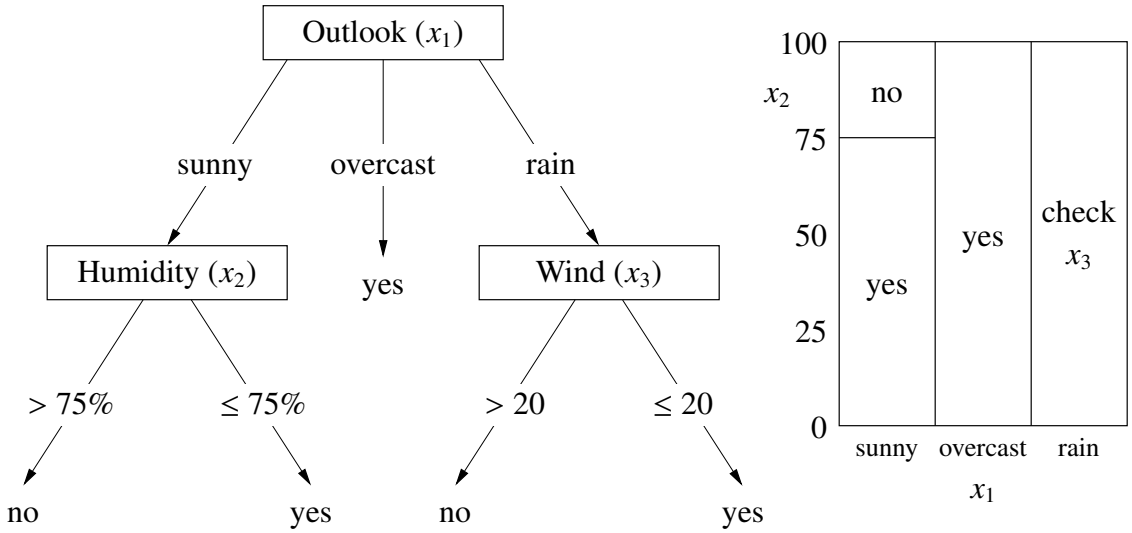
14 Decision Trees

DECISION TREES

Nonlinear method for classification and regression.

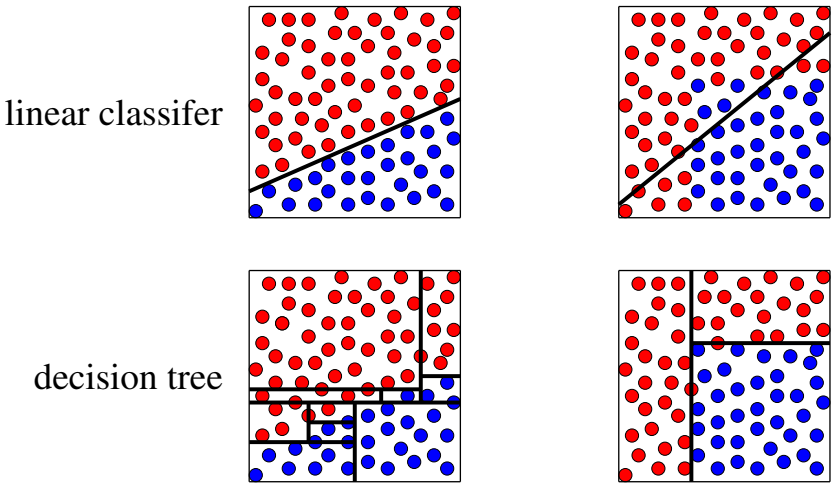
Uses tree with 2 node types:

- internal nodes test feature values (usually just one) & branch accordingly
- leaf nodes specify class $h(x)$



[Draw this by hand. dectree.pdf Deciding whether to go out for a picnic.]

- Cuts x-space into rectangular cells
- Works well with both categorical and quantitative features
- Interpretable result (inference)
- Decision boundary can be arbitrarily complicated



treelinearcompare2.pdf (redrawing of ISL, Figure 8.7) [Comparison of linear classifiers vs. decision trees on 2 examples.]

Consider classification first. Greedy, top-down learning heuristic:
[This algorithm is more or less obvious, and has been rediscovered many times. It's naturally recursive. I'll show how it works for classification first; later I'll talk about how it works for regression.]

Let X be $n \times d$ design matrix; $y \in \mathbb{R}^n$ be labels.
Let $S \subseteq \{1, 2, \ldots, n\}$ be set of sample point indices.
Top-level call: $S = \{1, 2, \ldots, n\}$.

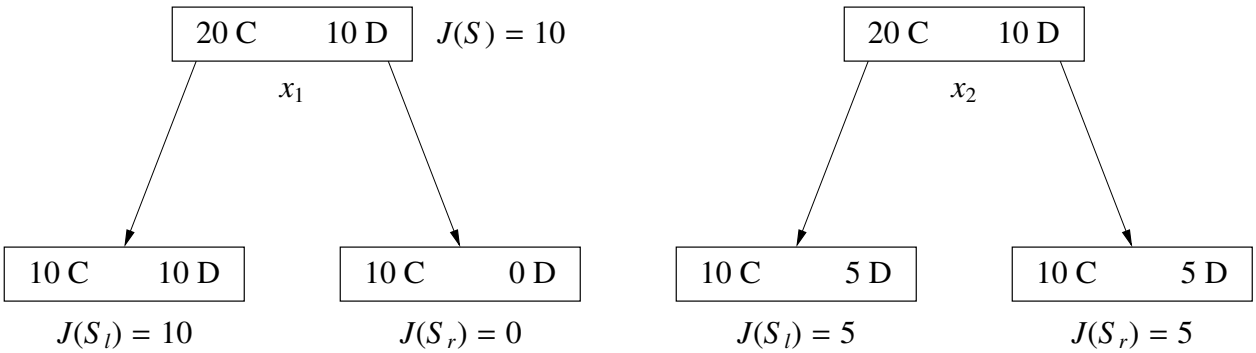
```
GrowTree(S)
    if (y_i = C for all i ∈ S and some class C) then {
        return new leaf(C)                    [We say the leaves are pure]
    } else {
        choose best splitting feature j and splitting value β     (*)
        S_l = {i ∈ S : X_ij < β}              [Or you could use ≤ and >]
        S_r = {i ∈ S : X_ij ≥ β}
        return new node(j, β, GrowTree(S_l), GrowTree(S_r))
    }
```

(*) How to choose best split?
- Try all splits. [All features, and all splits within a feature.]
- For a set S, let $J(S)$ be the cost of S.
- Choose the split that minimizes $J(S_l) + J(S_r)$; or,
  the split that minimizes weighted average $\dfrac{|S_l|J(S_l) + |S_r|J(S_r)}{|S_l| + |S_r|}$.

[Here, I'm using the vertical bars $|\cdot|$ to denote set cardinality.]

How to choose cost $J(S)$?
[I'm going to start by suggesting a mediocre cost function, so you can see why it's mediocre.]

Idea 1 (bad): Label S with the class C that labels the most points in S.
$J(S) \leftarrow$ # of points in S not in class C.



[Draw this by hand. badcost.pdf]

Problem: $J(S_l) + J(S_r) = 10$ for both splits, but left split is much better. Weighted avg prefers right split!

[There are many different splits that all have the same total cost. We want a cost function that better distinguishes between them.]

Idea 2 (good): Measure the entropy. [An idea from information theory.]
Let Y be a random class variable, and suppose $P(Y = \mathrm{C}) = p_\mathrm{C}$.
The surprise of Y being class C is $-\log_2 p_\mathrm{C}$. [Always nonnegative.]

- event w/prob. 1 gives us zero surprise.
- event w/prob. 0 gives us infinite surprise!

[In information theory, the surprise is equal to the expected number of bits of information we need to transmit which events happened to a recipient who knows the probabilities of the events. Often this means using fractional bits, which may sound crazy, but it makes sense when you're compiling lots of events into a single message; e.g., a sequence of biased coin flips.]

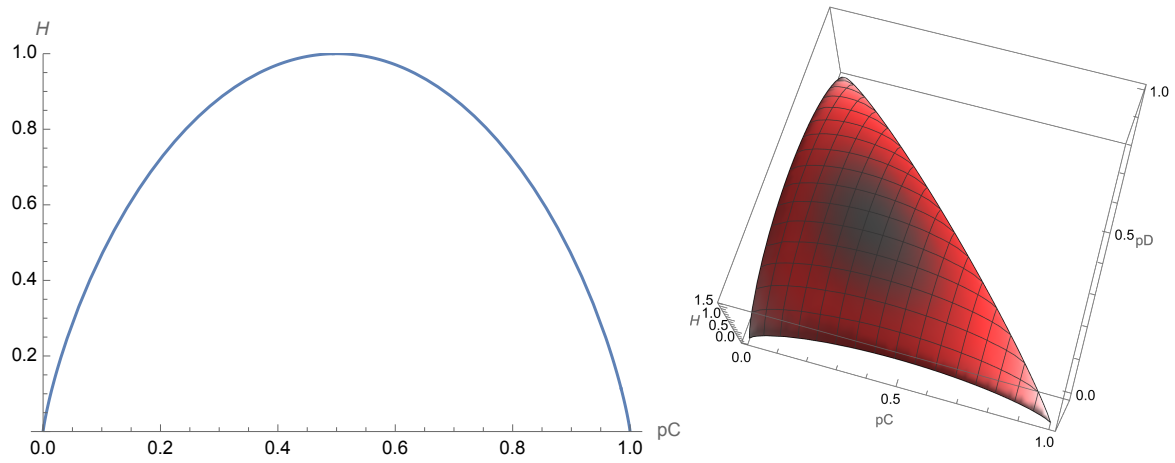
The entropy of an index set S is the average surprise [when you draw a point at random from S],

$$H(S) = -\sum_\mathrm{C} p_\mathrm{C} \log_2 p_\mathrm{C}, \qquad \text{where } p_\mathrm{C} = \frac{|\{i \in S : y_i = \mathrm{C}\}|}{|S|}.$$

[The proportion of points in S that are in class C.]

If all points in S belong to same class? $H(S) = -1 \log_2 1 = 0$.
Half class C, half class D? $H(S) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$.
n points, all different classes? $H(S) = -\log_2 \frac{1}{n} = \log_2 n$.

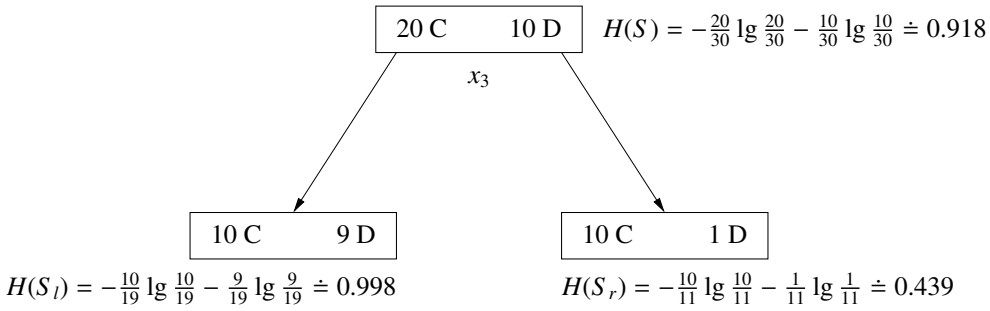
[The entropy is the expected number of bits of information we need to transmit to identify the class of a sample point in S chosen uniformly at random. It makes sense that it takes 1 bit to specify C or D when each class is equally likely. And it makes sense that it takes $\log_2 n$ bits to specify one of n classes when each class is equally likely.]



entropy.pdf [Left: plot of the entropy $H(p_\mathrm{C})$ when there are only two classes. The probability of the second class is $p_\mathrm{D} = 1 - p_\mathrm{C}$, so we can plot the entropy with just one dependent variable. Right: plot of the entropy $H(p_\mathrm{C}, p_\mathrm{D})$ when there are three classes. The probability of the third class is $p_\mathrm{E} = 1 - p_\mathrm{C} - p_\mathrm{D}$. Observe that the entropy is strictly concave.]

Weighted avg entropy after split is $H_{\text{after}} = \dfrac{|S_l|\,H(S_l) + |S_r|\,H(S_r)}{|S_l| + |S_r|}$.

Choose split that maximizes information gain $H(S) - H_{\text{after}}$. [Which is just the same as minimizing H_{after}.]

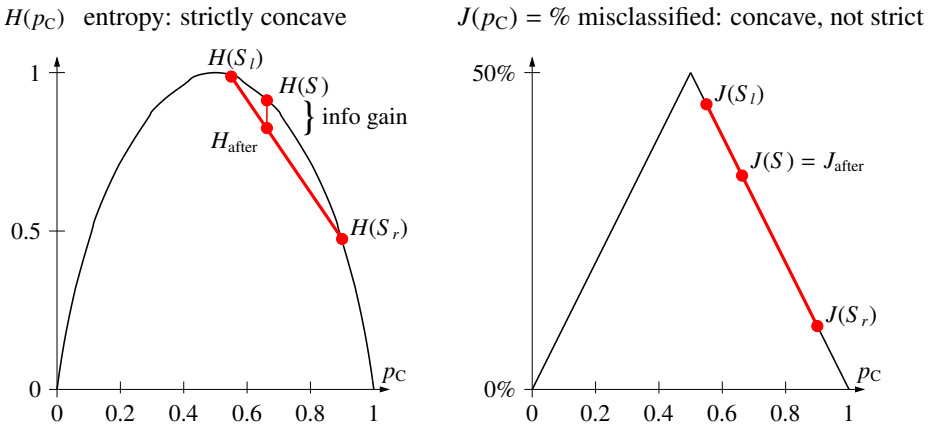


$H_{\text{after}} = 0.793$ info gain $= 0.125$

[Draw this by hand. infogain.pdf]

Info gain always positive *except* it is zero when one child is empty or
for all C, $P(y_i = \text{C} | i \in S_l) = P(y_i = \text{C} | i \in S_r)$. [Which is the case for the second split we considered.]

[Recall the graph of the entropy.]



[Draw this by hand on entropy.pdf. concave.pdf]

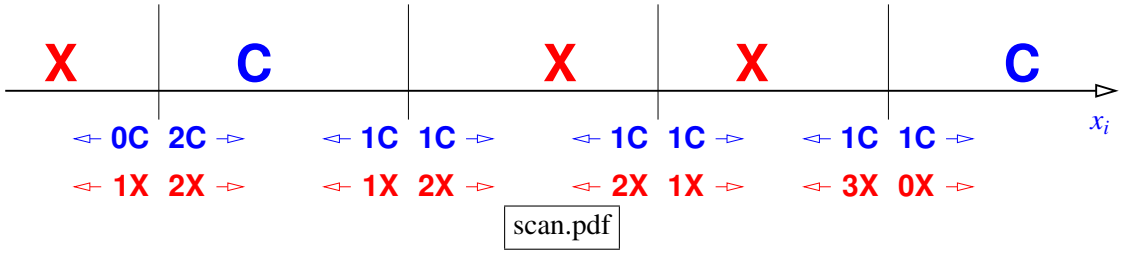
[Suppose we pick two points on the entropy curve, then draw a line segment connecting them. Because the entropy curve is strictly concave, the interior of the line segment is strictly below the curve. Any point on that segment represents a weighted average of the two entropies for suitable weights. If you unite the two sets into one parent set, the parent set's value p_C is the weighted average of the children's p_C's. Therefore, the point directly above that point on the curve represents the parent's entropy. The information gain is the vertical distance between them. So the information gain is positive unless the two child sets both have exactly the same p_C and lie at the same point on the curve.]

[On the other hand, for the graph on the right, plotting the % misclassified, if we draw a line segment connecting two points on the curve, the segment might lie entirely on the curve. In that case, uniting the two child sets into one, or splitting the parent set into two, changes neither the total misclassified sample points nor the weighted average of the % misclassified. The bigger problem, though, is that many different splits will get the same weighted average cost; this test doesn't distinguish the quality of different splits well.]

[By the way, the entropy is not the only function that works well. Many concave functions work fine, including the simple polynomial $p(1-p)$.]

More on choosing a split:

- For binary feature x_i: children are $x_i = 0$ & $x_i = 1$.
- If x_i has 3+ discrete values: split depends on application.
  [Sometimes it makes sense to use multiway splits; sometimes binary splits.]
- If x_i is quantitative: sort x_i values in S; try splitting between each pair of *unequal* consecutive values.
  [We can radix sort the points in linear time, and if n is huge we should.]
  Clever bit: As you scan sorted list from left to right, you can update entropy in $O(1)$ time per point![5]
  [This is important for obtaining a fast tree-building time.]
  [Draw a row of C's and X's; show how we update the # of C's and # of X's in each of S_l and S_r as we scan from left to right.]



scan.pdf

Algs & running times:

- Classify test point: Walk down tree until leaf. Return its label.
  Worst-case time is O(tree depth).
  For binary features, that's $\leq d$. [Quantitative features may go deeper.]
  Usually (not always) $\leq O(\log n)$.
- Training: For binary features, try $O(d)$ splits at each node.
  For quantitative features, try $O(n'd)$ splits; n' = points in node
  Either way $\Rightarrow O(n'd)$ time at this node
  [Training on quantitative features is asymptotically just as fast as training on binary features because of our clever way of computing the entropy for each split.]
  Each point participates in O(depth) nodes, costs $O(d)$ time in each node.
  [This is an amortized analysis: we are charging $O(d$ depth) time to each sample point.]
  Running time $\leq O(nd$ depth).
  [As nd is the size of the design matrix X, and the depth is often logarithmic, this is a surprisingly reasonable running time.]

[5] Let C be the number of class C sample points to the left of a potential split and c be the number to the right of the split. Let D be the number of class not-C points to the left of the split and d be the number to the right of the split. Update C, c, D, and d at each split (in $O(1)$ time per split) as you move from left to right. At each potential split, calculate the entropy of the left set as $-\frac{C}{C+D}\log_2\frac{C}{C+D} - \frac{D}{C+D}\log_2\frac{D}{C+D}$ and the entropy of the right set as $-\frac{c}{c+d}\log_2\frac{c}{c+d} - \frac{d}{c+d}\log_2\frac{d}{c+d}$. Note: $\log 0$ is undefined, but this formula works if we use the convention $0 \log 0 = 0$.

It follows that the weighted average of the two entropies is $-\frac{1}{n'}\left(C\log_2\frac{C}{C+D} + D\log_2\frac{D}{C+D} + c\log_2\frac{c}{c+d} + d\log_2\frac{d}{c+d}\right)$, where $n' = C + D + c + d$ is the total number of sample points stored in this treenode. Choose the split that minimizes this weighted average.