CS 886
Applied Machine Learning
Decision Trees

Dan Lizotte

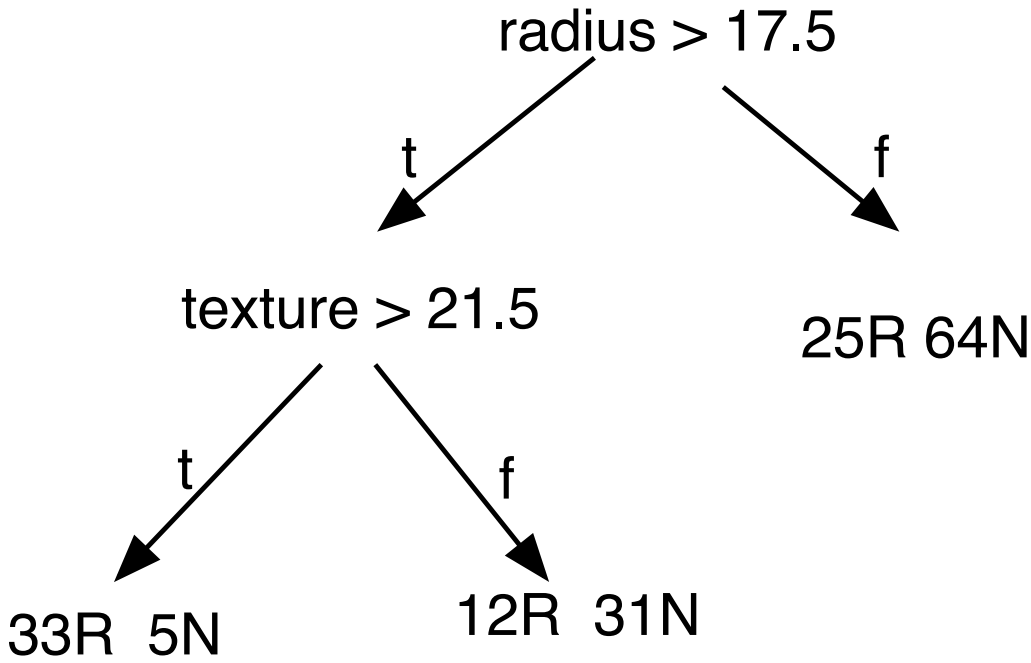
University of Waterloo

3 Oct 2014

Decision Trees

- What are decision trees?
- Methods for constructing decision trees
- Overfitting avoidance

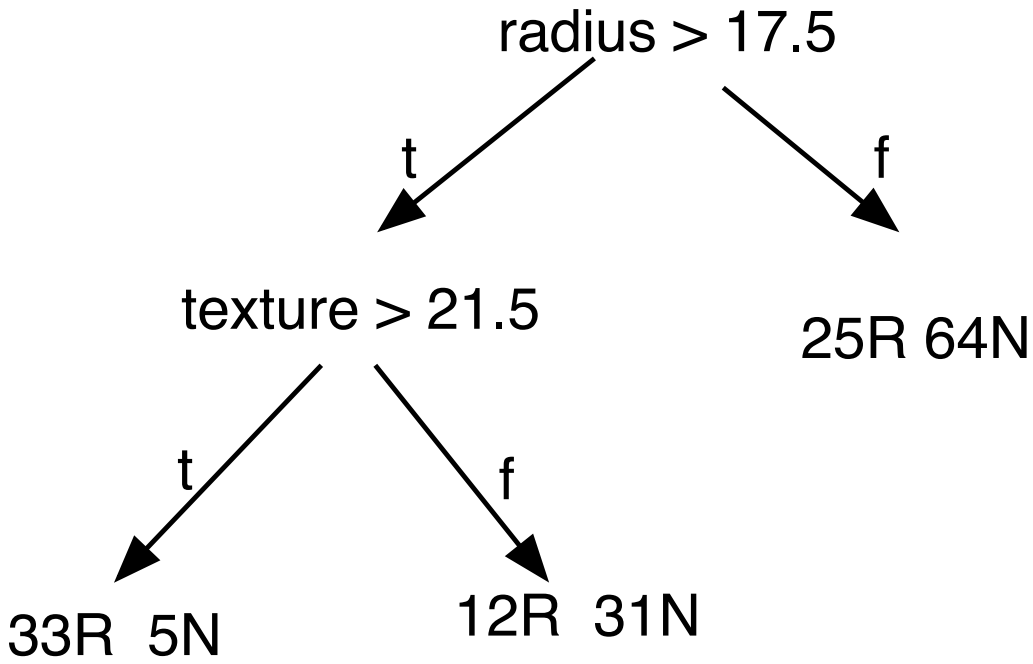
Example: Decision tree for Wisconsin data



- Internal nodes are tests on the values of different attributes
- Tests can be binary or multi-valued
- Each training example $\langle \mathbf{x}_i, y_i \rangle$ falls in precisely one leaf.

Using decision trees for classification

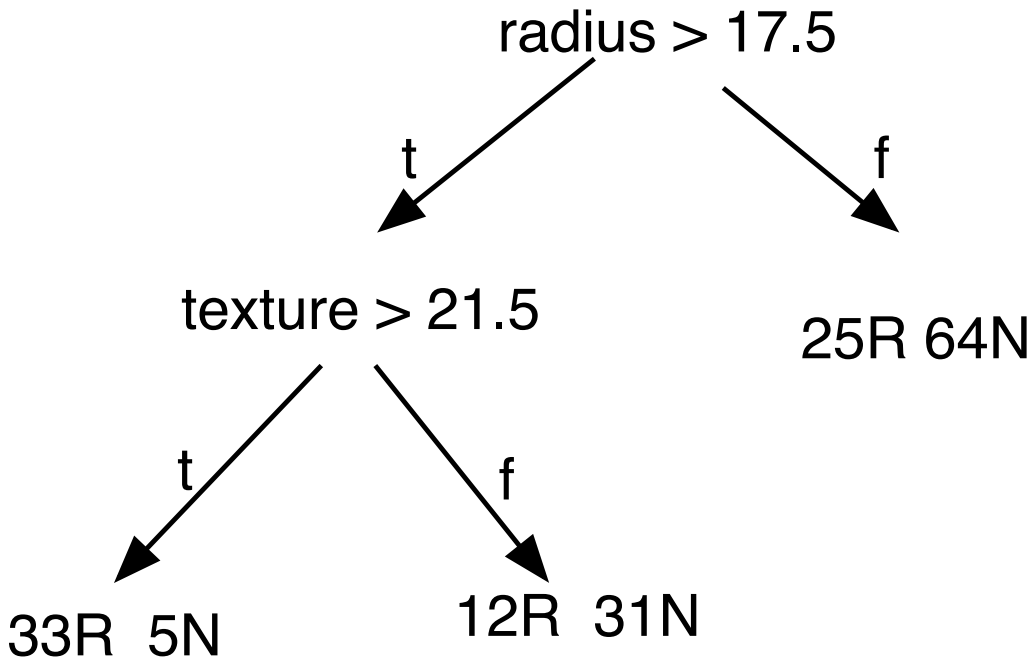
How do we classify a new a new instance, e.g.: *radius=18, texture=12, . . .*



- At every node, test the corresponding attribute
- Follow the appropriate branch of the tree
- At a leaf, one can predict the class of the majority of the examples for the corresponding leaf, or the probabilities of the two classes.

Decision trees as rules

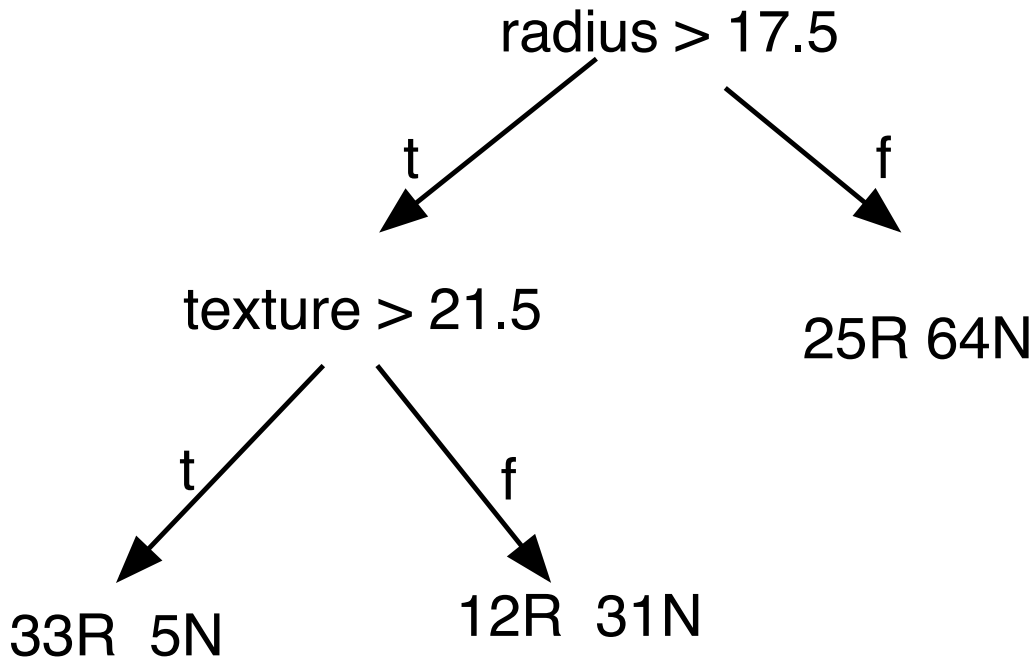
A decision tree can be converted an equivalent set of if-then rules.



IF	THEN most likely class is
radius>17.5 AND texture>21.5	R
radius>17.5 AND texture≤21.5	N
radius≤17.5	N

Decision trees as rules

A decision tree can be converted an equivalent set of if-then rules.



IF	THEN P(R) is
radius>17.5 AND texture>21.5	$33/(33+5)$
radius>17.5 AND texture$\leq$21.5	$12/(12+31)$
radius$\leq$17.5	$25/(25+64)$

Decision trees, more formally

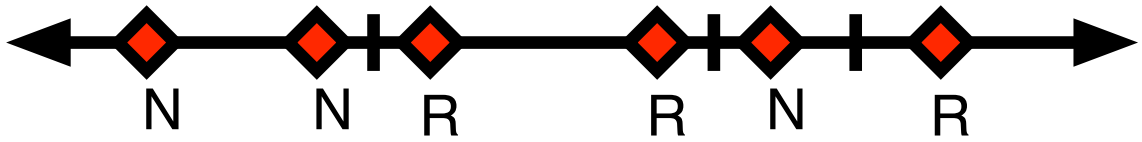
- Each internal node contains a *test*, on the value of one (typically) or more feature values
- A test produces discrete outcomes, e.g.,
  - radius > 17.5
  - radius $\in [12, 18]$
  - grade is $\{A, B, C, D, F\}$
  - grade is $\geq B$
  - color is RED
  - $2*\text{radius} - 3*\text{texture} > 16$
- For discrete features, typically branch on some, or all, possible values
- For real features, typically branch based on a threshold value

$\Rightarrow$ *A finite set of possible tests* is usually decided before learning the tree; learning comprises choosing the shape of the tree and the tests at every node.

More on tests for real-valued features

- Suppose feature j is real-valued,
- How do we choose a finite set of possible thresholds, for tests of the form $x_j > \tau$?
  - Regression: consider midpoints of the observed data values, $x_{1,j}, x_{2,j}, \ldots, x_{m,j}$

  - Classification: consider midpoints of data values with different y values



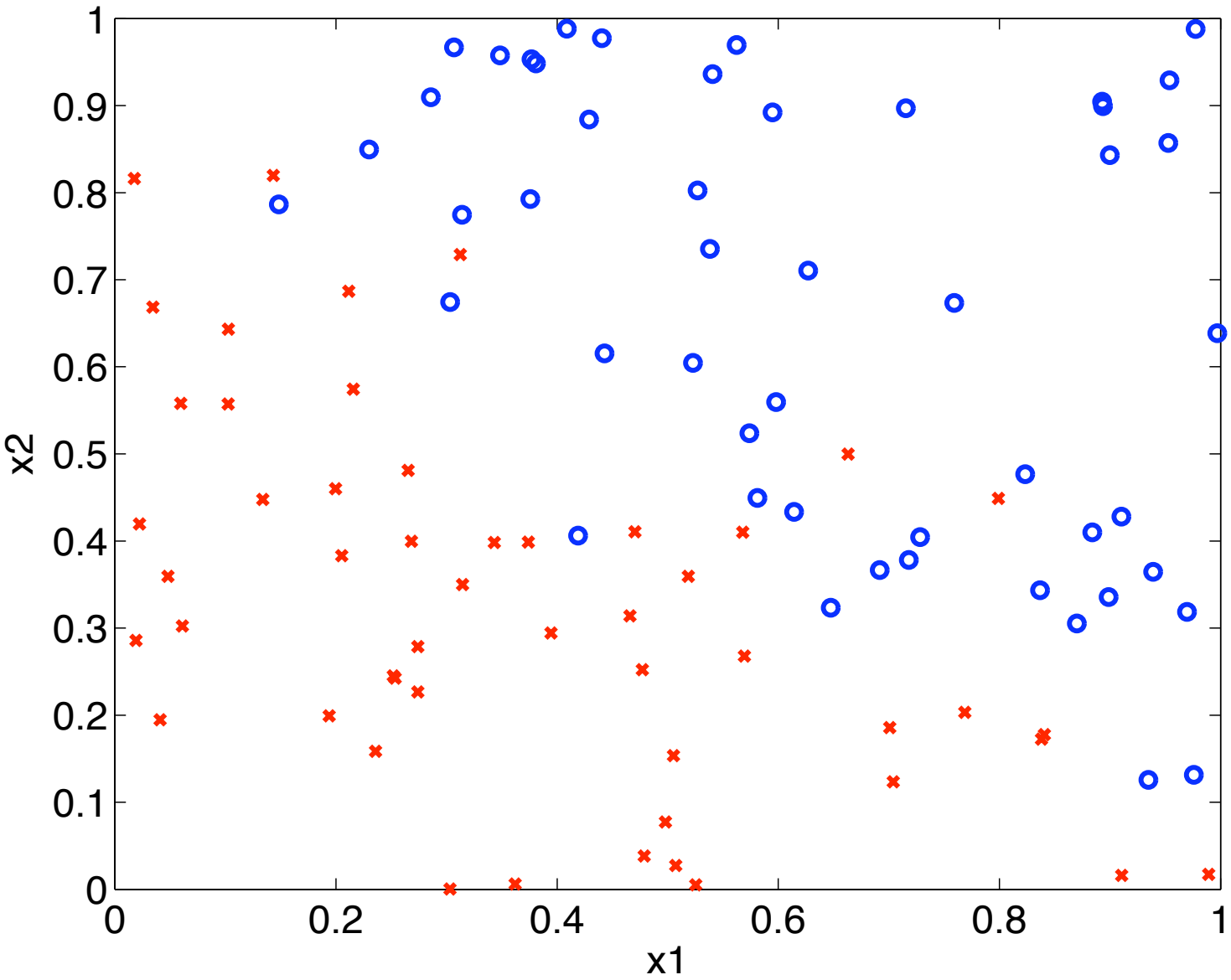
Representational power and efficiency of decision trees

- Suppose the input $\mathbf{x}$ consists of n binary features
- How can a decision tree represent:
  - $y = x_1$ AND x_2 AND $\ldots$ AND x_n
  - $y = x_1$ OR x_2 OR $\ldots$ OR x_n
  - $y = x_1$ XOR x_2 XOR $\ldots$ XOR x_n

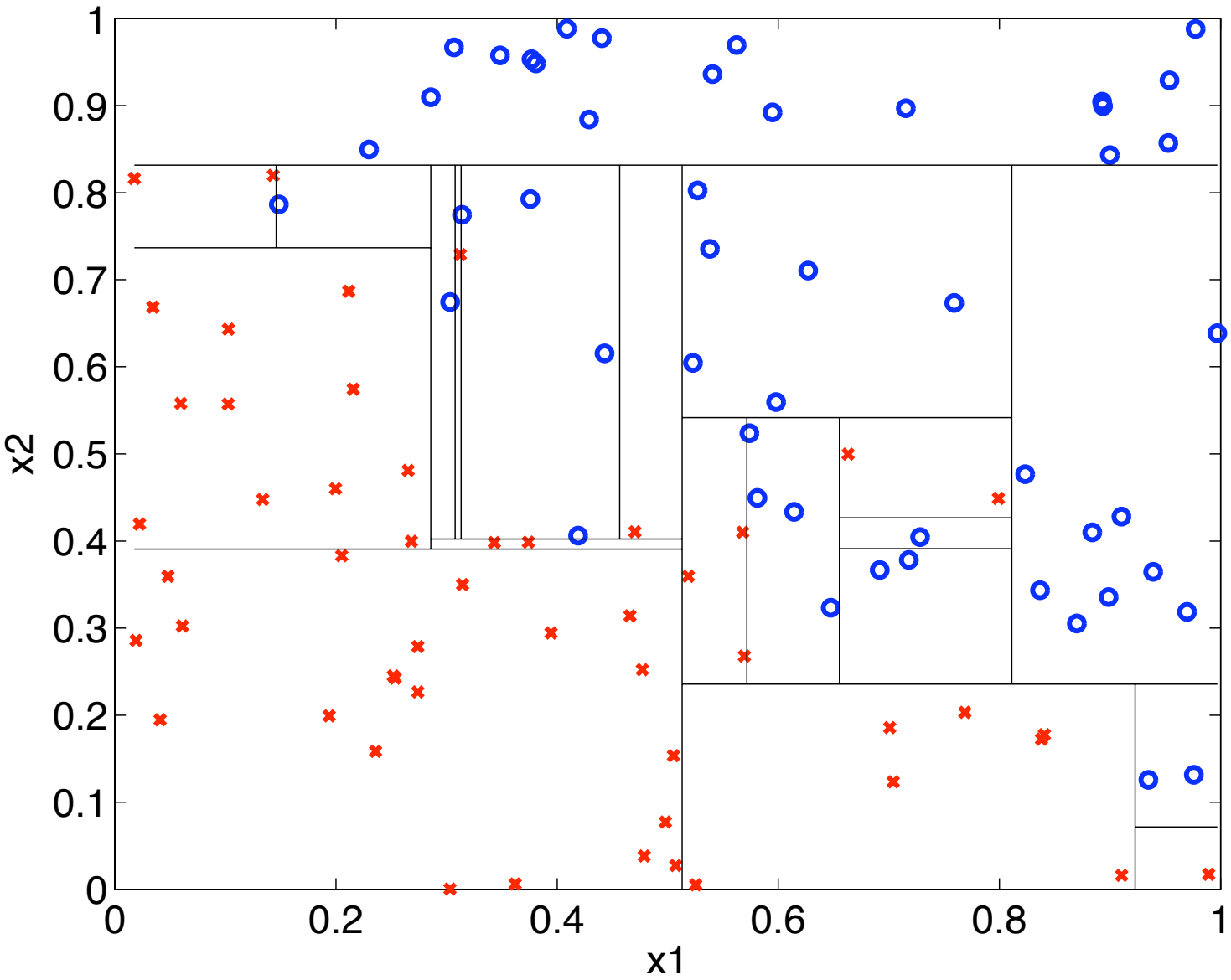
Representational power and efficiency of decision trees

- With typical univariate tests, AND and OR are easy, taking $O(n)$ tests
- Parity/XOR type problems are hard, taking $O(2^n)$ tests
- With real-valued features, decision trees are good at problems in which the class label is **constant in large, connected, axis-aligned regions of the input space.**

An artificial example



Example: Decision tree decision surface



How do we learn decision trees?

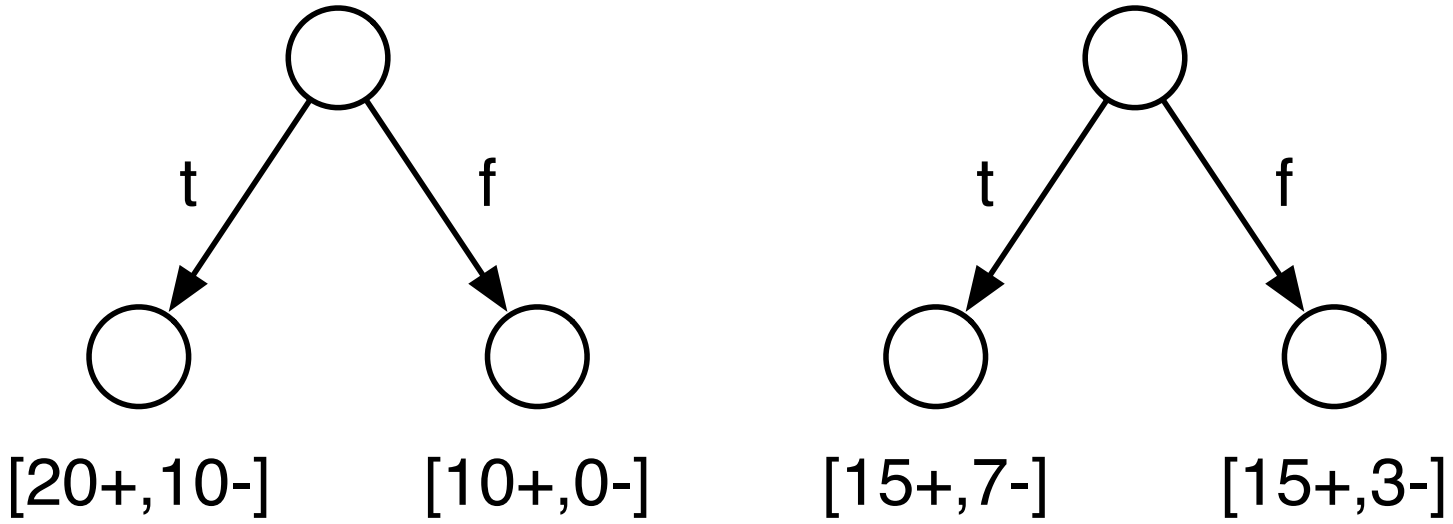
- We could enumerate all possible trees (assuming number of possible tests is finite),
  - Each tree could be evaluated using the training set or, better yet, a validation set
  - But there are many possible trees! Combinatorial problem...
  - We'd probably overfit the data anyway
- Usually, decision trees are constructed in two phases:
  1. An recursive, top-down procedure "grows" a tree (possibly until the training data is exactly fit)
  2. The tree is "pruned" back to avoid overfitting
- Both typically use *greedy heuristics*

Top-down induction of decision trees

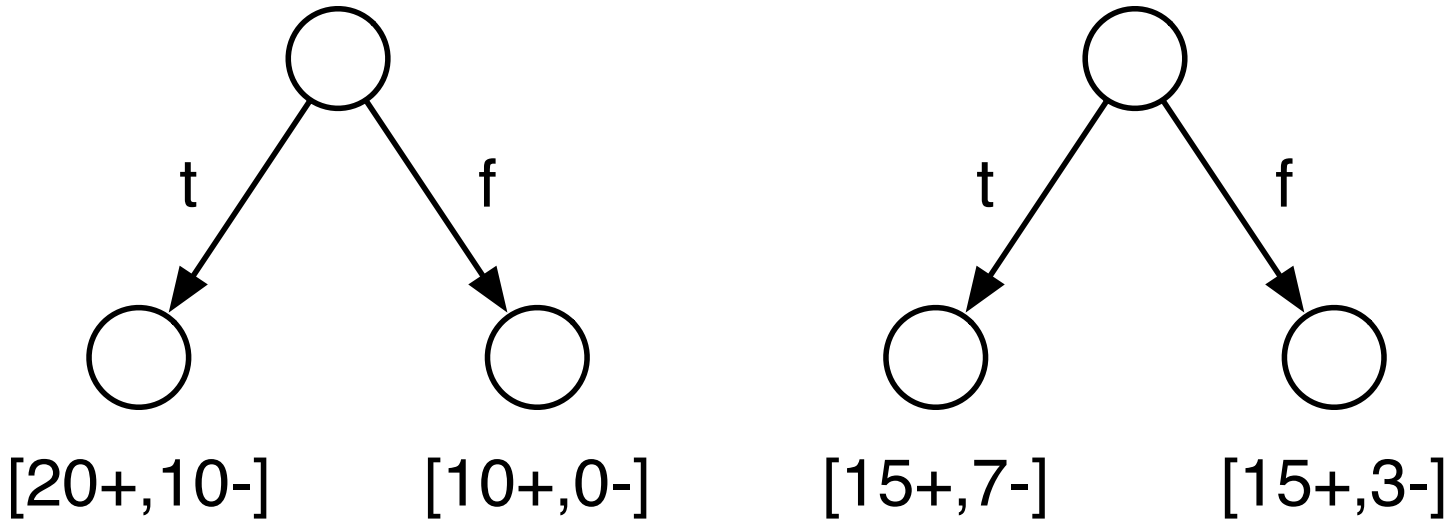
- For a classification problem:
  1. If all the training instances have the same class, create a leaf with that class label and exit.
  2. Pick the *best test* to split the data on
  3. Split the training set according to the outcome of the test
  4. Recurse on each subset of the training data
- For a regression problem - same as above, except:
  - Stop splitting when leaf sample size falls below some threshold
  - At a leaf, either predict the mean value, or do a linear fit

Which test is best?

- The test should provide *information* about the class label.
- Suppose we have 30 positive examples, 10 negative ones, and we are considering two tests that would give the following splits of instances:



Which test is best?



- Intuitively, we would like an attribute that *separates* the training instances as well as possible
- If each leaf was pure, the attribute would provide maximal information about the label at the leaf
- We need a mathematical measure for the purity of a set of instances

Information=Reduction in uncertainty

Let E be an event that occurs with probability $P(E)$. If we are told that E has occurred with certainty, then we received

$$I(E) = \log_2 \frac{1}{P(E)}$$

bits of information.

- You can also think of information as the amount of "surprise" in the outcome (e.g., consider $P(E) = 1$, $P(E) \approx 0$)
- E.g., result of a fair coin flip provides $\log_2 2 = 1$ bit of information
- E.g., result of a fair dice roll provides $\log_2 6 \approx 2.58$ bits of information.
- E.g., result of being told your own name (or any other deterministic event) produces 0 bits of information

Entropy

- Suppose we have an information source S which emits symbols from an alphabet $\{s_1, \ldots s_k\}$ with probabilities $\{p_1, \ldots p_k\}$. Each emission is independent of the others.
- What is the *average amount of information* when observing the output of S?

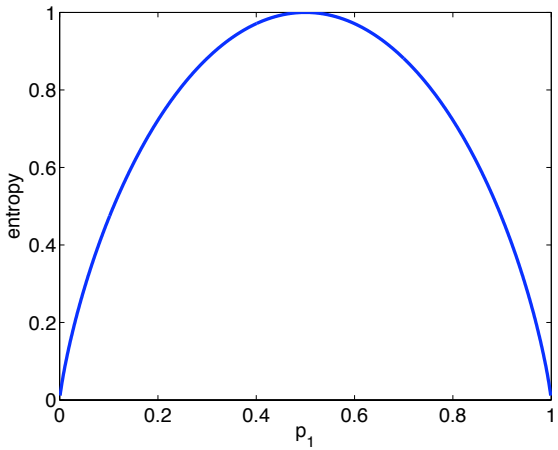
$$H(S) = \sum_i p_i I(s_i) = \sum_i p_i \log \frac{1}{p_i} = -\sum_i p_i \log p_i$$

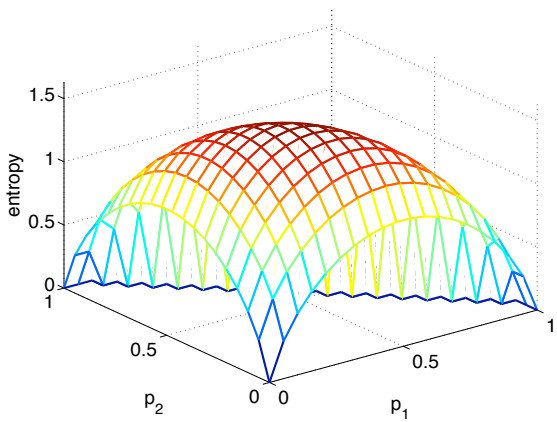
  This is called the *entropy* of S.
- Note that this depends *only on the probability distribution* and not on the actual alphabet (so we can really write $H(P)$).

Properties of entropy

$$H(P) = \sum_{i=1}^{k} -p_i \log p_i$$

- Non-negative: $H(P) \geq 0$, with equality if and only if any $p_i = 1$.
- $H(P) \leq \log k$ with equality if and only if $p_i = \frac{1}{k}, \forall i$
- The "further" P is from uniform, the lower the entropy P has





Entropy applied to binary classfication

- Consider data set D and let
  - $p_{\oplus}$ = the proportion of positive examples in D
  - $p_{\ominus}$ = the proportion of negative examples in D
- Entropy measures the impurity of D, based on empirical probabilities of the two classes:

$$H(D) \equiv p_{\oplus} \log_2 \frac{1}{p_{\oplus}} + p_{\ominus} \log_2 \frac{1}{p_{\ominus}}$$

So we can use it to measure purity!

Example

Suppose I am trying to predict output y and I have input x, e.g.:

x =HasKids	y =OwnsDoraVideo
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
No	No
No	No
Yes	No
Yes	No

- From the table, we can estimate $P(y = YES) = 0.5 = P(y = NO)$.
- Thus, we estimate $H(y) = 0.5 \log \frac{1}{0.5} + 0.5 \log \frac{1}{0.5} = 1$.

Example: Conditional entropy

x =HasKids	y =OwnsDoraVideo
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
No	No
No	No
Yes	No
Yes	No

Specific conditional entropy is the uncertainty in y given a particular x value. E.g.,

- $P(y = YES|x = YES) = \frac{2}{3}$, $P(y = NO|x = YES) = \frac{1}{3}$
- $H(y|x = YES) = \frac{2}{3} \log \frac{1}{(\frac{2}{3})} + \frac{1}{3} \log \frac{1}{(\frac{1}{3})} \approx 0.9183.$

Conditional entropy

- *The average conditional entropy, $E_X[H(y|X)]$*, is the average specific conditional entropy of y given the values for x:

$$H(y|x) = \sum_v P(x = v) H(y|x = v)$$

- E.g. (from previous slide),
  - $H(y|x = YES) = \frac{2}{3} \log \frac{1}{(\frac{2}{3})} + \frac{1}{3} \log \frac{1}{(\frac{1}{3})} \approx 0.9183$
  - $H(y|x = NO) = 0 \log \frac{1}{0} + 1 \log \frac{1}{1} = 0.$

$$E_X[H(y|X)] =$$
$$H(y|x = YES)P(x = YES) + H(y|x = NO)P(x = NO)$$
$$= 0.9183 * \frac{3}{4} + 0 * \frac{1}{4} \approx 0.6887$$

- Interpretation: the expected number of bits needed to transmit y if both the emitter and the receiver know the possible values of x (but before they are told x's specific value).

Information gain

- Think of X as a binary variable derived from a feature, that is, a **test**.
- Suppose one has to transmit y. How many bits on the average would it save if both the transmitter and the receiver knew x?

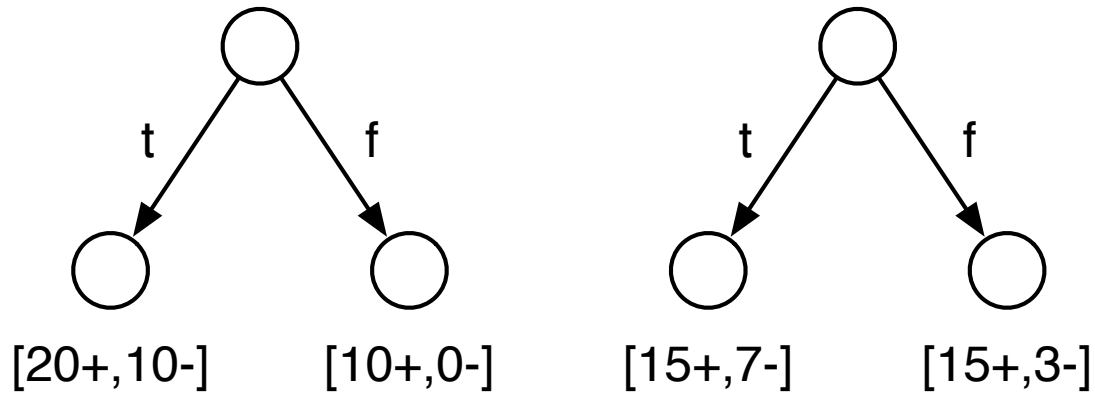
$$IG(y|X) = H(y) - E_X[H(y|X)]$$

  This is called *information gain*
- Alternative interpretation: what reduction in entropy would be obtained, on average, by knowing the value of X

Information gain to determine best test

- We choose, recursively at each interior node, the **test** that has highest information gain.
  Equivalently, test that results in lowest average conditional entropy.
- If tests are binary:

$$\begin{aligned} IG(D, \text{Test}) &= H(D) - E_{Test}[H(D|\text{Test})] \\ &= H(D) - \frac{|D_{\text{Test}}|}{|D|} H(D_{\text{Test}}) - \frac{|D_{\neg\text{Test}}|}{|D|} H(D_{\neg\text{Test}}) \end{aligned}$$



Check that in this case, Test1 wins.

Caveats on tests with multiple values

- If the outcome of a test is *multi-valued*, the number of possible values influences the information gain
- The more possible values, the higher the gain! (the more likely it is to form small, but pure partitions)
- C4.5/J48 (the most popular decision tree construction algorithm in ML community) uses only binary tests:
  - Attribute=Value for discrete attributes
  - Attribute $<$ or $>$ Value for continuous attributes
- Other approaches consider smarter metrics (e.g. gain ratio), which account for the number of possible outcomes

Alternative purity measures

- For classification, an alternative entropy is the *Gini index*:

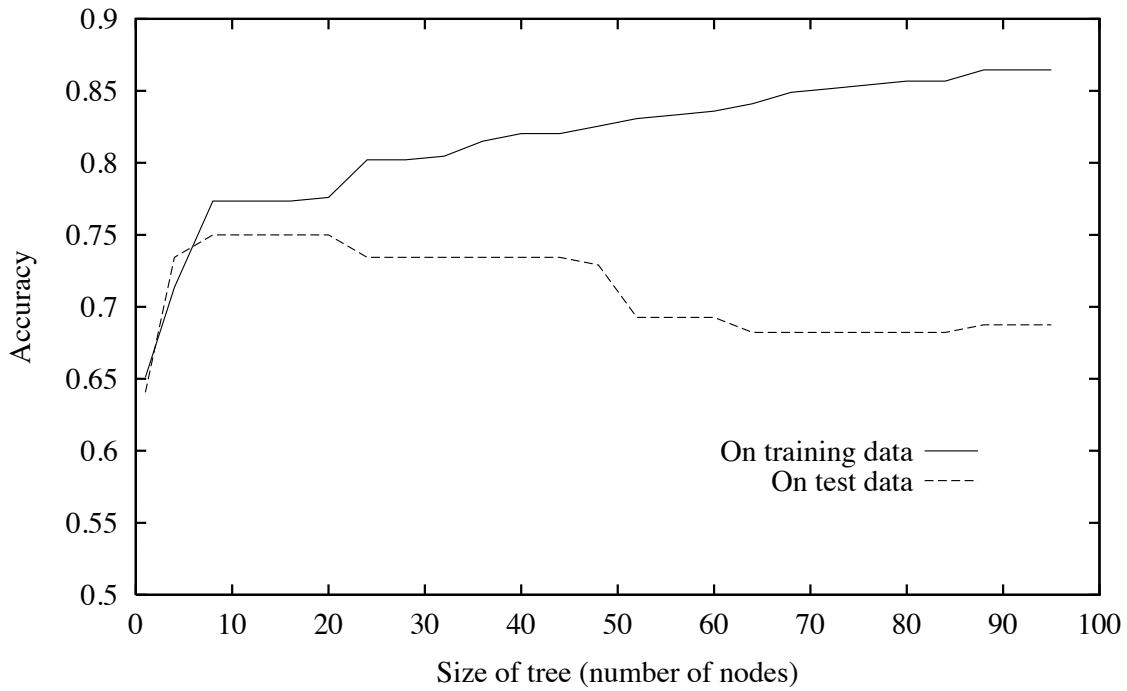
$$\sum_y P(y)(1 - P(y)) = 1 - \sum_y (P(y))^2$$

  Same qualitative behavior as the entropy (it is a 2nd-order approximation) but not the same interpretation

- For regression trees, purity is measured by the average mean-squared error at each leaf
  E.g. CART (Breiman et al., 1984) [HTF 9.2]

Overfitting in decision trees

- Typically, classification tree construction proceeds until all leaves are pure – all examples having the same y value.
- As the tree grows, the generalization performance starts to degrade, because the algorithm is finding *irrelevant attributes / tests*.



Example from (Mitchell, 1997)

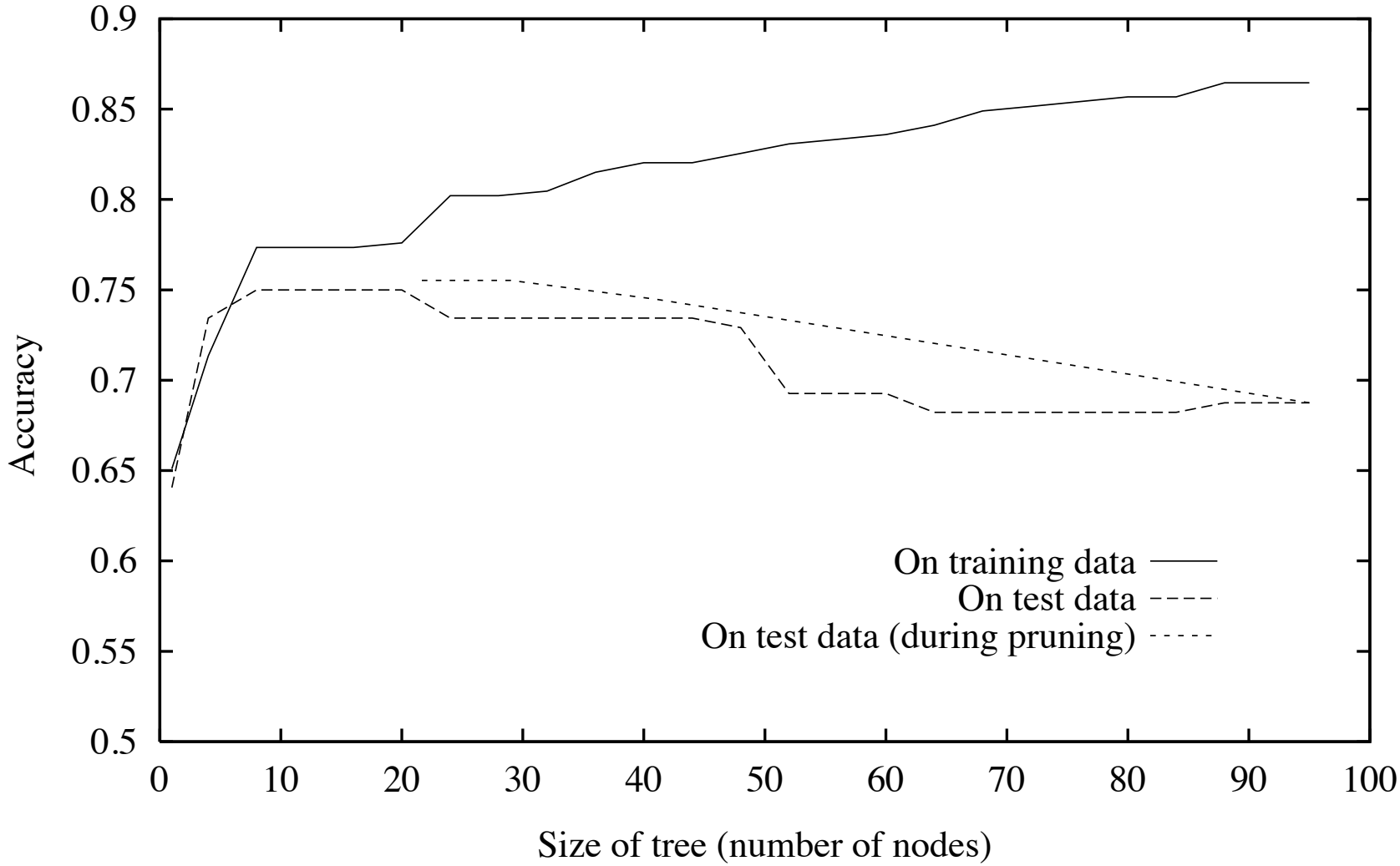
Avoiding overfitting

- Two approaches:
  1. Stop growing the tree when further splitting the data does not yield a statistically significant improvement
  2. Grow a full tree, then *prune* the tree, by eliminating nodes
- The second approach has been more successful in practice, because in the first case it might be hard to decide if the information gain is sufficient or not (e.g. for multivariate functions)
- We will select the best tree, for now, by measuring performance on a separate validation data set.

Example: Reduced-error pruning

1. Split the "training data" into a training set and a validation set
2. Grow a large tree (e.g. until each leaf is pure)
3. For each node:
   1. Evaluate the validation set accuracy of pruning the subtree rooted at the node
   2. Greedily remove the node that most improves validation set accuracy, with its corresponding subtree
   3. Replace the removed node by a leaf with the majority class of the corresponding examples.
4. Stop when pruning starts hurting the accuracy on the validation set.

Example: Effect of reduced-error pruning



Decision and regression tree summary

- Very fast learning algorithms (e.g. C4.5, CART)
- Attributes may be discrete or continuous, no preprocessing needed
- Provide a general representation of classification rules
- Easy to understand! Though...
  - Exact tree output may be sensitive to small changes in data
  - With many features, tests may not be meaningful
- In standard form, good for (nonlinear) piecewise axis-orthogonal decision boundaries – not good with smooth, curvilinear boundaries
- In regression, the function obtained is discontinuous, which may not be desirable
- Good accuracy in practice – many applications!
  - Equipment/medical diagnosis
  - Scene analysis and image segmentation

Decision Trees: Modern adaptations

- **Bagging trees:**
  1. Sample, with replacement, dataset of size n
  2. Build a tree
  3. Repeat until you have B trees, use voting to classify
- **Random forests:**
  1. Sample, with replacement, dataset of size n
  2. Choose a random subset of $\sqrt{p}$ features
  3. Build a tree using only these features
  4. Repeat until you have B trees, use voting to classify
- **Extremely randomized trees**
  1. Build a tree to purity, selecting features and splits uniformly randomly
  2. Repeat until you have B trees, use voting to classify