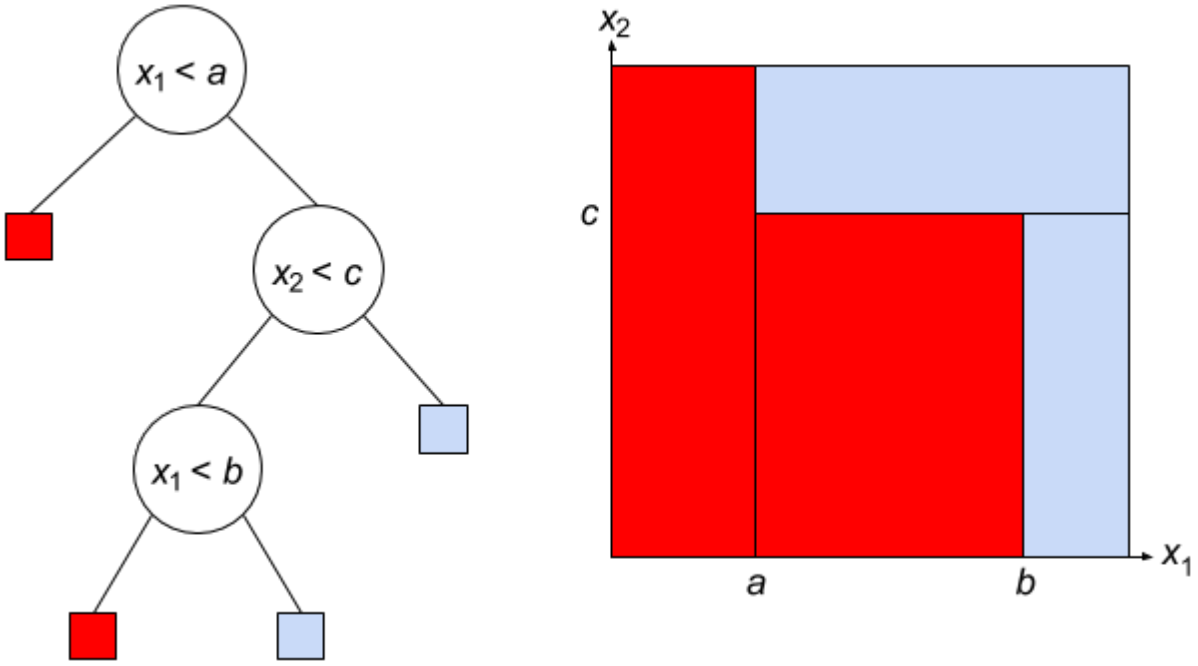
1 Decision Trees

A **decision tree** is a model that makes predictions by posing a series of simple tests on the given point. This process can be represented as a tree, with the non-leaf nodes of the tree representing these tests. At the leaves of this tree are the values to be predicted after descending the tree in the manner prescribed by each node's test. Decision trees can be used for both classification and regression, but we will focus exclusively on classification.

In the simple case which we consider here, the tests are of the form "Is feature j of this point less than the value v?"[1] These tests carve up the feature space in a nested rectangular fashion:



Given sufficiently many splits, a decision tree can represent an arbitrarily complex classifier and perfectly classify any training set.[2]

[1] One could use more complicated decision procedures. For example, in the case of a categorical variable, there could be a separate branch in the tree for each value that the variable could take on. Or multiple features could be used, e.g. "Is $x_1 > x_2$?" However, using more complicated decision procedures complicates the learning process. For simplicity we consider the single-feature binary case here.

[2] Unless two training points of different classes coincide.

Note 25, ©UCB CS 189, Spring 2018. All Rights Reserved. This may not be publicly shared without explicit permission.

1.1 Training

Decision trees are trained in a greedy, recursive fashion, downward from the root. After creating a node with some associated split, the children of that node are constructed by the same tree-growing procedure. However, the data used to train left subtree are only those points satisfying $x_j < v$, and similarly the right subtree is grown with only those points with $x_j \geq v$. Eventually we must each a base case where no further splits are to be made, and a prediction (rather than a split) is associated with the node.

We can see that the process of building the tree raises at least the following questions:

- How do we pick the split-feature, split-value pairs?
- How do we know when to stop growing the tree?

Typically decision trees will pick a split by considering all possible splits and choosing the one that is the best according to some criterion. We will discuss possible criteria later, but first it is worth asking what we mean by "all possible splits". It is clear that we should look at all features, but what about the possible values? Observe that in the case where tests are of the form $x_j < v$, there are infinitely many values of v we could choose, but only finitely many different resulting splits (since there are finitely many training points). Therefore it suffices to perform a one-dimensional sweep: sort the datapoints by their x_j values and only consider these values as split candidates.

Now let us revisit the criterion. Intuitively, we want to choose the split which most reduces our classifier's uncertainty about which class points belongs to. In the ideal case, the hyperplane $x_j = v$ perfectly splits the given data points such that all the instances of the positive class lie on one side and all the instances of the negative class lie on the other.

1.1.1 Entropy and information

One way to quantify the aforementioned "uncertainty", we'll use the ideas of **surprise** and **entropy**. The surprise of observing that a discrete random variable Y takes on value k is:

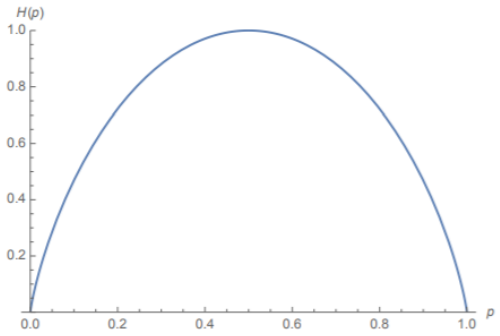
$$\log \frac{1}{P(Y=k)} = -\log P(Y=k)$$

As $P(Y=k) \to 0$, the surprise of observing that value approaches ∞, and conversely as $P(Y=k) \to 1$, the surprise of observing that value approaches 0.

The entropy of Y, denoted $H(Y)$, is the expected surprise:

$$\begin{aligned} H(Y) &= \mathbb{E}[-\log P(Y)] \\ &= -\sum_k P(Y=k) \log P(Y=k) \end{aligned}$$

Here's a graph of entropy vs. p for a Bernoulli(p) random variable:



Observe that as a function of p (the probability of the variable being 1), the entropy is strictly concave. Moreover, it is maximized at $p = \frac{1}{2}$, when the probability distribution is uniform with respect to the outcomes. That is to say, a coin that is completely fair ($P(Y = 0) = P(Y = 1) = \frac{1}{2}$) has more entropy than a coin that is biased. This is because we are less sure of the outcome of the fair coin than the biased coin *overall*. Even though we are more surprised when the biased coin comes up as its more unlikely outcome, the way that entropy is defined gives a higher uncertainty score to the fair coin. Generally speaking, a random variable has more entropy when the distribution of its outcomes is closer to uniform and less entropy when the distribution is highly skewed to one outcome.

This definition is for random variables, but in practice we work with data. The distribution is empirically defined by our training points $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. Concretely, the probability of class k is the proportion of datapoints having class k:

$$P(Y = k) = \frac{|\{i \mid y_i = k\}|}{n}$$

We know that when we choose a split-feature, split-value pair, we want to reduce entropy in some way. Let $X_{j,v}$ be an indicator variable which is 1 when $x_j < v$, and 0 otherwise. There are a few entropies to consider:

- $H(Y)$
- $H(Y|X_{j,v} = 1)$, the entropy of the distribution of points such that $x_j < v$.
- $H(Y|X_{j,v} = 0)$, the entropy of the distribution of points such that $x_j \geq v$.

$H(Y)$ is not really under our control: we start with the set of points with labels represented by Y, this distribution has some entropy, and now we wish to carve up those points in a way to minimize the entropy remaining. Thus, the quantity we want to minimize is a weighted average of the two sides of the split, where the weights are (proportional to) the sizes of two sides:

$$\text{minimize } H(Y|X_{j,v}) := P(X_{j,v} = 1)H(Y|X_{j,v} = 1) + P(X_{j,v} = 0)H(Y|X_{j,v} = 0)$$

This quantity $H(Y|X_{j,v})$ is known as the **conditional entropy** of Y given $X_{j,v}$. An equivalent way of seeing this is that we want to maximize the information we've learned, which is represented by how much entropy is reduced after learning whether or not $x_j < v$:

$$\text{maximize } I(X_{j,v}; Y) := H(Y) - H(Y|X_{j,v})$$

This quantity $I(X_{j,v}; Y)$ is known as the **mutual information** between $X_{j,v}$ and Y. It is always nonnegative, and it's zero iff the resulting sides of the split have the same distribution of classes as the original set of points. Let's say you were using a decision tree to classify emails as spam and ham. For example, you gain no information if you take a set of (20 ham, 10 spam) and split it on some feature to give you sets of (12 ham, 6 spam); (8 ham, 4 spam) because the empirical distribution of those two resulting sets is equal to the original one.

1.1.2 Gini impurity

Another way to assess the quality of a split is **Gini impurity**, which measures how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. It as defined as

$$G(Y) = \sum_k P(Y = k) \sum_{j \neq k} P(Y = j) = \sum_k P(Y = k)(1 - P(Y = k)) = 1 - \sum_k P(Y = k)^2$$

Exactly as with entropy, we can define a version of this quantity which is dependent on the split. For example, $G(Y|X_{j,v} = 1)$ would be the Gini impurity computed only on those points satisfying $x_j < v$. And we can define an analogous quantity

$$G(Y|X_{j,v}) := P(X_{j,v} = 1)G(Y|X_{j,v} = 1) + P(X_{j,v} = 0)G(Y|X_{j,v} = 0)$$

which is to be minimized.

Empirically, the Gini impurity is found to produce very similar results to entropy, and it is slightly faster to compute because we don't need to take logs.

1.1.3 Why not the misclassification rate?

Since we ultimately care about classification accuracy, it is natural to wonder why we don't directly use the misclassification rate by plurality vote as the measure of impurity:

$$M(Y) = 1 - \max_k P(Y = k)$$

It turns out that this quantity is insensitive in the sense that the quantity it induces for evaluating splits ($M(Y|X_{j,v})$) may assign the same value to a variety of splits which are not, in fact, equally good. Suppose[3] the current node has 40 training points of class 1 and 40 of class 2. Here $M(Y) = 1 - \frac{1}{2} = \frac{1}{2}$. Now consider two possible splits:

1. Separate into a region $x_j < v$ with 30 points of class 1 and 10 of class 2 ($10/40 = 1/4$ misclassified), and a region $x_j \geq v$ with 10 points of class 1 and 30 of class 2 ($10/40 = 1/4$ misclassified). Since $40/80 = 1/2$ of the points lie satisfy $x_j < v$ and $40/80 = 1/2$ of the points lie satisfy $x_j \geq v$,

$$M(Y|X_{j,v}) = \frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{4}$$

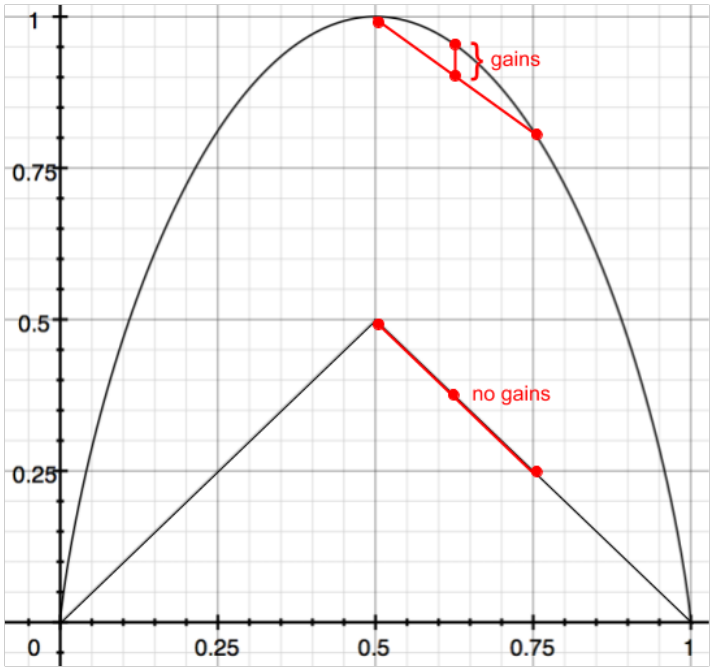
[3] This example is from `https://sebastianraschka.com/faq/docs/decision-tree-binary.html`

2. Separate into a region $x_j < v$ with 20 points of class 1 and 40 of class 2 ($^{20}/_{60} = {}^{1}/_{3}$ misclassified), and a region $x_j \geq v$ with 20 points of class 1 and 0 of class 2 ($^{0}/_{20} = 0$ misclassified). Since $^{60}/_{80} = {}^{3}/_{4}$ of the points lie satisfy $x_j < v$ and $^{20}/_{80} = {}^{1}/_{4}$ of the points lie satisfy $x_j \geq v$,

$$M(Y|X_{j,v}) = \frac{3}{4} \cdot \frac{1}{3} + \frac{1}{4} \cdot 0 = \frac{1}{4}$$

We see that the criterion value is the same for both splits, even though the second one appears to do a better job reducing our uncertainty for many of the points. And indeed, the conditional entropy and Gini impurity scores are lower (so the information gain is higher) for the second split.[4]

The limitation of this criterion can be understood mathematically in terms of **strict concavity**. This property means that the graph of the function lies strictly below the tangent line[5] at every point (except the point of tangency, of course). Consider the plots of entropy and misclassification rate for a binary classification problem:



Both curves are concave, but the one for misclassification rate is not strictly so; it has only two unique tangent lines. Because the conditional quantity is a convex combination (since it is weighted by probabilities/proportions) of the children's values, the strictly convex functions always yield positive information gain as long as the children's distributions are not identical to the parent's. We have no such guarantee in either linear region of the misclassification rate curve; any convex combination of points on a line also lies on the line, yielding zero information gain.

[4] Left as exercise (or see the source URL).

[5] Or, in higher dimensions, hyperplane

1.1.4 Stopping criteria

We mentioned earlier that sufficiently deep decision trees can represent arbitrarily complex decision boundaries, but of course this will lead to overfitting if we are not careful. There are a number of heuristics we may consider to decide when to stop splitting:

- Limited depth: don't split if the node is beyond some fixed depth depth in the tree
- Node purity: don't split if the proportion of training points in some class is sufficiently high
- Information gain criteria: don't split if the gained information/purity is sufficiently close to zero

Note that these are not mutually exclusive, and the thresholds can be tuned with validation. As an alternative (or addition) to stopping early, you can **prune** a fully-grown tree by re-combining splits if doing so reduces validation error.

2 Random Forests

Another way to combat overfitting is to combine the predictions of many varied models into a single prediction, typically by plurality vote in the case of classification and averaging in the case of regression. This is a general technique known as **ensemble learning**. To understand the motivation for averaging, consider a set of uncorrelated random variables $\{Y_i\}_{i=1}^n$ with common mean $\mathbb{E}[Y_i] = \mu$ and variance $\mathrm{Var}(Y_i) = \sigma^2$. The average of these has the same expectation

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} Y_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[Y_i] = \frac{1}{n}\cdot n\mu = \mu$$

but reduced variance compared to each of the individual Y_i's:

$$\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \left(\frac{1}{n}\right)^2\sum_{i=1}^{n}\mathrm{Var}(Y_i) = \frac{1}{n^2}\cdot n\sigma^2 = \frac{\sigma^2}{n}$$

In the context of ensemble methods, these Y_i are analogous to the prediction made by classifier i. The combined prediction has the same expected value as any individual prediction but lower variance. Real-world predictions will of course not be completely uncorrelated, but reducing correlation will generally reduce the final variance, so this is a goal to aim for.

Random forests are a specific ensemble method where the individual models are decision trees trained in a randomized way so as to reduce correlation among them. Because the basic decision tree building algorithm is deterministic, it will produce the same tree every time if we give it the same dataset and use the same algorithm hyperparameters (stopping conditions, etc.).

Random forests are typically randomized in the following ways:

- Per-classifier **bagging** (short for **bootstrap aggregating**): sample some number $m < n$ of datapoints uniformly with replacement, and use these as the training set.

- Per-split **feature randomization**: sample some number number $k < d$ of features as candidates to be considered for this split

Both the size of the random subsample of training points and the number of features at each split are hyperparameters which should be tuned through cross-validation.