

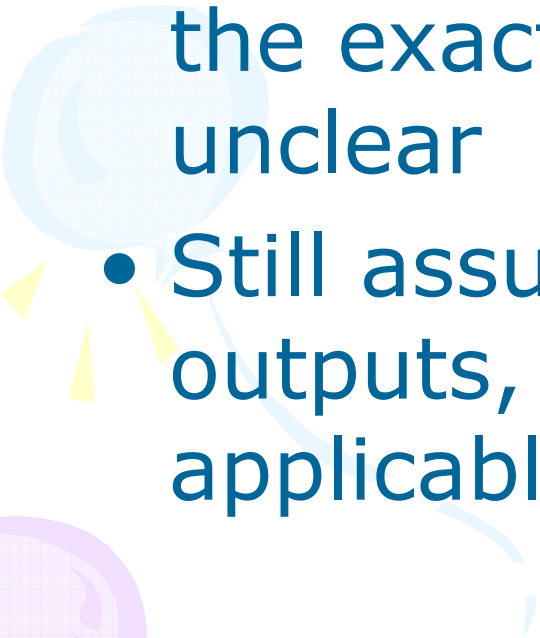
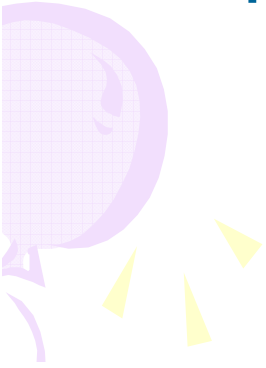


Principles of Artificial Intelligence(305450)

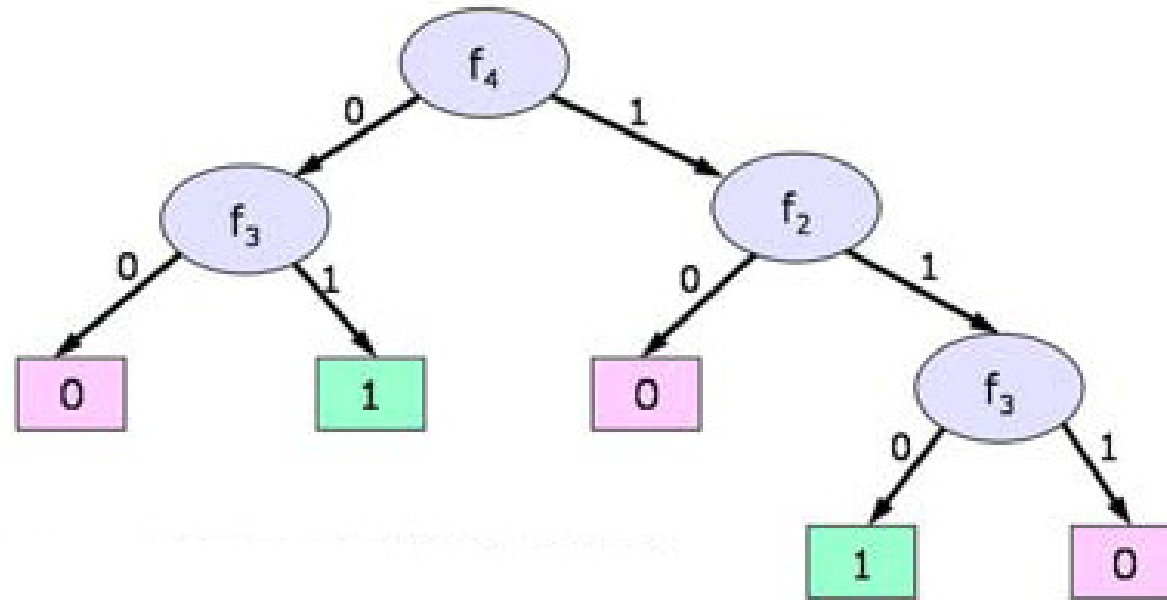
Lecture 9: Machine Learning I



Decision Trees

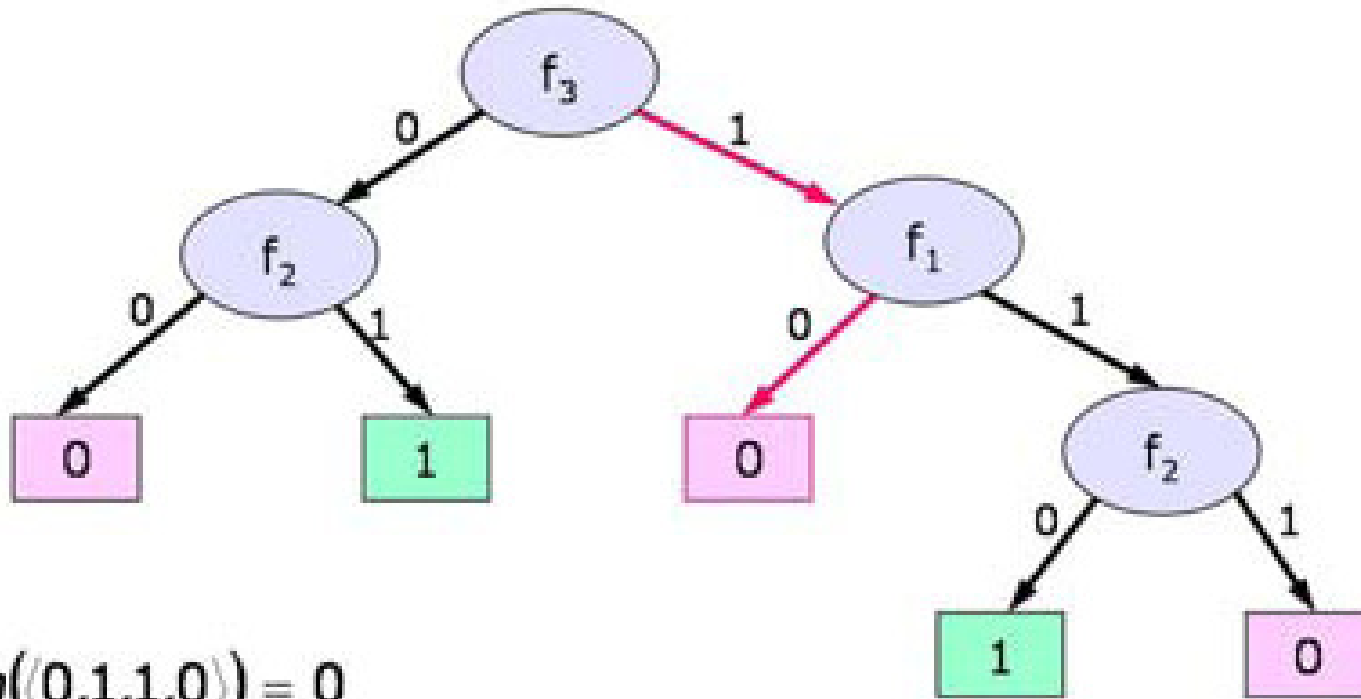
- DNF learning algorithm is a bit cumbersome and inefficient. Also, the exact effect of the heuristic is unclear
 - Still assume binary inputs and outputs, but much more broadly applicable
- 
- 

Hypothesis Class



- Internal nodes: feature name
- One child for each value of the feature (0, 1 here)
- Leaf nodes: output (0, 1 here since y 's are boolean)

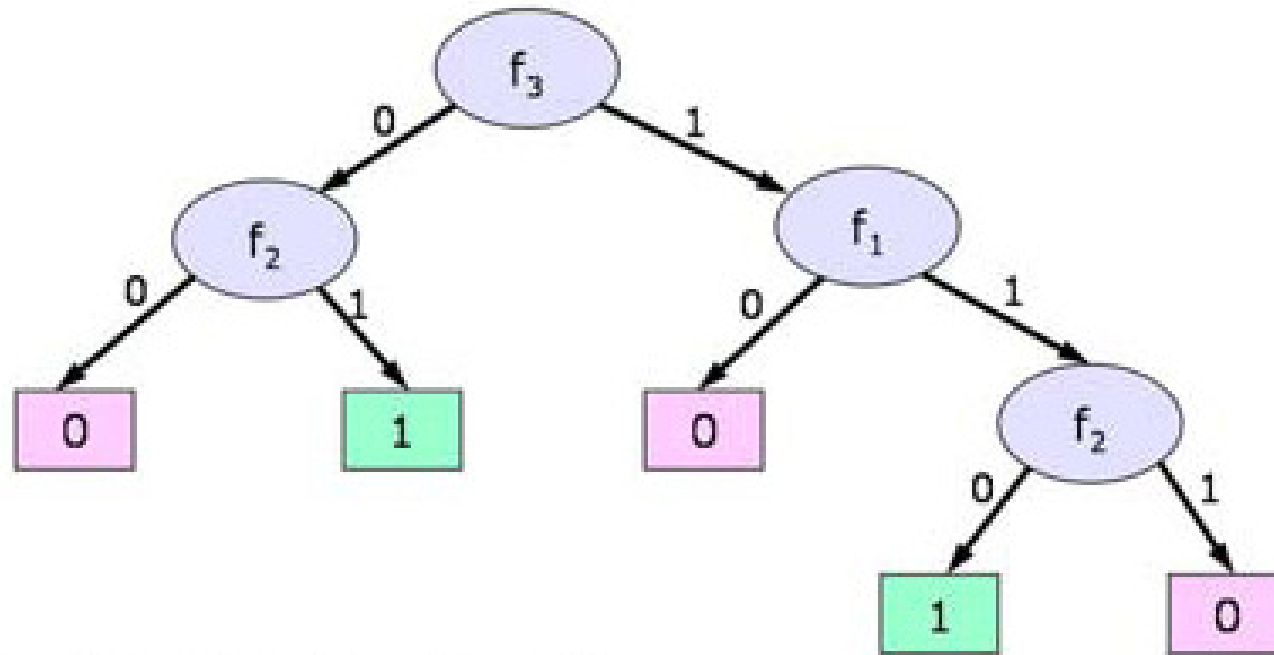
Hypothesis Class



$$h(\langle 0, 1, 1, 0 \rangle) = 0$$

- Trees represent Boolean functions from x 's (vectors of feature values) to Booleans.
- In this example, input $[0 \ 1 \ 1 \ 0]$ would generate an output of 0

Hypothesis Class

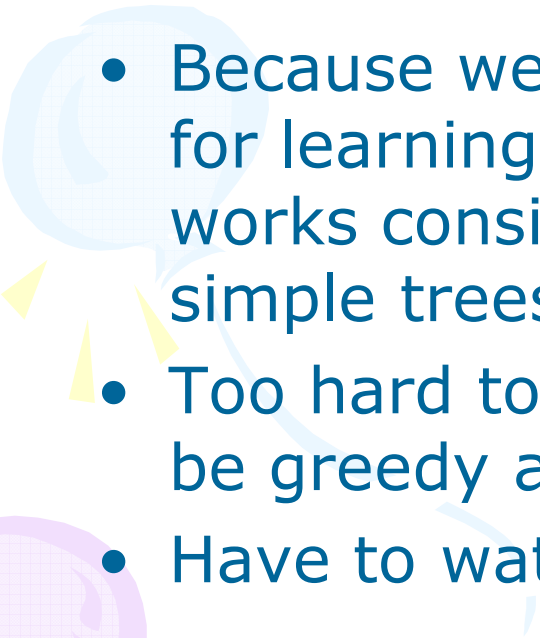



$$h = (\neg f_3 \wedge f_2) \vee (f_3 \wedge f_1 \wedge \neg f_2)$$

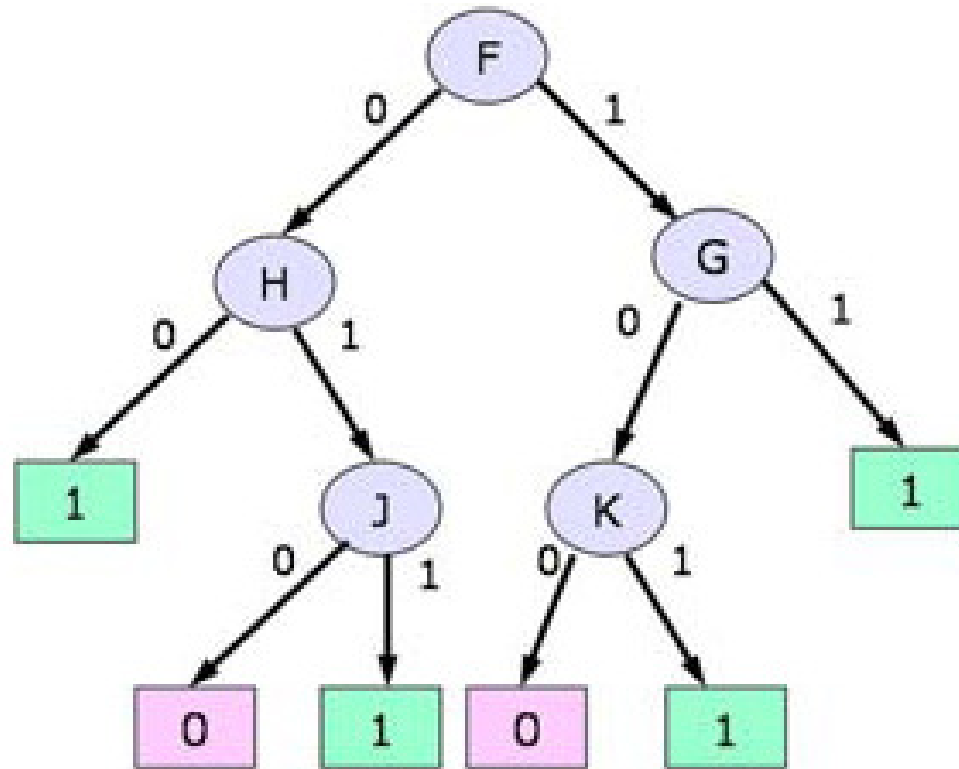
- Decision trees are a way to writing down Boolean expressions



Tree Bias

- Both decision trees and DNF with negation can represent any Boolean function. So why bother with trees?
 - Because we have a nice algorithm that is tailored for learning in the tree representation, which works consistently for both growing trees and simple trees (few nodes)
 - Too hard to find the smallest good tree, so we'll be greedy again
 - Have to watch out for overfitting
- 
- 

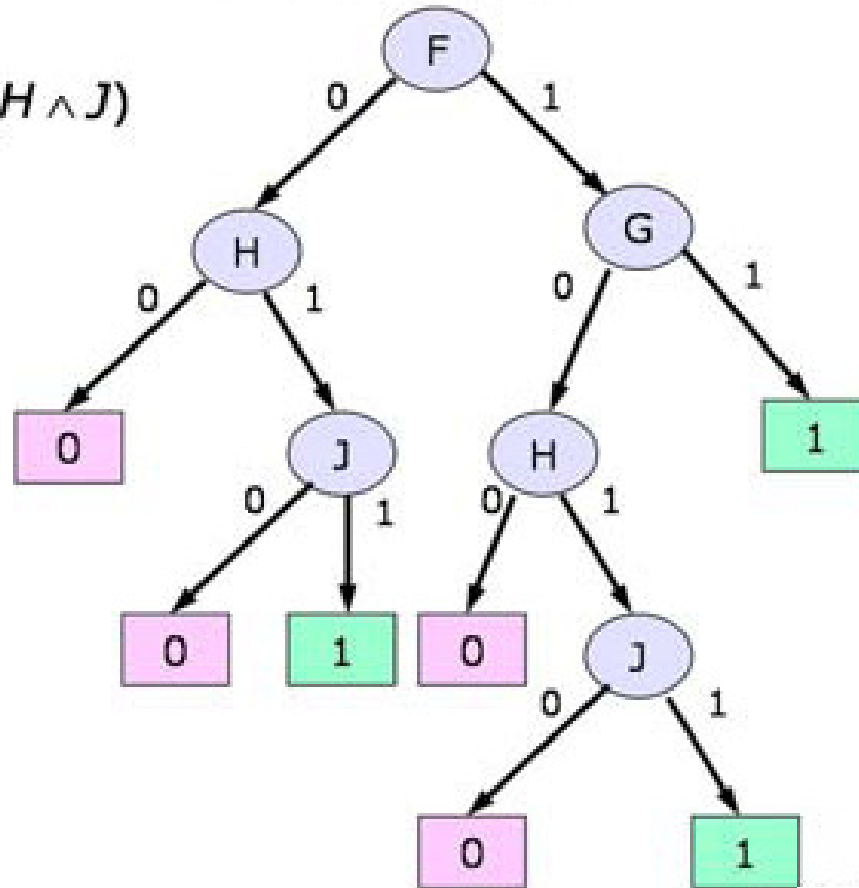
Tree vs DNF



$$(\neg F \wedge \neg H) \vee (\neg F \wedge H \wedge J) \vee (F \wedge \neg G \wedge K) \vee (F \wedge G)$$

Tree vs DNF

$$(F \wedge G) \vee (H \wedge J)$$





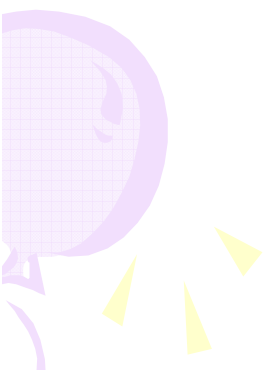
Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone



BuildTree (Data)

We will build the tree from the top down.
Here is pseudo code for the algorithm.
It will take as input a data set, and return a tree.





Algorithm


- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone



```
BuildTree (Data)
```

```
    if all elements of Data have the same y value, then
```

```
        MakeLeafNode (y)
```



We first test to see if all the data elements have the same y values. If so, we simply make a leaf node with that y value and we're done. This is the base case of our recursive algorithm.



Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone



```
BuildTree (Data)
```

```
  if all elements of Data have the same y value, then
```

```
    MakeLeafNode(y)
```

```
  else
```

```
    feature := PickBestFeature(Data)
```

```
    MakeInternalNode(feature,
```

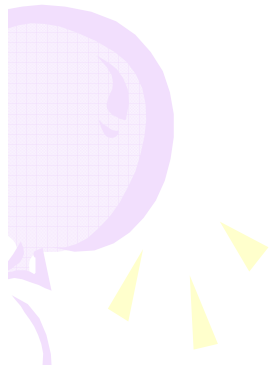
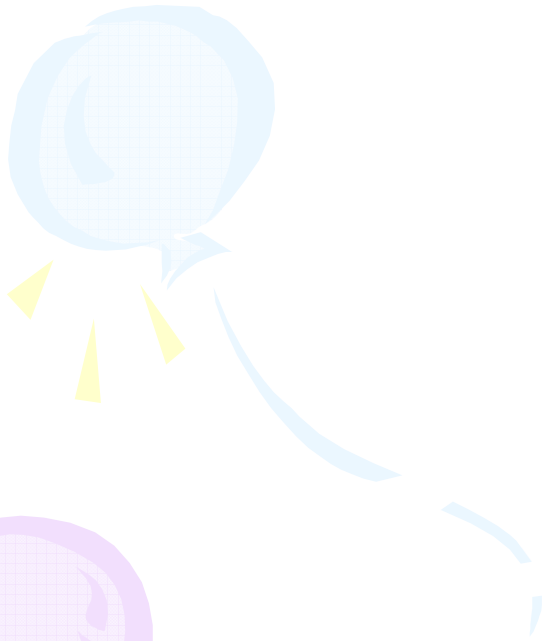
```
      BuildTree(SelectFalse(Data, feature)),
```

```
      BuildTree(SelectTrue(Data, feature)))
```



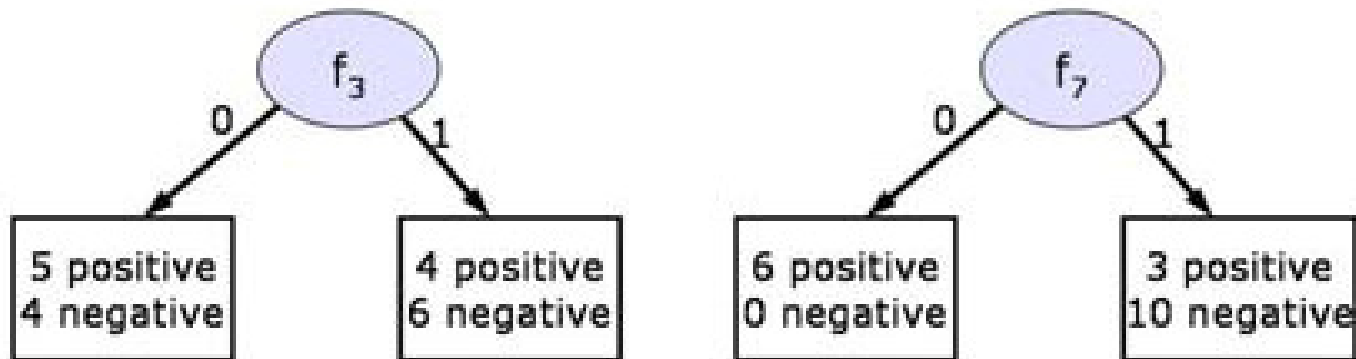
Let's split

D: 9 positive
10 negative



Let's split

D: 9 positive
10 negative



Our goal, in building trees, is to separate the negative instances from the positive instances with the fewest possible tests. In this example, splitting f_7 seems to be more helpful



Entropy

- p : proportion of positive examples in a data set


$$H = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

To formalize that intuition, we need to develop a measure of the degree of uniformity of the subsets of the data we'd get by splitting on a feature

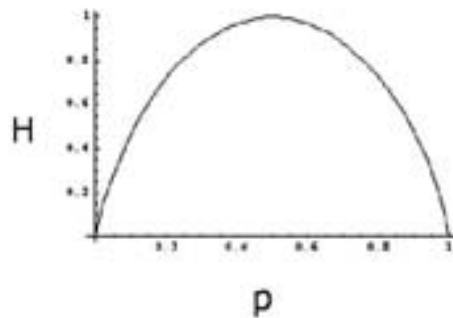


Let's start by looking at a standard measure of disorder, used in physics and information theory, called **entropy**

Entropy

- p : proportion of positive examples in a data set

$$H = -p \log_2 p - (1 - p) \log_2 (1 - p)$$



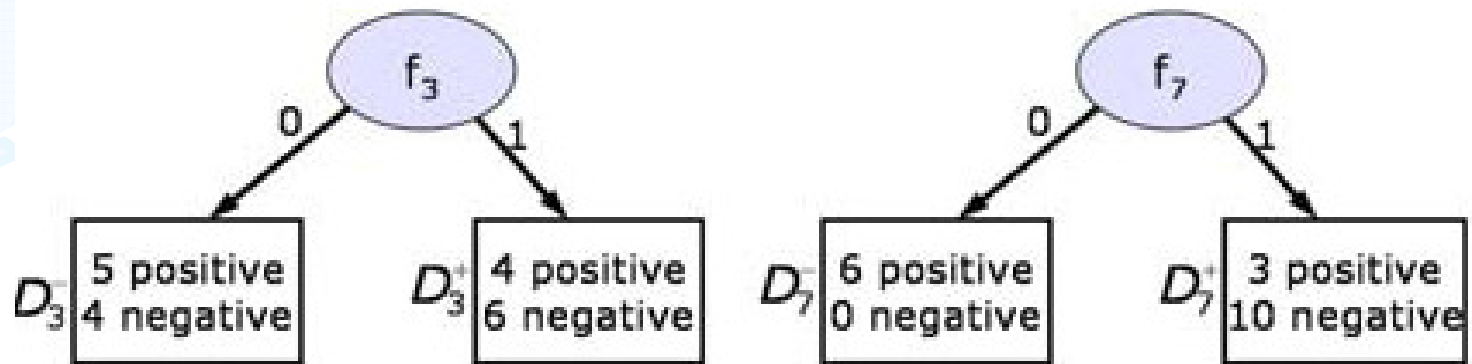
$$0 \log_2 0 = 0$$

$$\log 1 = 0$$

when $p = 0$, $\log 0$ is negative infinity, But 0 wins the battle, so $0 \log 0$ is also 0

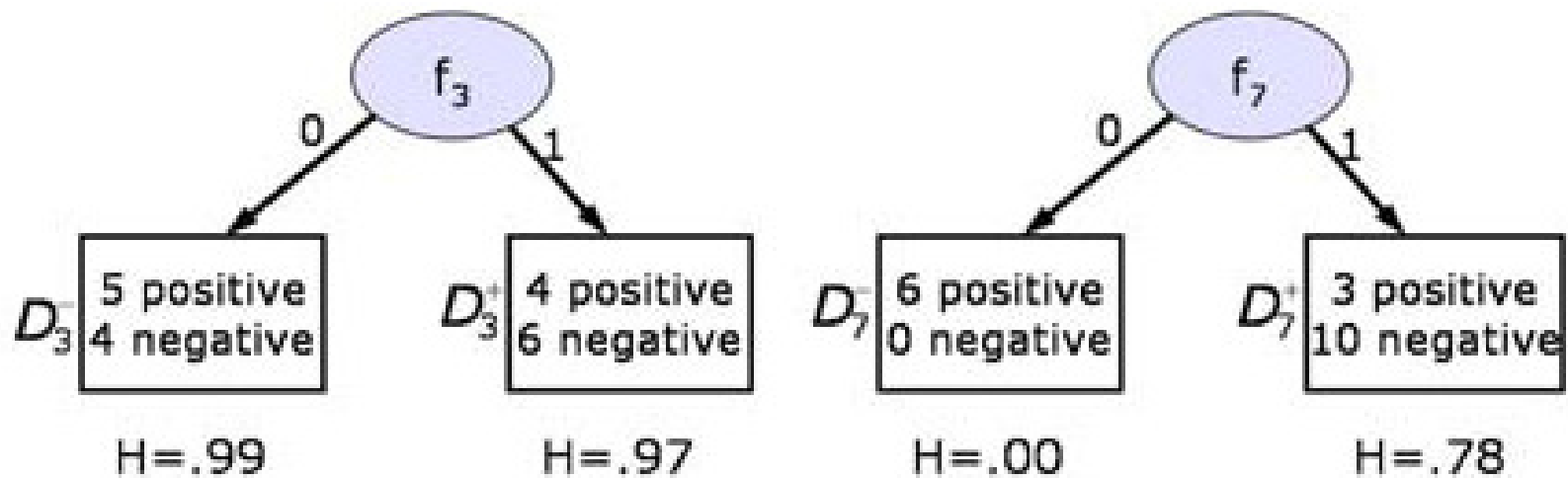
Let's split

D: 9 positive
10 negative

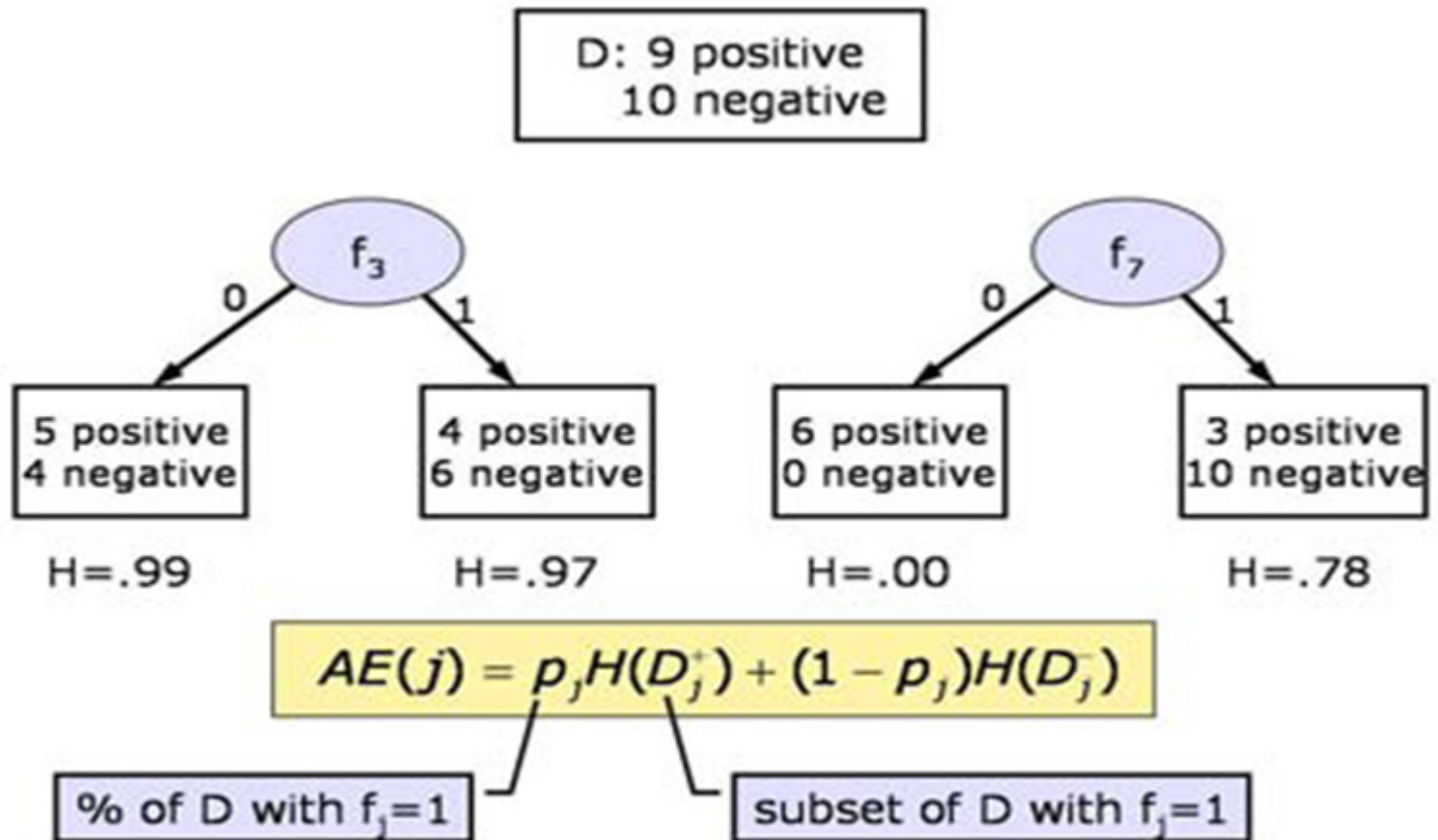


Let's split

D: 9 positive
10 negative

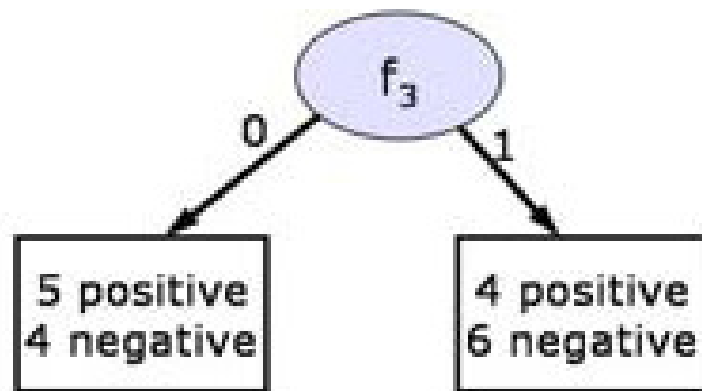


Let's split



Let's split

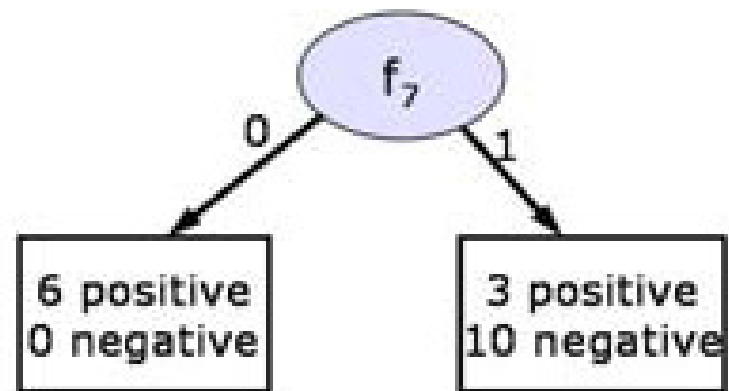
D: 9 positive
10 negative



$H = .99$

$H = .97$

$$AE = (9/19) \cdot .99 + (10/19) \cdot .97 = .98$$



$H = .00$

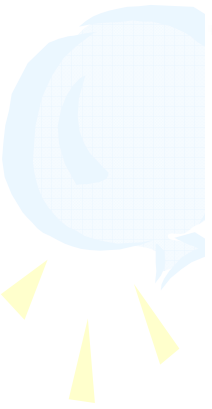
$H = .78$

$$AE = (6/19) \cdot 0 + (13/19) \cdot .78 = .53$$



Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone



```
BuildTree (Data)
  if all elements of Data have the same y value, then
    MakeLeafNode(y)
  else
    feature := PickBestFeature(Data)
    MakeInternalNode(feature,
      BuildTree(SelectFalse(Data, feature)),
      BuildTree(SelectTrue(Data, feature)))
```

- 
- Best feature minimizes average entropy of data in the children

Stopping

- Stop recursion if data contains only multiple instances of the same x with different y values

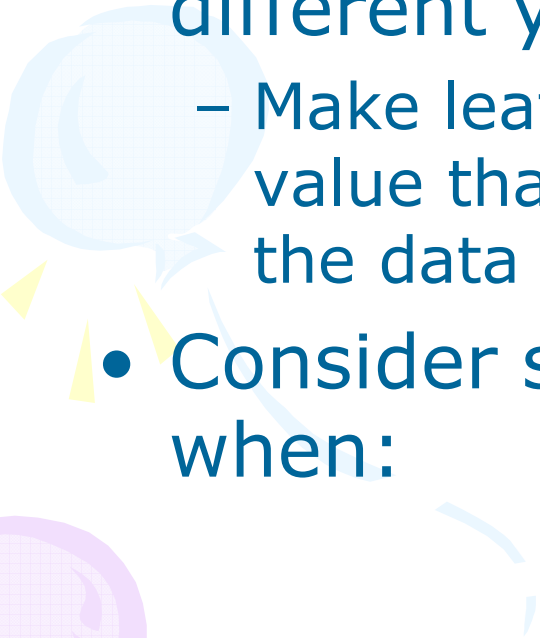
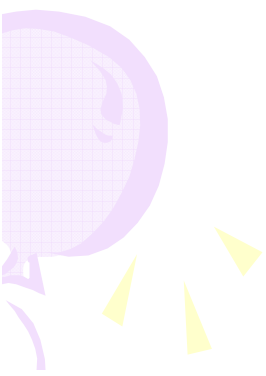


Stopping

- Stop recursion if data contains only multiple instances of the same x with different y values
 - Make leaf node with output equals to the y value that occurs in the majority of the class in the data



Stopping

- Stop recursion if data contains only multiple instances of the same x with different y values
 - Make leaf node with output equals to the y value that occurs in the majority of the class in the data
 - Consider stopping to avoid overfitting when:
- 
- 



Stopping

- Stop recursion if data contains only multiple instances of the same x with different y values
 - Make leaf node with output equals to the y value that occurs in the majority of the class in the data
- Consider stopping to avoid overfitting when:
 - entropy of a data set is below some threshold



Stopping

- Stop recursion if data contains only multiple instances of the same x with different y values
 - Make leaf node with output equals to the y value that occurs in the majority of the class in the data
- Consider stopping to avoid overfitting when:
 - entropy of a data set is below some threshold
 - number of elements in a data set is below threshold



Stopping

- Stop recursion if data contains only multiple instances of the same x with different y values
 - Make leaf node with output equals to the y value that occurs in the majority of the class in the data
- Consider stopping to avoid overfitting when:
 - entropy of a data set is below some threshold
 - number of elements in a data set is below threshold
 - best next split doesn't decrease average entropy

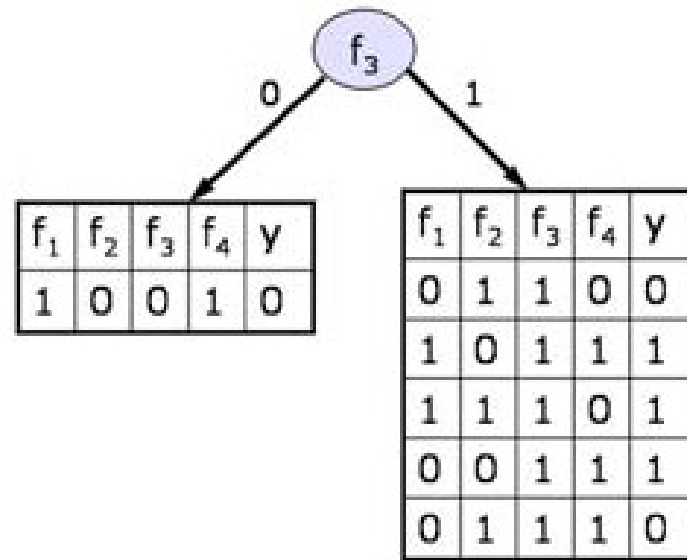
Simulation

- $H(D) = .92$
- $AE_1 = .92, AE_2 = .92, AE_3 = .81, AE_4 = 1$

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	0

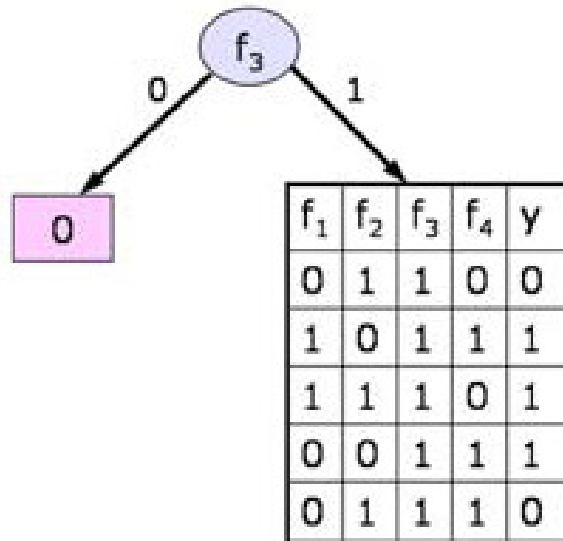
- Let's see how our tree-learning algorithm behaves on the example we used to demonstrate the DNF learning algorithm.
- Our data set has a starting entropy of .92.
- Then we can compute, for each feature, what the average entropy of the children would be if we were to split on that feature
- In this case, the best feature to split is f_3 .

Simulation

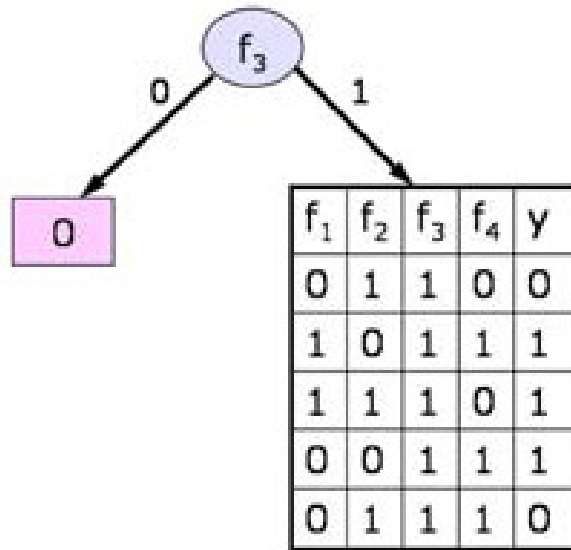


- The one on the left has only a single output (in fact, only a single data point)

Simulation



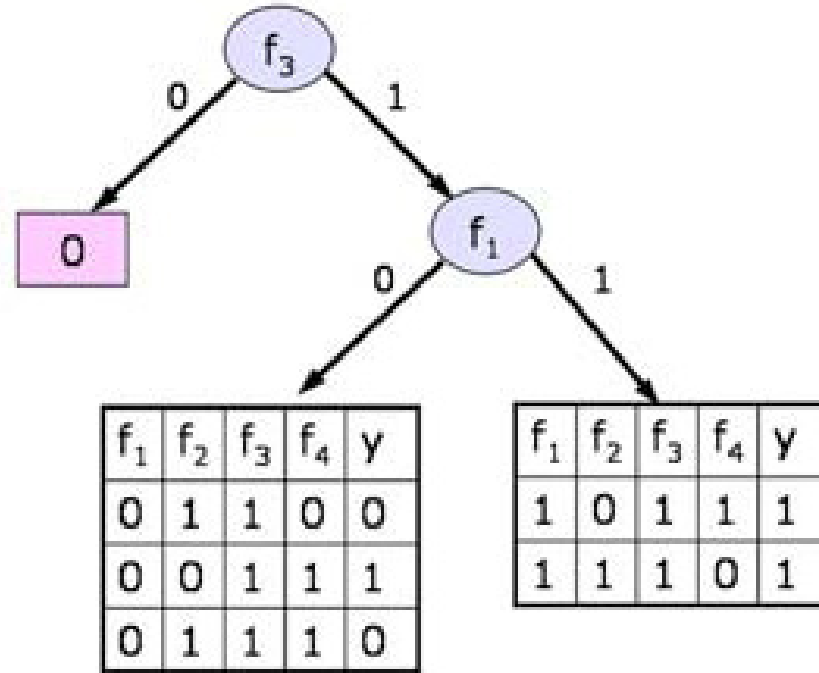
Simulation



- $AE_1 = .55$,
 $AE_2 = .55$, $AE_4 = .95$

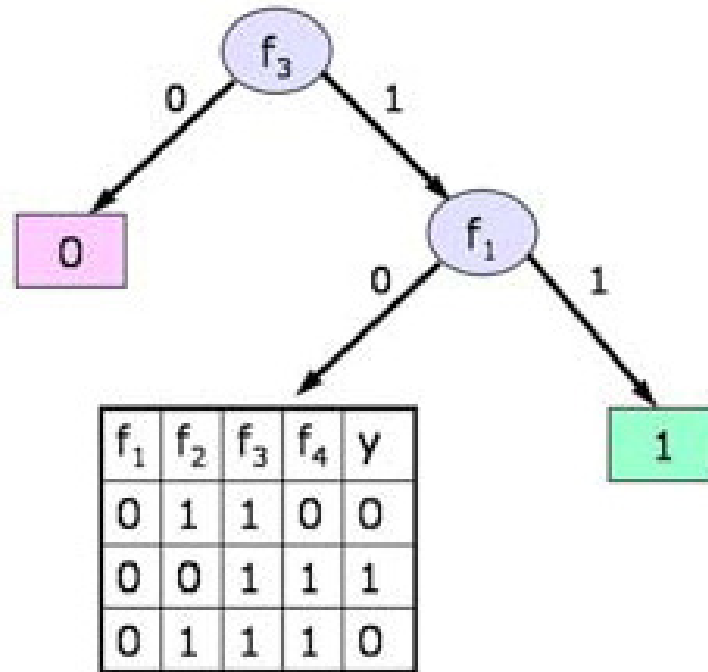
- Features 1 and 2 are equally useful, and feature 4 is basically no help at all.

Simulation

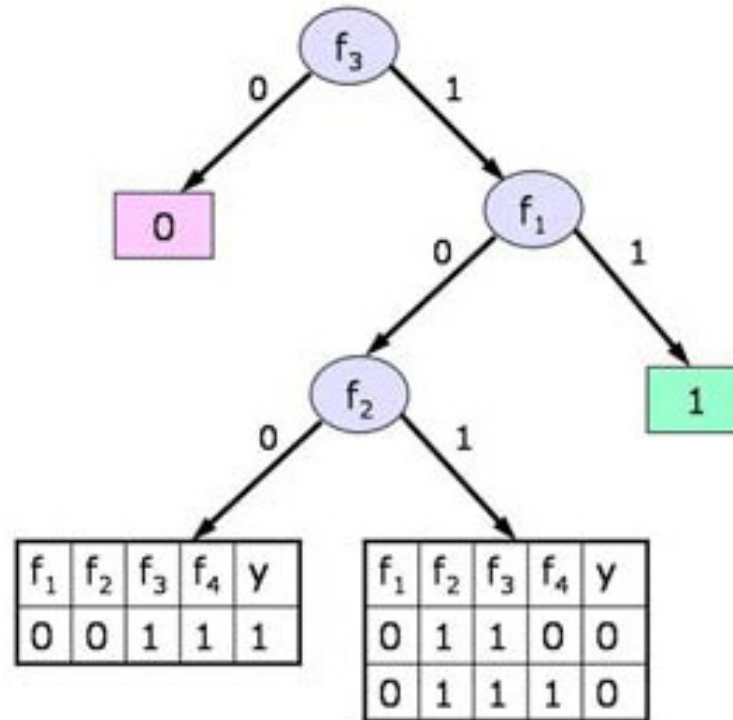


- All of the examples in the right-hand child have the same output

Simulation

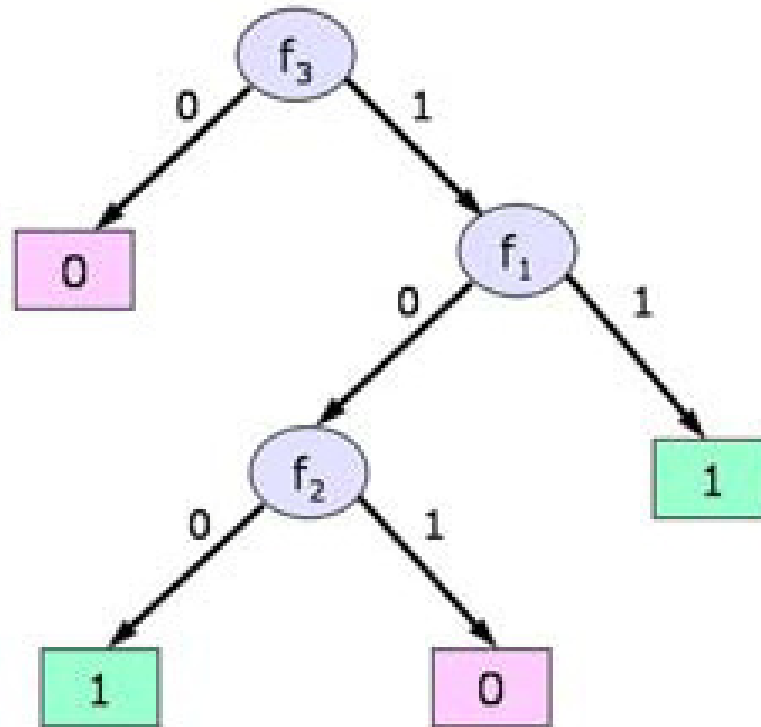


Simulation



- Now both children are homogeneous (all data points have the same output)

Simulation





Exclusive OR

$$(A \wedge \neg B) \vee (\neg A \wedge B)$$

- One class of functions that often haunts us in machine learning are those that are based on the “exclusive OR”
- Exclusive OR is the two-input boolean function that has value 1 if one input value 1 and the other has value 0.
- If both inputs are 1 or both are 0, then the output is 0
- This function is hard to deal with, because neither input feature is, by itself, detectably related to the output
- So methods that try to add one feature at a time can be easily misled by xor

Exclusive OR

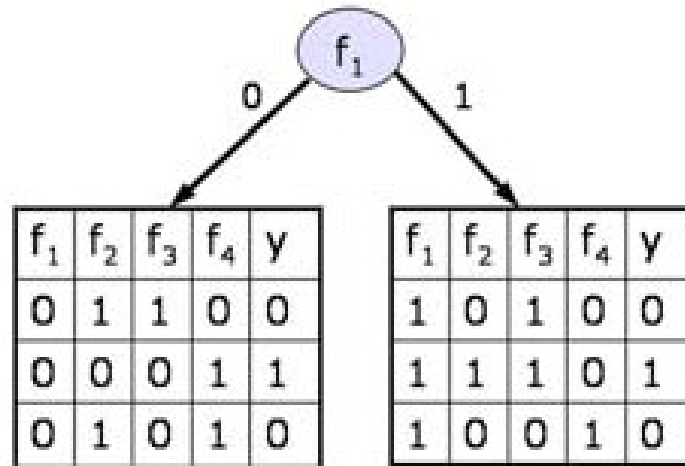
$$(A \wedge \neg B) \vee (\neg A \wedge B)$$

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	0	0
1	1	1	0	1
0	0	0	1	1
1	0	0	1	0
0	1	0	1	0

- $H(D) = .92$
- $AE_1 = .92, AE_2 = .92,$
 $AE_3 = .92, AE_4 = .92$

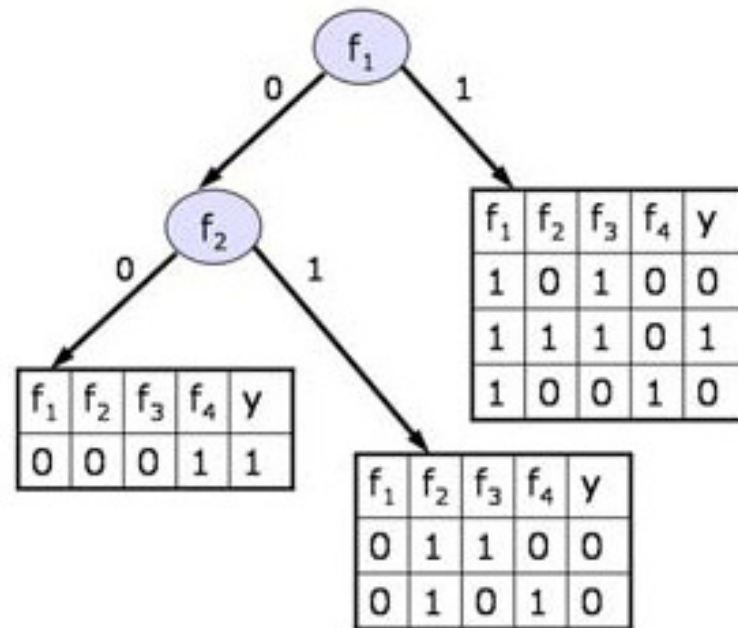
- In this example, no matter what attribute we split on, the average entropy is 0.92
- If we were using the stopping criterion that says we should stop when there is no split that improves the average entropy, we'd stop now

Exclusive OR



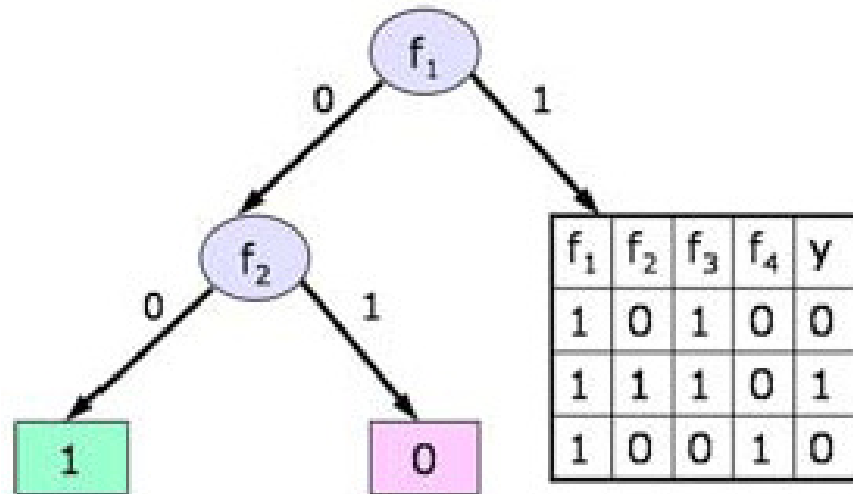
- But let's go ahead and split on feature 1
- Now, if we look at the left-hand data set, we'll see that feature 2 will have an average entropy of 0

Exclusive OR



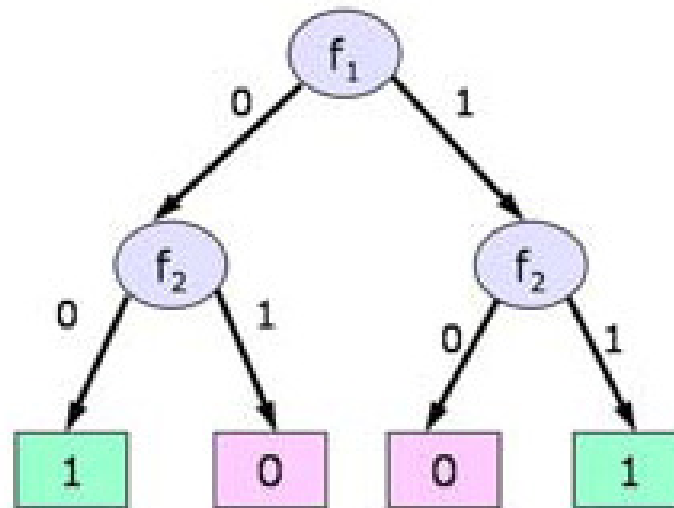
•So, we split on it and get homogeneous children

Exclusive OR



- Which we can replace with leaves
- Now it's easy to see that feature 2 will again be useful here

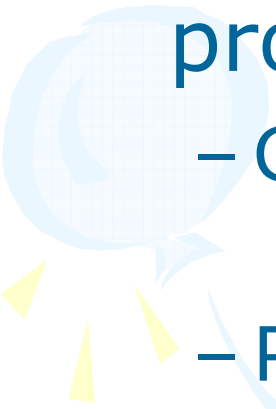
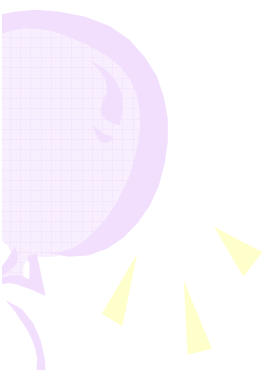
Exclusive OR



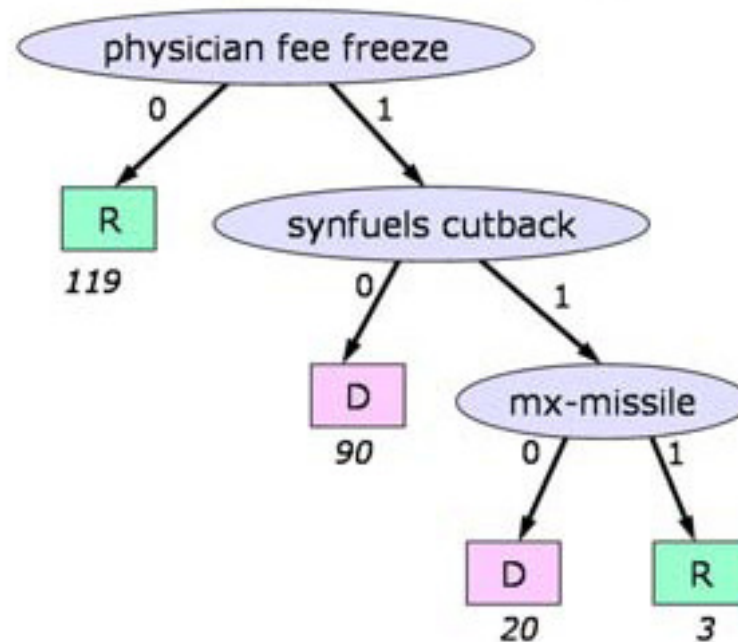
- And we go straight to leaves on this side
- So we can see that, although no single feature could reduce the average entropy of the child data sets, features 1 and 2 were useful in combination



Pruning

- Best way to avoid overfitting and not get tricked by short-term lack of progress
 - Grow tree as far as possible
 - Leaves are uniform or contain a single X
 - Prune the tree until it performs well on held-out data (using something like an entropy cut-off)
- 
- 

Congressional Voting



min leaf size = 20

- Instead of pruning, we use the minimum leaf size.
- If it reaches a node with fewer than that number of examples, it stops and makes a leaf with the majority output
- Here's the tree we get when the minimum leaf size is 20



Data Mining

- Decision trees very popular because
 - Easy to implement
 - Efficient (even on huge data sets)
 - Easy for humans to understand resulting hypotheses
- Data mining is the application of learning algorithms to problems of interest in many industries
 - Decision trees is one of the principle methods used in data mining – to make useful predictions (usually in corporate applications)