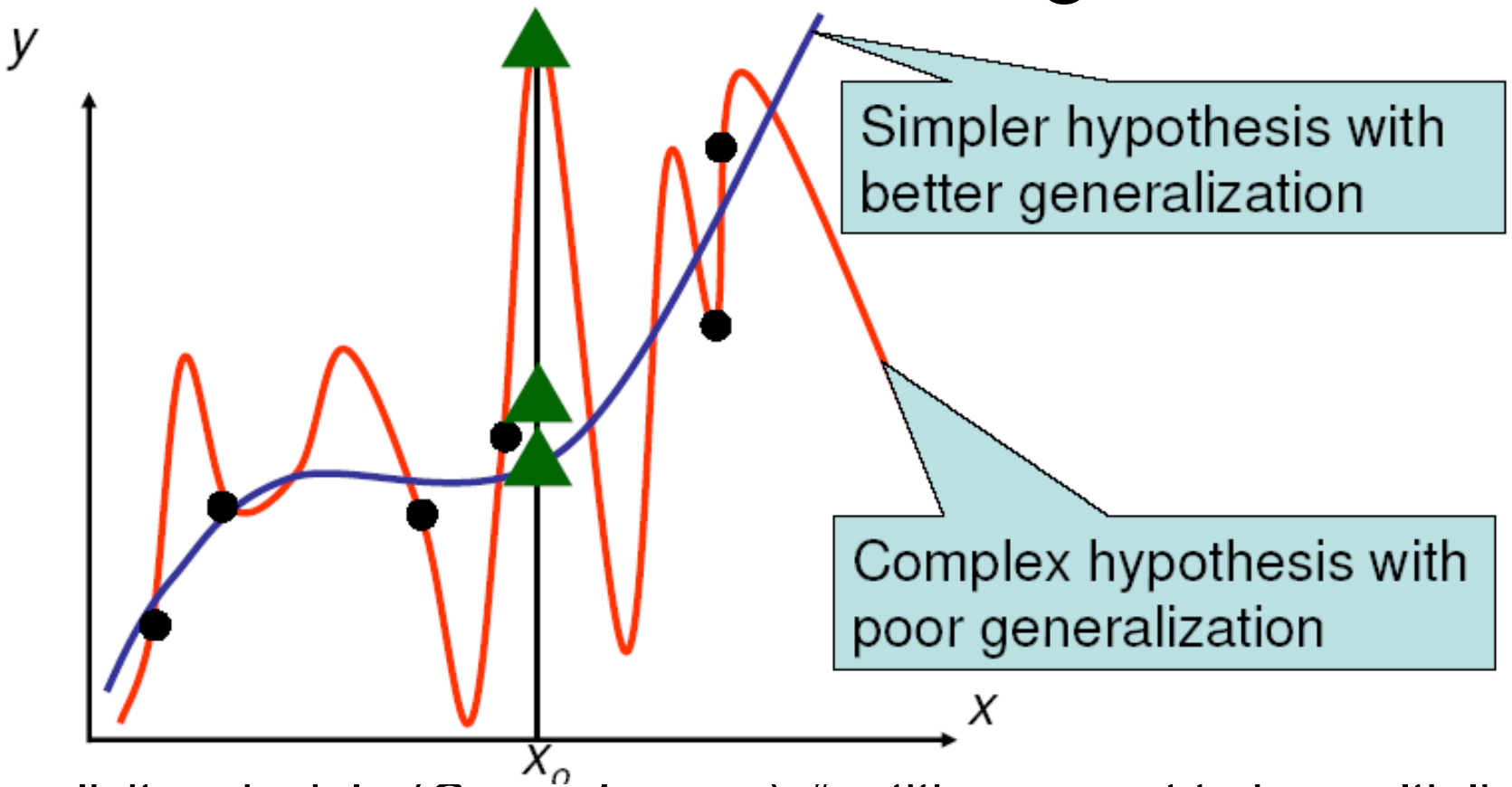
  and there are many of them; which one should we choose?…

Inductive Learning



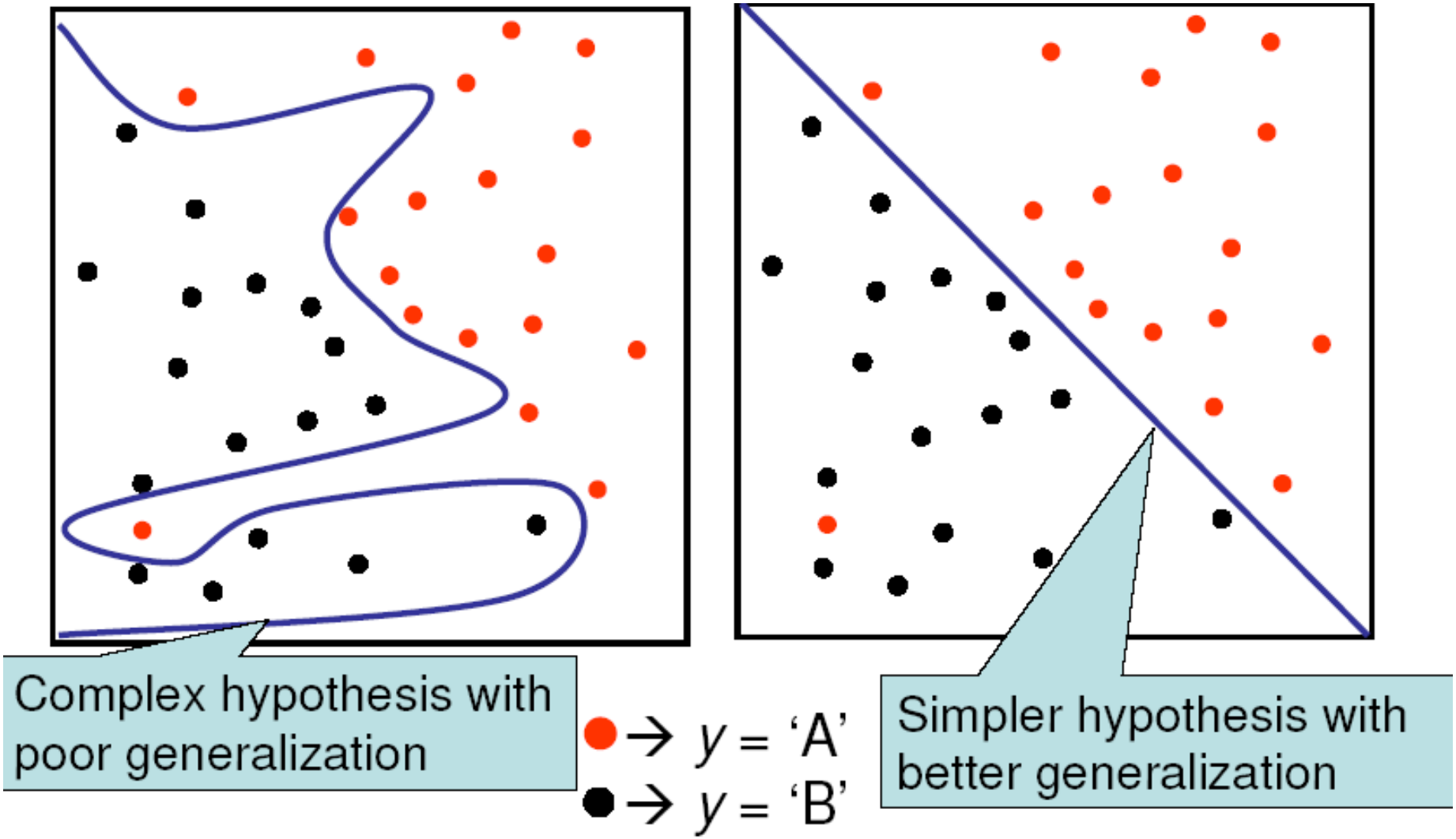
- Problems with a complex hypothesis:
  - It leads to completely wrong prediction on new test data…
  - It does not *generalize* beyond the training data…it overfits the training data

Inductive Learning



- Simplicity principle (*Occam's razor*): "entities are not to be multiplied beyond necessity"
- The simpler hypothesis is preferred
- Compromise between:
  - Error on data under hypothesis *h*
  - Complexity of hypothesis *h*

Inductive Learning



- Different illustration, same concept….

Inductive Learning

- Decision tree is one example of inductive learning
- In many supervised learning algorithms, the goal is to minimize:

  Error on data + complexity of model

Summary: Decision Trees

- Information Gain (IG) criterion for choosing splitting criteria at each level of the tree.
- Versions with continuous attributes and with discrete (categorical) attributes
- Basic tree learning algorithm leads to overfitting of the training data
- Pruning with validation data (not used for training)
- Example of inductive learning

Decision Trees on Real Problems

Must consider the following issues:

1. Assessing the performance of a learning algorithm
2. Inadequate attributes
3. Noise in the data
4. Missing values
5. Attributes with numeric values
6. Bias in attribute selection

Assessing the Performance of a Learning Algorithm

Performance task: predict the classifications of unseen examples

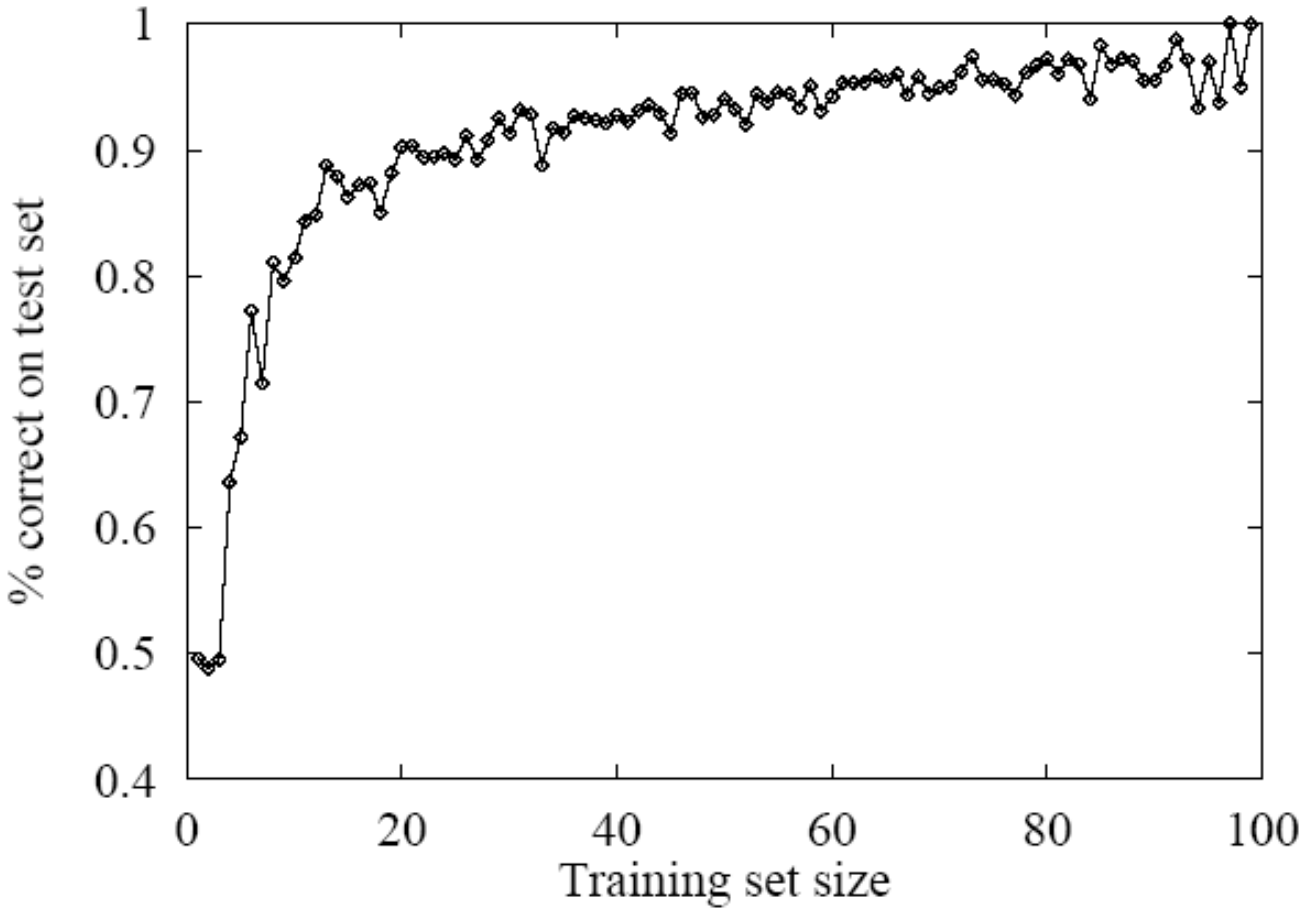
Assessing prediction quality after tree construction: check the classifier's predictions on a **test set**.

But this requires that we get more data after we have trained the classifier.

Evaluation Methodology

1. Collect a large set of examples
2. Divide it into two disjoint sets: the **training set** and the **test set**
3. Use the learning algorithm with the training set to generate a hypothesis H
4. Measure the percentage of examples in the test set that are classified correctly by H
5. Repeat steps 1 to 4 for different sizes of training sets and different randomly selected training sets of each size.

Learning Curve



Inadequate Attributes

- Cause inconsistent instances. Cannot find a classifier that is consistent with all of the training instances.

Solutions:

1. have each leaf node report the majority class
2. have each leaf report the estimated probabilities of each classification using the relative frequencies

Noisy Data

Incorrect attribute values. Incorrect class labels.

Noise can be caused by many factors, such as:

1. Faulty measurements
2. Subjective interpretation

A further complication: may or may not know whether data is noisy.

Solution: pruning

Unknown Attribute Values

1. Throw away instances during training; during testing, try all paths, letting leaves vote.
2. Take most common value.
3. Take fractional value.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	???	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Attributes with Numeric Values

Look for best splits.

1. Sort values
2. Create Boolean variables out of mid points
3. Evaluate all of these using information gain formula

- Example:

Length (L):	10	15	21	28	32	40	50
Class:	-	+	+	-	+	+	-

Bias in Attribute Selection

Problem: Metric chooses higher branching attributes

Solution: Take into account the branching factor

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes