Chapter 13

Decision Trees

In the previous chapter, we introduced key concepts of information theory. In this chapter, we will see how information-theoretic notions can be used to derive a powerful form of supervised classification models, known as decision trees.

13.1 Using Binary Tests to Partition Data

In the first part of this course, we focused on methods that specify decision boundaries. For example, the goal of the perceptron algorithm was to specify a linear decision boundary that could correctly classify our data. One way of re-framing such decision boundaries is that they specify *tests* that we can run on our data in order to provide *information* about what class each point belongs to.

13.1.1 Binary tests

Suppose we have a simple binary classification dataset

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

with univariate (i.e., one-dimensional) features $x_i \in \mathbb{R}$. We can think of a linear decision boundary

$$wx + b > 0 \tag{13.1}$$

as a binary test $b(x)$ on these features

$$b(x) = \begin{cases} 1 & \text{if } wx > -b \\ 0 & \text{otherwise.} \end{cases} \tag{13.2}$$

In other words, we test the data to see if $wx > -b$. The result of these test is a binary value: we get a 1 if the test is successful and a 0 otherwise.

In general, we can define all kinds of binary tests on our data. For example, rather than a test based on a linear decision boundary, we might consider an even simpler threshold-based test

$$b(x) = \begin{cases} 1 & \text{if } x > \alpha \\ 0 & \text{otherwise.} \end{cases} \tag{13.3}$$

Alternatively, we could consider building test based on more complex functions, such as

$$b(x) = \begin{cases} 1 & \text{if } \log(x) > \alpha \\ 0 & \text{otherwise.} \end{cases} \tag{13.4}$$

The key idea is that these binary tests can be used to probe our dataset, e.g., by allowing us to find all datapoints that have features above a certain threshold.

13.1.2 Partitioning data via information gain

As with decision boundaries, we can use binary tests to partition our data. In particular, using a binary test $b(x)$, we can partition our data into two sets

$$\mathcal{B}_1 = \{(x, y) \in \mathcal{D} : b(x) = 1\} \qquad \mathcal{B}_0 = \{(x, y) \in \mathcal{D} : b(x) = 0\}. \tag{13.5}$$

Now, in a perfect world, such a partitioning could perfectly classify our data, i.e., we would have that $(x, y) \in \mathcal{B}_1 \Rightarrow y = 1$ and that $(x, y) \in \mathcal{B}_0 \Rightarrow y = 0$. In practice, however, we will rarely be able to find a simple binary test (e.g., a linear decision boundary) that will be able to perfectly classify our data. Thus, we need some way to choose a *good* test, even if it is imperfect.

One such way to choose a test is based on the notion of *information gain*. Informally, our goal will be to choose the binary test that gives us the most information about the label y. We can formalize this notion by defining the information gain (IG) of a binary test b:

$$IG(b) = KL\left(P(y|b(x))||P(y)\right) \tag{13.6}$$

$$= H(y) - H(y|b(x)). \tag{13.7}$$

Here, $P(y)$ denotes the prior distribution of the class labels (i.e., the class prior), while $P(y|b(x))$ is the conditional distribution of the class prior given the test.

Another way of interpreting the information gain is that it computes the difference between the entropy of the class prior $H(y)$ and the entropy of the conditional distribution $H(y|b(x)$. In this way, we can measure how many bits of information we gain by running a particular test of the data. For instance, if our test perfectly separates the data, then $H(y|b(x)) = 0$—since y variable can be perfectly predicted based on the test—and, in this case, the information gained would be equal to $H(y)$, i.e., the uncertainty in the original label distribution.

Example of information gain Suppose we had the following dataset

$$\mathcal{D} = \{(0.5, 1), (0.3, 1), (-1.1, 1), (-0.1, 0), (-0.3, 0), (0.2, 0)\}. \tag{13.8}$$

In this case, we would have that the prior distribution is equal to $P(y = 1) = 0.5$, since there are equal number of points from each class. The entropy of the class prior would thus be $H(y) = -\log(0.5) = 1$.

Now, suppose we run the following binary test on the data

$$b(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0. \end{cases} \tag{13.9}$$

In this case, we would end up with the following conditional entropy:

$$H(y|b(x)) = -\sum_{k\in\{0,1\}} \sum_{q\in\{0,1\}} P(y = k, b(x) = q) \log\left(P(y = k|b(x) = q)\right) \tag{13.10}$$

$$\begin{aligned} &= -P(y = 0, b(x) = 0) \log\left(P(y = 0|b(x) = 0)\right) \\ &\quad - P(y = 1, b(x) = 0) \log\left(P(y = 1|b(x) = 0)\right) \\ &\quad - P(y = 0, b(x) = 1) \log\left(P(y = 0|b(x) = 1)\right) \\ &\quad - P(y = 1, b(x) = 1) \log\left(P(y = 1|b(x) = 1)\right) \end{aligned}$$

$$= -\frac{2}{6}\log\left(\frac{2}{3}\right) - \frac{1}{6}\log\left(\frac{1}{3}\right) - \frac{1}{6}\log\left(\frac{1}{3}\right) - \frac{2}{6}\log\left(\frac{2}{3}\right) \tag{13.11}$$

$$\approx 0.92 \tag{13.12}$$

Thus, we would have an information gain of

$$IG(b) = 1 - 0.92 = 0.08 \tag{13.13}$$

from this test. In other words, we gain roughly 0.08 bits of information about the target label by running this test.

13.1.3 Generalizing to more complex features

In the above discussion, we assumed that our data was a simple univariate dataset, i.e., with one-dimensional real-valued features. However, the idea of binary tests can easily be generalized to handle general datasets with m-dimensional feature vectors $\mathbf{x}$. Typically, when designing tests for m-dimensional feature vectors, we will focus each test on a specific feature. For example, we might have a threshold-based test that focuses on the jth feature:

$$b(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}[j] > \alpha \\ 0 & \text{otherwise,} \end{cases} \tag{13.14}$$

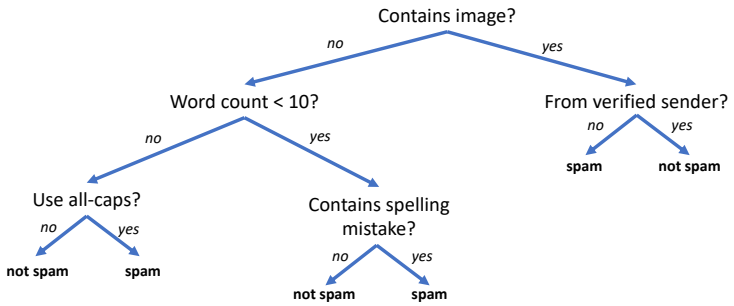

Figure 13.1: Simple example of a decision tree for spam email classification.

but we can also have tests that consider interactions between multiple features, e.g.,

$$b(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}[j]\mathbf{x}[k] > \alpha \\ 0 & \text{otherwise.} \end{cases} \tag{13.15}$$

Moreover, tests can easily be designed for binary and categorical features, in addition to real-valued features. Indeed, for binary features, one typically just tests if the feature is equal to zero or one.

13.2 From Binary Tests to Binary Decision Trees

The notion of information gain allows us to quantify how much information a binary test gives us about our target label. We can formalize this idea into a machine learning model, called *decision trees*.

The intuition behind binary decision trees is that we want to determine a set of binary tests that we can use to classify our data. The idea is that we start from the root of the binary decision tree, and we run our first test. Depending on the outcome of the first test, we then might run further tests until we can classify the data, with each test taking us down a different branch of the decision tree. An example of a decision tree for spam email classification is shown in Figure 13.1. The challenge of course is how we *learn* such a decision tree from data.

13.2.1 Learning a decision tree

One of the most popular strategies for learning a binary decision tree involves recursively finding the binary tests that provide the most information gain. This idea can be summarized via the following steps:

1. Generate a candidate set $\mathcal{S}$ of binary tests for dataset $\mathcal{D}$.
2. Select the test $b \in \mathcal{S}$ that gives the highest information gain.
3. Partition the dataset $\mathcal{D}$ into two sets $\mathcal{B}_0$ and $\mathcal{B}_1$ based on the test b.

4. Recurse and repeat the steps independently on the each of the two partitions $\mathcal{B}_0$ and $\mathcal{B}_1$.

The idea is that we choose the best test based on information gain, use this test to partition our data, and then recursively search for the best tests within each of the newly generated partitions. This recursive process naturally generates a binary tree of tests.

Typically, the learning process stops once our partitions at the leaves of the binary tree are *pure*, i.e., once the data is perfectly separated at the leaves of the binary tree. However, one can also stop the learning process early; in that case, the majority vote function can be used to classify the examples at the leaves of the tree.

13.2.2 Selecting candidate tests

One of the key steps in the decision tree algorithm is selecting the set of candidate tests to run over the data. In general, there can be infinitely many possible tests that we could run on any given dataset, so we must have a strategy for generating a finite number of candidate tests. The way that candidate tests are generated is one of the key distinguishing factors between different decision tree algorithms. Typically, efficient decision tree algorithms only consider single features for each test and limit themselves to threshold-based tests for real-valued features, using a finite set of possible thresholds.

13.2.3 The decision boundary of decision trees

Like all the machine learning methods discussed in this course, decision trees can be interpreted based on the decision boundaries that they specify. Unlike the linear methods that we previously introduced, decision trees do not specify a single decision boundary. Instead, decision trees induce a partitioning of the input space into a set of distinct regions. An example of such a partitioning is shown in Figure 13.2.

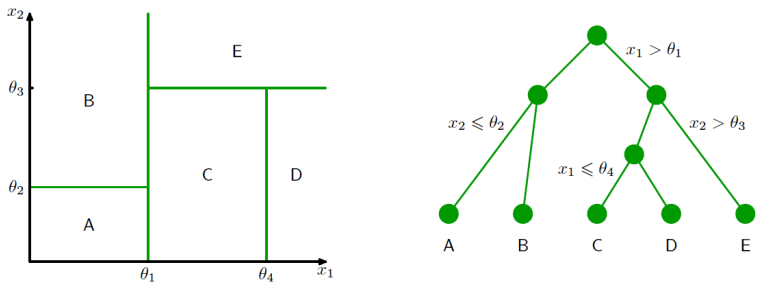

Figure 13.2: Partitioning induced by a decision tree.

13.2.4 More general decision trees

In this chapter, we introduced basic binary decision trees, which are learned via information gain. It is important to note, however, that there are many variants of decision trees in the literature, and not all decision trees use information gain as the metric to select tests. We will not cover all such variants in this course. In general, decision tree approaches use the general recursive learning strategies introduced here, but there is much room for diversity, including approaches that generate random candidate tests (e.g., Random Forests).