CS440/ECE448: Intro to Artificial Intelligence

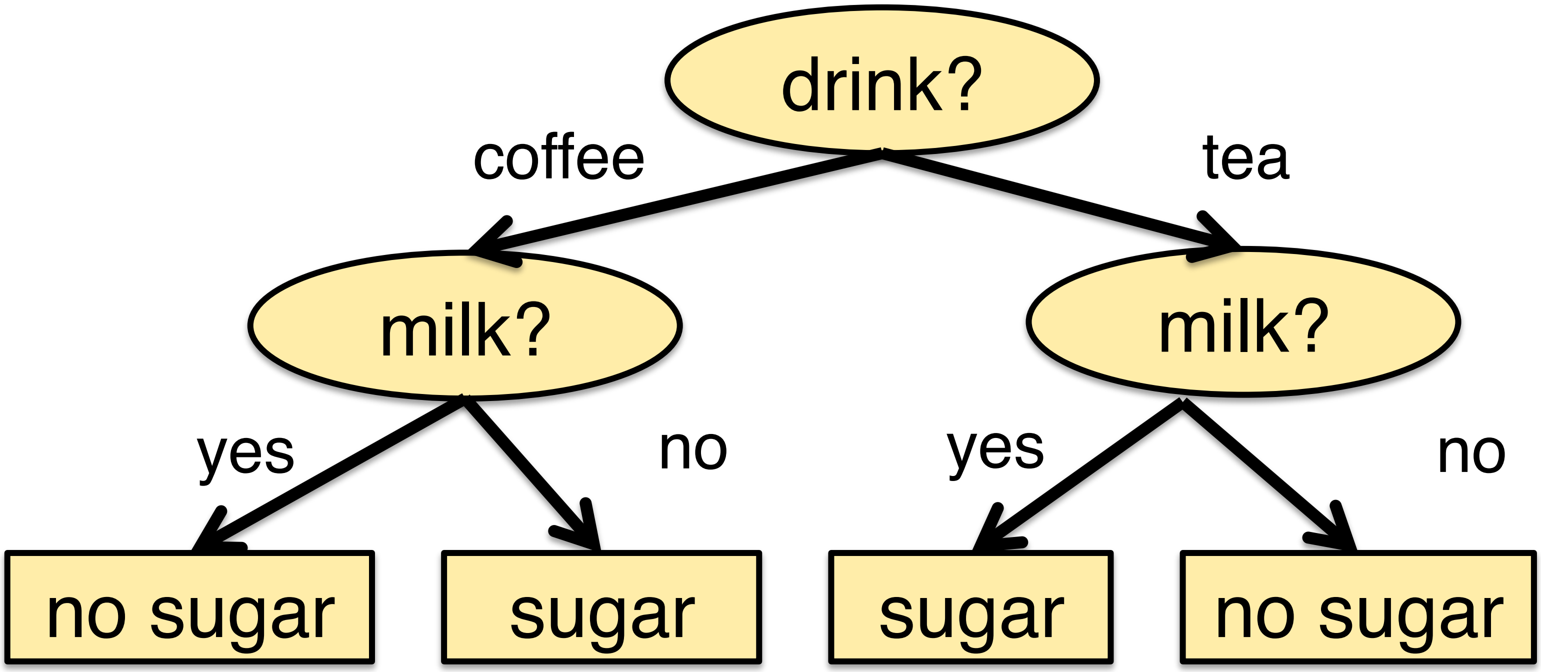
# Lecture 23: Decision Trees

Prof. Julia Hockenmaier
juliahmr@illinois.edu

http://cs.illinois.edu/fa11/cs440

# Decision trees

# Decision trees



# Decision tree learning

Training data $D = \{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}$

– each $\mathbf{x}^i = (x_1^i, \ldots., x_d^i)$ is a *d*-dimensional feature vector

– each $y_i$ is the target label (class) of the i-th data point

Training algorithm:

– Initial tree = the root, corresponding to all items in $D$

– A node is a leaf if all its data items have the same $y$

– At each non-leaf node: find the feature $x_i$ with the highest information gain, create a new child for each value of $x_i$, distribute the items accordingly.

# Information Gain

How much information are we gaining by splitting node S on attribute A with values V(A)?

Information required before the split:

$$H(S_{parent})$$

Information required after the split:

$$\sum_{i \in V(A)} P(S_{child\_i}) H(S_{child\_i})$$

$$Gain(S_{parent}, A) = H(S_{parent}) - \sum_{i \in V(A)}^{N} H(S_{child_i}) \frac{|S_{child_i}|}{|S_{parent}|}$$

# Dealing with numerical attributes

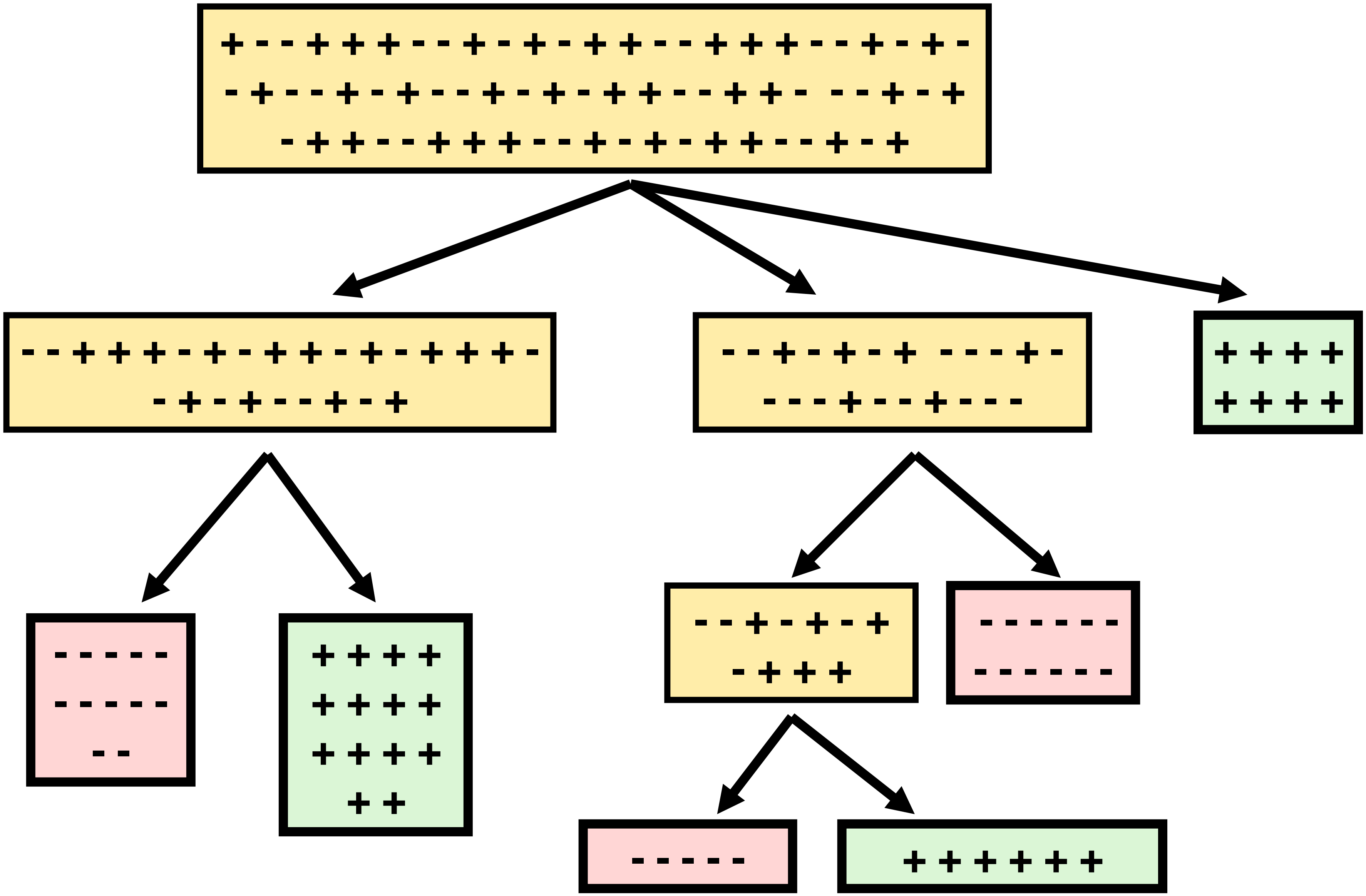
Many attributes are not boolean (0,1)
or nominal (classes)

- Number of times a word appears in a text
- RGB values of a pixel
- height, weight, ….

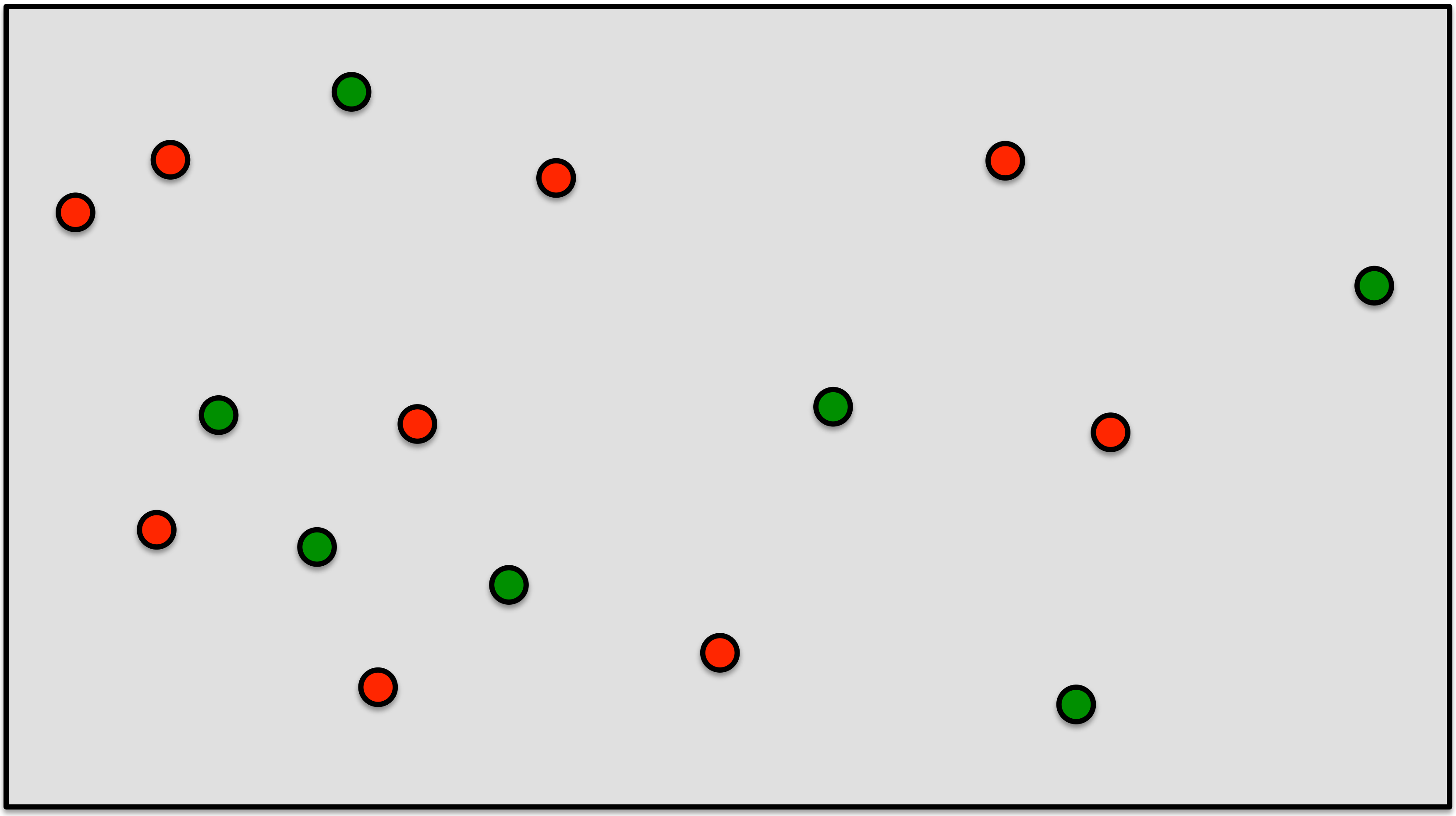
Splitting on integer or real-valued attributes:

- Find a split point: $A_i < \theta$ or $A_i \geq \theta$ ?

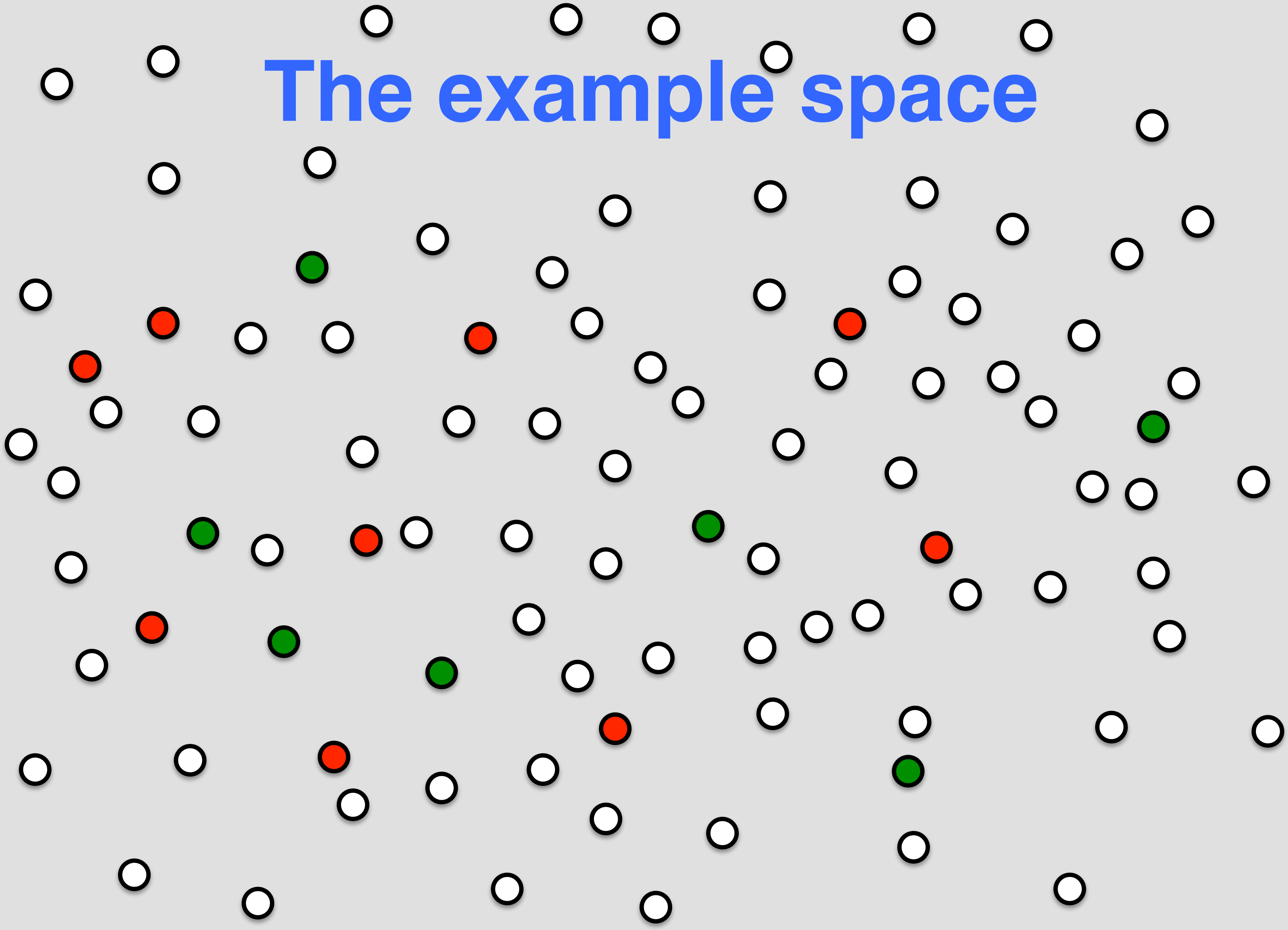
Complete Training Data



# Our training data



# The example space



# Generalization

We need to label unseen examples accurately.

But:
The training data is only a very small sample of the example space.

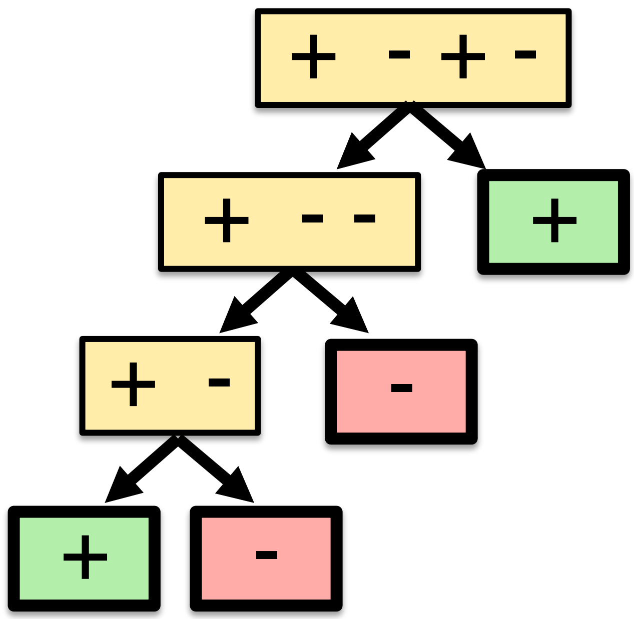
– We won't have seen all possible combinations of attribute values.

The training data may be noisy

– Some items may have incorrect attributes or labels

# When does learning stop?

The tree will grow until all leaf nodes have only one label.

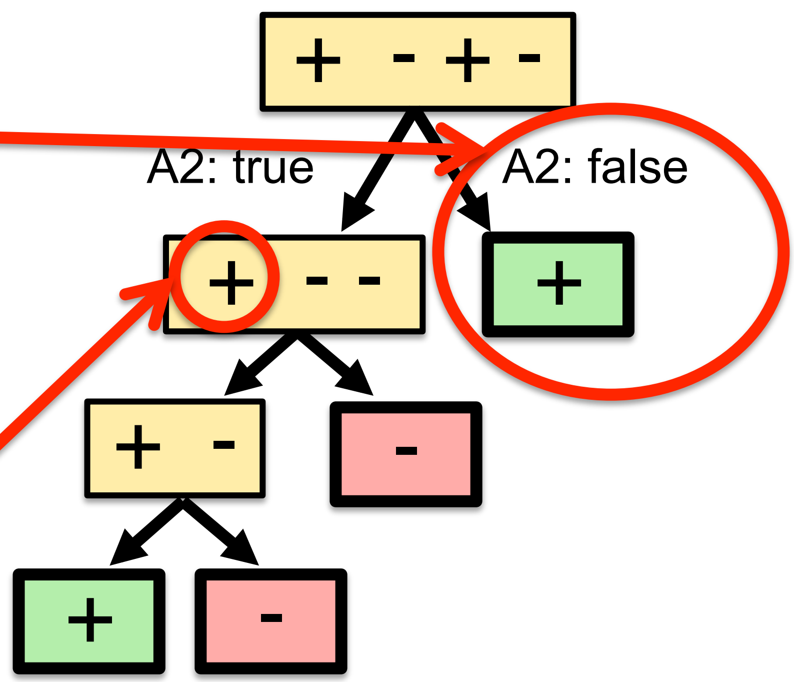


# The effect of noise

If the training data are noisy,
it may introduce incorrect splits.

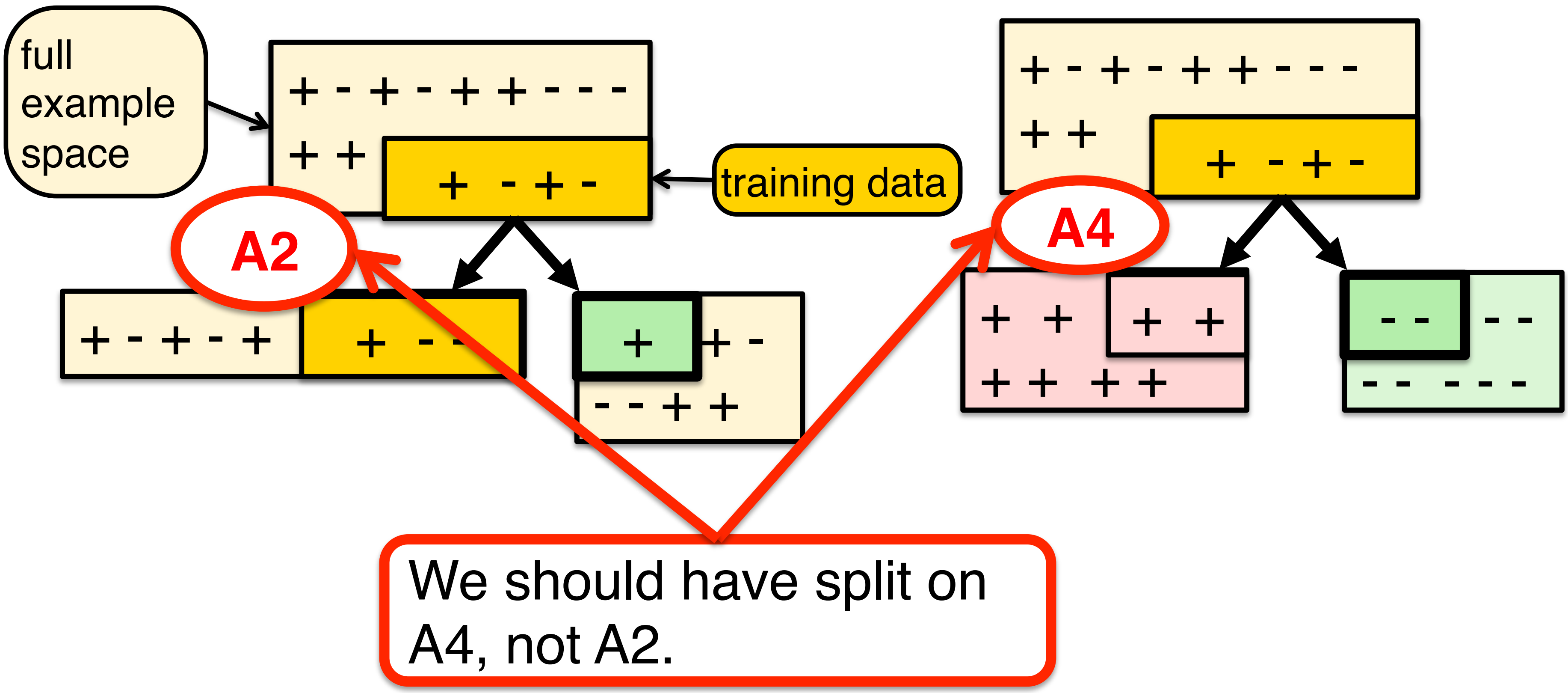
If this *false* **value** should have been *true*, we wouldn't split on *A2.*

If this + **label** should have been -, we wouldn't have to split any further.



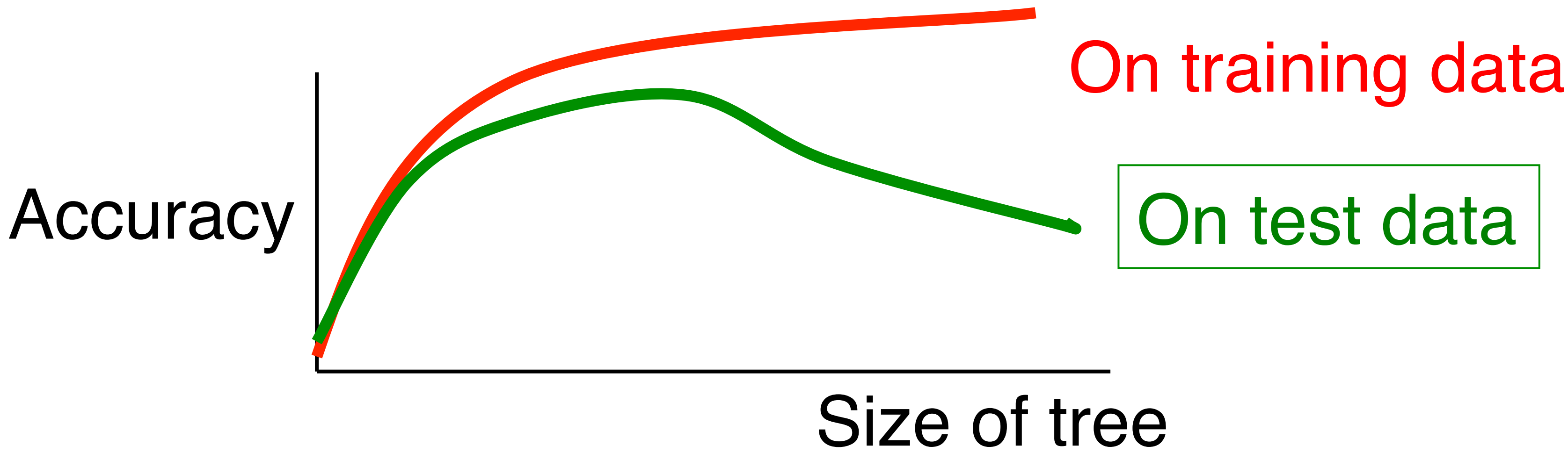
# The effect of incomplete data

If the training data are incomplete,
we may miss important generalizations.



# Overfitting

The decision tree might overfit the particularities of the training data.



# Reducing Overfitting in Decision Trees

Limit the depth of the tree

- No deeper than N (say 3 or 12 or 86 - how to choose?)

Require a minimum number of examples used to select a split

- Need at least M (is 10 enough? 20?)
- Want significance: Statistical hypothesis testing can help

BEST: Learn an overfit tree and prune, using validation (held-out) data

# Pruning a decision tree

1. Train a decision tree on training data
   (keep a part of training data as unseen validation data)

2. Prune from the leaves:
   Simplest method:
      Replace (prune) each non-leaf node whose children are all leaves with its majority label.
      Keep this change if the accuracy on validation set does not degrade.

# Dealing with overfitting

Overfitting is a very common problem in machine learning.

Many machine learning algorithms have parameters that can be tuned to improve performance (because they reduce overfitting).

We use a held-out data set to set these parameters.

# Bias-variance tradeoff

Bias: What kind of hypotheses do we allow?

We want rich enough hypotheses to capture the target function $f(\mathbf{x})$

Variance: How much does our learned hypothesis change if we resample the training data?

Rich hypotheses (e.g. large decision trees) need more data (which we may not have)

# Reducing variance: bagging

Create a new training set by sampling (with replacement) N items from the original data set.

Repeat this K times to get K training sets.
(K is an odd number, e.g. 3, 5, …)

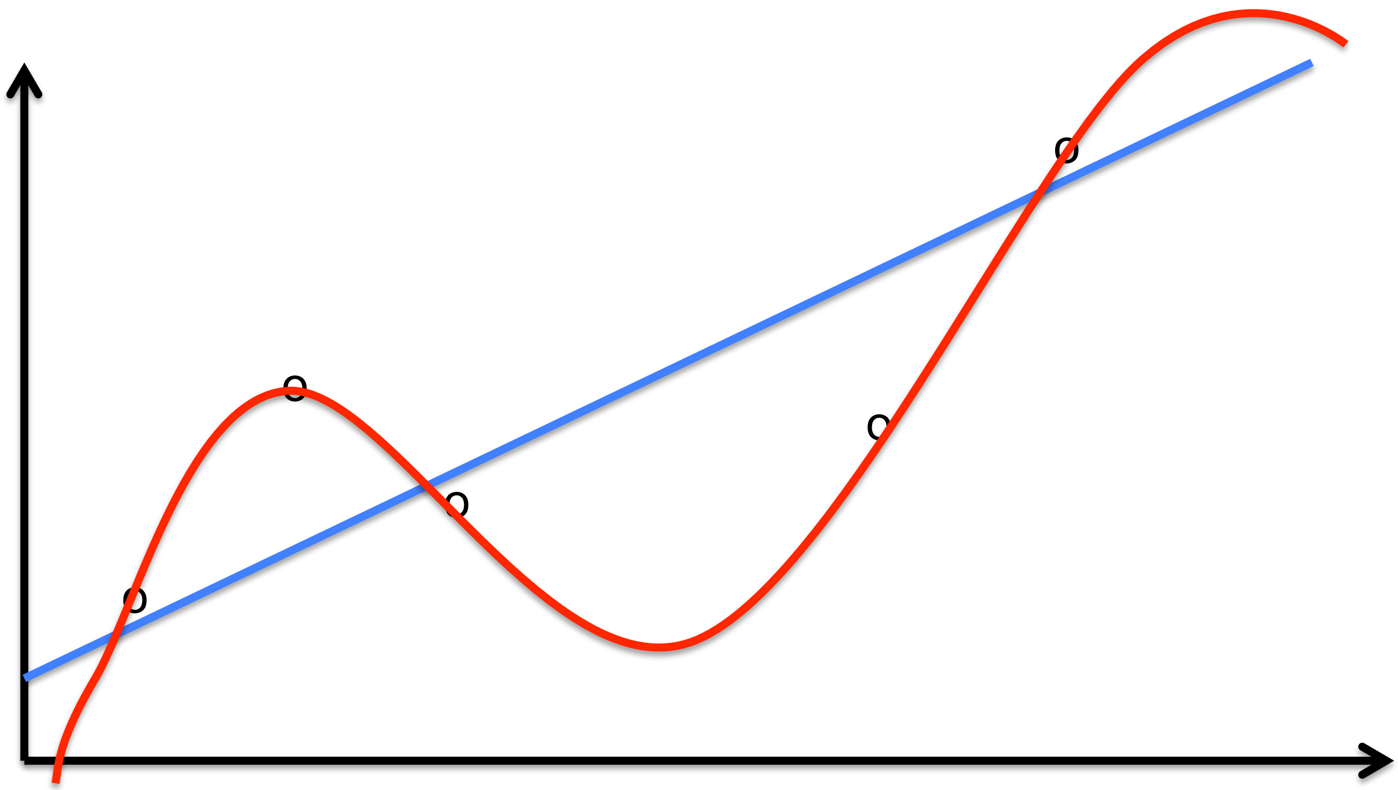
Train one classifier on each of the K training sets

Testing: take the majority vote of these K classifiers

# Regression

# Polynomial curve fitting

Given some data {(x,y)…}, with x, y ∈ R, find a function f such that f(x) = y.



# Polynomial curve fitting

$$f(x) = w_0 + w_1 x^1 + w_2 x^2 + \ldots + w_m x^m$$

$$= \sum_{i=0}^{m} w_i x^i$$

Task:

find weights $w_0 \ldots w_m$ to best fit the data.

This requires a loss (error) function

# Squared Loss

We want to find a weight vector **w** which minimizes the loss (error) on the training data $\{(x_1,y_1)\ldots(x_N, y_N)\}$

$$L(\mathbf{w}) = \sum_{i=1}^{N} L_2(f_{\mathbf{w}}(x_i), y_i)$$

$$= \sum_{i=1}^{N} (y_i - f_{\mathbf{w}}(x_i))^2$$

# Accounting for model complexity

We would like to find the simplest polynomial to fit our data.

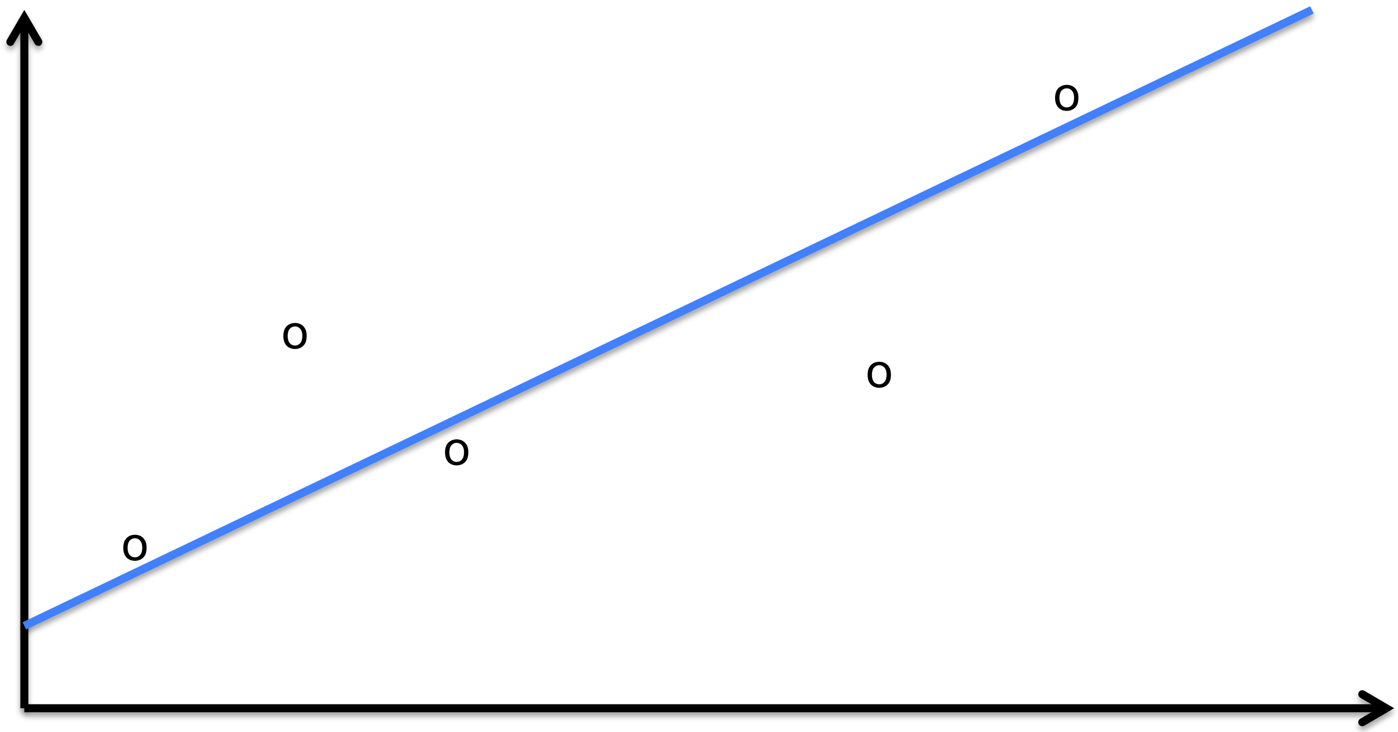
We need to penalize the degree of the polynomial.

We can add a regularization term to the loss which penalizes for overly complex functions)

# Regression

# Linear regression

Given some data {(x,y)…}, with $x, y \in R$,
find a function $f(x) = w_1 x + w_0$ such that $f(x) = y$.



# Linear regression

We need to minimize the loss on the training data: $\mathbf{w} = \text{argmin}_{\mathbf{w}} \text{Loss}(f_{\mathbf{w}})$

We need to set partial derivatives of $\text{Loss}(f_{\mathbf{w}})$ with respect to w1, w0 to zero.

This has a closed-form solution (see book).