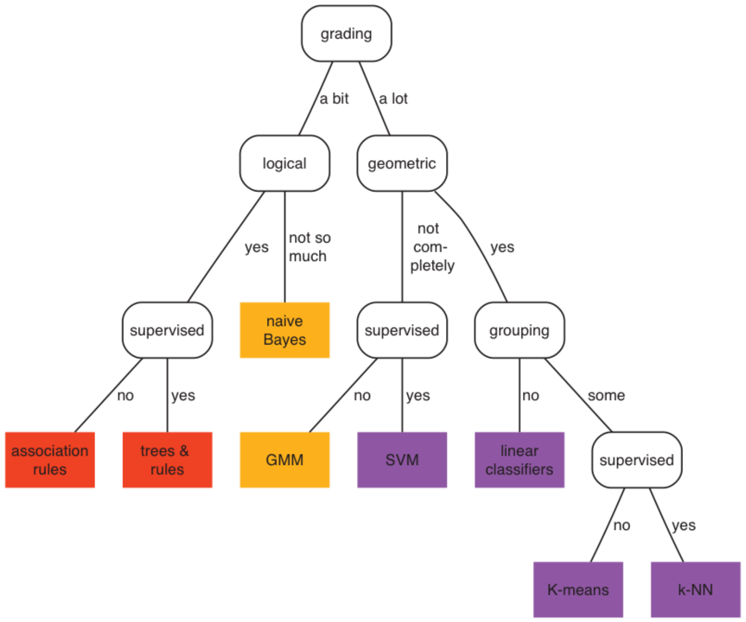


Figure 1.8. A taxonomy describing machine learning methods in terms of the extent to which they are grading or grouping models, logical, geometric or a combination, and supervised or unsupervised. The colours indicate the type of model, from left to right: logical (red), probabilistic (orange) and geometric (purple).

absolute, and some models combine both features. For instance, even though linear classifiers are a prime example of a grading model, it is easy to think of instances that a linear model can't distinguish, namely instances on a line or plane parallel to the decision boundary. The point is not so much that there aren't any segments, but that there are infinitely many. On the other end of the spectrum, regression trees combine grouping and grading features, as we shall see a little later. The overall picture is thus somewhat like what is depicted in Figure 1.7. A taxonomy of eight different models discussed in the book is given in Figure 1.8.[7] These models will be discussed in detail in Chapters 4–9.

1.3 Features: the workhorses of machine learning

Now that we have seen some more examples of machine learning tasks and models, we turn to the third and final main ingredient. Features determine much of the success of a machine learning application, because a model is only as good as its features. A fea-

[7]The figures have been generated from data explained in Example 1.7 below.

Model	geom	stats	logic	group	grad	disc	real	sup	unsup	multi
Trees	1	0	3	3	0	3	2	3	2	3
Rules	0	0	3	3	1	3	2	3	0	2
naive Bayes	1	3	1	3	1	3	1	3	0	3
kNN	3	1	0	2	2	1	3	3	0	3
Linear Classifier	3	0	0	0	3	1	3	3	0	0
Linear Regression	3	1	0	0	3	0	3	3	0	1
Logistic Regression	3	2	0	0	3	1	3	3	0	0
SVM	2	2	0	0	3	2	3	3	0	0
Kmeans	3	2	0	1	2	1	3	0	3	1
GMM	1	3	0	0	3	1	3	0	3	1
Associations	0	0	3	3	0	3	1	0	3	1

Table 1.4. The MLM data set describing properties of machine learning models. Both Figure 1.7 and Figure 1.8 were generated from this data.

ture can be thought of as a kind of measurement that can be easily performed on any instance. Mathematically, they are functions that map from the instance space to some set of feature values called the *domain* of the feature. Since measurements are often numerical, the most common feature domain is the set of real numbers. Other typical feature domains include the set of integers, for instance when the feature counts something, such as the number of occurrences of a particular word; the Booleans, if our feature is a statement that can be true or false for a particular instance, such as 'this e-mail is addressed to Peter Flach'; and arbitrary finite sets, such as a set of colours, or a set of shapes.

Example 1.7 (The MLM data set). Suppose we have a number of learning models that we want to describe in terms of a number of properties:

- the extent to which the models are geometric, probabilistic or logical;
- whether they are grouping or grading models;
- the extent to which they can handle discrete and/or real-valued features;
- whether they are used in supervised or unsupervised learning; and
- the extent to which they can handle multi-class problems.

The first two properties could be expressed by discrete features with three and two values, respectively; or if the distinctions are more gradual, each aspect could be rated on some numerical scale. A simple approach would be to measure each property on an integer scale from 0 to 3, as in Table 1.4. This table establishes a data set in which each row represents an instance and each column a feature. For example, according to this (highly simplified) data some models are purely grouping models (Trees, Associations) or purely grading models (the Linear models, Logistic Regression and GMM), whereas others are more mixed. We can also see that Trees and Rules have very similar values for most of the features, whereas GMM and Associations have mostly different values.

This small data set will be used in several examples throughout the book. In fact, the taxonomy in Figure 1.8 was adapted by hand from a decision tree learned from this small data set, using the models as classes. And the plot in Figure 1.7 was constructed using a dimensionality reduction technique which preserves pairwise distances as much as possible.

Two uses of features

It is worth noting that features and models are intimately connected, not just because models are defined in terms of features, but because a single feature can be turned into what is sometimes called a *univariate model*. We can therefore distinguish two uses of features that echo the distinction between grouping and grading models. A very common use of features, particularly in logical models, is to zoom in on a particular area of the instance space. Let f be a feature counting the number of occurrences of the word 'Viagra' in an e-mail, and let x stand for an arbitrary e-mail, then the condition $f(x) = 0$ selects e-mails that don't contain the word 'Viagra', $f(x) \neq 0$ or $f(x) > 0$ selects e-mails that do, $f(x) \geq 2$ selects e-mails that contain the word at least twice, and so on. Such conditions are called *binary splits*, because they divide the instance space into two groups: those that satisfy the condition, and those that don't. Non-binary splits are also possible: for instance, if g is a feature that has the value 'tweet' for e-mails with up to 20 words, 'short' for e-mails with 21 to 50 words, 'medium' for e-mails with 51 to 200 words, and 'long' for e-mails with more than 200 words, then the expression $g(x)$ represents a four-way split of the instance space. As we have already seen, such splits can be combined in a feature tree, from which a model can be built.

A second use of features arises particularly in supervised learning. Recall that a linear classifier employs a decision rule of the form $\sum_{i=1}^{n} w_i x_i > t$, where x_i is a numerical feature.[8] The linearity of this decision rule means that each feature makes an independent contribution to the score of an instance. This contribution depends on the weight w_i: if this is large and positive, a positive x_i increases the score; if $w_i \ll 0$, a positive x_i decreases the score; if $w_i \approx 0$, x_i's influence is negligible. Thus, the feature

[8]Notice we employ two different notations for features: sometimes we write $f(x)$ if it is more convenient to view a feature as a function applied to instance x, and sometimes we write x_i if it is more convenient to view an instance as a vector of feature values.

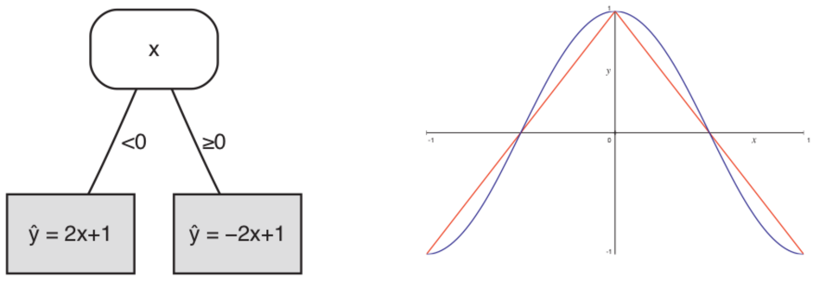


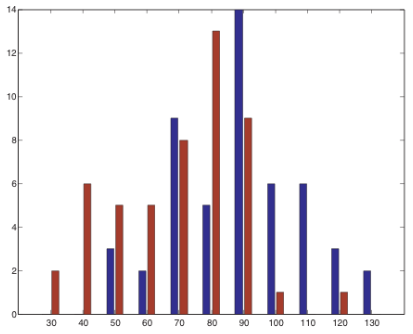
Figure 1.9. (left) A regression tree combining a one-split feature tree with linear regression models in the leaves. Notice how x is used as both a splitting feature and a regression variable. **(right)** The function $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$, and the piecewise linear approximation achieved by the regression tree.

makes a precise and measurable contribution to the final prediction. Also note that that individual features are not 'thresholded', but their full 'resolution' is used in computing an instance's score. These two uses of features – 'features as splits' and 'features as predictors' – are sometimes combined in a single model.

Example 1.8 (Two uses of features). Suppose we want to approximate $y = \cos \pi x$ on the interval $-1 \leq x \leq 1$. A linear approximation is not much use here, since the best fit would be $y = 0$. However, if we split the x-axis in two intervals $-1 \leq x < 0$ and $0 \leq x \leq 1$, we could find reasonable linear approximations on each interval. We can achieve this by using x both as a splitting feature and as a regression variable (Figure 1.9).

Feature construction and transformation

There is a lot of scope in machine learning for playing around with features. In the spam filter example, and text classification more generally, the messages or documents don't come with built-in features; rather, they need to be constructed by the developer of the machine learning application. This *feature construction* process is absolutely crucial for the success of a machine learning application. Indexing an e-mail by the words that occur in it (called a *bag of words* representation as it disregards the order of the words in the e-mail) is a carefully engineered representation that manages to amplify the 'signal' and attenuate the 'noise' in spam e-mail filtering and related classification tasks. However, it is easy to conceive of problems where this would be exactly



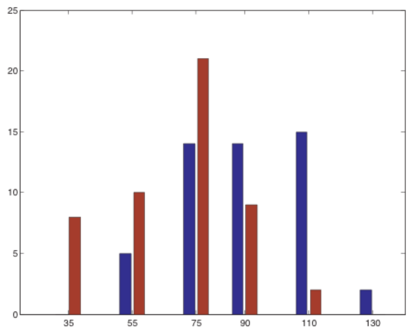


Figure 1.10. (left) Artificial data depicting a histogram of body weight measurements of people with (blue) and without (red) diabetes, with eleven fixed intervals of 10 kilograms width each. **(right)** By joining the first and second, third and fourth, fifth and sixth, and the eighth, ninth and tenth intervals, we obtain a discretisation such that the proportion of diabetes cases increases from left to right. This discretisation makes the feature more useful in predicting diabetes.

the wrong thing to do: for instance if we aim to train a classifier to distinguish between grammatical and ungrammatical sentences, word order is clearly signal rather than noise, and a different representation is called for.

It is often natural to build a model in terms of the given features. However, we are free to change the features as we see fit, or even to introduce new features. For instance, real-valued features often contain unnecessary detail that can be removed by *discretisation*. Imagine you want to analyse the body weight of a relatively small group of, say, 100 people, by drawing a histogram. If you measure everybody's weight in kilograms with one position after the decimal point (i.e., your precision is 100 grams), then your histogram will be sparse and spiky. It is hard to draw any general conclusions from such a histogram. It would be much more useful to discretise the body weight measurements into intervals of 10 kilograms. If we are in a classification context, say we're trying to relate body weight to diabetes, we could then associate each bar of the histogram with the proportion of people having diabetes among the people whose weight falls in that interval. In fact, as we shall see in Chapter 10, we can even choose the intervals such that this proportion is monotonically increasing (Figure 1.10).

The previous example gives another illustration of how, for a particular task such as classification, we can improve the signal-to-noise ratio of a feature. In more extreme cases of feature construction we transform the entire instance space. Consider Figure 1.11: the data on the left is clearly not linearly separable, but by mapping the instance space into a new 'feature space' consisting of the squares of the original features we see that the data becomes almost linearly separable. In fact, by adding in a third feature we can perform a remarkable trick: we can build this feature space classifier without actually constructing the feature space.

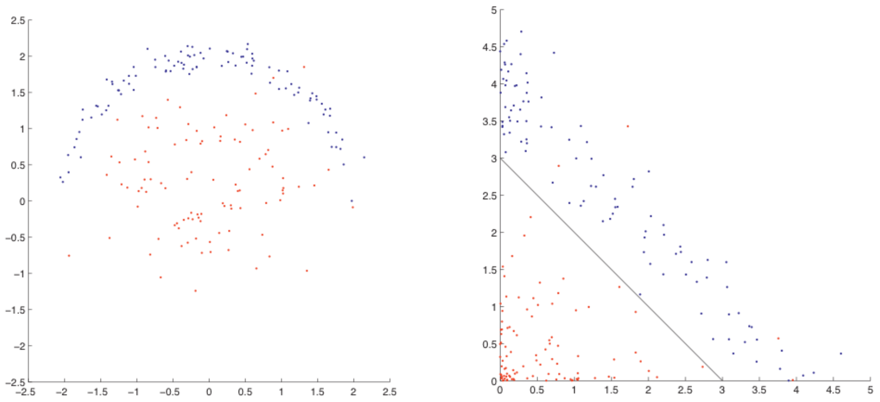


Figure 1.11. (left) A linear classifier would perform poorly on this data. **(right)** By transforming the original (x, y) data into $(x', y') = (x^2, y^2)$, the data becomes more 'linear', and a linear decision boundary $x' + y' = 3$ separates the data fairly well. In the original space this corresponds to a circle with radius $\sqrt{3}$ around the origin.

Example 1.9 (The kernel trick). Let $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ be two data points, and consider the mapping $(x, y) \mapsto (x^2, y^2, \sqrt{2}xy)$ to a three-dimensional feature space. The points in feature space corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ are $\mathbf{x}'_1 = (x_1^2, y_1^2, \sqrt{2}x_1 y_1)$ and $\mathbf{x}'_2 = (x_2^2, y_2^2, \sqrt{2}x_2 y_2)$. The dot product of these two feature vectors is

$$\mathbf{x}'_1 \cdot \mathbf{x}'_2 = x_1^2 x_2^2 + y_1^2 y_2^2 + 2x_1 y_1 x_2 y_2 = (x_1 x_2 + y_1 y_2)^2 = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$$

That is, by squaring the dot product in the original space we obtain the dot product in the new space *without actually constructing the feature vectors*! A function that calculates the dot product in feature space directly from the vectors in the original space is called a *kernel* – here the kernel is $\kappa(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$.

We can apply this *kernel trick* to the basic linear classifier if we modify the way the decision boundary is calculated. Recall that the basic linear classifier learns a decision boundary $\mathbf{w} \cdot \mathbf{x} = t$ with $\mathbf{w} = \mathbf{p} - \mathbf{n}$ being the difference between the mean of the positive examples and the mean of the negative examples. As an example, suppose we have $\mathbf{n} = (0, 0)$ and $\mathbf{p} = (0, 1)$, and let's assume for the sake of argument that the positive mean has been obtained from two training examples $\mathbf{p}_1 = (-1, 1)$ and $\mathbf{p}_2 = (1, 1)$. This means that $\mathbf{p} = \frac{1}{2}(\mathbf{p}_1 + \mathbf{p}_2)$ and we can rewrite the decision boundary as $\frac{1}{2}\mathbf{p}_1 \cdot \mathbf{x} + \frac{1}{2}\mathbf{p}_2 \cdot \mathbf{x} - \mathbf{n} \cdot \mathbf{x} = t$. Applying the kernel trick we obtain the following decision boundary: $\frac{1}{2}\kappa(\mathbf{p}_1, \mathbf{x}) + \frac{1}{2}\kappa(\mathbf{p}_2, \mathbf{x}) - \kappa(\mathbf{n}, \mathbf{x}) = t$. Using the kernel defined earlier we have $\kappa(\mathbf{p}_1, \mathbf{x}) = (-x+y)^2$, $\kappa(\mathbf{p}_2, \mathbf{x}) = (x+y)^2$ and $\kappa(\mathbf{n}, \mathbf{x}) = 0$, from which we derive the decision boundary $\frac{1}{2}(-x+y)^2 + \frac{1}{2}(x+y)^2 = x^2 + y^2 = t$, i.e., a circle around the origin with radius $\sqrt{t}$. Figure 1.11 illustrates this further for a larger data set.

The key point in this 'kernelisation' of the basic linear classifier is that we don't summarise the training data by the positive and negative means – rather, we keep the training data (here: $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{n}$), so that when classifying a new instance we can evaluate the kernel on it paired with each training example. In return for this more elaborate calculation we get the ability to construct much more flexible decision boundaries.

Interaction between features

One fascinating and multi-faceted aspect of features is that they may interact in various ways. Sometimes such interaction can be exploited, sometimes it can be ignored, and sometimes it poses a challenge. We have already seen an example of feature interaction when we talked about Bayesian spam filtering. Clearly, if we notice the term 'Viagra' in an e-mail, we are not really surprised to find that the e-mail also contains the phrase 'blue pill'. Ignoring this interaction, as the naive Bayes classifier does, means that we are overestimating the amount of information conveyed by observing both phrases in the same e-mail. Whether we can get away with this depends on our task: in spam e-mail classification it turns out not to be a big problem, apart from the fact that we may need to adapt the decision threshold to account for this effect.

We can observe other examples of feature interaction in Table 1.4 on p.39. Consider the features 'grad' and 'real', which assess the extent to which models are of the grading kind, and the extent to which they can handle real-valued features. You may observe that the values of these two features differ by at most 1 for all but one model. Statisticians say that these features are positively correlated (see Background 1.3). Another pair of positively correlated features is 'logic' and 'disc', indicating logical models and the ability to handle discrete features. We can also see some negatively correlated features, where the value of one goes up when the other goes down: this holds naturally for 'split' and 'grad', indicating whether models are primarily grouping or grading models; and also for 'logic' and 'grad'. Finally, pairs of uncorrelated features are 'unsup' and 'multi', standing for unsupervised models and the ability to handle more than two classes; and 'disc' and 'sup', the latter of which indicates supervised models.

In classification, features may be differently correlated depending on the class. For instance, it is conceivable that for somebody whose last name is Hilton and who works for the Paris city council, e-mails with just the word 'Paris' or just the word 'Hilton'

Random variables describe possible outcomes of a random process. They can be either discrete (e.g., the possible outcomes of rolling a die are $\{1,2,3,4,5,6\}$) or continuous (e.g., the possible outcomes of measuring somebody's weight in kilograms). Random variables do not need to range over integer or real numbers, but it does make the mathematics quite a bit simpler so that is what we assume here.

If X is a discrete random variable with probability distribution $P(X)$ then the *expected value* of X is $\mathbb{E}[X] = \sum_x xP(x)$. For instance, the expected value of tossing a fair die is $1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + \ldots + 6 \cdot \frac{1}{6} = 3.5$. Notice that this is not actually a possible outcome. For a continuous random variable we need to replace the sum with an integral, and the probability distribution with a probability density function: $\mathbb{E}[X] = \int_{-\infty}^{+\infty} xp(x)\,dx$. The idea of this rather abstract concept is that if we take a sample $x_1, \ldots, x_n$ of outcomes of the random process, the expected value is what we expect the *sample mean* $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ to be – this is the celebrated *law of large numbers* first proved by Jacob Bernoulli in 1713. For this reason the expected value is often called the *population mean*, but it is important to realise that the latter is a theoretical value, while the sample mean is an empirical *estimate* of that theoretical value.

The expectation operator can be applied to functions of random variables. For instance, the (population) *variance* of a discrete random variable is defined as $\mathbb{E}\left[(X-\mathbb{E}[X])^2\right] = \sum_x (x-\mathbb{E}[X])^2 P(x)$ – this measures the spread of the distribution around the expected value. Notice that

$$\mathbb{E}\left[(X-\mathbb{E}[X])^2\right] = \sum_x (x-\mathbb{E}[X])^2 P(x) = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$$

We can similarly define the *sample variance* as $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$, which decomposes as $\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \overline{x}^2$. You will sometimes see the sample variance defined as $\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$: dividing by $n-1$ rather than n results in a slightly larger estimate, which compensates for the fact that we are calculating the spread around the sample mean rather than the population mean.

The (population) *covariance* between two discrete random variables X and Y is defined as $\mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])] = \mathbb{E}[X \cdot Y] - \mathbb{E}[X] \cdot \mathbb{E}[Y]$ The variance of X is a special case of this, with $Y = X$. Unlike the variance, the covariance can be positive as well as negative. Positive covariance means that both variables tend to increase or decrease together; negative covariance means that if one variable increases, the other tends to decrease. If we have a sample of pairs of values of X and Y, *sample covariance* is defined as $\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) = \frac{1}{n}\sum_{i=1}^{n} x_i y_i - \overline{x}\,\overline{y}$. By dividing the covariance between X and Y by $\sqrt{\sigma_X^2 \sigma_Y^2}$ we obtain the *correlation coefficient*, which is a number between -1 and $+1$.

Background 1.3. Expectations and estimators.

are indicative of ham, whereas e-mails with both terms are indicative of spam. Put differently, within the spam class these features are positively correlated, while within the ham class they are negatively correlated. In such a case, ignoring these interactions will be detrimental for classification performance. In other cases, feature correlations may obscure the true model – we shall see examples of this later in the book. On the other hand, feature correlation sometimes helps us to zoom in on the relevant part of the instance space.

There are other ways in which features can be related. Consider the following three features that can be true or false of a molecular compound:

1. it has a carbon in a six-membered aromatic ring;
2. it has a carbon with a partial charge of -0.13;
3. it has a carbon in a six-membered aromatic ring with a partial charge of -0.13.

We say that the third feature is more *specific* (or less *general*) than the other two, because if the third feature is true, then so are the first and the second. However, the converse does not hold: if both first and second feature are true, the third feature may still be false (because the carbon in the six-membered ring may not be the same as the one with a partial charge of -0.13). We can exploit these relationships when searching for features to add to our logical model. For instance, if we find that the third feature is true of a particular negative example that we're trying to exclude, then there is no point in considering the more general first and second features, because they will not help us in excluding the negative either. Similarly, if we find that the first feature is false of a particular positive we're trying to include, there is no point in considering the more specific third feature instead. In other words, these relationships help us to structure our search for predictive features.

1.4 Summary and outlook

My goal in this chapter has been to take you on a tour to admire the machine learning landscape, and to raise your interest sufficiently to want to read the rest of the book. Here is a summary of the things we have been looking at.

- ☞ Machine learning is about using the right features to build the right models that achieve the right tasks. These tasks include: binary and multi-class classification, regression, clustering and descriptive modelling. Models for the first few of these tasks are learned in a supervised fashion requiring labelled training data. For instance, if you want to train a spam filter using machine learning, you need a training set of e-mails labelled spam and ham. If you want to know how good the model is you also need labelled test data that is distinct from the training data, as evaluating your model on the data it was trained on will paint too rosy a picture: a test set is needed to expose any overfitting that occurs.

☞ Unsupervised learning, on the other hand, works with unlabelled data and so there is no test data as such. For instance, to evaluate a particular partition of data into clusters, one can calculate the average distance from the cluster centre. Other forms of unsupervised learning include learning associations (things that tend to occur together) and identifying hidden variables such as film genres. Overfitting is also a concern in unsupervised learning: for instance, assigning each data point its own cluster will reduce the average distance to the cluster centre to zero, yet is clearly not very useful.

☞ On the output side we can distinguish between predictive models whose outputs involve the target variable and descriptive models which identify interesting structure in the data. Often, predictive models are learned in a supervised setting while descriptive models are obtained by unsupervised learning methods, but there are also examples of supervised learning of descriptive models (e.g., subgroup discovery which aims at identifying regions with an unusual class distribution) and unsupervised learning of predictive models (e.g., predictive clustering where the identified clusters are interpreted as classes).

☞ We have loosely divided machine learning models into geometric models, probabilistic models and logical models. Geometric models are constructed in Cartesian instance spaces, using geometric concepts such as planes and distances. The prototypical geometric model is the basic linear classifier, which constructs a decision plane orthogonal to the line connecting the positive and negative centres of mass. Probabilistic models view learning as a process of reducing uncertainty using data. For instance, a Bayesian classifier models the posterior distribution $P(Y|X)$ (or its counterpart, the likelihood function $P(X|Y)$) which tells me the class distribution Y after observing the feature values X. Logical models are the most 'declarative' of the three, employing if–then rules built from logical conditions to single out homogeneous areas in instance space.

☞ We have also introduced a distinction between grouping and grading models. Grouping models divide the instance space into segments which are determined at training time, and hence have a finite resolution. On each segment, grouping models usually fit a very simple kind of model, such as 'always predict this class'. Grading models fit a more global model, graded by the location of an instance in instance space (typically, but not always, a Cartesian space). Logical models are typical examples of grouping models, while geometric models tend to be grading in nature, although this distinction isn't clear-cut. While this sounds very abstract at the moment, the distinction will become much clearer when we discuss coverage curves in the next chapter.

☞ Last but not least, we have discussed the role of features in machine learning. No model can exist without features, and sometimes a single feature is enough to build a model. Data doesn't always come with ready-made features, and often we have to transform or even construct features. Because of this, machine learning is often an iterative process: we only know we have captured the right features after we have constructed the model, and if the model doesn't perform satisfactorily we need to analyse its performance to understand in what way the features need to be improved.

What you'll find in the rest of the book

In the next nine chapters, we will follow the structure laid out above, and look in detail at

☞ machine learning tasks in Chapters 2 and 3;

☞ logical models: concept learning in Chapter 4, tree models in Chapter 5 and rule models in Chapter 6;

☞ geometric models: linear models in Chapter 7 and distance-based models in Chapter 8;

☞ probabilistic models in Chapter 9; and

☞ features in Chapter 10.

Chapter 11 is devoted to techniques for training 'ensembles' of models that have certain advantages over single models. In Chapter 12 we will consider a number of methods for what machine learners call 'experiments', which involve training and evaluating models on real data. Finally, in the Epilogue we will wrap up the book and take a look ahead.