Trees

Muchang Bahng

Spring 2025

Contents

Say that you were looking at a picture and were told to classify it as a bird or a dog. You would probably look for some features and have the following thought process: if it has wings, then it is a bird, and if not, then it is a dog.

Now let's try a slightly harder classification. We have four classes consisting of two species of dogs (golden retriever, husky) and two species of birds (pigeon, hummingbird). Then you might work something like this.

1. If it has wings, and
   (a) it has a long beak, then it is a hummingbird.
   (b) it doesn't have a long beak, then it is a pigeon.
2. If it doesn't have wings, and
   (a) its fur color is yellow, then it is a golden retriever.
   (b) its fur color is not yellow, then it is a husky.

Decision trees attempt to model this method of thinking by using a tree structure, and hopefully this example should convince you that this type of model is worth studying. It is a discriminative model that learns to classify data by first identifying the relevant feature to look at (e.g. wings, beak length, fur color) and then deciding how to split it.

Surprisingly, the origin of tree models is not clear, though there have been some papers as early as 1959 that mentions a decision tree-like structure.[1] I personally would have thought it to be older given the simplicity of the idea.

[1] See https://stats.stackexchange.com/questions/257537/who-invented-the-decision-tree.

1 Decision Trees

In here, we define the decision tree model. It is most natural for classification, but there are variants of it for regression. To avoid confusion, I will distinguish them by calling them classification trees and regression trees, and I will use the umbrella term *decision tree*.

Many discriminative models can be written in a clean formula (e.g. $y = w^T x + \epsilon$ for linear regression, and even $y = \prod_i (\sigma_i \circ A_i)(x)$ for MLPs). However, we cannot find such a parameteric form for a tree, which is why they are nonparametric models. In full generality, all we can say is that they have a general tree structure, and there are many variants.

1.1 Classification Trees

Definition 1.1 (Classification Trees)

A **classification tree** is a nonparameteric discriminative model $f : X \to Y$, for finite Y, that uses a tree representing a set of decisions on an input x to predict a label y.

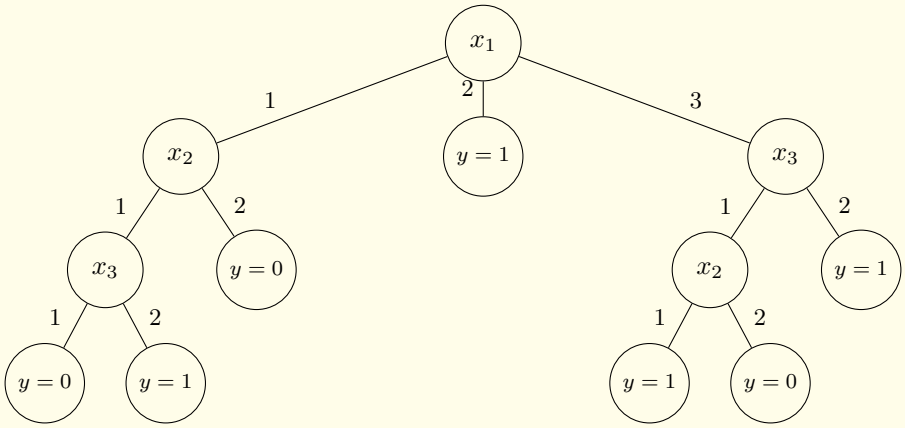


Figure 1: An example of a decision tree. Note that the same feature need not be split across all depths (e.g. in depth 1, the leftmost node is split on x_2 while the rightmost is split on x_3) and the path to a leaf can end early (e.g. there is a node of depth 1 that is a leaf).

The decision tree tries to take advantage of some nontrivial covariance between X and Y by constructing nested partitions of the dataset $\mathcal{D}$, and within a partition, it predicts the label that comprises the majority. Note that this model is extremely flexible in that we can have different properties of these trees. We will introduce them as we go.

Definition 1.2 (Binary Decision Tree)

A **binary decision tree** only allows the tree to split into two nodes.

Note that in density estimation or linear regression, we can derive the risk by first deriving the likelihood of the data, and then taking the negative logarithm of it to get our loss function which allows us to define our risk. In a decision tree, we have a *non-probabilistic* discriminative model, so there is no concept of likelihood. Therefore, we cannot use a pdf to define the loss. Fortunately, we can use the straightforward misclassification risk.

Theorem 1.1 (Expected and Empirical Risk of Decision Trees)

Given a classification tree f, the misclassification risk over the true data generating distribution $p(x, y)$, along with its empirical risk over a dataset $\mathcal{D} = (x^{(i)}, y^{(i)})_{i=1}^n$, is

$$\mathbb{R}(f) = \mathbb{E}_{x,y}\left[\mathbb{1}(y \neq f(x))\right] = \int \mathbb{1}(y \neq f(x))\, dx\, dy \tag{1}$$

$$\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(y^{(i)} \neq f(x^{(i)})) \tag{2}$$

where $\mathbb{1}(p)$ is an indicator function that equals 1 if p is true and 0 if false. Note that $\hat{R}(f)$ is simply 1 minus the accuracy.

Let's take a look at some examples. The behavior of splitting a node can be different depending on what the covariate is. Given covariate x_i,

1. if $x_i \in \{0, 1\}$, then we can simply split it with left and right children.
2. if $x_i \in \{1, \dots, K\}$, then we can split it across K children. We can also choose a cutoff value for a binary split, e.g. go left if $x_i < t$ and right if $x_i \geq t$. Or, we might choose some subset $S \subset \{1, \dots, K\}$, where we go left if $x_i \in K$ and right if $x_i \notin K$.
3. if $x_i \in \mathbb{R}$, then we must choose a cutoff value t or partition $\mathbb{R}$ to split. Usually, a cutoff is chosen in practice, since finding a partition leads to a much more difficult problem to learn.

Let's examine this in the following two examples.

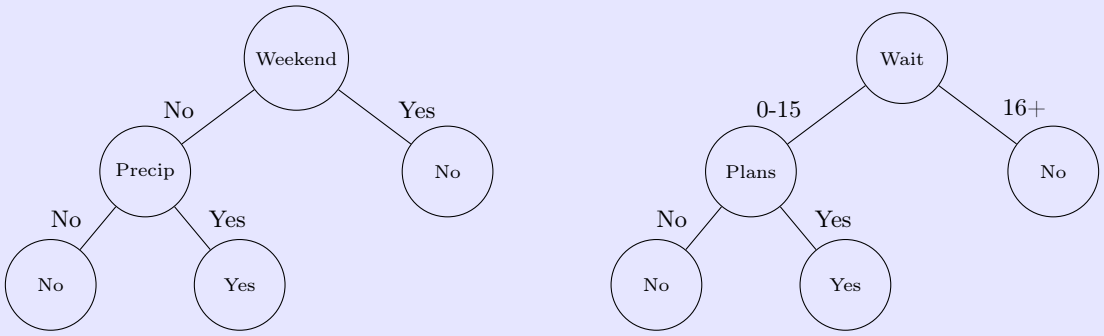
Example 1.1 (Categorical Covariates)

Consider the following dataset, where we consider a binary classification system with discrete covariates.

	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

Table 1: Dataset of whether to go to a restaurant for a date depending on certain factors.

This is a binary classification problem, and we can count that there are 6 positives and 6 negative labels. Let's evaluate the accuracy of some example trees.



(a) The accuracy of this tree is 7 out of 12.

(b) The accuracy of this tree is 9 out of 12.

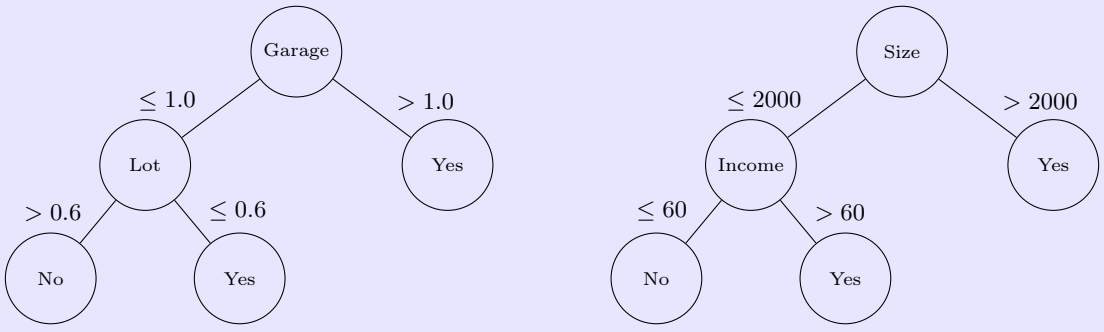
Figure 2: Even though the trees have the same structure, the features that they split on has high impact on the accuracy.

Example 1.2 (Continuous Covariates)

Now let's take a look at a dataset with continuous covariates.

	Size	Age	Bedrooms	Distance	Crime	Income	Schools	Garage	Lot	Expensive?
x_1	2400	8	3.0	2.1	1.2	85	8.7	2.0	0.31	Yes
x_2	1200	45	2.0	8.7	4.8	42	5.2	1.0	0.18	No
x_3	3100	12	4.0	1.4	0.9	92	9.1	2.5	0.45	Yes
x_4	2800	6	3.5	3.2	2.1	78	8.3	2.0	0.28	Yes
x_5	950	62	1.5	12.3	6.7	35	4.1	0.5	0.12	No
x_6	2650	15	3.0	2.8	1.7	68	7.9	2.0	0.35	Yes
x_7	1450	38	2.5	7.1	5.2	48	6.0	1.0	0.22	No
x_8	2200	22	3.0	4.5	2.8	71	7.4	1.5	0.26	Yes
x_9	1100	55	2.0	9.8	5.9	39	4.8	1.0	0.15	No
x_{10}	1800	28	2.5	6.2	3.4	55	6.7	1.5	0.20	No
x_{11}	1350	41	2.0	8.9	4.6	44	5.5	1.0	0.19	No
x_{12}	2900	10	4.0	1.8	1.4	88	8.9	2.5	0.42	Yes

Table 2: Dataset for predicting whether a house is expensive ($500K+) based on continuous features. Size (sq ft), Age (years), Bedrooms (count), Distance (miles to downtown), Crime (incidents per 1000), Income (neighborhood median in $1000s), Schools (rating 1-10), Garage (spaces), Lot (acres).



(a) The accuracy of this tree is 7 out of 12.

(b) The accuracy of this tree is 12 out of 12.

Figure 3

This is all nice in theory, but how are we supposed to optimize this in practice? First, the misclassification loss is not differentiable—and even worse—the gradient is 0 almost everywhere! This isn't as bad as it seems, since we can introduce a surrogate loss function and optimize that. The big problem is that f is not

parameteric, so we can't even gradients at all (with respect to what parameter?)! One solution is to try and create a very specific tree model—making it parameteric—and then using a surrogate loss to learn. In fact this done in practice, but for now let's keep things simple and hold off on this problem until later.

1.2 Regression Trees

Regression trees behave similarly to classification trees, but now the outputs are meant to be continuous. Clearly, a tree having a discrete set of leaf nodes cannot fit a continuum, but we can try to fit it with step functions.

Definition 1.3 (Regression Tree)

A **regression tree** is a nonparameteric discriminative model $f : X \to \mathbb{R}$, that uses a tree representing a set of decisions on an input x to predict a value y.

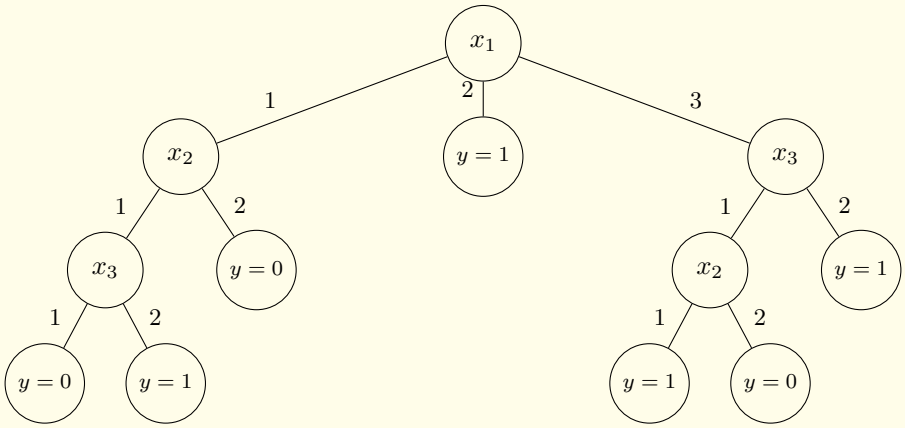
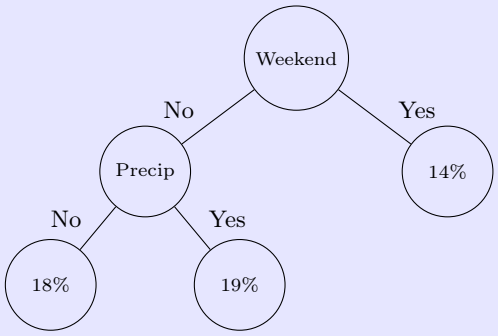


Figure 4: An example of a decision tree. Note that the same feature need not be split across all depths (e.g. in depth 1, the leftmost node is split on x_2 while the rightmost is split on x_3) and the path to a leaf can end early (e.g. there is a node of depth 1 that is a leaf).

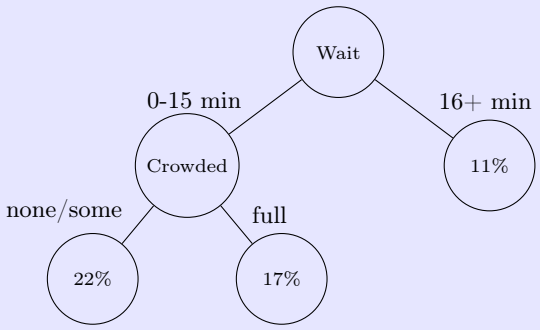
Example 1.3 (Regression on Categorical Covariates)

	OthOptions	Weekend	WaitArea	Plans	Price	Precip	Restaur	Wait	Crowded	Tip%
x_1	Yes	No	No	Yes	$$$	No	Mateo	0-5	some	22%
x_2	Yes	No	No	Yes	$	No	Juju	16-30	full	12%
x_3	No	No	Yes	No	$	No	Pizza	0-5	some	18%
x_4	Yes	Yes	No	Yes	$	No	Juju	6-15	full	20%
x_5	Yes	Yes	No	No	$$$	No	Mateo	30+	full	8%
x_6	No	No	Yes	Yes	$$	Yes	BlueCorn	0-5	some	25%
x_7	No	No	Yes	No	$	Yes	Pizza	0-5	none	15%
x_8	No	No	No	Yes	$$	Yes	Juju	0-5	some	23%
x_9	No	Yes	Yes	No	$	Yes	Pizza	30+	full	10%
x_{10}	Yes	Yes	Yes	Yes	$$$	No	BlueCorn	6-15	full	14%
x_{11}	No	No	No	No	$	No	Juju	0-5	none	16%
x_{12}	Yes	Yes	Yes	Yes	$	No	Pizza	16-30	full	19%

Table 3: Dataset for predicting tip percentage based on restaurant dining factors.



(a) Bad tree splits on Weekend first, then Precip. Mean Absolute Error: 4.2%



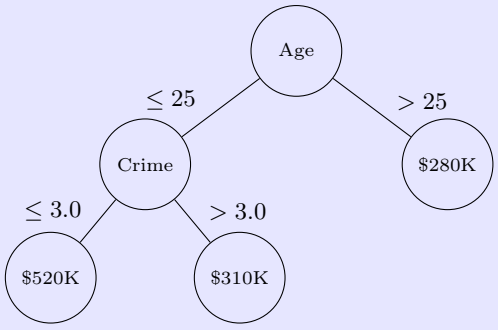
(b) Good tree splits on Wait first, then Crowded. Mean Absolute Error: 2.1%

Figure 5: Comparison of regression trees for predicting tip percentage. The good tree using Wait time and Crowded level achieves much better prediction accuracy than the bad tree using Weekend and Precipitation.

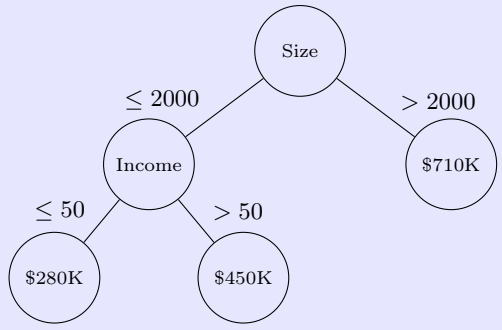
Example 1.4 (Regression on Continuous Covariates)

	Size	Age	Bedrooms	Distance	Crime	Income	Schools	Garage	Lot	Price
x_1	2400	8	3.0	2.1	1.2	85	8.7	2.0	0.31	\$645K
x_2	1200	45	2.0	8.7	4.8	42	5.2	1.0	0.18	\$285K
x_3	3100	12	4.0	1.4	0.9	92	9.1	2.5	0.45	\$825K
x_4	2800	6	3.5	3.2	2.1	78	8.3	2.0	0.28	\$710K
x_5	950	62	1.5	12.3	6.7	35	4.1	0.5	0.12	\$195K
x_6	2650	15	3.0	2.8	1.7	68	7.9	2.0	0.35	\$580K
x_7	1450	38	2.5	7.1	5.2	48	6.0	1.0	0.22	\$325K
x_8	2200	22	3.0	4.5	2.8	71	7.4	1.5	0.26	\$515K
x_9	1100	55	2.0	9.8	5.9	39	4.8	1.0	0.15	\$240K
x_{10}	1800	28	2.5	6.2	3.4	55	6.7	1.5	0.20	\$385K
x_{11}	1350	41	2.0	8.9	4.6	44	5.5	1.0	0.19	\$295K
x_{12}	2900	10	4.0	1.8	1.4	88	8.9	2.5	0.42	\$780K

Table 4: Dataset for predicting house prices based on continuous features. Size (sq ft), Age (years), Bedrooms (count), Distance (miles to downtown), Crime (incidents per 1000), Income (neighborhood median in \$1000s), Schools (rating 1-10), Garage (spaces), Lot (acres).



(a) Bad tree splits on Age first, then Crime. Mean Absolute Error: \$145K



(b) Good tree splits on Size first, then Income. Mean Absolute Error: \$65K

Figure 6: Even though the trees have the same structure, the features they split on have high impact on prediction accuracy. The good tree uses more predictive features (Size and Income) resulting in much lower prediction error.

The next question to ask is how we choose the threshold. Essentially, given a set of points $p_1 < p_2 < \ldots < p_n$, where we want to divide $p_1, \ldots, p_k$ and $p_{k+1} < \ldots < p_n$, which value of $t \in (p_k, p_{k+1})$ should we choose? This depends on the optimization algorithm we are using, but one would be to just choose the midpoint. Another would be to take a weighted average of the points.

1.3 Model Space

The fact that trees are nonparameteric means that we have extreme flexibility in designing our tree. However, this comes with the big risk of having too big of a model space to optimize over. This overcomplexity is one of the big challenges in trees.

For example, suppose that there are d covariates (independent variables, features) $x_1, \ldots, x_d$ all binary valued. We can design a decision tree that splits on x_1, then on x_2, then on x_1, then on x_2, and so on. This becomes unbounded and our model space a discrete infinite space, which is a bad combination since we don't have gradients to optimize over a continuum. We can try and handle this in two ways.

Example 1.5 (Splitting on Same Variable Multiple Times)

Splitting on covariate x_1 infinitely many times seems pretty unrealistic, so we should limit it in some way.

1. *Covariate can be split a maximum of once for each path from root to leaf.* This is a common assumption for simple problems but may lead to insufficient complexity. In some cases, we would like to look at feature x_i, then filter it through x_j, and then look at x_i again.
2. *Covariate can be split maximum of k times.* This can be a practical assumption, but if k is set too high, our model space may be too complex.

Example 1.6 (Depth of Tree)

Another similar—but distinct—way of restricting the model space is to limit the depth (defined as the maximum length of a path from root to any leaf) of the tree.

Now that we have some restrictions, let's try to analyze the model space.

Lemma 1.1 (Set of All Full Trees of a Certain Depth)

Let $x_1, \ldots, x_d$ be binary categorical variables. Let $\mathcal{F}$ be the set of all *perfect*[a] binary decision trees of depth k on a classification problem of C classes. Then,

$$|\mathcal{F}| = d^{2^{k-1}} \cdot C^{2^k} \tag{3}$$

[a] Every nonleaf node has a two children, and all leaves are on the same depth

Proof.

For depth k, there are a total of 2^k leaf nodes and 2^{k-1} internal nodes. Each of the internal nodes can split on any of the d covariates, and the 2^k leaf nodes can take any of the C classes.

This lemma should scare you in that the model complexity is super-exponential with respect to k. It would be much higher if we considered non-perfect classification trees and/or non-binary covariates.

Example 1.7 (Max Subnodes per Split)

If we did have a multiclass covariate x_i taking values in a set S, creating a general partition is

equivalent to identifying an ordered partition

$$S = \bigsqcup_{j=1}^{m} S'_j, \qquad S'_j \neq \emptyset \tag{4}$$

where if $x_i \in S'_j$, then it would get routed to the jth child. If a node must have m children, then we must consider all m-partitions of S, which is super-exponential. This becomes worse if a node can have *up to* m children. This is clearly too complex, so we could think of limiting the number of children so that $m = 2$, i.e. we are only allowed to do binary splits. Note that even with this, there are still $2^{|S|} - 2$ ways to split.[a]

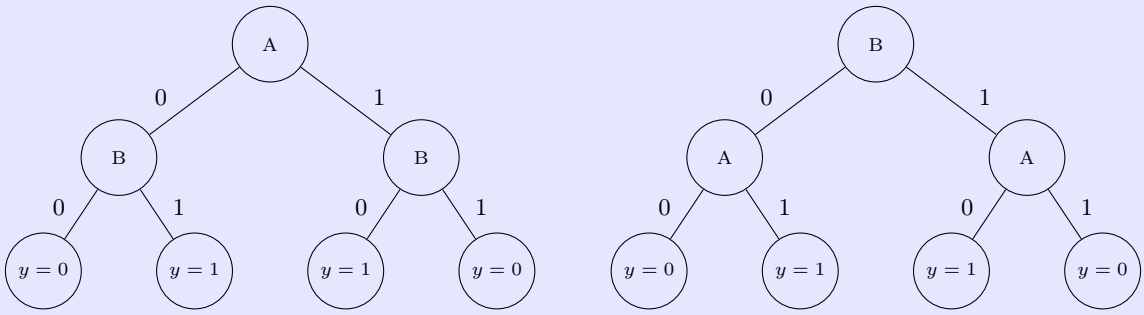
[a] We can choose any arbitrary subset $R \subset S$, which gives us the partition $R \sqcup R^c = S$. R gets routed to the right node and R^c gets routed to the left. We subtract 2 to make sure that $R, R^C \neq \emptyset$.

This is not all. It is well known that different trees are *functionally* the same.

Example 1.8 (Model Equivalence)

The trees below are functionally equal the function

$$f(x) = \begin{cases} 0 & \text{if } (A,B) = (0,0) \\ 1 & \text{if } (A,B) = (0,1) \\ 1 & \text{if } (A,B) = (1,0) \\ 0 & \text{if } (A,B) = (1,1) \end{cases} \tag{5}$$



(a) Tree splits on A first, then B

(b) Tree splits on B first, then A

Figure 7: Two functionally equivalent decision trees that implement the XOR function: $y = A \oplus B$

Therefore, there is repetition in our model space, and our misclassification loss will treat them equally. Even though they are functionally the same, it doesn't make them *practically* equal. In reality, with missing data, we would like the path to our leaves—the final decisions—to be short as possible. Therefore, we may not need to use every covariate to reach a decision. If we can avoid the uncertain covariates (e.g. ones with more missing values or noise), then we can get a more robust decision tree.

Example 1.9 (Robust Decision Tree without Always Querying Covariate A)

We can see that the two decision trees are functionally the same.

$$f(x) = \begin{cases} 0 & \text{if } (A,B,C) = (0,0,0) \\ 0 & \text{if } (A,B,C) = (0,0,1) \\ 1 & \text{if } (A,B,C) = (0,1,0) \\ 1 & \text{if } (A,B,C) = (0,1,1) \\ 0 & \text{if } (A,B,C) = (1,0,0) \\ 1 & \text{if } (A,B,C) = (1,0,1) \\ 0 & \text{if } (A,B,C) = (1,1,0) \\ 1 & \text{if } (A,B,C) = (1,1,1) \end{cases} \tag{6}$$

Note that the right tree can do the same as the left tree, but in some cases it does not need to query A at all. Therefore, in some cases, we can reach a decision even when A is missing.

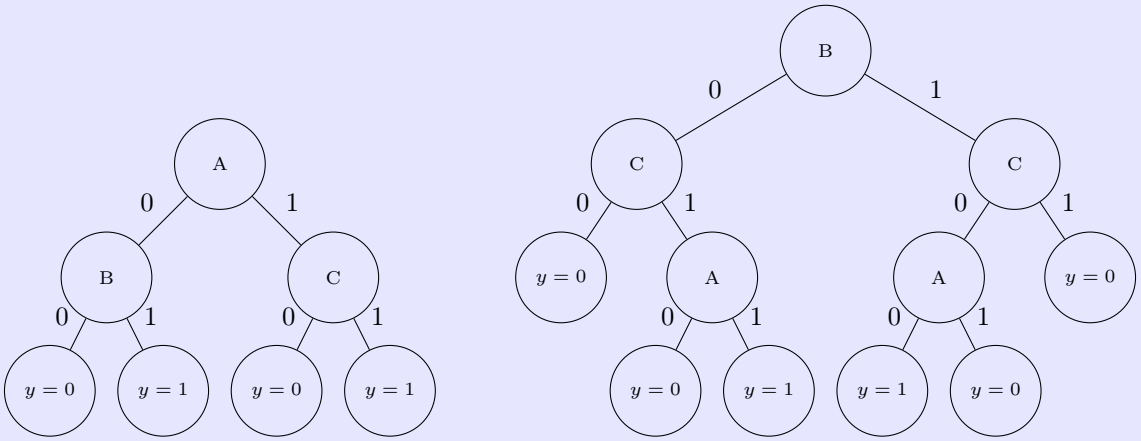


Figure 8: Credits to [MBD+25].

In 2025, McTavish showed an algorithm of finding such trees [MBD+25].

2 Greedy Optimization

Optimizing a decision tree is NP-complete [HR76].

Let us take a decision tree h and model the accuracy of it as a random variable: $1_{\{Y=h_0(X)\}} \sim \text{Bernoulli}(p)$, where p is the accuracy. A higher accuracy of h corresponds to a lower entropy, and so the entropy of the random variable is also a relevant indicator.

$$H(1_{\{Y=h_0(X)\}}) = p \log p + (1-p)\log(1-p) \tag{7}$$

Therefore, when we are building a tree, we want to choose the feature x_i to split based on how much it lowers the entropy of the decision tree.

To set this up, let us take our dataset $\mathcal{D}$ and set X_i as the random variable representing the distribution (a multinomial) of the $x_i^{(j)}$'s, and Y as the same for the $y^{(j)}$'s. This is our maximum likelihood approximation for the marginalized distribution of the joint measure $X \times Y = X_1 \times \ldots \times X_D \times Y$.

Given a single node, we are simply going to label every point to be whatever the majority class is in $\mathcal{D}$. Therefore, we start off with the entropy of our trivial tree $H(Y)$. Then, we want to see which one of the X_d features to split on, and so we can compute the conditional entropy $H(Y, X_d)$ to get the information gain $I(Y; X_d) = H(Y) - H(Y \mid X_d)$ for all $d = 1, \ldots, D$. We want to find a feature X_d that maximize this information gain, i.e. decreases the entropy as much as possible (a greedy algorithm), and we find the next best feature (with or without replacement), so that we have a decreasing sequence.

$$H(X) \geq H(X;Y) \geq H(X;Y,Z) \geq H(X;Y,Z,W) \geq \ldots \geq 0$$

Example 2.1 (Crowded Restaurants)

Continuing the example above, since there are 6 labels of 0 and 1 each, we can model this $Y \sim \text{Bernoulli}(0.5)$ random variable, with entropy

$$H(Y) = \mathbb{E}[-\log_2 p(Y)] = \frac{1}{2}\big(-\log_2 \frac{1}{2}\big) + \frac{1}{2}\big(-\log_2 \frac{1}{2}\big) = 1 \tag{8}$$

Now what would happen if we had branched according to how crowded it was, X_{crowded}. Then, our decision tree would split into 3 sections:

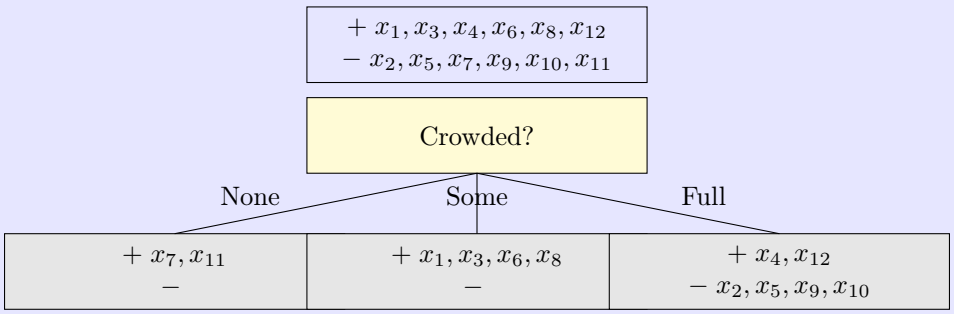


Figure 9: Visual of decision tree splitting according to how crowded it is.

In this case, we can define the multinomial distribution X_{crowded} representing the proportion of the data that is crowded in a specific level. That is, $X_{\text{crowded}} \sim \text{Multinomial}(\frac{2}{12}, \frac{4}{12}, \frac{6}{12})$, with

$$\mathbb{P}(X_{\text{crowded}} = x) = \begin{cases} 2/12 & \text{if } x = \text{ none} \\ 4/12 & \text{if } x = \text{ some} \\ 6/12 & \text{if } x = \text{ full} \end{cases} \tag{9}$$

Therefore, we can now compute the conditional entropy of this new decision tree conditioned on how crowded the store is

$$H(Y \mid X_{\text{crowded}}) = \sum_x \mathbb{P}(X_{\text{crowded}} = x) H(Y \mid X_{\text{crowded}} = x) \tag{10}$$

$$= \frac{2}{12} H(\text{Bern}(1)) + \frac{4}{12} H(\text{Bern}(0)) + \frac{6}{12} H(\text{Bern}(1/3)) = 0.459 \tag{11}$$

$$I(Y; X_{\text{crowded}}) = 0.541 \tag{12}$$

We would do this for all the features and greedily choose the feature that maximizes our information gain.

Example 2.2 (Ferrari F1 Race)

The Ferrari F1 team hired you as a new analyst! You were given the following table of the past race history of the team. You were asked to use information gain to build a decision tree to predict race wins. First, you will need to figure out which feature to split first.

Rain	Good Strategy	Qualifying	Win Race
1	0	0	0
1	0	0	0
1	0	1	0
0	0	1	1
0	0	0	0
0	1	1	1
1	0	1	0
0	1	0	1
0	0	1	1
0	0	1	1

Let $X \sim \text{Bernoulli}(1/2)$ be the distribution of whether a car wins a race over the data. Then its entropy is

$$H(X) = \mathbb{E}[-\log_2 p(x)] = \frac{1}{2}\big(-\log_2 \frac{1}{2}\big) + \frac{1}{2}\big(-\log_2 \frac{1}{2}\big) = 1 \tag{13}$$

Let $R \sim \text{Bernoulli}(4/10), G \sim \text{Bernoulli}(2/10), Q \sim \text{Bernoulli}(6/10)$ be the distribution of the features rain, good strategy, and qualifying over the data, respectively. Then, the conditional entropy of X conditioned on each of these random variables is

$$\begin{aligned}
H(X \mid R) &= \mathbb{P}(R=1)\, H(X \mid R=1) + \mathbb{P}(R=0)\, H(X \mid R=0) \\
&= \frac{4}{10} \cdot -\big(1 \cdot \log_2 1 + 0 \cdot \log_2 0\big) + \frac{6}{10} \cdot -\big(\frac{1}{6} \cdot \log_2 \frac{1}{6} + \frac{5}{6} \cdot \log_2 \frac{5}{6}\big) \approx 0.390 \\
H(X \mid G) &= \mathbb{P}(G=1)\, H(X \mid G=1) + \mathbb{P}(G=0)\, H(X \mid G=0) \\
&= \frac{2}{10} \cdot -\big(1 \cdot \log_2 1 + 0 \cdot \log_2 0\big) + \frac{8}{10} \cdot -\big(\frac{3}{8} \cdot \log_2 \frac{3}{8} + \frac{5}{8} \log_2 \frac{5}{8}\big) \approx 0.763 \\
H(X \mid Q) &= \mathbb{P}(Q=1)\, H(X \mid Q=1) + \mathbb{P}(Q=0)\, H(X \mid Q=0) \\
&= \frac{6}{10} \cdot -\big(\frac{4}{6} \cdot \log_2 \frac{4}{6} + \frac{2}{6} \cdot \log_2 \frac{2}{6}\big) + \frac{4}{10} \cdot -\big(\frac{1}{4} \log_2 \frac{1}{4} + \frac{3}{4} \log_2 \frac{3}{4}\big) \approx 0.875
\end{aligned}$$

Therefore, the information gain are

$$\begin{aligned}
I(X; R) &= 1 - 0.390 = 0.610 \\
I(X; G) &= 1 - 0.763 = 0.237 \\
I(X; Q) &= 1 - 0.875 = 0.125
\end{aligned}$$

And so I would split on R, the rain, which gives the biggest information gain.

Finally, we can use the Gini index of $X \sim \text{Bernoulli}(p)$, defined

$$G(X) = 2p(1-p) \tag{14}$$

Example 2.3 (Ferrari Example Continued)

We do the same as the Ferrari example above but now with the Gini reduction. Let $X \sim \text{Bernoulli}(1/2)$ be the distribution of whether a car wins a race over the data. Then its Gini index, which I will label with $\mathcal{G}$, is

$$\mathcal{G}(X) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

Let $R \sim \text{Bernoulli}(4/10), G \sim \text{Bernoulli}(2/10), Q \sim \text{Bernoulli}(6/10)$ be the distribution of the features rain, good strategy, and qualifying over the data, respectively. Then we compute the conditional expectation

$$\begin{aligned}
\mathbb{E}[\mathcal{G}(X \mid R)] &= \mathbb{P}(R=1)\,\mathcal{G}(X \mid R=1) + \mathbb{P}(R=0)\,\mathcal{G}(X \mid R=0) \\
&= \frac{4}{10}\left[2 \cdot \frac{4}{4} \cdot \frac{0}{4}\right] + \frac{6}{10}\left[2 \cdot \frac{1}{6} \cdot \frac{5}{6}\right] \approx 0.167 \\
\mathbb{E}[\mathcal{G}(X \mid G)] &= \mathbb{P}(G=1)\,\mathcal{G}(X \mid G=1) + \mathbb{P}(G=0)\,\mathcal{G}(X \mid G=0) \\
&= \frac{2}{10}\left[2 \cdot \frac{2}{2} \cdot \frac{0}{2}\right] + \frac{8}{10}\left[2 \cdot \frac{3}{8} \cdot \frac{5}{8}\right] \approx 0.375 \\
\mathbb{E}[\mathcal{G}(X \mid Q)] &= \mathbb{P}(Q=1)\,\mathcal{G}(X \mid Q=1) + \mathbb{P}(Q=0)\,\mathcal{G}(X \mid Q=0) \\
&= \frac{6}{10}\left[2 \cdot \frac{4}{6} \cdot \frac{2}{6}\right] + \frac{4}{10}\left[2 \cdot \frac{1}{4} \cdot \frac{3}{4}\right] \approx 0.417
\end{aligned}$$

Therefore, the Gini reduction, which I'll denote as $I_{\mathcal{G}}$, is

$$\begin{aligned}
I_{\mathcal{G}}(X; R) &= 0.5 - 0.167 = 0.333 \\
I_{\mathcal{G}}(X; G) &= 0.5 - 0.375 = 0.125 \\
I_{\mathcal{G}}(X; Q) &= 0.5 - 0.417 = 0.083
\end{aligned}$$

Since branching across the feature R, the rain, gives the biggest Gini reduction, we want to split on the rain feature first.

2.1 Surrogate Loss

2.2 CART

2.3 ID3

2.4 c4.5

3 Regularization

Given a dataset with D binary features, let $g(H, D)$ be the number of binary trees with depth at most H (including root node), with the restriction that the trees may not split on some variable multiple times within a path to a leaf node. Then, g can be defined recursively.

1. First, if $H = 1$, then $g(H, D) = 1$ always since we are just creating the trivial binary tree of one node.
2. If $D = 0$, then there are no features to split on and therefore we just have the single node $g(H, D) = 1$.
3. If $H > 1$ and $D > 0$, then say that we start with a node. We can either make this a leaf node by not performing any splitting at all, or split on one of the D variables. Then for each of the 2 nodes created on the split, we are now working with $D - 1$ features and a maximum height of $H - 1$ for each of the subtrees generated from the 2 nodes.

All this can be expressed as

$$g(H, D) = \begin{cases} 1 + D\left[g(H-1, D-1)\right]^2 & \text{if } H > 1, D > 0 \\ 1 & \text{if } H = 1 \text{ or } D = 0 \end{cases}$$

which is extremely large (in fact, NP hard). Therefore, some tricks like regularization must be implemented to limit our search space.

By defining the complexity of our decision tree $\Omega(h)$ as the number of nodes within the tree, we can modify our objective function to

$$L(h; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} 1_{\{y^{(i)} \neq h(x^{(i)})\}} + \lambda \Omega(h)$$

We can impose this constraint directly on the training algorithm, or we can calculate the regularized loss after the tree has been constructed, which is a method called **tree pruning**.

Given a large enough λ, we can in fact greatly reduce our search space by not considering any trees further than a certain point.

Theorem 3.1 ()

We describe a tree as a set of leaves, where leaf k is a tuple containing the logical preposition satisfied by the path to leaf k, denoted p_k, and the class label predicted by the leaf, denoted $\hat{y}_k$. For a dataset with d binary features, $p_k : \{0,1\}^d \to \{0,1\}$ is a function that returns 1 if a sample x_i satisfies the preposition, and 0 otherwise. That is, leaf k is $(p_k, \hat{y}_k)$, and a tree f with K leaves is described as a set $f = \{(p_1, \hat{y}_1), \dots, (p_K, \hat{y}_K)\}$. Assume that the label predicted by $\hat{y}_k$ is always the label for the majority of samples satisfying p_k. Finally, let $m_k = \sum_{i=1}^{n} p_k(x_i)$ denote the number of training samples "captured" by leaf k.
Given a (potentially optimal) tree

$$f = \{(p_1, \hat{y}_1), \dots, (p_\kappa, \hat{y}_\kappa), \dots, (p_K, \hat{y}_K)\},$$

the tree $f' = \{(p_1, \hat{y}_1), \dots, (p_{\kappa_1}, \hat{y}_{\kappa_1}), (p_{\kappa_2}, \hat{y}_{\kappa_2}), \dots, (p_K, \hat{y}_K)\}$ produced by splitting leaf $(p_\kappa, \hat{y}_\kappa)$ into two leaves $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ and $(p_{\kappa_2}, \hat{y}_{\kappa_2})$ and any tree produced by further splitting $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ or $(p_{\kappa_2}, \hat{y}_{\kappa_2})$ cannot be optimal if $m_\kappa < 2n\lambda$.

Proof.

Let c be the number of misclassifications in leaf $(p_\kappa, \hat{y}_\kappa)$. Since a leaf classifies according to the majority of m_κ, we must have

$$c \leq \frac{m_\kappa}{2} < n\lambda$$

By splitting leaf $(p_\kappa, \hat{y}_\kappa)$ into leaves $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ and $(p_{\kappa_2}, \hat{y}_{\kappa_2})$, assume that we have reduced the number of misclassifications by $b \leq c$. Then, we have

$$\ell(f', \mathbf{X}, \mathbf{y}) = \ell(f, \mathbf{X}, \mathbf{y}) - \frac{b}{n}$$

However, we have increased the number of leaves by 1, and so

$$\lambda s(f') = \lambda s(f) + \lambda$$

Combining the last two equations, we have obtained

$$R(f', \mathbf{X}, \mathbf{y}) = R(f, \mathbf{X}, \mathbf{y}) + \lambda - \frac{b}{n}$$

However, we know that

$$\begin{aligned} b \leq c &\implies \frac{b}{n} \leq \frac{c}{n} < \frac{n\lambda}{n} = \lambda \\ &\implies -\frac{b}{n} > -\lambda \\ &\implies \lambda - \frac{b}{n} > \lambda - \lambda = 0 \end{aligned}$$

and so $R(f', \mathbf{X}, \mathbf{y}) > R(f, \mathbf{X}, \mathbf{y})$. This means that f' cannot be optimal according to our regularized objective. We have also proved that further splitting $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ or $(p_{\kappa_2}, \hat{y}_{\kappa_2})$ cannot be optimal since we can just set $f = f'$, and apply the same argument.

3.1 Pruning

3.2 Splitting

4 Improved Optimization

4.1 GODST

5 Soft Decision Trees

5.1 Soft Splitting

5.2 Neural Decision Trees

References

[HR76] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[MBD+25] Hayden McTavish, Zachery Boner, Jon Donnelly, Margo Seltzer, and Cynthia Rudin. Leveraging predictive equivalence in decision trees, 2025.