Lecture 4

Oct - 6 - 2008

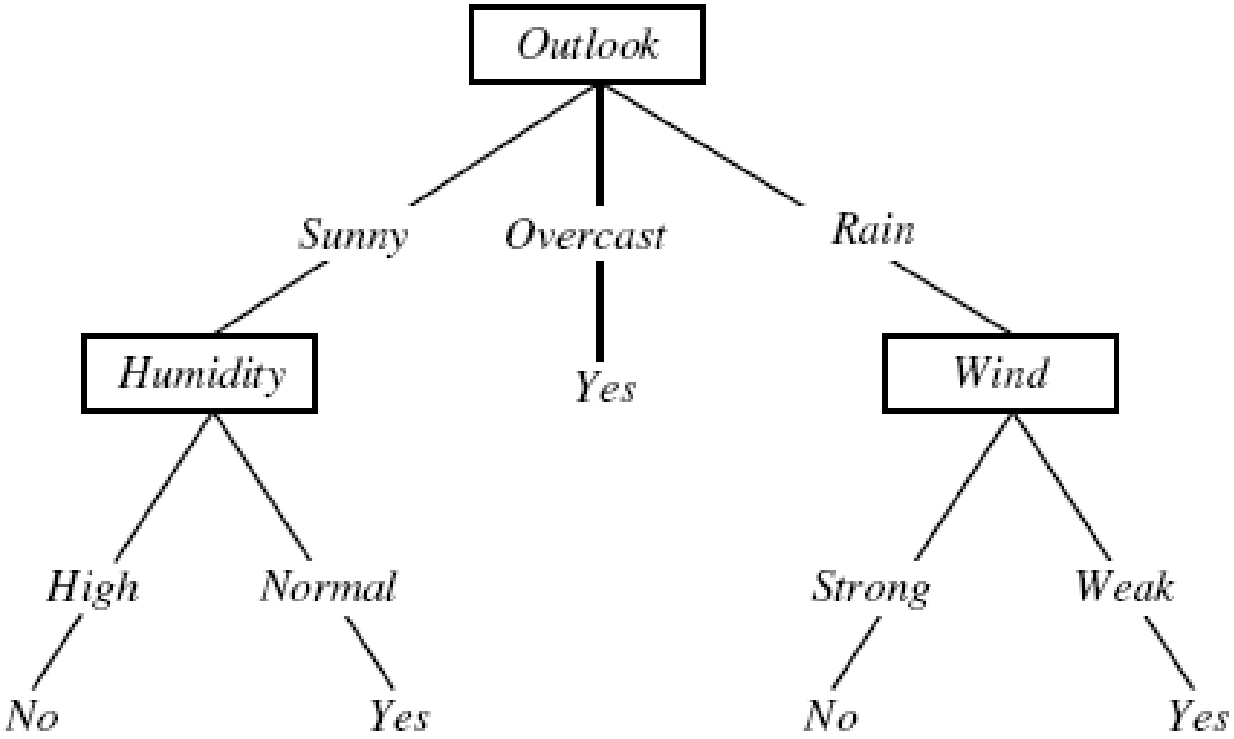
Review from last lecture

- Nearest neighbor classifier
  - A lazy learning algorithm
  - Decision boundary can be obtained by the Voronoi diagram of the training set
  - Complex, sensitive to label noise
- K-nearest neighbor, how to select k? a model selection problem
  - Use training error – bad idea
  - Use validation error – better
  - Use cross-validation error – even better
- Issues with KNN
  - Features need to be normalized to the same range
  - Computational cost – high
  - Irrelevant features – bad for kNN, which assumes all features are equally important
  - Last but not least: finding the right distance metric can be difficult

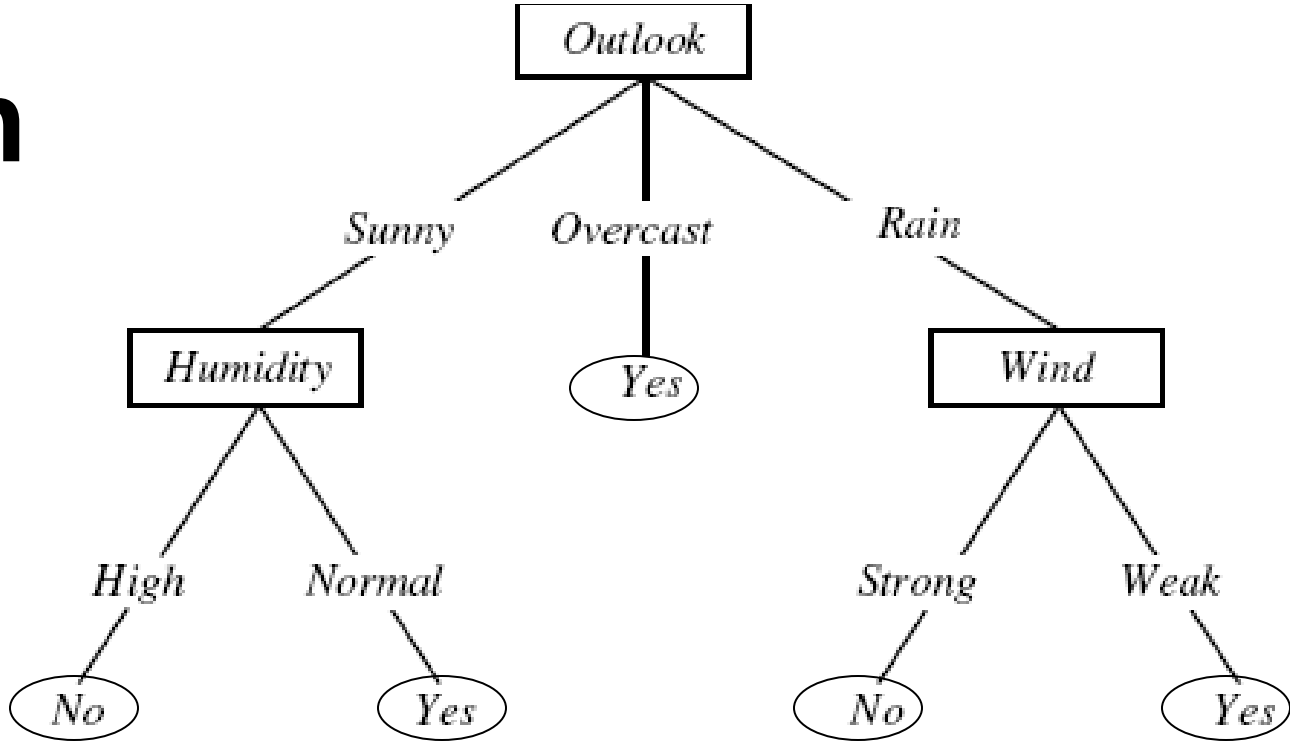
- So far we've learned two classifiers
  - Perceptron: LTU
  - KNN: complex decision boundary
- We have paid special attention on some of the issues such as
  - Is the learning algorithm robust to outliers?
  - Is the learning algorithm sensitive to irrelevant features?
  - Is the algorithm computationally scalable?
  - We will continue to pay attention to these issues as we introduce more learning algorithms

Decision Tree
One of the most popular off-the-shelf classifiers

What is decision tree: an example



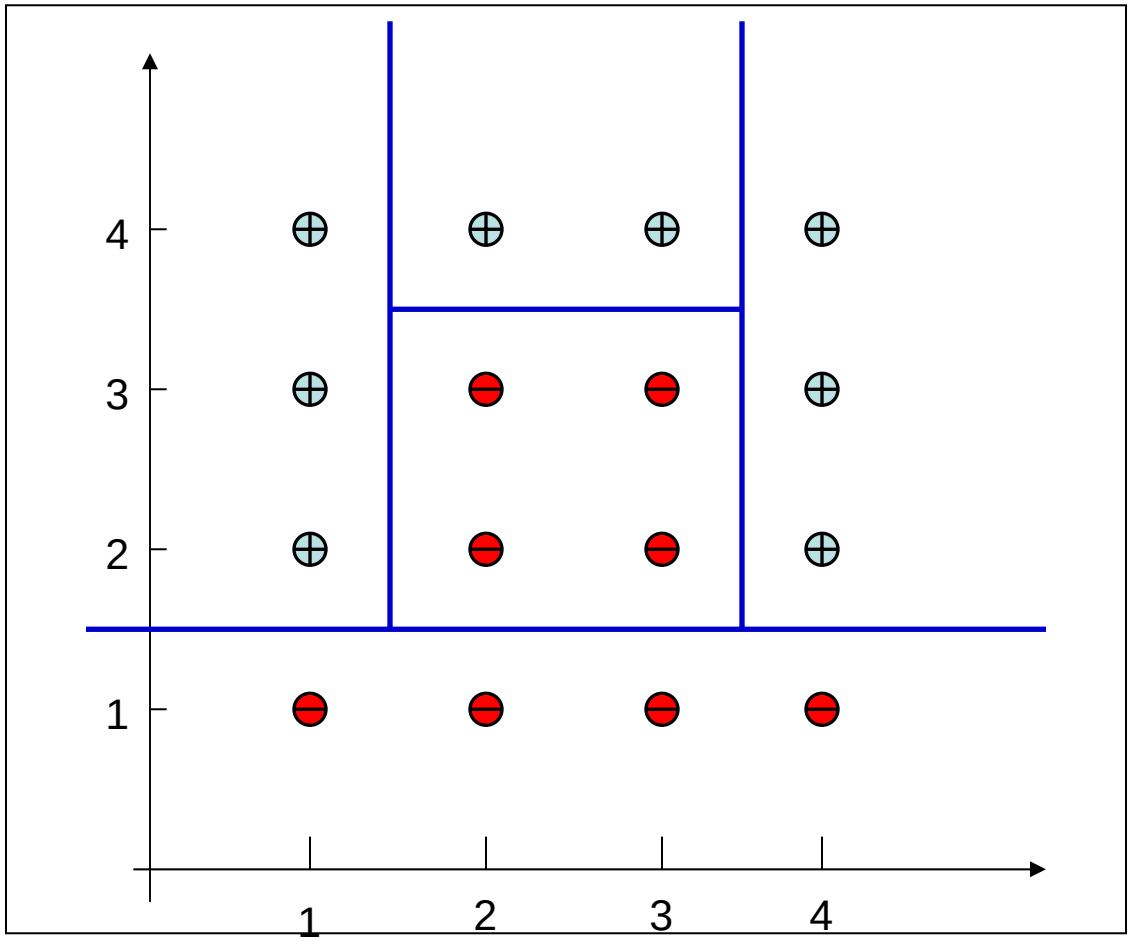
Definition

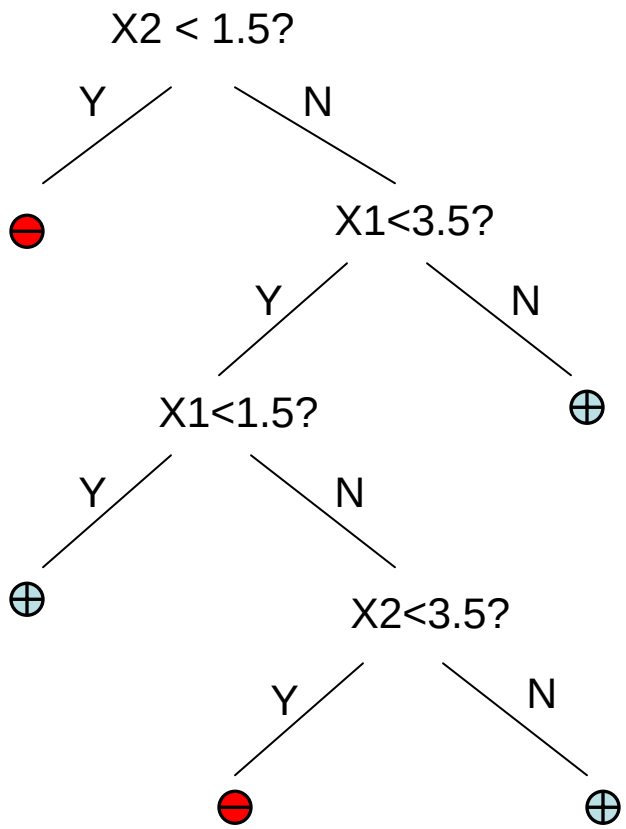


- **Internal nodes (rectangles)**
  - Each node presents a test on a particular attribute
  - Multiple possible outcomes lead to branches of the tree
  - For discrete attributes (outlook = sunny, overcast or wind)
    - n possible values -> n branches
  - Continuous attributes (temperature = 87 F)?
- **Leaf nodes (elipses)**
  - Each assign a class label to all examples that end up there,

Decision Tree Decision Boundaries

- Decision Trees divide the input space into axis-parallel rectangles and label each rectangle with one of the K classes





Characteristics of Decision Trees

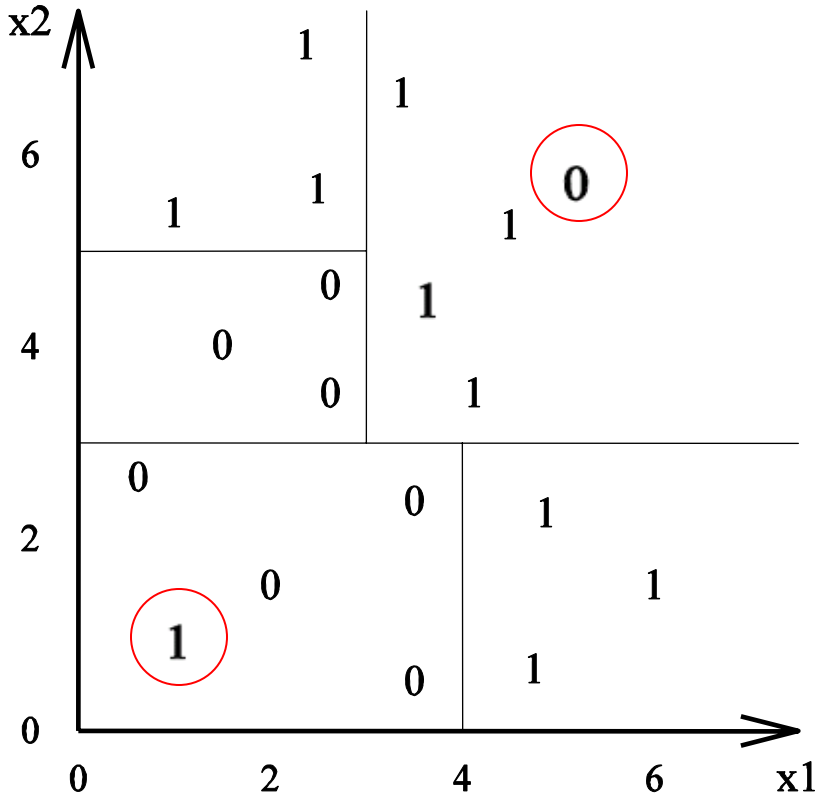
- Decision trees have many appealing properties
  - Similar to human decision process, easy to understand
  - Handle both **Discrete and Continuous** features
  - Highly flexible ***hypothesis space,*** as the # of nodes (or depth) of tree increases, decision tree can represent increasingly complex decision boundaries

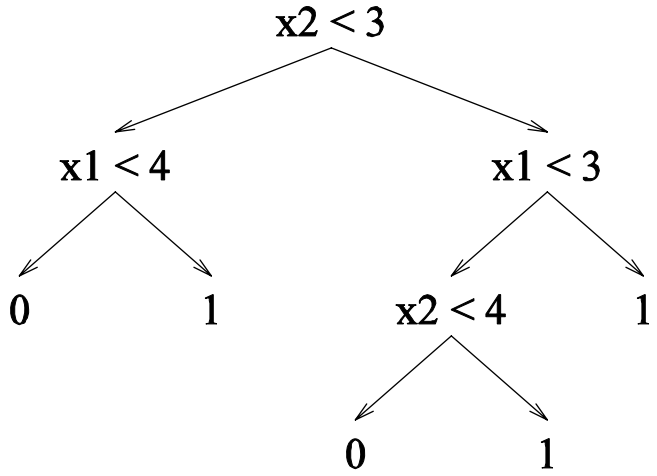
Definition: *Hypothesis space*

The space of solutions that a learning algorithm can possibly output. For example,

- For Perceptron: the hypothesis space is the space of all straight lines
- For nearest neighbor: the hypothesis space is infinitely complex
- For decision tree: it is a flexible space, as we increase the depth of the tree, the hypothesis space grows larger and larger

DT can represent arbitrarily complex decision boundaries





If needed, we can keep growing the tree until all examples are correctly classified, although it may not be the best idea.

So far we have looked at what is a decision tree, and what kind of decision boundaries decision trees produce, and its apealling properties. We now need to address:

How to learn decision trees

- Goal: Find a decision tree *h* that achieves **minimum misclassification errors on the training data**
- As our previous slides suggest, we can always achieve this by using large trees
- In fact, we can achieve this trivially: just create a decision tree with one path from root to leaf for each training example
  - Problem: Such a tree would just memorize the training data. It would not ***generalize to new data points*** *– i.e., capture regularities that are applicable to unseen data*
- Alternatively: find the <u>smallest</u> tree *h* that minimizes training error
  - Problem: This is NP-Hard

Greedy Learning For DT

There are different ways to construct trees from data. We will focus on the top-down, greedy search approach:

Instead of trying to optimize the whole tree together, we try to find one test at a time.

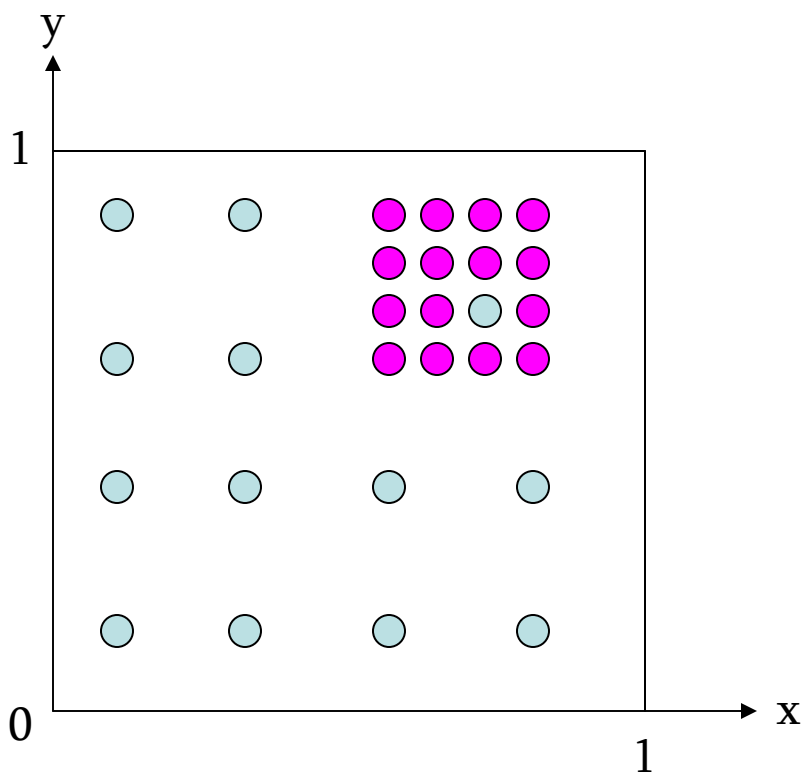
Basic idea: (assuming discrete features, relaxed later)

1. Choose the best attribute a* to place at the root of the tree.

2. Separate training set S into subsets $\{S_1, S_2, .., S_k\}$ where each subset S_i contains examples having the same value for a*

3. Recursively apply the algorithm on each new subset until examples have the same class or there are few of them.

Building DT: an example



13 ○ 15 ●

Training data contains

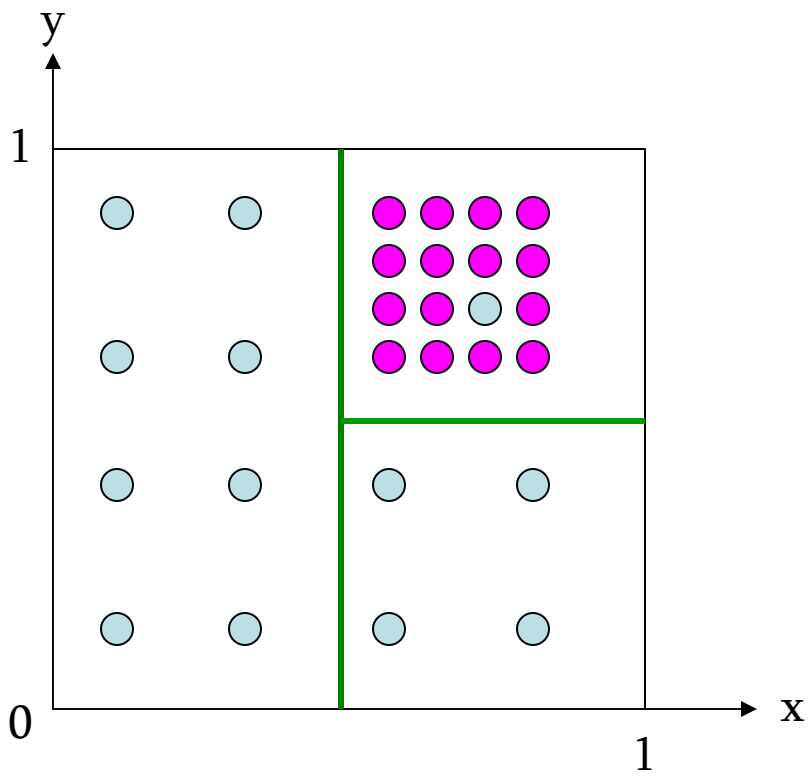
13 ○ 15 ●

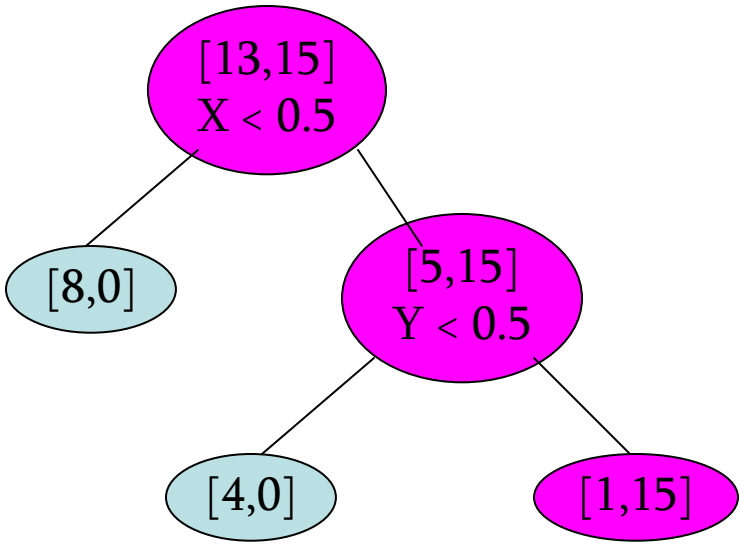
If we had to make a decision now, we'd pick ● . But there's too much uncertainty.

Based on training data, with probability 13/28 I would be wrong

Now if you are allowed to ask one question about your example to help the decision, which question will you ask?

One possible question: is x < 0.5?





Now we feel much better because the uncertainty in each leaf node is much reduced!

Building a decision tree

1. Choose the best attribute a* to place at the root of the tree.

   What do we mean by “best” – reduce the most uncertainty about our decision of the class labels

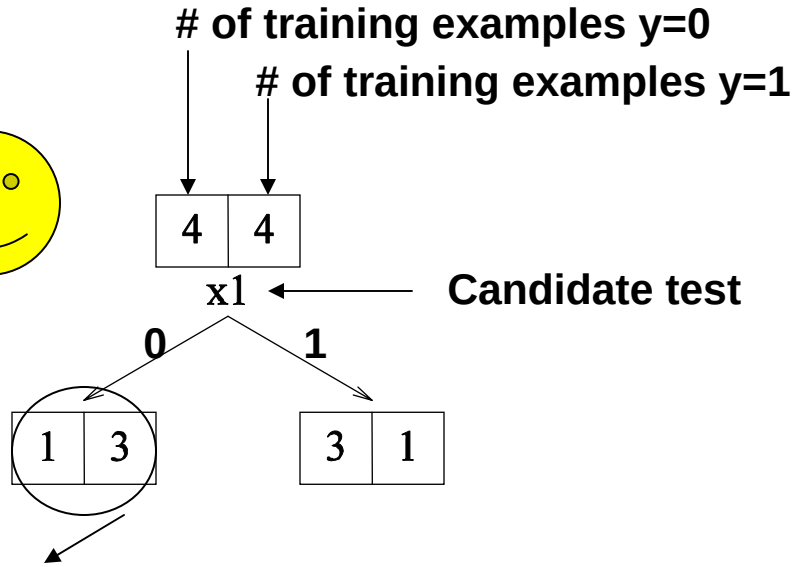
2. Separate the training set S into subsets $\{S_1, S_2, .., S_k\}$ where each subset S_i contains examples having the same value for a*

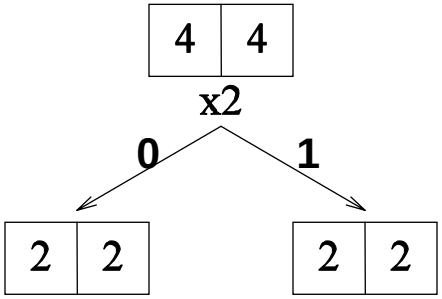
3. Recursively apply the algorithm on each new subset until all examples have the same class label

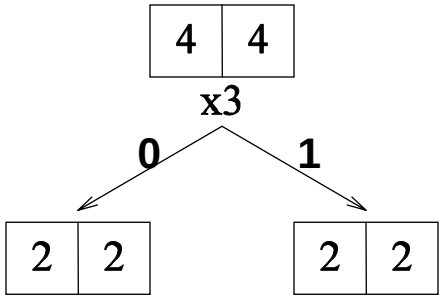
Choosing split: example

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







# of training mistakes can be used as a measure of uncertainty