FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS

Algorithms, Worked Examples, and Case Studies

John D. Kelleher
Brian Mac Namee
Aoife D'Arcy

The MIT Press
Cambridge, Massachusetts
London, England

This is an excerpt from the book **Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies** by John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy published by The MIT Press in 2015.

Machine learning is often used to build predictive models by extracting patterns from large datasets. These models are used in predictive data analytics applications including price prediction, risk assessment, predicting customer behavior, and document classification. This introductory textbook offers a detailed and focused treatment of the most important machine learning approaches used in predictive data analytics, covering both theoretical concepts and practical applications. Technical and mathematical material is augmented with explanatory worked examples, and case studies illustrate the application of these models in the broader business context.

After discussing the trajectory from data to insight to decision, the book describes four approaches to machine learning: information-based learning, similarity-based learning, probability-based learning, and error-based learning. Each of these approaches is introduced by a nontechnical explanation of the underlying concept, followed by mathematical models and algorithms illustrated by detailed worked examples. Finally, the book considers techniques for evaluating prediction models and offers two case studies that describe specific data analytics projects through each phase of development, from formulating the business problem to implementation of the analytics solution. The book, informed by the authors many years of teaching machine learning, and working on predictive data analytics projects, is suitable for use by undergraduates in computer science, engineering, mathematics, or statistics; by graduate students in disciplines with applications for predictive data analytics; and as a reference for professionals.

This extract is the first part of a chapter on **Information-based Learning**. In this extract a nontechnical explanation of the concepts underlying information-based learning are followed by the fundamental mathematical ideas used and the standard approach to using information theory to build predictive models.

4 Information-based Learning

Information is the resolution of uncertainty.
—Claude Elwood Shannon

In this chapter we discuss the ways in which concepts from **information theory** can be used to build prediction models. We start by discussing **decision trees**, the fundamental structure used in information-based machine learning, before presenting the fundamental measures of information content that are used: **entropy** and **information gain**. We then present the **ID3** algorithm, the standard algorithm used to induce a decision tree from a dataset. The extensions and variations to this standard approach that we present describe how different data types can be handled, how overfitting can be avoided using decision tree **pruning**, and how multiple prediction models can be combined in **ensembles** to improve prediction accuracy.

4.1 Big Idea

We'll start off by playing a game. *Guess Who* is a two-player game in which one player chooses a card with a picture of a character on it from a deck and the other player tries to guess which character is on the card by asking a series of questions to which the answer can only be yes or no. The player asking the questions wins by guessing who is on the card within a small number of questions and loses otherwise. Figure 4.1[2] shows the set of cards that we will use for our game. We can represent these cards using the dataset given in Table 4.1[2].

Now, imagine that we have picked one of these cards and you have to guess which one by asking questions. Which of the following questions would you ask first?

1. Is it a man?
2. Does the person wear glasses?

Most people would ask Question 1 first. Why is this? At first, this choice of question might seem ineffective. For example, if you ask Question 2, and we answer *yes*, you can be sure that we have picked *Brian* without asking any more questions. The problem with this reasoning, however, is that, on average, the answer to Question 2 will be *yes* only one out of every four times you play. That means that three out of every four times you ask Question 2, the answer

Figure 4.1
Cards showing character faces and names for the *Guess Who* game.

Table 4.1
A dataset that represents the characters in the *Guess Who* game.

Man	Long Hair	Glasses	Name
Yes	No	Yes	Brian
Yes	No	No	John
No	Yes	No	Aphra
No	No	No	Aoife

will be *no*, and you will still have to distinguish between the three remaining characters.

Figure 4.2[3] illustrates the possible question sequences that can follow in a game beginning with Question 2. In Figure 4.2(a)[3] we next ask, *Is it a man?* and then, if required, *Do they have long hair?* In Figure 4.2(b)[3] we reverse this order. In both of these diagrams, one path to an answer about the character on a card is 1 question long, one path is 2 questions long, and two paths are 3 questions long. Consequently, if you ask Question 2 first, the average number of questions you have to ask per game is

$$\frac{1+2+3+3}{4} = 2.25$$

On the other hand, if you ask Question 1 first, there is only one sequence of questions with which to follow it. This sequence is shown in Figure 4.3[4]. Irrespective of the answers to the questions, you always have to follow a path through this sequence that is 2 questions long to reach an answer about the character on a card. This means that if you always ask Question 1 first, the average number of questions you have to ask per game is

$$\frac{2+2+2+2}{4} = 2$$

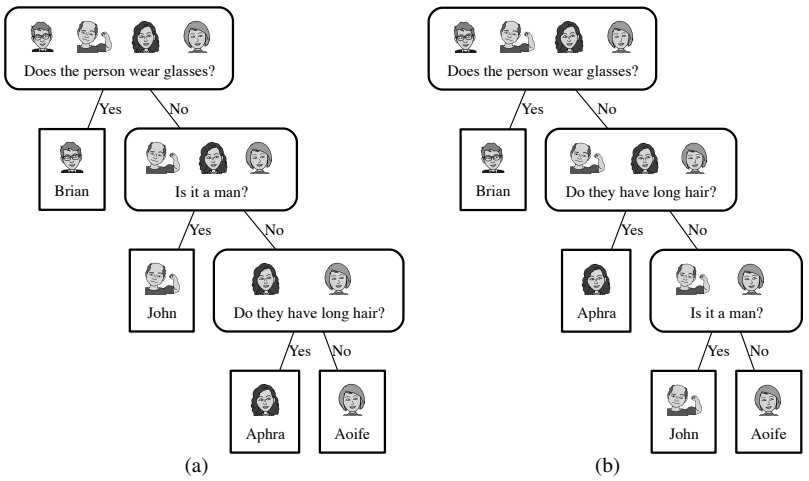


Figure 4.2

The different question sequences that can follow in a game of *Guess Who* beginning with the question *Does the person wear glasses?*

What is interesting here is that no matter what question you ask, the answer is always either *yes* or *no*, but, on average, an answer to Question 1 seems to carry more information than an answer to Question 2. This is not because of the literal message in the answer (either *yes* or *no*). Rather, it is because of the way that the answer to each question splits the character cards into different sets based on the value of the descriptive feature the question is asked about (MAN, LONG HAIR or GLASSES) and the likelihood of each possible answer to the question.

An answer to Question 1, *Is it a man?*, splits the game domain into two sets of equal size: one containing *Brian* and *John* and one containing *Aphra* and *Aoife*. One of these sets contains the solution, which leaves you with just one more question to ask to finish the game. By contrast, an answer to Question 2 splits the game domain into one set containing one element, *Brian*, and another set containing three elements: *John*, *Aphra*, and *Aoife*. This works out really well when the set containing the single element contains the solution. In the more likely case that the set containing three elements contains the solution, however, you may have to ask two more questions to uniquely identify the answer. So, when you consider both the likelihood of an answer and how an answer splits up the domain of solutions, it becomes clear that an answer to

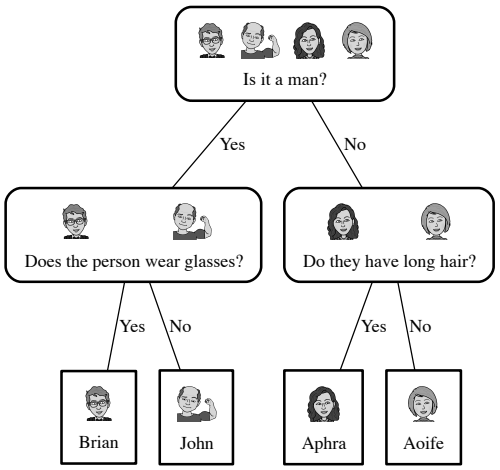


Figure 4.3

The different question sequences that can follow in a game of *Guess Who* beginning with the question *Is it a man?*

Question 2 leaves you with more work to do to solve the game than an answer to Question 1.

So, the big idea here is to figure out which features are the most informative ones to ask questions about by considering the effects of the different answers to the questions, in terms of how the domain is split up after the answer is received and the likelihood of each of the answers. Somewhat surprisingly, people seem to be able to easily do this based on intuition. Information-based machine learning algorithms use the same idea. These algorithms determine which descriptive features provide the most information about a target feature and make predictions by sequentially testing the features in order of their informativeness.

4.2 Fundamentals

In this section we introduce Claude Shannon's approach to measuring information,[1] in particular his model of **entropy** and how it is used in the **information**

1 Claude Shannon is considered to be the father of information theory. Shannon worked for AT&T Bell Labs, where he worked on the efficient encoding of messages for telephone communication. It

Table 4.2
An email spam prediction dataset.

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

gain measure to capture the informativeness of a descriptive feature. Before this we introduce **decision trees**, the actual prediction models that we are trying to build.

4.2.1 Decision Trees

Just as we did when we played *Guess Who*, an effective way to generate a prediction is to carry out a series of tests on the values of the descriptive features describing a query instance, and use the answers to these tests to determine the prediction. **Decision trees** take this approach. To illustrate how a decision tree works, we will use the dataset listed in Table 4.2[5]. This dataset contains a set of training instances that can be used to build a model to predict whether emails are *spam* or *ham* (genuine). The dataset has three binary descriptive features: SUSPICIOUS WORDS is *true* if an email contains one or more words that are typically found in spam email (e.g., *casino*, *viagra*, *bank*, or *account*); UNKNOWN SENDER is *true* if the email is from an address that is not listed in the contacts of the person who received the email; and CONTAINS IMAGES is *true* if the email contains one or more images.

Figure 4.4[6] shows two decision trees that are **consistent** with the spam dataset. Decision trees look very like the game trees that we developed for the *Guess Who* game. As with all tree representations, a decision tree consists of a **root node** (or starting node), **interior nodes**, and **leaf nodes** (or terminating nodes) that are connected by **branches**. Each non-leaf node (root and interior)

was this focus on encoding that motivated his approach to measuring information. In information theory, the meaning of the word **information** deliberately excludes the psychological aspects of the communication and should be understood as measuring the optimal encoding length of a message given the set of possible messages that could be sent within the communication.

in the tree specifies a test to be carried out on a descriptive feature. The number of possible levels that a descriptive feature can take determines the number of downward branches from a non-leaf node. Each of the leaf nodes specifies a predicted level of the target feature.

In the diagrams in Figure 4.4[6], ellipses represent root or interior nodes, and rectangles represent leaf nodes. The labels of the ellipses indicate which descriptive feature is tested at that node. The labels on the each branch indicate one of the possible feature levels that the descriptive feature at the node above can take. The labels on the rectangular leaf nodes indicate the target level that should be predicted when the tests on the interior nodes create a path that terminates at that leaf node.

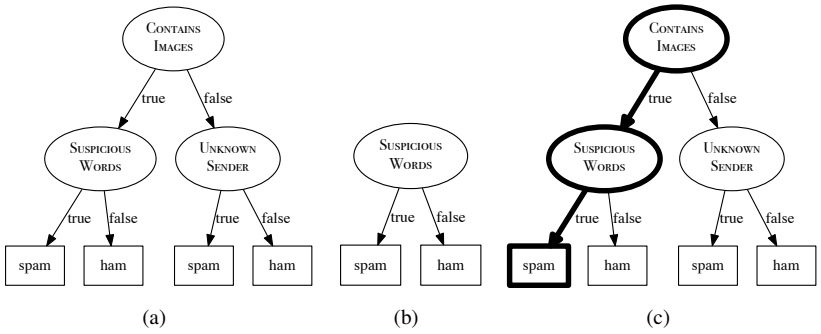


Figure 4.4

Two decision trees (a) and (b) that are consistent with the instances in the spam dataset; (c) the path taken through the tree shown in (a) to make a prediction for the query instance SUSPICIOUS WORDS = *true*, UNKNOWN SENDER = *true*, CONTAINS IMAGES = *true*.

The process of using a decision tree to make a prediction for a query instance starts by testing the value of the descriptive feature at the root node of the tree. The result of this test determines which of the root node's children the process should then descend to. These two steps of testing the value of a descriptive feature and descending a level in the tree are then repeated until the process comes to a leaf node at which a prediction can be made.

To demonstrate how this process works, imagine we were given the query email SUSPICIOUS WORDS = *true*, UNKNOWN SENDER = *true*, CONTAINS IMAGES = *true*, and asked to predict whether it is *spam* or *ham*. Applying the decision tree from Figure 4.4(a)[6] to this query, we see that the root node of this tree tests the CONTAINS IMAGES feature. The query instance value for CONTAINS IMAGES is *true* so the process descends the left branch from the

root node, labeled *true*, to an interior node that tests the SUSPICIOUS WORDS feature. The query instance value for this feature is *true*, so based on the result of the test at this node, the process descends the left branch, labeled *true*, to a leaf node labeled *spam*. As the process has arrived at a leaf node, it terminates, and the target level indicated by the leaf node, *spam*, is predicted for the query instance. The path through the decision tree for this query instance is shown in figure 4.4(c)[6].

The decision tree in Figure 4.4(b)[6] would have returned the same prediction for the query instance. Indeed, both of the decision trees in Figures 4.4(a)[6] and 4.4(b)[6] are consistent with the dataset in Table 4.2[5] and can generalize sufficiently to make predictions for query instances like the one considered in our example. The fact that there are, at least, two decision trees that can do this raises the question: How do we decide which is the best decision tree to use?

We can apply almost the same approach that we used in the *Guess Who* game to make this decision. Looking at the decision trees in Figures 4.4(a)[6] and 4.4(b)[6], we notice that the tree in Figure 4.4(a)[6] performs tests on two features in order to make a prediction, while the decision tree in Figure 4.4(b)[6] only ever needs to test the value of one feature. The reason for this is that SUSPICIOUS WORDS, the descriptive feature tested at the root node of the tree in Figure 4.4(b)[6], perfectly splits the data into a pure group of *spam* emails and a pure group of *ham* emails. We can say that because of the purity of the splits that it makes, the SUSPICIOUS WORDS feature provides more information about the value of the target feature for an instance than the CONTAINS IMAGES feature, so a tree that tests this descriptive feature at the root node is preferable.

This gives us a way to choose between a set of different decision trees that are all consistent with a set of training instances. We can introduce a preference for decision trees that use fewer tests, in other words, trees that are on average shallower.[2] This is the primary inductive bias that a machine learning algorithm taking an information-based approach encodes. To build shallow trees, we need to put the descriptive features that best discriminate between instances that have different target feature values toward the top of the tree. To do this we need a formal measure of how well a descriptive feature discriminates between

2 In fact, it can be argued that a preference toward shallower decision trees is a good idea in general and can be viewed as following **Occam's razor**. Occam's razor is the principle of keeping theories as simple as possible. It is named after a fourteenth century Franciscan monk, William of Occam (sometimes spelled Ockham), who was one of the first to formulate this principle. The *razor* in the title comes from the idea of shaving off any unnecessary assumptions from a theory.

the levels of the target feature. Similar to the way we analyzed the questions in the *Guess Who* game, we will measure the discriminatory power of a descriptive feature by analyzing the size and probability of each set of instances created when we test the value of the feature and how pure each set of instances is with respect to the target feature values of the instances it contains. The formal measure we will use to do this is Shannon's entropy model.

4.2.2 Shannon's Entropy Model

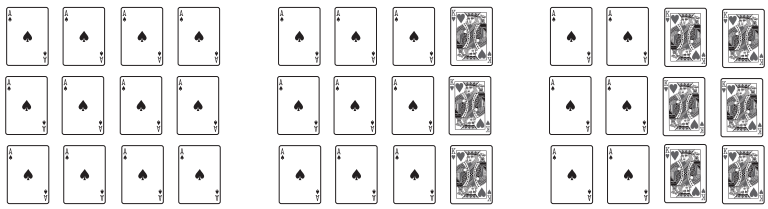
Claude Shannon's entropy model defines a computational measure of the impurity of the elements in a set. Before we examine the mathematical definition of entropy, we will first provide an intuitive explanation of what it means. Figure 4.5[9] illustrates a collection of sets of playing cards of contrasting entropy. An easy way to understand the entropy of a set is to think in terms of the uncertainty associated with guessing the result if you were to make a random selection from the set. For example, if you were to randomly select a card from the set in Figure 4.5(a)[9], you would have zero uncertainty, as you would know for sure that you would select an ace of spades. So, this set has zero entropy. If, however, you were to randomly select an element from the set in Figure 4.5(f)[9], you would be very uncertain about any prediction as there are twelve possible outcomes, each of which is equally likely. This is why this set has very high entropy. The other sets in Figure 4.5[9] have entropy values between these two extremes.

This gives us a clue as to how we should define a computational model of entropy. We can transform the probabilities[3] of the different possible outcomes when we randomly select an element from a set to entropy values. An outcome with a large probability should map to a low entropy value, while an outcome with a small probability should map to a large entropy value. The mathematical **logarithm**, or **log**, function[4] does almost exactly the transformation that we need.

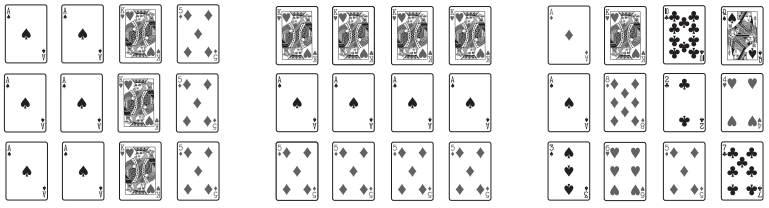
If we examine the graph of the **binary logarithm** (a logarithm to the base 2) of probabilities ranging from 0 to 1 in Figure 4.6(a)[10], we see that the logarithm function returns large negative numbers for low probabilities, and small

3 We use some simple elements of probability theory in this chapter. Readers unfamiliar with the way probabilities are calculated based on the relative frequencies of events should read the first section of Appendix **??**[??] before continuing with this chapter.

4 The *log* of a to the base b, written as $log_b(a)$, is the number to which we must raise b to get a. For example, $log_2(8) = 3$ because $2^3 = 8$ and $log_5(625) = 4$ because $5^4 = 625$.



(a) $H(card) = 0.00$ (b) $H(card) = 0.81$ (c) $H(card) = 1.00$



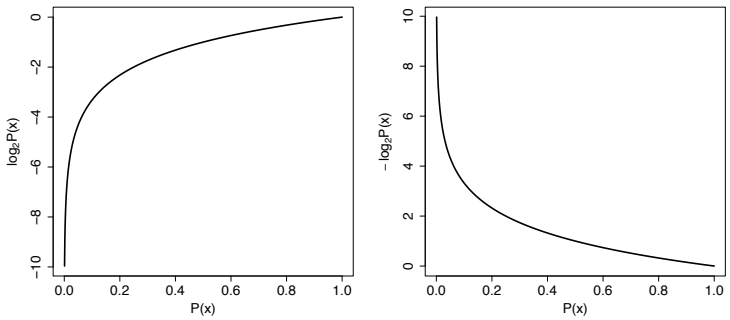
(d) $H(card) = 1.50$ (e) $H(card) = 1.58$ (f) $H(card) = 3.58$

Figure 4.5
The entropy of different sets of playing cards measured in bits.

negative numbers for high probabilities. Ignoring the fact that the logarithm function returns negative numbers, the magnitude of the numbers it returns is ideal as a measure of entropy: large numbers for low probabilities and small numbers (near zero) for high probabilities. It should also be noted that the range of values for the binary logarithm of a probability, $[-\infty, 0]$, is much larger than those taken by the probability itself $[0, 1]$. This is also an attractive characteristic of this function. It will be more convenient for us to convert the output of the log function to positive numbers by multiplying them by -1. Figure 4.6(b)[10] shows the impact of this.

Shannon's model of entropy is a weighted sum of the logs of the probabilities of each possible outcome when we make a random selection from a set. The weights used in the sum are the probabilities of the outcomes themselves so that outcomes with high probabilities contribute more to the overall entropy of a set than outcomes with low probabilities. Shannon's model of entropy is defined as

$$H(t) = -\sum_{i=1}^{l} (P(t = i) \times log_s(P(t = i))) \tag{4.1}$$



(a) $P(x)$ and $log_2 P(x)$ (b) $P(x)$ and $-log_2 P(x)$

Figure 4.6

(a) A graph illustrating how the value of a binary log (the log to the base 2) of a probability changes across the range of probability values; (b) the impact of multiplying these values by -1.

where $P(t = i)$ is the probability that the outcome of randomly selecting an element t is the type i, l is the number of different types of things in the set, and s is an arbitrary logarithmic base. The minus sign at the beginning of the equation is simply added to convert the negative numbers returned by the log function to positive ones (as described above). We will always use 2 as the base, s, when we calculate entropy, which means that we measure entropy in **bits**.[5] Equation (4.1)[9] is the cornerstone of modern **information theory** and is an excellent measure of the impurity, **heterogeneity**, of a set.

To understand how Shannon's entropy model works, consider the example of a set of 52 different playing cards. The probability of randomly selecting any specific card i from this set, $P(card = i)$, is quite low, just $\frac{1}{52}$. The entropy

5 Using binary logs, the maximum entropy for a set with two types of elements is 1.00 bit, but the entropy for a set with more than two types of elements may be greater than 1.00 bit. The choice of base when using Shannon's model in the context that it will be used later in this chapter is arbitrary. The choice of base 2 is partly due to a conventional computer science background and partly because it allows us to use the *bits* unit of information.

of the set of 52 playing cards is calculated as

$$\begin{aligned} H(card) &= -\sum_{i=1}^{52} P(card = i) \times log_2(P(card = i)) \\ &= -\sum_{i=1}^{52} 0.019 \times log_2(0.019) \\ &= -\sum_{i=1}^{52} -0.1096 \\ &= 5.700 \; bits \end{aligned}$$

In this calculation, for each possible card Shannon's model multiplies a small probability, $P(card) = i$, by a large negative number, $log_2(P(card) = i)$, resulting in a relatively large negative number. The individual relatively large negative numbers calculated for each card are then summed to return one large negative number. The sign of this is inverted to give a large positive value for the entropy of this very impure set.

By contrast, consider the example of calculating the entropy of a set of 52 playing cards if we only distinguish between cards based on their suit (hearts ♥, clubs ♣, diamonds ♦ or, spades ♠). This time there are only 4 possible outcomes when a random card is selected from this set, each with a reasonably large probability of $\frac{13}{52}$. The entropy associated with this set can be calculated as

$$\begin{aligned} H(suit) = & -\sum_{l \in \{\heartsuit, \clubsuit, \diamondsuit, \spadesuit\}} P(suit = l) \times log_2(P(suit = l)) \\ = & -\big((P(\heartsuit) \times log_2(P(\heartsuit))) + (P(\clubsuit) \times log_2(P(\clubsuit))) \\ & + (P(\diamondsuit) \times log_2(P(\diamondsuit))) + (P(\spadesuit) \times log_2(P(\spadesuit))) \big) \\ = & -\Big(\Big({}^{13}/_{52} \times log_2({}^{13}/_{52}) \Big) + \Big({}^{13}/_{52} \times log_2({}^{13}/_{52}) \Big) \\ & + \Big({}^{13}/_{52} \times log_2({}^{13}/_{52}) \Big) + \Big({}^{13}/_{52} \times log_2({}^{13}/_{52}) \Big) \Big) \\ = & -((0.25 \times -2) + (0.25 \times -2) + (0.25 \times -2) + (0.25 \times -2)) \\ = & \; 2 \; bits \end{aligned}$$

In this calculation Shannon's model multiples the large probability of selecting a specific suit, $P(suit = l)$, by a small negative number, $log_2(P(suit = l))$, to return a relatively small negative number. The relatively small negative numbers associated with each suit are summed to result in a small negative number

overall. Again, the sign of this number is inverted to result in a small positive value for the entropy of this much purer set.

To further explore the entropy, we can return to look at the entropy values of each set of cards shown in Figure 4.5[9]. In the set in Figure 4.5(a)[9], all the cards are identical. This means that there is no uncertainty as to the result when a selection is made from this set. Shannon's model of information is designed to reflect this intuition, and the entropy value for this set is 0.00 bits. In the sets in Figures 4.5(b)[9] and 4.5(c)[9], there is a mixture of two different types of cards, so these have higher entropy values, in these instances, 0.81 bits and 1.00 bit. The maximum entropy for a set with two types of elements is 1.00 bit, which occurs when there are equal numbers of each type in the set.

The sets in Figures 4.5(d)[9] and 4.5(e)[9] both have three types of cards. The maximum entropy for sets with three elements is 1.58 and occurs when there are equal numbers of each type in the set, as is the case in Figure 4.5(e)[9]. In Figure 4.5(d)[9] one card type is more present than the others, so the overall entropy is slightly lower, 1.50 bits. Finally, the set in Figure 4.5(f)[9] has a large number of card types, each represented only once, which leads to the high entropy value of 3.58 bits.

This discussion highlights the fact that entropy is essentially a measure of the heterogeneity of a set. As the composition of the sets changed from the set with only one type of element (Figure 4.5(a)[9]) to a set with many different types of elements each with an equal likelihood of being selected (Figure 4.5(f)[9]), the entropy score for the sets increased.

4.2.3 Information Gain

What is the relationship between a measure of heterogeneity of a set and predictive analytics? If we can construct a sequence of tests that splits the training data into pure sets with respect to the target feature values, then we can label queries by applying the same sequence of tests to a query and labeling it with the target feature value of instances in the set it ends up in.

To illustrate this we'll return to the spam dataset from Table 4.2[5]. Figure 4.7[13] shows how the instances in the spam dataset are split when we partition it using each of the three descriptive features. Looking at 4.7(a)[13], we can see that splitting the dataset based on the SUSPICIOUS WORDS feature provides a lot of information about whether an email is spam or ham. In fact, partitioning the data by this feature creates two pure sets: one containing only instances with the target level *spam* and the other set containing only instances with the

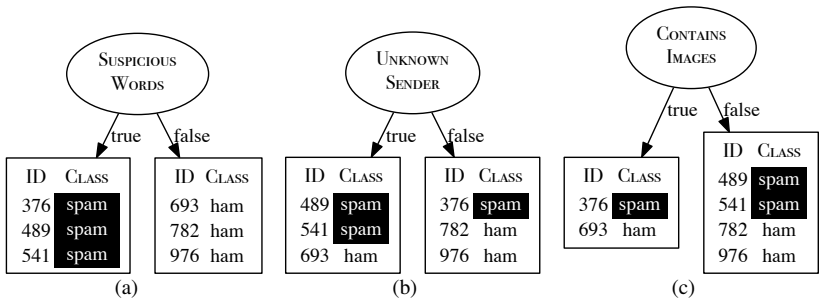


Figure 4.7

How the instances in the spam dataset split when we partition using each of the different descriptive features from the spam dataset in Table 4.2[5].

target level *ham*. This indicates that the SUSPICIOUS WORDS feature is a good feature to test if we are trying to decide whether a new email—not listed in the training dataset—is spam or not.

What about the other features? Figure 4.7(b)[13] shows how the UNKNOWN SENDER feature partitions the dataset. The resulting sets both contain a mixture of *spam* and *ham* instances. This indicates that the UNKNOWN SENDER feature is not as good at discriminating between spam and ham emails as the SUSPICIOUS WORDS feature. Although there is a mixture in each of these sets, however, it seems to be the case that when UNKNOWN SENDER = *true*, the majority of emails are *spam*, and when UNKNOWN SENDER = *false*, the majority of emails are *ham*. So although this feature doesn't perfectly discriminate between spam and ham it does give us some information that we might be able to use in conjunction with other features to help decide whether a new email is spam or ham. Finally, if we examine the partitioning of the dataset based on the CONTAINS IMAGES feature, Figure 4.7(c)[13], it looks like this feature is not very discriminatory for *spam* and *ham* at all. Both of the resulting sets contain a balanced mixture of spam and ham instances.

What we need to do now is to develop a formal model that captures the intuitions about the informativeness of these features described above. Unsurprisingly, we do this using Shannon's entropy model. The measure of informativeness that we will use is known as **information gain** and is a measure of the reduction in the overall entropy of a set of instances that is achieved by testing on a descriptive feature. Computing information gain is a three-step process:

1. Compute the entropy of the original dataset with respect to the target feature. This gives us an measure of how much information is required in order to organize the dataset into pure sets.
2. For each descriptive feature, create the sets that result by partitioning the instances in the dataset using their feature values, and then sum the entropy scores of each of these sets. This gives a measure of the information that remains required to organize the instances into pure sets after we have split them using the descriptive feature.
3. Subtract the remaining entropy value (computed in step 2) from the original entropy value (computed in step 1) to give the information gain.

We need to define three equations to formally specify information gain (one for each step). The first equation calculates the entropy for a dataset with respect to a target feature[6]

$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} (P(t = l) \times log_2(P(t = l))) \tag{4.2}$$

where $levels(t)$ is the set of levels in the domain of the target feature t, and $P(t = l)$ is the probability of a randomly selected instance having the target feature level l.

The second equation defines how we compute the entropy remaining after we partition the dataset using a particular descriptive feature d. When we partition the dataset $\mathcal{D}$ using the descriptive feature d we create a number of partitions (or sets) $\mathcal{D}_{d=l_1} \ldots \mathcal{D}_{d=l_k}$, where $l_1 \ldots l_k$ are the k levels that feature d can take. Each partition, $\mathcal{D}_{d=l_i}$, contains the instances in $\mathcal{D}$ that have a value of level l_i for the d feature. The entropy remaining after we have tested d is a weighted sum of the entropy, still with respect to the target feature, of each partition. The weighting is determined by the size of each partition—so a large partition should contribute more to the overall remaining entropy than a smaller partition. We use the term $rem(d, \mathcal{D})$ to denote this quantity and define it formally as

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \underbrace{\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|}}_{\text{weighting}} \times \underbrace{H(t, \mathcal{D}_{d=l})}_{\substack{\text{entropy of} \\ \text{partition } \mathcal{D}_{d=l}}} \tag{4.3}$$

6 This is almost identical to the definition of Shannon's entropy model given in Equation (4.1)[9]. We have extended the definition to include an explicit parameter for the dataset $\mathcal{D}$ for which we are computing the entropy, and we have specified the base as 2.

Using Equation (4.2)[14] and Equation (4.3)[14], we can now formally define information gain made from splitting the dataset $\mathcal{D}$ using the feature d as

$$IG(d,\mathcal{D}) = H(t,\mathcal{D}) - rem(d,\mathcal{D}) \tag{4.4}$$

To illustrate how information gain is calculated, and to check how well it models our intuitions described at the beginning of this section, we will compute the information gain for each descriptive feature in the spam dataset. The first step is to compute the entropy for the whole dataset using Equation (4.2)[14]:

$$\begin{aligned} H(t,\mathcal{D}) = & -\sum_{l\in\{spam,ham\}} (P(t=l)\times log_2(P(t=l))) \\ = & -\big((P(t=spam)\times log_2(P(t=spam)) \\ & \quad + (P(t=ham)\times log_2(P(t=ham))\big) \\ = & -\left(\left({}^3\!/_6 \times log_2({}^3\!/_6)\right) + \left({}^3\!/_6 \times log_2({}^3\!/_6)\right)\right) \\ = & \ 1\ bit \end{aligned}$$

The next step is to compute the entropy remaining after we split the dataset using each of the descriptive features. The computation for the SUSPICIOUS WORDS feature is[7]

$$\begin{aligned} & rem(\text{WORDS},\mathcal{D}) \\ & = \left(\frac{|\mathcal{D}_{\text{WORDS}=true}|}{|\mathcal{D}|}\times H(t,\mathcal{D}_{\text{WORDS}=true})\right) \\ & \quad + \left(\frac{|\mathcal{D}_{\text{WORDS}=false}|}{|\mathcal{D}|}\times H\left(t,\mathcal{D}_{\text{WORDS}=false}\right)\right) \\ & = \left({}^3\!/_6 \times \left(-\sum_{l\in\{spam,ham\}} P(t=l)\times log_2(P(t=l))\right)\right) \\ & \quad + \left({}^3\!/_6 \times \left(-\sum_{l\in\{spam,ham\}} P(t=l)\times log_2(P(t=l))\right)\right) \\ & = \left({}^3\!/_6 \times \left(-\left(\left({}^3\!/_3 \times log_2({}^3\!/_3)\right) + \left({}^0\!/_3 \times log_2({}^0\!/_3)\right)\right)\right)\right) \\ & \quad + \left({}^3\!/_6 \times \left(-\left(\left({}^0\!/_3 \times log_2({}^0\!/_3)\right) + \left({}^3\!/_3 \times log_2({}^3\!/_3)\right)\right)\right)\right) \\ & = 0\ bits \end{aligned}$$

7 Note that we have shortened feature names in these calculations to save space.

The remaining entropy for the UNKNOWN SENDER feature is

$$
\begin{aligned}
&rem\,(\text{SENDER}, \mathcal{D}) \\
&\quad = \left(\frac{|\mathcal{D}_{\text{SENDER}=true}|}{|\mathcal{D}|} \times H\left(t, \mathcal{D}_{\text{SENDER}=true}\right) \right) \\
&\qquad + \left(\frac{|\mathcal{D}_{\text{SENDER}=false}|}{|\mathcal{D}|} \times H\left(t, \mathcal{D}_{\text{SENDER}=false}\right) \right) \\
&\quad = \left({}^{3}/_{6} \times \left(- \sum_{l \in \{spam, ham\}} P(t=l) \times log_2(P(t=l)) \right) \right) \\
&\qquad + \left({}^{3}/_{6} \times \left(- \sum_{l \in \{spam, ham\}} P(t=l) \times log_2(P(t=l)) \right) \right) \\
&\quad = \left({}^{3}/_{6} \times \left(- \left(\left({}^{2}/_{3} \times log_2({}^{2}/_{3}) \right) + \left({}^{1}/_{3} \times log_2({}^{1}/_{3}) \right) \right) \right) \right) \\
&\qquad + \left({}^{3}/_{6} \times \left(- \left(\left({}^{1}/_{3} \times log_2({}^{1}/_{3}) \right) + \left({}^{2}/_{3} \times log_2({}^{2}/_{3}) \right) \right) \right) \right) \\
&\quad = 0.9183\ bits
\end{aligned}
$$

The remaining entropy for the CONTAINS IMAGES feature is

$$
\begin{aligned}
&rem\,(\text{IMAGES}, \mathcal{D}) \\
&\quad = \left(\frac{|\mathcal{D}_{\text{IMAGES}=true}|}{|\mathcal{D}|} \times H\left(t, \mathcal{D}_{\text{IMAGES}=true}\right) \right) \\
&\qquad + \left(\frac{|\mathcal{D}_{\text{IMAGES}=false}|}{|\mathcal{D}|} \times H\left(t, \mathcal{D}_{\text{IMAGES}=false}\right) \right) \\
&\quad = \left({}^{2}/_{6} \times \left(- \sum_{l \in \{spam, ham\}} P(t=l) \times log_2(P(t=l)) \right) \right) \\
&\qquad + \left({}^{4}/_{6} \times \left(- \sum_{l \in \{spam, ham\}} P(t=l) \times log_2(P(t=l)) \right) \right) \\
&\quad = \left({}^{2}/_{6} \times \left(- \left(\left({}^{1}/_{2} \times log_2({}^{1}/_{2}) \right) + \left({}^{1}/_{2} \times log_2({}^{1}/_{2}) \right) \right) \right) \right) \\
&\qquad + \left({}^{4}/_{6} \times \left(- \left(\left({}^{2}/_{4} \times log_2({}^{2}/_{4}) \right) + \left({}^{2}/_{4} \times log_2({}^{2}/_{4}) \right) \right) \right) \right) \\
&\quad = 1\ bit
\end{aligned}
$$

We can now complete the information gain calculation for each descriptive feature as

$$\begin{aligned} IG(\text{WORDS}, \mathcal{D}) &= H(\text{CLASS}, \mathcal{D}) - rem(\text{WORDS}, \mathcal{D}) \\ &= 1 - 0 \\ &= 1 \textit{ bit} \\ IG(\text{SENDER}, \mathcal{D}) &= H(\text{CLASS}, \mathcal{D}) - rem(\text{SENDER}, \mathcal{D}) \\ &= 1 - 0.9183 \\ &= 0.0817 \textit{ bits} \\ IG(\text{IMAGES}, \mathcal{D}) &= H(\text{CLASS}, \mathcal{D}) - rem(\text{IMAGES}, \mathcal{D}) \\ &= 1 - 1 \\ &= 0 \textit{ bits} \end{aligned}$$

The information gain of the SUSPICIOUS WORDS feature is 1 bit. This is equivalent to the total entropy for the entire dataset. An information gain score for a feature that matches the entropy for the entire dataset indicates that the feature is perfectly discriminatory with respect to the target feature values. Unfortunately, in more realistic datasets, finding a feature as powerful as the SUSPICIOUS WORDS feature is very rare. The feature UNKNOWN SENDER has an information gain of 0.0817 bits. An information gain score this low suggests that although splitting on this feature provides some information, it is not particularly useful. Finally, the CONTAINS IMAGES feature has an information gain score of 0 bits. This ranking of the features by information gain mirrors the intuitions we developed about the usefulness of these features during our earlier discussion.

We started this section with the idea that if we could construct a sequence of tests that splits the training data into pure sets with respect to the target feature values, then we can do prediction by applying the same sequence of tests to the prediction queries and labeling them with the target feature of the set they end up in. A key part of doing this is being able to decide which tests should be included in the sequence and in what order. The information gain model we have developed allows us to decide which test we should add to the sequence next because it enables us to select the best feature to use on a given dataset. In the next section, we introduce the standard algorithm for growing decision trees in this way.

4.3 Standard Approach: The ID3 Algorithm

Assuming that we want to use shallow decision trees, is there a way in which we can automatically create them from data? One of the best known decision tree induction algorithms is the **Iterative Dichotomizer 3** (**ID3**) algorithm.[8] This algorithm attempts to create the shallowest decision tree that is consistent with the data given.

The ID3 algorithm builds the tree in a recursive, depth-first manner, beginning at the root node and working down to the leaf nodes. The algorithm begins by choosing the best descriptive feature to test (i.e., the best question to ask first). This choice is made by computing the **information gain** of the descriptive features in the training dataset. A root node is then added to the tree and labeled with the selected test feature. The training dataset is then partitioned using the test. There is one partition created for each possible test result, which contains the training instances that returned that result. For each partition a branch is grown from the node. The process is then repeated for each branch using the relevant partition of the training set in place of the full training set and with the selected test feature excluded from further testing. This process is repeated until all the instances in a partition have the same target level, at which point a leaf node is created and labeled with that level.

The design of the ID3 algorithm is based on the assumption that a correct decision tree for a domain will classify instances from that domain in the same proportion as the target level occurs in the domain. So, given a dataset $\mathcal{D}$ representing a domain with two target levels C_1 and C_2, an arbitrary instance from the domain should be classified as being associated with target level C_1 with the probability $\frac{|C_1|}{|C_1|+|C_2|}$ and to target level C_2 with the probability $\frac{|C_2|}{|C_1|+|C_2|}$, where $|C_1|$ and $|C_2|$ refer to the number of instances in $\mathcal{D}$ associated with C_1 and C_2 respectively. To ensure that the resulting decision tree classifies in the correct proportions, the decision tree is constructed by repeatedly partitioning[9] the training dataset, until every instance in a partition maps to the same target level.

Algorithm 4.1[19] lists a pseudocode description of the ID3 algorithm. Although the algorithm looks quite complex, it essentially does one of two things each time it is invoked: it either stops growing the current path in the

8 This algorithm was first published in Quinlan (1986).

9 Hence the name **Iterative Dichotomizer**.

tree by adding a leaf node to the tree, Lines 1–6, or it extends the current path by adding an interior node to the tree and growing the branches of this node by repeatedly rerunning the algorithm, Lines 7–13.

Algorithm 4.1 Pseudocode description of the ID3 algorithm.

Require: set of descriptive features $\mathbf{d}$
Require: set of training instances $\mathcal{D}$
1: **if** all the instances in $\mathcal{D}$ have the same target level C **then**
2: **return** a decision tree consisting of a leaf node with label C
3: **else if** $\mathbf{d}$ is empty **then**
4: **return** a decision tree consisting of a leaf node with the label of the majority target level in $\mathcal{D}$
5: **else if** $\mathcal{D}$ is empty **then**
6: **return** a decision tree consisting of a leaf node with the label of the majority target level of the dataset of the immediate parent node
7: **else**
8: $\mathbf{d}[best] \leftarrow \arg\max_{d \in \mathbf{d}} IG(d, \mathcal{D})$
9: make a new node, $Node_{\mathbf{d}[best]}$, and label it with $\mathbf{d}[best]$
10: partition $\mathcal{D}$ using $\mathbf{d}[best]$
11: remove $\mathbf{d}[best]$ from $\mathbf{d}$
12: **for** each partition $\mathcal{D}_i$ of $\mathcal{D}$ **do**
13: grow a branch from $Node_{\mathbf{d}[best]}$ to the decision tree created by rerunning $ID3$ with $\mathcal{D} = \mathcal{D}_i$

Lines 1–6 of Algorithm 4.1[19] control when a new leaf node is created in the tree. We have already mentioned that the ID3 algorithm constructs the decision tree by recursively partitioning the dataset. An important decision to be made when designing any recursive process is what the base cases that stop the recursion will be. In the ID3 algorithm the base cases are the situations where we stop splitting the dataset and construct a leaf node with an associated target level. There are two important things to remember when designing these base cases. First, the dataset of training instances considered at each of the interior nodes in the tree is not the complete dataset; rather, it is the subset of instances considered at its parent node that had the relevant feature value for the branch from the parent to the current node. Second, once a feature has been tested, it is not considered for selection again along that path in the tree. A feature will only be tested once on any path in the tree, but it may occur several times in

the tree on different paths. Based on these constraints, the algorithm defines three situations where the recursion stops and a leaf node is constructed:

1. All the instances in the dataset have the same target feature level. In this situation, the algorithm returns a single leaf node tree with that target level as its label (Algorithm 4.1[19] Lines 1–2).
2. The set of features left to test is empty. This means that we have already tested every feature on the path between the root node and the current node. We have no more features we can use to distinguish between the instances, so we return a single leaf node tree with the majority target level of the dataset as its target level (Algorithm 4.1[19] Lines 3–4).
3. The dataset is empty. This can occur when, for a particular partition of the dataset, there are no instances that have a particular feature value. In this case we return a single leaf node tree with the majority target level of the dataset at the parent node that made the recursive call (Algorithm 4.1[19] Lines 5–6).

If none of these cases hold, the algorithm continues to recursively create interior nodes, Lines 7–13 of Algorithm 4.1[19]. The first step in creating an interior node is to decide which descriptive feature should be tested at this node (Line 8 of Algorithm 4.1[19]). When we first mentioned the ID3 algorithm, we stated that it tries to create the shallowest decision tree that is consistent with the data given. The feature of the ID3 algorithm that biases it toward shallow trees is the mechanism that it uses to determine which descriptive feature is the *most informative* one to test at a new node. The ID3 algorithm uses the **information gain** metric to choose the best feature to test at each node in the tree. Consequently, the selection of the best feature to split a dataset on is based on the purity, or homogeneity, of the resulting partitions in the datasets. Again, remember that each node is constructed in a context consisting of a dataset of instances containing a subset of the instances used to construct its parent node and the set of descriptive features that have not been tested on the path between the root node and parent node. As a result, the information gain for a particular descriptive feature may be different at different nodes in the tree because it will be computed on different subsets of the full training dataset. One consequence of this is that a feature with a low information gain at the root node (when the full dataset is considered) may have a high information gain score at one of the interior nodes because it is predictive on the subset of instances that are considered at that interior node.

Once the most informative feature, $\mathbf{d}[best]$, has been chosen, the algorithm adds a new node, labeled with the feature $\mathbf{d}[best]$, to the tree (Line 9). It then splits the dataset that was considered at this node, $\mathcal{D}$, into partitions, $\mathcal{D}_1, \ldots, \mathcal{D}_k$, according to the levels that $\mathbf{d}[best]$ can take, $\{l_1, \ldots, l_k\}$ (Line 10). Next, it removes the feature $\mathbf{d}[best]$ from the set of features considered for testing later on this path in the tree; this enforces the constraint that a feature can be tested only once on any particular path in the tree (Line 11). Finally, in Lines 12 and 13, the algorithm grows a branch in the tree for each of the values in the domain of $\mathbf{d}[best]$ by recursively calling itself for each of the partitions created at Line 10. Each of these recursive calls uses the partition it is called on as the dataset it considers and is restricted to selecting from the set of features that have not been tested so far on the path from the root node. The node returned by the recursive call to the algorithm may be the root of a subtree or a leaf node. Either way, it is joined to the current node with a branch labeled with the appropriate level of the selected feature.

4.3.1 A Worked Example: Predicting Vegetation Distributions

In this section we will work through an example to illustrate how the ID3 is used to induce a decision tree. This example is based on **ecological modeling**, an area of scientific research that applies statistical and analytical techniques to model ecological processes. One of the problems faced by ecological management practitioners is that it is often too expensive to do large-scale, high-resolution land surveys. Using predictive analytics, however, the results of small-scale surveys can be used to create predictive models that can be applied across large regions. These models are used to inform resource management and conservation activities[10], such as managing the distribution of animal species and vegetation across geographic regions. The descriptive features used by these models are often features that can be automatically extracted from digitized maps, aerial photographs, or satellite imagery—for example, the elevation, steepness, color, and spectral reflection of the terrain, and the presence or absence of features such as rivers, roads, or lakes.

Table 4.3 lists an example dataset from the ecological modeling domain.[11] In this example, the prediction task is to classify the type of vegetation that

10 See Guisan and Zimmermann (2000) and Franklin (2009) for an introduction to uses of predictive analytics in ecological modeling.

11 This artificially generated example dataset is inspired by the research reported in Franklin et al. (2000).

Table 4.3
The vegetation classification dataset.

ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	false	steep	high	chapparal
2	true	moderate	low	riparian
3	true	steep	medium	riparian
4	false	steep	medium	chapparal
5	false	flat	high	conifer
6	true	steep	highest	conifer
7	true	steep	high	chapparal

is likely to be growing in areas of land based only on descriptive features extracted from maps of the areas. Ecological modelers can use information about the type of vegetation that grows in a region as a direct input into their animal species management and conservation programs because areas covered in different types of vegetation support different animal species. By using a predictive model that only requires features from maps, the ecological modelers can avoid expensive ground-based or aerial surveys. There are three types of vegetation that should be recognized by this model. First, *chapparal* is a type of evergreen shrubland that can be fire-prone. The animal species typically found in this vegetation include gray foxes, bobcats, skunks, and rabbits. Second, *riparian* vegetation occurs near streams and is characterized by trees and shrubs. It is usually home to small animals, including raccoons, frogs, and toads. Finally, *conifer* refers to forested areas that contain a variety of tree species (including pine, cedar, and fir trees), with a mixture of shrubs on the forest floor. The animals that may be found in these forests include bears, deer, and cougars. The type of vegetation in an area is stored in the target feature, VEGETATION.

There are three descriptive features in the dataset. STREAM is a binary feature that describes whether or not there is a stream in the area. SLOPE describes the steepness of the terrain in an area and has the levels *flat*, *moderate*, and *steep*. ELEVATION describes the elevation of an area and has the levels *low*, *medium*, *high*, and *highest*.

The first step in building the decision tree is to determine which of the three descriptive features is the best one to split the dataset on at the root node. The algorithm does this by computing the information gain for each feature. The total entropy for this dataset, which is required to calculate information gain,

Table 4.4

Partition sets (Part.), entropy, remainder (Rem.), and information gain (Info. Gain) by feature for the dataset in Table 4.3[22].

Split by Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*true*	$\mathcal{D}_1$	$\mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_6, \mathbf{d}_7$	1.5000	1.2507	0.3060
	false	$\mathcal{D}_2$	$\mathbf{d}_1, \mathbf{d}_4, \mathbf{d}_5$	0.9183		
SLOPE	*flat*	$\mathcal{D}_3$	$\mathbf{d}_5$	0.0	0.9793	0.5774
	moderate	$\mathcal{D}_4$	$\mathbf{d}_2$	0.0		
	steep	$\mathcal{D}_5$	$\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_6, \mathbf{d}_7$	1.3710		
ELEVATION	*low*	$\mathcal{D}_6$	$\mathbf{d}_2$	0.0	0.6793	0.8774
	medium	$\mathcal{D}_7$	$\mathbf{d}_3, \mathbf{d}_4$	1.0		
	high	$\mathcal{D}_8$	$\mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_7$	0.9183		
	highest	$\mathcal{D}_9$	$\mathbf{d}_6$	0.0		

is computed as

$$\begin{aligned} & H(\text{VEGETATION}, \mathcal{D}) \\ & = - \sum_{l \in \left\{ \begin{array}{l} chapparal, \\ riparian, \\ conifer \end{array} \right\}} P(\text{VEGETATION} = l) \times log_2 \left(P(\text{VEGETATION} = l) \right) \\ & = - \left(\left({}^3/_7 \times log_2 \left({}^3/_7 \right) \right) + \left({}^2/_7 \times log_2 \left({}^2/_7 \right) \right) + \left({}^2/_7 \times log_2 \left({}^2/_7 \right) \right) \right) \\ & = 1.5567 \; bits \end{aligned} \tag{4.5}$$

Table 4.4[23] shows the calculation of the information gain for each feature using this result.

We can see from Table 4.4[23] that ELEVATION has the largest information gain of the three features and so is selected by the algorithm at the root node of the tree. Figure 4.8[24] illustrates the state of the tree after the dataset is split using ELEVATION. Notice that the full dataset has been split into four partitions (labeled $\mathcal{D}_6$, $\mathcal{D}_7$, $\mathcal{D}_8$, and $\mathcal{D}_9$ in Table 4.4[23]) and that the feature ELEVATION is no longer listed in these partitions as it has already been used to split the data. The $\mathcal{D}_6$ and $\mathcal{D}_9$ partitions each contain just one instance. Consequently, they are pure sets, and these partitions can be converted into leaf nodes. The $\mathcal{D}_7$ and $\mathcal{D}_8$ partitions, however, contain instances with a mixture of target feature levels, so the algorithm needs to continue splitting these partitions. To do this,

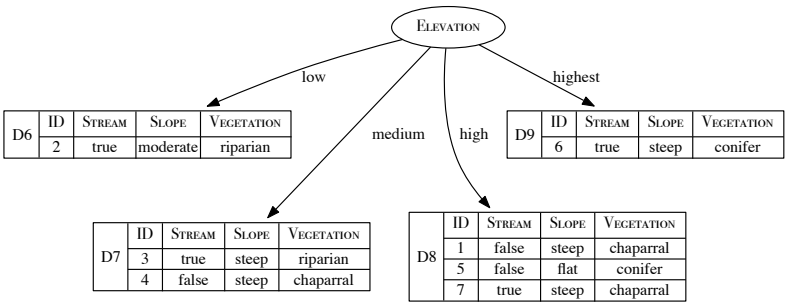


Figure 4.8

The decision tree after the data has been split using ELEVATION.

Table 4.5

Partition sets (Part.), entropy, remainder (Rem.), and information gain (Info. Gain) by feature for the dataset $\mathcal{D}_7$ in Figure 4.8[24].

Split by Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*true*	$\mathcal{D}_{10}$	$\mathbf{d}_3$	0.0	0.0	1.0
	false	$\mathcal{D}_{11}$	$\mathbf{d}_4$	0.0		
SLOPE	*flat*	$\mathcal{D}_{12}$		0.0	1.0	0.0
	moderate	$\mathcal{D}_{13}$		0.0		
	steep	$\mathcal{D}_{14}$	$\mathbf{d}_3, \mathbf{d}_4$	1.0		

the algorithm needs to decide which of the remaining descriptive features has the highest information gain for each partition.

To address partition $\mathcal{D}_7$, first the algorithm computes the entropy of $\mathcal{D}_7$ as

$$\begin{aligned} & H(\text{VEGETATION}, \mathcal{D}_7) \\ &= - \sum_{l \in \left\{ \begin{matrix} \textit{chapparal}, \\ \textit{riparian}, \\ \textit{conifer} \end{matrix} \right\}} P(\text{VEGETATION} = l) \times log_2 \left(P(\text{VEGETATION} = l) \right) \\ &= - \left(\left({}^1/_2 \times log_2({}^1/_2) \right) + \left({}^1/_2 \times log_2({}^1/_2) \right) + \left({}^0/_2 \times log_2({}^0/_2) \right) \right) \\ &= 1.0 \; \textit{bits} \end{aligned} \tag{4.6}$$

The information gained by splitting $\mathcal{D}_7$ for using STREAM and SLOPE is then computed as detailed in Table 4.5[24].

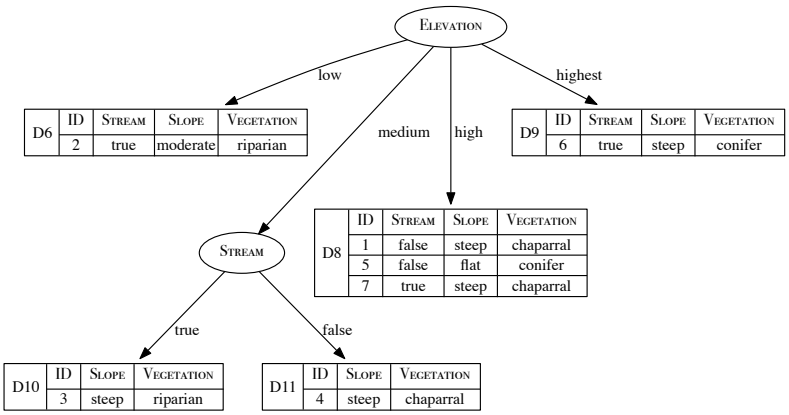


Figure 4.9

The state of the decision tree after the $\mathcal{D}_7$ partition has been split using STREAM.

The calculations in Table 4.5[24] show that STREAM has a higher information gain than SLOPE and so is the best feature with which to split $\mathcal{D}_7$. Figure 4.9[25] depicts the state of the decision tree after the $\mathcal{D}_7$ partition has been split. Splitting $\mathcal{D}_7$ creates two new partitions ($\mathcal{D}_{10}$ and $\mathcal{D}_{11}$). Notice that SLOPE is the only descriptive feature that is listed in $\mathcal{D}_{10}$ and $\mathcal{D}_{11}$. This reflects the fact that ELEVATION and STREAM have already been used on the path from the root node to each of these partitions and so cannot be used again. Both of these new partitions are pure sets with respect to the target feature (indeed, they only contain one instance each), and consequently, these sets do not need to be split any further and can be converted into leaf nodes.

At this point $\mathcal{D}_8$ is the only partition that is not a pure set. There are two descriptive features that can be used to split $\mathcal{D}_8$: STREAM and SLOPE. The decision regarding which of these features to split on is made by calculating which feature has the highest information gain for $\mathcal{D}_8$. The overall entropy for

Table 4.6

Partition sets (Part.), entropy, remainder (Rem.), and information gain (Info. Gain) by feature for the dataset $\mathcal{D}_8$ in Figure 4.9[25].

Split by Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*true*	$\mathcal{D}_{15}$	$\mathbf{d}_7$	0.0	0.6666	0.2517
	false	$\mathcal{D}_{16}$	$\mathbf{d}_1, \mathbf{d}_5$	1.0		
SLOPE	*flat*	$\mathcal{D}_{17}$	$\mathbf{d}_5$	0.0	0.0	0.9183
	moderate	$\mathcal{D}_{18}$		0.0		
	steep	$\mathcal{D}_{19}$	$\mathbf{d}_1, \mathbf{d}_7$	0.0		

$\mathcal{D}_8$ is calculated as

$$\begin{aligned} & H(\text{VEGETATION}, \mathcal{D}_8) \\ &= - \sum_{l \in \left\{ \begin{array}{l} chapparal, \\ riparian, \\ conifer \end{array} \right\}} P(\text{VEGETATION} = l) \times log_2 \left(P(\text{VEGETATION} = l) \right) \\ &= - \left(\left({}^2/_3 \times log_2({}^2/_3) \right) + \left({}^0/_3 \times log_2({}^0/_3) \right) + \left({}^1/_3 \times log_2({}^1/_3) \right) \right) \\ &= 0.9183 \; bits \end{aligned} \tag{4.7}$$

Table 4.6[26] details the calculation of the information gain for each descriptive feature in $\mathcal{D}_8$ using this result. It is clear from Table 4.6[26] that in the context of $\mathcal{D}_8$, SLOPE has a higher information gain than STREAM.

Figure 4.10[27] illustrates the state of the decision tree after $\mathcal{D}_8$ has been split. Notice that one of the partitions created by splitting $\mathcal{D}_8$ based on SLOPE is empty: $\mathcal{D}_{18}$. This is because there were no instances in $\mathcal{D}_8$ that had a value of *moderate* for the SLOPE feature. This empty partition will result in a leaf node that returns a prediction of the majority target level in $\mathcal{D}_8$, *chapparal*. The other two partitions created by splitting $\mathcal{D}_8$ are pure with respect to the target feature: $\mathcal{D}_{17}$ contains one instance with a *conifer* target level, and $\mathcal{D}_{19}$ contains two instances, both of which have a *chapparal* target level.

At this point all the remaining partitions are pure with respect to the target feature. Consequently, the algorithm now converts each partition into a leaf node and returns the final decision tree. Figure 4.11[27] shows this decision tree. If the prediction strategy encoded in this tree is applied to the original dataset in Table 4.3[22], it will correctly classify all the instances in the dataset. In machine learning terms, the induced model is **consistent** with the training data.

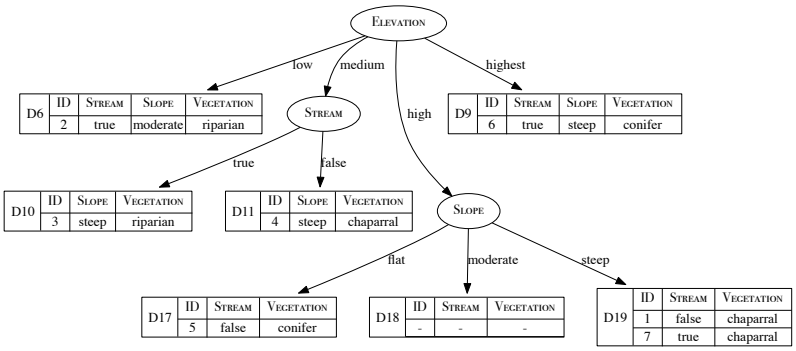


Figure 4.10
The state of the decision tree after the $\mathcal{D}_8$ partition has been split using SLOPE.

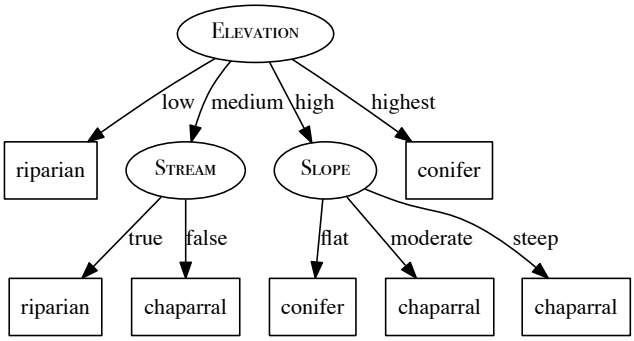


Figure 4.11
The final vegetation classification decision tree.

One final point: remember that the empty partition in Figure 4.10[27] ($\mathcal{D}_{18}$) has been converted into a leaf node that returns the *chapparal* target level. This is because *chapparal* is the majority target level in the partition at the parent node ($\mathcal{D}_8$) of this leaf node. Consequently, this tree will return a prediction of VEGETATION = *chapparal* for the following query:

STREAM = *true*, SLOPE = *moderate*, ELEVATION = *high*

This is interesting because there are no instances listed in Table 4.3[22] where SLOPE = *moderate* and VEGETATION = *chapparal*. This example illustrates one way in which the predictions made by the model generalize beyond the dataset. Whether the generalizations made by the model are correct will depend on whether the assumptions used in generating the model (i.e., the **inductive bias**) are appropriate.

The ID3 algorithm works in exactly the same way for larger, more complicated datasets; there is simply more computation involved. Since it was first proposed, there have been many modifications to the original ID3 algorithm to handle variations that are common in real-world datasets. We explore the most important of these modifications in the following sections.

Bibliography

Franklin, J. (2009). *Mapping Species Distributions: Spatial Inference and Prediction (Ecology, Biodiversity and Conservation)*. Cambridge Univ Press.

Franklin, J., P. McCullough, and C. Gray (2000). Terrain variables used for predictive mapping of vegetation communities in southern california. In J. C. G. John P. Wilson (Ed.), *Terrain analysis: principles and applications*. Wiley.

Guisan, A. and N. E. Zimmermann (2000). Predictive habitat distribution models in ecology. *Ecological modelling 135*(2), 147–186.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning 1*(1), 81–106.