Adding Learning to Knowledge-Based Systems: Taking the "Artificial" Out of AI

Daniel L. Schmoldt
USDA Forest Service
Brooks Forest Products Center
Blacksburg, Virginia 24061-0503
schmoldt@vt.edu

ABSTRACT. Knowledge-based systems (KBS) development and maintenance both require time-consuming analysis of domain knowledge. Where example cases exist, KBS can be built, and later updated, by incorporating learning capabilities into their architecture. This applies to both supervised and unsupervised learning scenarios. In this paper, the important issues for learning systems—memory, feedback, pattern formulation, and pattern recognition—are described in terms of an instance vector set, a prototype vector set, and a mapping between those sets. While learning systems can possess robustness, recency, adaptability, and extensibility, they also require: careful attention to example case security, correct interpretation of feedback, modification for uncertainty calculations, and treatment of ambiguous output. Despite the difficulties associated with adding learning to KBS, it is essential for ridding them of artificiality.

Knowledge-based systems (KBS), as the hegemonic version of AI, have gained substantial notoriety, both as topics for research and as application tools for managers and decision-makers. The number of system development projects were few in number during the early 1980's; there were less than 50 in business and industry before 1985 (Harmon and Sawyer 1990) By 1992 their number grew to more than 200 in the areas of agriculture and the environment alone (Durkin 1993). Carrascal and Pau (1992) reviewed 110 applications in agriculture and food processing. Many systems are still of the stand-alone variety, but an ever larger number imbed knowledge-based methods and knowledge bases in other applications, e.g., spatial data systems (q.v., Loh et al. 1994, Power and Saarenmaa 1995, Reynolds et al. 1996). As with any marriage of different techniques, the expectation is that a merger can borrow from the best of both approaches to create something even more useful.

While KBS and methods have demonstrated usefulness, the presence of any real intelligence in any of these systems is questionable; these systems lack one of the seminal characteristics of intelligence—an ability to learn (Schank 1987). Being capable of learning—behavioral modification over time—results

in agents that are adaptable to new inputs and that are robust with respect to erroneous inputs. Likewise, learning is also central to the operation of the scientific process (McRoberts et al. 1991).

Typically, however, rigidity and fragility are characteristics of most current KBS. The current generation of systems can only deal with problem instances that are expressly encoded for in their knowledge bases and often fail catastrophically when unusual problems are encountered (Schmoldt and Rauscher 1996). As our best example of learning agents, human beings exhibit a great deal of flexibility in dealing with the vagaries posed by life's experiences. So, as the first decade of AI applications in natural resource management draws to a close, we need to critically examine whether newly developed KBS should be evolving toward a more anthropomorphic level of intelligence.

Certainly, some efforts have been made to develop learning systems (e.g. BACON and GLAUBER, Langley et al. 1987; RIFFLE, Matthews et al. 1995), but these systems were designed expressly for learning or discovery. I am arguing, instead, that traditional, application-oriented KBS should also contain some learning capability, what Julien et al. (1992) referred to as on-site learning or apprentice systems. Those authors described how this might be accomplished for environmental evaluation applications. Regardless of how easy it is to revisit a knowledge base periodically and bring it up to date (and the ease of this task is questionable), a self-learning system, once created, would require little manual attention to upgrade system knowledge and would possess robustness, elegance, and economy. A self-learning knowledge base would be less like a data based that required human guardianship and would be more like an apprentice or assistant.

The intent here is not to cover the many different types of machine learning methods available; those are covered admirably by others (e.g., Julien et al. 1992, McQueen et al. 1995, Stockwell et al. 1990). Rather, in the remainder of this paper I examine memory, feedback, pattern formulation, and pattern recognition as the 4 important learning components that must be addressed in order to make learning part of KBS. First, however, I briefly review several ideas related to learning and introduce a generic nomenclature that applies to all automated learning systems.

Learning In General

Supervised and Unsupervised

In general, there are two types of learning—supervised and unsupervised. They differ by the presence/absence of an explicit target response for each learning instance. Nevertheless, it should be apparent from the following sections that, despite this difference, we can still view supervised and unsupervised learning as variants of the same process.

In supervised learning, an input signal is paired with a target response. An intelligent agent forms an associate between these 2 components of each training pair. The target response provides a feedback mechanism that steers the agent away from incorrect behavior and toward correct behavior. Learning becomes difficult when: (1) there is no clear, unambiguous, and 1-1 pairing of inputs and responses due to incomplete or noisy data, (2) the space of all possible inputs is large, or (3) the agent needs to make correct responses to previously unseen input signals. All 3 of these conditions are typical for supervised learning in most real-world situations. The generic task for an agent that is learning under supervision is "classification", where an input signal indicates one, particular output response more than others.

The situation is slightly different for unsupervised learning. Here, a target responses is lacking. So, the intent becomes to identify similarities and differences among input signals. As more and more example input signals are received, their similarities and differences begin to segregate those examples into groups whose members are more alike than unlike. Differences are most apparent between groups and much less so within groups. As with supervised learning, above, learning is difficult because: (1) there is no absolute nor obvious way to group input signals, (2) the space of all possible input signals is large, and (3) the agent needs to identify meaning groups without feedback. Unsupervised learning is often viewed as a clustering task, with the intent being to "discover" some relationship between similar signals, in contrast to dissimilar signal groups. This can also be called "concept formation"; the different groups are labeled with names to indicate previously unknown concepts.

Learning-System Components

Whether supervised or unsupervised, the process of learning can be viewed as containing three primary tasks. First, there needs to be some ability to store and recall various pieces of information, i.e. memory—more specifics about these pieces are given below. This task can be relatively trivial and is handled quite well by traditional data and information management techniques. Nevertheless, there are some issues surrounding the storage and recall of example cases which can have a large impact on system use and performance. These are discussed more below.

Second, learning requires that an agent be able to form patterns from examples (or instances) that it is faced with. This pattern formulation allows the agent to map specific instances onto prototypical representations that generalize individual cases. Generalization greatly compacts knowledge because each instance does not require treatment as a special case.

Third, when presented with an example problem, the agent needs to recognize what prototype pattern that example belongs to and react accordingly. The recognized pattern may explicitly code for a predefined action or for several actions. This pattern recognition capability can be viewed as memory recall along with application. Unsupervised learning contains no explicit action to perform given an input signal, but there needs to be something built into the learning process that eventually recognizes that certain groups are genuinely different and can be treated as such in a conceptual way. I'll get more detailed about pattern formation and pattern recognition in the following sections, but first I'll introduce some nomenclature to be used in later derivations.

Nomenclature

Instance Vectors

The following nomenclature is introduced to ensure unambiguous terminology and clarity of expression as topics are presented. The mathematical terminology of sets and mappings is used, but no attempt has been made to be entirely rigorous here. We can represent any problem instance, or example case, e_i as a value vector $e_i = (v_1, v_2, ..., v_n)$, where each of the v_j are values from corresponding parameter sets V_j. These parameters and their values completely describe a problem instance. The n-valued vectors e_i are taken from the set $\boldsymbol{E}$ of all possible such vectors. Not all problems instances, however, will have known values for all the parameters. So, we must augment each of the parameter sets V_j with "?". Then, some of the v_j for any e_i may take on the special value "?", for unknown. The set $\boldsymbol{E}$, then, could be described mathematically as $\{ (v_1, v_2, ..., v_n) \mid v_j \in V_j \cup \{?\} \}$.

As a concrete, but trivial, example, let's assume that we are identifying farm animals. Parameters in this case, might include *feathers_present*, *tail_type*, *wool_present*, *number_of_stomachs*, and *height*. A chicken instance might present a vector, such as (1, ?, ?, 0, small), where 0 and 1 indicate present and not_present, respectively. Another chicken instance might present the following vector (?, stubby, ?, 0, small). It should be obvious, then, that the same type of farm animal can have different representations, depending on the problem instance. Our farm animal example has small cardinality so it would be possible to exhaustively enumerate all possible vector combinations of parameter values, and thereby completely describe the elements of $\boldsymbol{E}$. In general, however, this may not be practical, and oftentimes various combinations of parameter values are illogical; e.g., both feathers and wool present on an animal would be unrealistic.

Decision Classes

Because the goal is to identify farm animals, we are dealing with a supervised learning situation. Alternatively, if we have no knowledge of *different* farm animals, we might try to learn the concepts of different farm animals based on descriptive instances of them (unsupervised learning). In supervised learning, we are trying to associate elements of $\boldsymbol{E}$ with particular types of farm animals, e.g. chicken, duck, goose, emu, rabbit, pig, goat, sheep, cow. Ideally, all pig instances would be associated with the pig class, duck instances with the duck class, etc. Let's refer to the class of farm animal types (decisions, in general), just enumerated, as $\boldsymbol{C}$. Because there are many different vector representations for a duck instance, it may be impractical or impossible to expressly memorize each such vector. Therefore, we need to generalize duck instances

so that we can associate generalized patterns from a subset of all duck instances to the full set of duck instances. Generalization does several things: it greatly economizes the work we need to do to correctly identify farm animals, it allows us to deal with instances that we haven't seen previously, and we can improve performance as new instances are introduced.

Prototype Vectors

These generalizing patterns can be represented as prototype vectors $p_k = (w_1, w_2, ..., w_n, c_i)$. Each of the w_j, $1 \le j \le n$, is either, a specific value, a descriptor for a range of values, or "I don't care" (*) for the corresponding parameter. This representation can be translated quite easily into decision trees or rules; an implicit *AND* exists between the parameter/parameter-value pairs. Just as we augmented the parameter sets V_j, above, with the value "?", here we augment the original parameter sets with the special symbol "*" instead. The target response $c_i \in \boldsymbol{C}$ is included as part of the prototype vector in the supervised learning case. We, then, define the set $\boldsymbol{P}$ of all prototype vectors as $\{ (w_1, w_2, ..., w_n, c_i) \mid w_j \in V_j\{*\}, c_i \in \boldsymbol{C} \}$. For example, the prototype vector (1, *, *, *, *, chicken) could be a generalization for fowl. To correctly represent "fowl" using this nomenclature, however, we would need similar prototype vectors, but with "chicken" replace with "duck" and with "goose", etc. So, the input signal would be paired with each of the different types of fowl. Then, an input vector descriptive for "fowl", in general, would match a prototype vector for each type in the class "fowl". In this way, the class "fowl" would be implicit in the set of prototype vectors activated. The prototype vector (1, *, *, *, small, chicken) also generalizes for fowl, but without the emu class perhaps (depending on whether we consider emu a fowl). For unsupervised learning, the pk's would not contain target responses, so $\boldsymbol{P}_u$ (unsupervised) would be $\{ (w_1, w_2, ..., w_n) \mid w_j \in V_j\{*\} \}$

Mapping Function

Because we need to create the p_k from the e_i, we develop, as part of the learning process, a function f that generates the elements of $\boldsymbol{P}$ from the elements of $\boldsymbol{E}$. The function f, then, maps the e_i onto the p_k. Using this nomenclature, the function f could simply be an element-wise copy (where "*" replaces "?"). Of course, applying rule induction or decision-tree construction methods would be much more involved than this, but the idea is the same. From the above examples of instance vectors and prototype vectors, it is obvious that a single instance vector may be mapped by f to multiple prototype vectors. So, in general, $f(e_i) = P_k$ for some i and k, where $P_k \subseteq \boldsymbol{P}$, i.e., each e_i maps onto a set of prototype vectors. For example, the first chicken vector (1, ?, ?, 0, small) maps onto both of the previous prototype vectors, in addition to others.

For supervised learning, the mapping f applied to a particular e_i completely determines the outcomes, or desired actions, for that problem instance. Because f is not a 1-1 mapping, however, multiple outputs will be indicated. Outputs that are most frequently associated with a particular input signal will receive greater weight (multiple prototype vectors for the same output). In the case of unsupervised learning, the mapping f determines possible conceptual groups. Prototype vectors that are most similar or "closer together"—regardless of how one defines proximity—will define unique groups. Because a single instance vector can map to multiple prototype vectors and these prototype vectors may belong to different groups, an instance vector may associate with several conceptual groups simultaneously.

In terms of learning, then, pattern formulation (generalization) generates the prototype's p_k's. These prototype vectors are later used in recall/application to map encountered problem instances e_i's to the appropriate classes c_k's for decision-making (supervised) or to previously recognized and distinct groups (unsupervised).

Adding Learning to KBS

In this section I discuss some of the important issues that need to be addressed as KBS are augmented with learning capabilities. These issues include: memory, feedback, pattern formulation, and pattern recognition.

Memory

While providing some memory capability can be a relatively trivial aspect of a learning system, there are also important things to consider. First, the type of data being stored can vary drastically. In some cases, the data may be numerical statistics (e.g., Fynn and Fraser 1993) that are easily stored in a data file. That report presents a unique example of the above nomenclature in which there is only a single parameter value in each instance vector and the prototype vector contains the beta distribution parameters for the entire data set. More typically, the data will be parameter values for a large number of example cases (Meyer and Flanagan 1992), residing in a data base. In the extreme case, more elaborate knowledge bases structures, e.g. spatial relations (Jones and Roydhouse 1994) or frame-like structures (Chiriatti and Plant 1996), may need to be stored. Second, some consideration must be given to the level of data extensibility that users are allowed, i.e., user access to example cases. In some situations, users may be given full control of data augmentation and modification, and in other cases, user access may be very limited (corporate data, for example). Third (related to the second), there should be some provision ensuring that previous performance levels can be recovered should the data set(s) become corrupted. If a system is learning from, and dependent on, example cases, then its performance can be severely compromised by erroneous data. Inadvertent, or erroneous, changes to example cases should be mitigated or, at least, recoverable. It should be possible to recover to a previous state of operation.

Feedback

For systems that learn in a supervised mode, explicit feedback is crucial to the learning process. Feedback steers the learning process toward meaningful concepts/patterns (prototype vectors) and away from inappropriate ones. The question, then, arises as to where feedback comes from. Does the user provide feedback? Or do the data contain it? Typically for supervised learning the examples contain explicit feedback in terms of a correct response from the decision set C. However, even in this case, a system will eventually produce an unsatisfactory/incorrect decision for a new instance vector using its current store of prototype vectors. For a system that learns prototype vectors in the form of decision trees or rules, it may only be necessary to provide the correct decision and let the system derive a new tree or new rules to accommodate the new example case. In this scenario, the system is able to take corrective measures autonomously.

If, however, the current problem instance is an anomaly, then incorporating this new example case into the knowledge base may create a prototype vector that is inconsistent with existing vectors or might create a new set of prototype vectors that are less effective than the old set. This revised system knowledge may reduce the system's ability to correctly respond to previously learned cases. In effect, the system is learning from a bad example. At that point, it is then up to the user to inform the system of this erroneous outcome and perhaps help guide the system to correct the faulty logic. This points to the need for there to be continual monitoring of system performance over time as new learning occurs. Monitoring will help ensure that new learning doesn't compromise existing knowledge and performance.

In case-based reasoning, two very important tasks are: matching similar cases (Meyer and Flanagan 1992) and translating the specifics of a matched case into comparable conceptual structures and recommendations within the new case (Chiriatti and Plant 1996). An incorrect decision in case-based reasoning may arise from a failure in either of these tasks. The stored cases *are* the prototype vectors, in this exemplar type of learning. Selecting the proper case (prototype vector) is critical for effective reasoning. Feedback comes from the user, as system recommendations are evaluated for correctness. Incorrect decisions are traced back to poor matching or to inaccurate translation of case specifics. This may require modification of system heuristics or algorithms. A third possibility, of course, is that the new case does not match with or translate well from existing cases because it is itself a new exemplar. Then it is added to the existing cases, and it becomes a new prototype vector for a new class of instance vectors.

The situation for unsupervised learning is very similar to the case-based reasoning scenario. When grouping a new example, user feedback may indicate a misclassification. This could signal either ineffective matching heuristics or, possibly, a new class of examples.

Pattern Formulation

Two closely related problems encountered in forming patterns from data are *overgeneralization* and *overspecialization*. In the former, patterns (prototype vectors) contain too many "*" ("I don't care") characters. The vector (1, *, *, *, *, chicken), from above, is an example of this. Overgeneralization produces patterns that are activated in too many situations, e.g., if we see an animal with feathers, then we always assume that it is a chicken. Systems that are overly general have the advantage that they almost always give an answer, but it's very often not the right one.

Overspecialization, on the other hand, creates patterns with very few "*" characters. That is, each parameter has a specific value, e.g. the prototype vector (1, stubby, 0, 0, small, chicken). Patterns with this degree of specificity are only activated by very detailed examples. Therefore, overspecialized patterns are not useful for a wide variety of examples. They don't give an answer for incomplete examples, but when they do give an answer it's usually the correct one.

Julien (1992) provides two approaches for avoiding overgeneralization and overspecialization. Both rely on external guidance by an expert, one uses input *before* patterns are formed and the other uses feedback to correct formed patterns. Effective learning cannot occur in a vacuum, but requires some external support.

Many of the most common pattern formulation methods can accommodate some data imperfections, such as incompleteness, irrelevancy, redundancy, noise, and errors (McQueen et al. 1995). However, none of these methods include uncertainty in their resulting patterns. That is, prototype vectors do not generally contain a value for the pattern's correctness or reliability. This means that the application of learned knowledge during pattern recognition produces results that are absolute, without any indication of likelihood or belief.

Pattern Recognition

Once a set of prototype vectors has been developed through learning, then new examples (instance vectors) can be applied to those patterns. When the training set is extensive, the prototype vector set $\boldsymbol{P}$ will produce good results for new instance vectors, provided that overspecialization has been avoided. For sparse training sets, results will be either inaccurate (general patterns only) or inconclusive (no special pattern available). This is another place where feedback is important, because incorrect results during application can support additional learning.

Any system that deals with incomplete or erroneous example data will produce an ambiguous pattern set. That is, because during learning the function that maps instance vectors to prototype vectors is not 1-to-1, each of those patterns may generate different decision class values. To deal with this either: (1) the patterns must be modified so that only one is activated for any instance vector (this may reduce system robustness by eliminating general patterns) or (2) another component of the KBS must interpret this ambiguity as uncertainty and report it to the user accordingly. The former is really a pattern formulation issue, whereas the latter addressed pattern recognition and interpretation, the things that traditional KBS do well.

Conclusions and Discussion

The advantages of KBS that can learn are many, and hopefully obvious, but nonetheless deserving of mention here. As noted earlier, manual updating of KBS is not generally an easy task. System logic can often be complex and dependencies can be confusing. System maintenance and updating tasks can be greatly simplified when they are performed automatically by a system that learns from past mistakes and from new examples. Although some manual attention is still required, it can be much less intensive and much less frequent.

Because learning systems adapt to new cases, knowledge is updated regularly. This recency of examples ensures that system knowledge is modified appropriately to reflect current types of problems and their solutions. Consequently, as overall strategies change for dealing with certain problems, application knowledge will keep pace simultaneously. Systems become self-evolving and adaptive, much like biological and social systems.

I have focused here mostly on system performance and modification, but obviously initial system construction can also proceed more quickly and easily when elaborate knowledge acquisition sessions can be reduced or eliminated. This can greatly

shorten the development time from concept to field application.

In addition to these advantages, some things make learning systems problematic, for now. First, because learning systems rely on a memory of past cases and use this memory to deal intelligently with novel situations, storage and access to this repository of experience should not be taken lightly. Second, a feedback mechanism that helps correct faulty reasoning—which may include the reasoning of the user—needs to distinguish between bad reasoning and bad examples. Third, the most common pattern formulation methods do not produce patterns with associated uncertainty values. Fourth, except for the machine learning workbench of McQueen et al. (1995), no one has yet brought these tools to the user—and even this workbench does not contain a pattern recognition, i.e. application, component, it only does pattern formulation. Fifth, learning patterns in example cases often leads to ambiguous patterns, which should be correctly interpreted as uncertainty.

One can view the meaning of the term, "artificial intelligence" in at least two ways. In one view, "AI" is interpreted as "intelligence artificially derived," i.e. man-made, yet possessing many of intelligence's essential qualities. The second view, and the one posited here, interprets "AI" as "intelligence (as in knowing certain things) that lacks essential qualities for true intelligence." Any system defected in this way can only be artificially, not genuinely, intelligent. While an artificial intelligence can be extremely proficient at one, certain task, it is basically quite stupid, as well. This is the "idiot savant" view of AI, and is consistent with the design, development, and use of AI's most notable products, i.e. KBS.

One of the missing essentials, emphasized in the previous pages, is the capability to learn—to formulate new patterns, to discover new knowledge, to become more proficient with experience. KBS have traditionally been good at pattern recognition for instances that their knowledge base was designed for, but are unable to form new patterns or to associate new instances with existing patterns. By way of an aphorism, one might say, "If it don't learn, it ain't AI." This phrase should come as no great revelation to anyone, but acknowledging it is very different from acting on it. Eventually, it's time that we address the real need for learning in KBS, and begin to design systems that can accommodate it.

References

Carrascal, M. J., and L. F. Pau. 1992. A survey of expert systems in agriculture and food processing. AI Applications 6(2): 27-49.

Chiriatti, K. C., and R. E. Plant. 1996. NPK: A prototype case-based planning system for crop fertilization decision support. AI Applications 10(2): 33-42.

Durkin, J. 1993. Expert systems catalog of applications. The University of Akron Printing Department, Akron, Ohio.

Fynn, R. P., and J. M. Fraser. 1993. Learning from data: The beta distribution and probabilities of solar irradiance ranges. AI Applications 7(4): 45-57.

Harmon, P., and B. Sawyer. 1990. Creating expert systems for business and industry. New York: Wiley.

Jones, E. K., and A. Roydhouse. 1994. Intelligent retrieval of historical meteorological data. AI Applications 8(3): 43-54.

Julien, B. 1992. Experience with four probability-based induction methods. AI Applications 6(2): 51-56.

Julien, B., S. J. Fenves, and M. J. Small. 1992. Knowledge acquisition methods for environmental evaluation. AI Applications 6(1): 1-20.

Langley, P., H. A. Simon, G. L. Bradshaw, and J. M. Zytkow. 1987. Scientific Discovery. The MIT Press, Cambridge, Massachusetts.

Loh, D. K., Y. T. C. Hsieh, Y. K. Choo, and D. R. Holtfrerich. 1994. Integration of a rule-based expert system with GIS through a relational database management system for natural resource management. Computers and Electronics in Agriculture 11(2/3): 215-228.

Matthews, G., R. Matthews, and W. Landis. 1995. Nonmetric conceptual clustering in ecology and ecotoxicology. AI Applications 9(1): 41-48.

McQueen, R. J., S. R. Garner, C. G. Nevill-Manning, and I. H. Witten. 1995. Applying machine learning to agricultural data. Computers and Electronics in Agriculture 12(4): 275-295.

McRoberts, R. E., D. L. Schmoldt, and H. M. Rauscher. 1991. Enhancing the scientific process with artificial intelligence: forest science applications. AI Applications 5(2): 5-26.

Meyer, C. R., and D. C. Flanagan. 1992. Application of case-based reasoning concepts to the WEPP soil erosion model. AI Applications 6(3): 63-71.

Power, J. M., and H. Saarenmaa. 1995. Object-oriented modeling and GIS integration in a decision support system for the management of eastern hemlock looper in Newfoundland. Computers and Electronics in Agriculture 12(1): 1-18.

Reynolds, K., P. Cunningham, L. Bednar, M. Saunders, M. Foster, R. Olson, D. Schmoldt, D. Latham, B. Miller, and J. Stephenson. 1996. A knowledge-based information management system for watershed analysis in the Pacific Northwest U.S. AI Applications 10(6): 9-22.

Schank, R. C. 1987. What is AI, anyway? AI Magazine 8(4): 57-66.

Schmoldt, D. L., and H. M. Rauscher. 1996. Building Knowledge-Based Systems for Natural Resource Management. Chapman and Hall, New York.

Stockwell, D. R. B., s. M. Davey, J. R. Davis, and I. R. Noble. 1990. Using induction of decision trees to predict greater glider density. AI Applications 4(4): 33-43.