Pergamon

Expert Systems With Applications, Vol. 10, No. 3/4, pp. 393–401, 1996

0957-4174/96 $15.00+0.00

S0957-4174(96)00018-8

Interactive Induction of Expert Knowledge

BINGCHIANG JENG, TING-PENG LIANG AND MINYANG HONG

Department of Information Management, National Sun Yat-Sen University, Kaohsiung, Taiwan 80424, Republic of China

Abstract—*The process of extracting, structuring and organizing elicited knowledge (called knowledge acquisition) is a bottleneck in developing knowledge-based systems. A manual approach that elicits domain knowledge by interviewing human experts typically has problems, because the experts are often unable to articulate their reasoning rules. An automatic approach that induces knowledge from a set of training cases also suffers from the unavailability of sufficient training cases. We present an integrated approach that combines the strengths of both methods to compensate for their weaknesses. In this approach, human experts are responsible for solving problems, whereas an inductive learning algorithm is responsible for reasoning and consistency checking. Copyright © 1996 Elsevier Science Ltd*

1. INTRODUCTION

KNOWLEDGE ACQUISITION is a process of extracting, structuring and organizing elicited knowledge from domain experts or other knowledge sources and converting the knowledge into rules or other forms of representation accessible with a computer program [16,25]. As the power of an expert system comes from the knowledge that it possesses [13], the acquisition of knowledge is a major stage in the development of an experts system.

The transfer of expertise from various sources to a knowledge base is difficult and challenging. A manual process that encodes domain knowledge by interviewing human experts suffers from several difficulties [31,40]. For example, human experts are typically not also expert in articulating their rules of reasoning because they are not explicitly aware of the structure of their knowledge. A paradox of expertise [20] claims that the better one is an expert, the worst one is to tell the details. Knowledge acquired during interview may be unreliable, as verbal reports and mental behavior are not necessarily correlated [2]. Incongruities may occur among what an expert says that he does, what he actually does and what he should have done [17].

Communication between the knowledge engineer and the domain expert is another difficult problem. The knowledge engineer may know little about the problem domain, and may not understand clearly the jargon used by the domain expert. On the other hand, the domain expert may not understand the process of knowledge acquisition and what the knowledge engineer needs. Even the cultural differences between the two parties of knowledge acquisition may undermine the development of a knowledge-based system.

When historical examples are available, an automatic process can be used to induce rules from fragments of knowledge [35,36]. The unique power of this approach is the "automatic" learning capability of its inductive algorithm. Many successful applications are reported, such as disease diagnoses [42,23,29], business applications [5,8,39,22] and others [5,10,26].

A major constraint of automatic learning is that it requires many training cases from which to induce knowledge. The quality of the training result depends on the quality of the data. Another drawback is that induction may generate knowledge that uses paths of reasoning different from those used by the expert. This may make human experts reluctant to accept the induced knowledge base regardless of its performance.

Given that each approach alone has limitations and their respective strengths and weaknesses seem complementary, we seek to integrate the two approaches to gather the advantages and to avoid the deficiencies. The idea is that the weaknesses of one approach be compensated by the strengths of the other. For example, the task of articulating a reasoning process for a human expert can be left to an inductive method to solve, or the restriction of inductive learning on training examples can be given to an experienced human expert to provide help. This approach is useful as human experts are good at solving problems whereas inductive learning algorithms are good at extracting rules of reasoning.

Previous research that aimed at a similar goal exists. For example, Parsaye [31] proposed an approach that combines interactive knowledge acquisition with rule induction. Based on the theory of repertory grids [3,21], the system can interactively interview an expert by

asking questions to capture the expert's knowledge. The captured knowledge is then generalized using rule induction. Buntine and Stirling [7] presented another approach that allows a human expert to interact with a set of automatically induced rules so that each other's knowledge is cross-checked. Recently Evans and Fisher [12] reported a similar approach that is successfully applied to a problem domain where the experts have only weak causal knowledge.

A limitation of Buntine and Stirling's approach is that it still depends on a set of pre-existing training examples. In some cases, these examples are hard to come by. Furthermore, the way an expert interacts with the preliminary rules is informal.

The approach presented in this paper extends their ideas to allow interactive knowledge acquisition from (nearly) scratch. A modified algorithm for inductive learning is designed so that an expert communicates with the system in an interrogative style during the inductive process. In addition to inducing a decision tree from existing training cases, the system also identifies near-miss cases falling on the classification borders for clarification and verification by human experts. These cases, after classification, become new training cases for accurate learning by the modified inductive algorithm. This way, the expert's knowledge is accumulated incrementally and the learning process can be repeated until the induced knowledge is satisfied by the human expert.

The remainder of the paper is organized as follows. The process of automatic knowledge acquisition and its related techniques is reviewed in Section 2. An interactive process for inductive learning that shows how to elicit knowledge from a domain expert is described in Section 3. Experimental results from the evaluation of the performance of the proposed approach is presented in Section 4. Section 5 summarizes the research.

2. AUTOMATING KNOWLEDGE ACQUISITION

There are many different methods for knowledge acquisition, which can be classified as: manual, semi-automatic and automatic [11,25]. Eliciting knowledge by manual methods is highly labor-intensive. Methods belonging to this category include structured/unstructured interviews, analysis of protocol, observations and so forth [40].

Semi-automatic methods are primarily designed to support either the expert or the knowledge engineer to perform the necessary task more effectively. Tools to elicit domain knowledge include ETS, KRITON, AQUINAS, MORE, MOLE etc. [4,24]. Others include TEIRESIAS [17] that assists the knowledge engineer to update a knowledge base, and KADS [14] that provides a collection of tools to support a knowledge engineer to extract, to structure, to analyze and to document expert knowledge.

Although semi-automatic methods expedite the work of knowledge engineers and/or domain experts, they still share problems of manual methods: the acquired knowledge is difficult to validate; correlation between verbal reports and mental behavior may be weak. In addition, Michie [28] stated that knowledge of certain types could not be elicited using manual methods, simply because the domain was so large and complicated that the expert would be unable to explain how it operates.

Automatic methods that use machine-learning techniques to extract knowledge require less or no participation by either knowledge engineers or domain experts. Therefore, they do not have the difficulties associated with human experts. Currently, the most popular automated method is rule induction.

Induction is a process of general inference from particular instances. Rule induction (or inductive learning) refers to a concept learning process by which a set of rules is created from training cases to explain or to predict a problem-solving behavior. When the induced structure of knowledge is represented in the form of a decision tree, it is also called (decision) tree induction [27,35]. A decision tree is considered as a set of rules in a compact form; it can be transformed into rules easily.

Early work on rule induction is traced to 1966 when Hunt, Martin and Stone developed a method for induction. The method was later implemented and expanded by Paterson and Niblett [32] to create ACLS (A Concept-Learning System) and by Quinlan [34,35] to develop the popular ID3. In the following, we shall use the ID3 algorithm to illustrate how rule induction works.

Input to ID3 is a collection of training cases. Each is described by a set of attributes associated with a class name (or outcome). An attribute can be either categorical (e.g. color) or numerical (e.g. age). A numerical attribute can adopt discrete or continuous values. The basic procedure of ID3 applies a divide-and-conquer approach to partition recursively the data set, based on a test on selected attribute values, into mutually exclusive subsets. The procedure continues until each subset contains cases of the same class (to avoid over-fitting the training data, termination conditions may be defined) or no attribute is available for further decomposition. If $S=\{(a_{i1}, \ldots, a_{in}; c_i) | a_{ij} \in A_j, c_i \in C)\}$ is a set of training cases, where A_j denotes the domain of attribute j and C for the class, the algorithm is shown in Fig. 1.

An example shown in Fig. 2 is the decision tree created by ID3 from financial data to analyze the risk of bankruptcy. A new case is classified by examining which leaf it reaches when traveling down through the tree. The traversed path is determined by a sequence of tests to the new case (i.e. a branch is selected depending on the outcome of a test that the new case generates). When it reaches a leaf in this way, it is considered similar to other existing cases contained in the leaf as they have all passed the same tests. Hence, the outcome of the new case is the same as its neighbors and is assigned with the

```
Procedure Induction (S: TrainSet, T: Tree)
Begin
   If S contains cases of the same class
   Then label T with the class name and exit
   Else Begin
      For each numerical attribute A_j, Do
         Find a value v_j to decompose the
         training set into two subsets,
         Calculate the entropy of the decomposition,
         Choose the decomposition whose
         entropy value is the smallest;
      For each categorical attribute,
         Decompose S by its classes
         and calculate its entropy;
      Partition S into two or more mutually exclusive
         subsets: S_i, i = 1,..., k, based on the
         attribute A_s whose entropy value is the
         smallest after decomposition.
      Create a node T_i for each subset S_i, and
         link it to the parent node as its child.
      For i = 1 to k, call Induction (S_i, T_i)
   End
End.
```

FIGURE 1. The ID3 algorithm.

class name labeled at the leaf. For example the prediction of bankruptcy for case (0, 0, 0, 0.02, 0.05, 0.6, 0.04) is YES, as it follows the path at the far left down to a leaf labeled "YES".

3. INTERACTIVE INDUCTION

The fundamental basis for methods of decision tree induction (or rule induction) is that two cases with similar features are classified into the same class. The success of such an approach relies on sufficient training cases to cover every aspect of the problem domain without inconsistency. Otherwise, the induced knowledge may be unreliable.

In practical applications, however, this assumption is commonly violated. The sources of training cases are, in general, from a domain expert or from historical data in a maintained database. Difficulties arise typically because the domain expert can provide only a few selected examples, whereas historical data, if they exist, may be obsolete or contain errors and missing values.

A further difficulty with inductive learning is in its explanation capability. Explaining the reasoning process by which a conclusion is reached is a key requirement for an expert system. Rules induced from the training set may, however, differ from those used by the expert. This makes its explanation sometimes less acceptable to human beings because the underlying process of reasoning used by the expert system may be incomprehensible to users.

It is thus useful to have interactions between human experts and inductive learning algorithms. This allows human experts to provide useful knowledge, such as hand-crafted tutorial examples, rules of thumb, general hints and problem-solving strategies, to help an inductive algorithm.

The inductive algorithm presented below is an example that supports interaction with experts during its learning process. After a knowledge structure is induced from its training cases, the algorithm identifies cases that cannot be correctly classified. These cases are brought to the expert in the form of questions to be solved. Once they are solved, they become new training cases to the inductive algorithm for further learning. In this way, expert knowledge is incrementally elicited and incorporated into the induced knowledge structure. A typical scenario of the interactive inductive process is as follows.

Initially, the knowledge engineer collects knowledge from all possible sources. Different forms of knowledge are stored differently. Rules (mostly from domain experts) are stored in the rule set and tutorial cases (mostly from the historical database) are stored in the training set. Then, the algorithm tests any inconsistencies

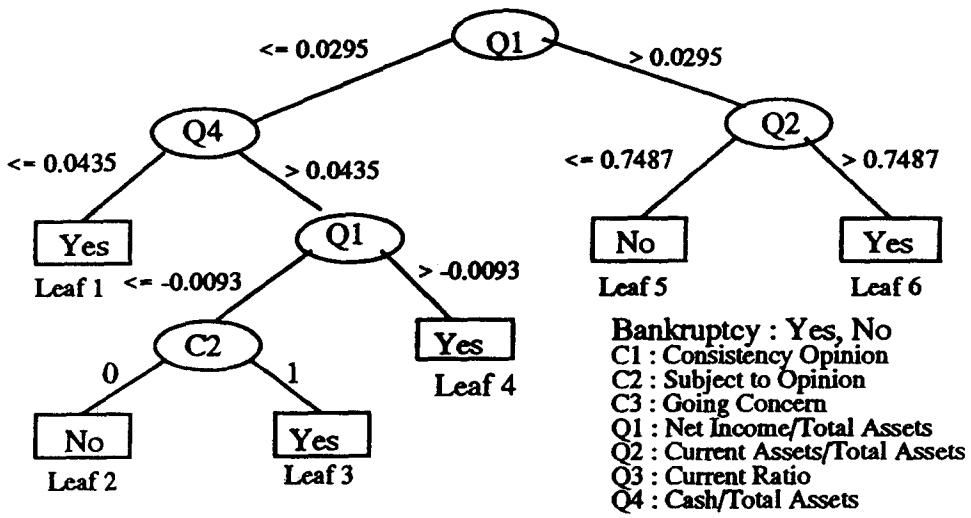
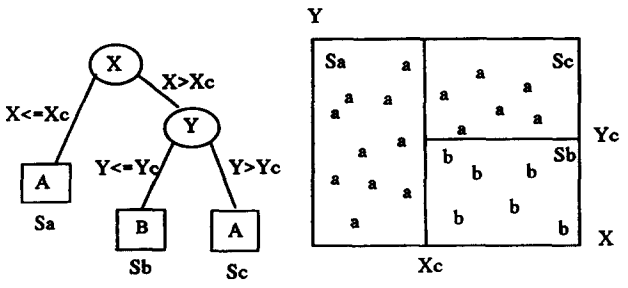


FIGURE 2. An example of a decision tree for bankruptcy prediction.



(a) A decision tree (b) Partition of feature space

FIGURE 3. Space partition by a decision tree.

in the knowledge structure represented by these two sets. As both sets are incomplete at the beginning, contradictory cases (i.e. cases classified differently by those two knowledge sources) are identified. These cases are then presented to the domain expert for further review.

Prompting experts to review contradictory cases has two purposes. First, certain knowledge ignored by the expert may be use to provoke him to describe the inference rules in more detail. Then the contradictory cases, after correction, become new examples to the inductive algorithm for further learning. Revision of the two sets triggers the cross-validation process again and the above procedure is repeated. A sketch of this approach is shown in Fig. 5.

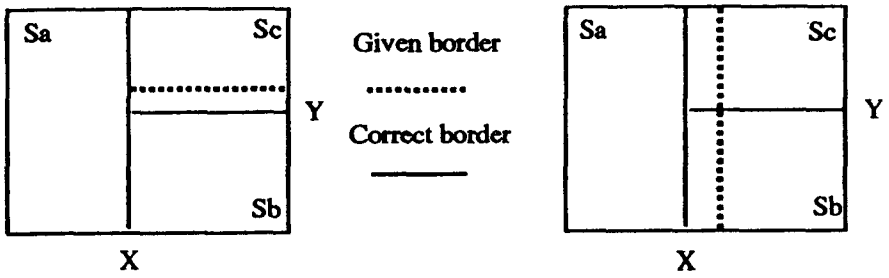
An important feature of the above algorithm is its ability to detect inconsistencies during the process of interactive induction. This is important when two different sources of knowledge are merged because inconsistencies may exist. In the following we shall present an algorithm for detecting inconsistencies.

3.1. Strategy for Border Validation

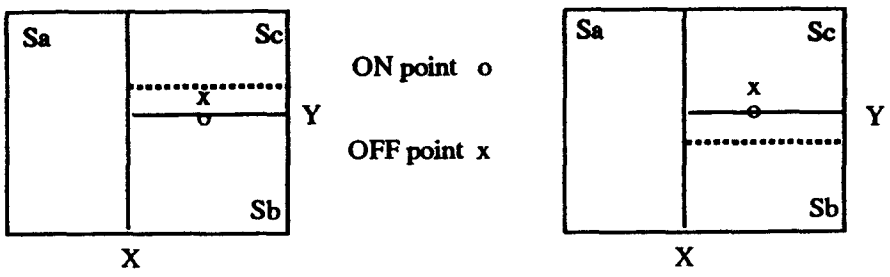
A decision tree created from training cases can be represented as a partitioned feature space. The induction of decision trees (or rules) in some sense is a process to maximize the internal similarity within the partitioned subspaces. Each leaf of the tree corresponds to a subspace, and each case in the problem domain corresponds to a point in it. For example, a decision tree shown in Fig. 3 partitions a two-dimensional feature space with attribute (X, Y) into three subspaces. Subspaces S_a and S_c represent class A and subspace S_b represents class B. A new case is evaluated based on the subspace into which it falls (e.g. a case with $X<X_c$ is classified into class A).

Similarly, rules acquired from human experts can also be illustrated as partitions of the feature space. The problem of verifying whether two knowledge sources (i.e. rules and the training set) have inconsistencies is thus equivalent to verifying whether their partitions are identical, which can be tested by simply determining if there are cases that fall into different subspaces in these two partitions. This is explained in detail in the following.

A *border* is defined as a boundary that separates two subspaces. A border is *redundant* if both subspaces adjacent to the border represent the same class (e.g. the one between subspace S_a and S_c in Fig. 3). Inconsistencies occur when more than one non-redundant border can be found to partition a space into subspaces.



(a) Borders X and Y are shifted



(b) Border shifts are detected with ON-OFF points

FIGURE 4. An example of border shift and the strategy to detect it.

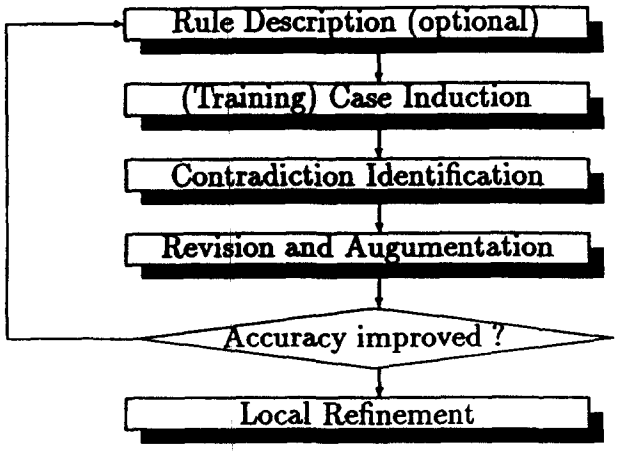


FIGURE 5. The process for interactive induction.

Such a situation is called a *border shift*, since at least one of them is incorrect. Inconsistency exists if and only if a border shift exists. Figure 4(a) shows two examples of border shifts.

A strategy for detecting border shifts, that requires only two points (ON and OFF), can be found in [18]. The *ON point* lies exactly on a given border to be verified, while the *OFF point* lies slightly off in the open side of the border. A border shift exists if the actual location of the border falls outside the OFF point or inside the ON point. In other words, no border shift exists if it passes exactly between the two points.

For example, if border Y lies in a different location (the broken line) in the left diagram of Fig. 4(b), the OFF point would have been classified differently. Similarly, the ON point would be classified differently in the right diagram of Fig. 4(b), if the solid-line border is shifted to the broken-line one. With carefully chosen testing points, the strategy can detect a border shift effectively, if it exists.

3.2. Algorithm for Test-Case Generation

The following is an algorithm for finding non-redundant borders and verifying their correctiveness in a partitioned feature space. We use the decision tree shown in Fig. 2 as an example to illustrate it.

The first step of the algorithm is to create a table of path conditions according to the decision tree. Each path from the root to a leaf of the decision tree corresponds to a row in the table. Attributes to partition the training cases correspond to columns. Each field in a row records an attribute condition defined by the path. The last column denotes the outcomes. Table 1 shows the path conditions of the decision tree in Fig. 2. Each cell in the table gives the upper and lower bounds of an attribute.

The purpose of the path condition table is to determine the adjacency relationship between two subspaces. In the following, we show how to identify a non-redundant border from the table. As non-redundant

TABLE 1
A Table of Path Conditions for the Bankrupt Decision Tree

Path	$F2$	$F3$	$F4$	$F5$	Class
1	–	$-\infty/.0295$	–	$-\infty/.0435$	Yes
2	=0	$-\infty/-.0093$	–	$.0435/\infty$	No
3	=1	$-\infty/-.0093$	–	$.0435/\infty$	Yes
4	–	$-.0093/.0295$	–	$.0435/\infty$	Yes
5	–	$.0295/\infty$	$-\infty/.7487$	–	No
6	–	$.0295/\infty$	$.7487/\infty$	–	Yes

borders require their adjacent subspaces to represent different classes, adjacency relationships are determined only between paths that represent distinct outcomes. In our example, paths 1, 3, 4 and 6 belong to one group and paths 2 and 5 belong to another. A non-redundant border is identified when a path from one group is adjacent to a path from the other.

The adjacency relationship of two paths is calculated by assessing whether the intersection of their path conditions recorded in the table is non-empty. This step amounts to assessing whether corresponding attribute conditions of the two paths overlap. The results of the intersection define the border. For example, we can determine whether paths 1 and 2 have an intersection by examining whether their attributes overlap. The data in Table 1 indicate that their intersections are $F2=0$, $F3 \leqslant -0.0093$, $F4$="don't care" and $F5=0.0435$. As the intersection is non-empty, paths 1 and 2 are considered adjacent to each other. Other non-redundant borders are calculated similarly.

Given a non-redundant border, one can select an ON point and an OFF point to verify its correctness. For instance, cases to test the previously generated border might be $F2=0$, $F3=-0.0093$, $F4$="don't care", $F5=0.0435$ with class=YES (ON point) and $F2=0$, $F3=-0.0093$, $F4$="don't care", $F5=0.0436$ with class=NO (OFF point).

In our strategy, whether the ON point is located exactly on the border is relative unimportant, as long as the OFF point is chosen to be close to it. The test cases are then compared with the outcomes predicted by inferences from the rule set. If knowledge of the human expert is incompletely included in the rule set or the training data are imperfect, contradictions would occur. These test cases are then presented to the human expert for review. Otherwise the process proceeds to identify the next non-redundant border and to generate other test cases. At the beginning of the interactive induction process, a relatively large number of contradictory cases may be identified. After certain revisions, contradictory cases reduce.

3.3. The Process of Interactive Induction

To summarize, the process of interactive induction as shown in Fig. 5 consists of six phases: rule description,

case induction, contradiction identification, revision and augmentation, local refinement and result validation. Iterations occur in the first four phases until a termination criterion is satisfied. Then it passes through the phases of local refinement (optional) and result validation. This process differs from the traditional inductive learning that only performs case induction and result validation. No interaction between the expert and the induced knowledge is allowed.

The phase of rule description is a major step where an expert can provide judgments and transform into rules. Any manual techniques for eliciting knowledge described before can be applied here. The set of rules is not required to be complete as it will be refined in later iterations. It is not very difficult to create such a rule set as an expert can generally provide rules of thumb easily. The more complete the rule set which describes the expertise, the fewer iterations it needs to perform later. If the rule set is not available, the step of identifying contradictory cases will check with a domain expert directly.

Although ID3 is the example that we used in this research, the induction phase can use any inductive learning algorithm. The only concern is how the algorithm handles new training cases. It would be inefficient if the algorithm flushes its old memory and redoes the learning process from the beginning every time a modification is involved. Utgoff has described an incrementally inductive algorithm ID5R [41] which processes new training cases by updating the existing decision tree, instead of creating a new one. Thus the incremental learning cost of the algorithm is relatively low. The training cases for induction can be historical data (if they exist), or hand-crafted tutorial examples given by the expert.

The phase of contradiction identification detects inconsistency between rules and training cases. It uses the validation strategy described in Section 3.1. Contradictory cases are passed over to the revision and augmentation phase in which human experts re-inspect the underlying process of reasoning and revise the rule or the case descriptions. Cases after correction are included in the training set that initiates another iteration of the induction process.

In addition to simply resolving conflicts, defining a sufficient feature set for an application domain is a challenging problem for machine learning. The ID3 algorithm, for example, would perform poorly, if the attributes of a set of training cases are not sufficient to describe a domain. This problem however, is mitigated in our approach as human experts may add features to solve contradictions that cannot be solved by rule revision. This way, the features in the feature set can grow.

The local refinement phase is optional. It is primarily designed to fine-tuning the overall predictive accuracy of inductive learning. For a problem domain to be modeled, the predictive accuracy cannot be improved infinitely. A limit is often reached after several iterations. This is primarily due to the orthogonal partition of feature space in ID3, which may not match the real situation exactly. In this case, predictive accuracy cannot be improved by further training. Local refinement that performs learning independently on selected subspaces can then come into play.

Finally, the induced knowledge needs to be validated. A set of criteria for evaluating the quality of the inductive result and determining when to terminate the inductive process are critical. Other tools that facilitate the validation and maintenance of the knowledge base can also be useful in this stage.

4. PERFORMANCE EVALUATION

To evaluate this process, a prototype system implementing the method has been developed. The kernel of the system consists of three modules: one for inductive learning based on ID5R (an incremental version of ID3) [41], one for rule inference based on CLIPS and another to select test cases. The user interface integrates these modules.

To evaluate whether the interactive induction approach is practically useful, we conducted experiments on four hypothetical and one real problem. The hypothetical problems are to learn a concept that defines a geometric pattern, such as a circle, a polygon and so on. The real world problem is to learn the rules that an expert uses at a blackjack game to decide whether to call for another card.

4.1. Learning Geometric Patterns

In this experiment, the system needs to learn a geometric pattern. First, a circle was used for experiment. A point falling inside a circle was assigned *class 1* and others were assigned *class 0*, as shown in Fig. 6(a). Initially, nine training cases were selected randomly from the diagram. The rule provided by the expert was

Rule 1: If $X \leq 3$ and $X \geq -3$ and $Y \leq 3$ and $Y \geq -3$ Then
Class = 1

Else Class = 0;

The concept learned after the first iteration of the induction is given in Fig. 6(b), from which the contradictory cases identified are shown in Fig. 6(c). The final concept induced after three iterations of the learning process is given in Fig. 6(d). One can see that the result approaches the actual circle pattern. In addition to the nine given training cases, the system generates a total of 20 questions for the human expert to answer in the learning process.

Figure 7 shows how other geometric patterns can be learned in experiments. They all achieve satisfactory results. The learned patterns are near the actual patterns.

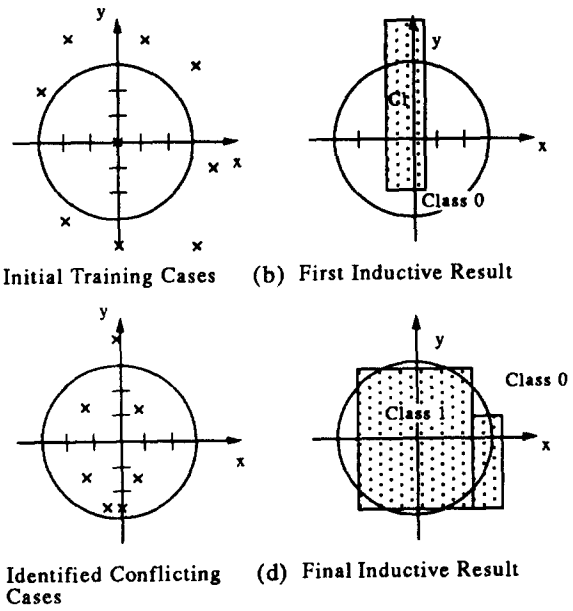


FIGURE 6. A learning process for the circle experiment.

The only limitation is that the nature of the orthogonal partition with ID3 does not allow a real curvilinear draw.

4.2. Learning Blackjack Games

Another experiment shows how a system can learn the rules that a human expert uses in playing blackjack. The game is played between a dealer and players. Each player receives two reversed cards, while the dealer receives one reversed and one obverse cards. An ace can count as either 1 point or 11 points. During the game, a player may request for more cards (to "hit") or to stay with the current hand (to "stand"). A player wins if the total points of cards in hand is below 21 points but greater than the dealer's.

A human expert decides whether to hit or to stand based on his current count in hand, with or without an ace, and the dealer's obverse card. These three attributes determine a decision. The expert's rules can be drawn in a decision diagram as shown in Fig. 8(a). Initially five randomly chosen cases were given for training. The rules learned after three iterations are shown in Fig. 8(b), which are pretty close to the actual rules used by the expert.

5. RELATED WORK

The subject discussed in this article can also be seen as a problem of concept learning in the machine learning field. In particular, the way a domain expert interacts with a learning system, can be formulated as a model of learning with membership queries [1], in which an instructor or oracle exists to answer any queries placed by a learner. In this model, the learner has control over what part of information it receives from a problem domain next.

Although our work starts initially from a different perspective, its result happens to be in accordance with most recent work in machine learning. Cohn et al. [9] presented an approach to learn a concept by generating queries from a region of uncertainty, an area in the domain where misclassification is still possible. An interesting coincidence in their neural network's implementation is that it uses two networks S and G to identify an uncertain case, when outcomes of the case are inconsistent between them. They demonstrated in several domains that the approach gives better learning results for a fixed number of training cases than simply learning from examples alone. This is encouraging since the rationale of identifying regions of uncertainty is similar to our identification of contradictory cases.

Another learning model similar to the above is the on-line (or incremental) learning model in which the learner answers a sequence of yes/no questions with immediate feedback provided after each question. A variant of the on-line learning model, called self-directed learning, has been recently proposed by Goldman and Sloan [15] which allows the learner to select the presentation order for the instance. Thus it can be seen as a variation of learning with membership queries in which the learner is only "charged" for queries whose outcomes are unpre-

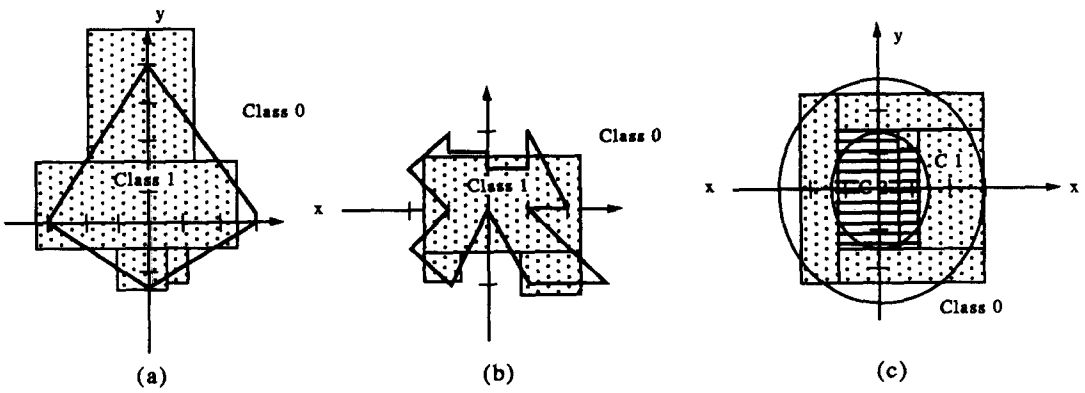
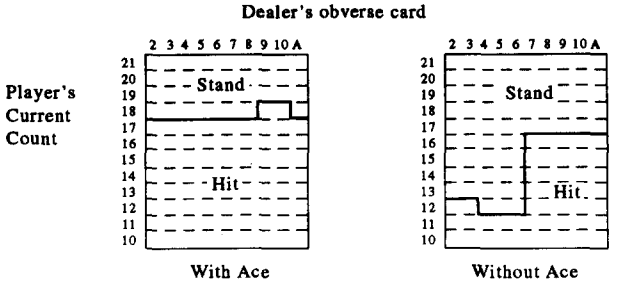
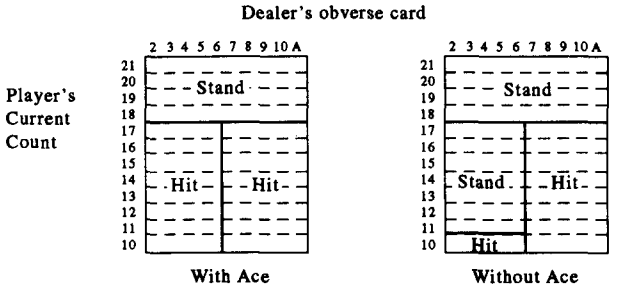


FIGURE 7. Learning other geometric patterns.



(a) Experts' rules to play blackjack



(b) Rules learned according to the inductive algorithm

FIGURE 8. Learning the blackjack playing rules.

dictable. Theoretical results given by Goldman and Sloan show that the performance of self-directed learning is the best among all other commonly studied on-line and query learning models.

Our process of learning by detecting contradictory cases near the classification border is also in accordance with the main results discovered from an autonomous exploratory learning system (i.e. without an external teacher). Winston [43] first drew attention to the critical role of *near-miss* training examples. Recently the Protos system, developed by Porter et al. [33], also used near-miss cases to learn an examplar difference before matching a new case to an existing examplar.

6. DISCUSSION AND CONCLUSIONS

In this article, we have presented an algorithm for interactive induction and evaluated the performance of the implemented system in learning goemetric patterns and blackjack game strategies. The algorithm is simple but efficient.

The contribution of this work is two-fold. First, it suggests an effective way of integrating manual knowledge acquisition and inductive learning to achieve a more accurate knowledge base. Second, it provides a new way of creating near-miss training examples and learning from these examples. This allows critical knowledge to be learned without a large number of training cases.

Further research following this includes replacing ID3 with other learning methods, testing in other domains and exploring the handling of certainty factors in interactive induction. The current approach can also be extended to acquire knowledge from multiple experts. The main difficulty for knowledge acquisition from multiple experts is how to combine their expertise and find out any inconsistencies systematically. Due to different subjective opinions, two competitive domain experts may disagree with each other occasionally. Our approach should be helpful in identifying conflicting rules or cases during the knowledge acquisition process.

REFERENCES

1. Angluin, D. (1988). Queries and concept learning. *Machine Learning*, **2**, 319–342.
2. Bainbridge, L. (1986). Asking questions and accessing knowledge. In *Future computing systems*. New York: Elsevier.
3. Boose, J. (1984). Personal construct theory and the transfer of human expertise. *Proc. of the Nat'l Conf on Artificial Intelligence*, Austin, TX.
4. Boose, J. & Gaines, B. R. (1989). Knowledge acquisition for knowledge-based system: Notes on the state-of-the-art. *Machine Learning*, **4**, 131–143.
5. Braun, H. & Chandler, J. S. (1987). Predicting stock market behavior through rule induction: an application of the learning-from-example approach. *Decision Sciences*, **18**, 415–429.
6. Buchanan, B. G. et al. (1983). Constructing an expert system. In F. Hayes-Roth, D. Waterman & D. Lenat (Eds), *Building expert systems*. Reading, Addison–Wesley.
7. Buntine, W. & Stirling, D. (1990). Interactive induction. In J. E. Hayes-Michie, D. Michie & E. Tyugu (Eds), *Machine intelligence*, Vol. 12, pp. 121–138, Oxford: Oxford University Press.
8. Carter, C. & Catlett, J. (1987). Assessing credit card applications using machine learning. *IEEE Expert*, **2**, 71–79.
9. Cohn, D., Atlas, L. & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, **15**, 201–221.
10. Chung, H. M. & Silver, M. S. (1992). Rule-based expert systems and linear models: An empirical comparison of learning-by-example methods. *Decision Science*, **23**, 687–707.
11. Eriksson, H. (1992). A survey of knowledge acquisition techniques and tools and their relationship to software engineering. *Journal of System and Software*, **19**, 97–107.
12. Evans, B. & Fisher, D. (1994). Overcoming process delays with decision tree induction. *IEEE Expert*, 60–66.
13. Feigenbaum, E. A. (1977). The art of artificial intelligence: Themes and case studies of knowledge engineering. *International Joint Conference on Artificial Intelligence*, Vol. 5, pp. 1014–1029.
14. Gaines, B. R. (1988). Knowledge acquisition: Development and advances.. In M. D. Oliff, (Ed.) *Expert system and intelligent programming*. New York: Elsevier.
15. Goldman, S. A. & Solan, R. H. (1994). The power of self-directed learning. *Machine Learning*, **14**, 271–294.
16. Hart, A. (1992). *Knowledge acquisition for expert systems*, 2nd edn. New York: McGraw–Hill.
17. Hayes-Roth, F., Waterman, D. A. & Lenat, D. (1983). *Building expert systems*. Reading MA: Addison–Wesley.
18. Jeng, B. & Weyuker, E. (1994). A simplified approach to domain testing. *ACM Transactions on Software Engineering and Methodology*, **3**, 254–270.
19. Jeng, B., Liang, T. P. & Jeng, Y. M. FILM: A fuzzy inductive learning method for automatic knowledge acquisition. Forthcoming in *Decision Support Systems*.
20. Johnson, P. E. (1983). What kind of expert should a system be? *Journal of Medicine and Philosophy*, **8**, 77–97.
21. Kelly, G. A. (1955). *The psychology of personal constructs.*, New York: Norton.
22. Liang, T. P. (1992). A composite approach to inducing knowledge for expert systems design. *Management Science*, **38**, 1–17.

23. Marchand, A., VanLente, F. & Galen, R. (1983). The assessment of laboratory tests in the diagnosis of acute appendicitis. *American Journal of Clinical Pathology*, **80:3**, 369–374.
24. Marcus, S. (1988). *Automating knowledge acquisition for expert systems*. Boston, MA: Kluwer.
25. McGraw, K. L. & Harbison-Briggs, K. (1989). *Knowledge acquisition: Principles and guidelines*. Englewood Cliffs, NJ: Prentice–Hall.
26. Messier, W. F. & Hansen, J. V. (1988). Inducing rules for expert system development: an example using default and bankruptcy data. *Management Science*, **34**, 1403–1415.
27. Michalski, R. S. & Chilausky, R. L. (1980). Knowledge acquisition by encoding expert rules versus computer induction from examples: A case study involving soybean pathology. *Int. J. Man–Machine Study*, **12**, 63–87.
28. Michie, D. (Ed.) (1984). *Introductory readings in expert systems*. New York: Breach.
29. Michalski, R., Mozetic, I., Hong, J. & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. of the 5th Annual Nat'l Conference on Artificial Intelligence*, pp. 1041–1045, Philadelphia, PA.
30. Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, **18**, 203–226.
31. Parsaye, K. (1988). Acquiring and verifying knowledge automatically. *AI Expert*, pp. 48–63.
32. Paterson, A. & Niblett, T. (1982). *ACLS user manual*. Scotland: Intelligence Terminal Ltd.
33. Porter, B.W., Bareiss, R. & Holte, R.C. (1990). Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, **45**, 229–263.
34. Quinlan, J. R. (1979). Discovering rules from large collections of examples: a case study. In D. Michie (Ed.), *Expert systems in the micro electronic age*. Scotland: Edinburgh University Press.
35. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, **1**, 81–106.
36. Rendell, L. A. (1986). A general framework for induction and a study of selective induction. *Machine Learning*, **1**, 177–220.
37. Ruff, R. A. & Dietterich, T. G. (1989). What good are experiments. *Proc. of the Sixth International Workshop on Machine Learning*, pp. 109–112, Ithaca, New York.
38. Scott, P. D. & Markovitch, S. (1993). Èxperience selection and problem choice in an exploratory learning system. *Machine Learning*, **12**, 49–67.
39. Shaw, M. J. & Gentry, J. A. (1988). Using an expert system with inductive learning to evaluate business loans. *Financial Management*, **17**, 45–56.
40. Turban, E. (1992). *Expert systems and applied artificial intelligence*. New York: Macmillan.
41. Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning*, **4**, 161–186.
42. Wardle, A. & Wardle, L. (1978). Computer aided diagnosis—a review of research. *Meth. Inform. Med.*, **17**, 15–28.
43. Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw–Hill.