

# Decision Trees

## Duke Course Notes

### Cynthia Rudin

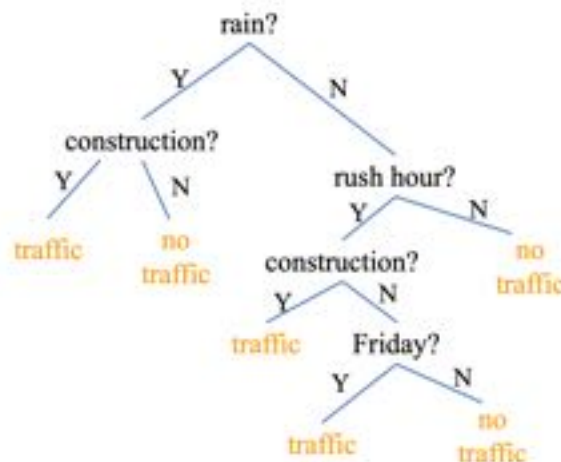
Credit: Russell & Norvig, Mitchell, Kohavi & Quinlan, Carter, Vanden Berghen

Why trees?

- interpretable/intuitive, popular in medical applications because they mimic the way a doctor thinks
- model discrete outcomes nicely
- can be very powerful, can be as complex as you need them
- C4.5 and CART - from “top 10” - decision trees are very popular

Some real examples (from Russell & Norvig, Mitchell)

- BP’s GasOIL system for separating gas and oil on offshore platforms - decision trees replaced a hand-designed rules system with 2500 rules. C4.5-based system outperformed human experts and saved BP millions. (1986)
- Learning to fly a Cessna on a flight simulator by watching human experts fly the simulator (1992)
- Can also learn to play tennis, analyze C-section risk, etc.



How to build a decision tree:

- Start at the top of the tree.
- Grow it by “splitting” attributes one by one. To determine which attribute to split, look at “node impurity.”
- Assign leaf nodes the majority vote in the leaf.
- When we get to the bottom, prune the tree to prevent overfitting

Why is this a good way to build a tree? Maybe it’s not...

I have to warn you that C4.5 and CART are not elegant by any means that I can define elegant. But the resulting trees can be very elegant. Plus there are 2 of the top 10 algorithms in data mining that are decision tree algorithms! So it’s worth it for us to know what’s under the hood... even though, well... let’s just say it ain’t pretty.

Example: Will the customer wait for a table? (adapted from Russell & Norvig)

Here are the attributes:

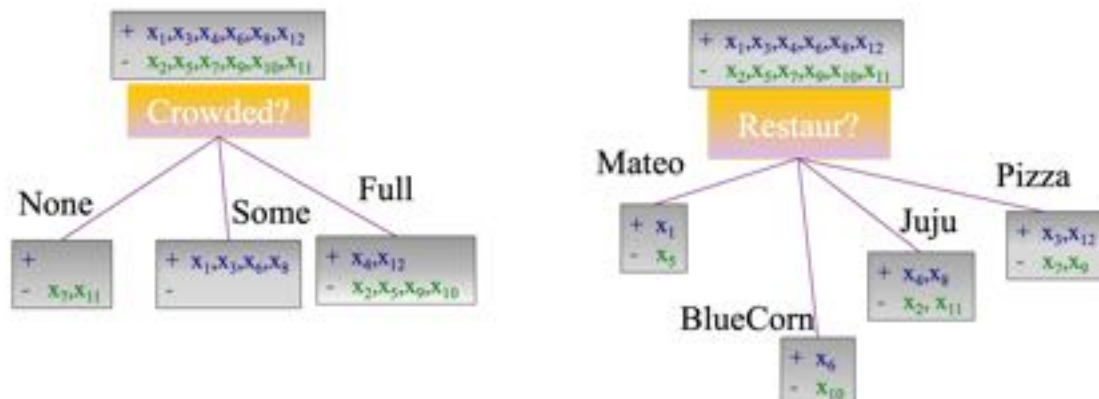
- Will the customer wait for a table at a restaurant?
- OthOptions: Other options, True if there are restaurants nearby.
- Weekend: This is true if it is Friday, Saturday or Sunday.
- WaitArea: Does it have a bar or other nice waiting area to wait in?
- Plans: Does the customer have plans just after dinner?
- Price: This is either \$, \$\$, \$\$\$, or \$\$\$\$
- Precip: Is it raining or snowing?
- Restaur:
  - Mateo (fancy),
  - Juju (nice),
  - Blue Corn Mexican Cafe (casual),
  - Pompieri Pizza (very casual)

- Wait: Wait time estimate: 0-5 min, 6-15 min, 16-30 min, or 30+
- Crowded: Whether there are other customers (no, some, or full)

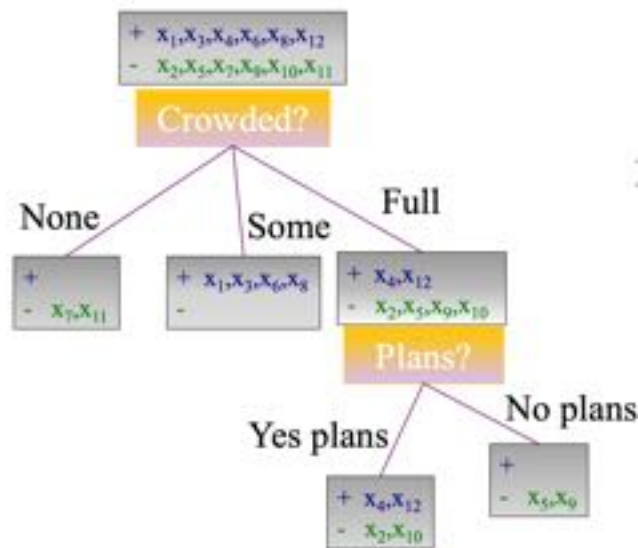
Here is the whole dataset. The label is on the right.

	OthOptions	Weekend	WaitArea	Plans	Price	Precip	Restaur	Wait	Crowded	Stay?
$x_1$	Yes	No	No	Yes	\$\$\$	No	Mateo	0-5	some	Yes
$x_2$	Yes	No	No	Yes	\$	No	Juju	16-30	full	No
$x_3$	No	No	Yes	No	\$	No	Pizza	0-5	some	Yes
$x_4$	Yes	Yes	No	Yes	\$	No	Juju	6-15	full	Yes
$x_5$	Yes	Yes	No	No	\$\$\$	No	Mateo	30+	full	No
$x_6$	No	No	Yes	Yes	\$\$	Yes	BlueCorn	0-5	some	Yes
$x_7$	No	No	Yes	No	\$	Yes	Pizza	0-5	none	No
$x_8$	No	No	No	Yes	\$\$	Yes	Juju	0-5	some	Yes
$x_9$	No	Yes	Yes	No	\$	Yes	Pizza	30+	full	No
$x_{10}$	Yes	Yes	Yes	Yes	\$\$\$	No	BlueCorn	6-15	full	No
$x_{11}$	No	No	No	No	\$	No	Juju	0-5	none	No
$x_{12}$	Yes	Yes	Yes	Yes	\$	No	Pizza	16-30	full	Yes

Here are two options for the first feature to split at the top of the tree. Which one should we choose? Which one gives me the most information?



The answer is obvious. The split on Restaur gave us no information about the label at all! The split on Crowded gives us a lot of information about the label. If we pick Crowded, we might then split afterwards on Plans, because that has some more information about the label.



In order to determine which split possibility is the best, we need is a formula to compute “information.”

We’ll build up to the derivation of **C4.5**. Origins: Hunt 1962, ID3 of Quinlan 1979 (600 lines of Pascal), C4 (Quinlan 1987). C4.5 is 9000 lines of C (Quinlan 1993). We start with some basic information theory.

### Information Theory (from slides of Tom Carter, June 2011)

“Information” from observing the occurrence of an event

$:=$  #bits needed to encode the probability of the event  $p = -\log_2 p$ .

E.g., a coin flip from a fair coin contains 1 bit of information. If the event has probability 1, we get no information from the occurrence of the event. If the event has a very low probability, we get a lot of information when we see it, and we’re surprised. That will give us a lot of information.

Where did this definition of information come from? Turns out it’s pretty cool. We want to define  $I$  so that it obeys all these things:

- $I(p) \geq 0, I(1) = 0$ ; the information of any event is non-negative, no information from events with prob 1
- $I(p_1 \cdot p_2) = I(p_1) + I(p_2)$ ; the information from two independent events should be the sum of their informations

- $I(p)$  is continuous, slight changes in probability correspond to slight changes in information

Together these lead to:

$$I(p^2) = 2I(p) \text{ or generally } I(p^n) = nI(p),$$

this means that

$$I(p) = I\left((p^{1/m})^m\right) = mI\left(p^{1/m}\right) \text{ so } \frac{1}{m}I(p) = I\left(p^{1/m}\right)$$

and more generally,

$$I\left(p^{n/m}\right) = \frac{n}{m}I(p).$$

This is true for any fraction  $n/m$ , which includes rationals, so just define it for all positive reals:

$$I(p^a) = aI(p).$$

The functions that do this are  $I(p) = -\log_b(p)$  for some  $b$ . Choose  $b = 2$  for “bits.”

---

Flipping a fair coin gives  $-\log_2(1/2) = 1$  bit of information if it comes up either heads or tails.

A biased coin landing on heads with  $p = .99$  gives  $-\log_2(.99) = .0145$  bits of information.

A biased coin landing on heads with  $p = .01$  gives  $-\log_2(.01) = 6.643$  bits of information.

---

Entropy. Say one of many of possible events could occur. What’s the mean information of those events? Assume the events  $v_1, \dots, v_J$  occur with probabilities  $p_1, \dots, p_J$ , where  $[p_1, \dots, p_J]$  is a discrete probability distribution.

$$\mathbf{E}_{p \sim [p_1, \dots, p_J]} I(p) = \sum_{j=1}^J p_j I(p_j) = - \sum_j p_j \log_2 p_j =: H(\mathbf{p})$$

where  $\mathbf{p}$  is the vector  $[p_1, \dots, p_J]$ .  $H(\mathbf{p})$  is called the **entropy** of discrete distribution  $\mathbf{p}$ .

So if there are only 2 events (binary), with probabilities  $\mathbf{p} = [p, 1 - p]$ ,

$$H(\mathbf{p}) = -p \log_2(p) - (1 - p) \log_2(1 - p).$$

If the probabilities were  $[1/2, 1/2]$ ,

$$H(\mathbf{p}) = -2 \frac{1}{2} \log_2 \frac{1}{2} = 1 \quad (\text{Yes, we knew that.})$$

Or if the probabilities were  $[0.99, 0.01]$ ,

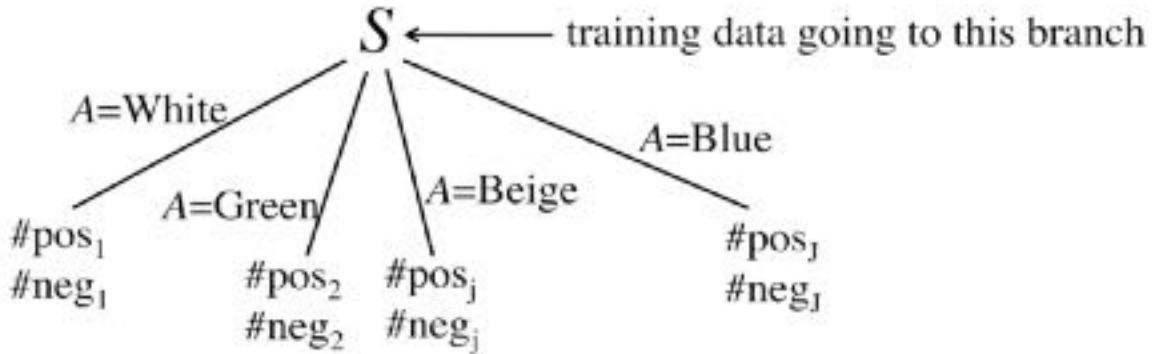
$$H(\mathbf{p}) = 0.08 \text{ bits.}$$

As one of the probabilities in the vector  $\mathbf{p}$  goes to 1,  $H(\mathbf{p}) \rightarrow 0$ , which is what we want.

Back to C4.5, which uses Information Gain as the splitting criteria.

**Back to C4.5** (source material: Russell & Norvig, Mitchell, Quinlan)

We consider a “test” split on attribute  $A$  at a branch.



In  $S$  we have  $\#pos$  positives and  $\#neg$  negatives. For each branch  $j$ , we have  $\#pos_j$  positives and  $\#neg_j$  negatives.

The training probabilities in branch  $j$  are:

$$\left[ \frac{\#pos_j}{\#pos_j + \#neg_j}, \frac{\#neg_j}{\#pos_j + \#neg_j} \right].$$

The Information Gain is calculated like this:

$$\begin{aligned}
\text{Gain}(S, A) &= \text{expected reduction in entropy due to branching on attribute } A \\
&= \text{original entropy} - \text{entropy after branching} \\
&= H\left(\left[\frac{\#\text{pos}}{\#\text{pos} + \#\text{neg}}, \frac{\#\text{neg}}{\#\text{pos} + \#\text{neg}}\right]\right) \\
&\quad - \sum_{j=1}^J \frac{\#\text{pos}_j + \#\text{neg}_j}{\#\text{pos} + \#\text{neg}} H\left[\frac{\#\text{pos}_j}{\#\text{pos}_j + \#\text{neg}_j}, \frac{\#\text{neg}_j}{\#\text{pos}_j + \#\text{neg}_j}\right].
\end{aligned}$$

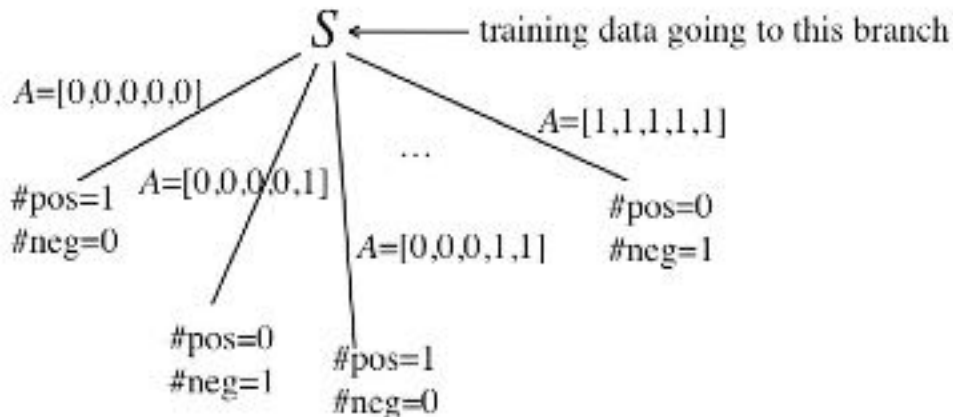
Back to the example with the restaurants.

$$\begin{aligned}
\text{Gain}(S, \text{Crowded}) &= H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) - \left[\frac{2}{12}H([0, 1]) + \frac{4}{12}H([1, 0]) + \frac{6}{12}H\left(\left[\frac{2}{6}, \frac{4}{6}\right]\right)\right] \\
&\approx 0.541 \text{ bits.}
\end{aligned}$$

$$\begin{aligned}
\text{Gain}(S, \text{Restaur}) &= 1 - \left[\frac{2}{12}H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) + \frac{2}{12}H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) \right. \\
&\quad \left. + \frac{4}{12}H\left(\left[\frac{2}{4}, \frac{2}{4}\right]\right) + \frac{4}{12}H\left(\left[\frac{2}{4}, \frac{2}{4}\right]\right)\right] \approx 0 \text{ bits.}
\end{aligned}$$

Actually Crowded has the highest gain among the attributes, and is chosen to be the root of the tree. In general, we want to choose the feature  $A$  that maximizes  $\text{Gain}(S, A)$ .

One problem with Gain is that it likes to partition too much, and favors numerous splits: e.g., if each branch contains 1 example:



Then,

$$H \left[ \frac{\#pos_j}{\#pos_j + \#neg_j}, \frac{\#neg_j}{\#pos_j + \#neg_j} \right] = 0 \text{ for all } j,$$

so all the terms for the entropy after branching would be zero and we'd choose that attribute over all the others.

An alternative to Gain is the *Gain Ratio*. We want to have a large Gain, but also we want a small number of partitions. We'll choose our attribute according to:

$$\begin{aligned} \text{Gain}(S, A) &\leftarrow \text{want large} \\ \text{SplitInfo}(S, A) &\leftarrow \text{want small} \end{aligned}$$

where  $\text{SplitInfo}(S, A)$  comes from the partition:

$$\text{SplitInfo}(S, A) = - \sum_{j=1}^J \frac{|S_j|}{|S|} \log \left( \frac{|S_j|}{|S|} \right)$$

where  $|S_j|$  is the number of examples in branch  $j$ . We want each term in the sum to be large. That means we want  $\frac{|S_j|}{|S|}$  to be large, which means we want  $|S_j|$  to be large, meaning that we want lots of examples in each branch.

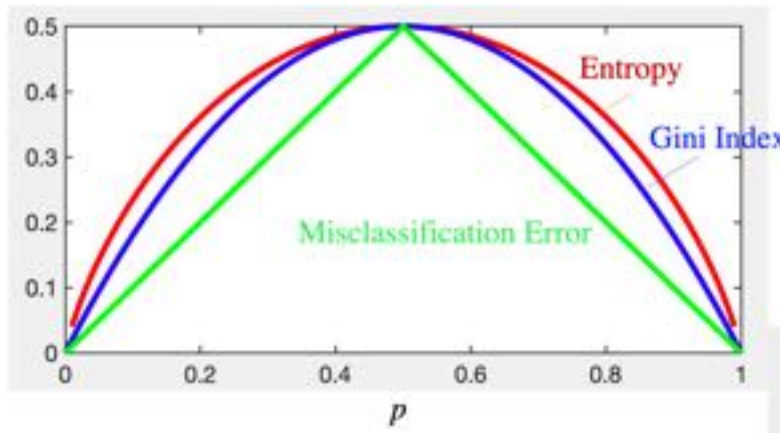
Keep splitting until:

- all examples have the same class
- no more attributes to split (we used them all earlier in the tree directly above the leaf we're working on)

Before I move on, I should talk about splitting a little bit more. There are possibilities to replace  $H([p, 1-p])$ ; it is not the only option. Here are some others.

- Gini index  $2p(1-p)$  used by CART.
- Misclassification error  $1 - \max(p, 1-p)$ . (Say an event has prob  $p$  of success. Using majority vote, we classify the event to happen when  $p > 1/2$  and classify the event not to happen when  $p \leq 1/2$ . This value is thus the proportion of time we guess incorrectly.)





You can see that often it will split in a similar way using any of these criteria.

## Pruning

Let's start with C4.5's pruning. C4.5 recursively makes choices as to whether to prune on an attribute:

- Option 1: leaving the tree as is
- Option 2: collapse that part of the tree into a leaf. The leaf has the most frequent label in the data  $S$  going to that part of the tree.
- Option 3: replace that part of the tree with one of its subtrees, corresponding to the most common branch in the split

To figure out which decision to make, C4.5 computes upper bounds on the probability of error for each option.

- Prob of error for Option 1  $\leq \text{UpperBound}_1$
- Prob of error for Option 2  $\leq \text{UpperBound}_2$
- Prob of error for Option 3  $\leq \text{UpperBound}_3$

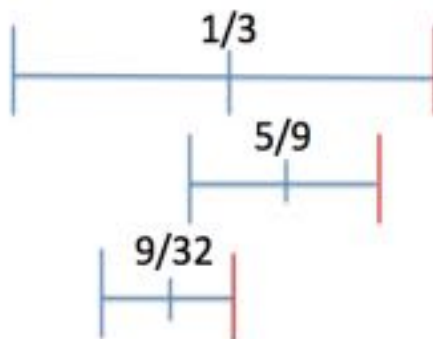
C4.5 chooses the option that has the lowest of these three upper bounds. This ensures that (w.h.p.) the error rate is fairly low.

E.g., which has the smallest upper bound:

- 1 incorrect out of 3

- 5 incorrect out of 9, or
- 9 incorrect out of 32?

Which of these could be the safest choice to reduce the misclassification rate?



To calculate the upper bounds, calculate confidence intervals on proportions. We can calculate this numerically without a problem.

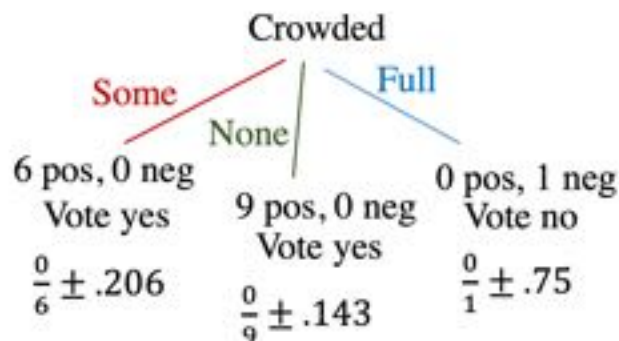
These confidence intervals use the binomial distribution. If you give me  $\alpha$ ,  $M$  and  $N$ , I can give you  $p_\alpha$ . C4.5 uses  $\alpha = .25$  by default.  $M$  for a given branch is how many misclassified examples are in the branch.  $N$  for a given branch is just the number of examples in the branch,  $|S_j|$ .

So we can calculate the upper bound on a branch, but it's still not clear how to calculate the upper bound on a subtree. Actually, we calculate an upper confidence bound on each branch on the subtree and average it over the relative frequencies of landing in each branch of the tree. It's best explained by example:

Let's consider a new restaurant dataset of 16 examples (adapted from the Kranf Site).

	Crowded	Price	Stay?
$x_1$	Some	\$\$	Yes
$x_2$	Some	\$\$\$	Yes
$x_3$	None	\$\$	Yes
$x_4$	Some	\$\$	Yes
$x_5$	None	\$\$	Yes
$x_6$	None	\$\$\$\$	Yes
$x_7$	None	\$\$	Yes
$x_8$	None	\$	Yes
$x_9$	Some	\$\$	Yes
$x_{10}$	None	\$\$	Yes
$x_{11}$	Some	\$	Yes
$x_{12}$	Full	\$\$	No
$x_{13}$	None	\$\$	Yes
$x_{14}$	None	\$	Yes
$x_{15}$	Some	\$\$\$	Yes
$x_{16}$	None	\$	Yes

Think of a split on Crowded.



Calculate the upper bound on the tree for Option 1: calculate  $p_{.25}$  for each branch, which are respectively .206, .143, and .75. You can check this easily by typing into matlab the commands `binocdf(0,6,.206)`, `binocdf(0,9,.143)`, and

$\text{binocdf}(0,1,.75)$ , and all of these values will be around .25.

The  $\text{binocdf}(M,N,p_\alpha)$  command tells you the cdf of the binomial distribution with parameters  $N$  (that's the number of trials, which here is the number of data points in the leaf) and  $p_\alpha$  (the probability of "success" for each trial, which in our case is the probability of misclassified examples in the leaf), at the value  $M$  (the number of misclassified points in the leaf). When we chose  $p_\alpha$  so that  $\alpha = .25$ , we chose the largest probability that is still "reasonable" (there's only a 25% chance the "true" probability is larger than this).

The average over the three leaves that comprise the tree is:

$$\text{Ave of the upper bounds for tree} = \frac{1}{16} (6 \cdot .206 + 9 \cdot .143 + 1 \cdot .75) = .204.$$

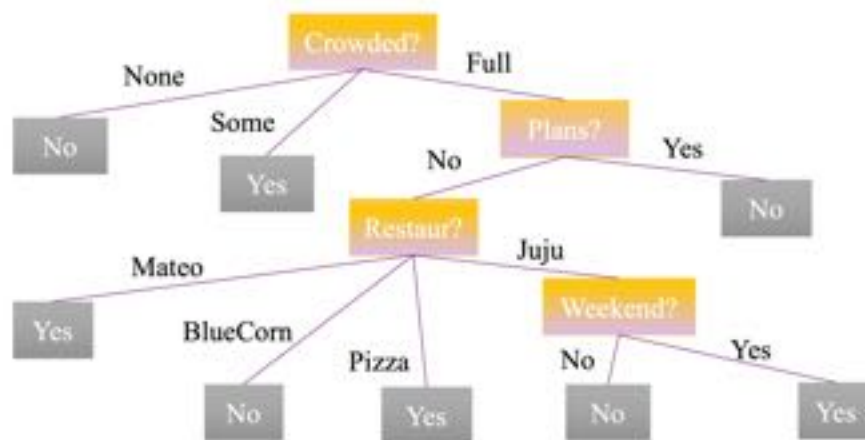
Calculate the upper bound on the tree for Option 2: where we'd collapse the tree to a leaf with  $6+9+1 = 16$  examples in it, where 15 are positive, and 1 is negative. Calculate  $p_\alpha$  that solves  $\alpha = \sum_{z=0}^1 \text{Bin}(z, 16, p_\alpha)$ , which is .157. The average is:

$$\text{Ave of the upper bounds for leaf} = \frac{1}{16} 16 \cdot .157 = .157.$$

Option 3 is the same as Option 2. (If a leaf had a subtree below it then Option 3 would have us keep that subtree, but it's just a leaf here.)

The upper bound on the error for Option 2 is lower, so we'll prune the tree to a leaf. Look at the data – does it make sense to do this? I think so. There's only one negative example, so best to collapse.

By the way, for the restaurant example, at the end we get this:




---

**CART** - Classification and Regression Trees (Breiman, Friedman, Olshen, Stone, 1984)

Does only binary splits, not multiway splits (less interpretable, but simplifies splitting criteria).

For splitting, CART uses the Gini index. The Gini index is

$$p(1 - p) = \frac{\text{variance of Bin}(n, p)}{n} = \text{variance of Bernoulli}(p).$$

For pruning, CART uses “minimal cost complexity.”

Each subtree is assigned a cost.

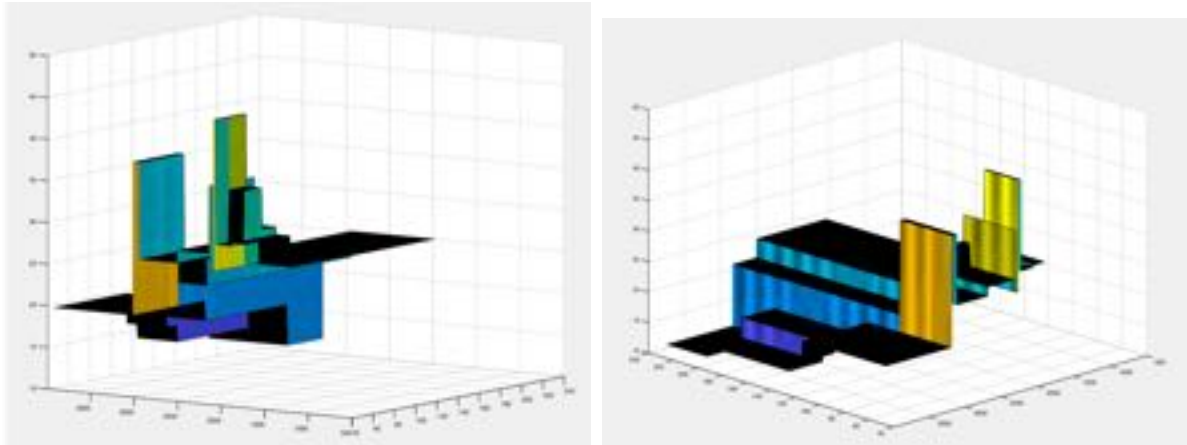
$$\text{cost}(\text{subtree}) = \sum_{\text{leaves } j} \sum_{x_i \in \text{leaf } j} \mathbf{1}_{[y_i \neq \text{leaf's class}]} + C [\#\text{leaves in subtree}].$$

It eliminates the subtree if it doesn't reduce the error rate enough relative to the number of leaves. We could create a sequence of nested subtrees by gradually increasing  $C$ . We can use a validation set or full-blown nested cross-validation to choose  $C$ .

This cost function is actually what decision trees should be optimizing all along. However, computers in the 1980's weren't so good, and even now it is challenging to optimize over the set of decision trees. However, we can optimize this

function over the set of decision trees for datasets that are not huge. This what the <https://arxiv.org/abs/2006.08690> (pronounced “ghost”) algorithm does (Lin, Chong et al., 2020).

**CART’s Regression Trees** Here are some regression trees for a 2D dataset on continuous variables. They are piecewise constant predictive models.



CART decides which attributes to split and where to split them. In each leaf, we’re going to assign  $f(x)$  to be a constant.

Can you guess what value to assign?

Consider the empirical error, using the least squares loss:

$$R^{\text{train}}(f) = \sum_i (y_i - f(x_i))^2$$

Break it up by leaves. Call the value of  $f(x)$  in leaf  $j$  by  $f_j$  since it’s a constant.

$$\begin{aligned} R^{\text{train}}(f) &= \sum_{\text{leaves } j} \sum_{i \in \text{leaf } j} (y_i - f(x_i))^2 \\ &= \sum_{\text{leaves } j} \sum_{i \in \text{leaf } j} (y_i - f_j)^2 =: \sum_{\text{leaves } j} R_j^{\text{train}}(f_j). \end{aligned}$$

To choose the value of the  $f_j$ ’s so that they minimize  $R_j^{\text{train}}$ , take the derivative, set it to 0. Let  $|S_j|$  be the number of examples in leaf  $j$ .

$$\begin{aligned}
0 &= \frac{d}{d\tilde{f}} \sum_{i \in \text{leaf } j} (y_i - \tilde{f})^2 \Big|_{\tilde{f}=f_j} \\
&= -2 \sum_i (y_i - \tilde{f}) \Big|_{\tilde{f}=f_j} = -2 \left( \left( \sum_i y_i \right) - |S_j| \tilde{f} \right) \Big|_{\tilde{f}=f_j} \\
f_j &= \frac{1}{|S_j|} \sum_{i \in \text{leaf } j} y_i = \bar{y}_{S_j},
\end{aligned}$$

where  $\bar{y}_{S_j}$  is the sample average of the labels for leaf  $j$ 's examples.

So now we know what value to assign for  $f$  in each leaf. How to split? Greedily want feature  $j$  and split point  $s$  solving the following.

$$\min_{\substack{j, s \\ \text{for each feature } j \text{ do} \\ \text{a linesearch over } s}} \left[ \min_{C_1} \sum_{x_i \in \{\text{leaf} | x^{(j)} \leq s\}} (y_i - C_1)^2 + \min_{C_2} \sum_{x_i \in \{\text{leaf} | x^{(j)} > s\}} (y_i - C_2)^2 \right].$$

The first term means that we'll choose the optimal  $C_1 = \bar{y}_{\{\text{leaf} | x^{(j)} \leq s\}}$ . The second term means we'll choose  $C_2 = \bar{y}_{\{\text{leaf} | x^{(j)} > s\}}$ .

For pruning, again CART does minimal cost complexity pruning:

$$\text{cost} = \sum_{\text{leaves } j} \sum_{x_i \in S_j} (y_i - \bar{y}_{S_j})^2 + C[\# \text{ leaves in tree}]$$

Now we've finished the dirty work of going over the heuristics of the decision tree algorithms. There are some advantages and disadvantages of trees. First, because they are based on heuristics, they don't always perform well, particularly for imbalanced data. Also if you want to change the loss function, you probably would want to change the splitting criteria. There is a whole cottage industry of academics whose goal it is to design the splitting and pruning criteria for various kinds of decision tree objectives!

From my perspective, going forward, we should aim to design optimal trees, like what GOSDT does. The user should only need to specify what loss function

they care about, and the algorithm should minimize that loss function, regularized by the number of leaves in the tree. Since we can do it for reasonably-sized datasets, we should continue to build that technology rather than working off the old agenda of CART and C4.5, which were built in an era in which full decision tree optimization wasn't really possible.

On the other hand, greedily grown trees (like CART and C4.5) are really useful for combining into more powerful models, like boosted decision trees and random forests (here we are aiming for accuracy, at the expense of interpretability).