The information measure can be changed in another way, which is to add a weight to the misclassifications. The idea is to consider the cost of misclassifying an instance of class i as class j (which we will call the risk in Section 2.3.1) and add a weight that says how important each datapoint is. It is typically labelled as λ_{ij} and is presented as a matrix, with element λ_{ij} representing the cost of misclassifying i as j. Using it is simple, modifying the Gini impurity (Equation (12.8)) to be:

$$G_i = \sum_{j \neq i} \lambda_{ij} N(i) N(j). \tag{12.10}$$

We will see in Section 13.1 that there is another benefit to using these weights, which is to successively improve the classification ability by putting higher weight on datapoints that the algorithm is getting wrong.

12.3.2 Regression in Trees

The new part about CART is its application in regression. While it might seem strange to use trees for regression, it turns out to require only a simple modification to the algorithm. Suppose that the outputs are continuous, so that a regression model is appropriate. None of the node impurity measures that we have considered so far will work. Instead, we'll go back to our old favourite—the sum-of-squares error. To evaluate the choice of which feature to use next, we also need to find the value at which to split the dataset according to that feature. Remember that the output is a value at each leaf. In general, this is just a constant value for the output, computed as the mean average of all the datapoints that are situated in that leaf. This is the optimal choice in order to minimise the sum-of-squares error, but it also means that we can choose the split point quickly for a given feature, by choosing it to minimise the sum-of-squares error. We can then pick the feature that has the split point that provides the best sum-of-squares error, and continue to use the algorithm as for classification.

12.4 CLASSIFICATION EXAMPLE

We'll work through an example using ID3 in this section. The data that we'll use will be a continuation of the one we started the chapter with, about what to do in the evening.

When we want to construct the decision tree to decide what to do in the evening, we start by listing everything that we've done for the past few days to get a suitable dataset (here, the last ten days):

Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

To produce a decision tree for this problem, the first thing that we need to do is work out which feature to use as the root node. We start by computing the entropy of S:

$$\begin{aligned}
\text{Entropy}(S) &= -p_{\text{party}} \log_2 p_{\text{party}} - p_{\text{study}} \log_2 p_{\text{study}} \\
&- p_{\text{pub}} \log_2 p_{\text{pub}} - p_{\text{TV}} \log_2 p_{\text{TV}} \\
&= -\frac{5}{10} \log_2 \frac{5}{10} - \frac{3}{10} \log_2 \frac{3}{10} - \frac{1}{10} \log_2 \frac{1}{10} - \frac{1}{10} \log_2 \frac{1}{10} \\
&= 0.5 + 0.5211 + 0.3322 + 0.3322 = 1.6855
\end{aligned} \tag{12.11}$$

and then find which feature has the maximal information gain:

$$\begin{aligned}
\text{Gain}(S, \text{Deadline}) &= 1.6855 - \frac{|S_{\text{urgent}}|}{10} \text{Entropy}(S_{\text{urgent}}) \\
&- \frac{|S_{\text{near}}|}{10} \text{Entropy}(S_{\text{near}}) - \frac{|S_{\text{none}}|}{10} \text{Entropy}(S_{\text{none}}) \\
&= 1.6855 - \frac{3}{10} \left(-\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3} \right) \\
&- \frac{4}{10} \left(-\frac{2}{4} \log_2 \frac{2}{4} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{4} \log_2 \frac{1}{4} \right) \\
&- \frac{3}{10} \left(-\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} \right) \\
&= 1.6855 - 0.2755 - 0.6 - 0.2755 \\
&= 0.5345
\end{aligned} \tag{12.12}$$

$$\begin{aligned}
\text{Gain}(S, \text{Party}) &= 1.6855 - \frac{5}{10} \left(-\frac{5}{5} \log_2 \frac{5}{5} \right) \\
&- \frac{5}{10} \left(-\frac{3}{5} \log_2 \frac{3}{5} - \frac{1}{5} \log_2 \frac{1}{5} - \frac{1}{5} \log_2 \frac{1}{5} \right) \\
&= 1.6855 - 0 - 0.6855 \\
&= 1.0
\end{aligned} \tag{12.13}$$

$$\begin{aligned}
\text{Gain}(S, \text{Lazy}) &= 1.6855 - \frac{6}{10} \left(-\frac{3}{6} \log_2 \frac{3}{6} - \frac{1}{6} \log_2 \frac{1}{6} - \frac{1}{6} \log_2 \frac{1}{6} - \frac{1}{6} \log_2 \frac{1}{6} \right) \\
&- \frac{4}{10} \left(-\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4} \right) \\
&= 1.6855 - 1.0755 - 0.4 \\
&= 0.21
\end{aligned} \tag{12.14}$$

Therefore, the root node will be the party feature, which has two feature values ('yes' and 'no'), so it will have two branches coming out of it (see Figure 12.6). When we look at the 'yes' branch, we see that in all five cases where there was a party we went to it, so we just put a leaf node there, saying 'party'. For the 'no' branch, out of the five cases there are three different outcomes, so now we need to choose another feature. The five cases we are looking at are:

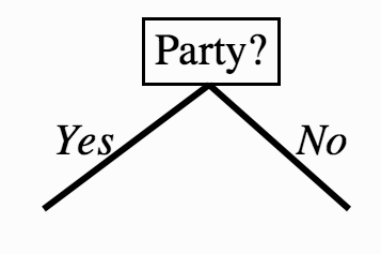

FIGURE 12.6 The decision tree after one step of the algorithm.

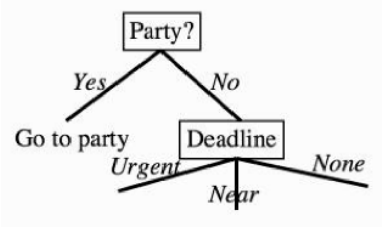

FIGURE 12.7 The tree after another step.

Deadline?	Is there a party?	Lazy?	Activity
Urgent	No	Yes	Study
None	No	Yes	Pub
Near	No	No	Study
Near	No	Yes	TV
Urgent	No	Yes	Study

We've used the party feature, so we just need to calculate the information gain of the other two over these five examples:

$$\begin{aligned}
\text{Gain}(S, \text{Deadline}) &= 1.371 - \frac{2}{5}\left(-\frac{2}{2}\log_2\frac{2}{2}\right) \\
&- \frac{2}{5}\left(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}\right) - \frac{1}{5}\left(-\frac{1}{1}\log_2\frac{1}{1}\right) \\
&= 1.371 - 0 - 0.4 - 0 \\
&= 0.971 \qquad (12.15) \\
\text{Gain}(S, \text{Lazy}) &= 1.371 - \frac{4}{5}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4}\right) \\
&- \frac{1}{5}\left(-\frac{1}{1}\log_2\frac{1}{1}\right) \\
&= 1.371 - 1.2 - 0 \\
&= 0.1710 \qquad (12.16)
\end{aligned}$$

This leads to the tree shown in Figure 12.7. From this point it is relatively simple to complete the tree, leading to the one that was shown in Figure 12.1.

FURTHER READING

For more information about decision trees, the following two books are of interest:

- J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Francisco, CA, USA, 1993.
- L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Chapman & Hall, New York, USA, 1993.

If you want to know more about information theory, then there are lots of books on the topic, including:

- T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, USA, 1991.
- F.M. Reza. *An Introduction to Information Theory*. McGraw-Hill, New York, USA, 1961.

The original paper that started the field is:

- C.E. Shannon. A mathematical theory of information. *The Bell System Technical Journal*, 27(3):379–423 and 623–656, 1948.

A book that covers information theory and machine learning is:

- D.J.C. MacKay. *Information Thoery, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

Other machine learning textbooks that cover decision trees include:

- Sections 8.2–8.4 of R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*, 2nd edition, Wiley-Interscience, New York, USA, 2001.
- Chapter 7 of B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.
- Chapter 3 of T. Mitchell. *Machine Learning*. McGraw-Hill, New York, USA, 1997.

PRACTICE QUESTIONS

Problem 12.1 Suppose that the probability of five events are P(first) = 0.5, and P(second) = P(third) = P(fourth) = P(fifth) = 0.125. Calculate the entropy. Write down in words what this means.

Problem 12.2 Make a decision tree that computes the logical AND function. How does it compare to the Perceptron solution?

Problem 12.3 Turn this politically incorrect data from Quinlan into a decision tree to classify which attributes make a person attractive, and then extract the rules.

Height	Hair	Eyes	Attractive?
Small	Blonde	Brown	No
Tall	Dark	Brown	No
Tall	Blonde	Blue	Yes
Tall	Dark	Blue	No
Small	Dark	Blue	No
Tall	Red	Blue	Yes
Tall	Blonde	Brown	No
Small	Blonde	Blue	Yes

Problem 12.4 When you arrive at the pub, your five friends already have their drinks on the table. Jim has a job and buys the round half of the time. Jane buys the round a quarter of the time, and Sarah and Simon buy a round one eighth of the time. John hasn't got his wallet out since you met him three years ago.

Compute the entropy of each of them buying the round and work out how many questions you need to ask (on average) to find out who bought the round.

Two more friends now arrive and everybody spontaneously decides that it is your turn to buy a round (for all eight of you). Your friends set you the challenge of deciding who is drinking beer and who is drinking vodka according to their gender, whether or not they are students, and whether they went to the pub last night. Use ID3 to work it out, and then see if you can prune the tree.

Drink	Gender	Student	Pub last night
Beer	T	T	T
Beer	T	F	T
Vodka	T	F	F
Vodka	T	F	F
Vodka	F	T	T
Vodka	F	F	F
Vodka	F	T	T
Vodka	F	T	T

Problem 12.5 Use the naïve Bayes classifier from Section 2.3.2 on the datasets that you used for the decision tree (this will involve some effort in turning the textual data into probabilities) and compare the results.

Problem 12.6 The `CPU` dataset in the UCI repository is a very good regression problem for a decision tree. You will need to modify the decision tree code so that it does regression, as discussed in Section 12.3.2. You will also have to work out the Gini impurity for multiple classes.

Problem 12.7 Modify the implementation to deal with continuous variables, as discussed in Section 12.2.5.

Problem 12.8 The misclassification impurity is:

$$N(i) = 1 - \max_j P(w_j). \tag{12.17}$$

Add this into the code and test the new version on some of the datasets above.

CHAPTER 13

Decision by Committee: Ensemble Learning

The old saying has it that two heads are better than one. Which naturally leads to the idea that even more heads are better than that, and ends up with decision by committee, which is famously useless for human activities (as in the old joke that a camel is a horse designed by a committee). For machine learning methods the results are rather more impressive, as we'll see in this chapter.

The basic idea is that by having lots of learners that each get slightly different results on a dataset—some learning certain things well and some learning others—and putting them together, the results that are generated will be significantly better than any one of them on its own (provided that you put them together well... otherwise the results could be significantly worse). One analogy that might prove useful is to think about how your doctor goes about performing a diagnosis of some complaint that you visit her with. If she cannot find the problem directly, then she will ask for a variety of tests to be performed, e.g., scans, blood tests, consultations with experts. She will then aggregate all of these opinions in order to perform a diagnosis. Each of the individual tests will suggest a diagnosis, but only by putting them together can an informed decision be reached.

Figure 13.1 shows the basic idea of ensemble learning, as these methods are collectively called. Given a relatively simple binary classification problem and some learner that puts an ellipse around a subset of the data, combining the ellipses can provide a considerably more complex decision boundary.

There are then only a couple of questions to ask: which learners should we use, how should we ensure that they learn different things, and how should we combine their results? The methods that we are investigating in this chapter can use any classifier at all. Although in general they only use one type of classifier at a time, they do not have to. A common choice of classifier is the decision tree (see Chapter 12).

Ensuring that the learners see different things can be performed in different ways, and it is the primary difference between the algorithms that we shall see. However, it can also come about naturally depending upon the application area. Suppose that you have lots and lots of data. In that case you could simply randomly partition the data and give different sets of data to different classifiers. Even here there are choices: do you make the partitions separate, or include overlaps? If there is no overlap, then it could be difficult to work out how to combine the classifiers, or it might be very simple: if your doctor always asks for opinions from two colleagues, one specialising in heart problems and one in sports injuries,

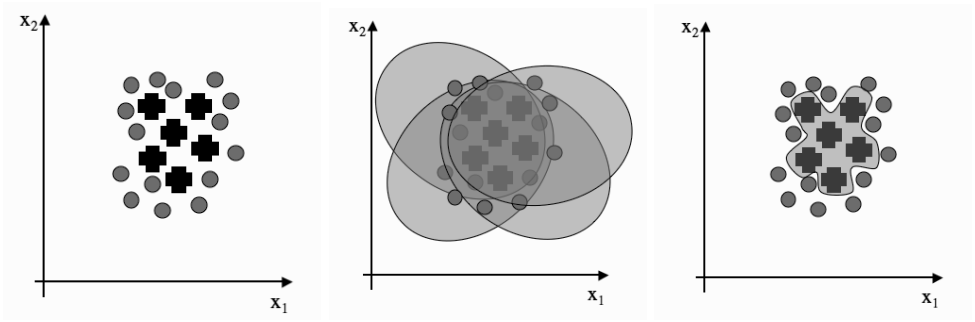

FIGURE 13.1 By combining lots of simple classifiers (here that simply put an elliptical decision boundary onto the data), the decision boundary can be made much more complicated, enabling the difficult separation of the pluses from the circles.

then upon discovering that your leg started hurting after you went for a run she would likely accord more weight to the diagnosis of the sports injury expert.

Interestingly, ensemble methods do very well when there is very little data as well as when there is too much. To see why, think cross-validation (Section 2.2.2). We used cross-validation when there was not enough data to go around, and trained lots of neural networks on different subsets of the data. Then we threw away most of them. With an ensemble method we keep them all, and combine their results in some way. One very simple way to combine the results is to use majority voting — if it's good enough for electing governments in elections, it's good enough for machine learning. Majority voting has the interesting property that for binary classification, the combined classifier will only get the answer wrong if more than half of the classifiers were wrong. Hopefully, this isn't going to happen too often (although you might be able to think of government elections where this has been the case in your view). There are alternative ways to combine the results, as we'll discuss. These things will become clearer as we look at the algorithms, so let's get started.

13.1 BOOSTING

At first sight the claim of the most popular ensemble method, boosting, seems amazing. If we take a collection of very poor (weak in the jargon) learners, each performing only just better than chance, then by putting them together it is possible to make an ensemble learner that can perform arbitrarily well. So we just need lots of low-quality learners, and a way to put them together usefully, and we can make a learner that will do very well.

The principal algorithm of boosting is named AdaBoost, and is described in Section 13.1.1. The algorithm was first described in the mid-1990s by Freund and Shapiro, and while it has had many variations derived from it, the principal algorithm is still one of the most widely used. The algorithm was proposed as an improvement on the original 1990 boosting algorithm, which was rather data hungry. In that algorithm, the training set was split into three. A classifier was trained on the first third, and then tested on the second third. All of the data that was misclassified during that testing was used to form a new dataset, along with an equally sized random selection of the data that was correctly classified. A second classifier was trained on this new dataset, and then both of the classifiers were tested on the final third of the dataset. If they both produced the same output, then that datapoint was ignored, otherwise the datapoint was added to yet another new

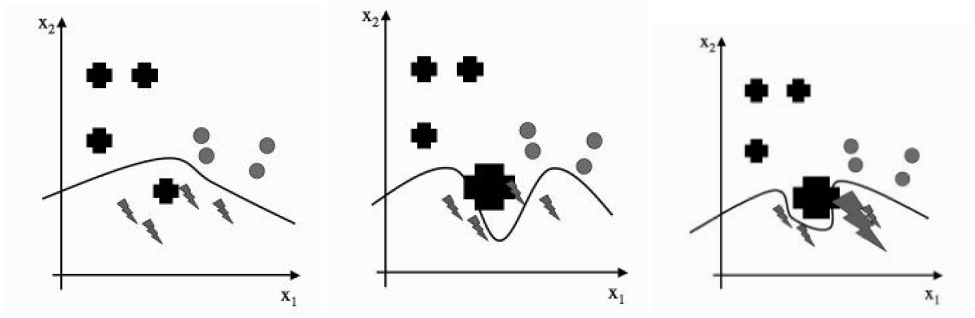

FIGURE 13.2 As points are misclassified, so their weights increase in boosting (shown by the datapoint getting larger), which makes the importance of those datapoints increase, making the classifiers pay more attention to them.

dataset, which formed the training set for a third classifer. Rather than looking further at this version, we will look at the more common algorithm.

13.1.1 AdaBoost

The innovation that AdaBoost (which stands for adaptive boosting) uses is to give weights to each datapoint according to how difficult previous classifiers have found to get it correct. These weights are given to the classifier as part of the input when it is trained.

The AdaBoost algorithm is conceptually very simple. At each iteration a new classifier is trained on the training set, with the weights that are applied to the training set for each datapoint being modified at each iteration according to how successfully that datapoint has been classified in the past. The weights are initially all set to the same value, $1/N$, where N is the number of datapoints in the training set. Then, at each iteration, the error (ϵ) is computed as the sum of the weights of the misclassified points, and the weights for incorrect examples are updated by being multiplied by $\alpha = (1 - \epsilon)/\epsilon$. Weights for correct examples are left alone, and then the whole set is normalised so that it sums to 1 (which is effectively a reduction in the importance of the correctly classified datapoints). Training terminates after a set number of iterations, or when either all of the datapoints are classified correctly, or one point contains more than half of the available weight.

Figure 13.2 shows the effect of weighting incorrectly classified examples as training proceeds, with the size of each datapoint being a measure of its importance. As an algorithm this looks like (where $I(y_n \neq h_t(x_n))$ is an indicator function that returns 1 if the target and output are not equal, and 0 if they are):