Machine Learning by Imitating Human Learning

KUO-CHIN CHANG
Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, 30050, R.O.C.

TZUNG-PEI HONG
Department of Information Management, Kaohsiung Polytechnic Institute, Kaohsiung County, Taiwan, 84008, R.O.C. (e-mail: tphong@nas05.kpi.edu.tw)

SHIAN-SHYONG TSENG
Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, 30050, R.O.C. (e-mail: sstseng@cis.nctu.edu.tw)

Abstract. Learning general concepts in imperfect environments is difficult since training instances often include noisy data, inconclusive data, incomplete data, unknown attributes, unknown attribute values and other barriers to effective learning. It is well known that people can learn effectively in imperfect environments, and can manage to process very large amounts of data. Imitating human learning behavior therefore provides a useful model for machine learning in real-world applications. This paper proposes a new, more effective way to represent imperfect training instances and rules, and based on the new representation, a Human-Like Learning (HULL) algorithm for incrementally learning concepts well in imperfect training environments. Several examples are given to make the algorithm clearer. Finally, experimental results are presented that show the proposed learning algorithm works well in imperfect learning environments.

Key words. Machine learning, human learning, rule, training instance, specialize, generalize.

1. Introduction

One of the most active research areas of machine learning in recent years has continued to be symbolic empirical learning. This area is concerned with learning through symbolic usage and without much prior domain knowledge. The most common topic in symbolic empirical learning is learning from examples. The question of how general concepts are learned from a set of training instances has become increasingly important to artificial intelligence researchers constructing knowledge-based systems [13,16,17]. Given a set of examples and counterexamples of a concept, the learning program tries to induce general concept descriptions that describe the positive training instances and exclude the counterexamples [19]. In real learning environments, training sets provided by experts, teachers, or users are usually large and imperfect. Different training instances may have different degrees of importance and precision. Inconclusive data [22,24], incorrect data [2,9,14], incomplete data [11], unknown attributes, or unknown attribute-values [20] may also exist.

Machine learning is especially useful for knowledge acquisition in expert systems. Expert systems are designed to represent and apply knowledge of specific areas of expertise to solve problems. Expert systems are usually knowledge-intensive; the process of acquiring the necessary knowledge by interviewing domain specialists is then tedious and difficult since the experts are usually unaware of how to effectively express their knowledge. This often causes knowledge acquired to be incomplete, inconsistent, or irrelevant. As an alternative, methods for inducing concept descriptions from examples have been proven useful in easing the bottleneck of knowledge acquisition for constructing-expert systems. In addition to learning from examples, there are still many other learning methods, such as learning from explanation, learning by discovery, learning by analogy, in the field of machine learning.

In particular, researchers are studying ways of building efficient and reliable learning models and algorithms for improving the performance of knowledge acquisition and ensuring the accuracy of obtained knowledge [1,10,12,15,18]. Among the strategies for learning from examples, ID3 and PRISM are two of the most famous. ID3 tries to form decision trees from a set of training instances by applying Information Theory. ID3 uses the heuristics of minimising the entropy in determining which attribute should be selected next in a decision tree. The decision tree can also be easily transformed into rules by tracing a path from the root to a leaf. The PRISM learning algorithm applies the idea of information gain instead of entropy in ID3 for inducing rules which are modular. PRISM concentrates on finding only relevant attribute-value pairs; while ID3 is concerned with finding only the attribute which is the most relevant overall, even though some values of that attribute may be irrelevant. These two algorithms, however, lack the capability of incremental learning, and all the training instances must be kept. Also, their power to manage incomplete or inconclusive data is weak.

It is well known that people can effectively learn in imperfect environments, and can process very large amounts of data. The power of machine learning is currently still far weaker than that of human being. However, studying human learning models can provide some interesting insight on machine learning. Imitating the intelligent human learning behavior provides a useful model for machine learning in real-world applications [12,15,23,25]. In this paper, we utilize some characteristics of human learning models to design a new learning-from-example algorithm. This new algorithm is not expected to compete the human power (currently, this is impossible); instead, it is expected to overweigh the existing learning algorithms, such as ID3 and PRISM, in some aspects through the study of human learning models.

This paper is organized as follows. The characteristics of human learning behavior are first reviewed. A new, more effective way to represent imperfect training instances and rules is proposed. Based on the new representation and the characteristics of human learning behavior, a Human-Like Learning algorithm (HULL) is then proposed to solve complex learning problems. This learning algorithm can incrementally learn concepts well and simulate the action of human memory (using small amount of memory space but keeping a large body of learned knowledge). It can also successfully manage noisy data, inconclusive data, incomplete data, unknown attributes and unknown attribute values. Several examples are also given to make the algorithm clearer. Finally, experimental results are presented that show the proposed learning algorithm works well in imperfect learning environments.

2. Characteristics of Human Learning

For convenience, the behavior of human learning is discussed in two parts [3,5,7,8]:

(i) learning behavior, and

(ii) memory behavior.

Since they are very complex, not all characteristics of human learning will be discussed here; instead, only those considered in the HULL algorithm will be stated.

2.1. CHARACTERISTICS OF LEARNING BEHAVIOR

Learning behavior generally consists of the following characteristics:

1. *Thinking:* The main difference between human learning and machine learning is that humans can think in a general way to reorganize learned knowledge and to solve problems they have never encountered before [5,6,26,28,29].

2. *Incremental learning:* Human knowledge and intelligence grows through steady, life-long learning [3,7].

3. Transfer of learning: Learning behavior is deeply affected by the past experience. There are three kinds of influence [3,7]:

   a. positive transfer: past experiences have a positive effect on new learning;
   b. negative transfer: past experiences have a negative effect on new learning;
   c. zero transfer: past experiences have little or no effect on new learning.

4. *Recognition*: Recognition means people can judge whether a particular event was experienced on an earlier occasion [3,7,8].

5. *Recall*: Recall is the reproduction of the learned knowledge. For example, if an examination question asks you to give the causes of the Civil War, you must dredge them up out of your memory and formulate a response that convinces your instructor you know them [28,29].

6. *Learning order*: The order in which training data is presented is important to

the final learned results. Different orders of presentation of the same set of training instances may cause different learning results [3,7].

2.2. CHARACTERISTICS OF MEMORY BEHAVIOR

In addition to the characteristics of learning behavior, learning usually entails memory behavior. Two kinds of memories exist [3,7,8]:

(i) short-term memory (STM), and

(ii) long-term memory (LTM).

Short-term memory can be thought of as a buffer, keeping only a few of the latest training instances by FIFO (first-in first-out strategy). The capacity of STM is about 5 to 9 storage units. Long-term memory, on the other hand, can be thought of as a nearly unlimited capacity for storage. Items held in LTM are rarely lost (forgotten) unless they are interfered with by contradictory concepts. Generally speaking, memory behavior consists of the following characteristics:

1. *Meaning of data*: More meaningful or important training data is more easily remembered in LTM [3,7]. For example, the string "human-like-learning-algorithm" is easier to memorize than the string "mhtirogla-gninrael-ekil-namuh".
2. *Overlearning*: Overlearning means that when the same thing has been repeated many times, it is strongly impressed on the memory. When an item is overlearned beyond a certain threshold number of training times, more training on the same item has no significant effect [7,29].
3. *Retention interference*: Forgetting may take place since new learning activities could cause retention interference with old related experiences [7]. Two kinds of retention interference exist:
   a. *proactive inhibition* (PI): the newly learned concepts are inhibited by past experiences;
   b. *retroactive inhibition* (RI): past experiences are inhibited by newly learned concepts.

3. Representation

Some researchers believe that symbolic systems will likely continue to play a preeminent role in the field of machine learning. In symbolic systems, facts and knowledge are represented by high-level symbols, with the details to derive the symbols being omitted. The symbols may be derived by some preprocessing steps, such as the techniques in image processing, or may be given by users (these are beyond our discussion here). In this section, a new representation of training instances and rules is proposed to effectively manage imperfect training data and learned knowledge.

3.1. REPRESENTATION OF TRAINING INSTANCES

Each training instance is represented as:

(class type $\delta_c \Rightarrow$ attribute-value pair$_1$, attribute-value pair$_2$, ..., attribute-value pair$_m$, certainty-weight).

Here use of the symbol "$\Rightarrow$" to connect the class type and the attribute-value pairs is very natural. For example, it is true to say that a monkey has 2 legs; however, we can not say that any animal with 2 legs is a monkey since many animals have 2 legs. In the representation, *m* is a variable representing the number of attributes of the given training instance. Different training instances may have different numbers of attributes-value pairs (problem of unknown attributes). Besides, the certainty-weight denotes the reliability degree of this description. The value of the certainty-weight is 1 if the data are completely reliable, and is zero if the data are completely unreliable. The certainty-weight of a training instance may be assigned in the following ways:

1. If the given training data is only a small subset of a very large potential data base, the certainty-weight of each training instance may then be obtained by probability calculation;
2. The certainty-weight can be obtained if accurate statistical information is available;
3. The certainty-weight can be subjectively assigned by experts, teachers or experienced users.

EXAMPLE 1. Assume there are two training instances involving chairs. One describes a chair as having four legs and a back. It has a certainty-weight of 1 because in that person's experience, all chairs have these characteristics. The second training instance describes a chair as having four legs and no back. It has a certainty-weight of 0.70 because in the second person's experience, seven out of ten chairs have these characteristics. These training instances are represented as follows:

1. (class type = chair $\Rightarrow$ legs = 4, chair-back = yes, certainty-weight = 1.0).
2. (class type = chair $\Rightarrow$ legs = 4, chair-back = no, certainty-weight = 0.70).

3.2. REPRESENTATION OF RULES

As is common in studies of human learning, rules are used here to represent the learned knowledge. Since the HULL algorithm learns concepts from imperfect environments, a learned rule will not necessarily conclude to only one class. Instead, rules with the same descriptions may conclude to multiple classes with different possibilities. The representation of rules is defined as follows:

(attribute-value pairs $\rightarrow$ class type δ_c, certainty-weight, experience-count, fuzzy-probability).

Here "*not*" is allowed in the attribute-value pairs. As an example, "not legs = 4" is allowed. Besides, the certainty-weight denotes the certainty or reliability of the attribute-value pairs when the class is known to be δ_c (estimated from the certainty-weights of positive training instances covered by this rule). The experience-count denotes the number of positive training instances covered by this rule. Finally, the fuzzy-probability denotes the importance of the rule in the conflict rule set (where all rules with the same attribute-value pairs conclude to different class types). If the fuzzy-probability is 1, this rule does not contradict any other rule; otherwise, more than one rule with the same attribute-value pairs and different class types exist.

EXAMPLE 2. Assume only those two training instances given in Example 1 are available. The learned rule (the learning algorithm will be introduced later) is:

(legs = 4 $\rightarrow$ class type = chair, certainty-weight = 0.86, experience-count = 2, fuzzy-probability = 1).

This rule means that if a piece of furniture has 4 legs, it is then a chair since its fuzzy-probability is 1 (which means no other furniture has four legs), and that a chair with four legs has possibility of 0.86 (a chair may then have other descriptions) as learned from 2 positive training instances (experience-count = 2).

4. Notation

In order to clearly describe the HULL algorithm, symbols used in this algorithm are defined as :

IV: (I: Instance, V: Valid) A threshold used to judge whether a given training instance is valid. If the certainty-weight of a training instance is smaller than IV, this instance is not worth considering and is then judged invalid.

IT: (I: Instance, T: Typical) A threshold used to judge whether a training instance is typical. If the certainty-weight of a training instance is larger than or equal to IT, this instance is not only valid but also typical (IT > IV).

Data_memory: A storage space used to memorize training instances worthy of retention. Data_memory is composed of two parts: short-term-memory (DSTM) and long-term memory (DLTM). DSTM is used as a memory buffer, keeping the latest *m* valid but not typical training instances; DLTM on the other hand keeps all the typical training instances.

RO: (R: Rule, O: Overlearning) A threshold used to judge whether a rule is overlearned or not. If the number of positive training instances included in this rule is more than or equal to **RO**, then it is overlearned and is regarded as a typical rule.

RT: (R: Rule, T: Typical) A threshold used to judge whether a rule is typical by its certainty-weight. A rule is typical if its certainty-weight is larger than or equal to **RT** or its experience-count is larger than or equal to **RO**.

RR: (R: Rule, R: Retention) It is a threshold to judge whether a rule remembered in memory (both RLTM and RSTM) will be forgotten when this rule is interfered with by another rule; restated, if the fuzzy-probability of a rule is smaller than RR, this rule will then be forgotten.

Rule_memory: A storage space used to memorize learned rules which are worthy of retention. Rule_memory is composed of two parts: short_term memory (RSTM) and long_term memory (RLTM). RSTM, with a limited space, keeps the rules which are not typical; RLTM keeps the typical rules with a limitless space.

Rule_memory(c): A rule set where every rule has as its conclusion the class type "c" (in both RSTM and RLTM).

Rule_memory(~ c): A rule set where every rule does not have as its conclusion the class type "c" (in both RSTM and RLTM).

Inconclusive_rule_set(r): A rule set where all rules have the same attribute-value pairs as **r**, but with different class types.

W(x): The certainty-weight of an instance x or a rule x.

Count(r): The experience-count of a rule r.

Fp(r): The fuzzy-probability of a rule r.

5. Heuristic Functions

The HULL algorithm is designed for learning concepts in imperfect environments. It uses the following heuristic functions in estimating the values of certainty-weight and fuzzy-probability of each learned rule.

1. **Certainty_Weight_Generating Function: CWG(r)** is defined to estimate the certainty-weight of a new rule *r* when *r* is generated from several training instances:

$$\mathbf{CWG(r)} = \sqrt{\frac{1}{\mathbf{Count(r)}} \times \sum_{k=1}^{\mathrm{Count\ (r)}} \mathbf{w}(I_k)^2},$$

where I_k represents the k-th training instance deriving the rule r.

2. **Certainty_Weight_Reestimating Function: CWR(r, i)** is defined to estimate the new certainty-weight of an old rule r when a new instance i is included in r:

$$\mathbf{CWR(r, i)} = \sqrt{\frac{\mathbf{Count(r)} \times \mathbf{W(r)}^2 + \mathbf{W(i)}^2}{\mathbf{Count(r)} + 1}}$$

3. **Certainty_Weight_Excluding Function: CWE(r, i)** is defined to estimate the new certainty-weight of an old rule r when a new negative training instance i is excluded out of r as an exception by conjuncting the negation of the instance to the rule:

$$\mathbf{CWE(r, i)} = \sqrt{\frac{\mathbf{Count(r)} \times \mathbf{W(r)}^2 + (1 - \mathbf{W(i)})^2}{\mathbf{Count(r)}}}$$

4. **Fuzzy_Probability_Estimating Function: FPE(r$_j$)** is defined to estimate the fuzzy-probability of a rule r_j in the inconclusive_rule_set(r_j):

$$\mathbf{FPE(r_j)} = \frac{\mathbf{Count(r_j)} \times \mathbf{W(r_j)}}{\sum_{i=1}^{n} \mathbf{Count(r_i)} \times \mathbf{W(r_i)}},$$

where the inconclusive_rule_set(r_j) includes $r_1, r_2, \ldots, r_n$.

6. The Human-Like Learning Algorithm

The Human-Like Learning (HULL) algorithm, taking the characteristics of human learning into consideration, is described here. Note that in the rest of this section, **Data_memory** contains both **DSTM** and **DLTM**, and **Rule_memory** contains both **RSTM** and **RLTM**. The flowchart is shown in Figure 1.

The Hull algorithm first judges a new training instance by its certainty-weight in order to put it in the appropriate data memory. The algorithm then matches the rules in the rule memory with the instance to decide whether the rules are consistent with the new instance. Hull then adopts a corresponding operation (among the four possible operations) to maintain its consistency. Finally, the rule memory is reorganized to take away unimportant rules. The same procedure then repeats for another new training instance until all the instances are processed. The detailed algorithm is described below:

HULL ALGORITHM

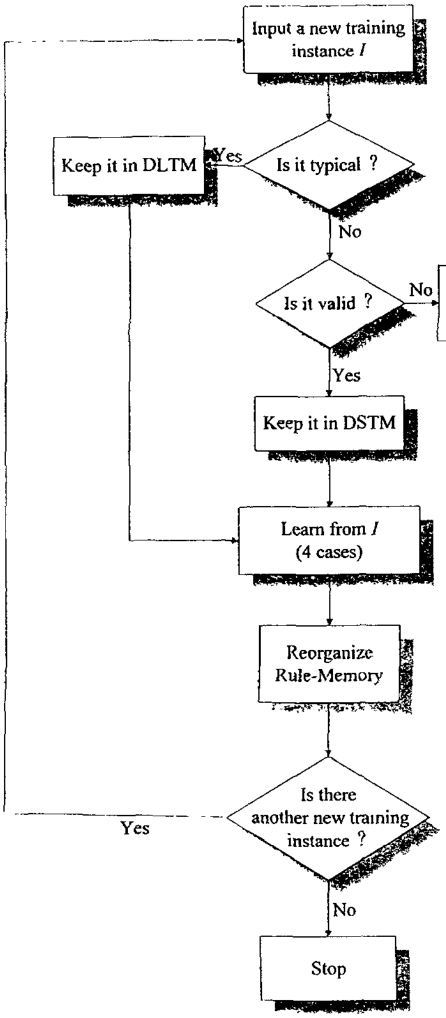


Fig. 1. The system flowchart.

INPUT: Old Rule_memory, old Data_memory, and a new training instance $\mathbf{I}_c$ of class **c**.

OUTPUT: New Rule_memory and new Data_memory through learning from $\mathbf{I}_c$.

STEP 1: Classify $\mathbf{I}_c$ as an invalid, valid, or typical training instance by its certainty-weight, with three cases possibly existing:

CASE 1: If $\mathbf{W(I_c) \geq IT}$ (typical), keep $\mathbf{I}_c$ in **DLTM**;

CASE 2: If $\mathbf{IT > W(I_c) \geq IV}$ (valid), keep $\mathbf{I}_c$ in **DSTM** (which keeps a fixed number of training instances using the **FIFO** strategy);

CASE 3: Otherwise (invalid), neglect $\mathbf{I}_c$ and stop the learning process.

*/*STEP 1 corresponds to the meaning of data */*

STEP 2: Learn from $\mathbf{I}_c$, with four cases possibly existing (Figure 2):

CASE 1: If exists a rule including $\mathbf{I}_c$ in **Rule_memory(c)** and exists no rule including $\mathbf{I}_c$ in **Rule_memory(~c)**, call PROCEDURE **Already_done_operation** to reestimate the related information of each rule including $\mathbf{I}_c$.

CASE 2: If exists no rule including $\mathbf{I}_c$ in **Rule_memory(c)** and in **Rule_memory(~c)**, call PROCEDURE **Including_operation** to include $\mathbf{I}_c$ in **Rule_memory(c)**.

CASE 3: If exist some rules in **Rule_memory(c)** including $\mathbf{I}_c$ and also exist some rules in **Rule_memory(~c)** including $\mathbf{I}_c$, call PROCEDURE **Excluding_operation** to exclude $\mathbf{I}_c$ out of **Rule_memory(~c)**.

CASE 4: If exists a rule in **Rule_memory(~c)** including $\mathbf{I}_c$, and exists no rule in **Rule_memory(c)** including $\mathbf{I}_c$, call PROCEDURE **Excluding_operation** to exclude $\mathbf{I}_c$ out of **Rule_memory(~c)**, and then call PROCEDURE **Including_operation** to include $\mathbf{I}_c$ in **Rule_memory(c)**.

/ STEP 2 corresponds to thinking, incrementally learning knowledge, and transfer of learning */*

STEP 3: For each altered or newly generated rule **r** in **Rule_memory**, use the heuristic function **FPE** to reestimate the fuzzy-probability of each rule in the **inconclusive_rule_set(r)**; if the reestimated fuzzy-probability of a rule is smaller than **RR**, remove it out of **Rule_memory** (this rule is then treated as a noisy rule and is forgotten).

/ STEP 3 corresponds to retention interference */*

STEP 4: Reallocate each altered or newly generated rule to its appropriate position in **Rule_memory** (**RSTM** or **RLTM**) according to its new certainty-weight and experience-count (the older rules in **RSTM** will be forgotten if the number of rules in **RSTM** is larger than the queue length of **RSTM**).

/ STEP 4 corresponds to memory organization */*

END ALGORITHM.

Procedures *Already_done_operation*, *Including_operation*, and *Excluding_operation* appearing in STEP 2 are presented here, each taking some characteristics of human learning into consideration. Procedure *Already_done_operation* is adopted when the rule_memory is consistent with the training instance. This procedure only updates the experience-count and certainty-weight of the rules which cover the new instance. The procedure is described below:

(1) PROCEDURE **Already_done_operation**

/ The **Rule_memory** has been consistent with I_c */*

STEP 1: For each rule $\mathbf{r}_c$ including $\mathbf{I}_c$, add 1 to its experience-count.

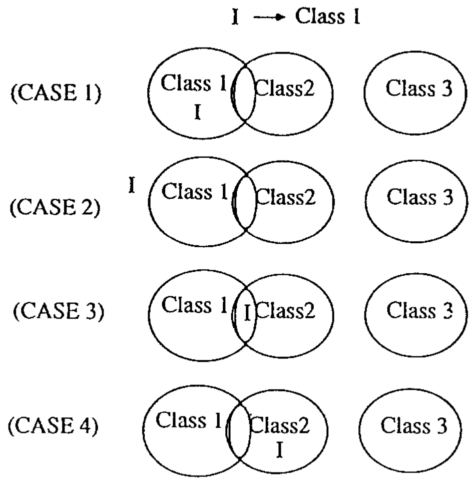


Fig. 2. Four possible cases in STEP 2.

```
    /* STEP 1 corresponds to overlearning */
    STEP 2: Use the heuristic function CWR($r_c$, $I_c$) to reestimate its new
            certainty-weight. If the new certainty-weight is larger than old
            W($r_c$), then update the certainty-weight of $r_c$ as the new certainty-
            weight; otherwise, old W($r_c$) remains as the new certainty-weight.
    /* STEP 2 corresponds to learning order */
END PROCEDURE
```

Procedure *Including_operation* (Figure 3) is adopted when the new training instance is not included by a rule of the same class. Old rules are then first tried to cover this new instance by the generalization process since this method will cause no increase of the rule number. If the above method cannot work, then a new general rule based on this instance is generated.

The procedure is described below:

```
(2) PROCEDURE Including_operation
    /* Rule_memory(c) must be altered to include $I_c$ */
    STEP 1: If a rule $r_c$ in Rule_memory(c) can be minimally generalized to
            include $I_c$ and no training instance of class ~c in Data_memory,
            and to contradict no rule in Rule_memory(~c), then minimally
            generalize $r_c$; call SUBPROCEDURE Reestimating_information to
            reestimate the related information of $r_c$, and stop this procedure;
            otherwise, do STEP 2.
            /* STEP 1 uses the generalization strategy to include $I_c$ */
    STEP 2: Call SUBPROCEDURE Generating_new_rule to generate a new
            rule Tr covering $I_c$.
```

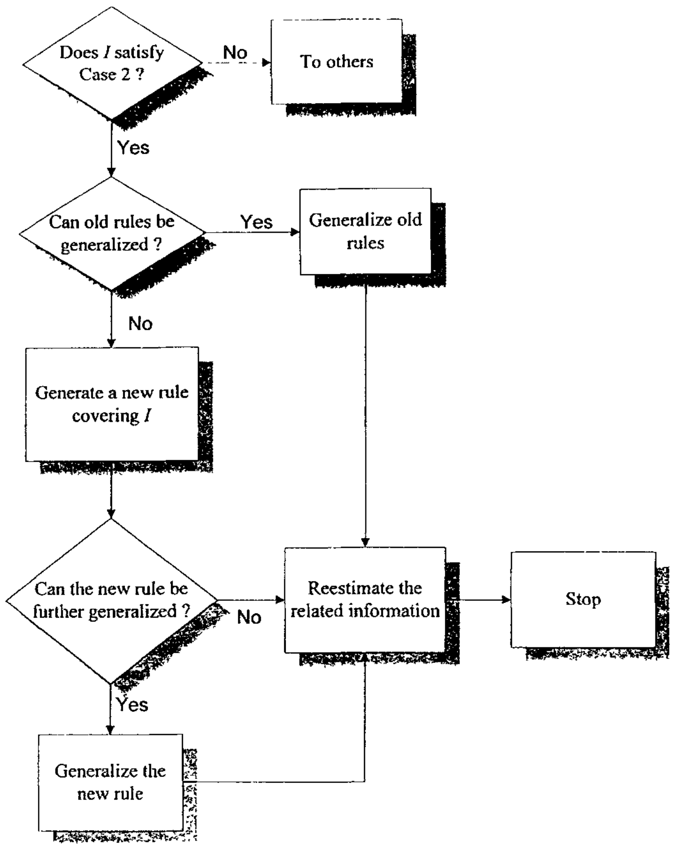

Does *I* satisfy Case 2 ?
No
To others
Yes
Can old rules be generalized ?
Yes
Generalize old rules
No
Generate a new rule covering *I*
Can the new rule be further generalized ?
No
Reestimate the related information
Stop
Yes
Generalize the new rule

Fig. 3. The flowchart for the procedure Including_Operation.

```
STEP 3: If Rule Tr can be minimally generalized to include some other
        positive training instances and no negative training instances in
        Data_memory, and to contradict no rule in Rule_memory(~c), then
        do the following substeps:
        SUBSTEP 1: Minimally generalize the rule Tr with as many other
                   positive training instances in Data_memory as possible
                   (to avoid causing any contradiction).
        SUBSTEP 2: Call SUBPROCEDURE Reestimating_information to
                   reestimate the related information of Tr.
                   /* STEPs 1, 2, 3 correspond to transfer of learning */
STEP 4: Add the new rule Tr into Rule_memory (if Tr is typical, keep it in
        RLTM; else, keep it in RSTM).
        /* STEP 4 corresponds to memory organization */
END PROCEDURE
```

Procedure *Excluding_operation* (Figure 4) is adopted when the new training

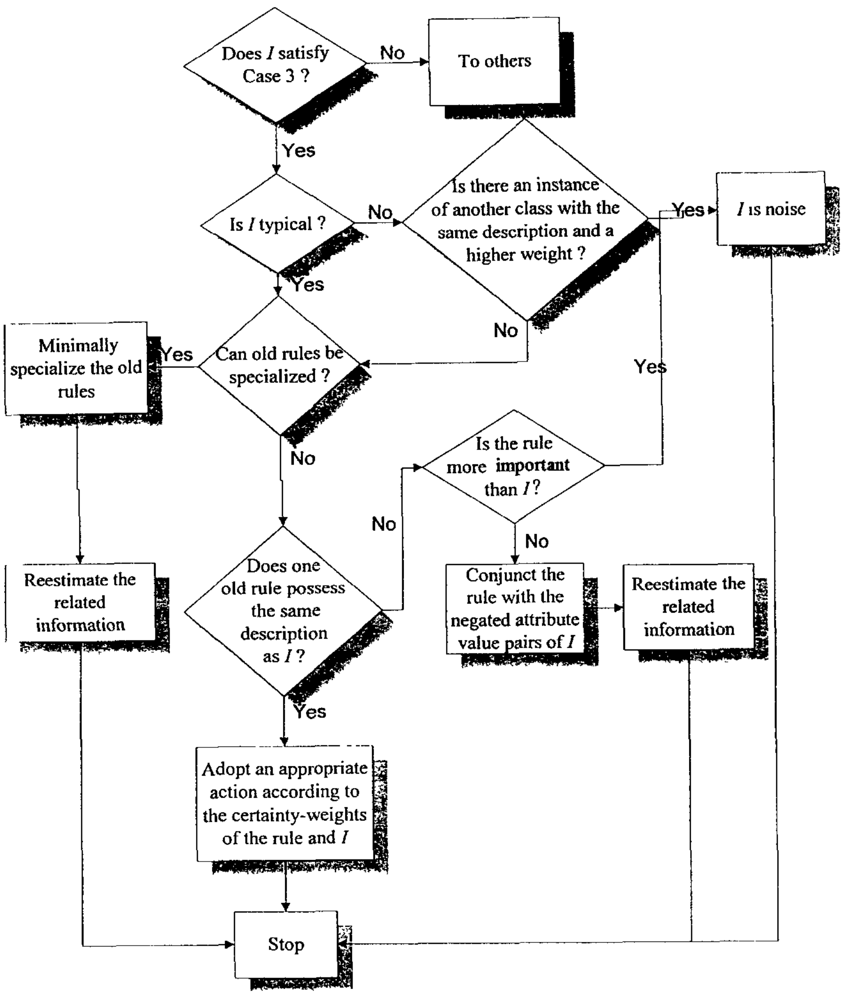

Does I satisfy Case 3 ?
No
To others
Yes
Is I typical ?
No
Is there an instance of another class with the same description and a higher weight ?
Yes
I is noise
Yes
Minimally specialize the old rules
Yes
Can old rules be specialized ?
No
Yes
No
Is the rule more important than I?
No
No
Reestimate the related information
Does one old rule possess the same description as I?
Conjunct the rule with the negated attribute value pairs of I
Reestimate the related information
Yes
Adopt an appropriate action according to the certainty-weights of the rule and I
Stop

Fig. 4. The flowchart for the procedure Excluding_Operation.

instance is included by a rule of another class, thus causing inconsistency. Old inconsistent rules are then first tried to exclude this new instance by the specialization process. If the above method cannot work, Hull checks whether the rules and the instance have the same attribute-value pairs. If they have the same attribute-value pairs, specializing the rules to exclude the instance is impossible. Some alternatives are thus taken according to the certainty-weights and the typical property of the rules and the instance. If the rules and the instance have different attribute-value pairs, then the instance is negated and conjuncted to the rules as an exception.

The procedure is described below:

(3) PROCEDURE **Excluding_operation**

```
/* Rule_memory(c) has been consistent, but Rule_memory($\sim c$) must be altered
to exclude $I_c$ */
STEP 1: If $I_c$ is not typical, then recall in Data_memory an instance $I_{\sim c}$ with
        the highest certainty-weight (among those with the same attribute-
        value pairs as $I_c$ but with different class types). If $W(I_{\sim c}) > W(I_c)$,
        then ignore $I_c$ (regarded as a noisy instance), and stop the whole
        learning process; else, do STEP 2.
        /* STEP 1 corresponds to proactive inhibition, negative transfer,
                    recall, and meaning of data */
STEP 2: Process in Rule_memory($\sim c$) each rule $r_{\sim c}$ including $I_c$ in the order
        of decreasing certainty-weights, with three cases possibly existing:
CASE 1: If $r_{\sim c}$ can be minimally specialized to exclude $I_c$ and to contradict no
        rule in Rule_memory(c), then minimally specialize $r_{\sim c}$ and call SUB-
        PROCEDURE Reestimating_information to reestimate the related
        information of the specialized rules.
        /* CASE 1 corresponds to retroactive inhibition */
    CASE 2: If $r_{\sim c}$ and $I_c$ are described by the same attribute-value pairs, four
            subcases possibly exist:
            SUBCASE 1: If $W(I_c) < W(r_{\sim c})$ and $I_c$ is not a typical instance,
                       then ignore the training instance $I_c$ (regarded as a
                       noisy instance), and stop the whole learning process.
            SUBCASE 2: If $W(I_c) < W(r_{\sim c})$ and $I_c$ is a typical instance, mark
                       $r_{\sim c}$ as an inconclusive rule.
            SUBCASE 3: If $W(I_c) \geq W(r_{\sim c})$, and $r_{\sim c}$ is not a typical rule, then
                       forget $r_{\sim c}$;
            SUBCASE 4: If $W(I_c) \geq W(r_{\sim c})$, and $r_{\sim c}$ is a typical rule, mark $r_{\sim c}$
                       as an inconclusive rule.
            /* CASE 2 corresponds to meaning of data and retroactive inhibi-
                        tion */
    CASE 3: Otherwise, alter $r_{\sim c}$ according to the certainty-weight of $I_c$, with
            two subcases possibly existing:
            SUBCASE 1: if $W(I_c) \geq W(r_{\sim c})$, then do the following substeps:
            SUBSTEP 1: Conjunct $r_{\sim c}$ with negated attribute value pairs of $I_c$
                       into a new rule $r_{\sim c}$.
            SUBSTEP 2: Use the heuristic function CWE($r_{\sim c}$, $I_c$) to reestimate
                       the certainty-weight of the new rule $r_{\sim c}$.
            /* $I_c$ is considered as an exception of $r_{\sim c}$ */
            SUBCASE 2: if $W(I_c) < W(r_{\sim c})$, then ignore the training instance
                       $I_c$ (regarded as a noisy instance).
            /* CASE 3 corresponds to meaning of data and retroactive inhibi-
                        tion */
END PROCEDURE
```

Subprocedures *Generating_new_rule* and *Reestimating_information* used above are defined here. Subprocedure *Generating_new_rule* generates a new rule to include the new instance and calculates its experience-count and certainty-weight (when the existing old rules cannot be generalized to cover the instance). The subprocedure is described below:

```
(i) SUBPROCEDURE Generating_new_rule
    /* A new rule Tr is generated to include $I_c$ */
    STEP 1: Recall in Data_memory all training instances with the same attribute-
            value pairs and class type as $I_c$.
            /* STEP 1 corresponds to recall */
    STEP 2: Generate a new rule Tr, with its attribute-value pairs the same as
            those of $I_c$, and its experience-count equal to the number of recalled
            training instances (including $I_c$).
            /* STEP 2 corresponds to positive transfer */
    STEP 3: Use the heuristic function CWG(Tr) to estimate the certainty-weight
            of Tr; set its fuzzy-probability to be 1.
END SUBPROCEDURE
```

Subprocedure *Reestimating_information* is used to recalculate the experience-count, certainty-weight, and fuzzy-probability of a rule when the rule memory is altered. The subprocedure is described below:

```
(ii) SUBPROCEDURE Reestimating_information(r)
     /* The values of experience-count, certainty-weight, and fuzzy-probability of
     Rule r is reestimated by this procedure */
     STEP 1: Set the experience-count of Rule r to be the number of training
             instances in Data_memory covered by this rule.
             /* STEP 1 corresponds to recognition and overlearning */
     STEP 2: Use the heuristic function CWG(r) to estimate the new certainty-
             weight of r.
     STEP 3: If the Inconclusive_rule_set(r) is not empty, then use the heuristic
             function FPE to estimate the new fuzzy-probability of each rule in
             Inconclusive_rule_set(r); otherwise, set the fuzzy-probability of r to
             be 1.
END SUBPROCEDURE
```

By the above procedures and subprocedures, the Hull algorithm can successfully process each new training instance and manage the data memory and the rule memory. The performance of the Hull algorithm is shown by experiments in Section 8. Below, an example is given to clearly illustrate the Hull algorithm.

7. Example

The data in Table I will be taken as the training set to show the reliability and the superiority of the HULL algorithm. Much uncertainty and noise exist in the

Table I. The training set

	Value of	attribute			Class	Certainty weight		Value of	attribute			Class	Certainty weight
No.	a,	b,	c,	d	δ_c	W	No.	a,	b,	c,	d, e	δ_c	W
1.	1,	1,	1,	1	1	0.85	16.	4,	2,	2,	1	3	0.75
2.	2,	1,	2,	1	1	0.75	17.	1,	3,	2,	1	3	0.70
3.	1,	2,	1,	1	1	0.80	18.	1,	2,	2,	1	3	0.75
4	3,	1,	1,	2	1	0.60	19.	3,	2,	1,	1	1	0.70
5.	2,	3,	3,	1	1	0.60	20.	4,	4,	3	2	2	0.45
6.	2,	2,	1,	2	2	0.60	21.	1.	4,	2,	1	3	0.60
7.	4.	2,	2,	1	2	0.76	22.	1,	2,	2,	1	1	0.80
8.	2,	2,	3,	2	2	0.60	23.	4,	2,	2,	1	2	0.72
9.	3,	4,	1,	1	2	0.70	24.	1,	2,	2,	1	3	0.72
10.	3,	4,	3,	2	2	0.90	25.	4,	2,	2,	1	3	0.78
11.	1,	3,	3,	2	3	0.65	26.	4,	2,	2,	1, 2	2	0.74
12.	3,	1,	1,	2	3	0.70	27.	1,	4,	2,	1	1	0.75
13.	1.	3,	3,	2	3	0.70	28.	1,	2,	2,	1, 1	1	0.75
14.	3,	3,	3,	3	3	0.70	29.	3,	4,	3,	1	2	0.72
15.	3,	3,	2,	1	3	0.80	30.	1,	3,	2,	2	3	0.70

training set. For example, training instances 7, 16, 23, and 25 are inconclusive and uncertain; training instances 26 and 28 consist of attribute *e* (the others don't); training instance 20 may be noisy since its certainty-weight is quite low. Assume the values of IT, IV, RO, RT and RR in the HULL algorithm are set respectively at 0.85, 0.5, 2, 0.8 and 0.1, the queue length in both DSTM and RSTM is set at 7, and the initial memory is empty. Assume also the HULL algorithm processes the training instances in Table I in the original order. The following examples illustrate execution of the HULL algorithm.

EXAMPLE 3. Assume the 20th training instance ($\delta_2 \Rightarrow a_4 \cap b_4 \cap c_3 \cap d_2$, w = 0.45) occurs. This training instance will be neglected since its weight is lower than IV (0.5), and therefore the rule base is not changed.

EXAMPLE 4. Assume after the HULL algorithm processes the first 21 training instances the memory is as follows:

Data_memory.LTM:

1. $\delta_1 \Rightarrow a_1 \cap b_1 \cap c_1 \cap d_1$, w = 0.85
2. $\delta_2 \Rightarrow a_4 \cap b_3 \cap c_3 \cap d_2$, w = 0.90

Data_memory.STM:

1. $\delta_3 \Rightarrow a_3 \cap b_3 \cap c_3 \cap d_3$, w = 0.70
2. $\delta_3 \Rightarrow a_3 \cap b_3 \cap c_2 \cap d_1$, w = 0.80
3. $\delta_3 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1$, w = 0.75
4. $\delta_3 \Rightarrow a_1 \cap b_3 \cap c_2 \cap d_1$, w = 0.70
5. $\delta_3 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, w = 0.75

6. $\delta_1 \Rightarrow a_3 \cap b_2 \cap c_1 \cap d_1$, $w = 0.70$
7. $\delta_3 \Rightarrow a_1 \cap b_4 \cap c_2 \cap d_1$, $w = 0.60$

Rule_memory.LTM:
1. $a_1 \cap b_1 \cap d_1 \rightarrow \delta_1$, $w = 0.82$, $c = 1$, $Fp = 1$
2. $a_2 \cap b_2 \cap d_2 \rightarrow \delta_2$, $w = 0.60$, $c = 2$, $Fp = 1$
3. $a_2 \cap d_1 \rightarrow \delta_1$, $w = 0.76$, $c = 2$, $Fp = 1$
4. $a_3 \cap b_4 \rightarrow \delta_2$, $w = 0.81$, $c = 2$, $Fp = 1$
5. $b_3 \rightarrow \delta_3$, $w = 0.71$, $c = 4$, $Fp = 1$
6. $a_1 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.72$, $c = 3$, $Fp = 1$
7. $c_1 \cap d_1 \rightarrow \delta_1$, $w = 0.78$, $c = 2$, $Fp = 1$

Rule_memory.STM:
1. $a_1 \cap b_1 \cap c_1 \rightarrow \delta_1$, $w = 0.78$, $c = 1$, $Fp = 1$
2. $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2$, $w = 0.76$, $c = 1$, $Fp = 1$
3. $a_3 \cap b_1 \cap c_1 \cap d_2 \rightarrow \delta_3$, $w = 0.70$, $c = 1$, $Fp = 1$

When Hull processes training instances 22 to 28, the memory is altered as shown in Figure 5. Explanation is given as follows.

(a) Assume the 22nd training instance ($\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, $w = 0.80$) occurs. Since it is a valid training instance according to its certainty-weight, it is put into the Data_Memory.STM. Since the queue length of Data_Memory.STM is limited to 7, the first training instance in the original Data_Memory.STM is then removed. Also, this training instance is inconsistent with the sixth rule ($a_1 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.72$, $c = 3$, $Fp = 1$) in Rule_memory.LTM, and is not included in any rule concluding to δ_1. It is then processed by the Case 4 of STEP 2 in HULL learning algorithm. For excluding the instance ($\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, $w = 0.80$), this rule is specialized into three rules containing (by the procedure of Excluding_Operation):

(i) $a_1 \cap b_1 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.75$, $c = 1$, $Fp = 1$,
(contradicting the first rule in Rule_memory.LTM)

(ii) $a_1 \cap b_3 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.70$, $c = 1$, $Fp = 1$,
(not a minimally-specialized rule)

(iii) $a_1 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.60$, $c = 1$, $Fp = 1$.

The rule ($a_1 \cap b_3 \cap c_2 \cap d_1 \rightarrow \delta_3$ $w = 0.70$, $c = 1$, $Fp = 1$) is covered by the fifth rule ($b_3 \rightarrow \delta_3$, $w = 0.71$, $c = 4$, $Fp = 1$) in Rule_memory.LTM, so it is neglected (not a minimally specialized rule). After the execution of the Excluding_Operation procedure, only rule (iii) is kept in Rule_memory.STM. Then the Including_Operation procedure is executed to include the new training instance. In STEP 2 of the Including_Operation procedure, a new rule ($a_1 \cap b_2 \cap$

Coming Instance	DLTM	DSTM	RLTM	RSTM
	1, 2	1, 2, 3, 4, 5, 6, 7	1, 2, 3, 4, 5, 6, 7	1, 2, 3
22	No change	Delete 1: $\delta_3 \Rightarrow a_3 \cap b_3 \cap c_3 \cap d_3, w=0.70$ Add 8: $\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1, w=0.80$	Delete 6: $a_1 \cap c_2 \cap d_1 \rightarrow \delta_3, w=0.72, c=3, Fp=1$ Add 8 (From RLTM 6): $a_1 b_2 \cap c_2 \cap d_1 \rightarrow \delta_1, w=0.80, c=1, Fp=1$	Add 4(From RLTM6): $a_1 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_3,$ $w=0.60, c=1, Fp=1$
	1, 2	2, 3, 4, 5, 6, 7, 8	1, 2, 3, 4, 5, 7, 8	1, 2, 3, 4
23	No change	Delete 2 $\delta_3 \Rightarrow a_3 \cap b_3 \cap c_2 \cap d_1, w=0.80$ Add 9 $\delta_2 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1, w=0.72$	Add 9 (From RSTM 2) $a_4 \cap b_2 \cap c_2 d_1 \rightarrow \delta_2, w=0.76, c=2, Fp=1$	Delete 2 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2,$ $w=0.76, c=1, Fp=1$
	1, 2	3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 7, 8, 9	1, 3, 4
24	No change	Delete 3 $\delta_3 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1, w=0.75$ Add 10 $\delta_3 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1, w=0.72$	Delete 8 $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1, w=0.80, c=1, Fp=1$ Add 10 and 11 $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1, w=0.80, c=1, Fp=0.35$ $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3, w=0.74, c=2, Fp=0.65$	No change
	1, 2	4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 7, 9, 10, 11	1, 3, 4
25	No change	Delete 4 $\delta_3 \Rightarrow a_1 \cap b_3 \cap c_2 \cap d_1, w=0.70$ Add 11 $\delta_3 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1, w=0.78$	Delete 9 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2, w=0.76, c=2, Fp=1$ Add 12 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3, w=0.76, c=2, Fp=0.66$	Add 5 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3,$ $w=0.78, c=1, Fp=0.34$
	1, 2	5, 6, 7, 8, 9, 10, 11	1, 2, 3, 4, 5, 7, 10, 11, 12	1, 3, 4, 5
26	No change	Delete 5 $\delta_3 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1, w=0.75$ Add 12 $\delta_2 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1 \cap e_2, w=0.74$	Delete 12 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2, w=0.76, c=2, Fp=0.66$ Add 13 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2, w=0.76, c=3, Fp=0.75$	Delete5 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3,$ $w=0.78, c=1, Fp=0.34$ Add 6 $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3,$ $w=0.78, c=1, Fp=0.25$
	1, 2	6, 7, 8, 9, 10, 11, 12	1, 2, 3, 4, 5, 7, 10, 11, 13	1, 3, 4, 6
27	No change	Delete 6 $\delta_1 \Rightarrow a_3 \cap b_2 \cap c_1 \cap d_1, w=0.70$ Add 13 $\delta_1 \Rightarrow a_1 \cap b_4 \cap c_2 \cap d_1, w=0.75$	No change	Delete4 $a_1 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_3,$ $w=0.60, c=1, Fp=1$ Add 7 $a_1 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_1,$ $w=0.75, c=1, Fp=1$
	1, 2	7, 8, 9, 10, 11, 12, 13	1, 2, 3, 4, 5, 7, 10, 11, 13	1, 3, 6, 7
28	No change	Delete 7 $\delta_3 \Rightarrow a_1 \cap b_4 \cap c_2 \cap d_1, w=0.60$ Add 14 $\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1 \cap e_1, w=0.75$	Delete 10,11 $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1, w=0.80, c=1, Fp=0.35$ $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3, w=0.74, c=2, Fp=0.65$ Add 14,15 $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1, w=0.80, c=2, Fp=0.52$ $a_1 \cap b_2 \cap c_2 \cap d_1 \cap \sim e_1 \rightarrow \delta_3,$ $w=0.74, c=2, Fp=0.48$	No change
	1, 2	8, 9, 10, 11, 12, 13, 14	1, 2, 3, 4, 5, 7, 13, 14, 15	1, 3, 6, 7

$c_2 \cap d_1 \rightarrow \delta_1$, $w = 0.80$, $c = 1$, $Fp = 1$) is then generated and put into the Rule_memory.LTM (its certainty weight is larger than RT).

(b) Assume the 23rd training instance ($\delta_2 = a_4 \cap b_2 \cap c_2 \cap d_1$, $w = 0.72$) occurs. The rule ($a_4 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_2$, $w = 0.76$, $c = 2$, $Fp = 1$) is reallocated from RSTM to RLTM (overlearning) and DTSM is altered.

(c) Assume the 24th training instance ($\delta_3 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, w = 0.72) occurs. This training instance is inconsistent with the rule ($a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1$, w = 0.80, c = 1, Fp = 1) in RLTM and the previous instance ($\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, w = 0.80) in DSTM. In STEP 1 of the Excluding_Operation procedure, the HULL algorithm will recall the previous training instance ($\delta_3 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, w = 0.75) kept in DSTM to construct a new rule ($a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0.80, c = 2, Fp = 1). The new rule is important enough to be remembered in RLTM (due to overlearning), such that even if the rule base is not consistent, the new rule is not considered as a noisy rule but rather an inconclusive rule. The algorithm will then keep both these rules at the same time, with their fuzzy-probability being recalculated (since their fuzzy-probabilities are larger than RR).

If this Rule_memory is used to classify an object with attribute-value pairs ($a_1 \cap b_2 \cap c_2 \cap d_1$), two possible classes (δ_1 and δ_3) may be concluded. There is about 65% evidence for supporting the unknown object to be class δ_3 and about 35% evidence for supporting to be the class δ_1. These rules also show that a training instance belonging to class δ_3 has the descriptions ($a_1 \cap b_2 \cap c_2 \cap d_1$) with the probability 0.74, and belonging to class δ_1 has the descriptions ($a_1 \cap b_2 \cap c_2 \cap d_1$) with the probability 0.80.

(d) Assume the 25th training instance ($\delta_3 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1$, w = 0.78) occurs as the next input instance. This instance is improperly included by the inconsistent rule ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2$, w = 0.76, c = 2, Fp = 1). Since the certainty-weight of the new instance ($\delta_3 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1$, w = 0.78) is larger than the previous training instance ($\delta_2 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1$, w = 0.72) in DSTM, this new instance can't be regarded as noise. Instead, it is used to generate a new rule ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0.78, c = 1, Fp = 0.34) to put in RSTM. Also, since the rule ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2$, w = 0.76, c = 2, Fp = 1) is in RLTM, it is thought of as an inconclusive rule instead of a noisy rule (SUBCASE 4 within CASE 2 in STEP 3 of the Excluding_Operation Procedure).

(e) Assume the 26th training instance ($\delta_2 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1 \cap e_2$, W = 0.74) occurs as the next input instance. This instance is simultaneously included by two inconclusive rules ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2$, w = 0.76, c = 2, Fp = 0.66) and ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0.78, c = 1, Fp = 0.34). The new instance can not be used to discriminate between these two inconclusive rules since its certainty-weight is lower than the rule ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0.78, c = 1, Fp = 0.34) (SUBCASE 2 within CASE 3 in STEP 3 of the Excluding_Operation Procedure). It can, however, increase the reliability of class δ_2 (PROCEDURE Already_done_operation).

(f) Assume the 27th training instance ($\delta_1 \Rightarrow a_1 \cap b_4 \cap c_2 \cap d_1$, w = 0.75) occurs as the next input instance. This instance is included by the inconsistent rule ($a_1 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0.60, c = 1, Fp = 1) and satisfies CASE 4 of STEP 2 in the HULL learning algorithm. Since this rule is not a typical rule and its weight is lower than the new instance, it is regarded as a noisy rule and is forgotten (SUBCASE 3 within CASE 2 of STEP 3 of the Excluding_Operation Procedure).

Also, a new rule is constructed to include the new training instance (Including_Operation Procedure).

(g) Assume the 28th training instance ($\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1 \cap e_1$, w = 0.75) occurs as the next input instance. As in (e), this instance is simultaneously included by two inconclusive rules ($a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1$, w = 0.80, c = 1, Fp = 0.35) and ($a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0.74, c = 2, Fp = 0.65). The certainty-weight of this training instance is, however, important enough to interfere with the inconclusive rule ($a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, w = 0,74, c = 2, Fp = 0.65) (SUB-CASE 1 within CASE 3 in STEP 3 of the Excluding_Operation Procedure).

The fifteenth rule remembered in RLTM shows the flexibility of the proposed learning algorithm for allowing negated notations. It means that if an object with attribute-value pairs is equal to ($a_1 \cap b_2 \cap c_2 \cap d_1$) and not equal to ($a_1 \cap b_2 \cap c_2 \cap d_1 \cap e_1$), it will be classified as class δ_3 with a 48% probability.

EXAMPLE 5. After processing all 30 training instances in Table I, the final learned result is:

Data_memory.LTM:

1. $\delta_1 \Rightarrow a_1 \cap b_1 \cap c_1 \cap d_1$, w = 0.85
2. $\delta_2 \Rightarrow a_4 \cap b_3 \cap c_3 \cap d_2$, w = 0.90

Data_memory.STM:

1. $\delta_3 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1$, w = 0.72
2. $\delta_3 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1$, w = 0.78
3. $\delta_2 \Rightarrow a_4 \cap b_2 \cap c_2 \cap d_1 \cap e_2$, w = 0.74
4. $\delta_1 \Rightarrow a_1 \cap b_4 \cap c_2 \cap d_1$, w = 0.75
5. $\delta_1 \Rightarrow a_1 \cap b_2 \cap c_2 \cap d_1 \cap e_1$, w = 0.75
6. $\delta_2 \Rightarrow a_3 \cap b_4 \cap c_3 \cap d_1$, w = 0.72
7. $\delta_3 \Rightarrow a_1 \cap b_3 \cap c_2 \cap d_2$, w = 0.70

Rule_memory.LTM:

1. $a_1 \cap b_1 \cap d_1 \rightarrow \delta_1$, w = 0.82, c = 1, Fp = 1
2. $a_2 \cap b_2 \cap d_2 \rightarrow \delta_2$, w = 0.60, c = 2, Fp = 1
3. $a_2 \cap d_1 \rightarrow \delta_1$, w = 0.76, c = 2, Fp = 1
4. $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2$, w = 0.76, c = 3, Fp = 0.75
5. $a_3 \cap b_4 \rightarrow \delta_2$, w = 0.81, c = 3, Fp = 1
6. $b_3 \rightarrow \delta_3$, w = 0.71, c = 6, Fp = 1
7. $c_1 \cap d_1 \rightarrow \delta_1$, w = 0.78, c = 2, Fp = 1
8. $a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1$, w = 0.80, c = 2, Fp = 0.52
9. $a_1 \cap b_2 \cap c_2 \cap d_1 \cap \sim(a_1 \cap b_2 \cap c_2 \cap d_1 \cap e_1) \rightarrow \delta_3$, w = 0.74, c = 2, Fp = 0.48

Rule_memory.STM:

1. $a_1 \cap b_1 \cap c_1 \rightarrow \delta_1$, w = 0.78, c = 1, Fp = 1

2. $a_3 \cap b_1 \cap c_1 \cap d_2 \rightarrow \delta_3$, $w = 0.70$, $c = 1$, $Fp = 1$
3. $a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.78$, $c = 1$, $Fp = 0.25$
4. $a_1 \cap b_4 \cap c_2 \cap d_1 \rightarrow \delta_1$, $w = 0.75$, $c = 1$, $Fp = 1$

If the result learned is used to recognize the data in Table I, the accuracy rate is 93% although two inconclusive rule sets may exist:

Set 1. ($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_2$, $w = 0.76$ $c = 3$ $F = 0.75$, and
($a_4 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_3$, $w = 0.78$, $c = 1$, $Fp = 0.25$).
Set 2. ($a_1 \cap b_2 \cap c_2 \cap d_1 \rightarrow \delta_1$, $w = 0.80$, $c = 2$, $Fp = 0.52$), and
($a_1 \cap b_2 \cap c_2 \cap d_1 \cap \sim e_1 \rightarrow \delta_3$, $w = 0.74$, $c = 2$, $Fp = 0.48$).

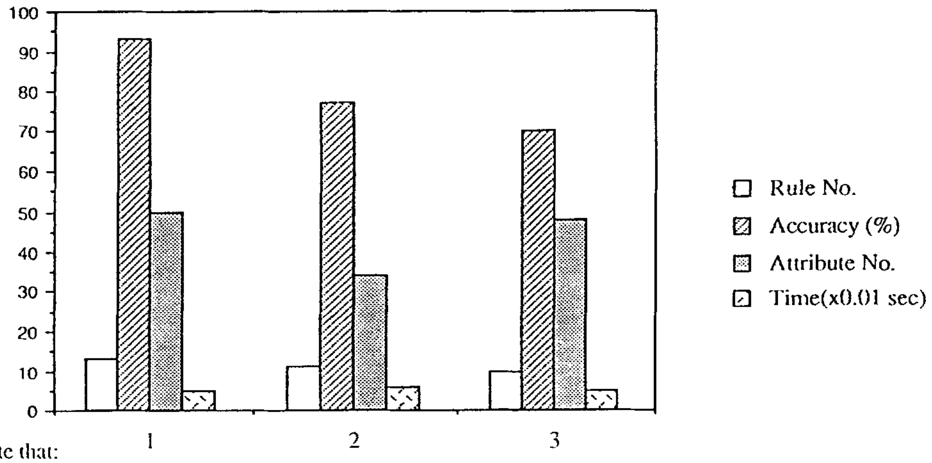
The above examples clearly show the HULL algorithm is a powerful incremental learning method suitable for imperfect environments.

8. Experiments

Experiments were conducted in Turbo-C on an IBM PC/AT to explore the consequence resulting from the parameters in the HULL learning algorithm. Every experiment was executed at least 500 times for various random orders of the data in Table I.

Figure 6 shows that the result learned is related to the order of training data (*i.e., the characteristic of learning order*), with the values of IT, IV, RO, RT, and RR being respectively set at 0.85, 0.5, 2, 0.8, and 0.1.

Figure 7 shows the consequence resulting from the parameter IT when the



Note that:
1 denotes the hightest recognition accuracy among 500 experiments.
2 denotes the average recognition accuracy of 500 experiments.
3 denotes the lowest recogintion accuracy among 500 experiments.

Fig. 6. comparison of random combinations of training instances.

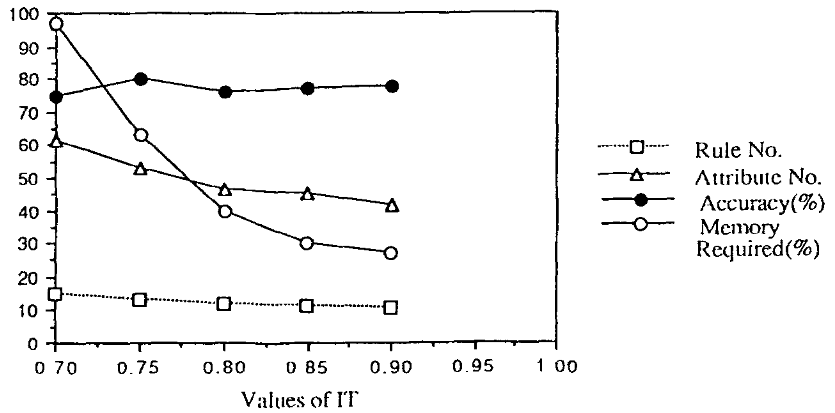


Fig. 7. Consequence of various values of IT.

values of IV, RO, RT, RR, and capacity of STM are set at 0.5, 2, 0.8, 0.1, and 7 respectively. Clearly, the memory capacity required for saving the training instances can be greatly reduced, and almost the same accuracy achieved, by increasing the value of IT.

Figure 8 shows the consequence resulting from the various lengths of DSTM when the values of IT, IV, RO, RT, and RR are set at 0.85, 0.5, 2, 0.8, and 0.1 respectively. When DSTM is larger than 8, increasing it will not have a signifcant effect. Therefore, using a suitable length of DSTM (instead of the number of the whole set of training instances) is sufficient for real applications.

Figure 9 shows the consequence resulting from the various values of RO when the values of IT, IV, RT, RR, and capacity of STM are set at 0.85, 0.5, 0.8, 0.1, and 7 respectively. If the threshold of overlearning is set larger, the number of

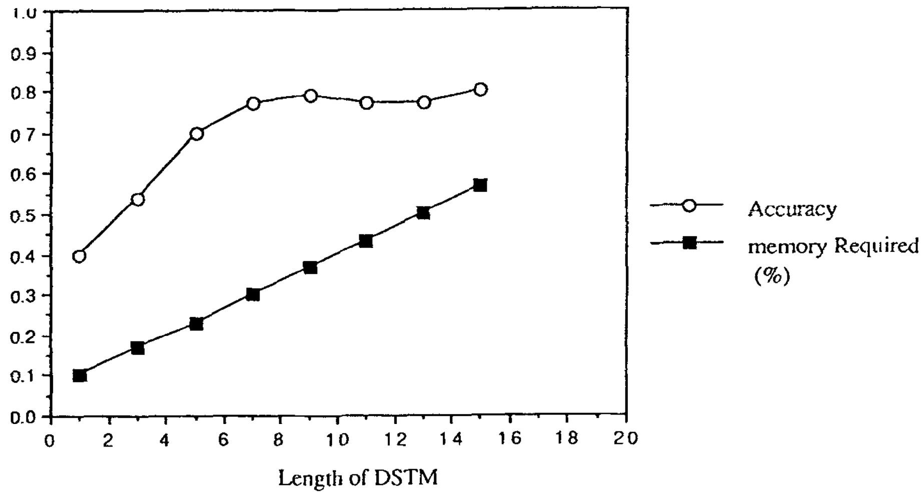


Fig. 8. Consequence of various lengths of DSTM.

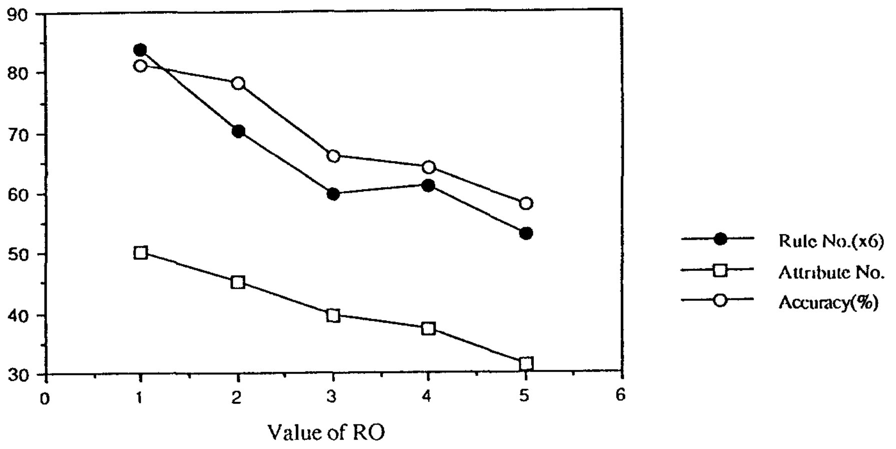


Fig. 9. Consequence of various values of RO.

rules will decrease, which also decreases accuracy (the learning algorithm with a small threshold of overlearning corresponds to a smart man).

Finally, Figure 10 shows the consequence resulting from the various values of RT when the values of IT, IV, RO, RR, and capacity of STM are set at 0.85, 0.5, 2, 0.1 and 7 respectively. The same as the consequence resulting from RO, if the value of RT is set larger, the number of rules will decrease, which also reduces the accuracy a little. The consequence, however, is quite minor.

Next, the data for fitting contact lenses [4] is used to compare the HULL algorithm with the ID3 [21] and the PRISM [4] algorithms in a noise-free environment. The results are in Tables II and III.

Finally, the domain for brain tumor diagnosis is used [27] as a noisy learning environment. The results are in Table IV.

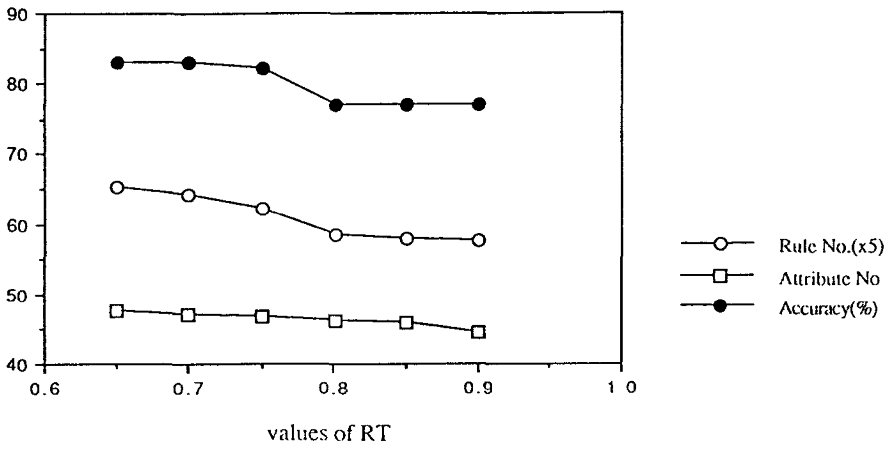


Fig. 10. Consequence of various values of RT.

Table II. Comparison of the HULL, PRISM and ID3 algorithms with no testing data

	Rule Number	Accuracy	Learning Time (0.01 Sec.)	Memory Required	Testing Data
HULL[1]	9.53	100%	10.48	1.00	No
HULL[2]	6.75	96%	8.95	0.29	No
ID3	9.00	100%	7.60	1.00	No
PRISM	9.00	100%	94.83	1.00	No

Note that: (i) Hull[1] denotes the length of DSTM being limitless. HULL[2] denotes DSTM length of only 7; (ii) RO = 2.

Table III. Comparison of the HULL, PRISM and ID3 algorithms with testing data

	Rule Number	Accuracy	Learning Time (0.01 Sec.)	Memory Required	Testing Data
HULL[1]	6.27	77%	6.18	1.00	33%
HULL[2]	5.03	67%	5.19	0.44	33%
ID3	6.86	67%	4.67	1.00	33%
PRISM	6.53	63%	83.93	1.00	33%

Table IV. Comparison of the HULL, PRISM and ID3 algorithms for brain tumor diagnosis

	Rule Number	Accuracy	Learning Time (0.01 Sec.)	Memory Required	Testing Data
HULL[1]	91.28	83.56%	12.63	1.00	25%
HULL[2]	78.76	81.96%	9.24	0.15	25%
ID3	87.06	74.89%	10.52	1.00	25%
PRISM	94.98	82.31%	76.08	1.00	25%

Note that: (i) Hull[1] denotes the length of DSTM being limitless. HULL[2] denotes DSTM length of only 12; (ii) RO = 2.

The above experiments show the HULL algorithm can not only solve some problems in imperfect environments, but can also do well in noise-free environments. From the experiments, the following points can be noted:

1. Currently, only a few learning algorithms can simultaneously manage noisy data, inconclusive data, incomplete data, unknown attributes, and unknown attribute values. The HULL learning algorithm can simultaneously manage all these cases.

2. Unlike ID3 and PRISM, the HULL learning algorithm can incrementally develop the knowledge base without the whole training instances kept. Also, the accuracy of the HULL learning algorithrn (HULL) is better than ID3 and PRISM, especially in noisy domains.

3. The HULL algorithm can easily make a trade-off between the memory space required and the accuracy. Larger memory space will cause higher accuracy, but will also take more learning time. The parameters of the HULL algorithm could be appropriately tuned to achieve this trade off.

4. Only a few strategies of learning from examples are connected to the human information processing models. This paper identifies the connection between the human cognitive behavior and machine learning models, and serves as a good example of experimental work in A.I. Especially, the HULL learning algorithm separates the knowledge base as short-term data, long_term data, short-term rules and long_term rules. HULL thus provides a new perspective on the management of the knowledge base.

The HULL learning algorithm is then suitable for deriving concept descriptions in imperfect learning environments.

Conclusions

In building a reliable and efficient learning model, it is very important that the representation is able to reasonably and effectively describe the meaning of the training data and the learned knowledge. In this paper, we have proposed a new representation of training instances and learned rules. This new representation has been shown to effectively describe the training instances and learned rules. Based on this representation and human learning behavior, we have also proposed the Human-Like Learning (HULL) algorithm in an attempt to solve the complex learning problems encountered in imperfect learning environments. Experimental results show the consequences resulting from the parameters in the HULL learning algorithm. In summary, imitating intelligent human learning behavior provides a good model for building an efficient and reliable learning system.

As mentioned, before, several other types of learning strategies, such as learning by analogy, learning by explanation, and learning by discovery, are also widely studied in the feld of machine learning. In the future, we will attempt to connect these strategies to the human learning models and propose new effcient learning algorithms. There is still much work to be done in this important area.

Acknowledgements

The authors would like to thank the anonymous referees for their very constructive comments. This research was supported by the National Science Council of the Republic of China under NSC81-0408-E009-16.

References

1. D.W. Aha and D. Kibler (1989), 'Noise-tolerant instance-based learning algorithm', *International Joint Conference on Artificial Intelligence*, pp. 794–799, Detroit, Michigan, USA.
2. D. Angluin (1988), 'Learning from noisy examples', *Machine Learning* 2, pp. 343–370.
3. A. Baddeley (1990), *Human Memory Theory and Practice*, Allyn and Bacon, Boston, USA.

4. J. Cendrowska (1987), PRISM : An algorithm for inducing modular rules', *International Journal of Man-Machine Studies* 27, pp. 349–370.
5. M. Cole and B. Means (1981), *Comparative Studies of How People Think: An Introduction*, Harvard University Press, Cambridge.
6. R. Dreistadt (1968), 'An analysis of the use of analogies and metaphors in science', *The Journal of Psychology*, pp. 97–116.
7. J.F. Hall (1989), *Learning and Memory*, Ed. 2nd, Allyn and Bacon, Boston, USA.
8. E.R. Hilgard and G.H. Bower (1975), *Theories of Learning*, Ed. 4th, Stanford Press, CA.
9. T.P. Hong (1992), A Study of Parallel Processing and Noise Management on Machine Learning, Ph.D. Thesis, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
10. T.P. Hong and S.S. Tseng (1994), 'Learning concepts in parallel based upon the strategy of version space', *IEEE Transactions on Knowledge and Data Engineering* 6-6, pp. 857–867.
11. G.J. Hwang and S.S. Tseng (1990), 'Building a multi-purpose medical system under uncertain and incomplete environment', *The Third IEEE Symposium on Computer-Based Medical Systems*, pp. 321–328, Chapel Hill, N.C.
12. R. Jones (1989), 'Learning to retrieve useful information for problem solving', *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, pp. 188–190.
13. Y. Kodratoff and R.S. Michalski (1990), *Machine Learning: An Artificial Intelligence Approach*, Vol. 3, Toiga, Palo Alto, CA.
14. Y. Kodratoff, M. Manago, and J. Blythe (1987), 'Generalization and noise', *International Journal of Man-Machine Studies* 27, pp. 181–204.
15. S. Markovitch and P.D. Scott (1988), 'The Role of Forgetting in Learning', *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, pp. 459–465.
16. R.S. Michalski, J.G. Carbonell and T.M. Mitchell (1983), *Machine Learning: An Artificial Intelligence Approach*, Vol. 1, Toiga, Palo Alto, CA.
17. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (1984), *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, Toiga, Palo Alto, CA.
18. D.A. Norman (1991), 'Approaches to the study of intelligence', *Artificial Intelligence* 49, pp. 327–346.
19. A.L. Ralescu and J.F. Baldwin (1989), 'Concept learning from examples and counter examples', *International Journal of Man-Machine Studies*, 30, pp. 329–354.
20. J.R. Quinlan (1989), 'Unknown attribute values in induction', *Proceedings of the Sixth International Workshop on Machine Learning*, pp. 164–168, Ithaca, New York.
21. J.R. Quinlan (1983), 'Learning efficient classification procedures and their application to chess end games', *Machine Learning: An Artificial Intelligence Approach*, Vol. 1. Toiga, Palo Alto, CA, pp. 463–482.
22. P.D. Scott and S. Markovitch (1989), 'Uncertainty based selection of learning experiences', *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, pp. 358–361.
23. J.C. Schlimmer, R.H. Granger, JR. (1986), 'Incremental learning from noisy data', *Machine Learning* 1, pp. 317–354.
24. S. Spangler, U.M. Fayyad and Ramasamy Uthurusamy (1989), 'Induction of decision trees from inconclusive data', *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, pp. 146–150.
25. M. Tambe and A. Newell (1988), 'Some chunks are expensive', *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI. pp. 451–458.
26. K. VanLehn (1989), 'Discovering problem solving strategies: What humans do and machines don't (yet)', *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, pp. 215–217.
27. C.H. Wang and S.S. Tseng (1990), 'A brain tumor diagnostic system with automatic learning abilities', *Proceedings of the Third IEEE Symposium on Computer-Based Medical Systems*, pp. 313–320, Chapel Hill, N.C.
28. P.G. Zimbardo (1980), *Essentials of Psychology and Life*, Ed. 10th, Scott, Foresman and Company Press, Dallas, Tex.
29. P.G. Zimbardo (1990), *Psychology and Life*, Ed. 12th, Scott, Foresman and Company Press, Dallas. Tex.