Trees

Muchang Bahng

Spring 2025

Contents

1 Classification Trees

Definition 1.1 (Decision Trees)

Like K nearest neighbors, **decision trees** are discriminative nonparametric classification algorithms that involves creating some sort of tree that represents a set of decisions using a given set of input data $\mathbf{x}^{(i)}$ with its given classification $\mathbf{y}^{(i)}$. When predicting the class of a new input $\hat{\mathbf{x}}$, we would look at its attributes in some order, e.g. $\hat{x_1}, \hat{x_2}, \hat{x_3}$, and make a decision on which class it is in.

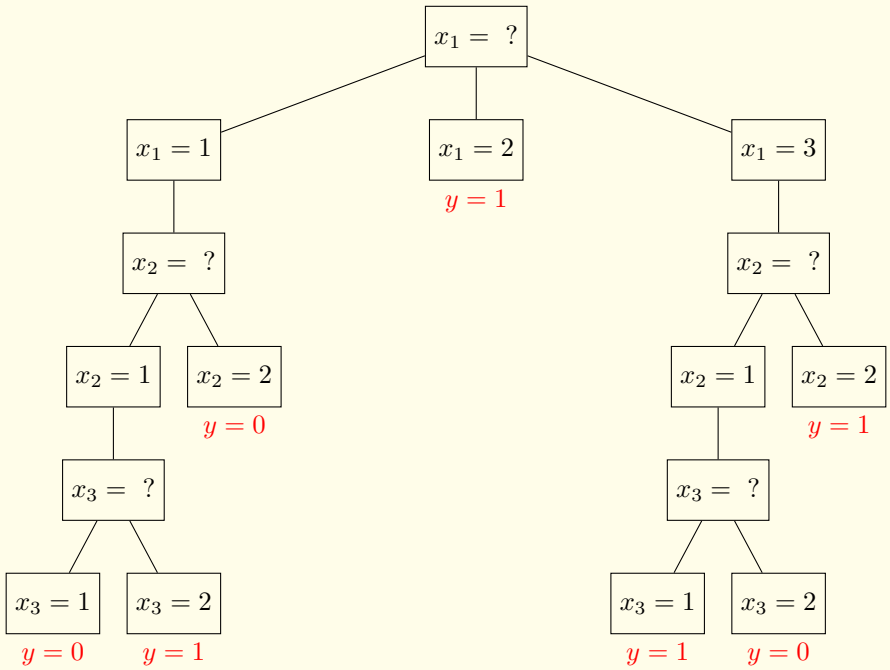

Figure 1: An example of a decision tree that splits at x_1 first, then x_2, and finally x_3. Note that you can still split on x_2 if $x_1 = 1$ and x_3 if $x_1 = 3$.

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.

For now, let us assume that $\mathcal{X}$ is a Cartesian product of discrete sets, and we will extend them to continuous values later. Let us look at an example to gain some intuition.

Example 1.1 (Restaurant Dataset)

Consider the following dataset.

	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.

Let us denote $\mathcal{D}$ as the dataset, and say that $F_1, \dots, F_d$ were the features. This is a binary classification problem, and we can count that there are 6 positives and 6 negative labels.

The simplest decision tree is the trivial tree, with one node that predicts the majority of the dataset. In this case, the data is evenly split, so without loss of generality we will choose $h_0(\mathbf{x}) = 1$. We want to quantify how good our model is, and so like always we use a loss function.

Just like how a linear model is completely defined by its parameter $\boldsymbol{\theta}$, a decision tree is completely defined by the sequences of labels that it splits on. Therefore, training this is equivalent to defining the sequence, but we can't define this sequence unless we can compare how good a given decision tree is, i.e. unless we have defined a proper loss function. Depending on the training, we can use a greedy algorithm or not, and we have the flexibility to choose whether or not we can split on the same feature multiple times.

Definition 1.2 (Misclassification Error)

We will simply use the misclassification loss function.

$$L(h; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} 1_{\{y^{(i)} \neq h(x^{(i)})\}} = 1 - \text{accuracy} \tag{1}$$

Minimizing this maximizes the accuracy, so this is a reasonable one to choose. How do we train this? Unlike regression, this loss is not continuous, so the gradient is 0, and furthermore the model isn't even parametric, so there are no gradients to derive!

Fortunately, the nature of the decision tree only requires us to look through the explanatory variables $x_1, \dots, x_n$ and decide which one to split.

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.

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

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 $d = 1, \dots, D$. 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.

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

Example 1.2 (Crowded Restaurants)

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

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

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:

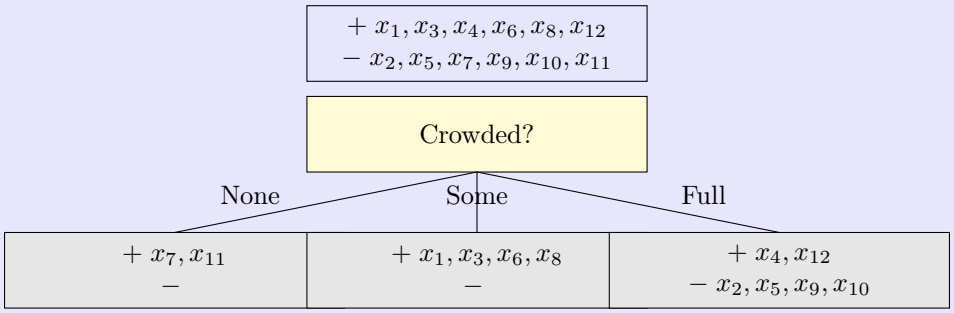

Figure 2: Visual of decision tree splitting according to how crowded it is.

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

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

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

$$H(Y \mid X_{\text{crowded}}) = \sum_x \mathbb{P}(X_{\text{crowded}} = x) H(Y \mid X_{\text{crowded}} = x) \tag{3}$$

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

$$I(Y; X_{\text{crowded}}) = 0.541 \tag{5}$$

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

Example 1.3 (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

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

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

$$G(X) = 2p(1-p) \tag{6}$$

Example 1.4 (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.

1.1 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 1.1 ()

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.

2 Regression Trees

References