CS 175: Project in Artificial Intelligence

Slides 6: Decision Trees

Topic 8: Decision Trees

Some slides taken from Prof. Welling, Smyth, & Ihler

Decision Trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

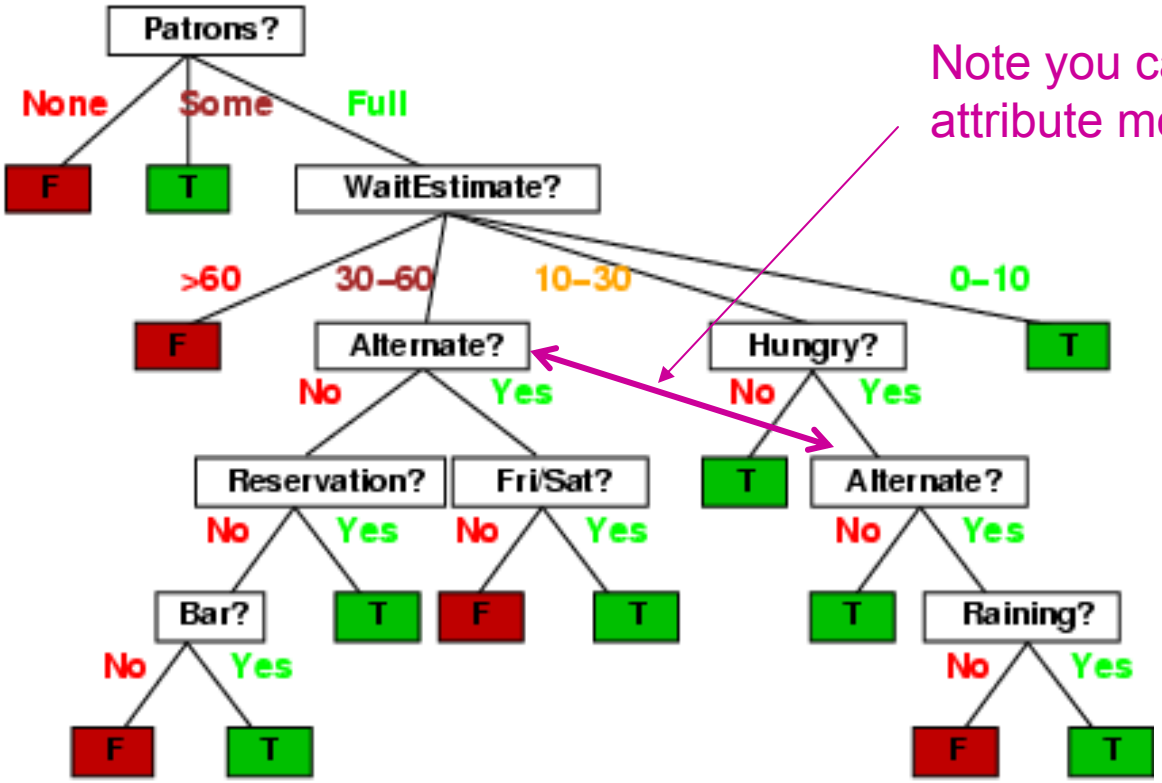
- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- Classification of examples is positive (T) or negative (F)
- General form for data: a number N of instances, each with attributes (x1,x2,x3,...xd) and target value y.

Decision trees

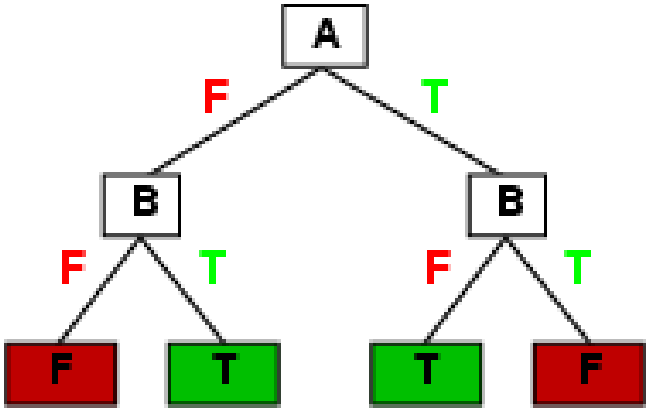
- One possible representation for hypotheses
- We imagine someone taking a sequence of decisions.
- E.g., here is the “true” tree for deciding whether to wait:



Expressiveness

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples

- Prefer to find more compact decision trees: we don't want to memorize the data, we want to find structure in the data!

Hypothesis spaces

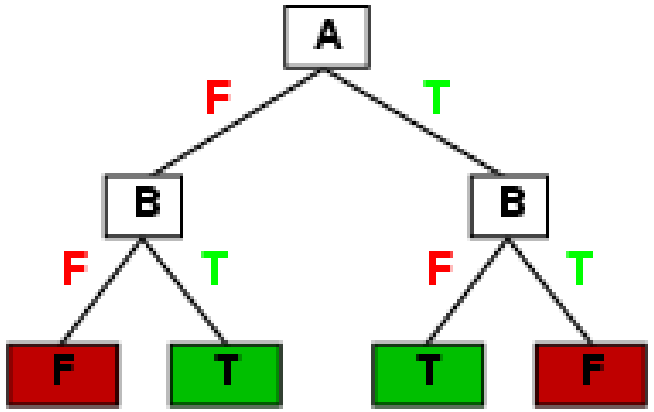
How many distinct decision trees with *n* Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



n=2: 2^2 = 4 rows. For each row we can choose T or F: 2^4 functions.

Decision tree learning

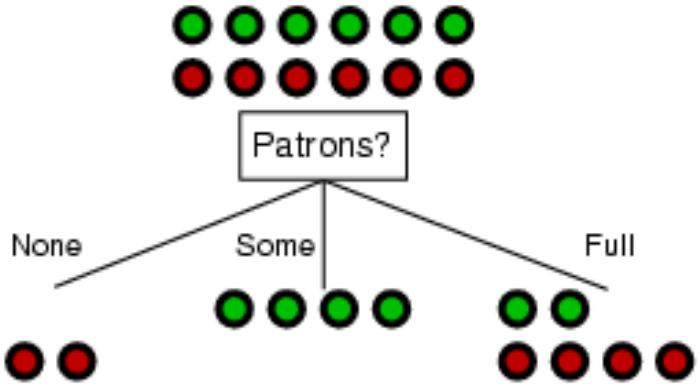
- If there are so many possible trees, can we actually search this space? (solution: greedy search).

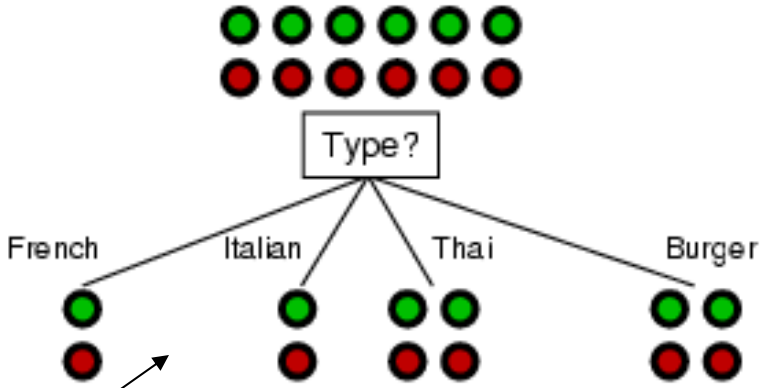
- Aim: find a small tree consistent with the training examples

- Idea: (recursively) choose "most significant" attribute as root of (sub)tree.

Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"





- *Patrons or type?*

To wait or not to wait is still at 50%.

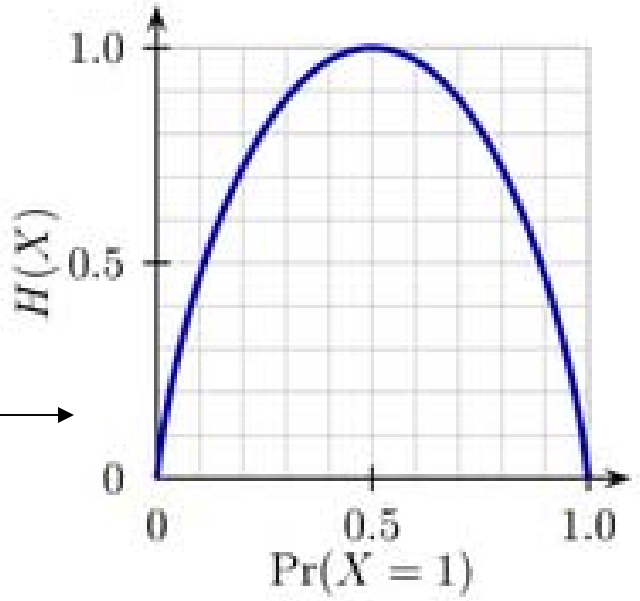
Using information theory

- Entropy measures the amount of uncertainty in a probability distribution:

Consider tossing a biased coin.
If you toss the coin VERY often,
the frequency of heads is, say, p,
and hence the frequency of tails is
1-p. (fair coin p=0.5).

The uncertainty in any actual outcome is given by the entropy:

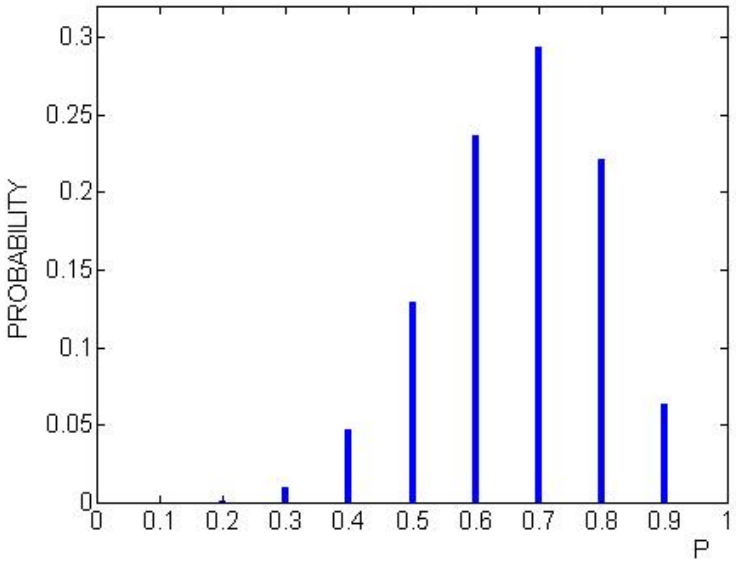
Note, the uncertainty is zero if p=0 or 1
and maximal if we have p=0.5.



Using information theory

- If there are more than two states s=1,2,..n we have (e.g. a die):

$$Entropy(p) = -p(s=1)\log[p(s=1)] - p(s=2)\log[p(s=2)] \ldots - p(s=n)\log[p(s=n)]$$



$$\sum_{s=1}^{n} p(s) = 1$$

Using information theory

- Imagine we have p examples which are true (positive) and n examples which are false (negative).

- Our best estimate of true or false is given by:

$$P(true) \approx p / p+n$$

$$p(false) \approx n / p+n$$

- Hence the entropy is given by:

$$Entropy(\frac{p}{p+n}, \frac{n}{p+n}) \approx -\frac{p}{p+n}\log\frac{p}{p+n} - \frac{n}{p+n}\log\frac{n}{p+n}$$

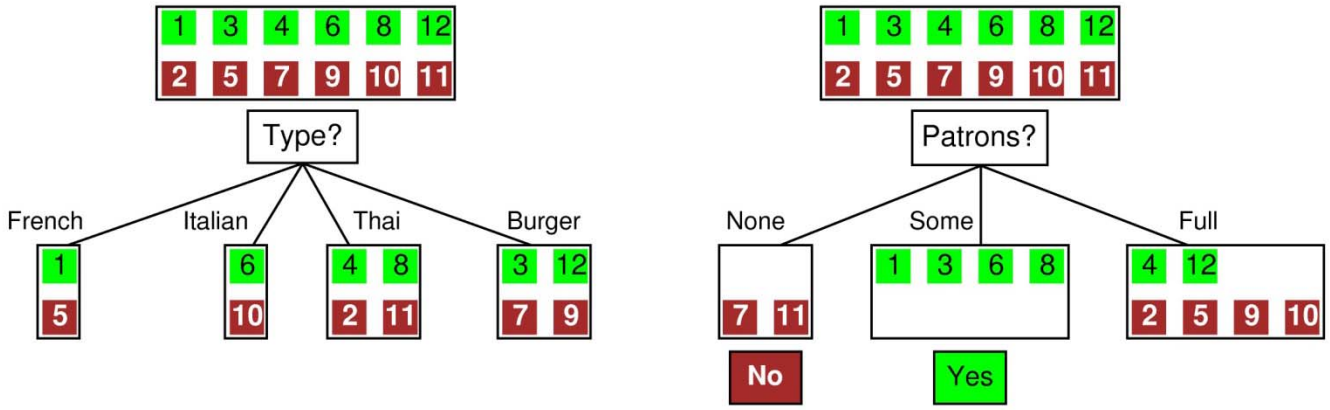
Using Information Theory

- How much information do we gain if we disclose the value of some attribute?

- Answer:

uncertainty before minus uncertainty after

Example



Before: Entropy = - ½ log(1/2) – ½ log(1/2)=log(2) = 1 bit:
There is "1 bit of information to be discovered".

After: for Type: If we go into branch "French" we have 1 bit, similarly for the others.

French: 1bit
Italian: 1 bit
Thai: 1 bit
Burger: 1bit

} On average: 1 bit ! We gained nothing!

After: for Patrons: In branch "None" and "Some" entropy = 0!,
In "Full" entropy = -1/3log(1/3)-2/3log(2/3).

So Patrons gains more information!

Information Gain

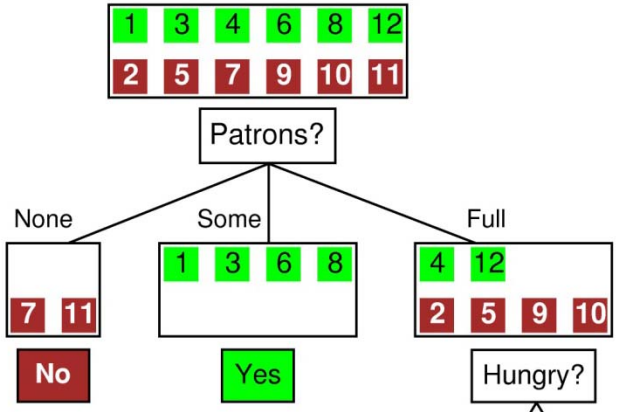
- How do we combine branches:

1/6 of the time we enter “None”, so we weight“None” with 1/6.
Similarly: “Some” has weight: 1/3 and “Full” has weight ½.

$$Entropy(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} Entropy\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

entropy for each branch.

weight for each branch



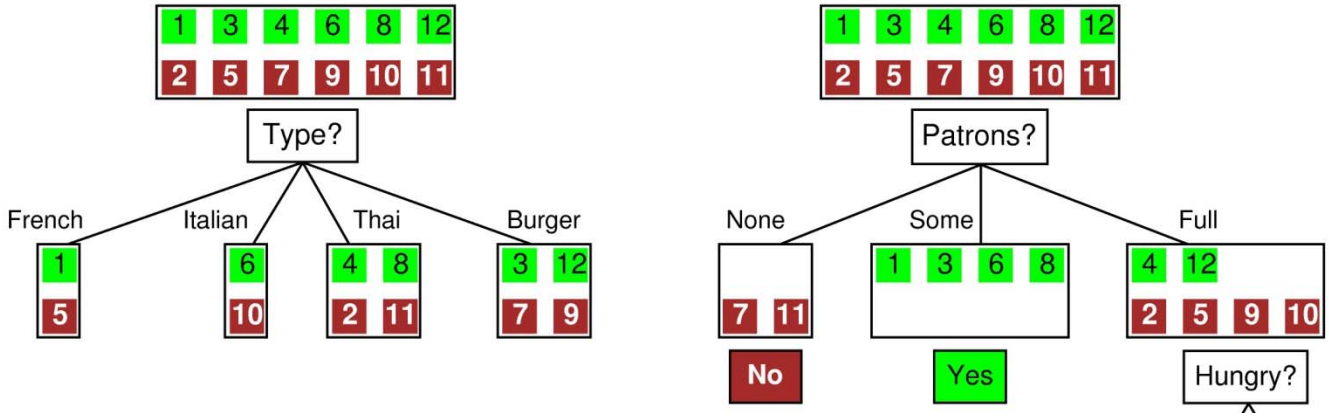
Information gain

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = Entropy\ before - Entropy\ after$$

- Choose the attribute with the largest IG

Information gain



For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .0541 \text{ bits}$$

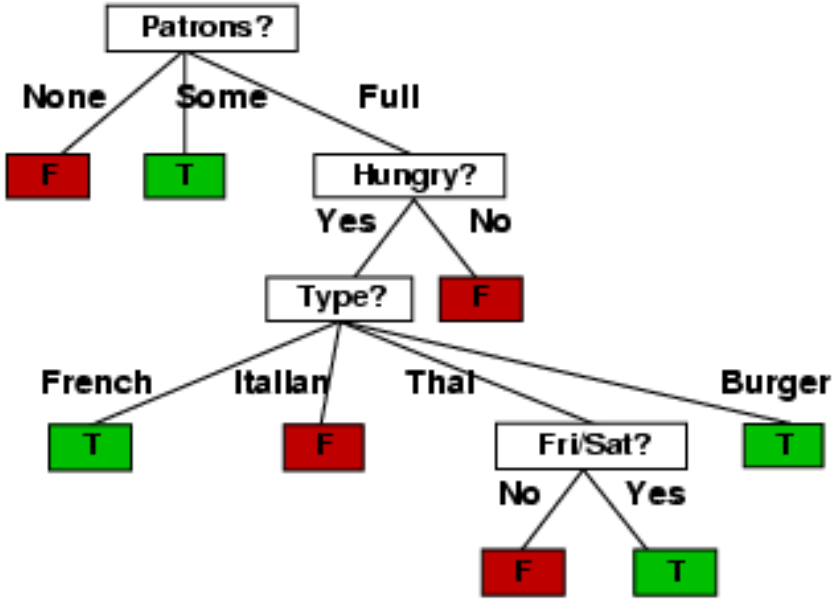
$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

Is it sensible to have the same attribute on a single branch of the tree (why)?

Example contd.

- Decision tree learned from the 12 examples:



- Substantially simpler than “true” tree---a more complex hypothesis isn’t justified by small amount of data

Gain-Ratio

- If 1 attribute splits in many more classes than another, it has an (unfair) advantage if we use information gain.

- The gain-ratio is designed to compensate for this problem,

$$Gain - Ratio = \frac{Info - Gain}{-\sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \log \frac{p_i + n_i}{p + n}}$$

- if we have n uniformly populated classes the denominator is log2(n) which penalized relative to 1 for 2 uniformly populated classes.

What to Do if...

- In some leaf there are no examples:

  Choose True or False according to the number of positive/negative examples at your parent.
- There are no attributes left

  Two or more examples have the same attributes but different label: we have an error/noise. Stop and use majority vote.

Continuous Variables

- If variables are continuous we can bin them, or..
- We can learn a simple classifier on a single dimension
- E.g. we can find a decision point which classifies all data to the left of that point in one class and all data to the right in the other (decision stump – next slide)
- We can also use a small subset of dimensions and train a linear classifier (e.g. logistic regression classifier).

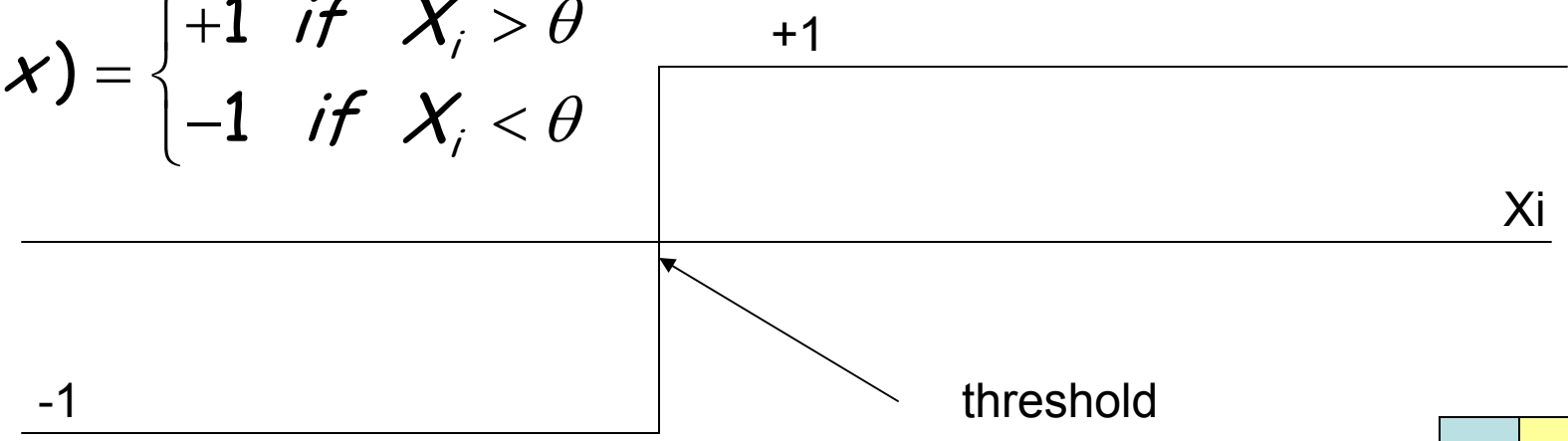
Decision Stump

- Data: {(X1,Y1),(X2,Y2),....,(Xn,Yn)}

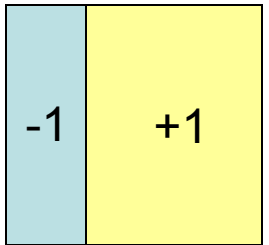
attributes (e.g. temperature outside)

label (e.g. True or False,
0 or 1
-1 or +1)

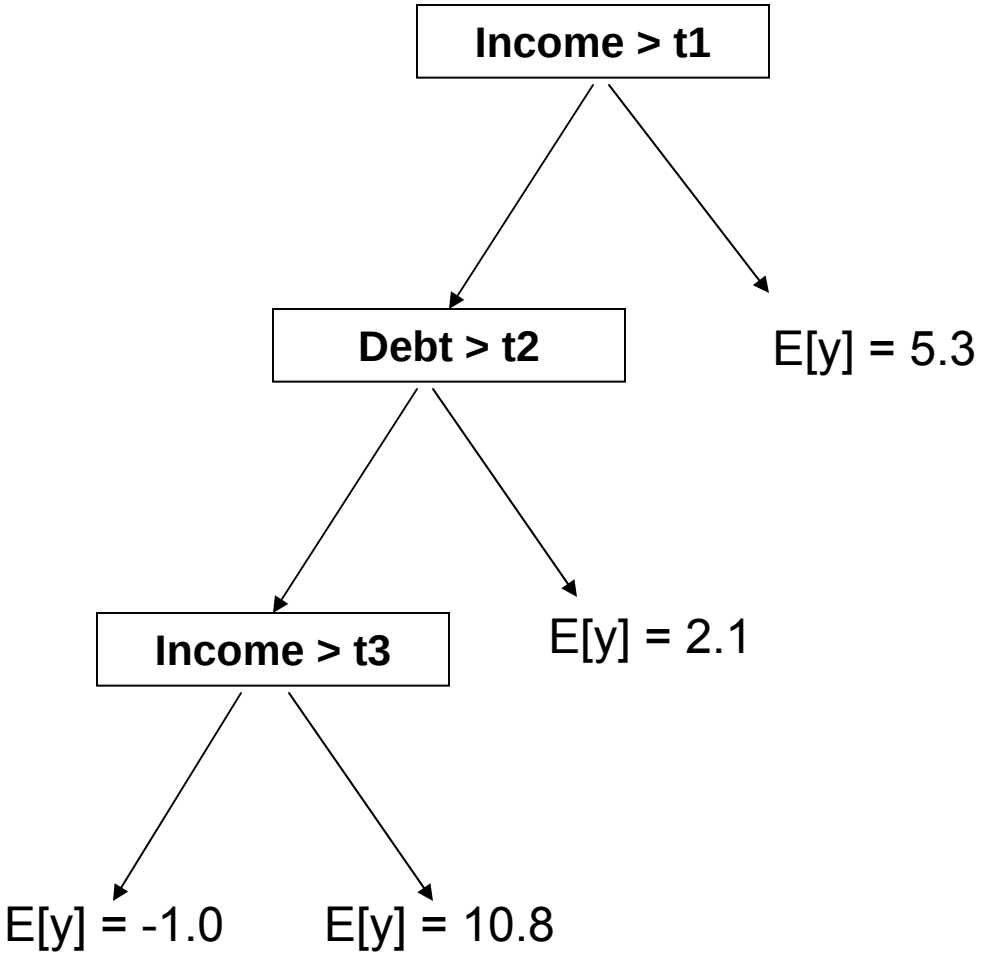
$$h(x) = \begin{cases} +1 & if \quad X_i > \theta \\ -1 & if \quad X_i < \theta \end{cases}$$



So, we choose one attribute “i”, a sign “+/-” and and a threshold θ
This determines a half space to be +1 and the other half -1.



Simple example of Regression Tree
(where target variable is continuous)



Greedy Search for Learning Regression Trees

- Binary_node_splitting, real-valued variables
  - For each variable x_j
    - For each possible threshold t_j , compute

$$MSE(y;t_j) = P(x \leq t_j) MSE(y|x \leq t_j) + P(x > t_j) MSE(y|x > t_j)$$

    - Select t_j with the lowest MSE for that variable
  - Select variable x_j and t_j with the lowest MSE
  - Split the training data into the 2 branches
  - For each branch
    - If leaf-node: prediction at this leaf node = mean value of y data points
    - If not: call binary_node_splitting recursively

When to Stop ?

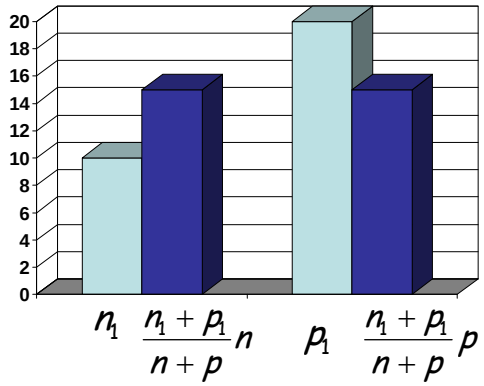
- If we keep going until perfect classification we might over-fit.

- Heuristics:
  - Stop when Info-Gain (Gain-Ratio) is smaller than threshold
  - Stop when there are M examples in each leaf node

- Penalize complex trees by minimizing $\alpha \times complexity + \sum_{all\ leafs} entropy(leaf)$ with "complexity" = # nodes.
  Note: if tree grows, complexity grows but entropy shrinks.

- Compute many full grown trees on subsets of data and test them on hold-out data. Pick the best or average their prediction (random forest – next slide)

- Do a statistical test: is the increase in information significant.



$$\chi^2_{1,\alpha} = \frac{\left(\frac{n_1+p_1}{n+p}n - n_1\right)^2}{\left(\frac{n_1+p_1}{n+p}n\right)} + \frac{\left(\frac{n_1+p_1}{n+p}p - p_1\right)^2}{\left(\frac{n_1+p_1}{n+p}p\right)}$$

Pruning & Random Forests

- Sometimes it is better to grow the trees all the way down and prune them back later. Later splits may become beneficial, but we wouldn't see them if we stopped early.

- We simply consider all child nodes and consider them for elimination, using any of the above criteria.

- Alternatively, we grow many trees on datasets sampled from the original dataset with replacement (a bootstrap sample) .

    - Draw K bootstrap samples of size N
    - Grow a DT, randomly sampling one or a few attributes/dimensions to split on
    - Average the predictions of the trees for a new query (or take majority vote)

- Evaluation: for each bootstrap sample about 30% of the data was left out. For each data-item determine which trees did not use that data-case and test performance using this sub-ensemble of trees.

- Random Forests are state of the art classifiers!

Bagging

- The idea behind random forests can be applied to any classifier and is called *bagging:*
  - Sample M bootstrap samples.
  - Train M different classifiers on these bootstrap samples
  - For a new query, let all classifiers predict and take an average (or majority vote)

- The basic idea that this works is that if the classifiers make independent errors, then their ensemble can improve performance.

- Stated differently: the variance in the prediction is reduced (we don't suffer from the random errors that a single classifier is bound to make).

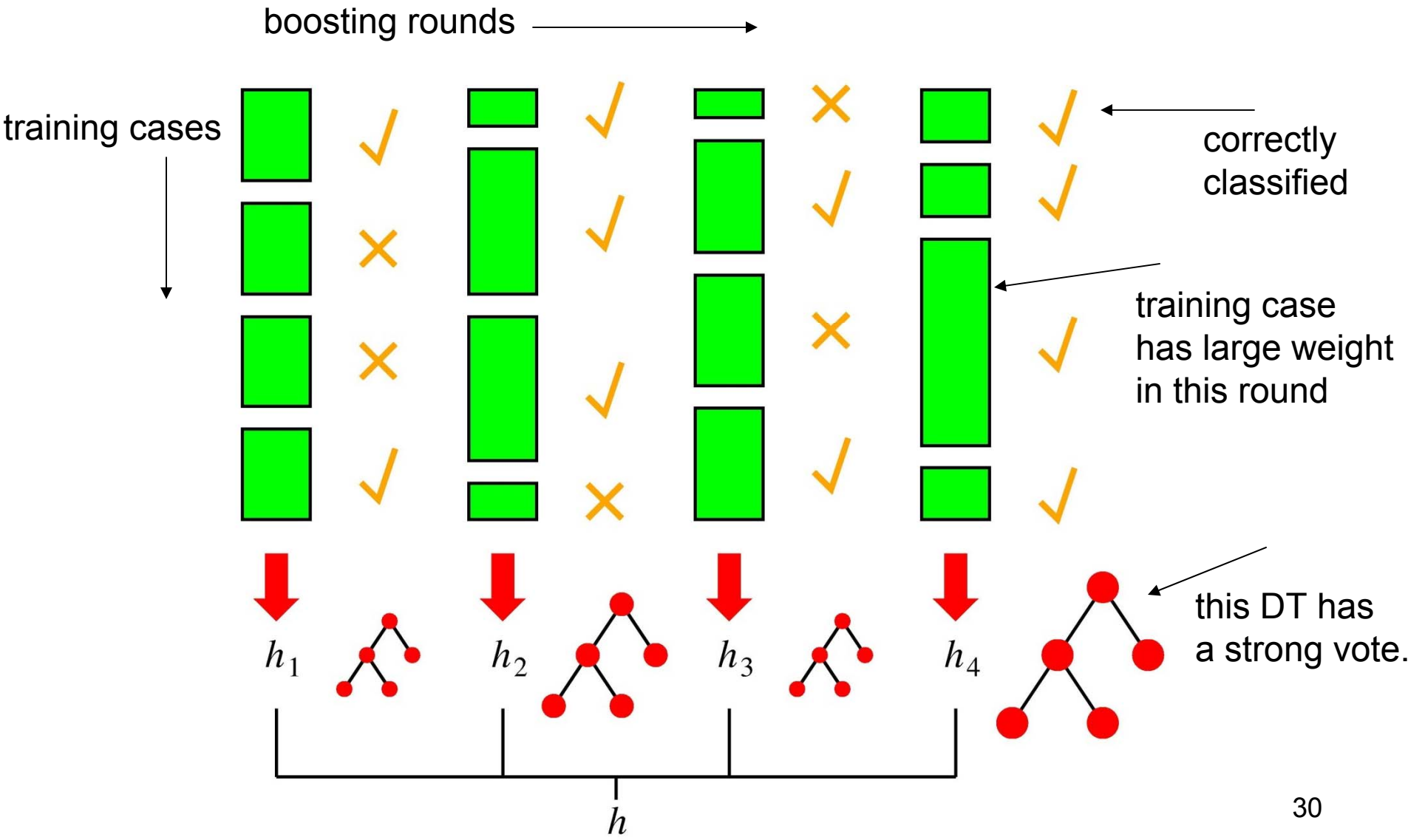
Boosting

- Main idea:
  - train classifiers (e.g. decision trees) in a sequence.
  - a new classifier should focus on those cases which were incorrectly classified in the last round.
  - combine the classifiers by letting them vote on the final prediction (like bagging).
  - each classifier could be (should be) very “weak”, e.g. a decision stump.

Boosting Intuition

- We adaptively weigh each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that that obtain low training error rate have high weight.
- We stop by using monitoring a hold out set (cross-validation).

Boosting in a Picture



Building a decision tree

- Pseudo-code

```
decisionTreeSplitData(X,Y)
 if (stopping condition) return decision for this node
 For each possible feature
  For each possible split
   (for cts features: sort & compute split points)
   Score the split  (e.g. information gain)
 Pick the feature & split with the best score
 Split the data at that point
 Recurse on each subset
```

Stopping conditions:

* # of data < K
* Depth > D
* All data indistinguishable (discrete features)
* Can later "prune" the tree using hold-out data

* Information gain threshold? Often not a good idea…
  Can instead grow a large tree and prune back, using cross-validation data

Decision trees in Matlab

- Stats toolbox
  - "classregtree" (older versions: "treefit")
  - If Y is a double array, regression
  - If Y is a logical array (or other things), classify
    - Uses Gini index by default

```
T=classregtree(x,logical(y),'splitmin',1);
% was T=treefit(...)
T,
Decision tree for classification
1   If x1 < 0.475 then node 2 else node 3
2   If x2 < 0.531 then node 4 else nod 5
3   class = 1
4   class = 0
5   If x1 < .125 then node 6 else node 7
6   class = 0
7   class = 1

view(T);    % was treedisp(T)
% NOTE: test returns 1 or 2, not 0 or 1...
```