Decision Trees

Learning Decision Trees

Decision trees provide a very popular and efficient hypothesis space.

- **Variable Size**. Any boolean function can be represented.

- **Deterministic**.

- **Discrete and Continuous Parameters**.

Learning algorithms for decision trees can be described as

- **Constructive Search**. The tree is built by adding nodes.

- **Eager**.

- **Batch** (although online algorithms do exist).

Decision Tree Hypothesis Space

- **Internal nodes** test the value of particular features x_j and branch according to the results of the test.

- **Leaf nodes** specify the class $h(\mathbf{x})$.
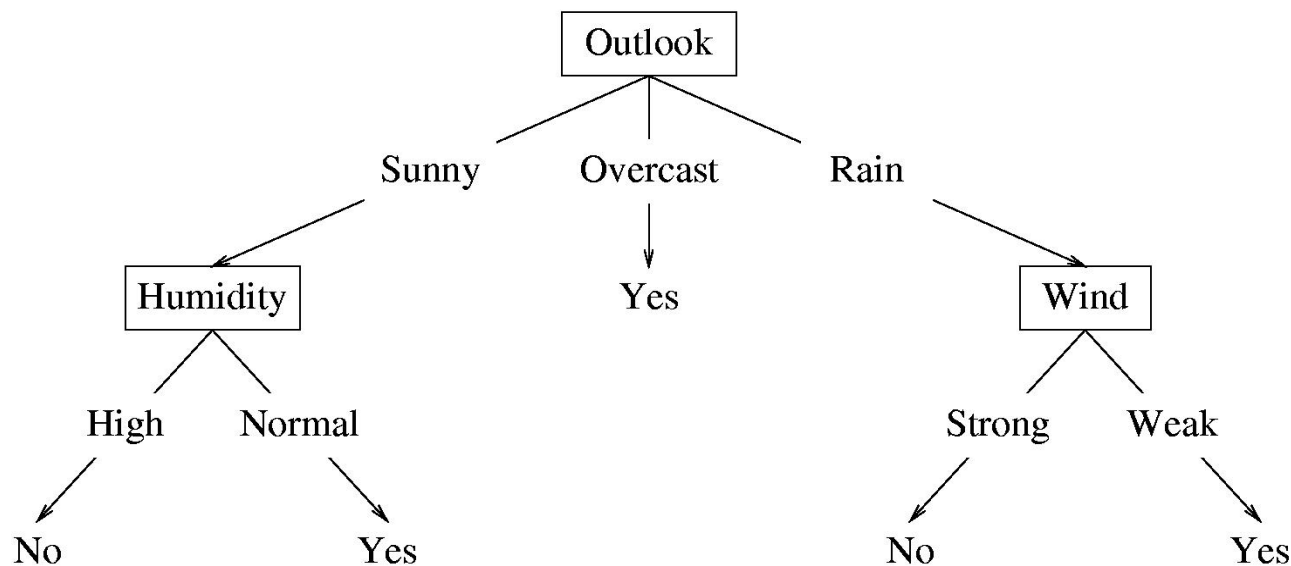
Suppose the features are **Outlook** (x_1), **Temperature** (x_2), **Humidity** (x_3), and **Wind** (x_4). Then the feature vector $\mathbf{x} = (Sunny, Hot, High, Strong)$ will be classified as **No**. The **Temperature** feature is irrelevant.
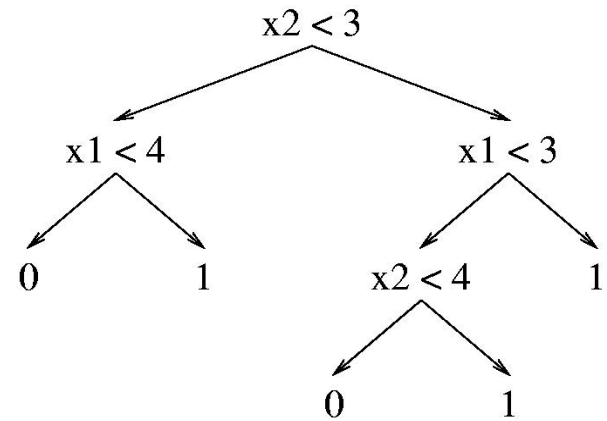
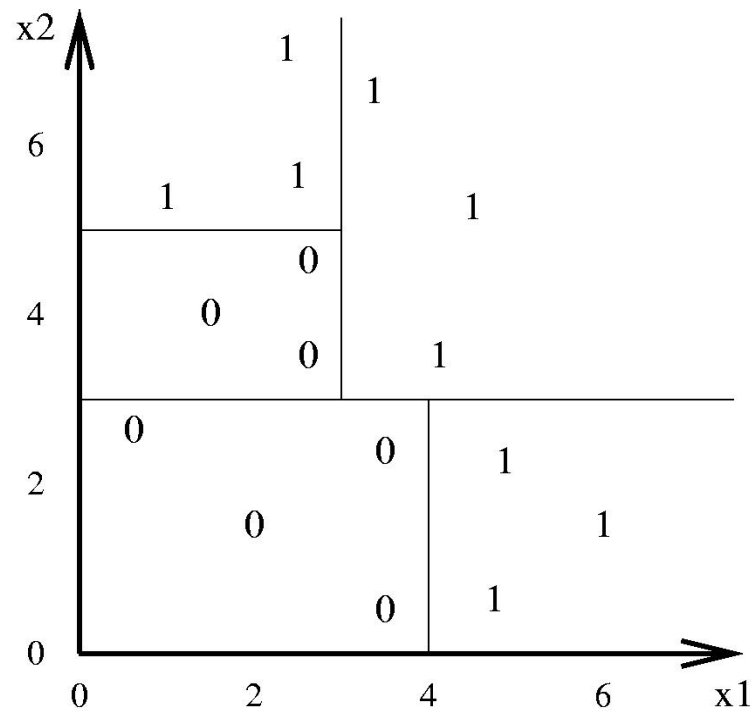# Decision Tree Hypothesis Space

If the features are continuous, internal nodes may test the value of a feature against a threshold.

Outlook

Sunny   Overcast   Rain

Humidity   Yes   Wind

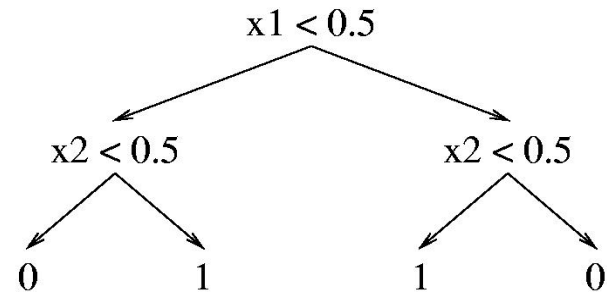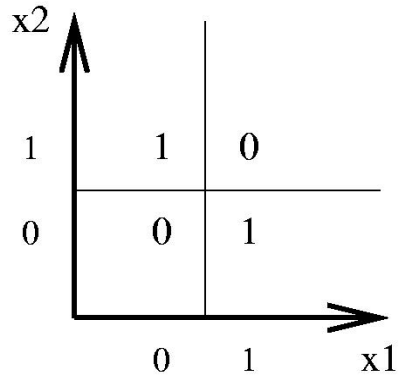> 75%   <= 75%   > 20   <= 20

No   Yes   No   Yes
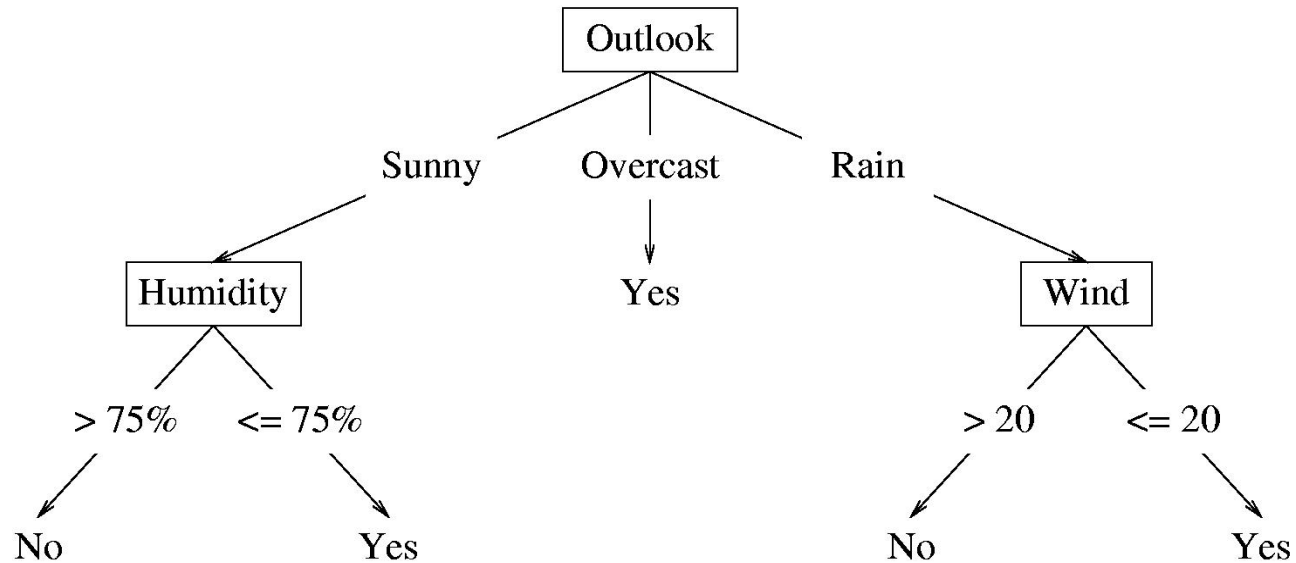
Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the K classes.

Decision Trees Can Represent Any Boolean Function

The tree will in the worst case require exponentially many nodes, however.

Decision Trees Provide Variable-Size Hypothesis Space

As the number of nodes (or depth) of tree increases, the hypothesis space grows

- **depth 1** ("decision stump") can represent any boolean function of one feature.

- **depth 2** Any boolean function of two features; some boolean functions involving three features (e.g., $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_3)$

- **etc.**

Learning Algorithm for Decision Trees

The same basic learning algorithm has been discovered by many people independently:

GROWTREE(S)

if ($y = 0$ for all $\langle \mathbf{x}, y \rangle \in S$) **return** new leaf(0)

else if ($y = 1$ for all $\langle \mathbf{x}, y \rangle \in S$) **return** new leaf(1)

else

    choose best attribute x_j

    $S_0 = $ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 0$;

    $S_1 = $ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 1$;

    return new node(x_j, GROWTREE(S_0), GROWTREE(S_1))

Choosing the Best Attribute

One way to choose the best attribute is to perform a 1-step lookahead search and choose the attribute that gives the lowest error rate on the training data.

CHOOSEBESTATTRIBUTE(S)

choose j to minimize J_j, computed as follows:

$S_0 = $ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 0$;

$S_1 = $ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 1$;

$y_0 = $ the most common value of y in S_0

$y_1 = $ the most common value of y in S_1

$J_0 = $ number of examples $\langle \mathbf{x}, y \rangle \in S_0$ with $y \neq y_0$

$J_1 = $ number of examples $\langle \mathbf{x}, y \rangle \in S_1$ with $y \neq y_1$

$J_j = J_0 + J_1$ (total errors if we split on this feature)

return j

A Better Heuristic From Information Theory

Let V be a random variable with the following probability distribution:

$P(V = 0)$	$P(V = 1)$
0.2	0.8

The *surprise*, $S(V = v)$ of each value of V is defined to be

$$S(V = v) = -\lg P(V = v).$$

An event with probability 1 gives us zero surprise.

An event with probability 0 gives us infinite surprise!

It turns out that the surprise is equal to the number of bits of information that need to be transmitted to a recipient who knows the probabilities of the results.

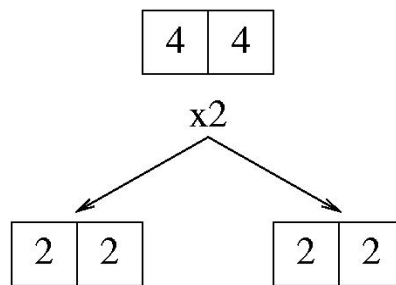This is also called the **description length** of $V = v$.

Fractional bits only make sense if they are part of a longer message (e.g., describe a whole sequence of coin tosses).
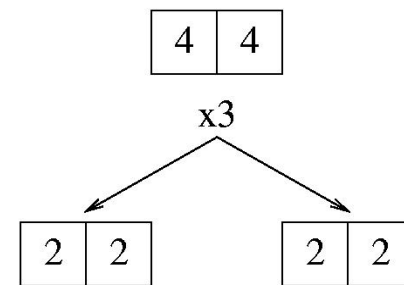
Choosing the Best Attribute—An Example

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

| 4 | 4 |

x1

| 1 | 3 | | 3 | 1 |

J=2

| 4 | 4 |

x2

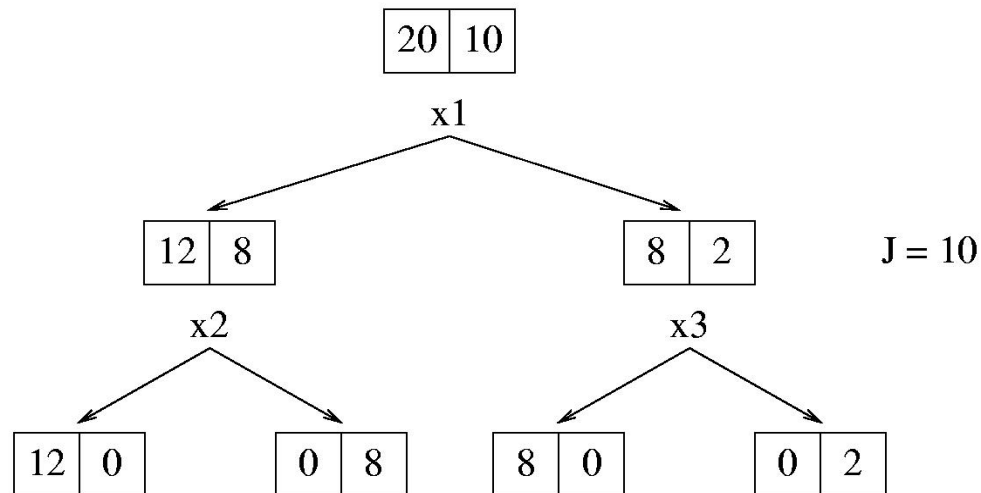| 2 | 2 | | 2 | 2 |

J=4

| 4 | 4 |

x3

| 2 | 2 | | 2 | 2 |
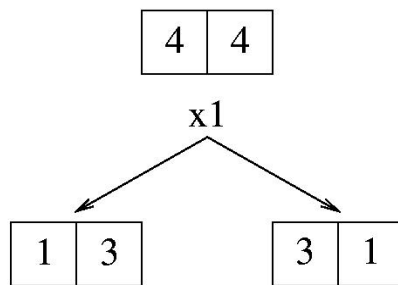
J=4

Choosing the Best Attribute (3)

Unfortunately, this measure does not always work well, because it does not detect cases where we are making "progress" toward a good tree.
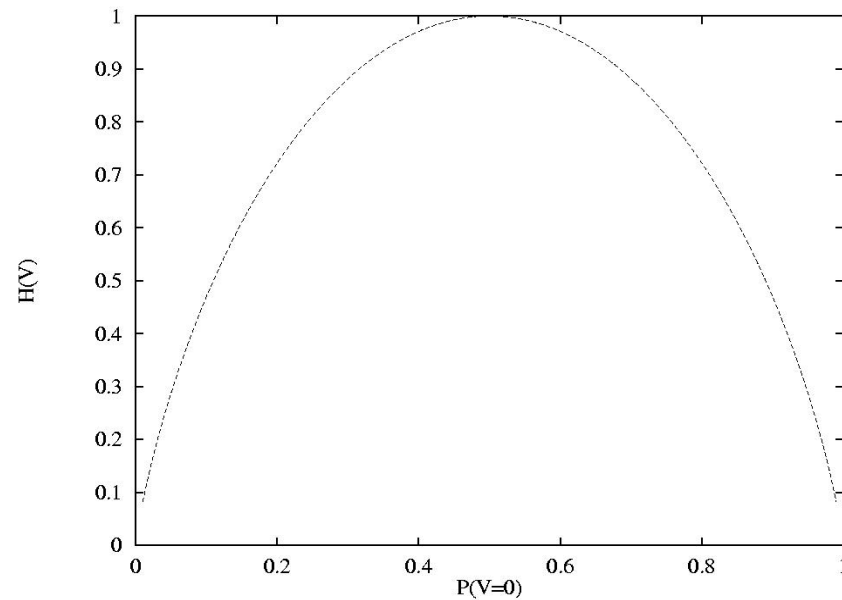
```
                    ┌────┬────┐
                    │ 20 │ 10 │
                    └────┴────┘
                        x1
             ┌──────────────────────┐
        ┌────┬────┐             ┌───┬───┐
        │ 12 │ 8  │             │ 8 │ 2 │        J = 10
        └────┴────┘             └───┴───┘
            x2                      x3
        ┌───────┐               ┌───────┐
 ┌────┬───┐  ┌───┬───┐    ┌───┬───┐  ┌───┬───┐
 │ 12 │ 0 │  │ 0 │ 8 │    │ 8 │ 0 │  │ 0 │ 2 │
 └────┴───┘  └───┴───┘    └───┴───┘  └───┴───┘
```

Entropy

The *entropy* of V, denoted $H(V)$ is defined as follows:

$$H(V) = \sum_{v=0}^{1} -P(H=v) \lg_2 P(H=v).$$

This is the average surprise of describing the result of one "trial" of V (one coin toss).
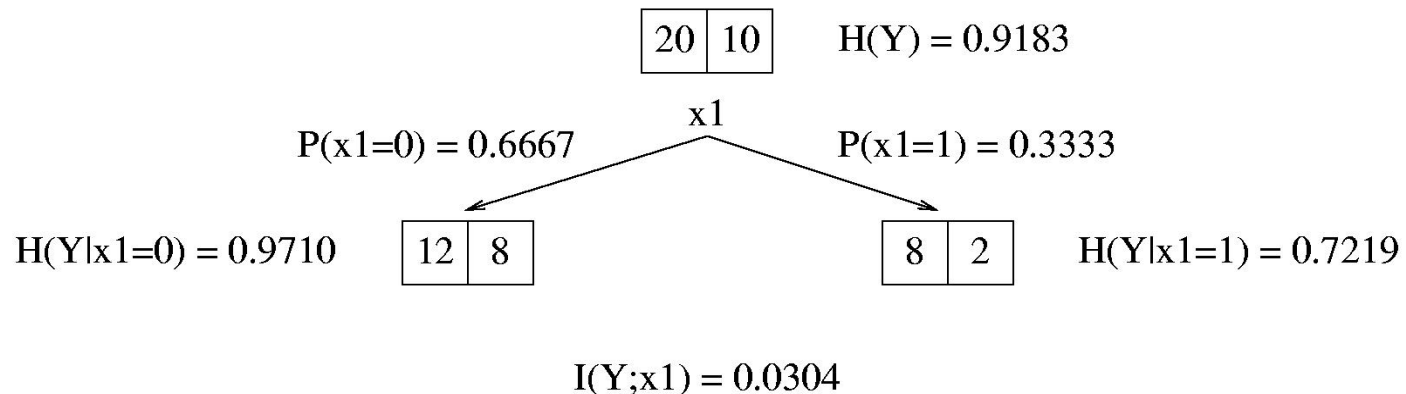

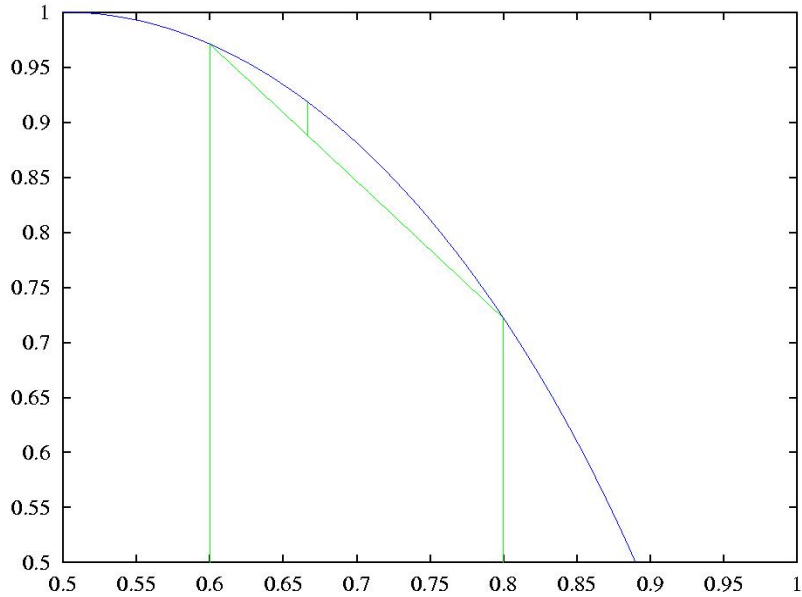Entropy can be viewed as a measure of uncertainty.

Mutual Information

Now consider two random variables A and B that are not necessarily independent. The *mutual information* between A and B is the amount of information we learn about B by knowning the value of A (and vice versa—it is symmetric). It is computed as follows:

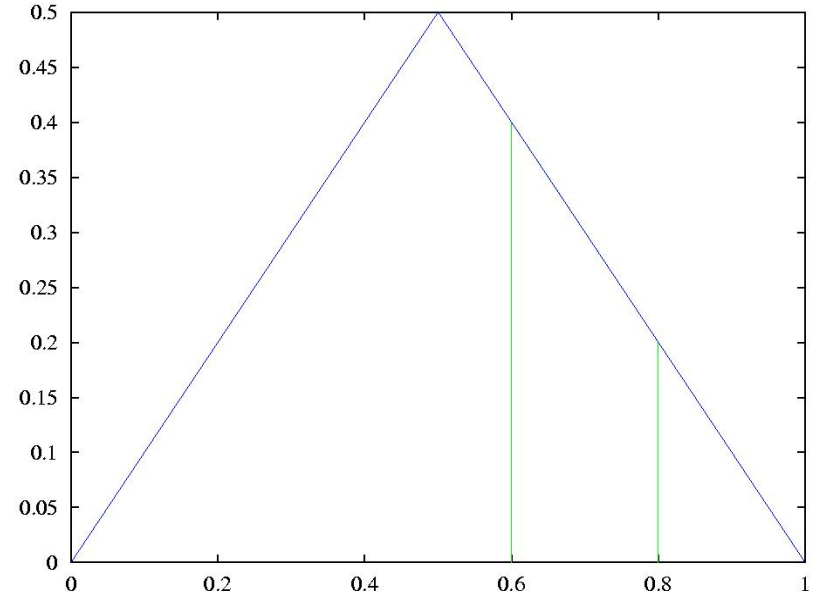$$I(A; B) = H(B) - \sum_b P(B = b) \cdot H(A|B = b)$$

In particular, consider the class Y of each training example and the value of feature x_1 to be random variables. Then the mutual information quantifies how much x_1 tells us about the value of the class Y.

| 20 | 10 | H(Y) = 0.9183

x1

P(x1=0) = 0.6667   P(x1=1) = 0.3333

H(Y|x1=0) = 0.9710 | 12 | 8 | | 8 | 2 | H(Y|x1=1) = 0.7219

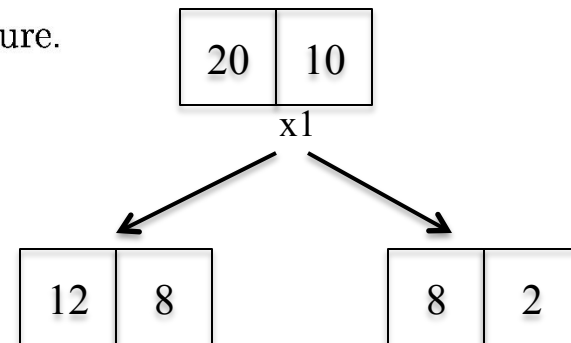I(Y;x1) = 0.0304

Visualizing Heuristics

Entropy

Absolute Error

Mutual information works because it is a convex measure.

Non-Boolean Features

- **Features with multiple discrete values**

  Construct a multiway split?

  Test for one value versus all of the others?

 Group the values into two disjoint subsets?

- **Real-valued features**

  Consider a threshold split using each observed value of the feature.

Whichever method is used, the mutual information can be computed to choose the best split.

Learning Parity with Noise

When learning exclusive-or (2-bit parity), all splits look equally good. If extra random boolean features are included, they also look equally good. Hence, decision tree algorithms cannot distinguish random noisy features from parity features.

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

4	4

x1

2	2

2	2

J=4

4	4

x2

2	2

2	2

J=4

4	4

x3

2	2

2	2

J=4
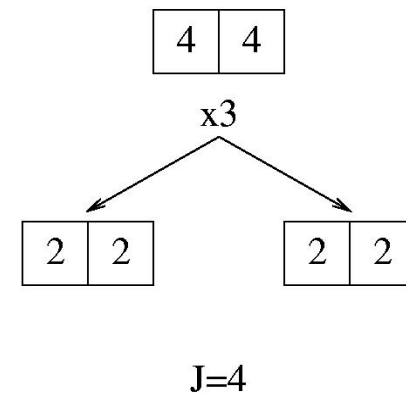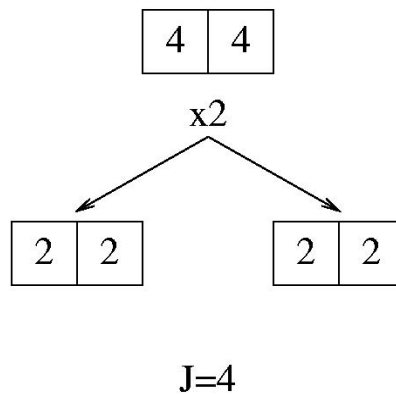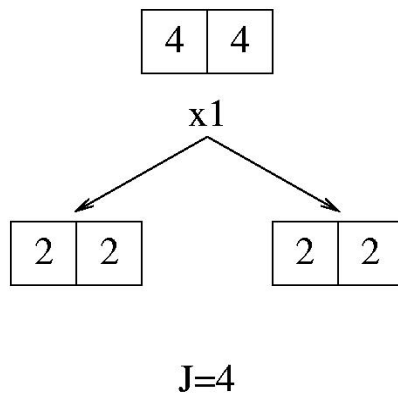
Overfitting in Decision Trees
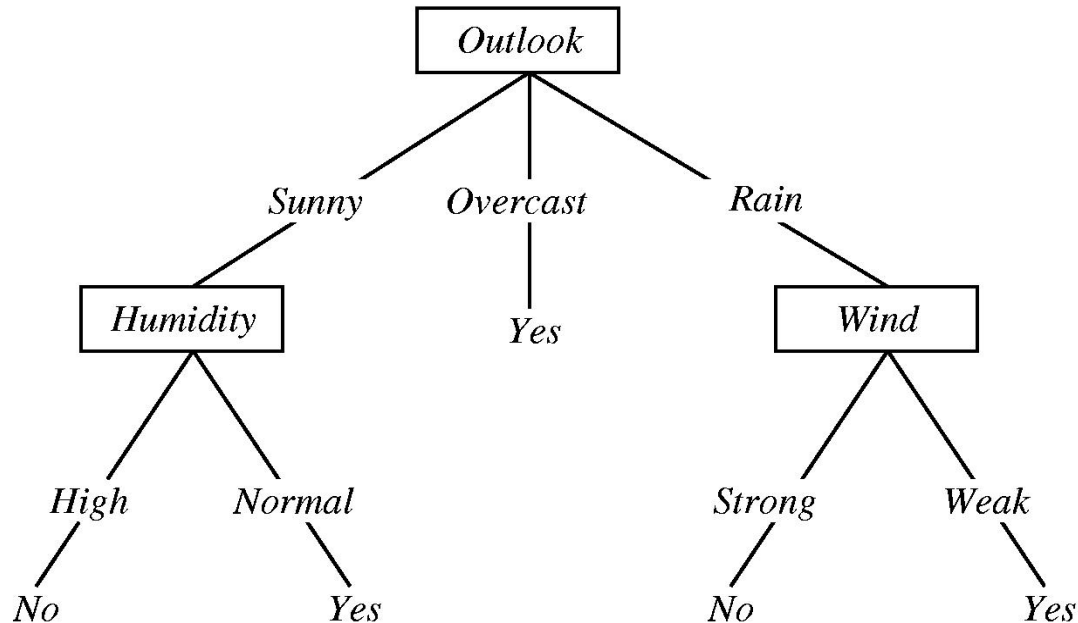
Consider adding a noisy training example:

Sunny, Hot, Normal, Strong, PlayTennis=No

What effect on tree?

Overfitting

Consider error of hypothesis h over

- training data: $error_{train}(h)$

- entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$
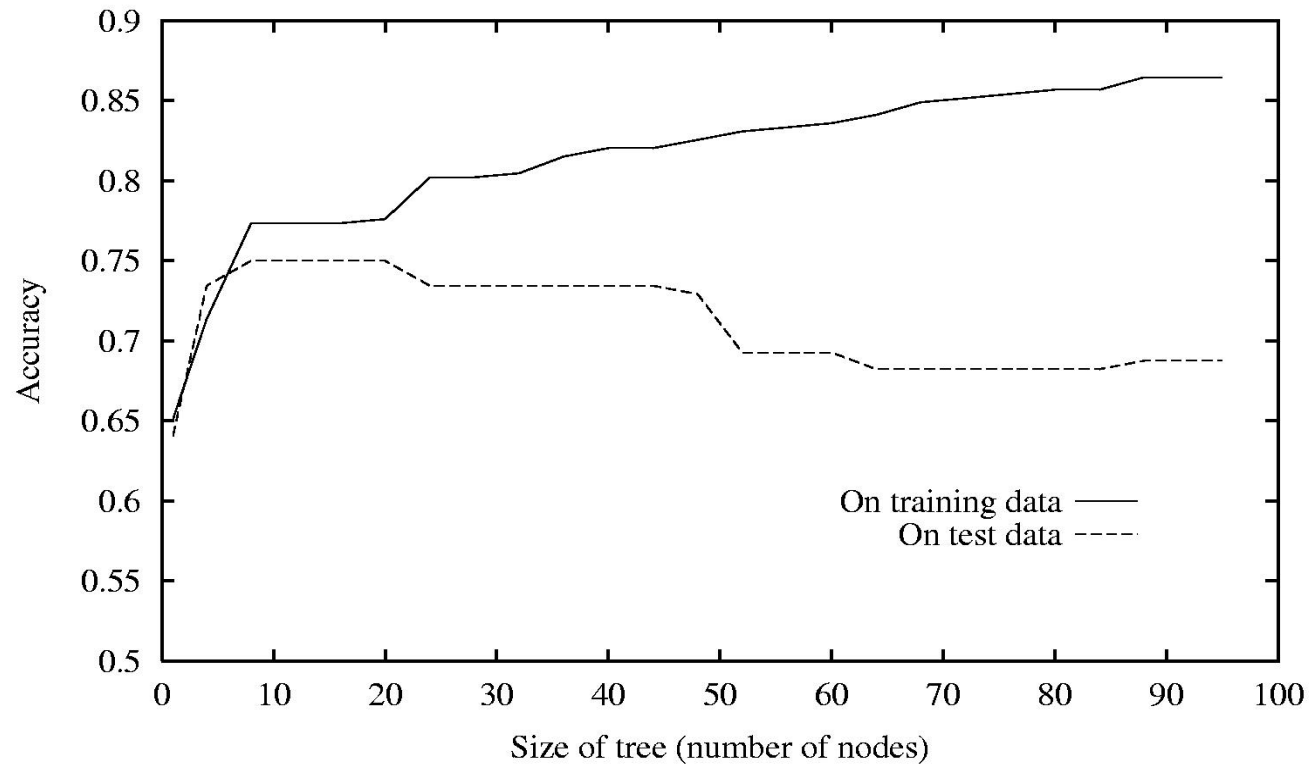
Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

Overfitting in Decision Tree Learning

Avoiding Overfitting

How can we avoid overfitting?

- Stop growing when data split not statistically significant

- Grow full tree, then post-prune

How to select "best" tree:

- Measure performance over training data

- Measure performance over separate validation data set

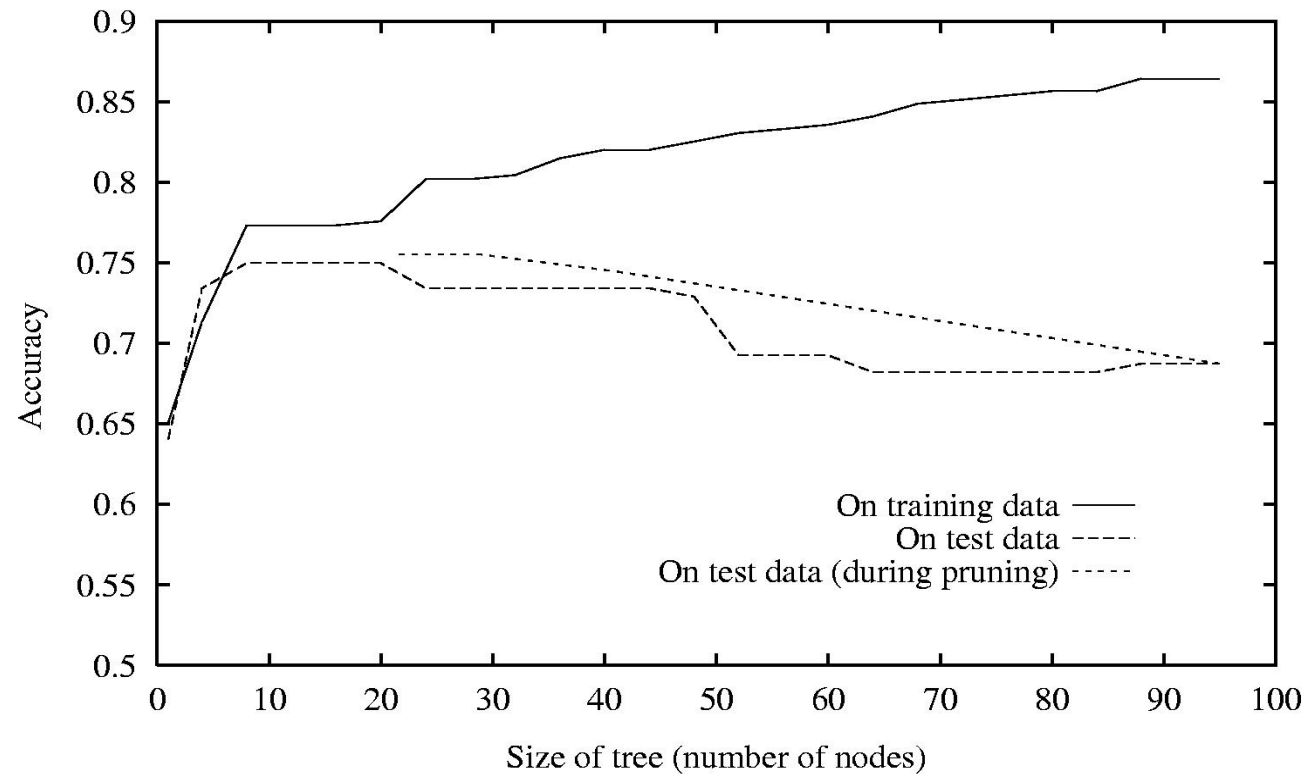- Add complexity penalty to performance measure

Reduced-Error Pruning

Split data into *training* and *validation* set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)

2. Greedily remove the one that most improves *validation* set accuracy

Effect of Reduced-Error Pruning

Summary: Decision Trees

- Representation
- Tree growth
- Heuristics
- Overfitting and pruning
- Scaling up