1

Introduction and Overview

Plus ça change, plus c'est la meme chose.

—Jean-Baptiste Alphonse Karr (1849)

There's even exponential growth in the rate of exponential growth.

—Ray Kurzweil (2001)

1.1 Introduction

Services, businesses, analytics, and other types of data services have moved from workstations, local area networks (LANs), and in-house IT infrastructure to the Internet, mobile devices, and more recently "the Cloud." This has broad implications in the privacy, security, versioning, and ultimately the long-term fate of data. These changes, however, provide a welcome opportunity for reconsidering the manner in which intelligent systems are designed, built and tested, deployed, and optimized during deployment. With the advent of powerful machine learning capabilities in the past two decades, it has become clear to the research community that learning algorithms, and systems based in all or part on these algorithms, are not only possible but also *essential* for modern business. However, the combined impact of mobile devices, ubiquitous services, and the cloud comprise a fundamental change in how systems themselves can—and I argue should—be designed. Services themselves can be transformed into learning systems, adaptive not just in terms of the specific parameters of their applications and algorithms but also in the repertoire (or set) and relationship (or architecture) between multiple applications and algorithms in the service.

With the nearly unlimited computing and data hosting possibilities now feasible, hitherto processor-bound and memory-bound applications, services, and decision-making (actionable analytics) approaches are now freed from many of their limitations. In fact, the cloud- and graphical processing unit (GPU)-based computation have made possible parallel processing on a grand scale. In recognition of this new reality, this book focuses on the algorithmic, analytic, and system patterns that can be used to better take advantage of this new norm of parallelism, and will help to move the fields of machine learning, analytics, inference, and classification to more squarely align with this new norm.

Meta-algorithmics: Patterns for Robust, Low-Cost, High-Quality Systems, First Edition. Steven J. Simske.

In this chapter, I overview at an often high, but thematic, level the broad fields of machine intelligence, artificial intelligence, data mining, classification, recognition, and systems-based analysis. Standing on the shoulders of giants who have pioneered these fields before this book, the intent is to highlight the salient differences in multiple approaches to useful solutions in each of these arenas. Through this approach, I intend to engage all interested readers—from the interested newcomer to the field of intelligent systems design to the expert with far deeper experience than myself in one or more of these arenas—in the central themes of this book. In short, these themes are:

1. Instead of finding the best possible intelligent algorithm, system, or engine for a task, the system architect should look for the best combination of algorithms, systems, or engines to provide the best accuracy, robustness, adaptability, cost, and so on.
2. Parallel approaches to machine-intelligence-driven tasks such as data mining, classification, and recognition naturally lead to parallelism by task, parallelism by component, and eventually parallelism by meta-algorithmics.
3. Meta-algorithmic approaches and patterns provide a toolbox of potential solutions for intelligent systems design and deployment—accommodating architects of widely varying domain expertise, widely varying mathematical background, and widely varying experience with system design.

1.2 Why Is This Book Important?

Jean-Baptiste Alphonse Karr, right after the 1848 Revolutions rocked Europe, made the famous observation that "the more that things change, the more they stay the same." This statement anticipated Darwin's treatise on the Origin of Species by a decade, and is germane to this day. In the fast-changing world of the twenty-first century, in which Ray Kurzweil's musing on the rapidly increasing growth in the rate of growth is nearly cliché and Luddite musings on humanity losing control of data are *de rigueur*, perhaps it may be time to reconsider how large systems are architected. Designing a system to be robust to change—to anticipate change—may also be the right path to designing a system that is optimized for accuracy, cost, and other important performance parameters. One objective of this book is to provide a straightforward means of designing and building intelligent systems that are optimized for changing system requirements (adaptability), optimized for changing system input (robustness), and optimized for one or more other important system parameters (e.g., accuracy, efficiency, and cost). If such an objective can be achieved, then rather than being insensitive to change, the system will *benefit* from change. This is important because more and more, every system of value is actually an *intelligent* system.

The vision of this book is to provide a practical, systems-oriented, statistically driven approach to parallel—and specifically meta-algorithmics-driven—machine intelligence, with a particular emphasis on classification and recognition. Three primary types of parallelism will be considered: (1) parallelism by task—that is, the assignment of multiple, usually different tasks to parallel pipelines that would otherwise be performed sequentially by the same processor; (2) parallelism by component—wherein a larger machine intelligence task is assigned to a set of parallel pipelines, each performing the same task but on a different data set; and (3) parallelism by meta-algorithmics. This last topic—parallelism by meta-algorithmics—is

in practice far more open to art as it is still both art and science. In this book, I will show how meta-algorithmics extend the more traditional forms of parallelism and, as such, can complement the other forms of parallelism to create better systems.

1.3 Organization of the Book

The book is organized in 11 chapters. In this first chapter, I provide the aims of the book and connect the material to the long, impressive history of research in other fields salient to intelligent systems. This is accomplished by reviewing this material in light of the book's perspective. In Chapter 2, I provide an overview of parallelism, especially considering the impact of GPUs, multi-core processors, virtualism, and cloud computing on the fundamental approaches for intelligent algorithm, system and service design. I complete the overview chapters of the book in Chapter 3, wherein I review the application domains within which I will be applying the different forms of parallelism in later chapters. This includes primary domains of focus selected to illustrate the depth of the approaches, and secondary domains to illustrate more fully the breadth. The primary domains are (1) document understanding, (2) image understanding, (3) biometrics, and (4) security printing. The secondary domains are (1) image segmentation, (2) speech recognition, (3) medical signal processing, (4) medical imaging, (5) natural language processing (NLP), (6) surveillance, (7) optical character recognition (OCR), and (8) security analytics. Of these primary and secondary domains, I end in each case with the security-related topics, as they provide perhaps the broadest, most interdisciplinary needs, thus affording an excellent opportunity to illustrate the design and development of complex systems.

In the next three chapters, the three broad types of parallelism are described and applied to the domains described in Chapter 3. Chapter 4 will address Parallelism by Task, which focuses on the use of multiple instances of (usually the same) data being analyzed in parallel by different algorithms, services, or intelligent engines. This chapter will also outline the advantages that cloud computing brings to this type of parallelism—namely, the ability to produce actionable output limited by the throughput of the slowest process. Chapter 5 then addresses Parallelism by Component, in which different partitions of the same data set are processed in parallel. The advances provided by GPUs will be highlighted in this chapter. The third and final broad category of parallelism—Parallelism by Meta-algorithm—will be introduced in Chapter 6. Because of their importance to the rest of the book, these approaches will be elaborated as belonging to three different classes—first-, second-, and third-order meta-algorithms—each with a specific set of design patterns for application. These patterns will be introduced in Chapter 6 before they are then applied to the domains of interest in the three chapters that follow.

In Chapter 7, the first-order meta-algorithmic patterns are explored. These relatively simple means of combining two or more sources of knowledge generation—algorithms, engines, systems, and so on—are shown to be generally applicable even when the combined generators are known only at the level of black box (input and output only). One pattern, Tessellation and Recombination, is shown to be especially useful for creating correct results even when none of the individual generators produces a correct result—a process called emergence. This pattern bridges us to the potentially more powerful patterns of Chapters 8 and 9. In Chapter 8, the second-order meta-algorithms are described. A very powerful tool for temporal and series-parallel design of meta-algorithmic systems, the confusion matrix, is explored in full.

In Chapter 9, third-order meta-algorithmic patterns—generally focused on feedback from the output to input and system-level machine learning—are overviewed.

The book concludes with Chapter 10—elaborating how parallelism by task, component, and meta-algorithm lead to more accurate, cost-sensitive, and/or robust systems—and Chapter 11, which looks to the future of intelligent systems design in light of the previous chapters.

It is clear that, in addition to the more straightforward parallel processing approaches (by task and by component), this book focuses on meta-algorithmics—or pattern-driven means of combining two or more algorithms, classification engines, or other systems. The value of specific patterns for meta-algorithmic systems stems from their ability to stand on the shoulders of the giants of intelligent systems; in particular, the giants in informatics, machine learning, data mining, and knowledge discovery. This book will cover some new theory in order to expostulate the meta-algorithmic approaches, but it is intended to be a practical, engineering-focused book—enough theory will be provided to make the academic reader comfortable with the systems eventually crafted using the meta-algorithmics and other parallelism approaches.

Building big systems for intelligence—knowledge discovery, classification, actionable analytics, and so on—relies on the interplay of many components. Meta-algorithmics position the system architect squarely in the "post-cloud" era, in which processing, storage, analysis, and other traditionally limited computing resources are much less scarce.

In the sections that follow, the background science to which this book owes its existence will be occasionally interpreted in light of meta-algorithmics, which themselves are not introduced until Section 2.5 or fully developed until Chapter 6. This should not be an impediment to reading this chapter, but concerned readers may feel free to look ahead at those sections if they so wish. More importantly, this background science will be reinterpreted in light of the needs of the so-called parallel forms of parallelism that comprise this book. We start with informatics.

1.4 Informatics

Informatics, like the term "analytics," is a broad field of knowledge creation with a plethora of definitions. In keeping with Dreyfus (1962), I herein consider informatics to include the study of algorithms, behavior, interactions, and structure of man-made systems that access, communicate, process, and store/archive information. Informatics is concerned with the timely delivery of the right information to the right person/people at the right time. Informatics, therefore, is innately amenable to parallelism. Figure 1.1 illustrates this in simplified form. Two sets of (one or more) *algorithms* are provided. The internals of these algorithms are not important to the overall *behavior* of the system, which is the function mapping the inputs to the outputs. The *interactions* between the two algorithmic subsystems are also a "black box" to the inputs and outputs, and the overall *structure* of the system is the set of all inputs, outputs, algorithms, and interactions.

Informatics, therefore, is the science of useful transformation of information. This implies that the outputs in Figure 1.1 are more sophisticated than—that is, contain information of increased value in comparison to—the inputs. A simple but still useful example of an informatics system based on the architecture shown in Figure 1.1 is given in Figure 1.2.

In Figure 1.2, a parallel architecture is used even though the task is simple enough to perform with a sequential design. The advantage of the parallel design is that subsections of the original

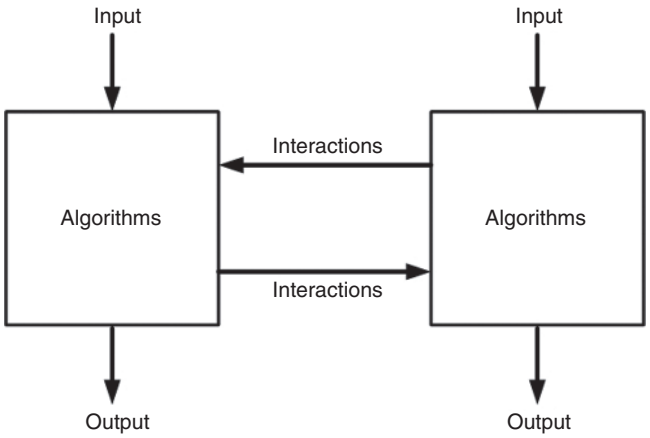

Figure 1.1 Simplified informatics system illustrating algorithms, behavior (mapping from input to output), interactions (data exchange between algorithm blocks), and structure (overall architecture)

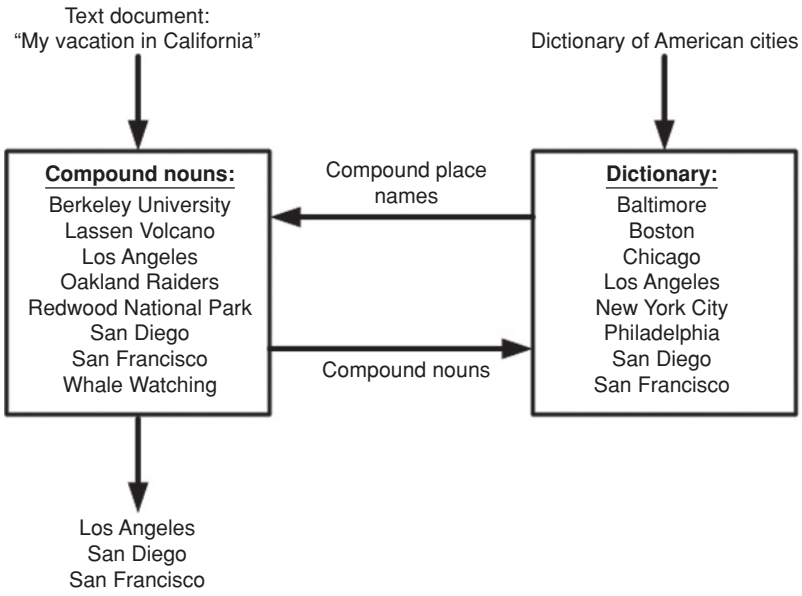

Figure 1.2 A simple system to pull compound place names from a document. The algorithm box to the left, "Compound nouns," extracts compound nouns from the input document. The algorithm box to the right, "Dictionary," inputs a dictionary of terms to search for in its input (which happens to be the output of the compound noun extractor). The terms occurring in each set are returned to the text document algorithm box and output as the set of compound place nouns: "Los Angeles," "San Diego," and "San Francisco"

document after being processed by the left (or "Compound nouns") algorithm, can be input to the right (or "Dictionary") algorithm and processed in parallel to the next subsection being processed by the "Compound nouns" algorithm.

1.5 Ensemble Learning

Informatics-based systems are thus a very general type of intelligent system. In this section, ensemble learning, which focuses on the handling of the output of two or more intelligent systems in parallel, is considered. Two reviews of ensemble learning are of particular utility—those provided by Berk (2004) and Sewell (2007). In Berk (2004), ensemble methods are defined as "bundled fits produced by a stochastic algorithm, the output of which is some combination of a large number of passes through the data." This bundling or combining of the fitted values from a number of fitting attempts is considered an algorithmic approach (Hothorn, 2003). In Berk (2004), classification and regression trees (CART), introduced in Breiman *et al.* (1984), are used to bridge from traditional modeling (e.g., mixture models, manifold-based systems, and others) to algorithmic approaches. Partitioning of the input is used to create subclasses of the input space, which correlate well with one among a plurality of classes. However, partitioning quickly leads to overfitting of the data and concomitant degradation of performance on test data when compared to training data.

To avoid this problem, ensemble methods are used. Bagging, random forests, and boosting are the three primary ensemble methods described in Berk (2004). Bagging, or "bootstrap aggregation," is shown to be definable as a simple algorithm: random samples are drawn N times with replacement and nonpruned classification (decision) trees are created. This process is repeated many times, after which the classification for each case in the overall data set is decided by majority voting. Overfitting is avoided by this "averaging" effect, but perhaps even more importantly by selecting an appropriate margin for the majority voting. This means some cases will go unclassified, but since multiple trees are created, these samples will likely be classified through another case. Should any samples be unassigned, they can be assigned by nearest neighbor or other decisioning approaches. Random forests (Breiman, 2001) further the randomness introduced by bagging via selecting a random subset of predictors to create the node splits during tree creation. They are designed to allow trade-off between bias and variance in the fitted value, with some success (Berk, 2004). Boosting (Schapire, 1999), on the other hand, is derived from a different learning approach, even though it may result in a very similar interpretative ability to that of random forests (Berk, 2004). Boosting is, generally speaking, the process by which the misclassified cases are more highly weighted after each iteration. It is argued that this approach avoids overfitting, and its famous incarnation, the AdaBoost (Freund and Schapire, 1996; Schapire, 1999), has certainly proven accurate in a number of machine learning problems. However, there are some concerns with the approach: the stopping criterion—usually the error value during training—is not always effective, and convergence is not guaranteed.

In Jain, Duin, and Mao (2000), 18 classifier combination schemes are overviewed. Among them are ensemble methods bagging and boosting, voting, and class set reduction. Interestingly, this article mentions the possibility of having individual classifiers use different feature sets and/or operate on different subsets of the input; for example, the random subspace method. This approach lays some of the groundwork for meta-algorithmics.

In Sewell (2007), ensemble learning is defined as an approach combining multiple learners. This review focuses on bagging, boosting, stacked generalization, and the random subset method. Here, Sewell refers to bootstrap aggregating, or bagging, as a "meta-algorithm," which is a special case of model averaging. Viewed this way, the bagging approach can be seen as an incipient form of the Voting meta-algorithmic pattern described in Section 6.2.3. It can be applied to classification or regression. However, as opposed to meta-algorithmic patterns, bagging operates on multiple related algorithms, such as decision stumps, and not on independently derived algorithms. Boosting is also described as a "meta-algorithm" that can be viewed directly as a model averaging approach. It, too, can be used for regression or classification. Boosting's value is in generating strong classifiers from a set of weak learners. This approach is an important part of the rationale for meta-algorithmics in general, as we will see in Chapters 6–9.

Stacked generalization (Wolpert, 1992) extends the training + validation approach to a plurality of base learners. This is a multiple model approach in that rather than implementing the base learner with the highest accuracy during validation, the base learners are combined, often nonlinearly, to create the "meta-learner." This paves the path for meta-algorithmic patterns such as Weighted Voting (Section 6.2.3), although stacked generalization is focused on combining weak learners, whereas meta-algorithmics are focused on combining strong learners, engines or intelligent systems.

The final ensemble method that introduces some of the conceptual framework for meta-algorithmics is the random subspace method (Ho, 1998), in which the original training set input space is partitioned into random subspaces. Separate learning machines are then trained on the subspaces and the meta-model combines the output of the models, usually through majority voting. This shares much in common with the mixture of experts approach (Jacobs *et al.*, 1991), which differs in that it has different components model the distribution in different regions of the input space and the gating function decides how to use these experts. The random subspace method leads to a single model—capable of classification or regression analysis—that can provide high accuracy even in the face of a highly nonlinear input space. Both the random subspace and mixture of experts approaches are analogous in some ways to the Predictive Selection meta-algorithmic approach (Section 6.2.4) and related meta-algorithmic patterns. However, as with the rest of the ensemble methods, these models stop at providing an improved *single model* for data analysis. Meta-algorithmics, on the other hand—as we will see in much of the rest of the book—use the output of ensemble methods, other classifiers, and other intelligent systems as their *starting points*. Meta-algorithmics combine multiple models to make better decisions, meaning that, for example, bagging, boosting, stacked generalization, and random subspace methods, could all be used together to create a more accurate, more robust, and/or more cost-effective system.

1.6 Machine Learning/Intelligence

The distinction between machine learning/intelligence and artificial intelligence is somewhat arbitrary. Here, I have decided to term those approaches that result in a readily interpretable, visible set of coefficients, equations, procedures, and/or system components as "machine learning." Those in which the details of how the system works are hidden are considered "artificial intelligence" systems. For the former, the intelligence is not "artificial," but rather based on an expert system, formula, or algorithm in line with human reasoning; for the latter,

it simply works with an "intelligence" that is not immediately obvious. The distinction is not particularly important other than to allow me to collect some thoughts on these broad topics in a (relatively) structured manner.

For this machine learning overview, I primarily consulted Bishop (2006), Hastie, Tibshirani, and Friedman (2009), and Marsland (2009). While there are many other texts on machine learning, the combination of these three was appealingly broad and deep. In providing different foci, they were just the right balance between rigor and heuristics, and I refer the reader to these books for a far more in-depth coverage of these topics than I can provide in this section. Additional background texts consulted, which helped frame the development of machine and artificial intelligence in the decade leading up to support vector and ensemble methods, included Fogel (1995), Goldberg (1989), Leondes (1998), and Tveter (1998): these provided rich context for the more current state of machine intelligence and also highlighted for me the need for parallel and other hybrid methods.

1.6.1 Regression and Entropy

Regression is perhaps the simplest form of machine intelligence. Linear regression is often the introduction to least squares error, since this is the most common choice of loss function in regression (since least squares estimates have the smallest variance among all linear unbiased estimates). Introductory statistics courses teach the student how to perform linear regression and in so doing determine a least squares best fit model that is described in its entirety by the $\{slope, intercept\}$, or $\{\beta_1, \beta_0\}$. This fits precisely one definition of machine learning above: a readily interpretable, visible set of coefficients. The machine has learned that

$$\hat{y} = \beta_0 + \beta_1 X.$$

That is, the dependent variable y, as well as its estimate $\hat{y}$, is predicted by the independent variable X using only the model coefficients.

One of the more interesting applications of regression is in determining the complexity of the overall data. I define complexity here by the combination of the order of the regression, the residual variability determined by $1.0 - r^2$, and the entropy of the residual variability. Table 1.1 illustrates the residual variability as a ratio of the initial variability and the entropy

Table 1.1 Residual variability as a percentage of the original variability and entropy, $H(\Delta y)$, of the residual variability determined from $H(\Delta y) = -\sum p(\Delta y) \ln p(\Delta y)$, where $p(\Delta y)$ are computed on a histogram of 100 bins (maximum entropy = 4.605)

Order of Regression	Input Space A		Input Space B	
	Residual Variability	Residual Entropy	Residual Variability	Residual Entropy
1	0.432	2.377	0.345	2.561
2	0.167	3.398	0.156	2.856
3	0.055	4.101	0.087	3.019

of the residuals, $\Delta y = |y - y|$, for first-, second-, and third-order regression models of two sets of data, Input Spaces A and B.

The entropy for Table 1.1 is taken for 1% subranges of the overall range of X. The entropy is computed from

$$H(\Delta y) = -\sum p(\Delta y) \ln p(\Delta y),$$

where the maximum entropy is, therefore, $-\ln(0.01) = 4.605$. For Input Space A, the residual variability decreases as the order of regression increases. The entropy increases significantly, as well. These data indicate that much of the variability in the data is explained by a third-order regression: only 5.5% of the variability remains and the remaining error values have high entropy.

For Input Space B, however, the residual variability drops less, to 8.7% after a third-order fit. The entropy also does not increase significantly from second- to third-order best fit, indicating that there is still structure (i.e., a fourth-order or higher fit is appropriate) in the residual error values. Interpreting the data in Table 1.1 in this way is appropriate, since the simple first-, second-, or third-order regression models are a linear combination of fixed basis functions. Since reduced entropy indicates nonrandom structure remains in the data, it indicates that the regression model does not contain sufficient dimensionality, or else that a different model for the regression is needed altogether. This latter possibility is important, as it may mean we need to rethink the approach taken and instead use, for example, a support vector machine (SVM) with a kernel to transform the input space into one in which simpler regression models suffice.

Regardless, this simple example illustrates an important connection between regression and other intelligent system operations, such as classification (Section 1.9) or alternative models (Section 1.7). For example, one may conclude that Input Space A represents a third-order data set, and so gear its classification model accordingly. Input Space B, on the other hand, will require a more complicated classification model. This type of upfront data analysis is critical to any of the three forms of parallelism that are the main topic of this book.

1.6.2 SVMs and Kernels

SVMs are two-class, or "binary," classifiers that are designed to provide simplified decision boundaries between the two classes. Support vectors create boundaries for which the margin between the two classes is maximized, creating what is termed optimal separation. It is obvious that such an approach is highly sensitive to noise for small- and medium-sized data sets, since the only relevant subset of input data—the support vectors—are used to define the boundary and its margin, or spacing to either side of the decision boundary. An example of a support vector for a two-dimensional (2D), two-class data set is given in Figure 1.3.

SVMs (Cortes and Vapnik, 1995; Vapnik, 1995) focus on transforming the input space into a higher-order dimensionality, in which a linear decision boundary is crafted. There is art, and mathematical prowess, involved in creating a decision boundary, analogous to the creation of classification manifolds, as in Belkin and Niyogi (2003), Grimes and Donoho (2005), and elsewhere. As noted in Bishop (2006), direct solution of the optimization problem created by the search for an optimum margin is highly complex, and some classification engineers may prefer genetic, near-exhaustive, and/or artificial neural network (ANN) approaches to the

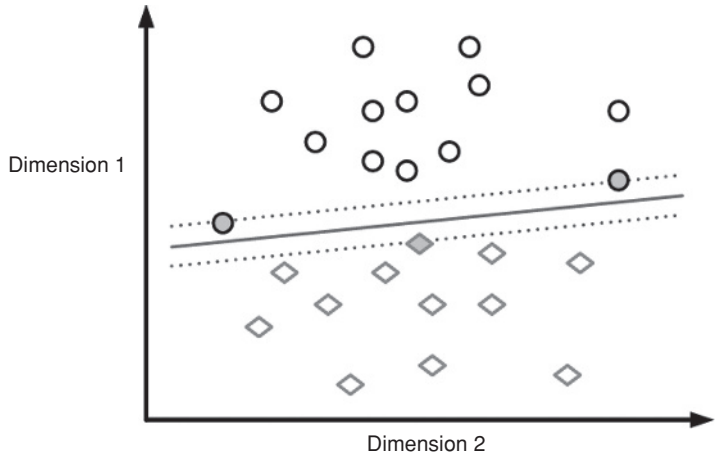

Figure 1.3 Example decision boundary (solid line) with margin to either side (dotted lines) as defined by the support vector (filled circles and diamond). The margin is the maximum width possible for the given data set in which classification errors are entirely eliminated (all circles at least one margin above the decision boundary, and all diamonds at least one margin below the decision boundary)

mathematically precise approach. However, some of this complexity is obviated by reducing the SVM optimization equation to what is known as the canonical representation of the decision hyperplane. This means the optimization is now recrafted as a familiar quadratic programming problem—a second-order function is optimized subject to a set of first-order inequality constraints.

The relationship between an SVM and regression models, such as described in the previous section, is subtle, as shall be illustrated in the next few paragraphs. The SVM is designed to maximize the margin of the decision boundary, as shown in Figure 1.3. As the size of the two populations increases, however, the percentage of points in each population that are being considered in forming the support vector drops. This is analogous to the much lower surface area to volume ratio of, say, an elephant in comparison to a mouse. It is clear that the margin *can* be optimized, but does that mean that it should? The answer to this somewhat troubling, but very important, classification question is, of course, "it depends."

To address this, I will push the SVM ramifications in another direction, trying to connect the decision boundary to regression and principle component analysis. Let us suppose that the margins around the decision boundary define a range of acceptable decision boundary slopes. This is shown in Figure 1.4 for the support vector introduced in Figure 1.3.

Figure 1.4 illustrates one means of using all of the samples in both of the classes to define the optimal decision boundary, and not just the support vectors. The decision boundary is allowed to be redefined such that none of the samples defining the support vector are misclassified, even though the slope of the decision boundary may change. The perpendiculars to the range limits of this new decision boundary are also shown.

In order to connect the range from line C to line D in Figure 1.4, a linear transformation of the data in Figures 1.3 and 1.4 is performed. In Figure 1.5, this transformation is performed by elongating the input space in a direction perpendicular to the original decision boundary. This

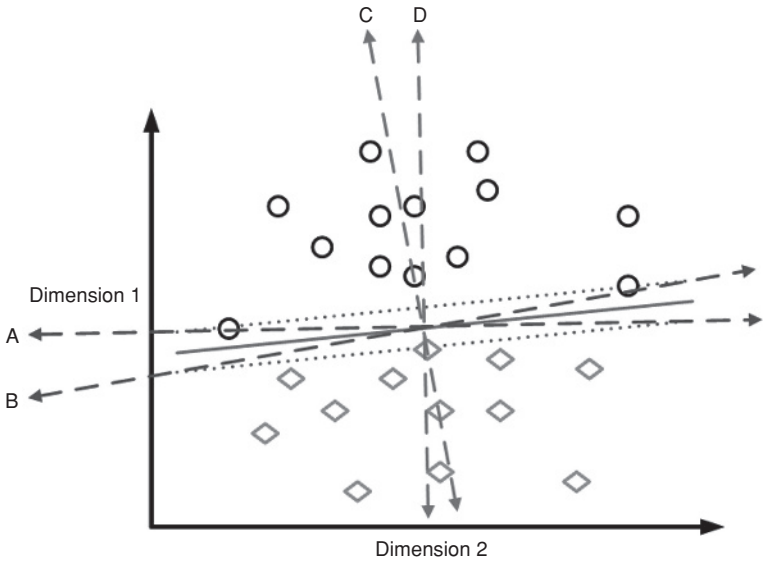

Figure 1.4 Support vector example from Figure 1.3 in which the Dimension 2 margins of the support vector (the "decision zone," or space within a margin of the optimal decision boundary) are used to define the range of slopes for the decision boundary. Line A indicates the minimum slope defined by the upper left and lower right limits of the decision zone, and line B indicates the maximum slope. The perpendiculars to lines A and B are lines D and C, respectively, and delimit the allowable range of slopes for the line of best fit to the overall data set

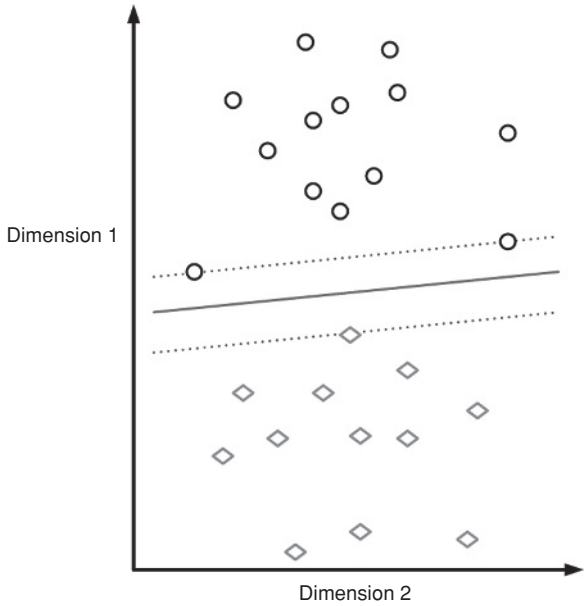

Figure 1.5 Support vector example from Figure 1.3 in which the data for the two classes (represented by circles and diamonds) are stretched vertically to ensure that the regression line of best fit and the principal component will both be roughly perpendicular to the decision boundary

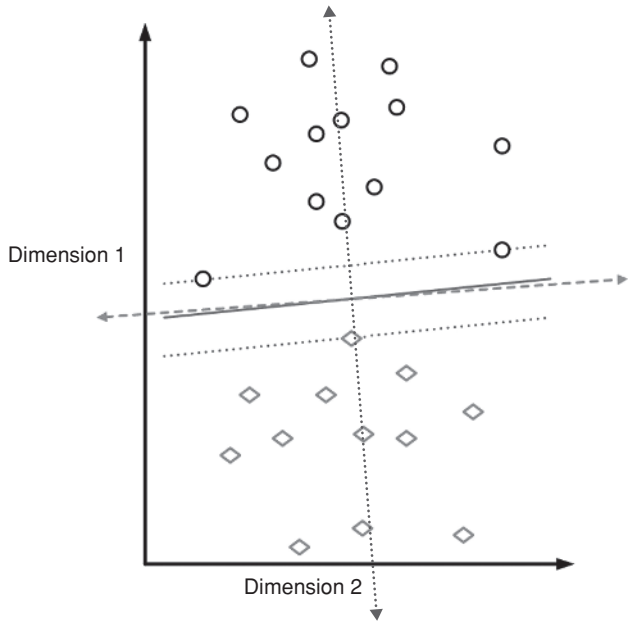

Figure 1.6 Redefinition of the decision boundary (dashed line, which passes through the centroid of the decision zone) based on defining it as perpendicular to the line of best fit to the transformed data ("dashed" and "dotted" line)

allows us to use regression and/or principal component analysis (PCA) to define the optimal decision boundary.

In this case, the decision boundary could be defined as the perpendicular to the line of best fit through the combined data. An illustration of this is given in Figure 1.6; the line of best fit is illustrated by the "dashed" and "dotted" line. This line of best fit is used to redefine the decision boundary as a perpendicular to it. In this example, the final system defined by the decision boundary is likely to be more robust than the original system because it minimizes the error of regression and yet is still compatible with the support vector decision zone.

Figures 1.3, 1.4, 1.5, and 1.6 provide some possibilities for modestly improving the support vector. However, it is more important to address less ideal systems in which the support vector cannot provide error-free classification. As noted in Figure 5.3 of the Marsland (2009) reference, one approach is to immediately assess each of the candidate margins for their errors: "If the classifier makes some errors, then the distance by which the points are over the border should be used to weight each error in order to decide how bad the classifier is." The equation for errors is itself turned into a quadratic programming problem, in order to simplify its computation. The error function to be minimized, however, incorporates an L1, or Manhattan, absolute difference, which may or may not be appropriate; for example, an L2 or Euclidean distance seems more appropriate based on the properties of least squares fitting described in Section 1.6.1.

The L2 distance is used in the example of Figures 1.7 and 1.8. In Figure 1.7, a problem in which the support vector decision boundary does not eliminate all classification errors is

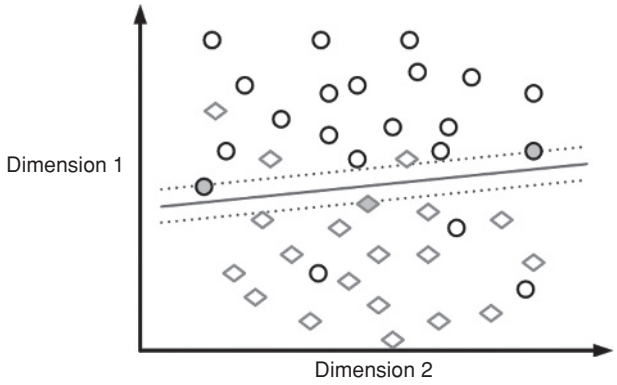

Figure 1.7 Decision boundary (solid line) with margin to either side (dotted lines) as defined by the support vector (filled circles and diamond). Here, there are six misclassifications

illustrated. In fact, 6 of 41 samples are misclassified. The support vector, however, provides a decision boundary and maximized margin.

Next, the entire set of data in Figure 1.7 is used in combination with the support vector decision zone to create a new decision boundary that has minimal squared error for the misclassified samples as a sum of their L2 values from the boundary. Figure 1.8 provides the optimum decision boundary that is compatible with the support vector range of allowable values as introduced in Figure 1.4. This can be compared with the decision boundary that minimizes the sum of L2 error distances overall in Figure 1.9.

Through this process, a boundary-based approach, the SVM, has been modified to be compatible with a population-based, "bottom-up," data-driven approach. Whether this approach provides a real advantage to SVM approaches compared to, for example, the kernel trick described next, remains to be seen. Certainly, it should be an area of research for the machine learning community. However, it does appear likely that such an approach, in minimizing the

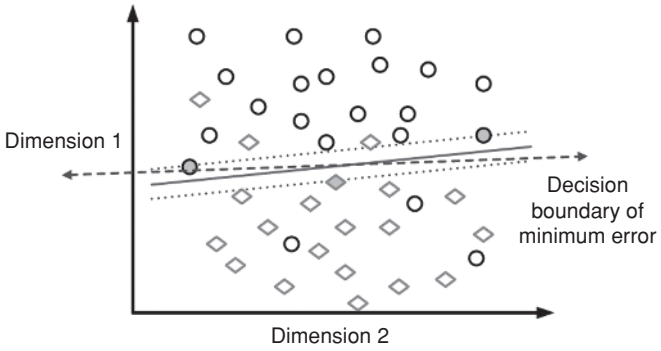

Figure 1.8 Decision boundary providing least squares error for the six misclassified samples of Figure 1.7 (dashed line), which does not deviate from the decision zone of the support vector

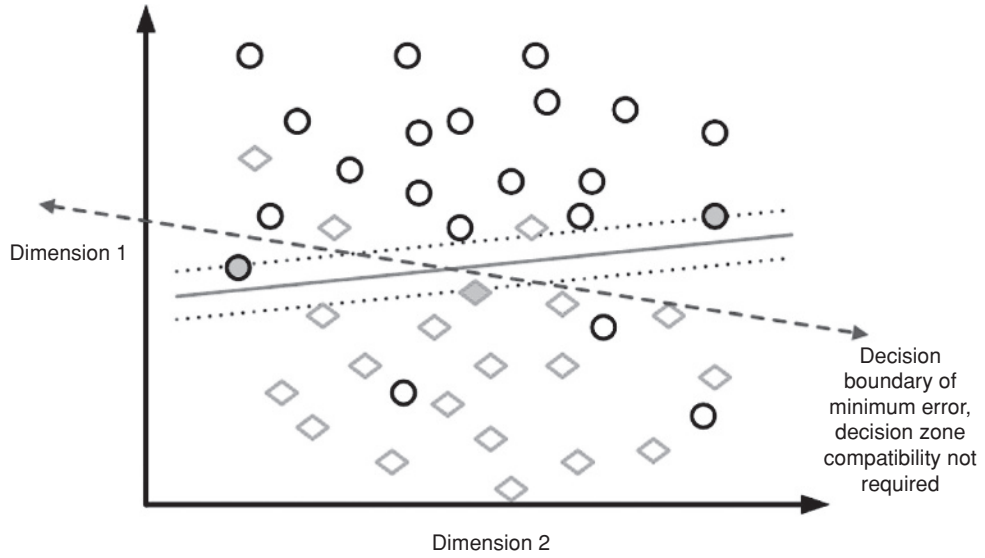

Figure 1.9 Decision boundary providing least squares error for the six misclassified samples of Figure 1.7 (dashed line), where it is allowed to deviated from the decision zone of the support vector

overall squared error and in maximizing the margin around the decision boundary, may be more robust.

Figures 1.7, 1.8, and 1.9 illustrate a coordinate space in which no linear separation of the class is possible. For linear separation, the following two equations must hold:

$$\text{If } (y > mx + b),\ y \in \text{Class}_A;$$
$$\text{If } (y < mx + b),\ y \in \text{Class}_B.$$

The kernel trick is introduced to find a linear decision boundary for separation of these two classes, since in real-world classification problems, the class separation is rarely as clean as shown in, for example, Figure 1.3. More commonly, the initial class separation, or decision boundary, is a Gerrymandered curve along the lines of that shown in Figure 1.10a. With the kernel trick we transform the decision boundary—albeit in a transformed coordinate system made possible by the increased dimensionality of the kernel—into a linear boundary as shown in Figure 1.10b.

Why is the transformation to a linear boundary important? My answer here is not based on SVM theory, as I am unaware of research to date that has addressed, let alone solved, this question. My interpretation is that, for one thing, it makes the boundary a minimal length pathway through the new dimension space, which minimizes the "error surface." For another, this single pathway boundary—when the right kernel is selected—minimizes overfitting, since the boundary is described by a surface the same order as the dimensionality of the class space.

In concluding this section on SVMs, I would like to point out their central role in current classification theory and practice. The kernel trick has, in some ways, allowed a classifier—the

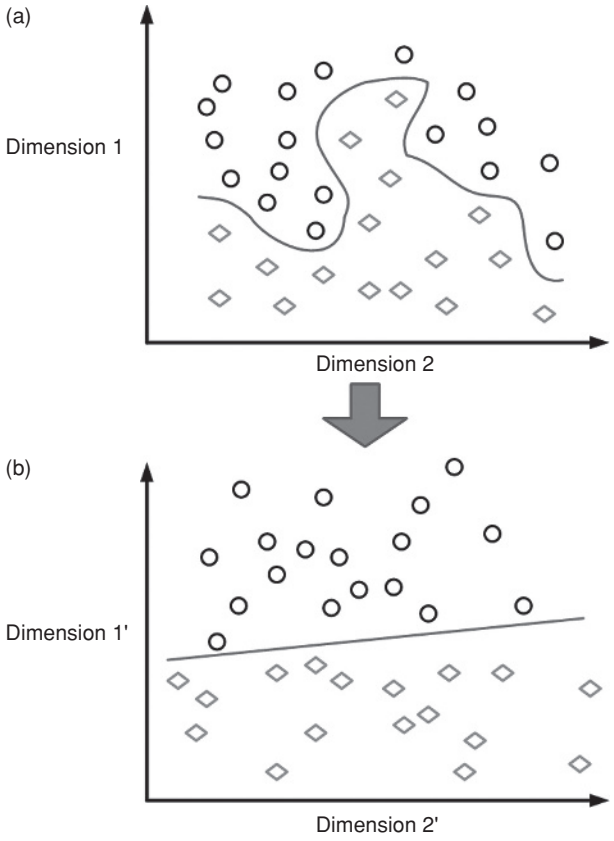

Figure 1.10 Decision boundary for two classes in the original 2D space (a) and in the transformed space (b) created within the kernel. In (b), the decision boundary is linear, which generally minimizes overfitting and differences between training and testing error rates

support vector—which might otherwise not be very effective for a given problem to become very effective when one or more kernels are utilized. Among these, determining the best results among the set of {linear, polynomial, radial basis, Fisher, Gaussian} kernels provide a good repertoire of choices that will generally result in one or more highly accurate SVM + kernel systems. For more on kernel engineering, please see Shawe-Taylor and Cristianini (2004).

1.6.3 Probability

Probability methods are important for a wide range of intelligent system tasks, including clustering, classification, and the determination of variance and covariance. Clustering is especially important to estimate the relative sizes and distribution of the classes of data, which in turn can be used to decide what, if any, transformation should be performed on each distribution before downstream analysis.

Probabilistic means of clustering data are as simple as the expectation maximization (EM)-based *k*-means clustering approach. The *k*-means clustering approach performs iterations consisting of two steps: updating the cluster assignment of the samples based on their location to the nearest cluster center, followed by updating the cluster centers based on the set of samples now belonging to them. Though simple, this algorithm can be readily modified to accept constraints, such as (1) having clusters with equal number of samples; (2) weighting different features used to determine the distances from each cluster center differently based on their predicted utility for later classification, regression, and so on; and (3) allowing predefined clusters to be accommodated concomitant to unknown clusters.

Another important set of approaches is based on joint distributions and conditional independence of features. In this case, the probability of an event A, given the occurrence of an event B, is the same as the probability of an event B, given the occurrence of an event A. The equation describing this is

$$p(A \mid B)^* p(B) = p(B \mid A)^* p(A),$$

from which the general form of Bayes' theorem is directly obtained:

$$p(A \mid B) = \frac{p(B \mid A)^* p(A)}{p(B)},$$

where, as noted above, $p(A \mid B)$ is the probability of event A given that the event B has already occurred. In many cases, we wish $p(A)$ and $p(B)$ to be very low (rarity of identifying event or events) but $p(A \mid B)$ to be very high (specificity of event or events). Rearranging Bayes' theorem to show the ratio of $p(A)$ and $p(B)$, we can see that the ratio of $p(A)/p(B)$ is equal to the ratio of $p(A \mid B)/(B \mid A)$:

$$\frac{p(A)}{p(B)} = \frac{p(A \mid B)}{p(B \mid A)}.$$

In many machine intelligence problems, we are concerned with events that occur with some probability when one or more of multiple other events occur. In fact, we may wish to find the probability of event A when considering all of a set of mutually exclusive events, denoted as B here. We thus rearrange to solve for $p(A)$:

$$p(A) = \frac{p(A \mid B)^* p(B)}{p(B \mid A)}.$$

Let us now consider the case where B can occur as one of N independent outcomes; that is, the sum of all $p(B_i) = 1.0$. We then obtain the following for $p(A)$:

$$p(A) = \frac{\sum_{i=1,\ldots,N} p(A \mid B_i)^* p(B_i)}{\sum_{i=1,\ldots,N} p(B_i \mid A)}.$$

This latter equation also governs the probability of event A for any subset of the outcomes of B; for example, under constrained situations in which one or more of the B_i is not allowed to, or cannot, occur. Why is this important? Because in many cases, we may not have a good estimate for the probability of event A. Event A, meanwhile, may be a triggering event for a specific downstream task, such as using a specific classifier tuned for the input data when the triggering event occurs. Thus, the rearranged generalized Bayesian equation allows us to compare different dependent events, here event A, for their overall probabilities against a mutually exclusive set of events, B.

1.6.4 Unsupervised Learning

Unsupervised learning techniques are concerned with clustering (aggregating like elements in a group), input space data distribution, or other operations that can be performed without training data. Training data is also referred to as ground-truthed data and as labeled data. With such unlabelled data, we have to rely on the structure of the data itself to infer patterns; that is, unsupervised learning.

One type of unsupervised learning is the k-means clustering approach described in Section 1.6.3. Related to the k-means clustering approach is the Gaussian mixture model (GMM), in which we know the number of classes that are in a data set, but have no labeled data. The GMM assumes that the data set is a function comprising the sum of multiple Gaussians, one each corresponding to the individual classes:

$$f(x) = \sum_{c=1}^{C} w_c G(x : \mu_c, \mathrm{X}_c),$$

where x is the set of features, $f(x)$ is the output based on the set of features, C is the number of classes, w_c is the weight assigned to class c, and $G(x : \mu_c, \boldsymbol{\chi}_c)$ is a C-order Gaussian function with mean μ_c and covariance matrix $\boldsymbol{\chi}_c$. The classifier consists, after the model is built (also using an EM approach, per Section 9.2.2 of Bishop (2006)), of finding the maximum of the following:

$$\max_k \left\{ p(x_a \in k : k = 1, \ldots, C) = \frac{w_k G(x_a : \mu_k, \mathrm{X}_k)}{\sum_{c=1}^{C} w_c G(x : \mu_c, \mathrm{X}_c)} \right\}.$$

Weighting the classes is governed by the constraint

$$\sum_{k=1}^{C} w_k = 1.0.$$

For the EM approach to determining the GMM, the weights of the individual classes are proportional to the number of samples assigned to the individual classes. This is called the *average responsibility approach*, since the weighting is proportional to the class' responsibility

in explaining the input data set. It should be noted that the individual features in the feature vector **x** can also be weighted differentially. There will be much more to say on this in later sections of the book.

Self-organizing feature maps (SOMs) (Kohonen, 1982) are an unsupervised learning approach to describe the topology of input data. This machine learning approach can be used to provide a lower-dimensional representation of the data in addition to its density estimation value. It should be noted that SOMs can be very useful at the front end of unsupervised clustering problems, as well. As will be discussed in detail in Chapter 8 and elsewhere in this book, one of the primary considerations in parallel system design is to decide how best to aggregate processes to run in parallel. For large intelligence systems, this may include deciding which aggregate classes to form first and from this determine the structure of a decision tree.

1.6.5 Dimensionality Reduction

SOMs, then, can be used for dimensionality reduction. Nearest neighbor techniques can also be used to reduce the dimensionality of a data set. This is important to classification problems too, since it is usually advantageous to eliminate isolated samples or even small clusters to avoid overtraining, the addition of ectopic classes, and other types of "output noise." Figure 1.11 illustrates how the 3-nearest neighbor approach is used to assign an unassigned sample to one of two classes. The process involved is the same as that used for assigning samples when the trained system is deployed.

The example of Figure 1.11 reduces the number of classifiers. More traditionally, however, *dimensionality reduction* is concerned with the removal of features from the feature space. In order to reduce the number of features, Marsland (2009) outlines three primary approaches:

1. Feature selection
2. Feature derivation
3. Clustering.

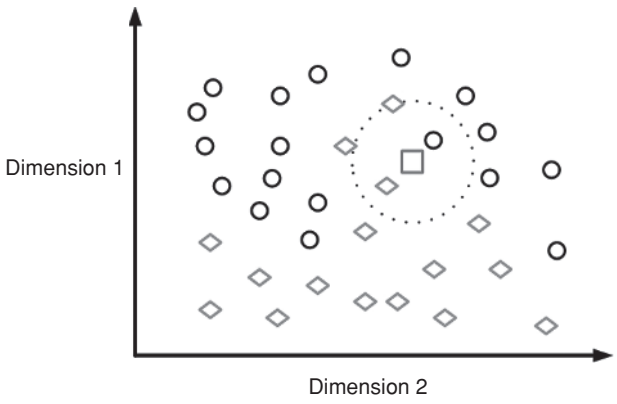

Figure 1.11 *k*-nearest neighbor classification applied to a formerly unclassified (square). The dotted circle has a radius sufficient to incorporate the nearest three neighbors. Based on the results, the square is assigned to the same class as the diamonds

Feature selection is a white box means of eliminating features inasmuch as we use our awareness of the relationship between a feature and the system output—for example, through correlation, confusion matrices, and so on—to decide whether or not to eliminate a feature. We can triage features through some relatively simple algorithms, including the following:

1. Select the feature with the lowest correlation with the output (alternatively, select the feature with the lowest accuracy output).
2. Remove the feature.
3. If system performance improves, drop the feature and return to Step 1.
4. If system performance does not improve, either terminate feature selection or jump to the next most poorly correlated (or accurate) feature and repeat Steps 1–3.

When employing Step 4 in this algorithm, we may choose to allow the "stack" to only get to a certain depth (say, 10% of the number of features) before terminating the search. Other forms of feature selection include the decision tree approaches and linear discriminant analysis (LDA). LDA is concerned with projecting to a lower-order dimension such that the samples in the higher-order dimension are optimally separated in the reduced dimension space. The approach is effectively half of an F-score approach: we wish to maximize the between-class scatter. The effect is similar to that discussed with regard to Figures 1.5 and 1.6. The "dashed" and "dotted" line in Figure 1.6, in fact, could be the output of an LDA, allowing the two dimensions in Figure 1.6 to be reduced to a single projected line.

Harkening back to the previous section, we observe that the LDA is performed on labeled data. However, there are several methods for dimensionality reduction that do not require labeled data. The first, PCA, is the process by which a matrix is transformed into its eigenvectors and eigenvalues; that is, orthogonal, uncorrelated components comprising a new set of dimensions in which the principal component is in the direction of maximum variance for the data set. Independent component analysis (ICA), on the other hand, assumes that the latent components are independent, and so strives to transform the input data space into a set of independent dimensions. Dimensionality reduction is readily achieved for PCA by dropping all of the eigenvectors whose eigenvalues are below a given threshold (Jolliffe, 1986), and for ICA based on the measurement of the mutual information of the components (Wang and Chang, 2006).

Feature derivation, a second method for dimensionality reduction, is concerned with transforming a (measured or original) feature set into another (derived or transformed) feature set. These new features can be combined features from the original space; for example, in imaging we may wish to combine red, green, and blue channel intensity into a *mean intensity* feature, which is simply the mathematical mean of the original three features. Note, however, that such transformations do not necessarily result in a reduced feature set immediately after transformation: in this example, we may replace the three features (red, green, and blue channel means) with three new ones (mean intensity, mean saturation, and mean hue) that provide us with a set better pruned using feature selection.

Both PCA and ICA, described above, perform feature derivation in addition to the feature selection. Each of them computes a principal component in the direction of maximum variance and derives the rest of the components based on orthogonality (PCA) or independence (ICA). Factor analysis (Dempster, Laird, and Rubin, 1977; Liu and Rubin, 1998) is another means of dimensionality reduction in which latent variables are discovered using an EM approach.

Multi-dimensional scaling (MDS) (Cox and Cox, 1994) and locally linear embedding (Roweis and Saul, 2000) are two other feature derivation processes.

The third method for dimensionality reduction, clustering, has been already addressed in light of *k*-means clustering, *k*-nearest neighbor (*k*-NN) clustering, and GMMs. As will be shown later in this chapter, a specific criterion, the F-score, can be used to decide whether or not to cluster.

Dimensionality reduction is important for several reasons. First off, in reducing the set of features on which regression, classification, and other machine learning tasks are based, it allows better compression of the data. Secondly, in reducing the correlation and/or dependency among the features, dimensionality reduction may make the system less sensitive to noise. Thirdly, dimensional reduction results in improved retrieval efficiency during querying.

1.6.6 Optimization and Search

Optimization is the process whereby, with a certain confidence, the best possible outcome for a system is obtained. Typically, a function of the output—for example, the cost of the system components, the cost of the processing, or the cost of correcting system errors—is to be optimized. For cost, the optimization is usually a minimization. Introductory calculus tells us that a smooth function can be effectively searched for its local optima using the gradient, or Jacobian operator. The steepest descent approach updates the vector $\mathbf{x}$ as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \Delta_k \nabla f(\mathbf{x}_k),$$

where Δ_k is the distance to travel in the direction from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ in order to reach a minimum. A related approach, which effectively uses the gradient descent method to create a trust region for a least squares approach, is the Levenberg–Marquardt algorithm (Levenberg, 1944). The conjugate gradient method is often a significant improvement on these methods, however, since it moves in conjugate directions sequentially—thereby avoiding consecutive small steps in the same direction.

For each of these approaches, one concern is that the gradient cannot move the vector $\mathbf{x}$ from the zone of a local optima. No method for searching other neighborhoods in the output space is provided. Thus, it is not unreasonable to combine such an optimization approach with a "location generator" that seeds an appropriate set of points in $\mathbf{x}$ to provide statistical confidence that an overall optimum has been achieved. This is illustrated in Figure 1.12, in which sufficient starting points exist to allow the finding of the overall optimum.

In later chapters of this book, first derivative approaches will largely focus on sensitivity analysis problems. However, it should be noted that the same concern applies to sensitivity analysis problems as do to the more general optimization problems. Namely, the input space must be seeded with a sufficient number of starting points to allow the overall optima to be found.

Another important machine intelligence system approach to optimization is search. The path to search is implicit in the grid pattern of seeded starting points in Figure 1.12. As the spacings between starting points, or nodes, in the grid become smaller, the odds of missing the overall optima also become smaller. Only optima with contours having effective radii of

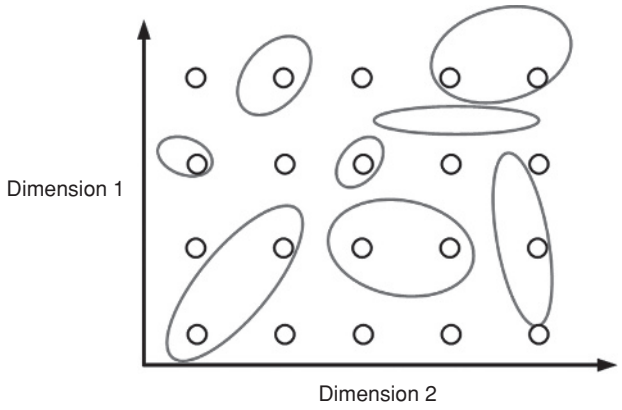

Figure 1.12 Example of an output space with sufficient seeding of starting points $\{x_j\}$. These are arranged in the grid (circles). The outer contours of regions for local optimization are shown in ovals. Half of the starting points belong to one of the local optima contours, and the average contour spans 1.25 of the starting points. It is highly unlikely that none of the starting points will reside inside the contour of the overall optimum

less than 70.7% of the internode distance could be unseeded. Assuming that the grid of seeded nodes is sufficient, therefore, the optima (or near-optima) can be found through an *exhaustive search* of the output values for all of the nodes. This search can be augmented by gradient methods as described above to find the optima around each of the nodes.

Another important use for search in machine intelligence is in optimizing pathways. Minimum pathways are important in network, traffic, and other multi-step processes. The benchmark for optimizing pathways, as for search, is exhaustive search, in which every possible pathway is attempted. Of course, this is hugely expensive computationally, even for a massive and massively parallel computing environment. The number of possible pathways, N_{PP}, for an N-node set of locations is given by

$$N_{\mathrm{PP}} = \frac{(N-1)!}{2}.$$

As N increases to more than a trivial problem space, the number of pathways becomes unwieldy for the exhaustive search approach. As a consequence, effective means for searching a subset of the possible pathways must be selected. In Simske and Matthews (2004), several methods for selecting the next node in a pathway were given. One was termed the "lowest remaining distance" strategy, which is also known as the greedy search approach (Marsland, 2009). This approach often results in excessive distances for the last few node–node transitions. Another approach was termed the "centroid + clockwise traversal" method, which rotates around the centroid of all the nodes to the nearest—by angular value—node from the current node. This method is readily extended to a hybrid clustering/clockwise traversal approach that effectively introduces supernodes that are internally sequential. A final method mentioned in Marsland (2009) is the hill climbing approach, in which pairs of nodes are randomly swapped

to see if improvements can be made over the best current set of pathways. This approach naturally leads to genetic algorithm approaches, which are described in Section 1.7.2.

1.7 Artificial Intelligence

In this overview chapter, artificial intelligence approaches are collectively viewed as machine reasoning systems in which the architecture of the connections is allowed to optimize without regard to a specific guiding formula or expert system rule set. Neural networks, which trade off architectural complexity with convergence speed, are overviewed first. Next, genetic algorithms are introduced and shown to be effective for searching a very large input space with a relatively small number of trials. We conclude this section with an overview of Markov methods, which are "memoryless" approaches in which the next state depends only on the current state.

1.7.1 Neural Networks

The term "neural network," or more correctly, "artificial neural network" (ANN), implies a system whose architecture is inspired by one or more physiological neural networks. While this may be the case in some instances, for the general case this is not an accurate representation of how ANNs are designed.

The human cerebral cortex, for example, is largely comprised six layers of neurons, with widely different relative layer thicknesses when comparing the different cortical lobes: frontal, parietal, temporal, and occipital (Kandel, Schwartz, and Jessell, 2000). This relatively structured architecture, however, is far different from that of the hippocampus—a brain structure associated with the incorporation of memory—or that of the cerebellum, a three-layered brain region containing more than half of the brain's neurons. The modal neuron in the brain makes thousands of synaptic connections to other neurons. These connections are characterized by the effects of released neurotransmitters on the postsynaptic neurons. These effects can be excitatory or inhibitory, synergistic or antagonistic, short-termed or long-termed, persistent or habituated, and even result in distinct effects on different postsynaptic neurons. Synaptic (and dendritic) projections can be local, into adjoining layers, or to distant brain regions. In other words, brain architecture is both complex and diverse.

Learning in the brain is also both complex and diverse. The efficacy of synapses, resulting in stronger or weaker influence on the postsynaptic neuron, can be altered through changes in neurotransmitter release, changes in neurotransmitter reuptake or removal from the synapse, changes in the morphology of the synapses, and changes in the number of synapses, depending on the strength and length of the reinforcement provided for a given connection. At a more macroscopic level, brain learning is also complex and diverse. Neural pathways are often carved rather than assembled—think of Michelangelo's *David* and not Picasso's *Baboon and Young*. This carving not only reinforces certain behaviors or preserves certain thoughts but also removes other behaviors and thoughts. This is in fact termed *neural sculpting*, and this too is not a uniform process throughout the brain; for example, it creates longer-lasting information pathways in the cerebral cortex than those in the hippocampus.

Let us compare these nature-made neural networks (NNN) to the artificial ones. For this comparison, the ANN of choice will be the multi-layer perceptron, or MLP, of Rumelhart,

Hinton, and McClelland (1986). This MLP connects an input layer to a hidden layer, and the hidden layer to the output layer, and comprises a three-layer network. The architecture of the MLP is far simpler than that of the NNNs described above in the following ways:

1. The ANN has a specific input and output layer. For an NNN, these would be functionally replaced by sensory (input) and motor (output) neurons, furthering the complexity of the NNN from that described above.
2. The ANN has only a single hidden layer that connects the input and output layers. The NNN, on the other hand, consists of three (cerebellum) or six (cerebral cortex) layers, or a much less structured, more highly interconnected architecture.
3. The number of connections in the MLP is very small. Typically, each input node, IN, connects to each hidden layer node, HLN, and each HLN in turn to each output node, ON. If the number of nodes in each of these three layers are N_{IN}, N_{HLN}, and N_{ON}, respectively, then the total number of connections, N_{C}, is

$$N_{\mathrm{C}} = N_{\mathrm{HLN}}(N_{\mathrm{IN}} + N_{\mathrm{ON}}),$$

for which the mean number of connections per node, $\mu_{\mathrm{C/N}}$ is

$$\mu_{\mathrm{C/N}} = \frac{N_{\mathrm{HLN}}(N_{\mathrm{IN}} + N_{\mathrm{ON}})}{N_{\mathrm{IN}} + N_{\mathrm{ON}} + N_{\mathrm{HLN}}}.$$

If the number of nodes at each of the three layers are equal, then

$$\mu_{\mathrm{C/N}} = 0.67N_{\mathrm{HLN}}.$$

If, however, N_{IN} and N_{ON} are $\gg N_{\mathrm{HLN}}$, as may be the case for imaging applications, then

$$\mu_{\mathrm{C/N}} \approx N_{\mathrm{HLN}}.$$

In other words, $\mu_{\mathrm{C/N}}$ is $O(N)$. Unless the number of hidden nodes is abnormally large—the range of hidden nodes typically used is from roughly 20 to 100—this is approximately two orders of magnitude less than the mean number of connections per node in the human brain. As a better direct comparison, though, we should compare the ANN to a ganglion, that is, a peripheral nerve center. For this NNN, the mean number of connections per node—I should say per neuron here—is intermediate to that of the MLP and the human brain regions described above.

Regardless, we must keep in mind that the relationship between the mean number of connections and the complexity of a neural network—not to mention its capacity for learning—is nonlinear. Assuming that all synapses are independent and contribute relatively equally to learning and neural processing, the relationship between $\mu_{\mathrm{C/N}}$ and ANN or NNN complexity is geometric.
4. The number of synapses is not equal to the number of connections in a living neural network, as it is in the ANN MLP. In an NNN, new synapses are grown during learning (Bailey and Kandel, 2008), allowing the same neuron not only to connect to new postsynaptic neurons but also to add new connections to existing postsynaptic connections.

5. Learning in an MLP consists of changing the weights on the connections between nodes. Learning in an NNN, as described above, is more complex. Here, learning includes the growth of new neurons, the growth of new synapses, and changes in the morphology and/or efficacy of synapses—along with the opposite of each of these in the case of neural sculpting.
6. Short-term and long-term memory also differ greatly when comparing the MLP and NNNs. In an MLP, the short-term learning is effectively the error feedback changing the weights of the connections. The long-term memory is the current state of the MLP, which is simply the set of weights on all of the connections. In an NNN, short-term and long-term memory are both quite complex. Synaptic plasticity, or the ability of chemical (neurotransmitter-based) synapses to change their efficacy, includes long-term potentiation (LTP) and increased synaptic interface surface area (ISISA). LTP is the process by which the amount of neurotransmitter released is altered—for example, greater amounts of excitatory neurotransmitter(s) are released into the synapse or lesser amounts of inhibitory neurotransmitter(s) are released into the synapse—or else the rate of removal of neurotransmitters from the synapse is altered. ISISA results in a greater relative postsynaptic potentials, enhancing the ability of the presynaptic neuron to affect the postsynaptic neuron. Both LTP and ISISA are associated with short-term learning, although ISISA can also be associated with long-term memory if the changes are significant—and permanent—enough. Of course, long-term memory in an NNN is even more complicated, since it includes the growth of new excitatory and/or inhibitory synapses, the projection of neuronal axons or dendrites into new regions, or even the growth of new neurons.

It is clear that ANNs such as the MLP are not as complex as the biological systems that (at least in theory) inspired them. It is certain that some ANN architecture will increase in complexity as their value for cognitive analysis is bettered, and better appreciated by the machine intelligence field. Even so, the general principles of the MLP are still quite likely to apply, and the various physiological attributes of NNNs are likely to be adopted slowly, and only in targeted applications.

The MLP is, in fact, simply an elaborate algorithm that is carried out on the structured sets of layers defined above. To initialize the MLP, an input vector is put into the INs. These nodes are arranged as a one-dimensional (1D) array, but 2D and 3D inputs are readily accommodated by simply mapping them to a 1D vector (e.g., the same way that 2D images are stored sequentially in memory, one raster line after the other). Weights along the connections are also randomly initiated, which contributes to the stochastic nature of ANN models. The input signals are fed forward through the network to the hidden layer and thence to the output layer. The state of the input signal, the weights along the connections, the sum of the weights at the next node, and the activation, or threshold, function (usually a sigmoid function) for determining whether the sum of weights causes firing of the hidden node, determine the set of states at the HLNs. The outputs of these nodes, the weights of the connections between HLNs and output layer nodes, and the same summing and activation approaches then determine the ON states.

Ground truthing or "target output" is then used to compute the error in the network. The expected, or targeted, output vector is compared to the actual output vector (the set of states of all ONs) and this error is then fed back into the network to update, in order, the hidden layer weights and then the input layer weights.

Each of these steps involves choosing certain system coefficients; for example, the activation function coefficient and the backward feedback coefficient. These coefficients are used to provide a trade-off between faster convergence time (preferred) and system convergence on a local optimum (not preferred). Increasing the learning rate (α) typically results in faster convergence, and also more likely results in the identification of a local optimum. Slower learning rates force the algorithm to inspect the topology at a higher resolution in exchange for slower convergence. This trade-off is, of course, problem dependent.

In this text, ANNs will be viewed as black boxes that produce a specific output, which we then use as input in a larger parallel processing intelligent system. For example, several neural networks can be run in parallel with different initial conditions to see if there is a preferred system convergence state or simply to provide multiple optima candidates. In this way, ANNs can be used in a method somewhat analogous to genetic algorithms, as described in the Section 1.7.2. Initial conditions based on an expert, or expert system, estimate may also converge faster and still provide an output that is optimal or very close to optimal. Regardless, ANNs are a very important type of algorithm to use in combination with other machine intelligence algorithms because they tend to work differently than many of the other common machine learning approaches, such as SVMs, boosting, genetic approaches, expert systems, and Bayesian systems. As such, they are a useful (and easy to implement) means of adding another intelligent system to a problem space. The advantages of this for the meta-algorithmic patterns outlined and elaborated in Chapters 6–9 will become obvious. The fact that there are a plethora of open source neural network software libraries is icing on the cake.

Note that in some ways, if I may reach a bit here, an ANN is analogous to the kernel method introduced in Section 1.6.2. The ANN—through the use of the hidden layer and a focus on optimizing connections between layers that are difficult, if not impossible, to map to any mathematical relationship between input and output—is able to solve problems without the solution architect having to really understand how the solution has been effected. In effect, a larger-dimensional solution space is created (the number of connections are much greater than the number of INs and ONs), which the MLP algorithm efficiently searches to create an intelligent system. The kernel trick also increases the dimensionality of the search space and in so doing allows a linear boundary to be found in that space that maps to a more complex—and more accurate—boundary in the lower-dimensional problem space.

1.7.2 Genetic Algorithms

Genetic algorithms (Goldberg, 1989) are efficient search approaches based on the principles and kinetics of natural selection and heredity. Specifically, genetic algorithms are based on the principle of a gene, which comprises a string of DNA or RNA triplets that code for a polypeptide (protein) chain or an RNA chain. For genetic algorithms, these triplets are represented with individual bits or loci, meaning that the sequences that can be optimized by the genetic algorithm can be binary in nature.

Any binary representation of a function to be searched can be handled by a genetic algorithm. In its simplest form, a genetic algorithm consists of the following elements and operations:

1. A set of strings that comprise a small subset of the possible search strings.
2. A means to populate this subset of search strings.
3. A fitness measurement for the search strings.

4. A means to select a next iteration of search strings, which includes replication, near-replication, crossover, mutation, and random selection.
5. A means of terminating the algorithm after a certain number of iterations of Steps 2–4.

These are relatively simple steps, and are well designed for searching over a specific output function.

As an example, suppose we wish to determine the best set of 10 query terms to include in a particular search. There are $2^{10} - 1 = 1023$ possible query sets (the null term query is disallowed), and we wish to get an acceptable recommended search query from just a small set of 50 search query candidates, called genes. We decided to use 10 randomly created genes, and then proceed from there with four rounds of gene shuffling. The initial set of 10 genes was determined by using a random number generator (RNG), with returned RNG values in the range [0.5, 1.0] being made 1s, and RNG values in the range [0.0, 0.5) being made 0s: example genes include {0001001110}, {1011011011}, and {0100001101}. Next, the fitness measurement was determined based on the relative success of finding the correct document using the search query based on each gene. The three genes shown have four, seven, and four terms, respectively (since the 1s indicate including the term). The fitness of the first and second queries (based on the genes) are weighted much higher than the third, and so these two are chosen—whereas the third is not—as the basis for the next set of 10 queries. A mutation rate of 5% and crossover rate of 90% is chosen. The first sequence {0001001110} is randomly assigned a crossover at the fourth bit with the second sequence {1011011011}, resulting in two new genes {0001011011} and {1011001110}. Next, one of these bits is mutated: the eighth bit of the second new gene, making it {1011001010}. Similar crossover and mutations led to an additional eight set of genes, for a total of 10 new genes in the second iteration. By the end of five such iterations, we have exhausted the 50 search query candidates, and the best quality search is, for example, 98.5% as good as the best single search out of all 1023 possible searches. Randomly selecting 50 queries would generally give a lower mean percentage of the best possible value, since the genetic algorithm preferentially reproduces the higher fitness genes, crossovers of two of them, and minor mutations on them.

Importantly, this set of steps, with only minor modification, can be used for other types of search, optimization, and classification. In a previous work, a genetic algorithm was used for a variant of the traveling salesman problem (TSP), in which a path through N locations is to be minimized (Simske and Matthews, 2004). In this case, the five elements and operations described above are implemented as follows:

1. A string of loci, numbered consecutively, each of which represents one particular location. For example, if L locations need to be passed through, then the locations may be listed alphabetically and assigned the values 1, 2, . . ., L.
2. Five different means of creating initial strings were used. Each of them involved the concept of next-node probabilities. These probabilities are an array of probabilities, normalized to sum to 1.0, which combined give the statistical likelihood of selecting each of the remaining nodes. These probabilities are recomputed each time a node is selected (using an RNG that is mapped into the probabilities). (a) The first methods used include naïve, in which all of the next-node probabilities are identical,

$$p_{i \in S(\text{unassigned})} = \frac{1}{N_{i \in S(\text{unassigned})}}.$$

The probabilities are equal for every unassigned node in the particular path being initialized. The set of unassigned nodes are in the set S(unassigned). (b) The second method assigns the next-node probabilities by relative distance,

$$p_{i\in S(\text{unassigned})} = \frac{D(\text{node}_i - \text{node}_{\text{current}})}{\sum_{k\in S(\text{unassigned})} D(\text{node}_k - \text{node}_{\text{current}})},$$

where $D(^*)$ is the Euclidean distance. (c) The third method assigns the next-node probabilities by cluster. Effectively, this reduces the problem space to a search within a cluster until all of the locations in a cluster are visited, and then the next cluster is appended. This is in effect a hybrid approach, and depends on an appropriate clustering algorithm—such as k-means clustering—to be performed first. The constraint can be relaxed such that the weighting within the clusters is simply relatively higher than the weighting outside of the clusters in comparison to what would otherwise be assigned by methods (a), (b), (d), or (e). Speaking of which, method (d) creates the next-node probabilities based on the direction of travel. If the current direction of the travel is clockwise (or counterclockwise) around the centroid of all the locations, then the next nodes in the clockwise (or counterclockwise) direction being traversed can be weighted more highly relative to its otherwise-assigned weight. The final method (e) weights the next-node probability based on the inverse of the distance; that is,

$$p_{i\in S(\text{unassigned})} \propto \frac{1}{D(\text{node}_i - \text{node}_{\text{current}})}.$$

Taking into account all remaining unassigned nodes,

$$p_{i\in S(\text{unassigned})} = \frac{\sum_{k\in S(\text{unassigned})} \frac{1}{D(\text{node}_k - \text{node}_{\text{current}})}}{D(\text{node}_i - \text{node}_{\text{current}})}.$$

3. The fitness measurement for each path is trivial to compute: it is the sum of the distances from the starting node through all the other nodes and back to the starting node. The results were not particularly surprising. As reported in Simske and Matthews (2004), when applying the inverse distance approach to both initialization and crossover (discussed below), excellent fitness and rapid convergence were observed.
4. The means to select a next iteration of search strings involved crossover, which involves simple reversal of a length of nodes: for example, if nine cities are visited in order from {4,2,5,8,3,9,1,7,6} and crossover is dictated to occur from nodes 3 to 7, then the order of cities visited becomes {4,2,1,9,3,8,5,7,6}. This effectively swaps part of one gene with a previous version of itself, and leads to generally much more change than a point mutation. Mutation in such a situation includes substitution, which for the example results in the swapping of nodes 3 and 7, producing the order {4,2,1,8,3,9,5,7,6}.
5. The algorithm described in points 2–4 is terminated after a certain number of iterations in which no overall improvement in fitness is observed.

The results of applying these steps to the TSP were immediate. In particular, the inverse distance approach resulted in much faster convergence and much less residual error after

convergence than the other methods. However, in the earlier work (Simske and Matthews, 2004), I did not have the opportunity to experiment on all five of the different means of creating initial strings. With the type of crossover used, it turns out that method (b) sometimes works, especially with asymmetric systems, since it helps create genes that are exploring new local optima. This overcomes one of the most common problems in genetic algorithms; namely, the difficulty of creating new searches that escape from the current local optimum. The mutation type implemented also helps prevent convergence on what is only a local optimum. Also, for both symmetric and asymmetric TSPs, the clustering method (c) works quite well when the clustering is attached to an F-score metric such as described later in this chapter.

The TSP solutions investigated here are a simplification of the real complexities involved in directing traffic. In real-world applications, the node–node transition costs are based on, among other things, both the distance and the traffic congestion between nodes. Other pragmatic considerations include the intelligent clustering of nodes based on similarity in the navigator's intents at each destination (e.g., cities to visit may be clustered by language in Europe), maximum distance preferred between nodes (so the traveler will not have too much driving for a day), and other pragmatic considerations.

Like neural networks described in the previous section, genetic algorithms are used to search a very wide space in an efficient manner. Genetic algorithms are not expected to find the global optima efficiently, but are expected to find something close to the overall optimum rather quickly. Thus, certain genetic algorithm approaches can provide the types of machine intelligence plasticity proffered by ANNs. One easy way to see a relationship is to consider how the initial conditions on the connections in a neural network are set. Suppose we choose to create a population of MLPs as introduced in Section 1.7.2. We may then initialize the MLPs using an RNG and score each MLP after just a small number of iterations using a fitness metric. The genetic "crossover" can then be used on the initial weightings to initialize a new MLP that can be deployed next, and this will presumably lead to a more optimal MLP more quickly than running a single MLP for many iterations.

Genetic algorithms have long been a favorite adaptive stochastic optimization algorithm, with the analogies to biological evolution often touted for robustness and convergence advantages. Another evolutionary analogy has often been overlooked: that of punctuated equilibrium (Gould and Eldredge, 1977). Punctuated equilibria comprise static periods in the evolution of a sequence of species that are interrupted by relatively short periods of rapid change. Genetic algorithms run the risk of remaining in stasis unless there is an impetus sufficient to effect change. As illustrated herein, this impetus can stem from changes in initialization, crossover, and/or mutations. It can also, importantly, stem from hybridization of a genetic algorithmic approach with that of clustering or other techniques.

1.7.3 Markov Models

Markov models are an important form of reinforcement learning (Marsland, 2009). This type of learning is based on knowing the right answer only, but not the correct means to arrive at it. This is analogous to many genetic algorithmic and ANN approaches, as described in the previous two sections. Markov models, however, have a very simple architecture. At each of the possible states in a Markov process, the conditional probability distribution for all possible next states is only dependent on the current state.

If the system state is fully observable, a Markov chain is used. If the system state is only partially observable, then a hidden Markov model (HMM) is used. The general Markov model is described by a joint probability model in which the probability of a sequence of S events is based on the conditional probabilities leading to each of the states $\{o_1, o_2, \ldots, o_S\}$. Thus, the probability of obtaining S specific outputs in this order is given by

$$p(o_1, o_2, \ldots, o_S) = \prod_{k=2}^{S} p(o_k \mid o_1, o_2, \ldots, o_{k-1}).$$

In most cases, the output set is too large, the relevancy of the many-termed conditional distributions is too low, and/or the training set is too small for all of these probabilities to be calculated. In most applications, a first-, second-, or third-order Markov model suffices. For a first-order model, the probability for the state sequence $\{o_1, o_2, \ldots, o_S\}$ is, therefore,

$$p(o_1, o_2, \ldots, o_S) = p(o_1) \prod_{k=2}^{S} p(o_k \mid o_{k-1}).$$

A first-order Markov model can be applied readily to classifier output. Examples of different states include characters in text recognition problems, phonemes in speech recognition problems, and different classes in any general classification problem. As an illustration, in Figure 1.13 the transition diagram associated with a three-state system is given. The conditional probabilities are first-order; that is, the probability of the next state is conditional only on the current state.

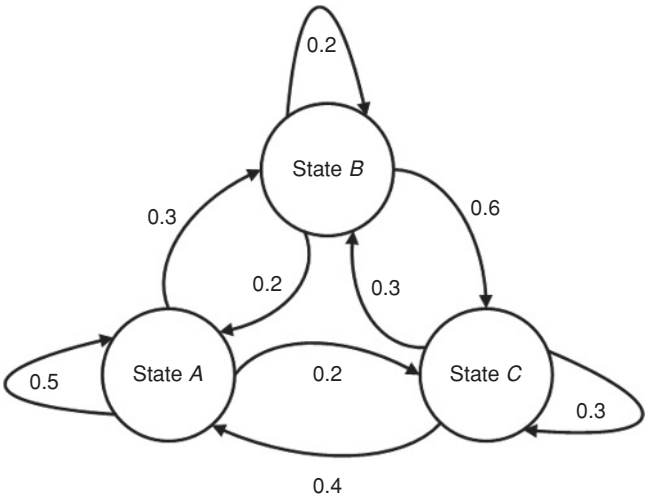

Figure 1.13 Transition diagram for a three-state Markov model. The model is first-order as all of the state transition probabilities depend only on the current state. The associated probabilities are indicated by the numerical values associated with the arrowed arcs. These probabilities are used to create the transition matrix, **T**, as described in the text

Extracting the transition probabilities from Figure 1.13, we obtain the transition matrix, **T**, given here:

$$\begin{array}{cc} & \begin{array}{ccc} \text{State } n & & \\ A & B & C \end{array} \\ \begin{array}{c} \text{State} \\ n-1 \\ \\ \end{array} \begin{array}{c} A \\ B \\ C \end{array} & \left[\begin{array}{ccc} 0.5 & 0.3 & 0.2 \\ 0.2 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.3 \end{array}\right] \end{array}.$$

This is a very convenient matrix to use for state probabilities. For the probability equation above, suppose we wish to compute probability of starting at state A and three transitions later ending at state C. Then

$$p(A, X_1, X_2, C) = \sum_{X_1 \in \{A,B,C\}} \sum_{X_2 \in \{A,B,C\}} [p(C \mid X_2) p(X_2 \mid X_1) p(X_1 \mid A)].$$

The first of these is $p(C \mid A)p(A \mid A)p(A \mid A)$, or $(0.2)(0.5)(0.5) = 0.050$. Summing all nine of these probabilities, we obtain

$$\begin{aligned} p(A, X_1, X_2, C) = {} & 0.050 + 0.090 + 0.030 + 0.012 + 0.036 \\ & + 0.054 + 0.016 + 0.036 + 0.018 = 0.342. \end{aligned}$$

Similarly, we can obtain the following probabilities:

$$\begin{aligned} p(B, X_1, X_2, C) &= 0.346, \\ p(C, X_1, X_2, C) &= 0.343. \end{aligned}$$

Perhaps not surprisingly, these values are all very similar, meaning that the state three cycles before is more or less unrecoverable for such a first-order Markov process. One would, based on these results, conclude that the system is in state C roughly 34.4% of the time.

I wrote software to perform millions of iterations on the states in this system, using an RNG to decide the state transitions. In fact, after 3×10^7 iterations, I found that the system spent 38.4% of its time in state A, 27.3% of its time in state B, and 34.3% of its time in state C. This result makes sense when we sum the columns in the transition matrix as shown:

$$\begin{array}{cc} & \begin{array}{ccc} \text{State } n & & \\ A & B & C \end{array} \\ \begin{array}{c} \text{State} \\ n-1 \\ \\ \\ \end{array} \begin{array}{c} A \\ B \\ C \\ \hline \end{array} & \left[\begin{array}{ccc} 0.5 & 0.3 & 0.2 \\ 0.2 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.3 \\ \hline 1.1 & 0.8 & 1.1 \end{array}\right] \end{array}.$$

Based on these sums alone, we expect the system to spend 36.3% of its time in state A, 26.3% of its time in state B, and 36.3% of its time in state C. Further eyeballing, we can see

that state *A* is more likely to stay in state *A* (50%) compared to state *C* (30%), and thus the overall 38.4–34.3% results observed, favoring state *A* over state *C*, also make sense.

The state sequences described herein are often captured in a lattice or trellis diagram. These diagrams are especially helpful, in my experience, for understanding the optimal pathways, such as the max-sum optimum as identified by the Viterbi algorithm (Viterbi, 1967).

Markov models with large output sets, farsighted conditional distributions, and insufficient training sets are unlikely to have reliable conditional probability values to use. The same caution should be noted for Bayesian networks, introduced in Section 1.6.3.

1.8 Data Mining/Knowledge Discovery

Data mining, often referred to as "knowledge discovery," brings together some of the algorithms and system approaches outlined in earlier parts of this chapter; in particular, probability, statistical NLP, and artificial intelligence. This is an important field to introduce here since it is so highly dependent not just on the algorithms used, but on how the algorithms interact with the hardware used to house the data. Data mining is tightly coupled to the type of data to be analyzed as well as the database upon which it performs search and analysis. For example, column-oriented storage organization significantly accelerates sequential record access within a database. However, this approach does not help for so-called object-oriented database (OODB) operations: these include commonplace transactional operations including refresh/update, delete, insert, and single-record search and retrieval. For OODB-like data mining, linked lists and other modern addressing and indexing approaches are more relevant.

In the end, data mining is the extraction of data from a data store (usually a database) for the purpose of upgrading its content: this can mean the creation of meta-data, the creation of new data structures, or raw analytics (data statistics). After this new data is created, it too should be represented in such a way as to optimize its downstream search and data mining on the given database, using the given data management system, and using the salient access protocols and structures. Thus, an important aspect of data mining is not just the updated content, but how the updated content will be represented and accessed thereafter.

Whereas genetic algorithms were shown to be associated with search—finding existing information from a large set of data—data mining is associated with discovery; that is, the finding of new content. Data mining is therefore analysis of large data sets in order to discover one or more of the following:

1. Patterns within and between large or multiple data sets. This *analysis of content* is designed to uncover relationships within data sets, including temporal relationships, occurrence frequencies, and other statistics of interest. The patterns can be based on model fitting (templated data discovery) or can be open-ended (statistical data discovery).
2. Unusual data, data sets, or data structures. This *analysis of abnormal content* is termed anomaly detection, and can include analysis of context. Data outside the range normally associated with a particular event will be tagged as anomalous, and will trigger an associated response (notification, corrective response, etc.).
3. Data dependencies, wherein terms co-occurring are identified and used to define association rules. These rules are used extensively in suggestive selling, advertising, promotional

pricing, and web usage mining. Combined, these rules are used to profile a person, group, or other user pool, and comprise *analysis of context* and/or *analysis of usage*.

Data mining is usually not the end of the analysis of the data. In many case, the analysis of content, context, and/or usage is the input to secondary, more specific analysis. For example, the data dependencies can be used to comprise a predictive model for later behavior. This predictive model may be used in synchrony with other input—for example, business policies, operational rules, personal knowledge, and business intelligence—to help define a decision support system that is used in the appropriate business context to more effectively solve problems and make better-informed decisions.

Knowledge extraction (KE), associated with data mining, is concerned with the repurposing of the data extracted for use in downstream processes. As such, KE is concerned with mapping mined data to a particular schema or structure that can be used in other contexts. Both structured and unstructured data are remapped into an ontology or taxonomy that can be more effectively incorporated into other decisioning processes. A key point here is that data mining and KE are more effective when they are part of a bigger, hybrid system comprising two or more machine intelligence algorithms, systems, and engines.

1.9 Classification

In many ways, classification is the central subject in intelligent systems, including the most sophisticated one: the human brain. Classification is the systematic placement of objects—concrete or abstract—into categories. Classification can be performed on structured data (in which the categories are known beforehand) or on unstructured data that has previously undergone cluster analysis (in which case *clustering* is performed first and later these clusters are associated with categories or otherwise assigned/combined).

SVMs are powerful classifiers, and they are overviewed in Section 1.6.2. SVMs are boundary-based approaches, which by definition focus on the *support vector*, or the set of samples that define the border zone between multiple classes. An example of a boundary-based classifier is given in Figures 1.14, 1.15, 1.16, and 1.17. In Figure 1.14, the 2D data is shown, the two dimensions are the two features measured for each sample. There is no clear linear boundary between the data.

Figure 1.15 shows the results of applying a boundary-based classification on the data in Figure 1.14. Here the boundary is loosely fit (e.g., a cubic spline fit).

Figure 1.16 shows a tightly fit decision boundary (TFDB) for the same data set. Here the boundary is able to avoid misclassification of any of the training data. This can be achieved by several of the methods described earlier in the chapter, including the SVM with the kernel trick and ANNs.

Figure 1.17 illustrates the samples that are part of the support vector for the boundary in Figure 1.16. More than half of the samples in the set are used for the definition of the vector space. As a consequence, the decision boundary in Figure 1.17 is likely fit too closely with the training data. The decision boundary created in Figure 1.16 will likely perform as well or better on test data.

Other classification approaches that are focused on decision boundaries are linear classifiers—which distinguish classes based on linear combination of the features (Simske,

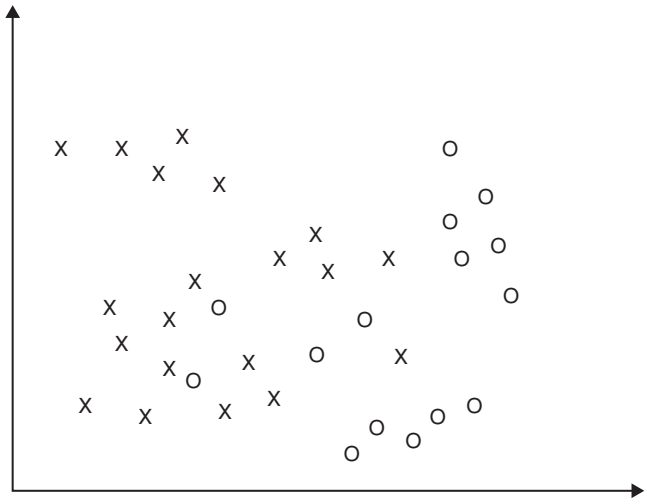

Figure 1.14 Simple 2D sample set ($n_X = 20$, $n_O = 15$) to be classified

Li, and Aronoff, 2005)—and quadratic classifiers, which separate classes based on a quadric surface.

All classifiers benefit from the availability of training data. For example, in Figures 1.14, 1.15, 1.16, and 1.17, if the amount of training data were greatly increased, we would have a better means of comparing the deployment accuracy of the two decision boundaries, shown in Figures 1.15 and 1.16. Figure 1.18 provides such a comparison through the addition of 55 more samples, which are considered a *validation set*. After adding the validation set data samples, the loosely fit decision boundary (LFDB) of Figure 1.15 has 87.8% accuracy on

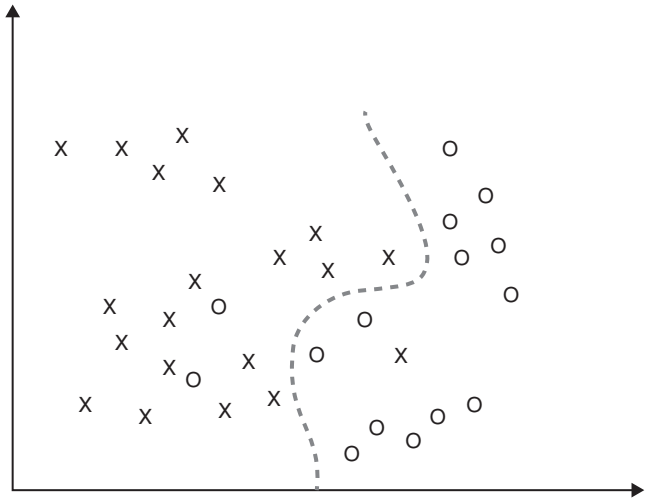

Figure 1.15 Data sample set of Figure 1.14 with a loose decision boundary

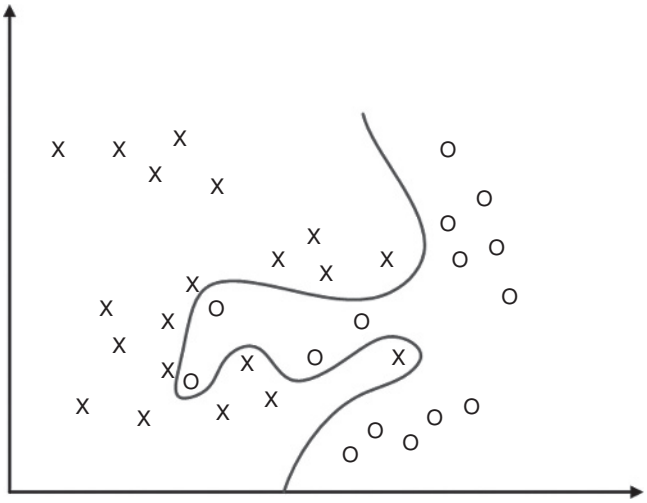

Figure 1.16 Data sample of Figure 1.14 with a tightly fit (overfit?) decision boundary

the combined training + validation data, slightly but statistically nonsignificantly greater than the 86.7% accuracy on the combined training + validation data observed for the TFDB of Figure 1.16. This is in spite of 100.0% accuracy when using the TFDB on the training set compared to the lower 91.4% accuracy when using the LFDB. Thus, the LFDB performed far better (85.5% accuracy) than the TFDB (78.2% accuracy) on the validation data, and is therefore the boundary of choice for deployment.

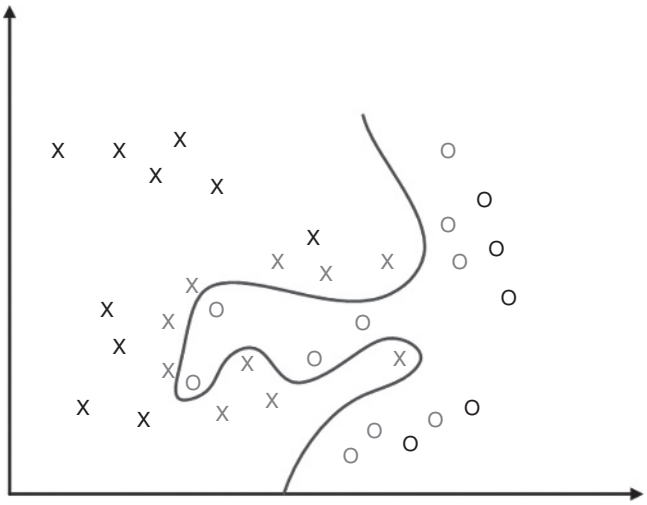

Figure 1.17 Data sample set of Figure 1.14 with the TFDB of Figure 1.16, with the support vector-relevant samples in light gray. In general, overfit decision boundaries will involve a higher than expected number of samples in the support vector

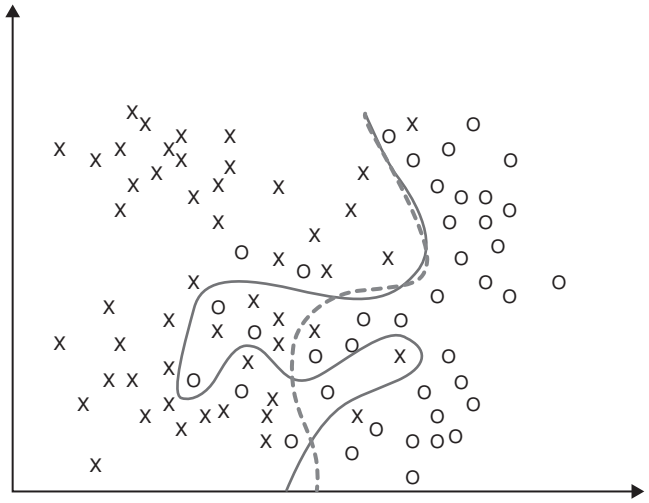

Figure 1.18 Data sample set of Figure 1.14 augmented with 30 more X and 25 more O samples ($n_X = 50$, $n_O = 40$). Here it becomes clear that the decision boundary of Figure 1.15 is most likely as good an approximation of the optimal decision boundary as is that of Figure 1.16. See text for details

Training data is the basis for the weightings used in boosting and the probabilities used in Bayesian classification and HMM classification. Both Bayesian- and HMM-based classifiers select as the assigned class the output with the highest summed probability.

Boosting classifiers such as AdaBoost (Freund and Schapire, 1996; Schapire, 1999) are ensemble classifiers that effectively used the consensus decisions of a (usually large) set of relatively inaccurate classifiers to provide a highly accurate decision. The random forest classifier (Breiman, 2001) is also an ensemble classifier, and it consists of many decision trees. After a large number of decision trees are formed, the most common output is accepted as the classification.

Clustering approaches are also used to create "bottom-up" classifiers; that is, classifiers that are based on aggregating rather than defining boundaries. Figure 1.19 shows a GMM clustering for the data of Figure 1.14, where five clusters are used to represent the "X" class, and four clusters are used to represent the "O" class.

The k-NN classifier is also an aggregation-based classification method. This classifier compares the test sample to the k nearest training samples and assigns the class as the modal neighbor class. Typically, k is in the range $\{1, \ldots, 5\}$, and is generally higher when the number of training samples is greater.

A helpful analytical measurement for clustering and for aggregation-based ("bottom-up") classifiers such as GMMs or simple linear combinations of Gaussian classifiers (Simske, Li, and Aronoff, 2005) is the F-score, which I use as the shorthand for the test statistic of an F-test such as is used in analysis of variance (ANOVA) and other statistical comparisons. The F-score is defined as

$$F = \frac{\text{MSE}_b}{\text{MSE}_w},$$

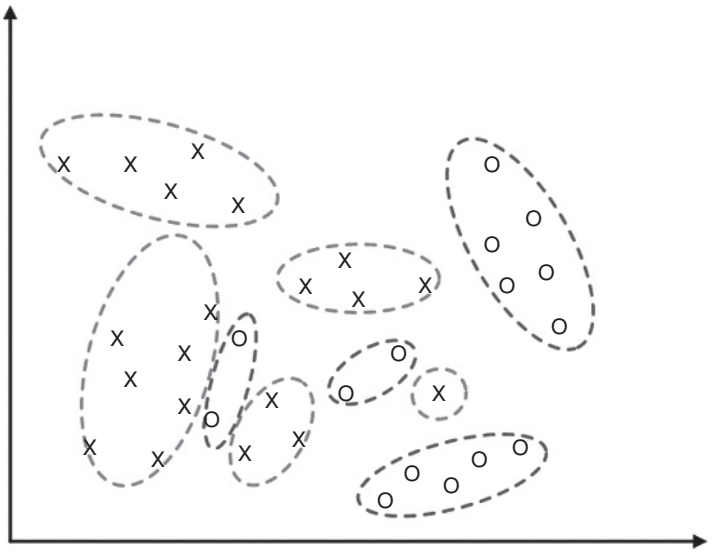

Figure 1.19 Data sample set of Figure 1.14 using a Gaussian mixture model for the classes. Here the number of Gaussians in the class are overfit, five for the X samples and four for the O samples

where MSE_b is the mean-squared error between the clusters and MSE_w is the mean-squared error within the clusters. Since the mean-squared errors are defined by the sum squared errors (SSE) and the degrees of freedom in the data sets, we rewrite the F-score as

$$F = \frac{\text{SSE}_b/df_b}{\text{SSE}_w/df_w},$$

where df are the degrees of freedom, generally 1 less than the members in a group. Next, we consider the values, V, of the data set. The mean value of cluster c, designated μ_c, is given by

$$\mu_c = \frac{\sum_{s=1}^{n(c)} V_{s,c}}{n(c)}.$$

Here, $V_{s,c}$ is sample s in cluster c. The number of samples in cluster c is $n(c)$ and the total number of clusters is n_c. From this, the value of MSE_w is given by the following:

$$\text{MSE}_w = \frac{\sum_{c=1}^{n_c}\sum_{s=1}^{n(c)} \left(V_{s,c} - \mu_c\right)^2}{\sum_{c=1}^{n_c} n(c) - n_c}.$$

The value of MSE_b is readily derived as follows:

$$\text{MSE}_b = \frac{\sum_{i=1}^{n_c}\sum_{j=i+1}^{n_c}\left(\mu_i - \mu_j\right)^2}{n_c(n_c-1)/2}.$$

Using the fact that

$$\sum_{s=1}^{n(c)}\left(V_{s,c}-\mu_c\right)^2 = \sum_{s=1}^{n(c)}\left(V_{s,c}^2 - 2V_{s,c}\mu_c + \mu_c^2\right) = \sum_{s=1}^{n(c)}V_{s,c}^2 - 2\mu_c^2 n(c) + \sum_{s=1}^{n(c)}\mu_c^2$$
$$= \sum_{s=1}^{n(c)}V_{s,c}^2 - 2\mu_c^2 n(c) + n(c)\mu_c^2,$$

the value of MSE_b is more conveniently written as

$$\text{MSE}_b = \frac{\sum_{c=1}^{n_c}\left(\mu_c - \mu_\mu\right)^2}{n_c - 1}.$$

Here, μ_μ is the mean of the means, which is the mean of all the samples if all of the clusters have the same number of samples, but usually not so if they have different numbers of samples. In terms of reducing computational overhead, then, MSE_b is

$$\text{MSE}_b = \frac{\sum_{c=1}^{n_c}\mu_c^2 - n_c\mu_\mu}{n_c - 1}.$$

Similarly, the computations for MSE_w can be reduced to

$$\text{MSE}_w = \frac{\sum_{c=1}^{n_c}\sum_{s=1}^{n(c)}V_{s,c}^2 - \sum_{c=1}^{n_c}n(c)\mu_c^2}{\sum_{c=1}^{n_c}n(c) - n_c}.$$

We now consider how the ratio of $\text{MSE}_b/\text{MSE}_w$—that is, the F-score—can be used to optimize clustering using a simple example. Consider the simple data set in Figure 1.20, which have two possible clusterings. The dashed lines indicate the three clusters:

$$\text{A} = \{(1,4),(3,4),(2,3)\},$$
$$\text{B} = \{(3,1),(4,2),(5,1)\},$$
$$\text{C} = \{(8,4),(9,5),(10,4)\}.$$

The solid lines indicate two clusters, $\text{D} = \text{A} \cup \text{B}$, and C.

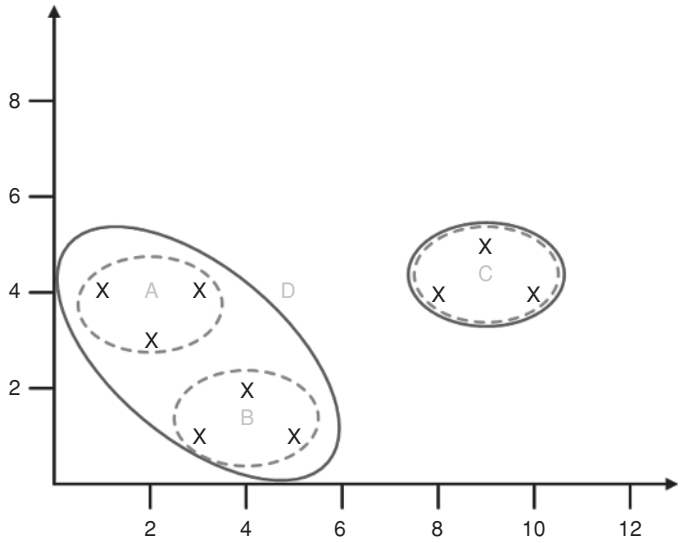

Figure 1.20 Data set for the F-score clustering example (see text). The dashed lines indicate a three-cluster representation, A, B, and C. The solid lines indicate a two-cluster representation, D = A ∪ B, and C

It is left to the reader to show that $F_{\{A,B,C\}} = 11.62$ and $F_{\{D,C\}} = 7.06$. From this, we determine that the three-cluster representation is optimal.

This example, while relatively trivial, illustrates the value of a simple metric—the F-score—to provide very important output—the clusters associated with a data set of any size. The F-score is of huge importance in parallelism by meta-algorithmics, as each of the clusters of data may be analyzed differently based on their different characteristics.

1.10 Recognition

Pattern recognition (Tveter, 1998) plays an important role in many intelligent systems, particularly signal processing and image understanding. The brief overview of recognition is placed here since it builds on many of the technologies introduced in earlier sections of this chapter.

Recognition can be considered in two distinct—both important—ways. The first is *absolute recognition*—wherein an item is categorized as being an object of a certain type, or class. This *identification* can be performed even if the object has not been observed before; for example, it can be identified based on rules or an intelligent grammar of rules (expert system). The second is an awareness that something observed has been observed before. This *relative recognition* is often achieved using *correlation*. Correlation in the broad sense is any measure of dependency of one set of data on another set(s): this means the *k*-NN classification method is a correlative method, since the sample to be classified best correlates, in a positional sense, with the modal class of its *k* nearest training samples.

Examples of relative recognition abound in NLP, in which lemmatization, part-of-speech tagging, summarization, keyword generation, and semantic analysis can be used to describe

P1: TIX/XYZ P2: ABC
JWST304-c01 JWST304-Simske April 2, 2013 7:21 Printer Name: Yet to Come Trim: 244mm × 170mm

a portion of text. Different text elements can be matched relatively, then, by having similar keywords or other salient meta-data. Words with matching lemmas are related conceptually to each other (absolute recognition), whereas words with matching stems may not be (relative recognition).

In imaging applications, correlation between an image and a distortion-free reference image is used to provide absolute recognition. Correlation between an image and a nonreference image is used to provide relative recognition. The differences in correlation can be used for quality assurance, surveillance alerts, tracking, and other object recognition and image understanding applications. Not surprisingly, relative differences in correlation can be used as the input data for clustering.

1.11 System-Based Analysis

This chapter has overviewed many of the core technologies used in machine intelligence systems. As systems get larger—cloud computing scale and beyond—it will be increasingly important to be able to treat the individual knowledge generators as black boxes. Though their internal workings vary significantly, their inputs, outputs, and transfer functions should be well understood by the would-be system architect, as this will allow them to be combined in a variety of parallel approaches.

Systems analysis is the study of interacting components that create a system, with the intent of providing a recommendation, usually related to improvement of the deployed system architecture. In this book, *system-based analysis* is used to further optimize not only the architecture of the system (interaction of components) but also the system components themselves. This is achieved by using three different forms of parallelism—the subject of the next chapter—to optimize the system components. For intelligent systems, these components range from algorithms to large intelligence engines such as automatic speech recognition and image understanding systems.

System-based analysis would be overwhelming without a set of tools to help perform the task. Patterns are essential to direct and even constrain the ways in which to optimize the intelligence components of the system. Much of this book will, in fact, be focused on these patterns.

1.12 Summary

This chapter outlined some of the most important technologies used in intelligent systems. These smart systems include machine intelligence, artificial intelligence, data mining, classification, and recognition systems. This chapter provided enough background to highlight the commonalities and differences among the key machine learning approaches; from SVMs to ensemble methods and from Bayesian to regression methods. Adaptive systems, such as genetic algorithms and neural networks, were shown to be robust to different input to output maps. Others, such as Bayesian systems, were shown to improve as more training data and/or successfully completed tasks were available. The wide diversity of machine learning approaches should be viewed as an opportunity, not a worry. The goal of this book is to show how this great diversity can be used to our advantage in designing systems.

This book is aimed at allowing one with a moderate understanding of statistics, calculus, logic, linear systems, and design patterns to be able to architect and deploy more intelligent

systems than even the best individual intelligent algorithm designer can achieve. The so-empowered meta-architect must understand both the relative advantages and disadvantages, and both the flexibility and the limitations, of each of the component systems. The meta-architect need not understand these individual technologies well enough to improve them as they are; rather, she must understand how to make them more valuable within a larger system involving two or more intelligent components. Let us now consider the types of systems that will require these new system-based analysis skills.

References

Bailey, C.H. and Kandel, E.R. (2008) Synaptic remodeling, synaptic growth and the storage of long-term memory in Aplysia. *Progress in Brain Research*, **169**, 179–198.

Berk, R.A. (2004) *An Introduction to Ensemble Methods for Data Analysis*, Department of Statistics, UCLA, 34 pp., July 25.

Belkin, M. and Niyogi, P. (2003) Using manifold structure for partially labelled classification, in *Advances in NIPS*, vol. 15, MIT Press, p. 8.

Bishop, C.M. (2006) *Pattern Recognition and Machine Learning*, Springer Science + Business Media LLC, New York, 738 pp.

Breiman, L. (2001) Random forests. *Machine Learning*, **45**, 5–32.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984) *Classification and Regression Trees*, Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA.

Cortes, C. and Vapnik, V. (1995) Support-vector networks. *Machine Learning*, **20**, 273–297.

Cox, T.F. and Cox, M.A.A. (1994) *Multidimensional Scaling*, 2nd edn, Chapman & Hall, London, 328 pp.

Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977) Maximum likelihood estimation from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society: Series B*, **39**, 1–38.

Dreyfus, P. (1962) *L'informatique*. Gestion, Paris, pp. 240–241.

Fogel, D.B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ, 272 pp.

Freund, Y. and Schapire, E. (1996) *Experiments with a New Boosting Algorithm*. Proceedings of the 13th International Conference on Machine Learning, pp. 148–156.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 412 pp.

Gould, S.J. and Eldredge, N. (1977) Punctuated equilibria: the tempo and mode of evolution reconsidered. *Paleobiology*, **3** (2), 115–151.

Grimes, C. and Donoho, D.L. (2005) Image manifolds isometric to Euclidean space. *Journal of Mathematical Imaging and Vision*, **23**, 5–24.

Hastie, T., Tibshirani, R., and Friedman, J. (2009) *The Elements of Statistical Learning*, 2nd edn, Springer Science + Business Media LLC, New York, 745 pp.

Ho, T.K. (1998) The random subspace method for constructing decision forests. *The IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20** (8), 832–844.

Hothorn, T. (2003) *Bundling Predictors in R*. Proceedings of the 3rd International Workshop on Distributed Statistical Computing, vol. 3, March 20–22, 10 pp.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J., and Hinton, G.E. (1991) Adaptive mixtures of local experts. *Neural Computation*, **3** (1), 79–87.

Jain, A.K., Duin, R.P.W., and Mao, J. (2000) Statistical pattern recognition: a review. *The IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22** (1), 4–37.

Jolliffe, I.T. (1986) *Principal Component Analysis*, 2nd edn, Springer, New York, 487 pp.

Kandel, E.R., Schwartz, J.H, and Jessell, T.M. (2000) *Principles of Neural Science*, 4th edn, McGraw-Hill, New York, 1414 pp.

Kohonen, T. (1982) Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43** (1), 59–69.

Leondes, C.T. (ed.) (1998) *Image Processing and Pattern Recognition*, Academic Press, San Diego, CA, 386 pp.

Levenberg, K. (1944) A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, **2**, 164–168.

Liu, C. and Rubin, R.B. (1998) Maximum likelihood estimation of factor analysis using the ECME algorithm with complete and incomplete data. *Statistica Sinica*, **8**, 729–747.

Marsland, S. (2009) *Machine Learning: An Algorithmic Perspective*, CRC Press, Boca Raton, FL, 390 pp.

Roweis, S. and Saul, L. (2000) Nonlinear dimensionality reduction for locally linear embedding. Science, **290** (5500), 2323–2326.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986) Learning internal representations by back-propagating errors. *Nature*, **323** (99), 533–536.

Schapire, R.E. (1999) *A Brief Introduction to Boosting*. Proceedings of the 19th International Joint Conference on Artificial Intelligence, vol. 2, pp. 1401–1406.

Sewell, M. (2007) *Ensemble Learning*. Department of Computer Science, University College London, April, revised August 2008, 16 pp.

Simske S. and Matthews, D. (2004) *Navigation Using Inverting Genetic Algorithms: Initial Conditions and Node-Node Transitions*. GECCO 2004, 12 pp.

Simske, S.J., Li, D., and Aronoff, J.S. (2005) A Statistical Method for Binary Classification of Images. *ACM Symposium on Document Engineering*, pp. 127–129.

Shawe-Taylor, J. and Cristianini, N. (2004) *Kernel Methods for Pattern Analysis*, Cambridge University Press, 476 pp.

Tveter, D.R. (1998) *The Pattern Recognition Basis of Artificial Intelligence*, IEEE Computer Society Press, 369 pp.

Vapnik, V. (1995) *The Nature of Statistical Learning Theory*, Springer, Berlin, 334 pp.

Viterbi, A.J. (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, **IT-13**, 260–267.

Wang, J. and Chang, C.-I. (2006) Independent component analysis-based dimensionality reduction with applications in hyperspectral image analysis. IEEE Transactions on Geoscience and Remote Sensing, **44** (6), 1586–1600.

Wolpert, D.H. (1992) Stacked generalization. *Neural Networks*, **5** (2), 241–259.