1

Pattern analysis

Pattern analysis deals with the automatic detection of patterns in data, and plays a central role in many modern artificial intelligence and computer science problems. By patterns we understand any relations, regularities or structure inherent in some source of data. By detecting significant patterns in the available data, a system can expect to make predictions about new data coming from the same source. In this sense the system has acquired generalisation power by '*learning*' something about the source generating the data. There are many important problems that can only be solved using this approach, problems ranging from bioinformatics to text categorization, from image analysis to web retrieval. In recent years, pattern analysis has become a standard software engineering approach, and is present in many commercial products.

Early approaches were efficient in finding linear relations, while nonlinear patterns were dealt with in a less principled way. The methods described in this book combine the theoretically well-founded approach previously limited to linear systems, with the flexibility and applicability typical of nonlinear methods, hence forming a remarkably powerful and robust class of pattern analysis techniques.

There has been a distinction drawn between statistical and syntactical pattern recognition, the former dealing essentially with vectors under statistical assumptions about their distribution, and the latter dealing with structured objects such as sequences or formal languages, and relying much less on statistical analysis. The approach presented in this book reconciles these two directions, in that it is capable of dealing with general types of data such as sequences, while at the same time addressing issues typical of statistical pattern analysis such as learning from finite samples.

1.1 Patterns in data

1.1.1 Data

This book deals with data and ways to exploit it through the identification of valuable knowledge. By data we mean the output of any observation, measurement or recording apparatus. This therefore includes images in digital format; vectors describing the state of a physical system; sequences of DNA; pieces of text; time series; records of commercial transactions, etc. By knowledge we mean something more abstract, at the level of relations between and patterns within the data. Such knowledge can enable us to make predictions about the source of the data or draw inferences about the relationships inherent in the data.

Many of the most interesting problems in AI and computer science in general are extremely complex often making it difficult or even impossible to specify an explicitly programmed solution. As an example consider the problem of recognising genes in a DNA sequence. We do not know how to specify a program to pick out the subsequences of, say, human DNA that represent genes. Similarly we are not able directly to program a computer to recognise a face in a photo. Learning systems offer an alternative methodology for tackling these problems. By exploiting the knowledge extracted from a sample of data, they are often capable of adapting themselves to infer a solution to such tasks. We will call this alternative approach to software design the *learning methodology*. It is also referred to as the *data driven* or *data based* approach, in contrast to the *theory driven* approach that gives rise to precise specifications of the required algorithms.

The range of problems that have been shown to be amenable to the learning methodology has grown very rapidly in recent years. Examples include text categorization; email filtering; gene detection; protein homology detection; web retrieval; image classification; handwriting recognition; prediction of loan defaulting; determining properties of molecules, etc. These tasks are very hard or in some cases impossible to solve using a standard approach, but have all been shown to be tractable with the learning methodology. Solving these problems is not just of interest to researchers. For example, being able to predict important properties of a molecule from its structure could save millions of dollars to pharmaceutical companies that would normally have to test candidate drugs in expensive experiments, while being able to identify a combination of biomarker proteins that have high predictive power could result in an early cancer diagnosis test, potentially saving many lives.

In general, the field of pattern analysis studies systems that use the learning methodology to discover *patterns in data.* The patterns that are sought include many different types such as classification, regression, cluster analysis (sometimes referred to together as *statistical pattern recognition*), feature extraction, grammatical inference and parsing (sometimes referred to as *syntactical pattern recognition*). In this book we will draw concepts from all of these fields and at the same time use examples and case studies from some of the applications areas mentioned above: bioinformatics, machine vision, information retrieval, and text categorization.

It is worth stressing that while traditional statistics dealt mainly with data in vector form in what is known as *multivariate statistics*, the data for many of the important applications mentioned above are non-vectorial. We should also mention that pattern analysis in computer science has focussed mainly on classification and regression, to the extent that pattern analysis is synonymous with classification in the neural network literature. It is partly to avoid confusion between this more limited focus and our general setting that we have introduced the term *pattern analysis.*

1.1.2 Patterns

Imagine a dataset containing thousands of observations of planetary positions in the solar system, for example daily records of the positions of each of the nine planets. It is obvious that the position of a planet on a given day is not independent of the position of the same planet in the preceding days: it can actually be predicted rather accurately based on knowledge of these positions. The dataset therefore contains a certain amount of redundancy, that is information that can be reconstructed from other parts of the data, and hence that is not strictly necessary. In such cases the dataset is said to be *redundant*: simple laws can be extracted from the data and used to reconstruct the position of each planet on each day. The rules that govern the position of the planets are known as Kepler's laws. Johannes Kepler discovered his three laws in the seventeenth century by analysing the planetary positions recorded by Tycho Brahe in the preceding decades.

Kepler's discovery can be viewed as an early example of pattern analysis, or data-driven analysis. By assuming that the laws are invariant, they can be used to make predictions about the outcome of future observations. The laws correspond to regularities present in the planetary data and by inference therefore in the planetary motion itself. They state that the planets move in ellipses with the sun at one focus; that equal areas are swept in equal times by the line joining the planet to the sun; and that the period P (the time

	D	P	D^2	P^3
Mercury	0.24	0.39	0.058	0.059
Venus	0.62	0.72	0.38	0.39
Earth	1.00	1.00	1.00	1.00
Mars	1.88	1.53	3.53	3.58
Jupiter	11.90	5.31	142.00	141.00
Saturn	29.30	9.55	870.00	871.00

Table 1.1. *An example of a pattern in data: the quantity D^2/P^3 remains invariant for all the planets. This means that we could compress the data by simply listing one column or that we can predict one of the values for new previously unknown planets, as happened with the discovery of the outer planets.*

of one revolution around the sun) and the average distance D from the sun are related by the equation $P^3 = D^2$ for each planet.

Example 1.1 From Table 1.1 we can observe two potential properties of redundant datasets: on the one hand they are *compressible* in that we could construct the table from just one column of data with the help of Kepler's third law, while on the other hand they are *predictable* in that we can, for example, infer from the law the distances of newly discovered planets once we have measured their period. The predictive power is a direct consequence of the presence of the possibly hidden relations in the data. It is these relations once discovered that enable us to predict and therefore manipulate new data more effectively.

Typically we anticipate predicting one feature as a function of the remaining features: for example the distance as a function of the period. For us to be able to do this, the relation must be invertible, so that the desired feature can be expressed as a function of the other values. Indeed we will seek relations that have such an explicit form whenever this is our intention. Other more general relations can also exist within data, can be detected and can be exploited. For example, if we find a general relation that is expressed as an invariant function f that satisfies

$$f(\mathbf{x}) = 0, \tag{1.1}$$

where $\mathbf{x}$ is a data item, we can use it to identify novel or faulty data items for which the relation fails, that is for which $f(\mathbf{x}) \neq 0$. In such cases it is, however, harder to realise the potential for compressibility since it would require us to define a lower-dimensional coordinate system on the manifold defined by equation (1.1).

Kepler's laws are accurate and hold for all planets of a given solar system. We refer to such relations as *exact*. The examples that we gave above included problems such as loan defaulting, that is the prediction of which borrowers will fail to repay their loans based on information available at the time the loan is processed. It is clear that we cannot hope to find an exact prediction in this case since there will be factors beyond those available to the system, which may prove crucial. For example, the borrower may lose his job soon after taking out the loan and hence find himself unable to fulfil the repayments. In such cases the most the system can hope to do is find relations that hold with a certain probability. Learning systems have succeeded in finding such relations. The two properties of compressibility and predictability are again in evidence. We can specify the relation that holds for much of the data and then simply append a list of the exceptional cases. Provided the description of the relation is succinct and there are not too many exceptions, this will result in a reduction in the size of the dataset. Similarly, we can use the relation to make predictions, for example whether the borrower will repay his or her loan. Since the relation holds with a certain probability we will have a good chance that the prediction will be fulfilled. We will call relations that hold with a certain probability *statistical*.

Predicting properties of a substance based on its molecular structure is hindered by a further problem. In this case, for properties such as boiling point that take real number values, the relations sought will necessarily have to be approximate in the sense that we cannot expect an exact prediction. Typically we may hope that the expected error in the prediction will be small, or that with high probability the true value will be within a certain margin of the prediction, but our search for patterns must necessarily seek a relation that is approximate. One could claim that Kepler's laws are approximate if for no other reason because they fail to take general relativity into account. In the cases of interest to learning systems, however, the approximations will be much looser than those affecting Kepler's laws. Relations that involve some inaccuracy in the values accepted are known as *approximate*. For approximate relations we can still talk about prediction, though we must qualify the accuracy of the estimate and quite possibly the probability with which it applies. Compressibility can again be demonstrated if we accept that specifying the error corrections between the value output by the rule and the true value, take less space if they are small.

The relations that make a dataset redundant, that is the laws that we extract by mining it, are called *patterns* throughout this book. Patterns can be deterministic relations like Kepler's exact laws. As indicated above other relations are approximate or only holds with a certain probability. We are interested in situations where exact laws, especially ones that can be described as simply as Kepler's, may not exist. For this reason we will understand a *pattern* to be any relation present in the data, whether it be exact, approximate or statistical.

Example 1.2 Consider the following artificial example, describing some observations of planetary positions in a two dimensional orthogonal coordinate system. Note that this is certainly not what Kepler had in Tycho's data.

x	y	x^2	y^2	xy
0.8415	0.5403	0.7081	0.2919	0.4546
0.9093	−0.4161	0.8268	0.1732	−0.3784
0.1411	−0.99	0.0199	0.9801	−0.1397
−0.7568	−0.6536	0.5728	0.4272	0.4947
−0.9589	0.2837	0.9195	0.0805	−0.272
−0.2794	0.9602	0.0781	0.9219	−0.2683
0.657	0.7539	0.4316	0.5684	0.4953
0.9894	−0.1455	0.9788	0.0212	−0.144
0.4121	−0.9111	0.1698	0.8302	−0.3755
−0.544	−0.8391	0.296	0.704	0.4565

The left plot of Figure 1.1 shows the data in the (x, y) plane. We can make many assumptions about the law underlying such positions. However if we consider the quantity $c_1x^2+c_2y^2+c_3xy+c_4x+c_5y+c_6$ we will see that it is constant for some choice of the parameters, indeed as shown in the left plot of Figure 1.1 we obtain a linear relation with just two features, x^2 and y^2. This would not generally the case if the data were random, or even if the trajectory was following a curve different from a quadratic. In fact this invariance in the data means that the planet follows an elliptic trajectory. By changing the coordinate system the relation has become linear.

In the example we saw how applying a change of coordinates to the data leads to the representation of a pattern changing. Using the initial coordinate system the pattern was expressed as a quadratic form, while in the coordinate system using monomials it appeared as a linear function. The possibility of transforming the representation of a pattern by changing the coordinate system in which the data is described will be a recurrent theme in this book.

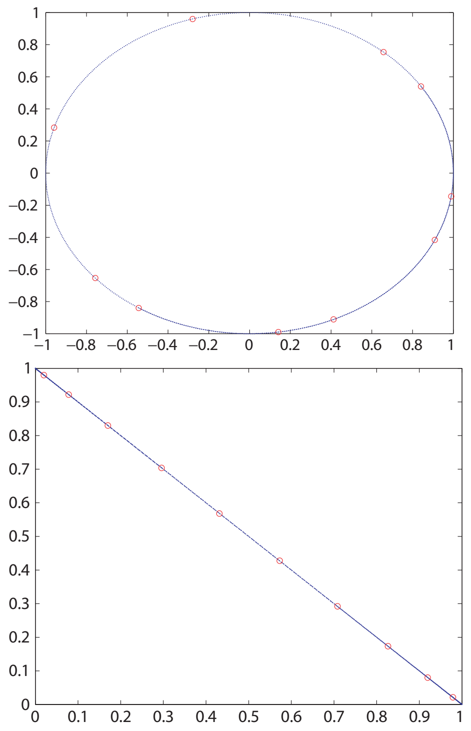

Fig. 1.1. The artificial planetary data lying on an ellipse in two dimensions and the same data represented using the features x^2 and y^2 showing a linear relation

The pattern in the example had the form of a function f that satisfied

$$f(\mathbf{x}) = 0,$$

for all the data points $\mathbf{x}$. We can also express the pattern described by Kepler's third law in this form

$$f(D, P) = D^2 - P^3 = 0.$$

Alternatively

$$g\,(D, P) = 2 \log D - 3 \log P = 0.$$

Similarly, if we have a function g that for each data item $(\mathbf{x}, \mathbf{y})$ predicts some output values $\mathbf{y}$ as a function of the input features $\mathbf{x}$, we can express the pattern in the form

$$f\,(\mathbf{x}, \mathbf{y}) = \mathcal{L}\,(g\,(\mathbf{x})\,, \mathbf{y}) = 0,$$

where $\mathcal{L} : Y \times Y \to \mathbb{R}^+$ is a so-called *loss function* that measures the disagreement between its two arguments outputting 0 if and only if the two arguments are the same and outputs a positive discrepancy if they differ.

Definition 1.3 A general exact pattern for a data source is a non-trivial function f that satisfies

$$f(\mathbf{x}) = 0,$$

for all of the data, $\mathbf{x}$, that can arise from the source. ■

The definition only covers exact patterns. We first consider the relaxation required to cover the case of approximate patterns. Taking the example of a function g that predicts the values $\mathbf{y}$ as a function of the input features $\mathbf{x}$ for a data item $(\mathbf{x}, \mathbf{y})$, if we cannot expect to obtain an exact equality between $g(\mathbf{x})$ and $\mathbf{y}$, we use the loss function $\mathcal{L}$ to measure the amount of mismatch. This can be done by allowing the function to output 0 when the two arguments are similar, but not necessarily identical, or by allowing the function f to output small, non-zero positive values. We will adopt the second approach since when combined with probabilistic patterns it gives a distinct and useful notion of probabilistic matching.

Definition 1.4 A general approximate pattern for a data source is a non-trivial function f that satisfies

$$f(\mathbf{x}) \approx 0$$

for all of the data $\mathbf{x}$, that can arise from the source. ■

We have deliberately left vague what approximately equal to zero might mean in a particular context.

Finally, we consider statistical patterns. In this case there is a probability distribution that generates the data. In many cases the individual data items can be assumed to be generate independently and identically, a case often referred to as independently and identically distributed or i.i.d. for short. We will use the symbol $\mathbb{E}$ to denote the *expectation* of some quantity under a distribution. If we wish to indicate the distribution over which the expectation is taken we add either the distribution or the variable as an index.

Note that our definitions of patterns hold for each individual data item in the case of exact and approximate patterns, but for the case of a statistical pattern we will consider the expectation of a function according to the underlying distribution. In this case we require the pattern function to be positive to ensure that a small expectation arises from small function values and not through the averaging of large positive and negative outputs. This can always be achieved by taking the absolute value of a pattern function that can output negative values.

Definition 1.5 A general statistical pattern for a data source generated i.i.d. according to a distribution $\mathcal{D}$ is a non-trivial non-negative function f that satisfies

$$\mathbb{E}_{\mathcal{D}} f(\mathbf{x}) = \mathbb{E}_{\mathbf{x}} f(\mathbf{x}) \approx 0.$$

■

If the distribution does not satisfy the i.i.d. requirement this is usually as a result of dependencies between data items generated in sequence or because of slow changes in the underlying distribution. A typical example of the first case is time series data. In this case we can usually assume that the source generating the data is ergodic, that is, the dependency decays over time to a probability that is i.i.d. It is possible to develop an analysis that approximates i.i.d. for this type of data. Handling changes in the underlying distribution has also been analysed theoretically but will also be beyond the scope of this book.

Remark 1.6 [Information theory] It is worth mentioning how the patterns we are considering and the corresponding compressibility are related to the traditional study of statistical information theory. Information theory defines the entropy of a (not necessarily i.i.d.) source of data and limits the compressibility of the data as a function of its entropy. For the i.i.d. case it relies on knowledge of the exact probabilities of the finite set of possible items.

Algorithmic information theory provides a more general framework for defining redundancies and regularities in datasets, and for connecting them with the compressibility of the data. The framework considers all computable functions, something that for finite sets of data becomes too rich a class. For in general we do not have access to all of the data and certainly not an exact knowledge of the distribution that generates it. ■

Our information about the data source must rather be gleaned from a finite set of observations generated according to the same underlying distribution. Using only this information a pattern analysis algorithm must be able to identify patterns. Hence, we give the following general definition of a pattern analysis algorithm.

Definition 1.7 [Pattern analysis algorithm] A *Pattern analysis algorithm* takes as input a finite set of examples from the source of data to be analysed. Its output is either an indication that no patterns were detectable in the data, or a positive pattern function f that the algorithm asserts satisfies

$$\mathbb{E}f(\mathbf{x}) \approx 0,$$

where the expectation is with respect to the data generated by the source. We refer to input data examples as the training instances, the training examples or the training data and to the pattern function f as the hypothesis returned by the algorithm. The value of the expectation is known as the generalisation error. ■

Note that the form of the pattern function is determined by the particular algorithm, though of course the particular function chosen will depend on the sample of data given to the algorithm.

It is now time to examine in more detail the properties that we would like a pattern analysis algorithm to possess.

1.2 Pattern analysis algorithms

Identifying patterns in a finite set of data presents very different and distinctive challenges. We will identify three key features that a pattern analysis algorithm will be required to exhibit before we will consider it to be effective.

Computational efficiency Since we are interested in practical solutions to real-world problems, pattern analysis algorithms must be able to handle very large datasets. Hence, it is not sufficient for an algorithm to work well on small toy examples; we require that its performance should scale to large datasets. The study of the computational complexity or scalability of algorithms identifies *efficient algorithms* as those whose resource requirements scale polynomially with the size of the input. This means that we can bound the number of steps and memory that the algorithm requires as a polynomial function of the size of the dataset and other relevant parameters such as the number of features, accuracy required, etc. Many algorithms used in pattern analysis fail to satisfy this apparently benign criterion, indeed there are some for which there is no guarantee that a solution will be found at all. *For the purposes of this book we will require all algorithms to be computationally efficient and furthermore that the degree of any polynomial involved should render the algorithm practical for large datasets.*

Robustness The second challenge that an effective pattern analysis algorithm must address is the fact that in real-life applications data is often corrupted by noise. By noise we mean that the values of the features for individual data items may be affected by measurement inaccuracies or even miscodings, for example through human error. This is closely related to the notion of approximate patterns discussed above, since even if the underlying relation is exact, once noise has been introduced it will necessarily become approximate and quite possibly statistical. *For our purposes we will require that the algorithms will be able to handle noisy data and identify approximate patterns.* They should therefore tolerate a small amount of noise in the sense that it will not affect their output too much. We describe an algorithm with this property as *robust.*

Statistical stability The third property is perhaps the most fundamental, namely that the patterns the algorithm identifies really are genuine patterns of the data source and not just an accidental relation occurring in the finite training set. We can view this property as the *statistical* robustness of the output in the sense that if we rerun the algorithm on a new sample from the same source it should identify a similar pattern. Hence, the output of the algorithm should not be sensitive to the particular dataset, just to the underlying source of the data. For this reason we will describe an algorithm with this property as *statistically stable* or *stable* for short. A relation identified by such an algorithm as a pattern of the underlying source is also referred to as *stable, significant* or *invariant. Again for our purposes we will aim to demonstrate that our algorithms are statistically stable.*

Remark 1.8 [Robustness and stability] There is some overlap between robustness and statistical stability in that they both measure sensitivity of the pattern function to the sampling process. The difference is that robustness emphasise the effect of the sampling on the pattern function itself, while statistical stability measures how reliably the particular pattern function will process unseen examples. We have chosen to separate them as they lead to different considerations in the design of pattern analysis algorithms. ■

To summarise: a pattern analysis algorithm should possess three properties: efficiency, robustness and statistical stability. We will now examine the third property in a little more detail.

1.2.1 Statistical stability of patterns

Proving statistical stability Above we have seen how discovering patterns in data can enable us to make predictions and hence how a stable pattern analysis algorithm can extend the usefulness of the data by learning general properties from the analysis of particular observations. When a learned pattern makes correct predictions about future observations we say that it has *generalised*, as this implies that the pattern has more general applicability. We will also refer to the accuracy of these future predictions as the *quality of the generalization.* This property of an observed relation is, however, a delicate one. Not all the relations found in a given set of data can be assumed to be invariant or stable. It may be the case that a relation has arisen by chance in the particular set of data. Hence, at the heart of pattern analysis is the problem of assessing the reliability of relations and distinguishing them from ephemeral coincidences. How can we be sure we have not been misled by a particular relation we have observed in the given dataset? After all it is always possible to find some relation between any finite set of numbers, even random ones, provided we are prepared to allow arbitrarily complex relations.

Conversely, the possibility of false patterns means there will always be limits to the level of assurance that we are able to give about a pattern's stability.

Example 1.9 Suppose all of the phone numbers stored in your friend's mobile phone are even. If (s)he has stored 20 numbers the probability of this occurring by chance is approximately 2×10^{-6}, but you probably shouldn't conclude that you would cease to be friends if your phone number were changed to an odd number (of course if in doubt, changing your phone number might be a way of putting your friendship to the test).

Pattern analysis and hypothesis testing The pattern analysis algorithm similarly identifies a stable pattern with a proviso that there is a small probability that it could be the result of a misleading dataset. The status of this assertion is identical to that of a statistical test for a property P. The null hypothesis of the test states that P does not hold. The test then bounds the probability that the observed data could have arisen if the null hypothesis is true. If this probability is some small number p, then we conclude that the property does hold subject to the caveat that there is a probability p we were misled by the data. The number p is the so-called *significance* with which the assertion is made. In pattern analysis this probability is referred to as the *confidence parameter* and it is usually denoted with the symbol δ.

If we were testing for the presence of just one pattern we could apply the methodology of a statistical test. Learning theory provides a framework for testing for the presence of one of a set of patterns in a dataset. This at first sight appears a difficult task. For example if we applied the same test for n hypotheses $P_1, \ldots, P_n$, and found that for one of the hypotheses, say P^*, a significance of p is measured, we can only assert the hypothesis with significance np. This is because the data could have misled us about any one of the hypotheses, so that even if none were true there is still a probability p for each hypothesis that it could have appeared significant, giving in the worst case a probability of np that one of the hypotheses appears significant at level p. It is therefore remarkable that learning theory enables us to improve on this worst case estimate in order to test very large numbers (in some cases infinitely many) of hypotheses and still obtain significant results.

Without restrictions on the set of possible relations, proving that a certain pattern is stable is impossible. Hence, to ensure stable pattern analysis we will have to restrict the set of possible relations. At the same time we must make assumptions about the way in which the data is generated by the source. For example we have assumed that there is a fixed distribution and that the data is generated i.i.d. Some statistical tests make the further assumption that the data distribution is Gaussian making it possible to make stronger assertions, but ones that no longer hold if the distribution fails to be Gaussian.

Overfitting At a general level the task of a learning theory is to derive results which enable testing of as wide as possible a range of hypotheses, while making as few assumptions as possible. This is inevitably a trade-off. If we make too restrictive assumptions there will be a misfit with the source and hence unreliable results or no detected patterns. This may be because for example the data is not generated in the manner we assumed; say a test that assumes a Gaussian distribution is used for non-Gaussian data or because we have been too miserly in our provision of hypotheses and failed to include any of the patterns exhibited by the source. In these cases we say that we have *underfit* the data. Alternatively, we may make too few assumptions either by assuming too much flexibility for the way in which the data is generated (say that there are interactions between neighbouring examples) or by allowing too rich a set of hypotheses making it likely that there will be a chance fit with one of them. This is called *overfitting* the data.

In general it makes sense to use all of the known facts about the data, though in many cases this may mean eliciting domain knowledge from experts. In the next section we describe one approach that can be used to incorporate knowledge about the particular application domain.

1.2.2 Detecting patterns by recoding

As we have outlined above if we are to avoid overfitting we must necessarily bias the learning machine towards some subset of all the possible relations that could be found in the data. It is only in this way that the probability of obtaining a chance match on the dataset can be controlled. This raises the question of how the particular set of patterns should be chosen. This will clearly depend on the problem being tackled and with it the dataset being analysed. The obvious way to address this problem is to attempt to elicit knowledge about the types of patterns that might be expected. These could then form the basis for a matching algorithm.

There are two difficulties with this approach. The first is that eliciting possible patterns from domain experts is not easy, and the second is that it would mean designing specialist algorithms for each problem.

An alternative approach that will be exploited throughout this book follows from the observation that *regularities can be translated.* By this we mean that they can be rewritten into different regularities by changing the representation of the data. We have already observed this fact in the example of the planetary ellipses. By representing the data as a feature vector of monomials of degree two, the ellipse became a linear rather than a quadratic pattern. Similarly, with Kepler's third law the pattern becomes linear if we include $\log D$ and $\log P$ as features.

Example 1.10 The most convincing example of how the choice of representation can make the difference between learnable and non-learnable patterns is given by cryptography, where explicit efforts are made to find representations of the data that appear random, unless the right representation, as revealed by the key, is known. In this sense, pattern analysis has the opposite task of finding representations in which the patterns in the data are made sufficiently explicit that they can be discovered automatically.

It is this viewpoint that suggests the alternative strategy alluded to above. Rather than devising a different algorithm for each problem, we fix on a standard set of algorithms and then transform the particular dataset into a representation suitable for analysis using those standard algorithms. The advantage of this approach is that we no longer have to devise a new algorithm for each new problem, but instead we must search for a recoding of the data into a representation that is suited to the chosen algorithms. For the algorithms that we will describe this turns out to be a more natural task in which we can reasonably expect a domain expert to assist. A further advantage of the approach is that much of the efficiency, robustness and stability analysis can be undertaken in the general setting, so that the algorithms come already certified with the three required properties.

The particular choice we fix on is the use of patterns that are determined by linear functions in a suitably chosen feature space. Recoding therefore involves selecting a feature space for the linear functions. The use of linear functions has the further advantage that it becomes possible to specify the feature space in an indirect but very natural way through a so-called *kernel function*. The kernel technique introduced in the next chapter makes it possible to work directly with objects such as biosequences, images, text data, etc. It also enables us to use feature spaces whose dimensionality is more than polynomial in the relevant parameters of the system, even though the computational cost remains polynomial. This ensures that even though we are using linear functions the flexibility they afford can be arbitrarily extended.

Our approach is therefore to design a set of efficient pattern analysis algorithms for patterns specified by linear functions in a kernel-defined feature space. Pattern analysis is then a two-stage process. First we must recode the data in a particular application so that the patterns become representable with linear functions. Subsequently, we can apply one of the standard linear pattern analysis algorithms to the transformed data. The resulting class of pattern analysis algorithms will be referred to as *kernel methods*.

1.3 Exploiting patterns

We wish to design pattern analysis algorithms with a view to using them to make predictions on new previously unseen data. For the purposes of benchmarking particular algorithms the unseen data usually comes in the form of a set of data examples from the same source. This set is usually referred to as the *test set*. The performance of the pattern function on random data from the source is then estimated by averaging its performance on the test set. In a real-world application the resulting pattern function would of course be applied continuously to novel data as they are received by the system. Hence, for example in the problem of detecting loan defaulters, the pattern function returned by the pattern analysis algorithm would be used to screen loan applications as they are received by the bank.

We understand by pattern analysis this process in all its various forms and applications, regarding it as synonymous with Machine Learning, at other times as Data Mining, Pattern Recognition or Pattern Matching; in many cases the name just depends on the application domain, type of pattern being sought or professional background of the algorithm designer. By drawing these different approaches together into a unified framework many correspondences and analogies will be made explicit, making it possible to extend the range of pattern types and application domains in a relatively seamless fashion.

The emerging importance of this approach cannot be over-emphasised. It is not an exaggeration to say that it has become a standard software engineering strategy, in many cases being the only known method for solving a particular problem. The entire Genome Project, for example, relies on pattern analysis techniques, as do many web applications, optical character recognition (OCR) systems, marketing analysis techniques, and so on. The use of such techniques is already very extensive, and with the increase in the availability of digital information expected in the next years, it is clear that it is destined to grow even further.

1.3.1 The overall strategy

All the conceptual issues discussed in the previous sections have arisen out of practical considerations in application domains. We have seen that we must incorporate some prior insights about the regularities in the source generating the data in order to be able to reliably detect them. The question therefore arises as to what assumptions best capture that prior knowledge and/or expectations. How should we model the data generation process and how can we ensure we are searching the right class of relations? In other words, how should we insert domain knowledge into the system, while still ensuring that the desiderata of efficiency, robustness and stability can be delivered by the resulting algorithm? There are many different approaches to these problems, from the inferring of logical rules to the training of neural networks; from standard statistical methods to fuzzy logic. They all have shown impressive results for particular types of patterns in particular domains.

What we will present, however, is a *novel, principled and unified* approach to pattern analysis, based on statistical methods that ensure stability and robustness, optimization techniques that ensure computational efficiency and enables a straightforward incorporation of domain knowledge. Such algorithms will offer many advantages: from the firm theoretical underpinnings of their computational and generalization properties, to the software engineering advantages offered by the modularity that decouples the inference algorithm from the incorporation of prior knowledge into the kernel.

We will provide examples from the fields of bioinformatics, document analysis, and image recognition. While highlighting the applicability of the methods, these examples should not obscure the fact that the techniques and theory we will describe are entirely general, and can in principle be applied to any type of data. This flexibility is one of the major advantages of kernel methods.

1.3.2 Common pattern analysis tasks

When discussing what constitutes a pattern in data, we drew attention to the fact that the aim of pattern analysis is frequently to predict one feature of the data as a function of the other feature values. It is therefore to be expected that many pattern analysis tasks isolate one feature that it is their intention to predict. Hence, the training data comes in the form

$$(\mathbf{x}, y),$$

where y is the value of the feature that the system aims to predict, and $\mathbf{x}$ is a vector containing the remaining feature values. The vector $\mathbf{x}$ is known as the *input*, while y is referred to as the *target output* or *label*. The test data will only have inputs since the aim is to predict the corresponding output values.

Supervised tasks The pattern analysis tasks that have this form are referred to as *supervised*, since each input has an associated label. For this type of task a pattern is sought in the form

$$f(\mathbf{x}, y) = \mathcal{L}(y, g(\mathbf{x})),$$

where g is referred to as the *prediction function* and $\mathcal{L}$ is known as a *loss function*. Since it measures the discrepancy between the output of the prediction function and the correct value y, we may expect the loss to be close to zero when a pattern is detected. When new data is presented the target output is not available and the pattern function is used to predict the value of y for the given input $\mathbf{x}$ using the function $g(\mathbf{x})$. The prediction that $f(\mathbf{x}, y) = 0$ implies that the discrepancy between $g(\mathbf{x})$ and y is small.

Different supervised pattern analysis tasks are distinguished by the type of the feature y that we aim to predict. *Binary classification*, refering to the case when $y \in \{-1, 1\}$, is used to indicate that the input vector belongs to a chosen category ($y = +1$), or not ($y = -1$). In this case we use the so-called discrete loss function that returns 1 if its two arguments differ and 0 otherwise. Hence, in this case the generalisation error is just the probability that a randomly drawn test example is misclassified. If the training data is labelled as belonging to one of N classes and the system must learn to assign new data points to their class, then y is chosen from the set $\{1, 2, \ldots, N\}$ and the task is referred to as *multiclass classification*. *Regression* refers to the case of supervised pattern analysis in which the unknown feature is real-valued, that is $y \in \mathbb{R}$. The term regression is also used to describe the case when $\mathbf{y}$ is vector valued, $\mathbf{y} \in \mathbb{R}^n$, for some $n \in \mathbb{N}$, though this can also be reduced to n separate regression tasks each with one-dimensional output but with potentially a loss of useful information. Another variant of regression is time-series analysis. In this case each example consists of a series of observations and the special feature is the value of the next observation in the series. Hence, the aim of pattern analysis is to make a forecast based on previous values of relevant features.

Semisupervised tasks In some tasks the distinguished feature or label is only partially known. For example in the case of *ranking* we may only have available the relative ordering of the the examples in the training set, while our aim is to enable a similar ordering of novel data. For this problem an underlying value function is often assumed and inference about its value for the training data is made during the training process. New data is then assessed by its value function output. Another situation in which only partial information is available about the labels is the case of *transduction*. Here only some of the data comes with the value of the label instantiated. The task may be simply to predict the label for the unlabelled data. This corresponds to being given the test data during the training phase.

Alternatively, the aim may be to make use of the unlabelled data to improve the ability of the pattern function learned to predict the labels of new data. A final variant on partial label information is the *query scenario* in which the algorithm can ask for an unknown label, but pays a cost for extracting this information. The aim here is to minimise a combination of the generalization error and querying cost.

Unsupervised tasks In contrast to supervised learning some tasks do not have a label that is only available for the training examples and must be predicted for the test data. In this case all of the features are available in both training and test data. Pattern analysis tasks that have this form are referred to as *unsupervised.* The information or pattern needs to be extracted without the highlighted ‘external’ information provided by the label. *Clustering* is one of the tasks that falls into this category. The aim here is to find a natural division of the data into homogeneous groups. We might represent each cluster by a centroid or prototype and measure the quality of the pattern by the expected distance of a new data point to its nearest prototype.

Anomaly or novelty-detection is the task of detecting new data points that deviate from the normal. Here, the exceptional or anomalous data are not available in the training phase and are assumed not to have been generated by the same source as the rest of the data. The task is tackled by finding a pattern function that outputs a low expected value for examples generated by the data source. If the output generated by a new example deviates significantly from its expected value, we identify it as exceptional in the sense that such a value would be very unlikely for the standard data. Novelty-detection arises in a number of different applications. For example engine monitoring attempts to detect abnormal engine conditions that may indicate the onset of some malfunction.

There are further unsupervised tasks that attempt to find low-dimensional representations of the data. Here the aim is to find a projection function P_V that maps X into a space V of a given fixed dimension k

$$P_V : X \longrightarrow V,$$

such that the expected value of the residual

$$f(\mathbf{x}) = \|P_V(\mathbf{x}) - \mathbf{x}\|^2$$

is small, or in other words such that f is a pattern function. The kernel principal components analysis (PCA) falls into this category.

A related method known as kernel canonical correlation analysis (CCA) considers data that has separate representations included in each input, for example $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$ for the case when there are two representations. CCA now seeks a common low-dimensional representation described by two projections P_V^A and P_V^B such that the residual

$$f(\mathbf{x}) = \left\|P_V^A\left(\mathbf{x}^A\right) - P_V^B\left(\mathbf{x}^B\right)\right\|^2$$

is small. The advantage of this method becomes apparent when the two representations are very distinct but our prior knowledge of the data assures us that the patterns of interest are detectable in both. In such cases the projections are likely to pick out dimensions that retain the information of interest, while discarding aspects that distinguish the two representations and are hence irrelevant to the analysis.

Assumptions and notation We will mostly make the statistical assumption that the sample of data is drawn i.i.d. and we will look for statistical patterns in the data, hence also handling approximate patterns and noise. As explained above this necessarily implies that the patterns are only identified with high probability. In later chapters we will define the corresponding notions of generalization error.

Now we introduce some of the basic notation. We denote the input space by X and for supervised tasks use Y to denote the target output domain. The space X is often a subset of $\mathbb{R}^n$, but can also be a general set. Note that if X is a vector space, the input vectors are given as column vectors. If we wish to form a row vector for an instance $\mathbf{x}$, we can take the transpose $\mathbf{x}'$. For a supervised task the *training set* is usually denoted by

$$S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\} \subseteq (X \times Y)^\ell ,$$

where ℓ is the number of training examples. For unsupervised tasks this simplifies to

$$S = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\} \subseteq X^\ell .$$

1.4 Summary

- Patterns are regularities that characterise the data coming from a particular source. They can be exact, approximate or statistical. We have chosen to represent patterns by a positive pattern function f that has small expected value for data from the source.
- A pattern analysis algorithm takes a finite sample of data from the source and outputs a detected regularity or pattern function.
- Pattern analysis algorithms are expected to exhibit three key properties: efficiency, robustness and stability.

  Computational efficiency implies that the performance of the algorithm scales to large datasets.
  Robustness refers to the insensitivity of the algorithm to noise in the training examples.
  Statistical stability implies that the detected regularities should indeed be patterns of the underlying source. They therefore enable prediction on unseen data.

- Recoding, by for example a change of coordinates, maintains the presence of regularities in the data, but changes their representation. Some representations make regularities easier to detect than others and fixing on one form enables a standard set of algorithms and analysis to be used.
- We have chosen to recode relations as linear patterns through the use of kernels that allow arbitrary complexity to be introduced by a natural incorporation of domain knowledge.
- The standard scenarios in which we want to exploit patterns in data include binary and multiclass classification, regression, novelty-detection, clustering, and dimensionality reduction.

1.5 Further reading and advanced topics

Pattern analysis (or recognition, detection, discovery) has been studied in many different contexts, from statistics to signal processing, to the various flavours of artificial intelligence. Furthermore, many relevant ideas have been developed in the neighboring fields of information theory, machine vision, data-bases, and so on. In a way, pattern analysis has always been a constant theme of computer science, since the pioneering days. The references [39], [40], [46], [14], [108], [38], [45] are textbooks covering the topic from some of these different fields.

There are several important stages that can be identified in the evolution of pattern analysis algorithms. Efficient algorithms for detecting linear relations were already used in the 1950s and 1960s, and their computational and statistical behaviour was well understood [109], [44]. The step to handling nonlinear relations was seen as a major research goal at that time. The development of nonlinear algorithms that maintain the same level of efficiency and stability has proven an elusive goal. In the mid 80s the field of pattern analysis underwent a *nonlinear revolution*, with the almost simultaneous introduction of both backpropagation networks and decision trees [19], [107], [55]. Although based on simple heuristics and lacking a firm theoretical foundation, these approaches were the first to make a step towards the efficient and reliable detection of nonlinear patterns. The impact of that revolution cannot be overemphasized: entire fields such as data-mining and bioinformatics became possible as a result of it. In the mid 90s, the introduction of kernel-based learning methods [141], [16], [32], [118] has finally enabled researchers to deal with nonlinear relations, while retaining the guarantees and understanding that have been developed for linear algorithms over decades of research.

From all points of view, computational, statistical, and conceptual, the nonlinear pattern analysis algorithms developed in this third wave are as efficient and as well-founded as their linear counterparts. The drawbacks of local minima and incomplete statistical analysis that is typical of neural networks and decision trees have been circumvented, while their flexibility has been shown to be sufficient for a wide range of successful applications. In 1973 Duda and Hart defined statistical pattern recognition in the context of classification in their classical book, now available in a new edition [40]. Other important references include [135], [46]. Algorithmic information theory defines random data as data not containing any pattern, and provides many insights for thinking about regularities and relations in data. Introduced by Chaitin [22], it is discussed in the introductory text by Li and Vitani [90]. A classic introduction to Shannon's information theory can be found in Cover and Thomas [29].

The statistical study of pattern recognition can be divided into two main (but strongly interacting) directions of research. The earlier one is that presented by Duda and Hart [40], based on bayesian statistics, and also to be found in the recent book [51]. The more recent method based on empirical processes, has been pioneered by Vapnik and Chervonenkis's work since the 1960s, [139], and has recently been greatly extended by several authors. Easy introductions can be found in [74], [5], [139]. The most recent (and most effective) methods are based on the notions of sharp concentration [38], [17] and notions of Rademacher complexity [9], [78], [132], [133].

The second direction will be the one followed in this book for its simplicity, elegance and effectiveness. Other discussions of pattern recognition via specific algorithms can be found in the following books: [14] and [108] for neural networks; [107] and [19] for decision trees, [32], and [100] for a general introduction to the field of machine learning from the perspective of artificial intelligence.

More information about Kepler's laws and the process by which he arrived at them can be found in a book by Arthur Koestler [76].

For constantly updated pointers to online literature and free software see the book's companion website: www.kernel-methods.net