1 Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges

THOMAS G DIETTERICH

School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, Oregon, 97331 USA

PAT LANGLEY

Institute for the Study of Learning and Expertise
2164 Staunton Court
Palo Alto, CA 94306 USA

1.1 INTRODUCTION

Clark, Partridge, Ramming, and Wroclawski [9, 36, 10] recently proposed a new vision for computer network management—the Knowledge Plane—that would augment the current paradigm of low-level data collection and decision making with higher-level processes. One key idea is that the Knowledge Plane would learn about its own behavior over time, making it better able to analyze problems, tune its operation, and generally increase its reliability and robustness. This suggests the incorporation of concepts and methods from machine learning [23, 28], an established field that is concerned with such issues.

Machine learning aims to understand computational mechanisms by which experience can lead to improved performance. In everyday language, we say that a person has 'learned' something from an experience when he can do something he could not,

dietterich-langley-loop.eps goes here

Fig. 1.1 Relationship between learning, performance, knowledge, and the environment.

or could not do as well, before that experience. The field of machine learning attempts to characterize how such changes can occur by designing, implementing, running, and analyzing algorithms that can be run on computers. The discipline draws on ideas from many other fields, including statistics, cognitive psychology, information theory, logic, complexity theory, and operations research, but always with the goal of understanding the computational character of learning.

There is general agreement that representational issues are central to learning. In fact, the field is often divided into paradigms that are organized around representational formalisms, such as decision trees, logical rules, neural networks, case libraries, and probabilistic notations. Early debate revolved around which formalism provided the best support for machine learning, but the advent of experimental comparisons around 1990 showed that, in general, no formalism led to better learning than any other. However, it also revealed that the specific features or representational encodings mattered greatly, and careful feature engineering remains a hallmark of successful applications of machine learning technology [26].

Another common view is that learning always occurs in the context of some *performance* task, and that a learning method should always be coupled with a performance element that uses the knowledge acquired or revised during learning. Figure 1 depicts such a combined system, which experiences the environment, uses learning to transform those experiences into knowledge, and makes that knowledge available to a performance module that operates in the environment. Performance refers to the behavior of the system when learning is disabled. This may involve a simple activity, such as assigning a label or selecting an action, but it may also involve complex reasoning, planning, or intepretation. The general goal of learning is to improve performance on whatever task the combined system is designed to carry out.

We should clarify a few more points about the relations between learning, performance, and knowledge. The figure suggests that the system operates in a continuing loop, with performance generating experiences that produce learning, which in turn leads to changes in performance, and so on. This paradigm is known as *on-line* learning, and characterizes some but not all research in the area. A more common approach, known as *off-line* learning, instead assumes that the training experiences are all available at the outset, and that learning transforms these into knowledge only once. The figure also includes an optional link that lets the system's current knowledge influence the learning process. This idea is not widely used in current research, but it can assist learning significantly when relevant knowledge is available.

In this chapter, we examine various aspects of machine learning that touch on cognitive approaches to networking. We begin by reviewing the major problem formulations that have been studied in machine learning. Then we consider three tasks that the Knowledge Plane is designed to support and the roles that learning

could play in them. Next we discuss some open issues and research challenges that the Knowledge Plane poses for the field of machine learning. Finally, we propose some methods and criteria for evaluating the contribution of machine learning to cognitive networking tasks.

1.2 PROBLEM FORMULATIONS IN MACHINE LEARNING

Treatments of machine learning (e.g., [23, 28]) typically organize the field along representational lines, depending on whether one encodes learned knowledge using decision trees, neural networks, case libraries, probabilistic summaries, or some other notation. However, a more basic issue concerns how one *formulates* the learning task in terms of the inputs that drive learning and the manner in which the learned knowledge is utilized. This section examines three broad formulations of machine learning.

1.2.1 Learning for Classification and Regression

The most common formulation focuses on learning knowledge for the performance task of *classification* or *regression*. Classification involves assigning a test case to one of a finite set of classes, whereas regression instead predicts the case's value on some continuous variable or attribute. In the context of network diagnosis, one classification problem is deciding whether a connection failure is due to the target site being down, the target site being overloaded, or the ISP service being down. An analogous regression problem might involve predicting the time it will take for the connection to return. Cases are typically described as a set of values for discrete or continuous attributes or variables. For example, a description of the network's state might include attributes for packet loss, transfer time, and connectivity. Some work on classification and regression instead operates over relational descriptors. Thus, one might describe a particular situation in terms of node connections and whether numeric attributes at one node (e.g., buffer utilization) are higher than those at an adjacent node.

In some situations, there is no special attribute that one knows at the outset will be predicted from others. Instead, one may need to predict the value of any unobserved attributes in terms of others that have been observed. This performance task, often called *pattern completion* or *flexible prediction*, can be used for symbolic attributes, continuous attributes, or a mixture of them. For example, given information about some network variables that are measured easily and cheaply, one might want to predict the values of other network variables that are more expensive to measure. A related task involves predicting the conditional probabilities that different values will occur for unknown variables given observed values for others. Alternatively, one may want to predict the joint probability distribution over the entire set of variables.

One can formulate a number of distinct learning tasks that produce knowledge for use in classification or regression. The most common, known as *supervised learning*, assumes the learner is given training cases with associated classes or values for the

attribute to be predicted. For example, one might provide a supervised learning method with 200 instances of four different types of connection failure, say 50 instances of each class, with each instance described in terms of the attributes to be used later during classification. The analogous version for regression would provide instead the time taken to restore the connection for each training instance.

There exist a variety of well-established paradigms for supervised learning, including decision-tree and rule induction [38, 11], neural network methods [39], support-vector machines [12], nearest neighbor approaches [1], and probabilistic methods [7]. These frameworks differ in the formalisms they employ for representing learned knowledge, as well as their specific algorithms for using and learning that knowledge. What these methods hold in common is their reliance on a target class or response variable to direct their search through the space of predictive models. They also share a common approach to evaluation, since their goal is to induce predictive models from training cases that have low error on novel test cases.

A second broad class of tasks, *unsupervised learning*, assumes that the learner is given training cases without any associated class information or any specific attribute singled out for prediction. For example, one might provide an unsupervised method with the same 200 instances as before, but not include any information about the type of connection failure or the time taken to restore the connection.

As with supervised learning, there exist many techniques for learning from unsupervised data, but these fall into two broad classes. One approach, known as *clustering* [17, 8], assumes the goal of learning is to assign the training instances to distinct classes of its own invention, which can be used to classify novel instances and make inferences about them, say through pattern completion. For example, a clustering algorithm might group the 200 training instances into a number of classes that represent what it thinks are different types of service interruption. Another approach, known as *density estimation* [37], instead aims to build a model that predicts the probability of occurrence for specific instances. For example, given the same data about service interruptions, such a method would generate a probability density function that covers both the training instances and novel ones.

A third formulation, known as *semi-supervised learning* [4], falls between the two approaches we have already discussed. In this framework, some of the training instances come with associated classes or values for predicted attributes, but others (typically the majority) do not have this information. This approach is common in domains such as text classification, where training cases are plentiful but class labels are costly. The goal is similar to that for supervised learning, that is, to induce a classifier or regressor that makes accurate predictions, but also to utilize the unlabeled instances to improve this behavior. For example, even if only 20 of the 200 training instances on service interruption included class information, one might still use regularities in the remaining instances to induce more accurate classifiers.

Classification and regression are the most basic capabilities for which learning can occur. As a result, the field has developed robust methods for these tasks and they have been applied widely to develop accurate and useful predictive models from data. Langley and Simon [26] review some early successes of these methods, and they have since formed the backbone for many commercial applications within the

data-mining movement. Methods for classification and regression learning can also play a role in more complex tasks, but such tasks also introduce other factors that require additional mechanisms, as discussed in the next section.

1.2.2 Learning for Acting and Planning

A second formulation addresses learning of knowledge for selecting actions or plans for an agent to carry out in the world. In its simplest form, action selection can occur in a purely reactive way, ignoring any information about past actions. This version has a straightforward mapping onto classification, with alternative actions corresponding to distinct classes from which the agent can choose based on descriptions of the world state. One can also map it onto regression, with the agent predicting the overall value or utility of each action in a given world state.

Both approaches can also be utilized for problem solving, planning, and scheduling. These involve making cognitive choices about future actions, rather than about immediate actions in the environment. Such activities typically involve search through a space of alternatives, which knowledge can be used to constrain or direct. This knowledge may take the form of classifiers for which action to select or regression functions over actions or states. However, it can also be cast as larger-scale structures called *macro-operators* that specify multiple actions that should be carried out together.

As with classification and regression, one can formulate a number of learning tasks that produce knowledge for action selection and search. The simplest approach, known as a *learning apprentice* [29] or an *adaptive interface* [25], embeds the learner within a larger system that interacts with a human user. This system may accept directions from the user about what choices to make or it may make recommendations to the user, who can then accept them or propose other responses. Thus, the user gives direct feedback to the system about each choice, effectively transforming the problem of learning to select actions into a supervised learning task, which can then be handled using any of the methods discussed earlier. A related paradigm, known as *programming by demonstration* [13, 34], focuses on learning macro-operators for later invocation by the user to let him accomplish things in fewer steps.

For example, one might implement an interactive tool for network configuration that proposes, one step at a time, a few alternative components to incorporate or connections among them. The human user could select from among these recommendations or reject them all and select another option. Each such interaction would generate a training instance for use in learning how to configure a network, which would then be used on future interactions. One can imagine similar adaptive interfaces for network diagnosis and repair.

A closely related formulation of action learning, known as *behavioral cloning* [40], collects traces of a human acting in some domain, but does not offer advice or interact directly. Again, each choice the human makes is transformed into a training case for use by supervised learning. The main difference is that behavioral cloning aims to create autonomous agents for carrying out a sequential decision-making task, whereas learning apprentices and adaptive interfaces aim to produce intelligent

assistants. For example, a system could watch a human expert execute a sequence of commands in configuring a computer network, transform these into supervised training cases for learning which actions to select or estimating the value of available choices. However, one might also attempt to extract, from the same trace, recurring sets of actions for composition into macro-operators that would let one solve the same problem in fewer steps.

A somewhat different formulation involves the notion of learning from delayed reward, more commonly known as *reinforcement learning*. Here the agent typically carries out actions in the environment and receives some reward signal that indicates the desirability of the resulting states. However, because many steps may be necessary before the agent reaches a desirable state (e.g., reestablishing a service connection), the reward can be delayed. Research in the reinforcement learning framework falls into two main paradigms. One represents control policies indirectly in terms of *value functions* that map state descriptions and available actions onto expected values (i.e., expected total future reward) [20, 44]. This approach involves propagating rewards backward over action sequences to assign credit, and may invoke a regression method to learn a predictor for expected values. Another paradigm instead encodes control policies more directly as a mapping from state descriptions onto actions, with learning involving search through the space of such policies [46, 31].

One might apply either approach to learning policies for dynamic network routing [6]. The reward signal here might be based on the standard metrics for route performance. The system would try establishing different routes, each of which involves a number of decision-making steps, and learn routing policies based on the observed performance. Over time, the routes selected by the learned policy would change, giving improved behavior for the overall network.

Another formulation is closely related to reinforcement learning, but involves *learning from problem solving* and mental search [42], rather than from actions in the environment. Here the agent has some model of the effects of actions or the resources they require which it can use to carry out mental simulations of action sequences. However, there typically exist many possible sequences, which introduces the problem of search through a problem space. Such search can produce one or more sequences that solve the problem, but it can also generate dead ends, loops, and other undesirable outcomes. Both successes and failures provide material for learning, in that they distinguish between desirable and undesirable choices, or at least suggest relative desirability.

Research on learning from problem-solving traces occurs within three broad paradigms. Some work focuses on learning local search-control knowledge for selecting, rejecting, or preferring actions or states. This knowledge may be cast as control rules or some related symbolic representation, or it may be stated as a numeric evaluation function. The latter approach is closely related to methods for estimating value functions from delayed reward, which has occasionally been used for tasks like scheduling [47] and integrated circuit layout [5]. Another paradigm emphasizes the formation from solution paths of macro-operators that take larger steps through the problem space in order to reduce the effective depth of search. A third framework,

analogical problem solving, also stores large-scale structures, but utilizes them in a more flexible manner by adapting them to new problems.

For example, one might apply any of these approaches to tasks like network routing and configuration. Such an application would require some model of the effects that individual choices would produce, so that the agent can decide whether a given state is desirable before actually generating it in the world. Thus, the system would start with the ability to generate routes or configurations, but it might do this very inefficiently if the search space is large. After repeated attempts at routing or configuration, it would acquire heuristic knowledge about how to direct its search, letting it produce future solutions much more efficiently without loss in quality.

A final formulation involves the *empirical optimization* of a complex system. Consider the problem of adjusting a chemical plant's parameters to improve its performance (e.g., reduce energy consumption, reduce waste products, increase product quality, increase rate of production, and so forth). If a predictive model of the plant is not available, the only recourse may be to try various settings of the parameters and see how the plant responds.

One example of this idea, *response surface methodology* ([33]) attempts to find the optimal operating point of a system by measuring system behavior at various points. The classic method designs and executes an experiment (e.g., some form of factorial design) about the current operating point and fits the results with a quadratic function to estimate the local shape of the objective function surface. Then it chooses a new operating point at the optimum of that quadratic surface and repeats the process.

Machine learning researchers have studied methods that make weaker assumptions and require fewer training examples. One approach [30] employs regression to analyze the results of previous experiments and determine a region of interest in which the objective function can be approximated well, then chooses a new test point that is distant from other test points while still lying within this region. An alternative approach [2] is more appropriate for searching discrete parameter spaces such as those that arise in network configuration. Given a set of parameter settings (configurations) for which the performance has been measured, one fits a probability distribution to predict where additional "good" points are located, then samples a new set of configurations according to that distribution, measures their performance, and continues until convergence.

Before closing, it is worth making two other points about learning for action selection and planning. First, in many domains, sensing requires active invocation, so that one can view it as a kind of action. Thus, an agent can learn policies for sensing, say to support efficient network diagnosis, just as it can for effectors, such as closing down a link in response to a suspected attack. Second, some methods for plan learning assume the availability of action models that describe the expected effects when actions are invoked, which leads in turn to the task of learning such action models from observations. This has many similarities to the problem of classification and regression learning, but aims to support higher-level learning about policies for acting and planning.

1.2.3 Learning for Interpretation and Understanding

A third formulation focuses on learning knowledge that lets one interpret and understand situations or events. Classification can be seen as a simple example of this idea, since one can 'understand' an instance as being an example of some class. However, more sophisticated approaches attempt to interpret observations in a more constructive manner, by combining a number of separate knowledge elements to explain them. The key difference is that classification and regression are content with models that make accurate predictions, whereas interpretive approaches require models that explain the data in terms of deeper structures. This process of explanation generation is often referred to as *abduction*.

The explanatory or abductive approach is perhaps most easily demonstrated in natural language processing, where a common performance task involves parsing sentences using a context-free grammar or some related formalism. Such a grammar contains rewrite rules that refer to nonterminal symbols for types of phrases and parts of speech, and a parse tree specifies how one can derive or explain a sentence in terms of these rules. One can apply similar ideas to other domains, including the interpretation and diagnosis of network behavior. For example, given anomalous data about the transfer rates between various nodes in a network, one might explain these observations using known processes, such as demand for a new movie that is available at one site and desired by others.

One can state a number of different learning tasks within the explanatory framework. The most tractable problem assumes that each training case comes with an associated explanation cast in terms of domain knowledge. This formulation is used commonly within the natural language community, where the advent of 'tree banks' has made available large corpora of sentences with their associated parse trees. The learning task involves generalizing over the training instances to produce a model that can be used to interpret or explain future test cases. Naturally, this approach places a burden on the developer, since it requires hand construction of explanations for each training case, but it greatly constrains the learning process, as it effectively decomposes the task into a set of separate classification or density estimation tasks, one for each component of the domain knowledge.

A second class of learning task assumes that training instances do not have associated explanations, but provides background knowledge from which the learner can construct them. This problem provides less supervision than the first, since the learner must consider alternative explanations for each training case and decide which ones are appropriate. However, the result is again some model that can be applied to interpret or explain future instances. This formulation is less burdesome on the developer, since he need not provide explanations for each training case, but only a domain theory from which the learner can construct them itself. Flann and Dietterich [18] have referred to this learning task as *induction over explanations*, but it is also closely related to some work on *constructive induction* [16] and *explanation-based generalization* [15].

A final variant on learning for understanding provides training cases with neither explanations nor background knowledge from which to construct them. Rather, the

learner must induce its own explanatory structures from regularities in the data, which it can then utilize to intepret and understand new test instances. An example from natural language involves the induction of context-free grammars, including both nonterminal symbols and the rewrite rules in which they occur, from legal training sentences [43]. Clearly, this task requires even less effort on the developer's part, but places a greater challenge on the learning system. This approach has gone by a variety of names in the machine learning literature, including *term generation*, *representation change*, and *constructive induction* (though this phrase has also been used for the second task).

Because learning tasks that produce explanatory models are generally more difficult than those for classification and regression, some researchers have formulated more tractable versions of them. One variant assumes the qualitative structure of the explanatory model is given and that learning involves estimating numeric parameters from the data. Examples of this approach include determining the probabilities in a stochastic context-free grammar, tuning the parameters in sets of differential equations, and inferring conditional probabilities in a Bayesian network. Another variation, known as *theory revision*, assumes an initial explanatory model that is approximately correct and utilizes training data to alter its qualitative structure [35]. Examples include revising Horn clause programs from classified training cases, improving sets of equations from quantitative data, and altering grammars in response to training sentences.

1.2.4 Summary of Problem Formulations

In summary, one can formulate machine learning tasks in a variety of ways. These differ in both the manner in which learned knowledge is utilized and, at a finer level, in the nature of the training data that drives the learning process. Table 1.1 summarizes the main formulations that have been discussed in this section. However, it is important to realize that different paradigms have received different degrees of attention within the machine learning community. Supervised approaches to classification and regression have been the most widely studied by far, with reinforcement learning being the second most common. Yet their popularity in the mainstream community does not imply they are the best ways to approach problems in computer networking, and research on the Knowledge Plane should consider all the available options.

Another important point is that one can often formulate a given real-world problem as a number of quite different learning tasks. For example, one might cast diagnosis of network faults as a classification problem that involves assigning the current network state to either a normal condition or one of a few prespecified faulty conditions. However, one could instead formulate it as a problem of understanding anomalous network behavior, say in terms of unobservable processes that, taken together, can explain recent statistics. Yet another option would be to state diagnosis as a problem of selecting active sensors that narrow down alternatives. Each formulation suggests different approaches to the diagnostic task, to learning knowledge in support of that task, and to criteria for evaluating the success of the learning component.

Table 1.1 Summary of Machine Learning Problem Formulations

Formulation	Performance Task
Classification & Regression	predict y given x predict rest of x given part of x predict $P(x)$ given x
Acting & Planning	iteratively choose action a in state s choose actions $\langle a_1, \ldots, a_n \rangle$ to achieve goal g find setting s to optimize objective $J(s)$
Interpretation & Understanding	parse data stream into tree structure of objects or events

1.3 TASKS IN COGNITIVE NETWORKING

The vision for the Knowledge Plane [9, 36, 10] describes a number of novel capabilities for computer networks. This section reviews three capabilities that the vision assumes in terms of the cognitive functionalities that are required. These include anomaly detection and fault diagnosis, responding to intruders and worms, and rapid configuration of networks.

1.3.1 Anomaly Detection and Fault Diagnosis

Current computer networks require human managers to oversee their behavior and ensure that they deliver the services desired. To this end, the network managers must detect unusual or undesirable behaviors, isolate their sources, diagnose the fault, and repair the problem. These tasks are made more challenging because large-scale networks are managed in a distributed manner, with individuals having access to information about, and control over, only portions of the system. Nevertheless, it will be useful to examine the activities in which a single network manager engages.

The first activity, *anomaly detection*, involves the realization that something unusual or undesirable is transpiring within the network. One possible approach to this problem, which applies recent advances in Bayesian networks, is to formulate it as a density estimation problem. Individual components, larger regions of the network, or, at some level, the entire internet could be modeled as the joint probability distribution of various quantities (queue lengths, traffic types, round-trip-times, and so on). An anomaly is defined as a low probability state of the network.

Another possible approach is sometimes called one-class learning or learning a characteristic description of a class. A classifier can be learned that attempts to find a compact description that covers a target percentile (e.g., 95%) of the "normal" traffic. Anything classified as "negative" by this classifier can then be regarded as an anomaly.

There are several issues that arise in anomaly detection. First, one must choose the level of analysis and the variables to monitor for anomalies. This may involve first applying methods for interpreting and summarizing sensor data. In the Knowledge Plane, one can imagine having whole hierarchies of anomaly detectors looking for changes in the type of network traffic (e.g., by protocol type), in routing, in traffic delays, in packet losses, in transmission errors, and so on. Anomalies may be undetectable at one level of abstraction but easy to detect at a different level. For example, a worm might escape detection at the level of a single host, but be detectable when observations from several hosts are combined.

The second issue is the problem of false alarms and repeated alarms. Certain kinds of anomalies may be unimportant, so network managers need ways of training the system to filter them out. Supervised learning methods could be applied to this problem.

The second activity, *fault isolation*, requires the manager to identify the locus of an anomaly or fault within the network. For example, if a certain route has an especially heavy load, this may be due to changes at a single site along that route rather than to others. Hence, whereas anomaly detection can be performed locally (e.g., at each router), fault isolation requires the more global capabilities of the Knowledge Plane to determine the scope and extent of the anomaly.

The activity of *diagnosis* involves drawing some conclusions about the cause of the anomalous behavior. Typically, this follows fault isolation, although in principle one might infer the presence of a specific problem without knowing its precise location. Diagnosis may involve the recognition of some known problems, say one the network manager has encountered before, or the characterization of a new problem that may involve familiar components.

One can apply supervised learning methods to let a network manager teach the system how to recognize known problems. This could be a prelude to automatically solving them, as discussed below.

Both fault isolation and diagnosis may require active measurements to gather information. For example, an anomaly found at a high level of aggregation would typically require making more detailed observations at finer levels of detail to understand the cause. In the "Why?" scenario, one can imagine active probes of both the local computer (e.g., its configuration) and the internet (e.g., "pings" to see if the destination is reachable and up). Diagnosis usually must balance the cost of gathering information against the potential informativeness of the action. For example, if the ping succeeds, it requires little time, but otherwise it can take much longer to time out. If the goal is to diagnose the problem as quickly as possible, then ping might be a costly action to perform.[1]

Fault isolation and diagnosis also typically require models of the structure of the system under diagnosis. Much recent effort in network research has sought to provide better ways of understanding and visualizing the structure of the internet.

[1]Recent work in an area known as "cost-sensitive learning" addresses this tradeoff between cost and informativeness.

Machine learning for interpretation could be applied to help automate this process. The resulting structural and behavioral models could then be used by model-based reasoning methods to perform fault isolation and diagnosis.

Once a network manager has diagnosed a problem, he is in a position to repair it. However, there may exist different courses of action that would eliminate the problem, which have different costs and benefits. Moreover, when multiple managers are involved in the decision, different criteria may come into play that lead to negotiation. Selecting a repair strategy requires knowledge of available actions, their effects on network behavior, and the tradeoffs they involve.

Supervised learning methods could be applied to learn the effects of various repair actions. Methods for learning in planning could be applied to learn repair strategies (or perhaps only to evaluate repair strategies suggested by a human manager). There may be some opportunity here for "collaborative filtering" methods that would provide an easy way for managers to share repair strategies.

As stated, the 'Why' problem [9, 36] requires diagnosis of an isolated fault, but one can imagine variations that involve answering questions about anomalies, fault locations, and actions taken to repair the system. Each of these also assumes some interface that lets the user pose a specific question in natural language or, more likely, in a constrained query language. Defining the space of Why questions the Knowledge Plane should support is an important research task.

1.3.2 Responding to Intruders and Worms

Responding to intruders (human, artificial, or their combination) and keeping networks and applications safe encompass a collection of tasks that are best explained depending on the time at which the network manager performs them. We can group them into tasks that occur before, during, or after the occurrence of an intrusion, as the temporal model in Figure 1.2 depicts.

`dietterich-langley-mdr.eps` goes here

Fig. 1.2 Time axis model of incident prevention, detection, and response tasks.

Prevention Tasks. Network managers try to minimize the likeliness of future intrusions by constantly auditing the system and eliminating threats beforehand. A network manager proactively performs security audits testing the computer systems for weaknesses—vulnerabilities or exposures. However, scan tools (e.g., Nessus, Satan, and Oval) used for penetration or vulnerability testing only recognize a limited number of vulnerabilities given the ever increasing frequency of newly detected possibilities for breaking into a computer system or disturbing its normal operation. Thus, network managers continually update scan tools with new plug-ins that permit them to measure new vulnerabilities. Once the existence of a vulnerability or exposure is perceived, network managers assess the convenience of discontinuing the service or application affected until the corresponding patch or intrusion detection

signature is available. A tradeoff between risk level and service level is made in every assessment.

Network managers aim at shrinking the *window of vulnerability*, the time gap between when a new vulnerability or exposure is discovered and a preventative solution (patch, new configuration, etc.) is provided. A basic strategy to accomplish that objective is based on two conservative tasks: first, minimizing the number of exposures (i.e., disable unnecessary or optional services by configuring firewalls to allow only the use of ports that are necessary for the site to function) and, second, increasing awareness of new vulnerabilities and exposures (e.g., the subscription model that Partridge discusses with relation to worms).

Finally, network managers continuously monitor the system so that pre-intrusion behavioral patterns can be understood and used for further reference when an intrusion occurs. *Monitoring* is an ongoing, preventive task.

Detection Tasks. The sooner an intrusion is detected, the more chances there are for impeding an unauthorized use or misuse of the computer system. Network managers monitor computer activities at different levels of detail: system call traces, operating system logs, audit trail records, resource usage, network connections, etc. They constantly try to fuse and correlate real-time reports and alerts stemming from different security devices (e.g., firewalls and intrusion detection systems) to stop suspicious activity before it has a negative impact (i.e., degrading or disrupting operations). Different sources of evidence are valuable given the evolving capabilities of intruders to elude security devices. The degree of suspicion and malignancy associated with each report or alert still requires continuous human oversight. Consequently, network managers are continually overwhelmed with a vast amount of log information and bombarded with countless alerts. To deal with this onslaught, network managers often tune security devices to reduce the number of false alerts even though this increases the risk of not detecting real intrusions.

The time at which an intrusion is detected directly affects the level of damage that an intrusion causes. An objective of network managers is to reduce the *window of penetrability*, the time span that starts when a computer system has been broken into and extends until the damage has been completely repaired. The correct diagnosis of an intrusion allows a network manager to initiate the most convenient response. However, a tradeoff between quality and rapidness is made in every diagnostic.

Response and Recovery Tasks. As soon as a diagnostic on an intrusion is available, network managers initiate a considered response. This response tries to minimize the impact on the operations (e.g., do not close all ports in a firewall if only blocking one IP address is enough). Network managers try to narrow the *window of compromisibility* of each intrusion—the time gap that starts when an intrusion has been detected and ends when the proper response has taken effect—deploying automatic intrusion response systems. Nevertheless, these systems are still at an early stage and even fail at providing assistance in manual responses. Therefore, network managers employ a collection of ad-hoc operating procedures that indicate how to respond and recover from a type of intrusion. The responses to an attack range from terminating a user job

or suspending a session to blocking an IP address or disconnecting from the network to disable the compromised service or host. Damage recovery or *repairing* often requires maintaining the level of service while the system is being repaired, which makes this process difficult to automate.

Once the system in completely recovered from an intrusion, network managers collect all possible data to thoroughly analyze the intrusion, trace back what happened, and evaluate the damage. Thus, system logs are continuously backed up. The goal of *post-mortem analysis* is twofold. On the one hand, it gathers forensic evidence (contemplating different legal requirements) that will support legal investigations and prosecution and, on the other hand, it compiles experience and provides documentation and procedures that will facilitate the recognition and repelling of similar intrusions in the future.

Ideally, the ultimate goal of a network manager is to make the three windows (vulnerability, penetrability, and compromisibility) of each possible intrusion converge into a single point in time. Tasks for responding to intruders (human, artificial or a combination of both) should not differ significantly from those tasks needed to recover from non-malicious errors or failures (Sections 1.3.1 and 1.3.2).

1.3.3 Network Configuration and Optimization

Network configuration and optimization can be viewed as an instance of the general problem of designing and configuring a system. In this section, we review the space of configuration problems and briefly describe the methods that have been developed in AI and machine learning to solve these problems.

A Spectrum of Configuration Tasks. The problem of the design and configuration of engineered systems has been studied in artificial intelligence since the earliest days [45]. Configuration is generally defined as a form of routine design from a given set of components or types of components (i.e., as opposed to designing the components themselves). As such, there is a spectrum of configuration problems of increasing difficulty, as shown in Table 1.2.

The simplest task is *parameter selection*, where values are chosen for a set of global parameters in order to optimize some global objective function. Two classic examples are the task of setting the temperature, cycle time, pressure, and input/output flows of a chemical reactor and the task of controlling the rate of cars entering a freeway and the direction of flow of the express lanes. If a model of the system is known, this becomes purely an optimization problem, and many algorithms have been developed in operations research, numerical analysis, and computer science to solve such problems.

The second task is *compatible parameter selection*. Here, the system consists of a set of components that interact with one another to achieve overall system function according to a fixed topology of connections. The effectiveness of the interactions is influenced by parameter settings which must be compatible in order for sets of components to interact. For example, a set of hosts on a subnet must agree on the network addresses and subnet mask in order to communicate using IP. Global system

Table 1.2 Configuration tasks in increasing order of complexity

Problem:	Global parameters	Local parameters	Topology	Components
Global Parameter Configuration	XX			
Compatible Parameter Configuration	XX	XX		
Topological Configuration	XX	XX	XX	
Component Selection and Configuration	XX	XX	XX	XX

performance can depend in complex ways on local configuration parameters. Of course, there may also be global parameters to select as well, such as the protocol family to use.

The third task is *topological configuration*. Here, the system consists of a set of components, but the topology must be determined. For example, given a set of hosts, gateways, file servers, printers, and backup devices, how should the network be configured to optimize overall performance? Of course, each proposed topology must be optimized through compatible parameter selection.

Finally, the most general task is *component selection and configuration*. Initially, the configuration engine is given a catalog of available types of components (typically along with prices), and it must choose the types and quantities of components to create the network (and then, of course, solve the Topological Configuration problem of arranging these components).

The Reconfiguration Process. The discussion thus far has dealt only with the problem of choosing a configuration. However, a second aspect of configuration is determining how to implement the configuration efficiently. When a new computer network is being installed (e.g., at a trade show), the usual approach is to install the gateways and routers; then the file and print servers; and finally individual hosts, network access points, and the like. The reason for this is that this order makes it easy to test and configure each component and it minimizes the amount of re-work. Automatic configuration tools (e.g., DHCP) can configure the individual hosts if the servers are in place first.

A different challenge arises when attempting to change the configuration of an existing network, especially if the goal is to move to the new configuration without significant service interruptions. Most configuration steps require first determining the current network configuration, and then planning a sequence of reconfiguration actions and tests to move the system to its new configuration. Some steps may cause network partitions that prevent further (remote) configuration. Some steps must be performed without knowing the current configuration (e.g., because there is already a network partition, congestion problem, or attack).

We now review some of the existing work on configuration within the artificial intelligence and machine learning communities.

Parameter Selection. As we discussed above, parameter selection becomes optimization (possibly difficult, non-linear optimization) if the model of the system is known. Statisticians have studied the problem of empirical optimization in which no system model is available.

Compatible Parameter Configuration. The standard AI model of compatible parameter configuration is known as the *constraint satisfaction problem* (CSP). This consists of a graph where each vertex is a variable that can take values from set of possible values and each edge encodes a pair-wise constraint between the values of the variables that it joins. A large family of algorithms have been developed for finding solutions to CSPs efficiently [22, 14]. In addition, it is possible to convert CSPs into Boolean satisfiability problems, and very successful randomized search algorithms, such as WalkSAT [41], have been developed to solve these problems.

The standard CSP has a fixed graph structure, but this can be extended to include a space of possible graphs and to permit continuous (e.g., linear algebraic) constraints. The field of *constraint logic programming* (CLP) [19] has developed programming languages based on ideas from logic programming that have a constraint solver integrated as part of the run-time system. The logic program execution can be viewed as conditionally expanding the constraint graph, which is then solved by the constraint system. Constraint logic programming systems have been used to specify and solve many kinds of configuration problems.

To our knowledge, there has been no work on applying machine learning to help solve compatible parameter configuration problems. There is a simple form of learning that has been applied to CSPs called "nogood learning", but it is just a form of caching to avoid wasting effort during CSP search. There are many potential learning problems, including learning about the constraints relating pairs of variables and learning how to generalize CSP solutions across similar problems.

Topological Configuration. Two principal approaches have been pursued for topological configuration problems: refinement and repair. Refinement methods start with a single "box" that represents the entire system to be configured. The box has an attached formal specification of its desired behavior. Refinement rules analyze the formal specification and replace the single box with two or more new boxes with specified connections. For example, a small office network might initially be specified as a box that connects a set of workstations, a file server, and two printers to a DSL line. A refinement rule might replace this box with a local network (represented as a single box connected to the various workstations and servers) and a router/NAT box. A second refinement rule might then refine the network into a wireless access point and a set of wireless cards (or alternatively, into an ethernet switch and a set of ethernet cards and cables). There has been some work on applying machine learning to learn refinement rules in the domain of VLSI design [29].

The repair-based approach to topological configuration starts with an initial configuration (which typically does not meet the required specifications) and then makes repairs to transform the configuration until it meets the specifications. For example, an initial configuration might just connect all computers, printers, and other devices

to a single ethernet switch, but this switch might be very large and expensive. A repair rule might replace the switch with a tree of smaller, cheaper switches. Repair-based approaches make sense when the mismatch between specifications and the current configuration can be traced to *local* constraint violations. A repair rule can be written that "knows how" to repair each kind of violation. Repair-based methods have been very successful in solving scheduling problems [48].

Machine learning approaches to repair-based configuration seek to learn a heuristic function $h(x)$ that estimates the quality of the best solution reachable from configuration x by applying repair operators. If h has been learned correctly, then a hill climbing search that chooses the repair giving the biggest improvement in h will lead us to the global optimum. One method for learning h is to apply reinforcement learning techniques. Zhang and Dietterich [47] describe a method for learning heuristics for optimizing space shuttle payload scheduling; Boyan and Moore [5] present algorithms that learn heuristics for configuring the functional blocks on integrated circuit chips.

In both refinement and repair-based methods, constraint satisfaction methods are typically applied to determine good parameter values for the current proposed configuration. If no satisfactory parameter values can be found, then a proposed refinement or repair cannot be applied, and some other refinement or repair operator must be tried. It is possible for the process to reach a dead end, which requires backtracking to some previous point or restarting the search.

Component Selection and Configuration. The refinement and repair-based methods described above can also be extended to handle component selection and configuration. Indeed, our local network configuration example shows how refinement rules can propose components to include in the configuration. Similar effects can be produced by repair operators.

Changing Operating Conditions. The methods discussed so far only deal with the problem of optimizing a configuration under fixed operating conditions. However, in many applications, including networking, the optimal configuration may need to change as a result of changes in the mix of traffic and the set of components in the network. This raises the issue of how data points collected under one operating condition (e.g., one traffic mix) and be used to help optimize performance under a different operating condition. To our knowledge, there is no research on this question.

1.4 OPEN ISSUES AND RESEARCH CHALLENGES

Most research in the field of machine learning has been motivated by problems in pattern recognition, robotics, medical diagnosis, marketing, and related commercial areas. This accounts for the predominance of supervised classification and reinforcement learning in current research. The networking domain requires several shifts in focus and raises several exciting new research challenges, which we discuss in this section.

1.4.1 From Supervised to Autonomous Learning

As we have seen above, the dominant problem formulation in machine learning is supervised learning, where a "teacher" labels the training data to indicate the desired response. While there are some potential applications of supervised learning in Knowledge Plane applications (e.g., for recognizing known networking misconfigurations and intrusions), there are many more applications for autonomous learning that does not require a teacher. In particular, many of the networking applications involve looking for anomalies in real-time data streams, which can be formulated as a combination of unsupervised learning and learning for interpretation.

Anomaly detection has been studied in machine learning, but usually it has considered only a fixed level of abstraction. For networking, there can be anomalies at the level of individual packets, but also at the level of connections, protocols, traffic flows, and network-wide disturbances. A very interesting challenge for machine learning is to develop methods that can carry out simultaneous unsupervised learning at all of these levels of abstraction. At very fine levels of detail, network traffic is continually changing, and therefore, is continually novel. The purpose of introducing levels of abstraction is to hide unimportant variation while exposing important variation.

Anomaly detection at multiple levels of abstraction can exploit regularities at these multiple levels to ensure that the anomaly is real. A similar idea—multi-scale analysis—has been exploited in computer vision, where it is reasonable to assume that a real pattern will be observable at multiple levels of abstraction. This helps reduce false alarms.

1.4.2 From Off-Line to On-Line Learning

Most applications of machine learning involve off-line approaches, where data is collected, labeled, and then provided to the learning algorithm in a batch process. Knowledge Plane applications involve the analysis of real-time data streams, and this poses new challenges and opportunities for learning algorithms.

In the batch framework, the central constraint is usually the limited amount of training data. In contrast, in the data stream setting, new data is available at every time instant, so this problem is less critical. (Nonetheless, even in a large data stream, there may be relatively few examples of a particular phenomenon of interest, so the problem of sparse training data is not completely eliminated.)

Moreover, the batch framework assumes that the learning algorithm has essentially unlimited amounts of computing time to search through the space of possible knowledge structures. In the on-line setting, the algorithm can afford only a fixed and limited amount of time to analyze each data point.

Finally, in the batch framework, the criterion to be minimized is the probability of error on new data points. In the on-line framework, it makes more sense to consider the response time of the system. How many data points does it need to observe before it detects the relevant patterns? This can be reformulated as a mistake-bounded criterion: how many mistakes does the system make before it learns to recognize the pattern?

1.4.3 From Fixed to Changing Environments

Virtually all machine learning research assumes that the training sample is drawn from a stationary data source—the distribution of data points and the phenomena to be learned are not changing with time. This is not true in the networking case. Indeed, the amount of traffic and the structure of the network are changing continuously. The amount of traffic continues to rise exponentially, and new autonomous systems are added to the internet almost every day. New networking applications (including worms and viruses) are introduced frequently.

An additional challenge is that while some of the changes in the networking environment result simply from new applications and traffic growth, other changes are driven by adversaries who are trying to elude existing intrusion detection mechanisms. This calls for new approaches to machine learning that explicitly consider the game-theoretic aspects of the problem.

Research in machine learning needs to formalize new criteria for evaluating learning systems in order to measure success in these changing environments. A major challenge is to evaluate anomaly detection systems, because by definition they are looking for events that have never been seen before. Hence, they cannot be evaluated on a fixed set of data points, and measures are needed to quantify the degree of novelty of new observations.

1.4.4 From Centralized to Distributed Learning

Another important way in which Knowledge Plane applications differ from traditional machine learning problems is that, in the latter, it has usually been possible to collect all of the training data on a single machine and run the learning algorithm over that data collection. In contrast, a central aspect of the Knowledge Plane is that it is a distributed system of sensors, anomaly detectors, diagnostic engines, and self-configuring components.

This raises a whole host of research issues. First, individual anomaly detectors can form models of their local traffic, but they would benefit from traffic models learned elsewhere in the Knowledge Plane. This would help them detect a new event the first time they see it, rather than having to be exposed multiple times before the event pattern emerges.

Second, some events are inherently distributed patterns of activity that cannot be detected at an individual network node. The research challenge here is to determine what kinds of statistics can be collected at the local level and pooled at the regional or global level to detect these patterns. This may involve a bi-directional process of information exchange in which local components report summary statistics to larger-scale “think points”. These think points detect a possible pattern that requires additional data to verify. So they need to request the local components to gather additional statistics. Managing this bi-directional statistical reasoning is an entirely new topic for machine learning research.

1.4.5 From Engineered to Constructed Representations

An important ingredient in the success of existing learning systems is the careful engineering of the attributes describing the training data. This "feature engineering" process is not well understood, but it involves combining background knowledge of the application domain with knowledge about learning algorithms. To illustrate this, consider a very simple example in networking arises in intrusion detection: Rather than describing network traffic using absolute IP addresses, it is better to describe packets according to whether they share the same or different IP addresses. This ensures that the learned intrusion detector is not specific to a single IP address but instead looks for patterns among a set of packets sharing a common address, regardless of the absolute value of the address.

A critical challenge for machine learning is to develop more automatic ways of constructing the representations given to the learning algorithms. This requires making explicit the design principles currently used by human data analysts.

1.4.6 From Knowledge-Lean to Knowledge-Rich Learning

An important factor influencing the development of machine learning has been the relative cost of gathering training data versus building knowledge bases. The constructing and debugging of knowledge bases is a difficult and time-consuming process, and the resulting knowledge bases are expensive to maintain. In contrast, there are many applications where training data can be gathered fairly cheaply. This is why speech recognition and optical character recognition systems have been constructed primarily from training data. Any literate adult human is an expert in speech recognition and optical character recognition, so it is easy for them to label data points to training a learning system.

There are other domains (including networking), where there are very few experts available, and their time is perhaps better employed in developing formal representations of the knowledge they possess about network architectures and configurations. This is particularly true in the area of network diagnosis and configuration, where experts can help construct models of network components and prescribe rules for correct configuration. This raises the challenge of how to combine training data with human-provided models and rules. This should become an important goal for future machine learning research.

1.4.7 From Direct to Declarative Models

Most machine learning systems seek to induce a function that maps directly from inputs to outputs and therefore requires little inference at run time. In an optical character recognition system, for example, the learned recognizer takes a character image as input and produces the character name as output without any run-time inference. We will call this "direct knowledge", because the learned knowledge performs the task directly.

However, as applications become more complex, a simple view of the performance element as a classifier (or direct decision maker) is no longer adequate. Diagnosis and configuration tasks require a more complex performance element that makes a sequence of interacting decisions at run time. These performance elements typically require declarative knowledge such as "the probability that a misconfigured gateway will exhibit symptom X is P" or "it is illegal to simultaneously select configuration options Y and Z." An important goal for machine learning is to learn these forms of declarative knowledge (i.e., knowledge that makes minimal assumptions about how it will be used by the performance element).

Declarative knowledge is easier for people to understand, and it can be more easily combined with human-provided knowledge as well. Hence, acquiring declarative knowledge is an important challenge for machine learning in the context of the Knowledge Plane.

1.5 CHALLENGES IN METHODOLOGY AND EVALUATION

Machine learning research has a long history of experimental evaluation, with some examples dating back to the 1960s, well before the field was a recognized entity. However, the modern experimental movement began in the late 1980s, when researchers realized the need for systematic comparisons (e.g., [21]) and launched the first data repository. Other approaches to evaluation, including formal analysis and comparison to human behavior, are still practiced, but, over the past decade, experimentation has come to dominate the literature on machine learning, and we will focus on that approach in our discussions of cognitive networking.

Experimentation involves the systematic variation of independent factors to understand their impact on dependent variables that describe behavior. Naturally, which dependent measures are most appropriate depends on the problem being studied. For fault diagnosis, these might involve the system's ability to infer the correct qualitative diagnosis, its ability to explain future network behaviors, and the time taken to detect and diagnose problems. Similar measures seem appropriate for responding to intruders and worms, though these might also include the speed and effectiveness of response. For studies of configuration, the dependent variables might concern the time taken to configure a new system and the resulting quality, which may itself require additional metrics. Similarly, routing studies would focus on the efficiency and effectiveness of the selected routes.

Note that these behavioral measures have nothing directly to do with learning; they are same measures one would use to evaluate a nonlearning system and even the abilities of a human network manager. Because learning is defined as improvement in performance, we can only measure the effectiveness of learning in terms of the performance it aims to improve. Note also that the metrics mentioned above are quite vague, and they must be made operational before they can be used in experimental evaluations. In doing so, it may seem natural to use variables associated with one's selected formulation of the learning problem, such as predictive accuracy for classification or received reward for action selection. However, we should resist this

temptation and instead utilize variables that measure directly what is desired from a networking perspective.

An experimental study also requires the variation of one or more independent factors to determine their effects on behavior. In general, these can deal with

- the effects of experience, such as the number of observations available to the learning system;
- the effects of data characteristics, such as the degree of noise or percentage of features missing;
- the effects of task characteristics, such as the complexity of a configuration problem or the number of simultaneous faults;
- the effects of system characteristics, such as the inclusion of specific learning modules or sensitivity to parameter settings; and
- the effects of background knowledge, such as information about network structure and bandwidth.

Again, which variables are appropriate will depend largely on the networking problem at hand and the specific learning methods being used. However, a full understanding of how machine learning can assist cognitive networking will require studies that examine each of the dimensions above.

Of course, one cannot carry out experiments in the abstract. They require specific domains and problems that arise within them. To study the role of learning in network management, we need a number of testbeds that can foster the experimental evaluation of alternative approaches to learning. At least some of these should involve actual networks, to ensure the collection of realistic data for training and testing the learning methods. However, these should be complemented with simulated networks, which have the advantage of letting one systematically vary characteristics of the performance task, the learning task, and the available data. Langley [24] has argued that experiments with both natural and synthetic data are essential, since the former ensures relevance and the latter lets one infer source of power and underlying causes.

Much of the success of the last 15 years of machine learning research can be traced to the establishment of a collection of data sets at the University of California, Irvine [32, 3, 27]. The UCI data sets provided a common set of problems on which to evaluate learning algorithms and greatly encouraged comparative studies. The data sets span a wide range of application problems ranging from basic science and medicine to optical character recognition and speech recognition.

Ideally, we want an analog of this repository to enable the careful evaluation of machine learning in networking domains. However, because the Knowledge Plane envisions an adaptive network that learns about itself over time, it is important that this resource not be limited to static data sets, but also include simulated networks that allow learning methods, and their associated performance elements, to interact with the network environment in an on-line manner.

1.6 SUMMARY

In this chapter, we have reviewed the current state of research in machine learning, and highlighted those aspects that are relevant to the emerging vision of the Knowledge Plane [9, 36, 10]. We began by discussing the major problem formulations in machine learning: learning for classification and regression, learning for acting and planning to act, and learning for interpretation and understanding. We then discussed the various tasks that are raised by the Knowledge Plane, and examined how each of these tasks could be mapped to existing problem formulations. In some cases, there is a direct mapping, but in other cases, such as complex configuration and diagnosis, there is hardly any existing work in machine learning.

Furthermore, even when networking problems fit well into existing formulations, this does not mean that existing algorithms can be applied directly. This is because many aspects of computer networking impose new kinds of requirements that are not met by existing algorithms. We reviewed these requirements in the third section of the chapter, where we discussed the need for learning that is autonomous, on-line, distributed, knowledge-rich, and as well as able to deal with changing environments and to represent learned knowledge declaratively. Finally, we discussed the need for clear articulation of performance criteria for evaluating learning systems and the importance of creating simulation environments to support controlled experimentation.

Computer networking in general, and the Knowledge Plane vision in particular, pose enormous challenges for machine learning research. Conversely, existing algorithms and tools from machine learning have an important role to play in making the Knowledge Plane vision a reality. We can look forward to many exciting research advances in the coming years.

Acknowledgments

The authors thank Christopher Ramming for asking the hard questions that led us to write this chapter and Francisco Martin for providing material on existing practices in intrusion prevention, detection, and repair. The authors gratefully acknowledge the support of the Defense Advanced Research Projects Agency under AFRL Contract F30602-03-2-0191 (to Oregon State University) and Grant F30602-03-2-0198 (to the Institute for the Study of Learning and Expertise). Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government, of AFRL, or of DARPA.

REFERENCES

1. D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37, 1991.

2. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *The Proceedings of the 12th Annual Conference on Machine Learning*, pages 38–46, San Francisco, CA, 1995. Morgan Kaufmann.

3. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

4. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. 11th Annu. Conf. on Comput. Learning Theory*, pages 92–100. ACM Press, New York, NY, 1998.

5. J. Boyan and A. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

6. J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 671–678. Morgan Kaufmann Publishers, Inc., 1994.

7. W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8:195–210, 1996.

8. P Cheeseman, M. Self, J. Kelly, W. Taylor, D. Freeman, and J. Stutz. Bayesian classification. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 607–611, Cambridge, MA, 1988. AAAI Press/MIT Press.

9. D. Clark. A new vision for network architecture. Technical report, MIT, 2002.

10. D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A knowledge plane for the internet. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10, New York, NY, USA, 2003. ACM Press.

11. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261, 1988.

12. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines (and other kernel-based learning methods)*. Cambridge University Press, 2000.

13. A. Cypher. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, 1993.

14. R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, 2003.

15. G. DeJong. Toward robust real-world inference: A new perspective on explanation-based learning. In *Machine Learning: ECML 2006; Lecture Notes in Computer Science*, volume 4212, pages 102–113, Berlin, 2006. Springer Verlag.

16. G. Drastal, R. Meunier, and S. Raatz. Error correction in constructive induction. In A. Maria Segre, editor, *Proceedings of the Sixth International Workshop on*

Machine Learning (ML 1989), Cornell University, Ithaca, New York, USA, June 26-27, 1989, pages 81–83, San Francisco, 1989. Morgan Kaufmann.

17. D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139, 1987.

18. N. S. Flann and T. G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.

19. J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

20. L. Pack Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

21. D. Kibler and P. Langley. Machine learning as an experimental science. In *Proceedings of the Third European Working Session on Learning*, pages 81–92, Glasgow, 1988. Pittman.

22. V. Kumar. Algorithms for constraints satisfaction problems: A survey. *The AI Magazine*, 13(1):32–44, 1992.

23. P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, San Francisco, 1995.

24. P. Langley. Relevance and insight in experimental studies. *IEEE Expert*, pages 11–12, 1996.

25. P. Langley. User modeling in adaptive interfaces. In *Proceedings of the Seventh International Conference on User Modeling*, pages 357–370, New York, 1999. Springer.

26. P. Langley and H. A. Simon. Applications of machine learning and rule induction. *Communications of the ACM*, 38:55–64, 1995.

27. C. J. Merz and P. M. Murphy. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1996.

28. T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

29. T. M. Mitchell, S. Mahadevan, and L. I. Steinberg. LEAP: A learning apprentice for VLSI design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 573–580, Los Angeles, CA, 1985. Morgan Kaufmann.

30. A. Moore, J. Schneider, J. Boyan, and M. Soon Lee. Q2: Memory-based active learning for optimizing noisy continuous functions. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference of Machine Learning*, pages 386–394, 340 Pine Street, 6th Fl., San Francisco, CA 94104, 1998. Morgan Kaufmann.

31. D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

32. P.M. Murphy and D.W. Aha. UCI repository of machine learning databases [machine-readable data repository]. Technical report, University of California, Irvine, 1994.

33. R. H. Myers and D. C. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. John Wiley & Sons, Inc., New York, NY, 1995.

34. D. Oblinger, V. Castelli, and L. Bergman. Augmentation-based learning. In *IUI2006: 2006 International Conference on Intelligent User Interfaces*, pages 202–209, New York, 2006. ACM Press.

35. D. Ourston and R. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815–820. AAAI Press, 1990.

36. C. Partridge. Thoughts on the structure of the knowledge plane. Technical report, BBN, Cambridge, MA, 2003.

37. C. E. Priebe and D. J. Marchette. Adaptive mixture density estimation. *Pattern Recognition*, 26(5):771–785, 1993.

38. J. R. Quinlan. *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA, 1993.

39. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP research group., editors, *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations.* MIT Press, Cambridge, MA, 1986.

40. C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In D. Sleeman, editor, *Proceedings of the Ninth International Conference on Machine Learning*, pages 385–393, San Francisco, 1992. Morgan Kaufmann.

41. B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In Michael Trick and David S. Johnson, editors, *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, Providence RI, 1993.

42. D. Sleeman, P. Langley, and T. Mitchell. Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, 3:48–52, 1982.

43. A. Stolcke and S. Omohundro. Inducing probabilistic grammars by bayesian model merging. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical*

Inference and Applications: Proceedings of the Second International Colloquium on Grammatical Inference, pages 106–118. Springer Verlag, 1994.

44. R. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

45. F. M. Tonge. Summary of a heuristic line balancing procedure. In Edward A. Feigenbaum and Julian Feldman, editors, *Computers and Thought*, pages 168–190. AAAI Press/MIT Press, Menlo Park, CA, 1963.

46. R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229, 1992.

47. W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *1995 International Joint Conference on Artificial Intelligence*, pages 1114–1120. Morgan Kaufmann, San Francisco, CA, 1995.

48. M. Zweben, B. Daun, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, chapter 8, pages 241–255. Morgan Kaufmann, San Francisco, CA, 1994.