
Invited papers: Norbert Wiener Award Winners

CLIP3: Cover learning using integer programming

Krzysztof J. Cios, Daniel K. Wedding
University of Toledo, Toledo, Ohio, USA, and

Ning Liu
DANA Corporation, Ann Arbor, Michigan, USA

Introduction

The CLIP3 algorithm is a descendant of the CLILP2 algorithm (Cios and Liu, 1995a and 1995b). By incorporating the ideas from tree-based algorithms, CLIP3 is able to overcome the problems of overfitting. Overfitting is one form of false learning that occurs when the description of the training examples becomes so complicated, in order to cover all the examples, that the general concept is obscured or completely lost. Overfitting is further exacerbated by data that contain inaccuracies. Data sets that contain inaccuracies are referred to as noisy and algorithms that can generate accurate rules in the presence of noise are referred to as “noise-tolerant”. CLIP3 is noise-tolerant, since it separates the data into subsets of like data. This separation process, the “tree-growing”, moves noisy data into small isolated subsets and by using the tree-pruning technique (Cestnik *et al.*, 1987; Quinlan, 1983, 1990) the noisy data are removed from the data set.

After CLIP3 partitions training examples into noise-free subsets it then generates rules from each of the subsets. This is referred to as “covering” (Michalski, 1990; Michalski and Larson, 1978; Michalski *et al.*, 1986). Any algorithm that combines noise tolerance and covering is highly effective (Clark and Niblett, 1989; Michalski, 1990).

It was the authors’ intent to design a learning algorithm that combines the tree-based and rule-based families into a single, reliable, and easily coded algorithm. The result is the CLIP3 algorithm. CLIP3 is then compared with other machine learning algorithms to check for accuracy and efficiency. The MONK’s data (Thrun *et al.*, 1991) was chosen as the test bed for this comparison because a number of machine learning algorithms has been tested on it. The test results from the original MONK’s paper (Thrun *et al.*, 1991) and subsequent papers (Wu, 1993) serve as the basis of our comparison. In addition, CLIP3 and C4.5 were both run on the Breast Cancer Data (Bennet and Mangasarian, 1992;

Brief overview of integer linear programming

CLIP3 selects features and generates rules using an integer linear programming (IP) model. IP models have been used for years in the field of operations research mainly for resource allocation. This family of models is primarily used for minimization or maximization of a function that is subject to a large number of constraints.

One of the most famous IP applications is the travelling salesman problem. In this problem a path must be found between all the US capitals while minimizing the distance that the salesman must travel. A function must be minimized, in this case the sum of 50 values representing the distance between 50 state capitals, and is subjected to a list of constraints, in this case the distances between each of the state capitals and all other capitals; and each capital must be passed through once and only once. It is intuitive that a good solution is one that starts at one corner of the country and works its way across. A poor solution would be one that would force the salesman to “zig-zag” across the country passing by, but not stopping in, many capitals on his way to the next capitol. Obviously, an exhaustive search would reveal the best (a global minimum) path eventually, but that would not be practical in most applications.

To solve this type of problem effectively an IP is used. A very simple IP model in standard form is shown in Figure 1 (Ravindran *et al.*, 1987) to demonstrate the general structure of an IP model.

In most cases there are several solutions to an IP model. The solution that is arrived at depends on the method used to solve it. To find solutions to an IP model, there exist many polynomial algorithms (Chvatal, 1979; Hochbaum, 1982) and non-polynomial algorithms (Ravindran *et al.*, 1987). This integer linear programming problem is known in the literature as the set-covering problem (Bellmore and Ratliff, 1971; Grafinkel and Nembauer, 1972).

CLIP3 uses training data to generate an IP model and then lets a standard IP routine solve it. The solution returned from the IP model indicates the most

$$\begin{aligned}
 &\text{Minimize:} \\
 &3X_1 + X_2 + X_3 + X_4 = Z \\
 &\text{Subject to:} \\
 &-2X_1 + 2X_2 + X_3 \quad = 4 \\
 &3X_1 + X_2 \quad + X_4 = 6 \\
 &X_1 \quad \quad \quad \geq 0 \\
 &\quad X_2 \quad \quad \quad \geq 0 \\
 &\quad \quad X_3 \quad \quad \geq 0 \\
 &\quad \quad \quad X_4 \geq 0
 \end{aligned}$$

Solution: $Z = 4$

When: $X_1 = 1, X_2 = 3, X_3 = 0$ and $X_4 = 0$

Figure 1.
An example IP model in standard form and its solution

important features to be used in the generation of rules. Some of the IP solution types used in this paper include approaches used in other machine learning algorithms. One is the *largest complex first* (Michalski and Larson, 1978), that generates IP solutions that include the features that cover the largest number of positive examples. If background knowledge is known about the problem, the *background knowledge first* (Michalski and Larson, 1978) preference may be used. This preference generates IP solutions that include user-chosen features if possible. Other more obscure relationships can be coded into the IP-solving routines, such as, solve for a user-defined number of features. This is easily done by adding these relationships in as constraints to the model in standard form. The ability to alter the IP-solving routine makes the CLIP3 algorithm very versatile.

CLIP3 algorithm

The CLIP3 algorithm is presented in two parts, a pseudocode and an in-depth explanation. The pseudocode is broken up into five parts to follow closely the standard “main routine” and “subroutine” coding structure. Next, a detailed explanation of the algorithm, including the procedure explanation and the intent of each of the CLIP3 routines is presented for each part of the pseudocode.

Pseudocode for CLIP3 (main routine)

```
GIVEN:      Set of positive examples (POS)
           Set of negative examples (NEG)
INITIALIZE: Rule_list = nil
           ALL_NEG = the number of negative examples
           ALL_POS = the number of positive examples
           ALL_NODES = 1
           Best_rule_accuracy = 1
           Prune_thresh = User Defined (default = 0)
           Stop_thresh = User Defined (default = 0)
DO UNTIL ((All_pos <= Stop_thresh) OR (Best_rule_accuracy <= 0.5))
  {
  Phase I
  Phase II
  Phase III
  }
```

Pseudocode for Phase I (subroutine branch)

```
DO I = 1 TO ALL_NEG
DO J = 1 TO ALL_NODES
  {
  (1) Generate BINij matrix using example NEGi and NODEj
  (2) Using BINij matrix, generate the IP model
  (3) Solve the IP model (call IP_Solve), save the SOLij
  }
```

Kybernetes
26,5

516

```
DO J = 1 TO ALL_NODES
{
(4) Using SOLij and NODEj generate the branch nodes.
(It is important that this step be performed after all BIN matrices have
been generated and solved.
This step cannot be included in the above loop).
}
(5) Eliminate redundant nodes
(6) Update the value ALL_NODES
}
```

Pseudocode for Phase II (subroutine template)

```
DO I = 1 TO ALL_NODES
{
DO J = 1 TO ALL_POS
{
(1) Generate a Template matrix
}
}
(2) Use the Template matrix to generate an IP model, and solve it
(3) Using the IP Solution eliminate redundant nodes
(4) Generate multiple rules
```

Pseudocode for Phase III (subroutine update)

```
(1) Check accuracy of each rule
(2) Choose the best rule
(3) Add the Best rule to Rule_list
(4) Eliminate the examples covered by Rule_list from POS
(5) Update ALL_POS
(6) Update Best_rule_accuracy
```

Pseudocode for IP model solving routine (subroutine IP_solve)

```
GIVEN:    BIN = A binary matrix where each row represents an example
           and each column represents a feature
INITIALIZE: SOL = a null list
           BIN_Row = Number of rows in BIN
DO until (BIN_Row <= Pruning_threshold)
{
(1) Sum each column of BIN one at a time
(2) Determine the column that has the largest summed value
(3) Add the column number to SOL
(4) Update BIN and BIN_Row
}
```

Phase I explanation

In phase I, the matrix of positive examples is partitioned into subsets in many ways so that rules can be generated from smaller, noise-free matrices. The partitioning is done in a similar manner to growing a search tree. Generating the search tree is done by breaking apart (called branching) a matrix (called a node or parent node) into smaller matrices (also called nodes or subnodes). Only the final nodes are used to generate the rules. So, once the subnodes are generated from their parent node, the parent node may be discarded. By keeping only the current nodes, and not the complete tree, memory requirement is greatly reduced.

We start with the first negative example, neg_1 , and all of the positive examples in one node, $NODE_1$ (the root node of the decision tree). Because all the positive examples, POS , are in $NODE_1$ the two can be used interchangeably. Then we generate a binary matrix by a feature-by-feature comparison of the negative example with all the positive examples in the root node. When a positive feature is different from a negative feature, it becomes a candidate for a rule, so the binary matrix is loaded with a 1. In contrast, when a positive feature and a negative feature are the same no rule could be generated from it, so the binary matrix is loaded with a 0. The binary matrices are generated for all existing nodes (for neg_1 there is only one node, the root node).

From this binary matrix an IP model is constructed and solved. The solution of an IP model, generated from the binary matrix, indicates the features that are most different from the negative example. These features will be branched from to generate new subnodes. To prevent a drastic increase in the number of nodes all redundant (same) nodes (and nodes that are a subset of another node) are eliminated. Redundant nodes are eliminated because they offer no new information, while they use up memory and increase execution time.

Once the new nodes are generated the next negative example is selected. This negative example is used to generate the next set of binary matrices and their IP solutions. After all the solutions to all the binary matrices are formed, they are used to split their respective nodes. The process continues for all the negative examples. The nodes that remain at the end are made up of only positive examples and are assumed to be noise free and have common features that will set them apart from all of the negative examples. These nodes will be used to generate the rules.

Because the first step of phase I in the CLIP3 algorithm is the generation of a binary matrix based on feature similarity, the learning set must be made up of discrete feature descriptions.

Phase II explanation

In learning Phase II the final nodes generated from Phase I are reduced in number. A subset of nodes is chosen, so all of the original positive examples are covered by the fewest number of nodes and the rest of the nodes are eliminated. This step eliminates matrices that are generated by overlapping rules. Consider the rule:

If Feature $A = 1$ or Feature $B = 3$ then the example is in the positive set, and all other examples are in the negative set.

Phase I would generate three subsets from the positive set; a subset of examples where $A = 1$, a subset of examples where $B = 3$, and a subset of examples where $A = 1$ and $B = 3$. Since the examples in the subset ($A = 1$ and $B = 3$) are totally included in the other two subsets it provides no new information and it will be eliminated in this step. As a result, rules are not generated for the overlapping examples and thus fewer, more accurate rules can be generated.

The rules that are generated by CLIP3 are combinational in nature. As an example, consider a universe of balls used for a variety of sports with the positive set being tennis balls. A description for the tennis balls might be *fuzzy and orange* or *fuzzy and green*. Further assume that the first feature is texture (smooth, rough, fuzzy, etc.) and the sixth feature is colour (blue, green, red, etc.). A description generated by CLIP3 for this set of tennis balls could be $\langle F1 = \text{fuzzy} \rangle \langle F6 = \text{green, orange} \rangle$. This rule is read as:

If (F1 is fuzzy and F6 is green) or (F1 is fuzzy and F6 is orange)

then the example is in the positive set.

In contrast, if the rule was *fuzzy and orange* or *smooth and green* then CLIP3 would generate two rules, the first:

$\langle F1 = \text{orange} \rangle \langle F6 = \text{fuzzy} \rangle$

and the second $\langle F1 = \text{green} \rangle \langle F6 = \text{smooth} \rangle$.

CLIP3 would generate two rules because the combinational rule $\langle F1 = \text{green, orange} \rangle \langle F6 = \text{smooth, fuzzy} \rangle$ includes two subsets that are not in the positive examples (namely smooth orange and fuzzy green tennis balls). The generation of combinational rules allows for a fewer number of rules to describe more examples.

Using all the nodes generated from Phase I, a template matrix is generated. This is done by assigning each node a column and each example from the original positive matrix a row. A "1" is assigned to the node's column and example's row if the example is present in the node. A "0" is assigned to all other entries. This template matrix is converted to an IP model and solved. The solution to this IP model determines which of the final nodes will generate the rules. This solution ensures that the most informative nodes will be used to generate the rules.

The chosen nodes are back-checked, in a similar manner to the generation of a binary matrix, against all original negative examples to generate the feature values used in the rules. These rules are candidates for the "best" rule.

Phase III explanation

In Phase III the best rule, from all of the rules generated in Phase II, is chosen. The best rule is the rule that describes the most positive examples and the fewest negative examples. Because the Phase II rules were generated from

subsets of the positive examples, their accuracy must be compared against the complete positive set, not the subsets that spawned them. Once the best rule is chosen, the examples described by the best rule are then eliminated from the positive training set and all values are updated accordingly.

The process is repeated starting at Phase I using the reduced data set. This allows the next best rule to be generated without the confusion of the examples already described by previous rules. By training on the reduced training set, the examples that are already described by the first rule do not influence the generation of subsequent rules. This can be better demonstrated when referring back to the example used under Phase II explanation, with the positive examples described as $A = 1$ or $B = 3$. If the best rule after the first pass was $A = 1$, then examples in the positive training set that have $A = 1$ would be eliminated. The next pass of CLIP3 would only then have to generate rule $B = 3$. This becomes an easier task because the $A = 1$ subset was directly eliminated and the $A = 1$ and $B = 3$ subset was inherently eliminated.

The process is repeated until the reliability of the best rule drops below 50 per cent or until a small, user-specified number of positive points remain. This user-specified stopping threshold is one of the pruning techniques incorporated, its purpose is to prevent a drastic increase in the number of rules just to cover the last few examples that may be quite dissimilar from all other examples.

IP solution generator explanation

The algorithm for the IP solution generator used in this paper incorporates a pruning technique and a greedy solution algorithm. This heuristic solves the IP model using the smallest number of features while covering the largest number of examples.

The IP solution generator is searching for a solution to a binary matrix where each row represents an example and each column represents a describing feature. The solution will be the shortest list of features that can be used to describe the most examples. The solution generator incorporates a pruning threshold that allows the solution generator to stop if just a few examples remain. This is done to prevent a drastic increase in the number of describing features included in the solution. If a threshold of 0 is chosen, the solution will continue to include features in the solution until all examples are covered and there will be no pruning.

By summing each column of the binary matrix, the feature that is most dissimilar will have the highest value. This feature is added to the solution list and examples covered by this feature (examples that have a 1 for this feature) are eliminated. The process is repeated until the number of examples is less than the pruning threshold value. Because this solution is used in the branching, any point not covered by the solution will be pruned in the node generation.

Illustrative example

To illustrate the basic ideas of the CLIP3 algorithm the following example will be used. Each training example is described by ten features. The values of each feature have a range of discrete values from 1 to 5. There are 13 positive training examples and 11 negative examples. Matrices storing them are called POS and NEG. They are shown in Figure 2.

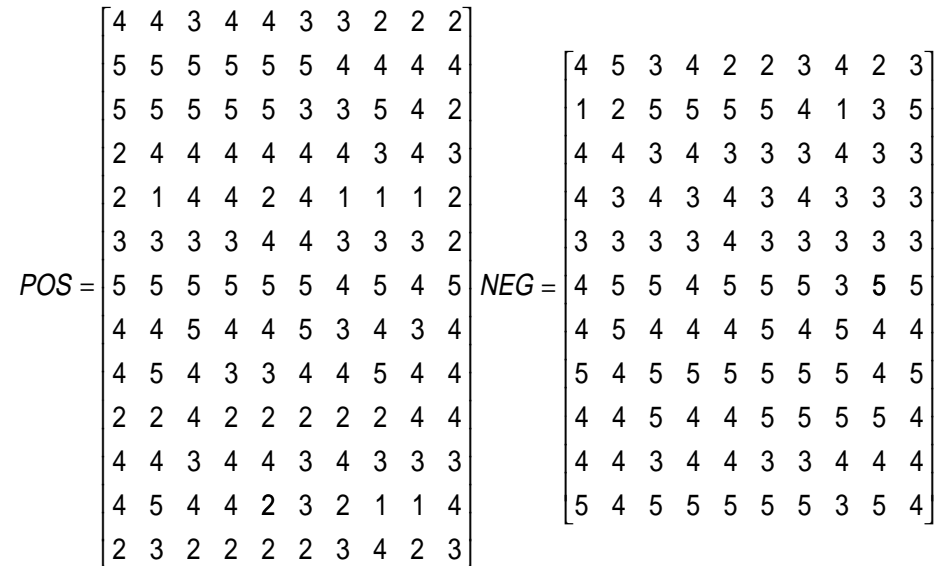


Figure 2.
Training data

Phase I

In Phase I, two major operations are performed: partition of the positive examples and elimination of noise. This is done in a similar manner to growing a search tree. There is one major difference between this phase and growing a search tree. The difference is that the complete tree is not saved; only the current levels' nodes are saved, which results in tremendous memory savings.

Tree expansion and binary matrix formation

The first step in the tree expansion is to generate the binary matrix, BIN_x . This is done by a feature-by-feature comparison of a negative example, with examples stored at every node. At the start, all positive examples are stored in one node ($NODE_1$). In order to generate matrix BIN_1 the first negative example (neg_i where $i = 1$) is selected from matrix NEG_1 and compared, feature by feature, with all examples in $NODE_1$. Note that $NODE_1$ is the same as matrix POS . If a feature of a positive example in $NODE_1$, pos_j , has the same value as the feature of the negative example, neg_j , then it cannot be used to distinguish pos_j from neg_j . The element of matrix BIN associated with this feature is loaded with a 0. Conversely, if the features are different then the element of matrix BIN is

loaded with a 1. The first binary matrix, BIN_1 , is shown in Figure 3. The binary matrix, BIN_1 , represents all possible ways of branching from this node. To reduce branching, and thus to reduce the number of distinguishing features, BIN_1 is converted into an IP model, as shown in Figure 4, and solved.

IP solution and thresholding

An integer threshold is used in conjunction with the IP solving routine to help cope with noisy data. A threshold of zero implies that the data are free of noise and that no examples should be eliminated. The larger the threshold value, the more noise is assumed to be present in the data. Also, the higher the threshold, the more general the IP solution becomes.

521

$$BIN_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.
First level binary matrix

The threshold allows the solution to cover all but the threshold number of examples in the original IP model. For example, if the threshold is set at 2 and the IP model contained 40 examples, the IP solution that is generated will cover at least 38 of the examples. This thresholding is done so the IP solution does not become excessively complicated to cover one or two remaining examples. This procedure prunes noisy examples. If the example is not covered by any IP solution it is assumed to be noise, and will not be passed on to the next level of the tree.

Each IP problem can have multiple solutions, each equally valid, but resulting in totally different tree branchings. A solution matrix, SOL, to the IP model in Figure 4 is shown in Figure 5. Each row in matrix SOL is a unique solution to the IP problem. The first row in matrix SOL shows that features F1 and F6 are the only features needed to discern example neg_1 from all the positive examples in matrix $NODE_1$. This corresponds to $F1 = 4$ and $F6 = 2$, the

Figure 4.
IP problem derived from
BIN₁

$$\begin{aligned}
 & \text{Minimize:} \\
 & X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8 + X_9 + X_{10} = Z \\
 & \text{Subject to:} \\
 & \begin{array}{r}
 X_1 \quad X_2 \quad \quad \quad + X_5 + X_6 \quad \quad + X_8 \quad \quad + X_{10} \geq 1 \\
 X_1 \quad \quad + X_3 + X_4 + X_5 + X_6 + X_7 \quad + X_8 \quad + X_9 + X_{10} \geq 1 \\
 X_1 \quad \quad + X_3 + X_4 + X_5 + X_6 \quad \quad + X_8 \quad + X_9 + X_{10} \geq 1 \\
 X_1 + X_2 + X_3 \quad \quad + X_5 + X_6 + X_7 \quad + X_8 + X_9 \quad \geq 1 \\
 X_1 + X_2 + X_3 \quad \quad + X_6 + X_7 + X_8 + X_9 + X_{10} \geq 1 \\
 X_1 + X_2 \quad \quad + X_4 + X_5 + X_6 \quad \quad + X_8 + X_9 + X_{10} \geq 1 \\
 X_1 \quad \quad + X_3 + X_4 + X_5 + X_6 + X_7 \quad + X_8 \quad + X_9 + X_{10} \geq 1 \\
 X_1 \quad \quad X_2 + X_3 + X_4 + X_5 + X_6 \quad \quad + X_7 + X_8 + X_9 + X_{10} \geq 1 \\
 X_1 \quad \quad X_2 + X_3 + X_4 + X_5 + X_6 \quad \quad + X_7 + X_8 + X_9 + X_{10} \geq 1 \\
 X_1 + X_2 \quad X_3 + X_4 + X_5 + X_6 \quad + X_7 + X_8 + X_9 + X_{10} \geq 1 \\
 X_1 \quad \quad X_2 \quad \quad + X_5 + X_6 + X_7 + X_8 + X_9 \quad \geq 1 \\
 X_1 \quad \quad X_2 \quad X_3 + X_4 \quad \quad + X_6 + X_7 + X_8 + X_9 + X_{10} \geq 1
 \end{array}
 \end{aligned}$$

Figure 5.
Possible solutions to the
IP problem

$$SOL = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

first and sixth feature of neg₁ from matrix NEG (Figure 2). The remaining rows in matrix SOL are other possible solutions that could be branched from, forming different trees. In this illustration, the branching will be done using only the first row of each matrix SOL that is generated, and the IP threshold will be zero (all of the positive examples will be covered for each model). As a rule we will use the first available solution, the first row. If the first solution to the IP problem is used, features F1 and F6 are kept and the rest of the features are not branched from. The result is two new nodes, generated from NODE₁, that were generated using features 1 and 6.

The first branching node, BRANCH_{1,1}, is formed by all the examples in matrix POS that **do not** have feature F1 = 4 (the first feature of neg₁ from Figure 2). The second branch matrix, BRANCH_{1,2}, is formed by the examples in matrix POS that **do not** have feature F6 = 2 (the sixth feature of neg₁ from Figure 2). These two newly formed nodes are shown in Figure 6. It is important to note that NODE₁ no longer needs to be kept. As we can see, there is some overlap between BRANCH_{1,1} and BRANCH_{1,2}, i.e. some positive examples appear in both matrices; however, the advantage of this approach is that we are reducing the size of the original problem.

The rest of the examples in matrix NEG ($\text{neg}_i, i = 2, \dots, 11$) are then compared with all of the branched nodes. For example, neg_2 is compared with the two nodes of level 2, which are shown in Figure 6. After the generation of the binary matrices and their solutions are generated, they are then again independently branched from. This procedure in turn produces the nodes that neg_3 will operate on, and so on.

$$\text{BRANCH } 2_1 = \begin{bmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 3 & 3 & 5 & 4 & 2 \\ 2 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 4 & 3 \\ 2 & 1 & 4 & 4 & 2 & 4 & 1 & 1 & 1 & 2 \\ 3 & 3 & 3 & 3 & 4 & 4 & 3 & 3 & 3 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 9 & 5 & 9 & 5 \\ 2 & 2 & 4 & 2 & 2 & 2 & 2 & 2 & 4 & 4 \\ 2 & 3 & 2 & 2 & 2 & 2 & 3 & 4 & 2 & 3 \end{bmatrix} \quad \text{BRANCH}2_2 = \begin{bmatrix} 4 & 4 & 3 & 4 & 4 & 3 & 3 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 3 & 3 & 5 & 4 & 2 \\ 2 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 4 & 3 \\ 2 & 4 & 1 & 4 & 2 & 4 & 1 & 1 & 1 & 2 \\ 3 & 3 & 3 & 3 & 4 & 4 & 3 & 3 & 3 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 9 & 5 & 9 & 5 \\ 4 & 4 & 5 & 4 & 4 & 5 & 3 & 4 & 3 & 4 \\ 4 & 5 & 4 & 3 & 3 & 4 & 4 & 5 & 4 & 4 \\ 4 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 3 \\ 4 & 5 & 4 & 4 & 2 & 3 & 2 & 1 & 1 & 4 \end{bmatrix}$$

Figure 6.
Second level nodes

Elimination of redundant matrices

At every level, the nodes are checked for matrices that are redundant, i.e. they are the same or form a subset of another matrix. If any exist, they are discarded (clipped) so that redundant or similar branches are not allowed to grow. This is done to speed up computation time and reduce the total number of matrices stored at nodes at the bottom (last) level. In this example, with a zero threshold, one final result may be (depending on the IP solution used) six node matrices that make up the 11th level (because there are 11 negative examples).

By repeating Phase I many times, on a still reduced set of positive examples, we achieve the goal of finding a small number of tight covers/rules describing the positive examples. Otherwise, we would have many rules covering the same examples in many different ways.

Learning phase II

Once the final node matrices, $\text{BRANCH}11_1 - \text{BRANCH}11_6$, (further referred to as end branch matrices) are derived, the end branch matrices that represent overlapping descriptions will be eliminated. These overlapping descriptions come into play since several branch matrices retain the same examples from original matrix POS. The goal is to have all POS examples covered by the fewest number of end branch matrices. This is done with the template matrix.

Template matrix formation

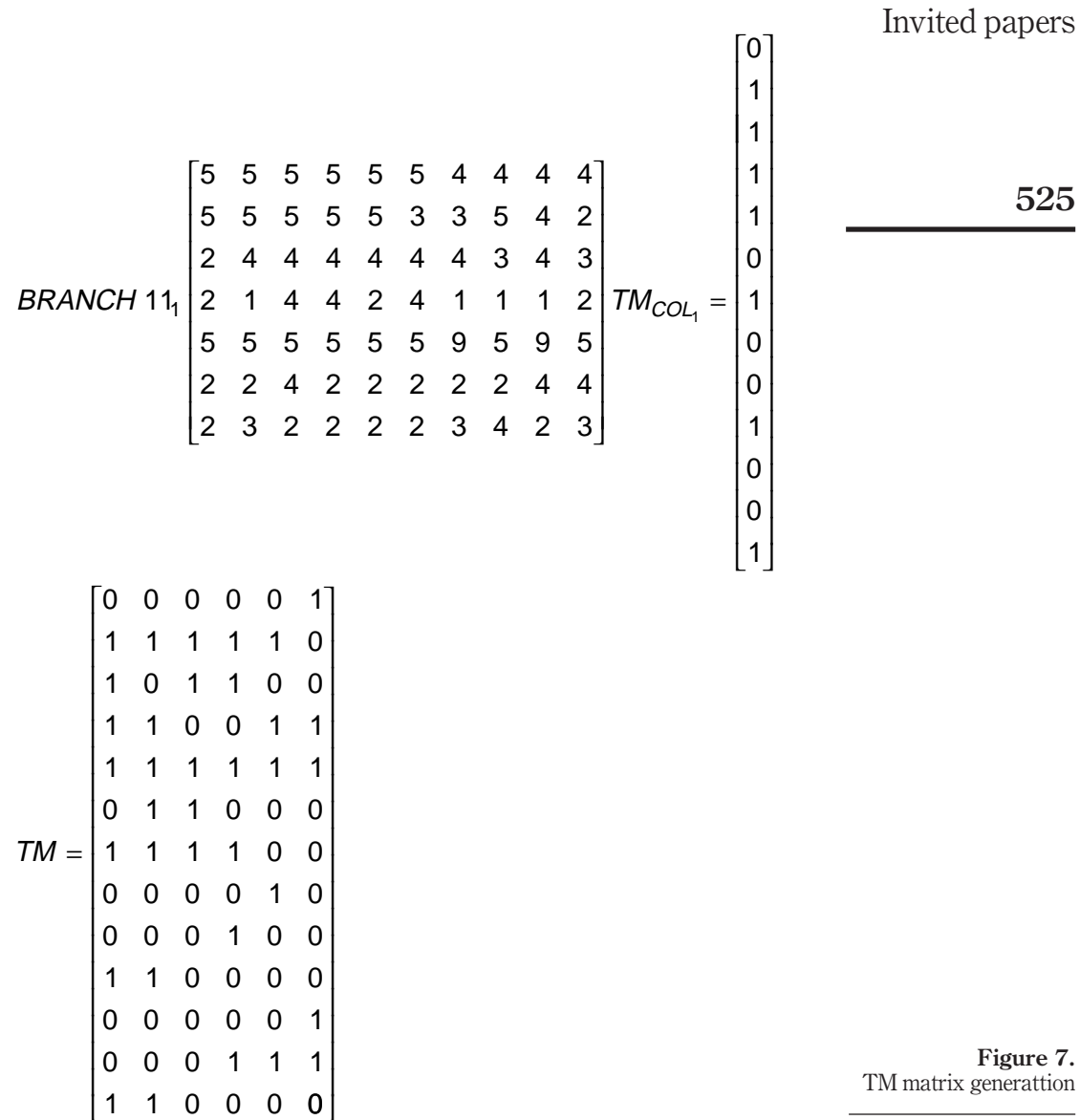
To eliminate these overlapping end branch matrices a template matrix, TM, is constructed with j rows (the number of positive examples in the original matrix POS, in this case 13) and k columns (equal to the number of end branch matrices, in this case 6). Thus, matrix TM is of dimension 13×6 . Matrix TM is generated by assigning each end branch matrix one column, while each row is assigned one example from matrix POS. If an end branch matrix has an example from matrix POS, a 1 is loaded into the branch's column and the example's row; all other elements are 0. An example of generating the first column in TM from the first end branch matrix, and the final TM matrix are shown in Figure 7. We can see that we will have in any column as many 1s as there are examples in the end branch matrix used for creating this column (BRANCH11₁ has seven examples, so TM_{COL1} has seven 1s placed at rows corresponding to their location in the original matrix POS).

Once the matrix TM is formed, it is then converted to an IP problem and solved. A different pruning threshold may be used for this solution. The solutions to this IP problem are the particular end branch matrices that contain all (or most depending on the pruning threshold) of the examples in matrix POS. One possible solution for this matrix TM's IP model is TM SOL₁ = [110111]. This means that end branch matrices 1, 2, 4, 5, and 6 are needed to include all the original positive examples from matrix POS. End branch matrix 3, BRANCH11₃, is eliminated because all of its examples are included in the other five end branch matrices.

Back checking the end matrices

Back checking of the end matrices is done in order to identify key features for distinguishing positive examples from all negative examples. The five remaining end branch matrices are back checked with the original matrix NEG, by "overstriking" matrix NEG with all the examples in each end branch matrix, one at a time. The overstriking is done by comparing (positive) examples in a single end branch matrix, feature-by-feature, to all the examples in the original matrix NEG. If a feature is the same it is replaced by a 0, in the original matrix NEG, since the feature cannot be used to distinguish the two matrices. Otherwise it is left unchanged. Once the overstriking is complete the result is a checked matrix, CM. This process generates five checked matrices (CM₁ – CM₅) of the same dimension as the original matrix NEG. Let us notice that the values feature F1 takes on are 1, 3, and 4, while feature F7 takes on one value of 5 (see Figure 2). The CM matrices are then converted to binary matrices by changing all non-zero values to 1. The binary matrices are converted to IP problems and solved for the features to be used in generating a rule. The generation of CM₁ from BRANCH11₁ (see Figure 7), and one possible IP solution (CM₁ SOL₁) to the IP model generated from CM₁ is shown in Figure 8.

This first solution (CM₁ SOL₁) indicates that features 1 and 7 are the key features to separate all the elements of node matrix BRANCH11₁ (a subset of



matrix POS) from all the negative examples stored in matrix NEG. The same is done for matrices CM₂ through CM₅.

Rule generation

The CM matrix is generated with the NEG matrix to determine what are the discriminating features for each end branch matrix. This is done by back

Kybernetes
26,5

526

Figure 8.
Checked matrix 1
generated from matrix
NEG, and the IP
solution

$$CM_1 = \begin{bmatrix} 4 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 4 & 0 & 3 & 0 & 3 & 0 & 0 & 0 & 3 & 0 \\ 4 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 3 & 0 \\ 3 & 0 & 3 & 3 & 0 & 0 & 0 & 0 & 3 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 5 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 5 & 0 \\ 4 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 5 & 0 \end{bmatrix}$$
$$CM_1 SOL_1 = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0]$$

checking. In short, the CM matrices determine the features needed to separate matrix NEG from each end branch matrix. The complement to these features are the features needed to generate the description for matrix POS.

To generate the rules from this solution, the features present in the solution are chosen and assigned all possible values they can take on. In our example they are in the range from 1 through 5. The values that remain in matrix CM are then eliminated from this set. The rule generated from matrix CM_1 using $CM_1 SOL_1$ is:

$$\langle F1 = 1, 2, 3, 4, 5 \rangle \langle F7 = 1, 2, 3, 4, 5 \rangle.$$

After eliminating values 1, 3, and 4 from F1, and 5 from F7 the simplified rule is:

$$\langle F1 = 2, 5 \rangle \langle F7 = 1, 2, 3, 4 \rangle.$$

This rule reads:

IF [(F1 = 2 or 5) and (F7 = 1 or 2 or 3 or 4)] THEN the example belongs to category POS

ELSE the example belongs to category NEG.

Four more such rules will be generated from the remaining matrices CM_2 through CM_5 .

Phase III

Because the rules were generated using the end branch matrices, that are subsets of matrix POS, the individual rules will cover only a subset of the matrix POS. Once all the rules are generated from the multiple solutions of each CM matrix, the best rule is chosen. The best rule is one that describes the most

positive examples and the fewest negative examples. The next step in the CLIP3 algorithm gives it the power to generate highly accurate rules.

Once the best rule is chosen, the positive examples that it describes are then eliminated from the original positive training matrix POS. If the above rule ($\langle F1 = 2, 5 \rangle \langle F7 = 1, 2, 3, 4 \rangle$) is chosen as the best rule, the seven original examples the rule describes are eliminated from the original matrix POS. The new training set is shown in Figure 9. Eliminating the examples from matrix POS will make the next set of rules clearer because the next set of rules will not be cluttered trying to describe examples already covered by the previous rule. The examples in the matrix NEG are never eliminated because every example in matrix NEG was used to break down the original matrix POS into subsets (nodes). Using this reduced training set, the CLIP3 algorithm is run again.

$$\begin{array}{l}
 \text{POS} = \begin{bmatrix} 4 & 4 & 3 & 4 & 4 & 3 & 3 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 4 & 4 & 3 & 3 & 3 & 2 \\ 4 & 4 & 5 & 4 & 4 & 5 & 3 & 4 & 3 & 4 \\ 4 & 5 & 4 & 3 & 3 & 4 & 4 & 5 & 4 & 4 \\ 4 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 3 \\ 4 & 5 & 4 & 4 & 2 & 3 & 2 & 1 & 1 & 4 \end{bmatrix} \\
 \text{NEG} = \begin{bmatrix} 4 & 5 & 3 & 4 & 2 & 2 & 3 & 4 & 2 & 3 \\ 1 & 2 & 5 & 5 & 5 & 5 & 4 & 1 & 3 & 5 \\ 4 & 4 & 3 & 4 & 3 & 3 & 3 & 4 & 3 & 3 \\ 4 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 4 & 3 & 3 & 3 & 3 & 3 \\ 4 & 5 & 5 & 4 & 5 & 5 & 5 & 3 & 5 & 5 \\ 4 & 5 & 4 & 4 & 4 & 5 & 4 & 5 & 4 & 4 \\ 5 & 4 & 5 & 5 & 5 & 5 & 5 & 5 & 4 & 5 \\ 4 & 4 & 5 & 4 & 4 & 5 & 5 & 5 & 5 & 4 \\ 4 & 4 & 3 & 4 & 4 & 3 & 3 & 4 & 4 & 4 \\ 5 & 4 & 5 & 5 & 5 & 5 & 5 & 3 & 5 & 4 \end{bmatrix}
 \end{array}$$

Figure 9.
Reduced training POS
data set

The process of generating rules and eliminating the examples they describe continues until some stopping criterion is met. The process is stopped if either there are no more examples in the matrix POS (or less than some user-specified number) or the best rule's accuracy rating drops below 50 per cent. When the accuracy of the rules drops to 50 per cent, the rules generated become simply random guesses. Once all the rules are generated, they are "OR"ed together.

Experiments and results

The following is a comparison of the CLIP3 algorithm with other algorithms run on the MONK's problems, (Thrun *et al.*, 1991).

The MONK's problems

The MONK's problems are chosen from a domain defined by a set of robots that have six attributes. The robot attributes are, three head shapes (round, square, or octagon), three body shapes (round, square, or octagon), two facial expressions (smiling or frowning), three objects being held (sword, flag, or a

Kybernetes
26,5

528

balloon), four jacket colours (red, yellow, green, or blue), and two tie definitions (wearing a tie, or not wearing a tie). This produces a domain of 432 unique robots; from this domain three tests were derived. The training sets used in the CLIP3 experiment were exactly the same training sets, including the same noise corrupted examples in test 3, that were used in the MONK's comparison paper (Thrun et al., 1991). These data sets were chosen so that results would be unbiased and could be compared with the results of the original test.

The first MONK's test, M1, defines its positive set as (*head shape = body shape*) or (*jacket = red*). This produces 216 positive examples, and 216 negative examples. From this set 124 noise-free examples were chosen at random, by the authors of the original MONK's comparison paper, as the training set. This is a disjunctive normal form problem, and was designed to be easily learned by symbolic learning algorithms.

The second MONK's test, M2, defines its positive set as (*exactly two of the attributes have their first value*), or more simply put, there are exactly two number ones in the six digit code defining the robot. This produces 142 positive examples, and 290 negative examples. From this set 189 noise-free examples were chosen at random by the authors of the original MONK's comparison paper, as the training set. This is a conjunctive normal form problem, and resembles a parity problem and thus makes it difficult to generate rules using attributes alone.

The third MONK's test, M3, defines its positive set as (*jacket is green and holding a sword*) or (*jacket is not blue and the body shape is not an octagon*). This produces 156 positive examples, and 276 negative examples. From this set 122 examples were chosen at random, again by the authors of the original MONK's comparison paper, as the training set. Then 5 per cent were misclassified, by the authors of the original MONK's comparison paper, to induce noise. This is a disjunctive normal form problem and was designed to test the algorithm's ability to cope with noisy data.

CLIP3 testing procedures

In the testing of CLIP3 with the MONK's problems, the same IP solution generator was used in all three MONK's tests. The IP solution generator was the same as the solution generator described in the IP Solution Generator explanation. A threshold of 1 was used on all three MONK's data sets and then a second time with a threshold of 2. This was done to demonstrate the generalizing nature of the noise threshold. Both results are given in the Tables later.

To choose the best rule for the three tests the same algorithm was used. All generated rules were tested on both positive and negative training sets. The number of positive and negative examples that the rule covered were recorded as correct recognition and incorrect recognition, respectively. If multiple IP solutions were the same (with different values for the features) the rules were tested individually, and the number of correct and incorrect recognitions added together.

Rules with the same IP solution are considered to be the same, but the values of each rule are assumed to be “tied” together logically. The rules cannot be combined. The rule with the smallest ratio of incorrect recognitions to correct recognitions was kept as a final rule.

If more than one rule had the same or nearly the same ratio, then other “tie-breaking” criteria were used to choose the best rule: (1) the total number of positive examples covered, and (2) the number of times the rule was repeated. For example if two rules had a ratio of zero (neither rule described any examples in matrix NEG) but the number of positive examples covered by rule one was three and the number of positive examples covered by rule two was 25, then rule two was chosen. Because redundant matrices are eliminated, repeated rules show a strong tendency that a rule is describing a true concept. This factor was used if all other methods were indeterminate.

The criterion for the stopping threshold was ten. When fewer than ten positive examples remain in $NODE_1$ or the best rule generated has a ratio of number of incorrect recognitions to number of correct recognitions greater than 0.5 (since at this point random guessing would give better results).

Once all the rules were generated for the three MONK’s training sets they were tested for accuracy in the same manner as it was calculated in the MONK’s paper (Thrun *et al.* 1991). The rules were used to classify each example of the full set, as either a positive example or a negative example. The accuracy is defined as the total number of correctly identified positive and negative examples to the total number of examples.

The MONK’s data were converted into numerical data in the following manner: head shape, (1 = round, 2 = square, 3 = octagon); body shape, (1 = round, 2 = square, 3 = octagon); facial expression, (1 = smiling, 2 = frowning); object holding (1 = sword, 2 = flag, 3 = balloon); jacket colour, (1 = red, 2 = yellow, 3 = green, 4 = blue); and tie definition, (1 = wearing a tie, 2 = not wearing a tie). For example, for the data set M1, a positive learning example (+) is a square-headed, octagon-bodied, smiling, balloon-holding, red-jacketed, and tieless robot, is recorded as (+ 2 3 1 3 1 2).

Testing CLIP3 with the MONK’s problems

The first MONK’s test, M1, was run using CLIP3 with a threshold value of 1 for the IP solution generator. This case was treated as a noise-corrupted data set. The results of this test were four rules that correctly identified 100 per cent of the data. The threshold was then increased to 2 and the data set was run again. Increasing the threshold on this data set made no difference; CLIP3 generated the same four rules that correctly identified 100 per cent of the data.

The second MONK’s test, M2, was run with a threshold of 1, and generated ten rules that correctly identified 82.7 per cent of the examples. When the threshold was increased to 2, CLIP3 generated seven rules that correctly identified 72.7 per cent of the examples. The M2 data set is a complicated but noise-free data set, therefore it is obvious that if more examples are eliminated

(by increasing the noise threshold), the rules would become more general and less accurate.

The third MONK's test, M3, was run with a threshold of 1, and generated three rules that correctly identified 88.9 per cent of the examples. When the threshold was increased to 2, CLIP3 generated two rules that correctly identified 97.2 per cent of the examples. Because this was a noise-corrupted data set, the increase of the noise threshold helped identify the major pattern in the data; however, the noise-eliminating nature of the algorithm also eliminated the positive rule that covered a smaller subset of examples.

A point should be noted at this time, the reliability of the test is not influenced by which matrix is specified as the positive matrix and which as the negative matrix. The rules always describe the positive set; so by inverting the POS and NEG matrices and generating new rules, the rules change but are still accurate. To demonstrate this, the positive and negative training matrices were swapped for the M1 MONK's test and the data sets were run on CLIP3. The rules generated were also 100 per cent accurate. However, the rules described the original negative set (the current positive set). The rules generated for the NEG matrix were (head shape \neq body shape) or (jacket = yellow, green, or blue).

Generated rules

The four rules generated for M1 with a threshold of 1 are:

- (1) <F1=1> <F2=1>
- (2) <F1=2> <F2=2>
- (3) <F1=3> <F2=3>
- (4) <F5=1>

The four rules generated for M1 with a threshold of 2 are:

- (1) <F1=1> <F2=1>
- (2) <F1=2> <F2=2>
- (3) <F1=3> <F2=3>
- (4) <F5=1>

The ten rules generated for M2 with a threshold of 1 are:

- (1) <F1=2,3> <F2=2,3> <F3=2> <F4=1> <F5=2,3,4> <F6=1>
- (2) <F1=2,3> <F2=2,3> <F3=1> <F4=2,3> <F5=2,3,4> <F6=1>
- (3) <F1=1> <F3=1> <F4=2,3> <F5=2,3,4> <F6=2>
- (4) <F2=1> <F3=2> <F5=2> <F6=1>
- (5) <F2=2,3> <F3=2> <F5=1> <F6=2>
- (6) <F2=1> <F3=1> <F4=2,3> <F5=2,3,4> <F6=2>
- (7) <F1=2,3> <F3=2> <F4=2,3> <F5=1>
- (8) <F1=1> <F2=2,3> <F3=2> <F4=2,3> <F6=1>
- (9) <F1=3> <F2=1,3> <F4=1> <F6=2>
- (10) <F2=2> <F3=1> <F4=1> <F5=2,3,4> <F6=2>

The seven rules generated for M2 with a threshold of 2 are:

- (1) <F1=2,3> <F2=2,3> <F3=2> <F4=1> <F5=2,3,4> <F6=1>
- (2) <F1=2,3> <F3=1> <F4=2,3> <F5=2,3> <F6=1>
- (3) <F1=1> <F3=1> <F4=2,3> <F5=2,3,4> <F6=2>
- (4) <F1=3> <F2=1> <F4=1,2> <F6=2>
- (5) <F3=1> <F4=1> <F5=3,4> <F6=2>
- (6) <F3=2> <F4=2,3> <F5=1>
- (7) <F2=1> <F3=1> <F5=2,3> <F6=2>

The three rules generated for M3 with a threshold 1 are:

- (1) <F2=1,2> <F5=1,2>
- (2) <F2=1,2> <F3=2> <F5=1,2,3>
- (3) <F4=1> <F5=3>

The two rules generated for M3 with a threshold 2 are:

- (1) <F2=1,2> <F5=1,2>
- (2) <F2=1,2> <F5=1,3>

The following example shows one way to code a set of rules (in this case the rules generated for M3 with a threshold of 2):

```

if ((f2 == 1 or f2 == 2) and (f5 == 1 or f5 == 2))
    { example = POSITIVE_SET; }
else if ((f2 == 1 or f2 == 2) and (f5 == 1 or f5 == 3))
    { example = POSITIVE_SET; }
else
    { example = NEGATIVE_SET; }
end if

```

Comparison of results

The results for each test are compared with the results previously obtained from the MONK's tests (Quinlan, 1993; Thrun *et al.*, 1991; Wu, 1993). Because of some questions raised by Wu (Wu, 1993) on the method of generating the final rule base for the ASSISTANT algorithm in the original paper by Thrun *et al.*, ASSISTANT was left off of the comparison tables.

Table I shows a comparison of the number of rules generated by different algorithms. Table II is a comparison of the accuracy of these rules. On this standardized test CLIP3 did better than many of its predecessors and, more importantly, it generated much fewer number of rules than any other algorithm. We see that only C4.5 algorithm was close in accuracy to the CLIP3 algorithm. Both of these algorithms show one unique phenomenon of the MONK'S tests 2 and 3, that is, it is difficult to do well on both tests at the same time.

Kybernetes
26,5

532

Table I.
Number of rules
generated for the three
MONK's tests

Algorithm	M1	M2	M3
CLIP3 (Threshold 1)	4	10	3
CLIP3 (Threshold 2)	4	7	2
ID3 without windowing	62	110	31
ID3 with windowing	28	110	29
ID5R	52	99	28
AQR	36	83	36
CN2	10	58	24
HCV	7	39	18
C4.5 decision trees	a	a	a
C4.5 tree rules	a	a	a
C4.5 trees with -S	a	a	a
C4.5 -S rules	a	a	a

Note: ^a Data not available

Table II.
Accuracy for the three
MONK's tests

Algorithm	M1 (per cent)	M2 (per cent)	M3 (per cent)
CLIP3 (Threshold 1)	100	82.7	88.9
CLIP3 (Threshold 2)	100	72.7	97.2
ID3 without windowing	83.2	69.1	95.6
ID3 with windowing	98.6	67.9	94.4
ID5R	79.8	69.2	95.3
AQR	95.9	79.6	87.0
CN2	100	69.0	89.1
HCV	100	81.3	90.3
C4.5 decision trees	75.7	65.0	97.2
C4.5 tree rules	100	65.3	96.3
C4.5 trees -S	100	70.4	100
C4.5 -S tree rules	100	67.1	100

Breast cancer data

A second test was performed on a real data set to determine the performance of CLIP3. CLIP3 was run on the Breast Cancer Data (Bennet and Mangasarian, 1992; Mangasarian and Wolberg, 1990; Mangasarian *et al.*, 1990; Wolberg and Mangasarian, 1990). This data set was chosen because it has a large number of examples (683 points), is somewhat noisy, is a well known set, and is made up of integers. An integer data set was chosen to avoid a bias in favour of either of the algorithms. Each data point is made up of the 11 features listed below:

-
- Feature 1: Sample code number (ID number)
 - Feature 2: Clump thickness
 - Feature 3: Uniformity of cell size
 - Feature 4: Uniformity of cell shape
 - Feature 5: Marginal adhesion
 - Feature 6: Single epithelial cell size
 - Feature 7: Bare nuclei
 - Feature 8: Bland chromatin
 - Feature 9: Normal nucleoli
 - Feature 10: Mitoses
 - Feature 11: Class (benign or malignant)

For this experiment, Feature 1 was dropped. Feature 11 was converted to a “+” for Malignant or “-” for Benign and was used only to check the results and set up the training set. The nine remaining features all have an integer range from 1 to 10 and were unaltered.

The breast cancer data have been tested with many different pattern recognition algorithms since they were first generated in 1990. The results from previous experiments have intentionally been omitted from this comparison of algorithms because the database is always being added to, a comparison of algorithm A run in 1991 when the database contained 367 points cannot be compared to algorithm B run in 1995 when the database contained 683 points.

Testing procedures and results

A training set was generated by choosing every fifth point from the complete set, generating a training set of approximately 20 per cent of the total. This training set was used to generate rules with the CLIP3 algorithm following the same procedures that were followed in the MONK’s test. CLIP3 was run with a threshold of 0, 1, and 2. The same data set was then used to generate rules with the C4.5 algorithm using the *-u* option and then the *-s* option.

Once all the rules were generated for both algorithms, they were checked for accuracy against the complete data set. The complete data set was used because some errors occur in the training procedure and these errors must also be counted. This is similar to the procedure followed for the MONK’S test.

Each point is classified by the generated rules and then checked for accuracy. A point is either correctly classified or incorrectly classified. The accuracy is determined by the number of correct classifications divided by the total number of points. The results of the second test are given in Table III.

This test shows that application of both CLIP3 and C4.5 results in very close classification accuracies. However, the advantage of CLIP3 over C4.5 is that CLIP3 does not calculate entropy, which is computation-expensive, and CLIP3 does not have to store the entire decision tree to generate the rules, which is

Kybernetes
26,5

memory-expensive. A word of caution here: there will always be data sets that any machine learning algorithm will classify with high accuracy and there will be data sets on which the same algorithm will do a poor job. This is true for all machine learning and pattern recognition algorithms and this is why no algorithm can claim that it is “best”.

534

	Algorithm	Accuracy (per cent)
Table III. Breast cancer test results for 20 per cent training data set	CLIP3 (Threshold 0)	89.6
	CLIP3 (Threshold 1)	86.8
	CLIP3 (Threshold 2)	92.4
	C4.5 (-U option)	89.3
	C4.5 (-S option)	90.1

The CLIP3 algorithm was successfully used in a work by Torkzadeh, Cios and Pflughoeht (1996) to discriminate between respondents and examine the dimensionality of an end-user computing satisfaction instrument. As a result an instrument shorter than the one used originally for obtaining user responses was developed.

Conclusions

The major advantage of CLIP3 is that it generates several hypotheses to describe a given concept. Testing unknown data with several different hypotheses for a concept results in a better chance of correctly recognizing the test data. The power of CLIP3 comes from its three-step learning process: it partitions the training examples in many ways, selects the best rules to cover the positive training examples, and eliminates the positive examples covered by these rules, so it can perform subsequent learning on a smaller data set.

CLIP3 generates multiple hypotheses using integer linear programming models. The first IP model is used to trim the decision tree, and the second is used to generate concept descriptions with the minimum number of features. Different hypothesis and rule generation preference criteria can be easily implemented by changing the manner in which the IP solution is found, or by choosing a different IP solution from the multiple solutions generated.

CLIP3 can learn from noisy data without overfitting the rules. This is done with the use of the noise threshold in the IP solution. It lets the user drop examples that cause the rules to become excessively complicated. If training data are excessively corrupted with noise, the noise threshold value can be increased to give more generalized rules.

CLIP3 machine learning algorithm is just another good data-mining tool since it is simple and generates very compact rules.

References and further reading

- Bellmore, M. and Ratliff, H.D. (1971), "Set covering and involuntary bases", *Management Science*, Vol. 18 No. 3, November.
- Bennett, K.P. and Mangasarian, O.L. (1992), "Robust linear programming discrimination of two linearly inseparable sets", *Optimization Methods and Software*, No. 1, Gordon & Breach Science Publishers, pp. 23-34.
- Cestnik, B., Konoenko, I. and Bratko, I. (1987), "ASSISTANT 86: a knowledge elicitation tool for sophisticated users", in Bratko, I. and Lavrac, N. (Eds), *Proceedings of the Second European Working Session on Learning*, Bled, Yugoslavia, May.
- Chvatal, V. (1979), "A greedy-heuristic for the set-covering problem", *Mathematical Operations Research*, Vol. 4 No. 3.
- Cios, K.J. and Liu, N. (1995a), "An algorithm which learns multiple covers via integer linear programming: Part I – the CLLP2 algorithm", *Kybernetes*, Vol. 24 No. 2, pp. 29-50.
- Cios, K.J. and Liu, N. (1995b), "An algorithm which learns multiple covers via integer linear programming: Part II – experimental results and conclusions", *Kybernetes*, Vol. 24 No. 3, pp. 28-40.
- Cios, K.J. and Torkzadeh, G. (1992), "Fuzzy clustering and neural networks for instrument validation", *Heuristics, The Journal of Knowledge Engineering*, Vol. 5 No. 1, pp. 90-6.
- Clark, P. and Niblett, T. (1989), "The CN2 algorithm", *Machine Learning*, Vol. 3, pp. 261-83.
- Grafinkel, R.S. and Nembauser, G.L. (1972), *Integer Programming*, John Wiley & Sons, New York, NY.
- Hochbaum, D.S. (1982), "Approximation algorithm for the weighted set-covering and vertex cover problems", *Siam Journal of Computations*, Vol. 11 No. 3, August.
- Mangasarian, O.L. and Wolberg, W.H. (1990), "Cancer diagnosis via linear programming", *Siam News*, Vol. 23 No. 5, September, pp. 1 and 18.
- Mangasarian, O.L. Setiono, R. and Wolberg, W.H. (1990), "Pattern recognition via linear programming: theory and application to medical diagnosis", in Colman, T.F. and Li, Y. (Eds), *Large-scale Numerical Optimization*, Siam Publications, Philadelphia, PA, pp. 22-30.
- Michalski, R.S. (1969), "On the quasi-minimal solution of the general covering problem", *Proceedings of the International Symposium on Information Processing (FCIP 69)*, Vol. A3 (Switching Circuits), Bled, Yugoslavia, pp. 125-8.
- Michalski, R.S. (1974), "Variable-valued logic: systems VL1", *Proceedings of the 1974 International Symposium on Multiple-Valued Logic on Pattern Recognition*, West Virginia University, Morgantown, pp. 323-46.
- Michalski, R.S. (1990), "Learning flexible concepts: fundamental ideas and a method based on two-tiered representation", in Kodratoff, Y. and Michalski, R.S. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. III, Morgan Kaufmann, San Mateo, pp. 63-102.
- Michalski, R.S. and Larson, J. (1978), "Selection of most representative training examples and incremental generation of VL1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11", *Reports of the Department of Computer Science*, No. 867, University of Illinois, Urbana, IL.
- Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N. (1986), "The multipurpose incremental learning system AQ15 and its testing application to three medical domains", *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 1041-50.
- Niblett, T. (1987), "Constructing decision trees in noisy domains", *Proceedings of the Second European Working Session on Learning*, Sigma Press, Bled, Yugoslavia, pp. 67-78.
- Quinlan, J.R. (1983), "Learning effective classification procedures and their application to chess ending games", in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, San Mateo, CA.

- Quinlan, J.R. (1986), "The effect of noise on concept learning", in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. II, Morgan Kaufmann, San Mateo, CA.
- Quinlan, J.R. (1987a), "Simplifying decision trees", *International Journal of Man-Machine Studies*, No. 27, pp. 221-34.
- Quinlan, J.R. (1987b), "Generating production rules from decision trees", *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Milan, Italy, pp. 304-7.
- Quinlan, J.R. (1990), "Probablistic decision trees", in Kodratoff, Y. and Michalski, R.S. (Eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. III, Morgan Kaufmann, San Mateo, CA, pp. 140-52.
- Quinlan, J.R. (1993), *C4.5 programs for machine learning*, Morgan Kaufmann, San Mateo, CA.
- Ravindran, A., Phillips, D.T. and Solberg, J.J. (1987), *Operations Research, Principles and Practice*, John Wiley & Sons, New York, NY, pp. 1-69.
- Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fahlman, S.E., Hamann, R., Kaufman, K., Keller, S., Konoenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J. and Zhang, J. (1991), "The MONK's problems: a performance comparison of different learning algorithms", School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, October.
- Torkzadeh, G., Cios, K.J. and Pflughoeht, K.A. (1996), "Inductive machine learning for instrument development", *Information and Management: The International Journal of Information Systems Appl.*, Vol. 31 No. 1, Elsevier, North Holland, pp. 47-55.
- Wolberg, W.H. and Mangasarian, O.L. (1990), "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", *Proceedings of the National Academy of Sciences*, Vol. 87, December, pp. 22-30.
- Wu, X. (1993), "The HCV induction algorithm", *Proceedings of the 21st ACM Computer Science Conference*, ACM Press, pp. 186-95).