CHAPTER

Moving on: Applications and Beyond

9

Machine learning is a burgeoning new technology for mining knowledge from data, a technology that a lot of people are beginning to take seriously. We don't want to oversell it. The kind of machine learning we know is not about the big problems: futuristic visions of autonomous robot servants, philosophical conundrums of consciousness, metaphysical issues of free will, evolutionary (or theological) questions of where intelligence comes from, linguistic debates over language learning, psychological theories of child development, or cognitive explanations of what intelligence is and how it works. For us, it's far more prosaic: Machine learning is about algorithms for inferring structure from data and ways of validating that structure. These algorithms are not abstruse and complicated, but they're not completely obvious and trivial either.

Looking forward, the main challenge ahead is applications. Opportunities abound. Wherever there is data, things can be learned from it. Whenever there is too much data for people to pore over themselves, the mechanics of learning will have to be automatic. But the inspiration will certainly not be automatic! Applications will come not from computer programs, nor from machine learning experts, nor from the data itself, but from the people who work with the data and the problems from which it arises. That is why we have written this book, and that is what the Weka system described in Part III is for—to empower those who are not machine learning experts to apply these techniques to problems that arise in daily working life. The ideas are simple. The algorithms are here. The rest is really up to you!

Of course, development of the technology is certainly not finished. Machine learning is a hot research topic, and new ideas and techniques continually emerge. To give a flavor of the scope and variety of research fronts, we close Part II by looking at some topical areas in the world of data mining.

9.1 APPLYING DATA MINING

In 2006 a poll was taken by the organizers of the International Data Mining Conference to identify the top 10 data mining algorithms. Table 9.1 shows the results, in order. It is good to see that they are all covered in this book! The conference organizers divided the algorithms into rough categories, which are also shown. Many of

Table 9.1 Top 10 Algorithms in Data Mining

	Algorithm	Category	Book Section
1	C4.5	Classification	4.3, 6.2
2	*k*-means	Clustering	4.8
3	SVM	Statistical learning	6.4
4	Apriori	Association analysis	4.5, 6.3
5	EM	Statistical learning	6.8
6	PageRank	Link mining	9.6
7	Adaboost	Ensemble learning	8.4
8	kNN	Classification	4.7, 6.5
9	Naïve Bayes	Classification	4.2
10	CART	Classification	6.1

Note: *The information here was obtained during a 2006 poll conducted by the International Data Mining Conference.*

the assignments are rather arbitrary—Naïve Bayes, for example, is certainly a statistical learning method, and we have introduced EM as a statistically based clustering algorithm. Nevertheless, the emphasis on classification over other forms of learning, which reflects the emphasis in this book, is evident in the table, as is the dominance of C4.5, which we have also noted. One algorithm in Table 9.1 that has not been mentioned so far is the PageRank algorithm for link mining, which we were a little surprised to see in this list. Section 9.6 contains a brief description.

We have repeatedly stressed that productive use of data mining is not just a matter of finding some data and then blindly applying learning algorithms to it. Of course, the existence of the Weka workbench makes that easy to do—and therein lies a danger. We have seen many publications that seem to follow this methodology: The authors run a plethora of learning algorithms on a particular dataset and then write an article claiming that such-and-such a machine learning method is best for such-and-such a problem—with little apparent understanding of what those algorithms do, the nature of the data, or consideration of statistical significance. The usefulness of such studies is questionable.

A related but rather different issue concerns the improvements in machine learning methods that have been reported over the years. In a 2006 paper provocatively entitled "Classifier technology and the illusion of progress," David Hand, a prominent statistician and machine learning researcher, points out that a great many algorithms have been devised for supervised classification, and a great many comparative studies have been conducted that apparently establish the superiority of new methods over their predecessors. Yet he contends that the continued steady progress that publication of these studies seems to document is, in fact, to a large extent illusory. This message brings to mind the 1R machine learning scheme some 15

years earlier with which we began Chapter 4. As pointed out there, 1R was never really intended as a machine learning "method" but was devised to demonstrate that putting high-powered inductive inference methods to work on simple datasets is like using a sledgehammer to crack a nut. That insight underlies the simplicity-first methodology that pervades this book, of which Hand's recent paper is a salutary reminder.

How can progress be largely illusory, given documented improvements in measured classification success? The claim is basically that the differences in performance are very small and, in practical applications, are likely to be swamped by other sources of uncertainty. There are many reasons for this. Simple methods may not perform as well as complex ones, but they often perform nearly as well. An extremely simple model—always choose the majority class—sets a baseline upon which any learning method should be able to improve. Consider the improvement over the baseline achieved by a simple method as a proportion of the improvement over the baseline achieved by a sophisticated method. For a variety of randomly chosen datasets, it turns out that a very simple method achieved more than 90% of the improvement yielded by the most sophisticated scheme. This is not so surprising. In standard classification schemes such as decision trees and rules, a huge proportional gain in predictive accuracy is achieved at the beginning of the process when the first branch or rule is determined, and subsequent gains are small—usually very small indeed.

Small improvements are easily swamped by other factors. A fundamental assumption of machine learning is that the training data is representative of the distribution from which future data will be chosen—the assumption is generally that the data is independent and identically distributed (often abbreviated to IID). But in real life, things drift. Yet training data is always retrospective. And it might be quite old. Consider the loan scenario introduced in Section 1.3 (page 22). To collect a substantial volume of training data (and thorough training needs a substantial volume), we must wait until many loans have been issued. And then we must wait until the end of the loan period (two years? five years?) for the outcome to be known. By the time we use it for training, the data is quite old. And what has changed in the meantime? There are new ways of doing things. The bank has changed the way it defines measurements on which the features are based. New features have become available. Policies have altered. Is that ancient data really representative of today's problem?

Another fundamental problem is the reliability of the class labels in the training data. There may be small errors—random or even systematic ones—in which case, perhaps, we should stick to simpler models because the higher-order terms of more complex models may be very inaccurate. In determining class labels, someone, somewhere, may be mapping a gray world onto a black-and-white one, which requires judgment and invites inconsistency. And things may change: The notion of a "defaulter" on a loan—say, unpaid bills for three months—may be subtly different today than it was before—perhaps, in today's economic climate, hard-pressed customers will be given another couple of month's leeway before calling in the bailiffs. The point is not that learning will necessarily fail. The changes may be fairly subtle,

and the learned models may still work well. The point is that the extra few percent gained by a sophisticated model over a simple one may be swamped by other factors.

Another issue, when looking at comparative experiments with machine learning methods, is who is doing the driving. It's not just a matter of firing up the various different methods and recording the results. Many machine learning schemes benefit from tweaking—optimization to fit the problem at hand. Hopefully the data used for tweaking is kept entirely separate from that used for training and testing (otherwise the results are dishonest). But it is natural that an expert in some particular method—maybe the person who developed it—can squeeze more performance out of it than someone else. If they are trying to get their work published, they will certainly want to present the new method in the best possible light. They may not be so experienced at squeezing good performance out of existing, competitive methods—or so diligent. New methods always look better than old ones; also, more complicated schemes are harder to criticize than simpler ones!

The upshot is that small gains in laboratory performance, even though real, may be swamped by other factors when machine learning is applied to a practical data mining problem. If you want to do something worthwhile on a practical dataset, you need to take the entire problem context into account.

9.2 LEARNING FROM MASSIVE DATASETS

The enormous proliferation of very large databases in today's companies and scientific institutions makes it necessary for machine learning algorithms to operate on massive datasets. Two separate dimensions become critical when any algorithm is applied to very large datasets: space and time.

Suppose the data is so large that it cannot be held in main memory. This causes no difficulty if the learning scheme works in an incremental fashion, processing one instance at a time when generating the model. An instance can be read from the input file, the model can be updated, the next instance can be read, and so on—without ever holding more than one training instance in main memory. This is "data stream learning," and we discuss it in the next section. Other methods, such as basic instance-based schemes and locally weighted regression, need access to all the training instances at prediction time. In that case, sophisticated caching and indexing mechanisms have to be employed to keep only the most frequently used parts of a dataset in memory and to provide rapid access to relevant instances in the file.

The other critical dimension when applying learning algorithms to massive datasets is time. If the learning time does not scale linearly (or almost linearly) with the number of training instances, it will eventually become infeasible to process very large datasets. In some applications the number of attributes is a critical factor, and only methods that scale linearly in the number of attributes are acceptable. Alternatively, prediction time might be the crucial issue. Fortunately, there are many learning algorithms that scale gracefully during both training and testing. For example,

the training time for Naïve Bayes is linear in both the number of instances and the number of attributes. For top-down decision tree inducers, we saw in Section 6.1 (page 199) that training time is linear in the number of attributes and, if the tree is uniformly bushy, log-linear in the number of instances (if subtree raising is not used).

When a dataset is too large for a particular learning algorithm to be applied, there are three ways to make learning feasible. The first is trivial: Instead of applying the scheme to the full dataset, use just a small subset for training. Of course, information is lost when subsampling is employed. However, the loss may be negligible because the predictive performance of a learned model often flattens out long before all the training data is incorporated into it. If this is the case, it can easily be verified by observing the model's performance on a holdout test set for training sets of different sizes.

This kind of behavior, called the *law of diminishing returns*, may arise because the learning problem is a simple one, so that a small volume of training data is sufficient to learn an accurate model. Alternatively, the learning algorithm might be incapable of grasping the detailed structure of the underlying domain. This is often observed when Naïve Bayes is employed in a complex domain: Additional training data may not improve the performance of the model, whereas a decision tree's accuracy may continue to climb. In this case, of course, if predictive performance is the main objective, you should switch to the more complex learning algorithm. But beware of overfitting! Take care not to assess performance on the training data.

Parallelization is another way of reducing the time complexity of learning. The idea is to split the problem into smaller parts, solve each using a separate processor, and combine the results together. To do this, a parallelized version of the learning algorithm must be created. Some algorithms lend themselves naturally to parallelization. Nearest-neighbor methods, for example, can be easily distributed among several processors by splitting the data into parts and letting each processor find the nearest neighbor in its part of the training set. Decision tree learners can be parallelized by letting each processor build a subtree of the complete tree. Bagging and stacking (although not boosting) are naturally parallel algorithms. However, parallelization is only a partial remedy because with a fixed number of processors, the algorithm's asymptotic time complexity cannot be improved.

A simple way to apply any algorithm to a large dataset is to split the data into chunks of limited size and learn models separately for each one, combining the results using voting or averaging. Either a parallel bagging-like scheme or a sequential boosting-like scheme can be employed for this purpose. Boosting has the advantage that new chunks can be weighted based on the classifiers learned from previous chunks, thus transferring knowledge between chunks. In both cases, memory consumption increases linearly with dataset size; thus, some form of pruning is necessary for very large datasets. This can be done by setting aside some validation data and only adding a model from a new chunk to the committee classifier if it increases the committee's performance on the validation set. The validation set can also be used to identify an appropriate chunk size by running the method with several different chunk sizes in parallel and monitoring performance on the validation set.

The best but most challenging way to enable a learning paradigm to deal with very large datasets would be to develop new algorithms with lower computational complexity. In some cases, it is provably impossible to derive exact algorithms with lower complexity. Decision tree learners that deal with numeric attributes fall into this category. Their asymptotic time complexity is dominated by the sorting process for the numeric attribute values, a procedure that must be performed at least once for any given dataset. However, stochastic algorithms can sometimes be derived that approximate the true solution but require a much smaller amount of time.

Background knowledge can make it possible to vastly reduce the amount of data that needs to be processed by a learning algorithm. Depending on which attribute is the class, most of the attributes in a huge dataset might turn out to be irrelevant when background knowledge is taken into account. As usual, it pays to carefully engineer the data that is passed to the learning scheme and make the greatest possible use of any prior information about the learning problem at hand. If insufficient background knowledge is available, the attribute filtering algorithms described in Section 7.1 (page 308) can often drastically reduce the amount of data—possibly at the expense of a minor loss in predictive performance. Some of these—for example, attribute selection using decision trees or the 1R learning scheme—are linear in the number of attributes.

To give a feeling for the volume of data that can be handled by straightforward implementations of machine learning algorithms on ordinary microcomputers, we ran Weka's decision tree learner J4.8 on a dataset with 4.9 M instances, 40 attributes (almost all numeric), and a class with 25 values.[1] We used a reasonably modern Linux machine running Sun's 64-bit Java Virtual Machine (Java 1.6) in server mode with 6 Gb of heap space (half of this was required just to load the data). The resulting tree, which had 1388 nodes, took two hours to build. (A method that presorts the attributes and uses reduced-error pruning took only 30 minutes.) In general, Java is a little slower than equivalent C/C++ code—but less than twice as slow.

There are datasets today that truly deserve the adjective *massive*. Scientific datasets from astrophysics, nuclear physics, earth science, and molecular biology are measured in terabytes. So are datasets containing records of financial transactions. Application of standard programs for machine learning to such datasets in their entirety is a very challenging proposition.

9.3 DATA STREAM LEARNING

One way of addressing massive datasets is to develop learning algorithms that treat the input as a continuous data stream. In the new paradigm of data stream mining, which has developed during the last decade, algorithms are developed that cope

[1]We used the 1999 KDD Cup data at *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*. (See also *http://iscx.ca/NSL-KDD*.)

naturally with datasets that are many times the size of main memory—perhaps even indefinitely large. The core assumption is that each instance can be inspected once only (or at most once) and must then be discarded to make room for subsequent instances. The learning algorithm has no control over the order in which instances are processed and must update its model incrementally as each one arrives. Most models also satisfy the "anytime" property—they are ready to be applied at any point during the learning process. Such algorithms are ideal for real-time learning from data streams, making predictions in real time while adapting the model to changes in the evolving input stream. They are typically applied to online learning from data produced by physical sensors.

For such applications, the algorithm must operate indefinitely yet use a limited amount of memory. Even though we have stipulated that instances are discarded as soon as they have been processed, it is obviously necessary to remember at least something about at least some of the instances; otherwise, the model would be static. And as time progresses, the model grows—inexorably. But it must not be allowed to grow without bound. When processing big data, memory is quickly exhausted unless limits are enforced on every aspect of its use. Moving from space to time, algorithms intended for real-time application must process instances faster than they arrive, dealing with each one within a fixed, constant, preferably small, time bound. This does not allow, for example, for occasional complex reorganizations of a tree model—unless the cost can be amortized over several instances, which introduces a further level of complexity.

Naïve Bayes is a rare example of an algorithm that needs no adaptation to deal with data streams. Training is incremental: It merely involves updating a fixed set of numeric parameters. Memory usage is small because no structure is added to the model. Other classifiers with the same properties include 1R and the basic perceptron. Multilayer neural nets usually have a fixed structure as well, and as we saw in Section 6.4 (page 238), stochastic backpropagation updates weights incrementally after each training instance has been processed, rather than in a batch operation, and thus is suitable for online learning. Rules with exceptions make modifications incrementally by expressing exceptions to existing rules rather than reengineering the entire set, and thus could be rendered suitable for data stream learning—although care would need to be taken to ensure that memory usage did not increase inexorably as the number of exceptions increased. Instance-based algorithms and related methods such as locally weighted linear regression are also incremental, but do not usually operate within a fixed memory bound.

To convey the flavor of how a standard algorithm might be adapted for stream processing, we will examine the case of decision trees, which have the advantage of evolving structure in a form that is interpretable. Early work on incremental induction of decision trees devised methods for creating a tree and allowing it to be restructured when sufficient evidence had accumulated that an alternative version would be better. However, a large amount of information needs to be retained to support the restructuring operation—in some cases, all of the training data. Furthermore, restructuring tends to be slow—sometimes slower than recreating the entire

tree from scratch. Although interesting, these methods do not support indefinite processing of data streams in real time.

Their problem is that they adopt the usual paradigm of squeezing as much information as possible out of the available instances. With data streams, this is not necessarily appropriate—it is perfectly acceptable to discard some information about the instances because if it is important it will always reappear. A new paradigm of "Hoeffding trees" was introduced in 2000, which builds models that can be proven equivalent to standard decision trees if the data is static and the number of examples is large enough.

Hoeffding trees are based on a simple idea known as the *Hoeffding bound*. It makes intuitive sense that, given enough independent observations, the true mean of a random variable will not differ from the estimated mean by more than a certain amount. In fact, the Hoeffding bound states that with probability $1 - \delta$, a random variable of range R will not differ from the estimated mean after n observations by more than

$$\varepsilon = \sqrt{\frac{\ln(1/\delta)}{2n}} \times R$$

This bound holds regardless of the probability distribution that underlies the values. Being general, it is more conservative than distribution-dependent bounds. Although tighter bounds are known for particular distributions, the Hoeffding formulation works well empirically.

The basic issue in decision tree induction is to choose an attribute to branch on at each stage. To apply the Hoeffding bound, first set a small value of δ (say 10^{-7}), which is the probability that the choice of attribute will be incorrect. The random variable being estimated is the difference in information gain between the best two attributes, and R is the base two logarithms of the number of possible class labels. For example, if the difference in gain between the best two attributes is estimated to be 0.3, and the preceding formula yields a value for ε of 0.1, the bound guarantees that the actual difference in gain exceeds 0.2 with high probability, which represents positive separation for the best attribute. Thus, it is safe to split.

If the difference in information gain between the best two attributes is less than ε, it is not safe to split. However, ε will decrease as n continues to increase, so it is simply a matter of waiting until more examples have been seen—although, of course, this may alter the estimate of which are the two best attributes and how far apart they are.

This simple test is the core principle of Hoeffding trees: to decide, with probability $1 - \delta$, that a particular attribute exhibits greater information gain than all the others. That is, the gap between it and its closest competitor exceeds ε. The bound decays rapidly as more examples are seen—for example, for a two-class problem ($R = 1$) with $\delta = 10^{-7}$, it falls below 0.1 after the first 1000 examples and below 0.01 after the first 100,000. One might object that as the number of leaves grows indefinitely, the probability of making incorrect decisions will continually increase even

though the probability of error at each one falls below δ. This is true—except that, working within finite memory, the number of leaves cannot grow indefinitely. Given a maximum tree size, keeping the overall probability of error within a given bound is just a matter of choosing an appropriate value for δ. The basic principle can be applied to measures other than the information gain and to learning methods other than decision trees.

There are many other issues. A tie-breaking strategy is advisable to permit further development of the tree in situations where the top two attributes exhibit very similar information gains. Indeed, the presence of two identical attributes could block any development of the tree at all. To prevent this, nodes should be split whenever the Hoeffding bound falls below a small prespecified tie-breaking parameter, no matter how close the next best option. To increase efficiency the Hoeffding test may be performed periodically for each leaf, after k new instances have reached it, and only when a mix of classes have reached the leaf; otherwise, there is no need to split. Prepruning is another simple possibility. The algorithm can incorporate this by also evaluating the merit of not splitting at all—that is, by splitting only if the best attribute's information gain at the node exceeds zero. Unlike prepruning in the batch learning setting, this is not a permanent decision: Nodes are only prevented from splitting until it appears that a split will be useful.

Now consider memory usage. What must be stored within a leaf is simply counts of the number of times each class label reaches that leaf, for each attribute value. This causes problems for numeric attributes, which require separate treatment. Unsupervised discretization is easy, but supervised prediscretization is infeasible because it is inconsistent with stream-based processing. A Gaussian approximation can be made for numeric attributes on a per-class basis and updated using simple incremental update algorithms for mean and variance. To prevent indefinite growth in memory requirements, a strategy must be devised to limit the total number of nodes in the tree. This can be done by deactivating leaves that look insufficiently promising in terms of the accuracy gain that further development might yield. The potential gain is bounded by the expected number of mistakes a leaf might make, so this is an obvious candidate for measuring its promise. Leaves can periodically be ordered from most to least promising and deactivated accordingly. A further possibility for saving space is to abandon attributes that seem to be poor predictors and discard their statistics from the model.

Although this section has focused on decision trees for classification, researchers have studied stream-based versions of all the classical data mining problems: regression, clustering, ensemble methods, association rules, and so on. An open-source system called Moa, for Massive Online Analysis, is closely related to Weka and contains a collection of online learning algorithms, as well as tools for evaluation.[2]

[2]See *http://moa.cs.waikato.ac.nz*. The moa, like the weka, is a flightless New Zealand bird, but it is very large—and also, unfortunately, extinct.

9.4 INCORPORATING DOMAIN KNOWLEDGE

Throughout this book we have emphasized the importance of getting to know your data when undertaking practical data mining. Knowledge of the domain is absolutely essential for success. Data about data is often called *metadata*, and one of the frontiers in machine learning is the development of ways to allow learning methods to take metadata into account in a useful way.

You don't have to look far for examples of how metadata might be applied. In Chapter 2 (page 52), we divided attributes into nominal and numeric. But we also noted that many finer distinctions are possible. If an attribute is numeric an ordering is implied, but sometimes there is a zero point and sometimes not (for time intervals there is, but for dates there is not). Even the ordering may be nonstandard: Angular degrees have a different ordering from integers because 360° is the same as 0° and 180° is the same as −180° or indeed 900°. Discretization schemes assume ordinary linear ordering, as do learning schemes that accommodate numeric attributes, but it would be a routine matter to extend them to circular orderings. Categorical data may also be ordered. Imagine how much more difficult our lives would be if there were no conventional ordering for letters of the alphabet. (Looking up a listing in the Hong Kong telephone directory presents an interesting and nontrivial problem!) And the rhythms of everyday life are reflected in circular orderings: days of the week, months of the year. To further complicate matters there are many other kinds of ordering, such as partial orderings on subsets: subset A may include subset B, or subset B may include subset A, or neither may include the other. Extending ordinary learning schemes to take account of this kind of information in a satisfactory and general way is an open research problem.

Metadata often involves relations among attributes. Three kinds of relation can be distinguished: semantic, causal, and functional. A *semantic* relation between two attributes indicates that if the first is included in a rule, the second should be, too. In this case, it is known a priori that the attributes only make sense together. For example, in the agricultural data that we have analyzed, an attribute called *milk production* measures how much milk an individual cow produces, and the purpose of our investigation meant that this attribute had a semantic relationship with three other attributes: *cow-identifier*, *herd-identifier*, and *farmer-identifier*. In other words, a milk production value can only be understood in the context of the cow that produced the milk, and the cow is further linked to a specific herd owned by a given farmer. Semantic relations are, of course, problem dependent: They depend not just on the dataset but also on what you are trying to do with it.

Causal relations occur when one attribute causes another. In a system that is trying to predict an attribute caused by another, we know that the other attribute should be included to make the prediction more meaningful. For example, in the agricultural data mentioned previously there is a chain from the farmer, herd, and cow identifiers, through measured attributes such as milk production, down to the attribute that records whether a particular cow was retained or sold by the farmer. Learned rules should recognize this chain of dependence.

Functional dependencies occur in many databases, and the people who create databases strive to identify them for the purpose of normalizing the relations in the database. When learning from the data, the significance of a functional dependency of one attribute on another is that if the latter is used in a rule there is no need to consider the former. Learning schemes often rediscover functional dependencies that are already known. Not only does this generate meaningless or, more accurately, tautological rules, but also other, more interesting patterns may be obscured by the functional relationships. However, there has been much work in automatic database design on the problem of inferring functional dependencies from example queries, and the methods developed should prove useful in weeding out tautological rules generated by learning schemes.

Taking these kinds of metadata, or prior domain knowledge, into account when doing induction using any of the algorithms we have met does not seem to present any deep or difficult technical challenges. The only real problem—and it is a big one—is how to express the metadata in a general and easily understandable way so that it can be generated by a person and used by the algorithm.

It seems attractive to couch the metadata knowledge in just the same representation as the machine learning scheme generates. We focus on rules, which are the norm for much of this work. The rules that specify metadata correspond to prior knowledge of the domain. Given training examples, additional rules can be derived by one of the rule-induction schemes we have already met. In this way, the system might be able to combine "experience" (from examples) with "theory" (from domain knowledge). It would be capable of confirming and modifying its programmed-in knowledge based on empirical evidence. Loosely put, the user tells the system what he or she knows and gives it some examples, and the system figures the rest out for itself!

To make use of prior knowledge expressed as rules in a sufficiently flexible way, it is necessary for the system to be able to perform logical deduction. Otherwise, the knowledge has to be expressed in precisely the right form for the learning algorithm to take advantage of it, which is likely to be too demanding for practical use. Consider causal metadata: If attribute A causes B and B causes C, we would like the system to deduce that A causes C rather than having to state that fact explicitly. Although in this simple example explicitly stating the new fact presents little problem, in practice, with extensive metadata, it will be unrealistic to expect users to express all logical consequences of their prior knowledge.

A combination of deduction from prespecified domain knowledge and induction from training examples seems like a flexible way of accommodating metadata. At one extreme, when examples are scarce (or nonexistent), deduction is the prime (or only) means of generating new rules. At the other, when examples are abundant but metadata is scarce (or nonexistent), the standard machine learning techniques described in this book suffice. Practical situations span the territory between.

This is a compelling vision, and methods of inductive logic programming, mentioned in Section 3.4 (page 75), offer a general way of specifying domain knowledge explicitly through statements in a formal logic language. However, current logic programming solutions suffer serious shortcomings in real-world environments. They

tend to be brittle and to lack robustness, and they may be so computation intensive as to be completely infeasible on datasets of any practical size. Perhaps this stems from the fact that they use first-order logic—that is, they allow variables to be introduced into the rules. The machine learning schemes we have seen, the input and output of which are represented in terms of attributes and constant values, perform their machinations in propositional logic, without variables, greatly reducing the search space and avoiding all sorts of difficult problems of circularity and termination.

Some aspire to realize the vision without the accompanying brittleness and computational infeasibility of full logic programming solutions by adopting simplified reasoning systems. Others place their faith in the general mechanism of Bayesian networks, introduced in Section 6.7, in which causal constraints can be expressed in the initial network structure and hidden variables can be postulated and evaluated automatically. Probabilistic logic learning offers a way to cope with both the complexity and the uncertainty of the real world by combining logic programming with statistical reasoning. It will be interesting to see whether systems that allow flexible specification of different types of domain knowledge will become widely deployed.

9.5 TEXT MINING

Data mining is about looking for patterns in data. Likewise, text mining is about looking for patterns in text: It is the process of analyzing text to extract information that is useful for particular purposes. Compared with the kind of data we have been talking about in this book, text is unstructured, amorphous, and difficult to deal with. Nevertheless, in modern Western culture, text is the most common vehicle for the formal exchange of information. The motivation for trying to extract information from it is compelling—even if success is only partial.

The superficial similarity between text and data mining conceals real differences. In the Preface (page xxi), we characterized data mining as the extraction of implicit, previously unknown, and potentially useful information from data. With text mining, however, the information to be extracted is clearly and explicitly stated in the text. It is not hidden at all—most authors go to great pains to make sure that they express themselves clearly and unambiguously. From a human point of view, the only sense in which it is "previously unknown" is that time restrictions make it infeasible for people to read the text themselves. The problem, of course, is that the information is not couched in a manner that is amenable to automatic processing. Text mining strives to bring it out in a form that is suitable for consumption by computers or by people who do not have time to read the full text.

A requirement common to both data and text mining is that the information extracted should be potentially useful. In one sense, this means *actionable*—capable of providing a basis for actions to be taken automatically. In the case of data mining, this notion can be expressed in a relatively domain-independent way: Actionable patterns are ones that allow nontrivial predictions to be made on new data from the same source. Performance can be measured by counting successes and failures,

statistical techniques can be applied to compare different data mining methods on the same problem, and so on. However, in many text mining situations it is hard to characterize what "actionable" means in a way that is independent of the particular domain at hand. This makes it difficult to find fair and objective measures of success.

As we have emphasized throughout this book, "potentially useful" is often given another interpretation in practical data mining: The key to success is that the information extracted must be *comprehensible* in that it helps to explain the data. This is necessary whenever the result is intended for human consumption rather than (or as well as) a basis for automatic action. This criterion is less applicable to text mining because, unlike data mining, the input itself is comprehensible. Text mining with comprehensible output is tantamount to summarizing salient features from a large body of text, which is a subfield in its own right: *text summarization*.

We have already encountered one important text mining problem: *document classification*, in which each instance represents a document and the instance's class is the document's topic. Documents are characterized by the words that appear in them. The presence or absence of each word can be treated as a Boolean attribute, or documents can be treated as bags of words, rather than sets, by taking word frequencies into account. We encountered this distinction in Section 4.2 (page 97), where we learned how to extend Naïve Bayes to the bag-of-words representation, yielding the multinomial version of the algorithm.

There is, of course, an immense number of different words, and most of them are not very useful for document classification. This presents a classic feature-selection problem. Some words—for example function words, often called *stop-words*—can usually be eliminated a priori, but although these occur very frequently there are not all that many of them. Other words occur so rarely that they are unlikely to be useful for classification. Paradoxically, infrequent words are common—nearly half the words in a document or corpus of documents occur just once. Nevertheless, such an overwhelming number of words remain after these word classes are removed that further feature selection may be necessary using the methods described in Section 7.1 (page 307). Another issue is that the bag-of-words (or set-of-words) model neglects word order and contextual effects. There is a strong case for detecting common phrases and treating them as single units.

Document classification is supervised learning: The categories are known beforehand and given in advance for each training document. The unsupervised version of the problem is called *document clustering*. Here there is no predefined class, but groups of cognate documents are sought. Document clustering can assist information retrieval by creating links between similar documents, which in turn allows related documents to be retrieved once one of the documents has been deemed relevant to a query.

There are many applications of document classification. A relatively easy categorization task, *language identification*, provides an important piece of metadata for documents in international collections. A simple representation that works well for language identification is to characterize each document by a profile that consists of the *n-grams*, or sequences of n consecutive letters (for some small value such as

$n = 3$), that appear in it. The most frequent 300 or so n-grams are highly correlated with the language. A more challenging application is *authorship ascription*, where a document's author is uncertain and must be guessed from the text. Here, the stopwords, not the content words, are the giveaway because their distribution is author dependent but topic independent. A third problem is the *assignment of key phrases* to documents from a controlled vocabulary of possible phrases, given a large number of training documents that are tagged from this vocabulary.

Another general class of text mining problems is *metadata extraction*. Metadata was mentioned earlier as data about data: In the realm of text the term generally refers to salient features of a work, such as author, title, subject classification, subject headings, and keywords. Metadata is a kind of highly structured (and therefore actionable) document summary. The idea of metadata is often expanded to encompass words or phrases that stand for objects or "entities" in the world, leading to the notion of *entity extraction*. Ordinary documents are full of such terms: phone numbers, fax numbers, street addresses, email addresses, email signatures, abstracts, tables of contents, lists of references, tables, figures, captions, meeting announcements, web addresses, and more. In addition, there are countless domain-specific entities, such as International Standard Book Numbers (ISBNs), stock symbols, chemical structures, and mathematical equations. These terms act as single vocabulary items, and many document-processing tasks can be significantly improved if they are identified as such. They can aid searching, interlinking, and cross-referencing between documents.

How can textual entities be identified? Rote learning—that is, dictionary lookup—is one idea, particularly when coupled with existing resources—lists of personal names and organizations, information about locations from gazetteers, or abbreviation and acronym dictionaries. Another is to use capitalization and punctuation patterns for names and acronyms; titles (e.g., Ms.), suffixes (e.g., Jr.), and baronial prefixes (e.g., von); or unusual language statistics for foreign names. Regular expressions suffice for artificial constructs such as uniform resource locators (URLs); explicit grammars can be written to recognize dates and sums of money. Even the simplest task opens up opportunities for learning to cope with the huge variation that real-life documents present. As just one example, what could be simpler than looking up a name in a table? But the name of Libyan leader Muammar Qaddafi is represented in 47 different ways in documents that have been received by the Library of Congress!

Many short documents describe a particular kind of object or event, combining entities into a higher-level composite that represents the document's entire content. The task of identifying the composite structure, which can often be represented as a template with slots that are filled by individual pieces of structured information, is called *information extraction*. Once the entities have been found, the text is parsed to determine relationships among them. Typical extraction problems require finding the predicate structure of a small set of predetermined propositions. These are usually simple enough to be captured by shallow parsing techniques such as small finite-state grammars, although matters may be complicated by ambiguous pronoun

references and attached prepositional phrases and other modifiers. Machine learning has been applied to information extraction by seeking rules that extract fillers for slots in the template. These rules may be couched in pattern-action form, the patterns expressing constraints on the slot-filler and words in its local context. These constraints may involve the words themselves, their part-of-speech tags, and their semantic classes.

Taking information extraction a step further, the extracted information can be used in a subsequent step to learn rules—not rules about how to extract information but rules that characterize the content of the text itself. These rules might predict the values for certain slot-fillers from the rest of the text. In certain tightly constrained situations, such as Internet job postings for computing-related jobs, information extraction based on a few manually constructed training examples can compete with an entire manually constructed database in terms of the quality of the rules inferred.

Text mining is a burgeoning technology that is still, because of its newness and intrinsic difficulty, in a fluid state—akin, perhaps, to the state of machine learning in the mid-1980s. There is no real consensus about what it covers: Broadly interpreted, all natural language processing comes under the ambit of text mining. It is usually difficult to provide general and meaningful evaluations because the mining task is highly sensitive to the particular text under consideration. Automatic text mining techniques have a long way to go before they rival the ability of people, even without any special domain knowledge, to glean information from large document collections.

9.6 WEB MINING

The World Wide Web is a massive repository of text. Almost all of it differs from ordinary "plain" text because it contains explicit structural markup. Some markup is internal and indicates document structure or format; other markup is external and defines explicit hypertext links between documents. Both these information sources give additional leverage for mining web documents. *Web mining* is like text mining but takes advantage of this extra information and often improves results by capitalizing on the existence of topic directories and other information on the Web.

Consider internal markup. Internet resources that contain relational data—telephone directories, product catalogs, and so on—use HyperText Markup Language (HTML) formatting commands to clearly present the information they contain to Web users. However, it is quite difficult to extract data from such resources in an automatic way. To do so, software systems use simple parsing modules called *wrappers* to analyze the page structure and extract the requisite information. If wrappers are coded by hand, which they often are, this is a trivial kind of text mining because it relies on the pages having a fixed, predetermined structure from which information can be extracted algorithmically. But pages rarely obey the rules. Their structures vary; web sites evolve. Errors that are insignificant to human readers throw

automatic extraction procedures completely awry. When change occurs, adjusting a wrapper manually can be a nightmare that involves getting your head around the existing code and patching it up in a way that does not cause breakage elsewhere.

Enter *wrapper induction*—learning wrappers automatically from examples. The input is a training set of pages along with tuples representing the information derived from each page. The output is a set of rules that extracts the tuples by parsing the page. For example, it might look for certain HTML delimiters—paragraph boundaries (*<p>*), list entries (*<li>*), or boldface (*<b>*)—that the web page designer has used to set off key items of information, and learn the sequence in which entities are presented. This could be accomplished by iterating over all choices of delimiters, stopping when a consistent wrapper is encountered. Then recognition will depend only on a minimal set of cues, providing some defense against extraneous text and markers in the input. Alternatively, one might follow Epicurus' advice at the end of Section 5.9 (page 186) and seek a robust wrapper that uses multiple cues to guard against accidental variation. The great advantage of automatic wrapper induction is that when errors are caused by stylistic variants it is a simple matter to add these to the training data and reinduce a new wrapper that takes them into account. Wrapper induction reduces recognition problems when small changes occur and makes it far easier to produce new sets of extraction rules when structures change radically.

One of the problems with the Web is that a lot of it is rubbish. In order to separate the wheat from the chaff, a metric called PageRank was introduced by the founders of Google; it is used in various guises by other search engines too, and in many other web mining applications. It attempts to measure the prestige of a web page or site, where *prestige* is, according to a dictionary definition, "high standing achieved through success or influence." The hope is that this is a good way to determine authority, defined as "an accepted source of expert information or advice." Recall that the PageRank algorithm was identified earlier in Table 9.1 as one of the top 10 data mining algorithms, the only one that we have not encountered so far. It is perhaps questionable whether it should be classed as a data mining algorithm, but it is worth describing all the same.

The key is external markup in the form of hyperlinks. In a networked community, people reward success with links. If you link to my page, it's probably because you find it useful and informative—it's a successful web page. If a host of people link to it, that indicates prestige: My page is successful and influential. Look at Figure 9.1, which shows a tiny fraction of the Web, including links between pages. Which ones do you think are most authoritative? Page **F** has five incoming links, which indicates that five people found it worth linking to, so there's a good chance that this page is more authoritative than the others. **B** is second best, with four links.

Merely counting links is a crude measure. Some web pages have thousands of outgoing links, whereas others have just one or two. Rarer links are more discriminating and should count more than others. A link from your page to mine bestows more prestige if your page has few outlinks. In Figure 9.1 the many links emanating from page **A** mean that each one carries less weight simply because **A** is a prolific linker. From **F**'s point of view, the links from **D** and **E** may be more valuable than

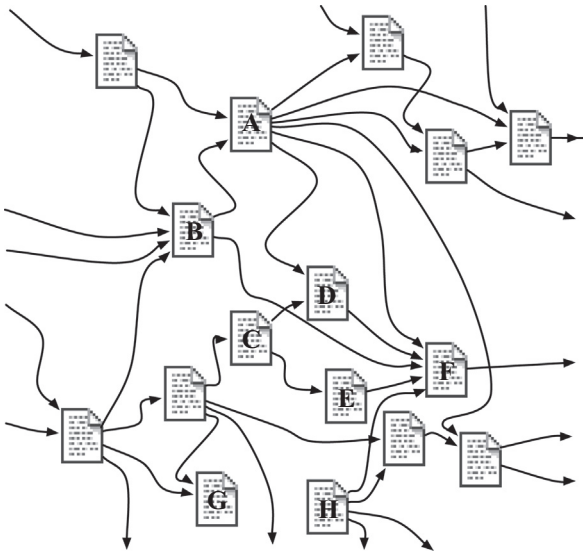

FIGURE 9.1

A tangled "web."

the one from **A**. There is another factor: A link is more valuable if it comes from a prestigious page. The link from **B** to **F** may be better than the others into **F** because **B** is more prestigious. Admittedly, this factor involves a certain circularity, and without further analysis it's not clear that it can be made to work. But indeed it can.

Here are the details. We define the PageRank of a page to be a number between 0 and 1 that measures its prestige. Each link into the page contributes to its PageRank. The amount it contributes is the PageRank of the linking page divided by the number of outlinks from it. The PageRank of any page is calculated by summing that quantity over all links into it. The value for **D** in Figure 9.1 is calculated by adding one-fifth of the value for **A** (because it has five outlinks) to one-half the value for **C**.

A simple iterative method is used to resolve the apparently circular nature of the calculation. Start by randomly assigning an initial value to each page. Then recompute each page's PageRank by summing the appropriate quantities, described earlier, over its inlinks. If the initial values are thought of as an approximation to the true value of PageRank, the new values are a better approximation. Keep going, generating a third approximation, a fourth, and so on. At each stage, recompute the PageRank for every page in the Web. Stop when, for every page, the next iteration turns out to give almost exactly the same PageRank as the previous one.

Subject to two modifications discussed later, this iteration is guaranteed to converge, and fairly quickly too. Although the precise details are shrouded in secrecy, today's search engines probably seek an accuracy for the final values of between

10^{-9} and 10^{-12}. An early experiment reported 50 iterations for a much smaller version of the Web than the one that exists today, before the details became commercial; several times as many iterations are needed now. Google is thought to run programs for several days to perform the PageRank calculation for the entire Web, and the operation is—or at any rate, used to be—performed every few weeks.

There are two problems with the calculation we have described. You probably have a mental picture of PageRank flowing through the tangled "web" of Figure 9.1, coming into a page through its inlinks and leaving it through its outlinks. What if there are no inlinks (page **H**)? Or no outlinks (page **G**)?

To operationalize this picture, imagine a web surfer who clicks links at random. He takes the current page, chooses an outlink at random, and goes to that link's target page. The probability of taking any particular link is smaller if there are many outlinks, which is exactly the behavior we want from PageRank. It turns out that the PageRank of a given page is proportional to the probability that the surfer randomly searching lands on that page.

Now the problem raised by a page with no outlinks becomes apparent: It's a PageRank sink because when surfers come in they cannot get out. More generally, a set of pages might link to each other but not to anywhere else. This incestuous group is also a PageRank sink: The random surfer gets stuck in a trap. And a page with no inlinks? Random surfers never reach it. In fact, they never reach any group of pages that has no inlinks from the rest of the Web, even though it may have internal links and outlinks to the Web at large.

These two problems mean that the iterative calculation described above does not converge, as we earlier claimed it would. But the solution is simple: *teleportation*. With a certain small probability, just make the surfer arrive at a randomly chosen page instead of following a link from the one she is on. That solves both problems. If surfers are stuck at **G** they will eventually teleport out of it. And if they can't reach **H** by surfing, they will eventually teleport into it.

The teleport probability has a strong influence on the rate of convergence of the iterative algorithm—and on the accuracy of its results. At the extreme, if it were equal to 1, meaning that the surfer always teleported, the link structure would have no effect on PageRank, and no iteration would be necessary. If it were 0 and the surfer never teleported, the calculation would not converge at all. Early published experiments used a teleportation probability of 0.15; some speculate that search engines increase it a little to hasten convergence.

Instead of teleporting to a randomly chosen page, you could choose a predetermined probability for each page, and—once you had decided to teleport—use that probability to determine where to land. This does not affect the calculation. But it does affect the result. If a page was discriminated against by receiving a smaller probability than the others, it would end up with a smaller PageRank than it deserves. This gives search engine operators an opportunity to influence the results of the calculation—an opportunity that they probably use to discriminate against certain sites (e.g., ones they believe are trying to gain an unfair advantage by exploiting the PageRank system). This is the stuff of which lawsuits are made.

9.7 ADVERSARIAL SITUATIONS

A prime application of machine learning is junk email filtering. When we wrote the second edition of this book (2005), the scourge of unwanted email was a burning issue; now, as we write the third edition (2011), the problem seems to have abated despite the continual growth of spam (by some estimates it accounts for 95% of all emails). This is largely due to the widespread use of spam filtering, which often uses learning techniques. At first blush, junk email filtering appears to present a standard problem of document classification: Divide documents into "ham" and "spam" on the basis of the text they contain, guided by training data, of which there are copious amounts. But it differs from ordinary document classification because it involves an adversarial aspect. The documents that are being classified are not chosen at random from an unimaginably huge set of all possible documents; they contain emails that are carefully crafted to evade the filtering process, designed specifically to beat the system.

Early spam filters simply discarded messages containing "spammy" words that connote such things as sex, lucre, and quackery. Of course, much legitimate correspondence concerns gender, money, and medicine: A balance must be struck. So filter designers recruited Bayesian text classification schemes that learned to strike an appropriate balance during the training process. Spammers quickly adjusted with techniques that concealed the spammy words by misspelling them; overwhelmed them with legitimate text, perhaps printed in white on a white background so that only the filter saw it; or simply put the spam text elsewhere, in an image or a URL that most mail readers download automatically.

The problem is complicated by the fact that it is hard to compare spam detection algorithms objectively. Although training data abounds, privacy issues preclude publishing large public corpora of representative email. And there are strong temporal effects. Spam changes character rapidly, invalidating sensitive statistical tests such as cross-validation. Finally, the bad guys can also use machine learning. For example, if they could get hold of examples of what your filter blocks and what it lets through, they could use this as training data to learn how to evade filtering.

There are, unfortunately, many other examples of adversarial learning situations in our world today. Closely related to junk email is search engine spam: sites that attempt to deceive Internet search engines into placing them prominently in lists of search results. Highly ranked pages yield direct financial benefits to their owners because they present opportunities for advertising, providing strong motivation for profit seekers. Then there are the computer virus wars, in which designers of viruses and virus protection software react to one another's innovations. Here the motivation tends to be general disruption and denial of service rather than monetary gain.

Computer network security is a continually escalating battle. Protectors harden networks, operating systems, and applications, and attackers find vulnerabilities in all three areas. Intrusion detection systems sniff out unusual patterns of activity that might be caused by a hacker's reconnaissance activity. Attackers realize this and try to obfuscate their trails, perhaps by working indirectly or by spreading their activities

over a long time—or, conversely, by striking very quickly. Data mining is being applied to this problem in an attempt to discover semantic connections among attacker traces in computer network data that intrusion detection systems miss. This is a large-scale problem: Audit logs used to monitor computer network security can amount to gigabytes a day even in medium-size organizations.

Many automated threat detection systems are based on matching current data to known attack types. The U.S. Federal Aviation Administration developed the Computer-Assisted Passenger Prescreening System (CAPPS), which screens airline passengers on the basis of their flight records and flags individuals for additional checked baggage screening. Although the exact details are unpublished, CAPPS is, for example, thought to assign higher threat scores to cash payments. However, this approach can only spot known or anticipated threats. Researchers are using unsupervised approaches such as anomaly and outlier detection in an attempt to detect suspicious activity. As well as flagging potential threats, anomaly detection systems can be applied to the detection of illegal activities such as financial fraud and money laundering.

Data mining is being used today to sift through huge volumes of data in the name of homeland defense. Heterogeneous information such as financial transactions, healthcare records, and network traffic is being mined to create profiles, construct social network models, and detect terrorist communications. This activity raises serious privacy concerns and has resulted in the development of privacy-preserving data mining techniques. These algorithms try to discern patterns in the data without accessing the original data directly, typically by distorting it with random values. To preserve privacy, they must guarantee that the mining process does not receive enough information to reconstruct the original data. This is easier said than done.

On a lighter note, not all adversarial data mining is aimed at combating nefarious activity. Multi-agent systems in complex, noisy, real-time domains involve autonomous agents that must both collaborate as a team and compete against antagonists. If you are having trouble visualizing this, think soccer. Robo-soccer is a rich and popular domain for exploring how machine learning can be applied to such difficult problems. Players must not only hone low-level skills but must also learn to work together and adapt to the behavior patterns of different opponents.

Finally, machine learning has been used to solve an actual historical literary mystery by unmasking a prolific author who had attempted to conceal his identity. As Koppel and Schler (2004) relate, Ben Ish Chai was the leading rabbinic scholar in Baghdad in the late nineteenth century. Among his vast literary legacy are two separate collections of about 500 Hebrew–Aramaic letters written in response to legal queries. He is known to have written one collection. Although Chai claims to have found the other in an archive, historians suspect that he wrote it, too but attempted to disguise his authorship by deliberately altering his style. The problem this case presents to machine learning is that there is no corpus of work to ascribe to the mystery author. There were a few known candidates, but the letters could equally well have been written by anyone. A new technique appropriately called *unmasking* was developed that creates a model to distinguish the known author's

work A from the unknown author's work X, iteratively removes those features that are most useful for distinguishing the two, and examines the speed with which cross-validation accuracy degrades as more features are removed.

The hypothesis is that if work X is written by work A's author, who is trying to conceal his identity, whatever differences there are between work X and work A will be reflected in only a relatively small number of features compared with the differences between work X and the works of a completely different author, say the author of work B. In other words, when work X is compared with works A and B, the accuracy curve as features are removed will decline much faster for work A than it does for work B. Koppel and Schler concluded that Ben Ish Chai did indeed write the mystery letters, and their technique is a striking example of original and creative use of machine learning in an adversarial situation.

9.8 UBIQUITOUS DATA MINING

We began this book by pointing out that we are overwhelmed with data. Nowhere does this impact the lives of ordinary people more than on the World Wide Web. At present, the Web contains around 10 to 20 billion documents, totaling perhaps 50 Tb—and it continues to grow. No one can keep pace with the information explosion. Whereas data mining originated in the corporate world because that's where the databases are, text mining is moving machine learning technology out of the companies and into the home. Whenever we are overwhelmed by data on the Web, text mining promises tools to tame it. Applications are legion. Finding friends and contacting them, maintaining financial portfolios, shopping for bargains in an electronic world, using data detectors of any kind—all of these could be accomplished automatically without explicit programming. Already, text mining techniques are being used to predict what link you're going to click next, to organize documents for you, sort your mail, and prioritize your search results. In a world where information is overwhelming, disorganized, and anarchic, text mining may be the solution we so desperately need.

Many believe that the Web is but the harbinger of an even greater paradigm shift known as *ubiquitous computing*. Small portable devices are everywhere—mobile telephones, personal digital assistants, personal stereo and video players, digital cameras, mobile Web access. Already some devices integrate all these functions. They know our location in physical time and space, help us communicate in social space, organize our personal planning space, recall our past, and envelop us in global information space. It is easy to find dozens of processors in a American middle-class home today. They do not communicate with one another or with the global information infrastructure—yet. But they will, and when they do the potential for data mining will soar.

Take consumer music. Popular music leads the vanguard of technological advance. Sony's original Walkman paved the way to today's ubiquitous portable electronics. Apple's iPOD pioneered large-scale portable storage. Napster's network

technology spurred the development of peer-to-peer protocols. Recommender systems such as Firefly brought computing to social networks. Content-aware music services are migrating to portable devices. Applications for data mining in networked communities of people will be legion: discovering musical trends, tracking preferences and tastes, and analyzing listening behaviors.

Ubiquitous computing will weave digital space closely into real-world activities. To many—extrapolating their own computer experiences of extreme frustration, arcane technology, perceived personal inadequacy, and machine failure—this sounds like a nightmare. But proponents point out that it can't be like that because, if it is, it won't work. Today's visionaries foresee a world of "calm" computing in which hidden machines silently cooperate behind the scenes to make our lives richer and easier. They'll reach beyond the big problems of corporate finance and school homework to the little annoyances such as where are the car keys, can I get a parking place, and is that shirt I saw last week at Macy's still on the rack? Clocks will find the correct time after a power failure, the microwave will download new recipes from the Internet, kid's toys will refresh themselves with new games and new vocabularies. Clothes labels will track washing, coffee cups will alert cleaning staff to mold, light switches will save energy if no one is in the room, and pencils will digitize everything we draw. Where will data mining be in this new world? Everywhere!

It's difficult to point to examples of a future that does not yet exist. But the advances in user interface technology are suggestive. Many repetitive tasks in direct-manipulation computer interfaces cannot be automated with standard application tools, forcing users to perform the same interface actions over and over again. This typifies the frustrations alluded to previously: Who's in charge—me or it? Experienced programmers might write a script to carry out such tasks on their behalf, but as operating systems accrue layer upon layer of complexity, the power of programmers to command the machine is eroded; it vanishes altogether when complex functionality is embedded in appliances rather than in general-purpose computers.

Research in *programming by demonstration* enables ordinary users to automate predictable tasks without requiring any programming knowledge at all. The user need only know how to perform the task in the usual way to be able to communicate it to the computer. One system, called *Familiar*, helps users automate iterative tasks involving existing applications on Macintosh computers. It works across applications and can work with completely new ones never before encountered. It does this by using Apple's scripting language to glean information from each application and exploiting that information to make predictions. The agent tolerates noise. It generates explanations to inform the user about its predictions, and it incorporates feedback. It's adaptive: It learns specialized tasks for individual users. Furthermore, it is sensitive to each user's style. If two people were teaching a task and happened to give identical demonstrations, Familiar would not necessarily infer identical programs—it's tuned to users' habits because it learns from their interaction history.

Familiar employs standard machine learning techniques to infer the user's intent. Rules are used to evaluate predictions so that the best one can be presented to the

user at each point. These rules are conditional so that users can teach classification tasks such as sorting files based on their type and assigning labels based on their size. They are learned incrementally: The agent adapts to individual users by recording their interaction history.

Many difficulties arise. One is scarcity of data. Users are loath to demonstrate several iterations of a task—they think the agent should immediately catch on to what they are doing. Whereas a data miner would consider a 100-instance dataset miniscule, users bridle at the prospect of demonstrating a task even half a dozen times. A second difficulty is the plethora of attributes. The computer desktop environment has hundreds of features that any given action might depend on. This means that small datasets are overwhelmingly likely to contain attributes that are apparently highly predictive but nevertheless irrelevant, and specialized statistical tests are needed to compare alternative hypotheses. A third difficulty is that the iterative, improvement-driven development style that characterizes data mining applications fails. It is impossible *in principle* to create a fixed training and testing corpus for an interactive problem, such as programming by demonstration, because each improvement in the agent alters the test data by affecting how users react to it. A fourth difficulty is that existing application programs provide limited access to application and user data: Often, the raw material on which successful operation depends is inaccessible, buried deep within the application program.

Data mining is already widely used at work. Text and web mining is bringing the techniques in this book into our own lives as we read our email and surf the Web. As for the future, it will be stranger than we can imagine. The spreading computing infrastructure will offer untold opportunities for learning. Data mining will be in there, behind the scenes, playing a role that will turn out to be foundational.

9.9 FURTHER READING

Wu et al. (2008) describe the process of identifying the top 10 algorithms in data mining for presentation at the International Conference on Data Mining in 2006 in Hong Kong; they have followed this up with a book that describes all the algorithms (Wu and Kumar, 2009). The paper on the "illusion of progress" in classification is by Hand (2006), and it was he who found that a very simple method achieves more than 90 percent of the classification improvement yielded by the most sophisticated scheme.

There is a substantial volume of literature that treats the topic of massive datasets, and we can only point to a few references here. Fayyad and Smyth (1995) describe the application of data mining to voluminous data from scientific experiments. Shafer et al. (1996) describe a parallel version of a top-down decision tree inducer. A sequential decision tree algorithm for massive disk-resident datasets has been developed by Mehta et al. (1996). The technique of applying any algorithm to a large dataset by splitting it into smaller chunks and bagging or boosting the result

is described by Breiman (1999); Frank et al. (2002) explain the related pruning and selection scheme.

Early work on incremental decision trees is reported by Utgoff (1989) and Utgoff et al. (1997). The Hoeffding tree was introduced by Domingos and Hulten (2000). Our description of it, including extensions and improvement, closely follows Kirkby's Ph.D. thesis (2007). The Moa system is described by Bifet et al. (2010).

Despite its importance, comparatively little seems to have been written about the general problem of incorporating metadata into practical data mining. A scheme for encoding domain knowledge into propositional rules and its use for both deduction and induction has been investigated by Giraud-Carrier (1996). The related area of inductive logic programming, which deals with knowledge represented by first-order logic rules, is covered by Bergadano and Gunetti (1996). Probabilistic logic learning is covered by de Raedt (2008).

Text mining is an emerging area, and there are few comprehensive surveys of the area as a whole: Witten (2004) provides one. A large number of feature-selection and machine learning techniques have been applied to text categorization (Sebastiani, 2002). Martin (1995) describes applications of document clustering to information retrieval. Cavnar and Trenkle (1994) show how to use *n*-gram profiles to ascertain with high accuracy the language in which a document is written. The use of support vector machines for authorship ascription is described by Diederich et al. (2003); the same technology was used by Dumais et al. (1998) to assign key phrases from a controlled vocabulary to documents on the basis of a large number of training documents. The use of machine learning to extract key phrases from document text has been investigated by Turney (1999), Frank et al. (1999), and Medelyan and Witten (2008).

Appelt (1996) describes many problems of information extraction. Many authors have applied machine learning to seek rules that extract slot-fillers for templates, for example, Soderland et al. (1995), Huffman (1996), and Freitag (2002). Califf and Mooney (1999) and Nahm and Mooney (2000) investigated the problem of extracting information from job ads posted by Internet newsgroups. An approach to finding information in running text based on compression techniques has been reported by Witten et al. (1999a). Mann (1993) notes the plethora of variations of Muammar Qaddafi in documents received by the U.S. Library of Congress.

Chakrabarti (2003) has written an excellent and comprehensive book on techniques of web mining. Kushmerick et al. (1997) developed techniques of wrapper induction. The founders of Google wrote an early paper that introduced the PageRank algorithm (Brin and Page, 1998). At the same time, Kleinberg (1998) described a system called HITS (hypertext-induced topic selection) that has some superficial similarities with PageRank but produces strikingly different results.

The first paper on junk email filtering was written by Sahami et al. (1998). Our material on computer network security is culled from Yurcik et al. (2003). The information on the CAPPS system comes from the U.S. House of Representatives Subcommittee on Aviation (2002), and the use of unsupervised learning for threat

detection is described by Bay and Schwabacher (2003). Problems with current privacy-preserving data mining techniques have been identified by Datta et al. (2003). Stone and Veloso (2000) surveyed multi-agent systems of the kind that are used for playing robo-soccer from a machine learning perspective. The fascinating story of Ben Ish Chai and the technique used to unmask him is from Koppel and Schler (2004).

The vision of calm computing, as well as the examples we have mentioned, is from Weiser and Brown (1997). More information on different methods of programming by demonstration can be found in compendia by Cypher (1993) and Lieberman (2001). Mitchell et al. (1994) report some experience with learning apprentices. Familiar is described by Paynter (2000). Permutation tests (Good, 1994) are statistical tests that are suitable for small sample problems: Frank (2000) describes their application in machine learning.