P-91-46

Computer Science

The MONK's Problems – A Performance Comparison of Different Learning Algorithms

S.B. Thrun J. Bala E. Bloedorn I. Bratko B. Cestnik J. Cheng
K. De Jong S. Dzeroski D. Fisher S.E. Fahlman R. Hamann
K. Kaufman S. Keller I. Kononenko J. Kreuziger R.S. Michalski
T. Mitchell P. Pachowicz Y. Reich H. Vafaie W. Van de Welde
W. Wenzel J. Wnek J. Zhang

December 1991
CMU-CS-91-197

Carnegie
Mellon

The MONK's Problems – A Performance Comparison of Different Learning Algorithms

S.B. Thrun J. Bala E. Bloedorn I. Bratko B. Cestnik J. Cheng
K. De Jong S. Dzeroski D. Fisher S.E. Fahlman R. Hamann
K. Kaufman S. Keller I. Kononenko J. Kreuziger R.S. Michalski
T. Mitchell P. Pachowicz Y. Reich H. Vafaie W. Van de Welde
W. Wenzel J. Wnek J. Zhang

December 1991
CMU-CS-91-197

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

This report summarizes a comparison of different learning techniques which was performed at the 2nd European Summer School on Machine Learning, held in Belgium during summer 1991. A variety of symbolic and non-symbolic learning techniques - namely AQ17-DCI, AQ17-HCI, AQ17-FCLS, AQ14-NT, AQ15-GA, Assistant Professional, mFOIL, ID5R, IDL, ID5R-hat, TDIDT, ID3, AQR, CN2, CLASSWEB, ECOBWEB, PRISM, Backpropagation, and Cascade Correlation - are compared on three classification problems, the *MONK's problems*.

The MONK's problems are derived from a domain in which each training example is represented by six discrete-valued attributes. Each problem involves learning a binary function defined over this domain, from a sample of training examples of this function. Experiments were performed with and without noise in the training examples.

One significant characteristic of this comparison is that it was performed by a collection of researchers, each of whom was an advocate of the technique they tested (often they were the creators of the various methods). In this sense, the results are less biased than in comparisons performed by a single person advocating a specific learning method, and more accurately reflect the generalization behavior of the learning techniques as applied by knowledgeable users.

Authors' affiliations: J. Cheng, S.E. Fahlman, T. Mitchell, Y. Reich, and S.B. Thrun are with Carnegie Mellon University; J. Bala, E. Bloedorn, K. De Jong, K. Kaufman, R.S. Michalski, P. Pachowicz, H. Vafaie, J. Wnek, and J. Zhang are with George Mason University (USA), I. Bratko, B. Cestnik, S. Dzeroski, and I. Kononenko are with Josef Stefan Institute (Slovenia), W. Van de Welde is with Vrije Universiteit Brussel (Belgium), J. Kreuziger, R. Hamann, and W. Wenzel are with University of Karlsruhe (Germany), S. Keller is with University of Zuerich (Switzerland), and D. Fisher is with Vanderbilt University (USA).

S.B. Thrun gratefully acknowledges the financial support of Siemens Corp.

Keywords: Machine Learning, MONK's problems, AQ17-DCI, AQ17-HCI, AQ17-FCLS, AQ14-NT, AQ15-GA, Assistant Professional, mFOIL, ID5R, IDL, ID5R-hat, TDIDT, ID3, AQR, CN2, CLASSWEB, ECOBWEB, PRISM, Backpropagation, Cascade Correlation

Once upon a time, in July 1991, the monks of Corsendonk Priory were faced with a school held in their priory, namely the 2^{nd} European Summer School on Machine Learning. After listening more than one week to a wide variety of learning algorithms, they felt rather confused: Which algorithm would be optimal? And which one to avoid? As a consequence of this dilemma, they created a simple task on which all learning algorithms ought to be be compared: the *three MONK's problems.*

This report summarizes the results.

Contents

Results – a short overview

	# 1	# 2	# 3
J. Bala, E. Bloedorn, K. De Jong, K. Kaufman, R.S. Michalski, P. Pachowicz, H. Vafaie, J. Wnek, and J. Zhang			
AQ17-DCI	100%	100%	94.2%
AQ17-HCI	100%	93.1%	100%
AQ17-FCLS		92.6%	97.2%
AQ14-NT			100%
AQ15-GA	100%	86.8%	100%
B. Cestnik, I. Kononenko, and I. Bratko			
Assistant Professional	100%	81.3%	100%
S. Džeroski			
mFOIL	100%	69.2%	100%
W. Van de Velde			
ID5R	81.7%	61.8%	
IDL	97.2%	66.2%	
ID5R-hat	90.3%	65.7%	
TDIDT	75.7%	66.7%	
J. Kreuziger, R. Hamann, and W. Wenzel			
ID3	98.6%	67.9%	94.4%
ID3, no windowing	83.2%	69.1%	95.6%
ID5R	79.7%	69.2%	95.2%
AQR	95.9%	79.7%	87.0%
CN2	100%	69.0%	89.1%
CLASSWEB 0.10	71.8%	64.8%	80.8%
CLASSWEB 0.15	65.7%	61.6%	85.4%
CLASSWEB 0.20	63.0%	57.2%	75.2%
S. Keller			
PRISM	86.3%	72.7%	90.3%
Y. Reich, and D. Fisher			
ECOBWEB leaf prediction	71.8%	67.4%	68.2%
ECOBWEB l.p. & information utility	82.7%	71.3%	68.0%
S. Thrun			
Backpropagation	100%	100%	93.1%
Backprop. with weight decay	100%	100%	97.2%
S. Fahlman			
Cascade Correlation	100%	100%	97.2%

Chapter 1

The MONK's Comparison Of Learning Algorithms – Introduction and Survey

Sebastian B. Thrun
Tom Mitchell
John Cheng

Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213
e-mail: Sebastian.Thrun@cs.cmu.edu, Tom.Mitchell@cs.cmu.edu, John.Cheng@cs.cmu.edu

1.1 The problem

The MONK's problems rely on the an artificial robot domain, in which robots are described by six different attributes [Wnek, Sarma, Wahab and Michalski, 1991]:

x_1: head_shape	$\in$	round, square, octagon
x_2: body_shape	$\in$	round, square, octagon
x_3: is_smiling	$\in$	yes, no
x_4: holding	$\in$	sword, balloon, flag
x_5: jacket_color	$\in$	red, yellow, green, blue
x_6: has_tie	$\in$	yes, no

The learning task is a binary classification task. Each problem is given by a logical description of a class. Robots belong either to this class or not, but instead of providing a complete class description to the learning problem, only a subset of all 432 possible robots with its classification is given. The learning task is then to generalize over these examples and, if the particular learning technique at hand allows this, to derive a simple class description.

- **Problem M_1:**
  (head_shape = body_shape) or (jacket_color = red)
  From 432 possible examples, 124 were randomly selected for the training set. There were no misclassifications.

- **Problem M_2:**
  exactly two of the six attributes have their *first* value.
  (E.g.: body_shape = head_shape = round implies that robot is not smiling, holding no sword, jacket_color is not red and has no tie, since then exactly two (body_shape and head_shape) attributes have their first value) From 432 possible examples, 169 were randomly selected. Again, there was no noise.

- **Problem M_3:**
  (jacket_color is green and holding a sword) or (jacket_color is not blue and body_shape is not octagon)
  From 432 examples, 122 were selected randomly, and among them there were 5% misclassifications, i.e. noise in the training set.

Problem 1 is in standard disjunctive normal form and is supposed to be easy learnable by all symbolic learning algorithms as AQ and Decision Trees. Conversely, problem 2 is similar to parity problems. It combines different attributes in a way which makes it complicated to describe in DNF or CNF using the given attributes only. Problem 3 is again in DNF and serves to evaluate the algorithms under the presence of noise.

1.2 Visualization

All contibutions in this report have two things in common: firstly, they refer to the same problems – the MONK's problems –, and secondly, most results are visualized by a two-dimensional diagram. Due to the difficulties in representing a six-dimensional space on a conventional sheet of paper, the plot is unfolded, as might be found in [Wnek, Sarma, Wahab and Michalski, 1991]. The resulting diagrams of training and testing sets may be found below.

In all training set diagrams, positive examples are marked by "#" and negative ones by "–". Misclassifications, as in the presence of noise, are indicated by boxes. Correspondingly, in all test sets positive examples are marked by "#", while empty fields indicate negative examples.

In turn, we will plot the results of all learning algorithms in the same way: # indicates that the learning algorithm classifies the entity as a positive member, and a blank as a non-member. However, an additional square will indicate misclassifications, i.e. if the classification obtained by the algorithm is wrong.

Acknowledgements

The authors thank Walter Van de Welde for the excellent organization of 2^{nd} European School on Machine Learning, at which this comparison was created. We would also like to thank all participants in this comparison, including Bruno Roger.

References

J. Wnek, J. Sarma, A. Wahab, and R. Michalski: Comparison learning paradigms via diagrammatic Visualization: A case study in single concept learning using symbolic, neural net and genetic algorithm methods, Technical Report, George Mason University, Computer Science Department, 1990

M_1: Training set(124 examples, no noise) and test set

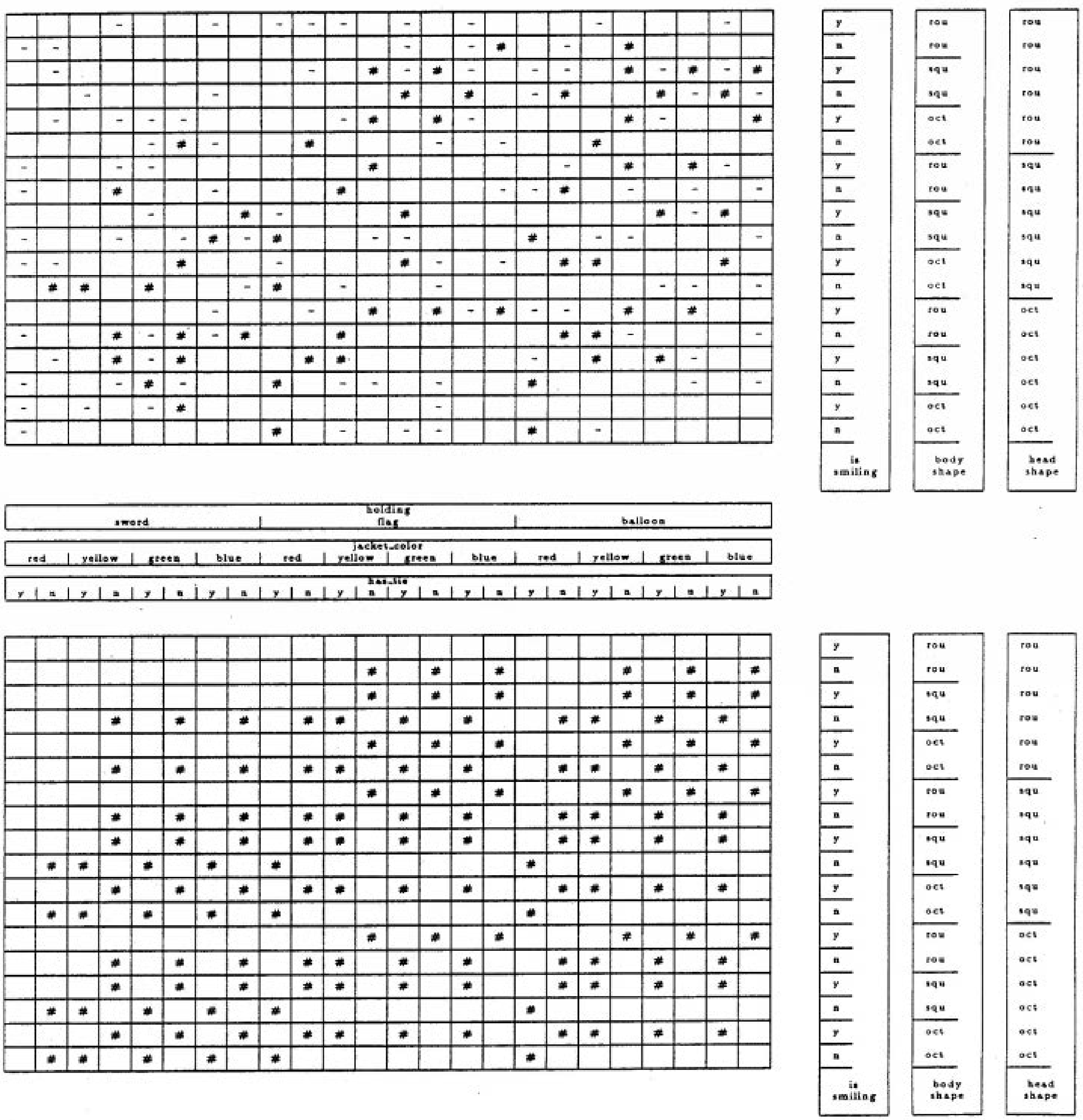
$$(\text{head\_shape} = \text{body\_shape}) \text{ or } (\text{jacket\_color} = \text{red})$$

head shape	body shape	is smiling	sword red y	sword red n	sword yellow y	sword yellow n	sword green y	sword green n	sword blue y	sword blue n	flag red y	flag red n	flag yellow y	flag yellow n	flag green y	flag green n	flag blue y	flag blue n	balloon red y	balloon red n	balloon yellow y	balloon yellow n	balloon green y	balloon green n	balloon blue y	balloon blue n
rou	rou	y					#	#													#			#		
rou	rou	n			#	#									#		#			#						
rou	squ	y		#	-		-			-	#				-	-		-			-					-
rou	squ	n				-										-	-	-				-	-	-		
rou	oct	y			-				-				-				-			#		-	-		-	-
rou	oct	n				-						#		-		-	-	-	#		-				-	-
squ	rou	y					-	-			#	#		-	-		-	-							-	
squ	rou	n				-	-			-					-			-				-			-	
squ	squ	y			#	#	#									#			#	#		#		#		#
squ	squ	n	#					#	#	#			#												#	
squ	oct	y	#								#				-					#			-			-
squ	oct	n						-			#	#	-											-		
oct	rou	y	#	#							#			-								-				
oct	rou	n	#			-								-		-						-				
oct	squ	y	#							-		#						-								
oct	squ	n	#	#				-											#		-				-	
oct	oct	y	#		#					#						#		#		#	#	#				#
oct	oct	n	#					#	#	#										#		#		#		

head shape	body shape	is smiling	sword red y	sword red n	sword yellow y	sword yellow n	sword green y	sword green n	sword blue y	sword blue n	flag red y	flag red n	flag yellow y	flag yellow n	flag green y	flag green n	flag blue y	flag blue n	balloon red y	balloon red n	balloon yellow y	balloon yellow n	balloon green y	balloon green n	balloon blue y	balloon blue n
rou	rou	y	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
rou	rou	n	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
rou	squ	y	#	#							#	#							#	#						
rou	squ	n	#	#							#	#							#	#						
rou	oct	y	#	#							#	#							#	#						
rou	oct	n	#	#							#	#							#	#						
squ	rou	y	#	#							#	#							#	#						
squ	rou	n	#	#							#	#							#	#						
squ	squ	y	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
squ	squ	n	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
squ	oct	y	#	#							#	#							#	#						
squ	oct	n	#	#							#	#							#	#						
oct	rou	y	#	#							#	#							#	#						
oct	rou	n	#	#							#	#							#	#						
oct	squ	y	#	#							#	#							#	#						
oct	squ	n	#	#							#	#							#	#						
oct	oct	y	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
oct	oct	n	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

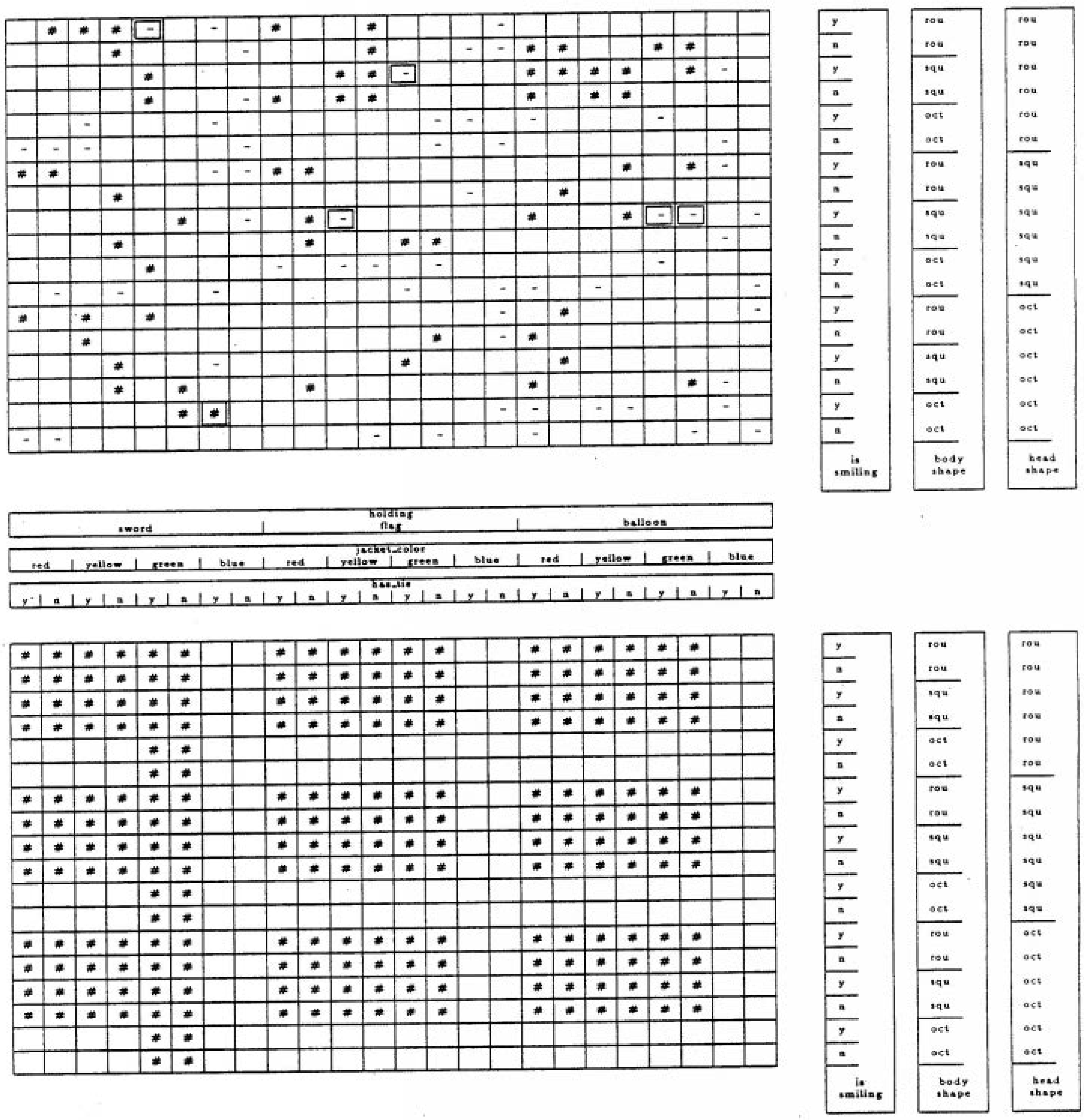
M_2: Training set (169 examples, no noise) and test set

"exactly two of the six attributes have their *first value*"



M_3: Training set (122 examples, 6 misclassifications due to noise) and test set

(jacket_color is green and holding a sword)
or (jacket_color is not blue and body_shape is not octagon)



Chapter 2

Applying Various AQ Programs to the MONK's Problems: Results and Brief Description of the Methods

J. Bala
E. Bloedorn
K. De Jong
K. Kaufman
R.S. Michalski
P. Pachowicz
H. Vafaie
J. Wnek
J. Zhang

Center for Artificial Intelligence, George Mason University, 4400 University Drive, Faixfax, VA 22030

2.1 Introduction

This chapter describes briefly results from applying various AQ learning programs to the MONKS' problems. The MONKS' problems are concerned with learning concept descriptions from examples. All examples come from the same event space, which spans 6 multiple-valued attributes. The sizes of the value sets of the attributes, x1, x2, ..., x6, are 3, 3, 2, 3, 4, and 2, respectively. Consequently, the space consists of the total of 3x3x2x3x4x2 = 432 possible events (examples).

There are three different MONKS' problems. As described in Chapter 1, the problems differ in the type of the target concept to be learned, and in the amount of noise in the data. The training and testing sets of examples were provided by the creators of the problems, Thrun, Mitchell and Cheng. A listing of all the data is in the Appendix. Here is a brief summary of the data for each problem.

- **Problem 1.** There were 124 training examples, which represented 30% of the total event space (62 positive and 62 negative). The testing examples were all possible examples (216 positive and 216 negative).
- **Problem 2.** There were 169 training examples, which represented 40% of the total event space (105 positive and 64 negative). The testing examples were all possible examples (190 positive and 142 negative).
- **Problem 3.** There were 122 training examples, which represented 30% of the total event space (62 positive and 60 negative). The testing examples were all possible examples (204 positive and 228 negative). We were informed that 5% of the examples were misclassified.

The following AQ programs were used in the experiments:

- AQ17-DCI (a version of AQ program with data-driven constructive induction)
- AQ17-HCI (a version of AQ program with hypothesis-driven constructive induction)
- AQ15-GA (a version of AQ program combined with a genetic algorithm)
- AQ15-FCLS (a version of AQ program oriented toward learning flexible concepts)
- AQ14-NT (a version of AQ program oriented toward noisy data)

Rules generated by different programs were tested using the ATEST program that computes a confusion matrix (Reinke, 1984). The program computes the so-called *consonance degree* between an unknown example and the rules for each decision class. The output from this program includes numerical evaluations of the the accuracy of the rules based on the percentage of the testing examples correctly classified (by choosing the rule that best fits the example), and the percentage of examples precisely matched by the correct decision rule.. These percentages are output by ATEST as OVERALL % CORRECT- FLEX-MATCH and OVERALL % CORRECT-100% MATCH, respectively.

Details of these programs, and of the AQ algorithm underlying these programs are given in Section 2.5. It should be noted that results are not always presented for each of these programs as applied to each of the three problems. As indicated above, these programs derive from the same basic method, each adding features appropriate to specific types of problems. The different programs derived basically the same rule for the first problem; the ones shown here are the ones whose knowledge representation schema allowed for the most elegant presentation of the output. We felt that for the sake of brevity and emphasis on the matching of the programs different features with the types of problems to be solved, we should present only the results of the programs better suited for the given type of problem. For example, we felt that there was no reason to apply AQ14-NT, a program with special features to cope with noisy data, to Problem 2, a problem in which data were without noise, and the testing events were 100% correctly classified by the rules obtained by other programs. For the same reason, we did apply the data-driven constructive induction program AQ17-DCI to Problem 3, because it is a strictly data driven method, and as such is less suitable for learning from noisy data than other AQ programs.

2.2 Results for the 1st problem (M_1)

2.2.1 Rules obtained by AQ17-DCI

These are the rules obtained by AQ17-DCI, a version of the AQ program that employs data-driven constructive induction. The results include one rule for Class 0 (that represents positive examples of the concept), and one rule for Class 1 (that represents the negative examples):

```
Class 0:
   Rule 1   [jacket_color > 1] & [head_shape <> body_shape]  (total:62, unique:62)

Class 1:
   Rule 1   [head_shape=body_shape]                           (total:41, unique:33)
   Rule 2   [jacket_color=1]                                  (total:29, unique:21)
```

Expressions in [ ] denote individual conditions in a rule. Values 1, 2, 3 and 4 of the "jacket_color" attribute denote red, yellow, green, and blue, respectively. The body-shape and the head-shape attributes had values 1-round, 2-square, and 3-octagon. In the above rules, "total" means the total number of training examples of the given class covered by the rule, and "unique" means the number of training examples covered by that rule only, and not by any other rules.

There is only one rule for Class 0, and there two rules for Class 1. The latter means that if any of the rules is matched by a given instance, then that instance is classified to Class 1. A set of such rules is logically equivalent to a disjunction of conjunctions. The syntax of the rules is defined formally according to the variable-valued logic calculus VL1. Individual rules correspond to "complexes" in VL1.

The results of applying the rules to the testing examples are presented below.

RESULTS	
OVERALL % CORRECT FLEX MATCH:	100.00
OVERALL % CORRECT 100% MATCH:	100.00

where:
OVERALL % CORRECT FLEX MATCH means the percentage of the correctly classified examples within the total set of testing examples, using a flexible matching function (see Reinke, 1984), and OVERALL CORRECT % 100% MATCH means that the percentage of correctly classified examples that matched the rules exactly.

The number of testing events satisfying individual rules in the correct class description is given in the table below:

		RULES	
		R1	R2
CLASS	0	215	
CLASS	1	144	108

2.2.2 Rules obtained by AQ17-HCI

These are the rules obtained by AQ17-HCI, a version of the AQ program that employs hypothesis-driven constructive induction. The results include one rule for Class 0 that represents positive examples of the concept, and one rule for Class 1 that represents the negative examples:

```
Class 0:
   Rule 1   [Neg17=false]  (total:62, unique:62)

Class 1:
   Rule 1   [Pos16=false]  (total:62, unique:62)
```

where Neg17 and Pos16 are attributes constructed from the original ones, or intermediate ones, as defined below (these rules, as one can check, are logically equivalent to the AQ17-DCI generated rules)

```
c01 <:: [head_shape=1] & [body_shape=2,3] & [jacket_color>1]
c05 <:: [head_shape=2] & [body_shape=1,3] & [jacket_color>1]
c08 <:: [head_shape=3] & [body_shape=1,2] & [jacket_color>1]
c10 <:: [head_shape=1] & [body_shape=1]
c12 <:: [jacket_color=1]
c13 <:: [head_shape=2] & [body_shape=2]
c15 <:: [head_shape=3] & [body_shape=3]
Pos <:: [c10=false] & [c12=false] & [c13=false] & [c15=false]
Neg <:: [c01=false] & [c05=false] & [c08=false]
```

TEST RESULTS - SUMMARY	
OVERALL % CORRECT FLEX MATCH:	100.00
OVERALL % CORRECT 100% MATCH:	100.00

Number of testing events satisfying individual rules in the correct class description:

		RULES
		R 1
CLASS	0	215
CLASS	1	216

Other programs either were not used on this problem, or generated similar results.

2.3 Results for the 2nd problem (M_2)

2.3.1 Rules obtained by AQ17-DCI

The rules below were obtained by AQ17-DCI, which is capable of generating all kinds of new attributes from the original attributes. For the problem at hand, the program found that a new attribute that expresses the number of variables in the learning examples that have some specific value is highly relevant to this problem. Such an attribute is assigned by the program the name #VarEQ(x), which means "the number of variables with value of rank k (in their domain)" in an example. The lowest value in the domain has rank 1, the next lowest has rank 2, etc. In this case, the relevant attribute was #VarEQ(1). Based on this attribute, the program constructed appropriate decision rules. There were two one-condition rules for Class 0, representing the positive examples of the concept, and one rule for Class 1 that represents the negative examples. The rule for Class 1 is logically equivalent to the negation of the union (disjunction) of the rules for Class 0.

```
Class 0:
   Rule 1   [#VarEQ(1)>=3]
   Rule 2   [#VarEQ(1)<=1]

Class 1:
   Rule 1   [#VarEQ(1)=2]
```

The rules say that the number of variables that take the lowest value from their domain is 1 or greater than 2 (i.e, not equal to 2.

The results of applying the rules to the testing examples were:

RESULTS	
OVERALL % CORRECT FLEX MATCH:	100.0
OVERALL % CORRECT 100% MATCH:	100.0

2.3.2 Rules obtained by AQ17-HCI

There are 4 top level rules for Class 0 (positive examples), and 6 top level rules for Class 1(negative examples):

```
Class 0:
   Rule  1   [Pos73=true]  (total:90, unique:49)
   Rule  2   [c14=false] & [c26=false] & [c53=false] & [c67=false] &
             [c72=false] & [Neg74=false]  (total:38, unique:6)
   Rule  3   [holding=2,3] & [c6=false] & [c20=false] & [Neg74=false]   (total:22, unique:5)
   Rule  4   [head_shape=2] & [has_tie=2] & [c44=false] &
             [c50=false] & [Neg74=false]  (total:6, unique:2)

Class 1:
   Rule 1   [Neg74=true]  (total:43, unique:30)
   Rule 2   [jacket_color=2,3,4] & [has_tie=1] & [c60=true] & [Pos73=false]  (total:17, unique:4)
   Rule 3   [head_shape=2,3] & [body_shape=2] & [c28=false] & [Pos73=false]  (total:16, unique:7)
   Rule 4   [body_shape=3] & [c48=true] & [c66=true]  (total:4, unique:2)
   Rule 5   [jacket_color=3] & [c43=true] & [c52=false] & [c53=false] &
            [c55=false] & [c69=true] & [Pos73=false]  (total:4, unique:2)
   Rule 6   [body_shape=3] & [c9=false] & [c10=true] & [c23=true] & [c32=true]  (total:3, unique:1)
```

Attributes "ci, i=2..72" "Pos73," and "Neg74" were constructed during the learning process. The following were relevant to the discovered rules:

```
 c2 <::  [jacket_color=1,4]
 c4 <::  [body_shape=2,3] & [is_smiling=2]
 c5 <::  [head_shape=2,3] & [is_smiling=2]
 c6 <::  [head_shape=2,3] & [body_shape=2,3]
 c7 <::  [holding=1,2] & [jacket_color=1,3,4]
 c9 <::  [head_shape=1,3] & [jacket_color=2,3,4]
c10 <:: [holding=1,2] & [jacket_color=2,3,4]
c14 <:: [jacket_color=2,3,4] & [has_tie=2]
c15 <:: [is_smiling=1] & [jacket_color=2,3,4]
c16 <:: [holding=2,3] & [has_tie=2]
c17 <:: [holding=2,3] & [jacket_color=2,3,4]
c18 <:: [is_smiling=2] & [jacket_color=2,3,4]
c20 <:: [jacket_color=2,3,4] & [has_tie=1]
c21 <:: [body_shape=2,3] & [holding=2,3]
c22 <:: [is_smiling=2] & [holding=1,2]
c23 <:: [holding=1,3] & [jacket_color=2,3,4]
c26 <:: [head_shape=2,3] & [jacket_color=2,3,4]
c28 <:: [body_shape=1,3] & [jacket_color=2,3,4]
c32 <:: [head_shape=2,3] & [jacket_color=1,2,3]
c33 <:: [head_shape=2,3] & [has_tie=2]
c37 <:: [is_smiling=2] & [holding=2,3]
c38 <:: [c21=false] & [c37=false]
c39 <:: [c6=true] & [c17=true]
c40 <:: [c5=true] & [c17=true]
c41 <:: [c15=false] & [c28=false]
c42 <:: [holding=2,3] & [c39=false]
c43 <:: [body_shape=2,3] & [c39=false]
c44 <:: [holding=2,3] & [jacket_color=2,3,4]
c46 <:: [c15=false] & [c39=false]
c47 <:: [c7=false] & [c39=false]
c48 <:: [jacket_color=1,2,4] & [c7=false]
c49 <:: [c17=false] & [c33=true]
c50 <:: [body_shape=2,3] & [c22=false]
c52 <:: [jacket_color=2,3,4] & [c14=false]
c53 <:: [jacket_color=2,3,4] & [c21=true]
c55 <:: [holding=1,2] & [c14=false]
c56 <:: [holding=1,3] & [c14=false]
c59 <:: [jacket_color=2,4]
c60 <:: [c38=false] & [c49=false]
c61 <:: [body_shape=2,3] & [jacket_color=2,3,4]
c65 <:: [c20=false] & [c39=false]
c66 <:: [jacket_color=1,2,3] & [c46=true]
c67 <:: [c38=false] & [c49=true]
c68 <:: [c40=false] & [c55=false]
c69 <:: [c16=false] & [c55=false]
c70 <:: [jacket_color=2,3,4] & [c18=false]
c72 <:: [jacket_color=1,2,3] & [c37=true]
```

```
Pos73 <:: [c4=false] & [c16=false] & [c33=false] & [c39=false] & [c40=false] or
          [c15=false] & [c43=false] & [c47=false] & [c68=false] or
          [body_shape=1,2] & [c21=false] & [c41=true] & [c44=false] & [c65=true] & [c67=false] or
          [c33=true] & [c60=true]

Neg74 <:: [c4=false] & [c42=true] & [c56=false] & [c65=true] & [c68=true] or
          [c2=false] & [c4=false] & [c16=false] & [c17=true] & [c26=true] or
          [is_smiling=2] & [holding=2,3] &  [c14=false] & [c41=true] &
               [c43=true] & [c59=false] & [c69=false] & [c70=false] or
          [has_tie=2] & [c5=true] & [c44=false] & [c61=false]
```

TEST RESULTS – SUMMARY	
OVERALL % CORRECT FLEX MATCH:	93.06
OVERALL % CORRECT 100% MATCH:	86.57

The above summary of the results shows that the rules generated by AQ17-HCI approximate quite well the concept in Problem 2 although they use only logical operators. This result is quite interesting because concepts such as the one in Problem 2 are among the most difficult to learn using solely logic-based inductive learners (classical rule learning or decision tree learning programs). This result demonstrates the power of hypothesis-driven constructive induction.

Number of testing events satisfying individual complexes in the correct class description:

	RULES R 1	R 2	R 3	R 4	R 5	R 6
CLASS 0	232	84	54	12		
CLASS 1	77	44	32	10	5	4

2.3.3 Rules obtained by AQ17-FCLS

These are the rules obtained by AQ17-FCLS, a version of the AQ program that learns flexible concepts by generating rules that permit partial matching. The threshold parameter indicates the minimum percentage of the individual conditions in the rule that must be satisfied for the rule to apply. The results include two rules for Class 0 that represent positive examples of the concept, and 18 rules for Class 1 that represent the negative examples. The discovered rules fully encompass Class 0, but they failed to get a complete grasp of the concept of Class 1:

```
Class 0:
   Rule 1    [head_shape = 1] & [body_shape = 1] & [is_smiling = 1] &
             [holding = 1] & [jacket_color = 1] & [has_tie = 1]
             with THRESHOLD = 50 %
             (Total positive examples covered: 64)
```

This rule says that three or more variables must be equal to 1 (recall that for "is-smiling" and "has-tie" attributes, the value 1 means "yes" and value 2 means "no" ; for attibrute "holding" the value 1 means "sword," 2 means "balloon," and 3 means "flag").

```
   Rule 2    [head_shape = 2 , 3] & [body_shape = 2 , 3] & [is_smiling = 2] &
             [holding = 2 , 3] & [jacket_color = 2 , 3 , 4] & [has_tie = 2]
             with THRESHOLD = 83 % (5/6)
             (Total positive examples covered: 41)
```

This rule says that five or six out of six variables must be greater than 1, or equivalently, that at most one variable can be equal to 1. Thus the disjunction of these two rules above means that the number of variables which have value 1 cannot be equal to 2.

These rules classified 100% all the examples of Class 0.

```
Class 1.
```

Since the current program does not have the ability to express the negation of the above two rules for Class 0, to program generated many "light-weight" rules to cover all examples of Class 1. The overall performance using the flexible match was not 100% because in some cases when an example matched equally well the rules for both classes, an incorrect class was chosen. In the next version of the program, we will include the missing negation operator.

```
   Rule 1    [is_smiling = 1] & [holding = 2 , 3] & [jacket_color = 2] &
             [has_tie = 2]
              with THRESHOLD = 100 %           (total positive examples covered: 8)

   Rule 2    [head_shape = 2 , 3] & [body_shape = 2 , 3] & [is_smiling = 1] &
             [holding = 2 , 3] & [jacket_color = 2 , 3 , 4]  & [has_tie = 1]
              with THRESHOLD = 100 %           (total positive examples covered: 9)

   Rule 3    [head_shape = 2 , 3] & [body_shape = 2 , 2] & [is_smiling = 2] &
             [holding = 2 , 3] & [jacket_color = 2] & [has_tie = 2]
              with THRESHOLD = 100 %           (total positive examples covered: 7)

   Rule 4    [head_shape = 3] & [body_shape = 1] & [is_smiling = 1] &
             [holding = 1] & [jacket_color = 3] & [has_tie = 2]
```

```
         with THRESHOLD = 83 %          (total positive examples covered: 5)

Rule 5   [head_shape = 1] & [is_smiling = 1] & [holding = 2 , 3] &
         [jacket_color = 3 , 4] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 5)

Rule 6   [head_shape = 2 , 3] & [body_shape = 1] & [is_smiling = 2] &
         [holding = 1 , 2] & [jacket_color = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 4)

Rule 7   [head_shape = 1] & [body_shape = 2 , 3] & [is_smiling = 2] &
         [holding = 2 , 3] & [jacket_color = 2 , 3 , 4] & [has_tie = 1]
          with THRESHOLD = 100 %        (total positive examples covered: 5)

Rule 8   [head_shape = 2 , 3] & [is_smiling = 2] & [jacket_color = 1] &
         [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 3)

Rule 9   [head_shape = 2 , 3] & [body_shape = 2 , 3] & [is_smiling = 2] &
         [holding = 1] & [jacket_color = 2 , 3 , 4] & [has_tie = 1]
          with THRESHOLD = 100 %        (total positive examples covered: 4)

Rule 10  [head_shape = 1 , 3] & [body_shape = 1] & [holding = 1 , 2] &
         [jacket_color = 4] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 3)

Rule 11  [head_shape = 2] & [body_shape = 2] & [is_smiling = 1] &
         [holding = 1] & [jacket_color = 2 , 3 , 4] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 5)

Rule 12  [head_shape = 1 , 2] & [body_shape = 3] & [holding = 2 , 3] &
         [jacket_color = 1] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 2)

Rule 13  [head_shape = 1] & [body_shape = 1] & [is_smiling = 2] &
         [holding = 3] & [jacket_color = 2] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 1)

Rule 14  [head_shape = 1] & [body_shape = 3] & [is_smiling = 2] &
         [holding = 1] & [jacket_color = 1 , 3] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 1)

Rule 15  [head_shape = 1] & [body_shape = 2] & [is_smiling = 2] &
         [holding = 2 , 3] & [jacket_color = 1] & [has_tie = 2]
          with THRESHOLD = 100 %        (total positive examples covered: 1)

Rule 16  [head_shape = 2] & [body_shape = 1] & [is_smiling = 1] &
         [holding = 3] & [jacket_color = 2 , 3]
          with THRESHOLD = 100 %        (total positive examples covered: 2)

Rule 17  [head_shape = 3] & [body_shape = 2] & [is_smiling = 1] &
         [holding = 2] & [ jacket_color = 1 , 2]
          with THRESHOLD = 100 %        (total positive examples covered: 2)

Rule 18  [head_shape = 2 , 3] & [body_shape = 1] & [is_smiling = 2] &
         [holding = 2 , 3] &  [jacket_color = 2 , 3 , 4] & [has_tie = 1]
          with THRESHOLD = 100 %        (total positive examples covered: 3)
```

TEST RESULTS - SUMMARY	
The percentage of correctly classified testing events:	92.6%
The percentage of correctly classified testing events in Class 0:	100.0%
The percentage of correctly classified testing events in Class 1:	85.2%
The total number of rules in the descriptions:	2 for Class 0 18 for Class 1.
The total number of conditions in the descriptions:	110

2.4 Results for the 3rd problem (M_3)

2.4.1 Rules obtained by AQ17-HCI

Below are the rules obtained by the hypothesis-driven constructive induction method:

```
Class 0:
   Rule 1    [Pos1=true]                                          (total:49, unique:49)
   Rule 2    [body_shape=2,3] & [holding=2,3] & [jacket_color=3]  (total:11, unique:11)
   Rule 3    [body_shape=1] & [holding=1] & [jacket_color=3]      (total:1, unique:1)
   Rule 4    [body_shape=2] & [holding=2] & [jacket_color=2]      (total:1, unique:1)

Class 1:
   Rule 1    [Neg2=true]                                          (total:57, unique:57)
   Rule 2    [body_shape=3] & [holding=1] & [jacket_color=3,4]    (total:3, unique:3)
```

where Pos1 and Neg2 are attributes constructed from the original ones (Wnek & Michalski, 1991)

```
Pos1 <:: [jacket_color=4]    or    [body_shape=3] & [jacket_color=1,2,4]

Neg2 <:: [body_shape=1,2] & [jacket_color=1,2,3]
```

TEST RESULTS - SUMMARY	
OVERALL % CORRECT FLEX MATCH:	100.00
OVERALL % CORRECT 100% MATCH:	86.11

Since this problem involves noisy data, the flexible match should always be used. The results from 100% match are shown just for comparison.

Number of testing events satisfying individual complexes in the correct class description:

	RULES			
	R 1	R 2	R 3	R 4
CLASS 0	180	24	0	0
CLASS 1	216	12		

2.4.2 Rules obtained by AQ14-NT

These are the rules obtained by AQ14-NT, a version of the AQ program that employs a noise-filtration technique. The results include one rule for Class 0 that represents positive examples of the concept, and one rule for Class 1 that represents negative examples.

After only two loops of concept-driven filtration of training dataset (with truncation parameter equal to 10%) and repeated learning, we received the following set of rules:

```
Class 0:
   Rule 1      [jacket_color=4]
   Rule 2      [body_shape=3] & [holding=2..3]
   Rule 3      [body_shape=3] & [jacket_color=1..2]
Class 1:
   Rule 1      [body_shape=1..2] & [jacket_color=1..3]
   Rule 2      [holding=1] & [jacket_color=3]
```

These rules recognized all test data correctly, i.e., on the 100% level.
Since there was supposed to be noise in the data, we are somewhat surprised by such a high degree of recognition.

2.4.3 Rules obtained by AQ17-FCLS

These are the rules obtained by AQ17-FCLS. The results include two rules for Class 0 that represent positive examples of the concept, and one rule for Class 1 that represents the negative examples. The threshold parameter indicates the minimum percentage of selectors in the rule that must be true for the rule to apply. This set of rules is intentionally incomplete and inconsistent with the training set since it was generated with a 10% error tolerance. This produced better results than other tolerances that were tried:

```
Class 0:
   Rule 1     [head_shape > 1] & [body_shape = 3] & [jacket_color = 4]
              with THRESHOLD = 67 %           (Total positive examples covered: 42)

   Rule 2     [head_shape = 1] & [body_shape = 3] & [jacket_color = 4]
              with  THRESHOLD = 67 %           (Total positive examples covered: 26)

Class 1:
   Rule 1     [body_shape = 1 , 2] & [jacket_color = 1 , 2 , 3]
              with  THRESHOLD = 100 %          (Total positive examples covered: 57)
```

TEST RESULTS - SUMMARY	
The percentage of correctly classified events:	97.2%
The percentage of correctly classified events in Class 0:	100.0%
The percentage of correctly classified events in Class 1:	94.7%
The total number of rules in the descriptions:	2 for Class 0 1 for Class 1
The total number of conditions in the descriptions:	8

2.4.4 Rules obtained by AQ15-GA

Below are the rules obtained by AQ15-GA, a program that uses a genetic algorithm in conjunction with the AQ rule-generation algorithm. The first rule is for the positive examples of the concept, Class 0, and the second for the negative examples, Class 1. A genetic algorithm determined that 3 attributes (body_shape, holding, and jacket_color) were the most meaningful. Using these, the rules discovered were as follows:

```
Class 0:
   Rule 1   [jacket_color=4]
   Rule 2   [body_shape=3] & [jacket_color=1..2]
   Rule 3   [body_shape=2..3] & [holding=2..3] & [jacket_color=3]
   Rule 4   [body_shape=1] & [holding=1] & [jacket_color=3]
   Rule 5   [body_shape=2] & [holding=2] & [jacket_color=2]

Class 1:
   Rule 1   [body_shape=1..2] & [jacket_color=1..3]
   Rule 2   [body_shape=3] & [holding=1] & [jacket_color=3..4]
```

Results on testing the rules on testing events using program ATEST:

TEST RESULTS - SUMMARY	
OVERALL % CORRECT FLEX MATCH:	100.00
OVERALL % CORRECT 100% MATCH:	100.00

2.5 A Brief Description of the Programs and Algorithms

2.5.1 AQ17-DCI (Data-driven constructive induction)

This program is based on the classical AQ algorithm, but it includes an algorithm for constructive induction that generates a number of new attributes. The quality of any generated attribute is evaluated according to a special Quality Function (QF) for attributes, and if that function exceeds a certain threshold value, then the attribute is selected. A brief description of the algorithm for constructive induction (Bloedorn and Michalski, 1991) is given below. The program works in two phases.

Phase 1.

1. Identify all numeric-valued attributes.
2. Repeat steps 3 through 5 for each possible combination of these attributes, starting with the pairs of attributes, and extending them if their quality was found acceptable according to the attribute Quality Function (QF).
3. Repeat steps 4 and 5 for each constructive induction operator. The current operators include addition, subtraction, multiplication, integer division and logical comparison of attributes (Bloedorn and Michalski, 1991).
4. Calculate the values of the given attribute pair for the given constructive induction operator.
5. Evaluate the discriminatory power of this newly constructed attribute using the attribue Quality Function (QF), described by Bloedorn and Michalski (1991). If the QF for an attribute is above an assumed threshold, then the attribute is stored, else it is discarded.

6. Repeat steps 4 and 5 for each available function operator that takes as argument an entire event (example), and calculate various global functions (properties) of it.

The program has a default list of global functions, but allows the user to modify the list to fit the problem at hand. The default list of functions include MAX (the maximum of the values of the numerical attributes in an event), MIN (the minimum value), AVE (the average value), MF (the most-frequent value), LF (least-frequent), and #VarEQ(x), which measures the number of variables (attributes) that take the value x in an example of a given class.

Phase 2.

1. Identify in the data all attributes that are binary.
2. Search for pairwise symmetry among the attributes and then for larger symmetry or approximate symmetry groups, based on the ideas described in (Michalski, 1969a; Jensen, 1975.)
3. For each candidate symmetry group, create a new attribute that is the arithmetic sum of the attributes in the group.
4. Determine the quality function (QF) of the newly created attributes, and select the best attribute.
5. Enhance the dataset with values of this attribute, and induce new decision rules.

The method described above allows the system to express simply symmetric or partially symmetric Boolean functions and k-of-n functions, as well as more complex functions that depend on the presence of a certain number of attribute values in the data. Such functions are among the most difficult functions to express in terms of conventional logic operators.

2.5.2 AQ17-FCLS (Flexible concept learning)

This method (Zhang and Michalski, 1991) combines both symbolic and numeric representations in generating a concept description. The program is oriented toward learning flexible concepts, i.e, imprecise and context-dependent. To characterize such concepts the program creates two-tiered descriptions, which consist of a Basic Concept Representation (BCR) and an Inferential Concept Interpretation (ICI) to handle exceptions. In the program, the BCR is in the form of rules, and the ICI is in the form of a weighted evaluation function which sums up the contributions of individual conditions in a rule, and compares it with a THRESHOLD. The learning program learns both the rules and an appropriate value for the THRESHOLD.

Each rule of a concept description is learned in two steps, the first step is similar to the STAR algorithm in AQ that generates a general rule, and the second step optimizes the rule by specializing it and adjusting the accuracy threshold.

2.5.3 AQ17-HCI (Hypothesis-driven constructive induction)

AQ17-HCI (Hypothesis-Driven Constructive Induction) is a module employed in the AQ17 attribute-based multistrategy constructive learning system. This module implements a new iterative constructive induction capability in which new attributes are generated based on the analysis of the hypotheses produced in the previous iteration (Wnek and Michalski, 1991). Input to the HCI module consists of the example set and a set of rules (in this case generated by the AQ15 program). The rules are then evaluated according to a rule quality criterion, and the rules that score the best for each decision class are combined into new attributes. These attributes are incorporated into the set of training examples, and the learning process is repeated. The process continues until a termination criterion is satisfied. The method is a special implementation of the idea of the "survival of the fittest," and therefore can be viewed as a combination of symbolic learning with a form of genetic algorithm-based learning.

A brief description of the HCI algorithm follows:

1. Induce rules for each decision class using a standard AQ algorithm (as implemented in AQ-15) from a subset of the available training examples.
2. Identify variables from the original set that are not present in the rules, and classify them.
3. For each decision class, generate a new attribute that represents the disjunction of the highest quality.
4. Modify the training examples by adding the newly constructed attributes and removing the ones found to be irrelevant.
5. Induce rules from this modified training set.
6. Test these rules against the remainder of the training set. If the performance is not satisfactory, return to step 1. Otherwise, extend the initial complete set of training examples with the attributes from the obtained rules. Induce the final set of rules from this set of examples.

In these examples, the induction in steps 1, 5 and 6 was performed using the learning algorithm implemented in the AQ15 program.

2.5.4 AQ14-NT (noise-tolerant learning from engineering data)

The program implements an algorithm specially designed for learning from noisy engineering data (Pachowicz and Bala, 1991a and 1991b). The acquisition of concept descriptions (in the form of a set of decision rules) is performed in the following two phases:

- **Phase 1:**

  Concept-driven closed-loop filtration of training data, where a single loop of gradual noise removal from the training dataset is composed of the following three stages:

  1. Induce the decision rules from a given dataset using the AQ14 (NEWGEM) inductive learning program.
  2. Truncation of concept descriptions by removing "least significant" rules, that is rules that cover only a small portion of the training data (this step is performed using the so-called TRUNC procedure).
  3. Create a new training dataset that includes only training examples that are covered by the truncated concept descriptions.

- **Phase 2:**

  Acquire concept descriptions from improved training dataset using the AQ14 learning program.

A justification for Phase 1 is that the noise in the data is unlikely to constitute any strong patterns in the data, and therefore will require separate rules to account for it. Thus, the example covered by the "light rules" are likely to represent noise, and therefore are removed from the dataset. Experiments with AQ14-NT applied to a variety of engineering and computer vision problems have shown that it systematically produces classification rules that both perform better and are also much simpler.

2.5.5 AQ15-GA (AQ15 with attribute selection by a genetic algorithm)

In this approach we use genetic algorithms in conjunction with AQ15. Genetic algorithms are used to explore the space of all subsets of a given attribute set. Each of the selected attribute subsets is evaluated (its fitness measured) by invoking AQ15 and measuring the recognition rate of the rules produced.

The evaluation procedure as shown is divided into three main steps. After an attribute subset is selected, the initial training data, consisting of the entire set of attribute vectors and class assignments corresponding to examples from each of the given classes, is reduced. This is done by removing the values for attributes that were eliminated from the original attribute vector. The second step is to apply a classification process (AQ15) to the new reduced training data. The decision rules that AQ15 generates for each of the given classes in the training data are then used for classification. The last step is to use the rules produced by the AQ algorithm in order to evaluate the classification and hence, recognition with respect to the test data.

In order to use genetic algorithms as the search procedure, it is necessary to define a fitness function which properly assesses the decision rules generated by the AQ algorithm. The fitness function takes as an input a set of attribute or attribute definitions, a set of decision rules created by the AQ algorithm, and a collection of testing examples defining the attribute values for each example. The fitness function then views the AQ-generated rules as a form of class description that, when applied to a vector of attribute or attribute values, will evaluate to a number. It is evaluated for every attribute subset by applying the following steps: For every testing example a match score is evaluated for all the classification rules generated by the AQ algorithm, in order to find the rule(s) with the highest or best match. At the end of this process, if there is more than one rule having the highest match score, one rule will be selected based on the chosen conflict resolution process. This rule then represents the classification for the given testing example. If this is the appropriate classification, then the testing example has been recognized correctly. After all the testing examples have been classified, the overall fitness function will be evaluated by adding the weighted sum of the match score of all of the correct recognitions and subtracting the weighted sum of the match score of all of the incorrect recognitions.

2.5.6 The AQ Algorithm that underlies the programs

All the above programs use AQ as the basic induction algorithm. Here is a brief description of the AQ algorithm:

1. Select a seed example from the set of training examples for a given decision class.
2. Using the *extend against* operator (Michalski 1983), generate a set of alternative most general rules (a star) that cover the seed example, but do not cover any negative examples of the class.
3. Select the "best" rule from the star according to a multi-criteria rule quality function (called LEF - the lexicographical evaluation function), and remove the examples covered by this rule from from the set of positive examples yet to be covered.
4. If this set is not empty, select a new seed from it and go to step 2. Otherwise, if another decision class still requires rules to be learned, return to step 1, and perform it for the other decision class.

Acknowledgements

The authors thank Bill Deichler for his comments and criticism of this paper. This research was done in the Artificial Intelligence Center of George Mason University. The activities of the Center are supported in part by the Defense Advanced Research Projects Agency under the grants administered by the Office of Naval Research, No. N00014-87-K-0874 and No. N00014-91-J-1854, in part by the Office of Naval Research under grants No. N00014-88-K-0397, No. N00014-88-K-0226, No. N00014-90-J- 4059, and No. N00014-91-J-1351, and in part by the National Science Foundation under grant No. IRI-9020266.

References

Bala, J.W. and Pachowicz, P.W., "Recognizing Noisy Patterns of Texture via Iterative Optimization and Matching of their Rule Description", *Reports of Machine Learning and Inference Laboratory 90-12*, Center for Artificial Intelligence, George Mason University, Fairfax, Va. 1990.

Bloedorn, E., and Michalski, R.S., "Data-driven Constructive Induction in AQ17-DCI: A Method and Experiments," *Reports of Machine Learning and Inference Laboratory*, Center for Artificial Intelligence, George Mason University, 1991.

Bloedorn, E., Michalski, R.S. and Wnek, J., "AQ17 - A Multistrategy Constructive Learning System," to appear in *Reports of Machine Learning and Inference Laboratory*, Center for Artificial Intelligence, George Mason University, 1991.

Jensen, G.M, "SYM-1: A Program that Detects Symmetry of Variable-Valued Logic Functions," Report No. 729, Department of Computer Science, University of Illinois, Urbana, May 1975.

Michalski, R.S, "Recognition of Total or Partial Symmetry in a Completely or Incompletely Specified Switching Function," *Proceedings of the IV Congress of the International Federation on Automatic Control (IFAC)*, Vol. 27 (Finite Automata and Switching Systems), pp. 109-129, Warsaw, June 16-21, 1969.

Michalski, R.S., "On the Quasi-Minimal Solution of the Covering Problem" *Proceedings of the V International Symposium on Information Processing* (FCIP 69), Vol. A3 (Switching Circuits), Bled, Yugoslavia, pp. 125-128, 1969.

Michalski, R.S., "Discovering Classification Rules Using Variable-Valued Logic System VL_1," *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pp. 162-172. Stanford, California, August 20-23, 1973.

Michalski, R.S., "A Theory and Methodology of Inductive Learning," Chapter in the book, *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J. Carbonell and T. Mitchell (Eds.), pp. 83-134, Morgan Kaufmann Publishing Co., Mountain View, CA, 1983.

Michalski, R.S. and Larson, J.B., "Selection of the Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: the Underlying Methodology and the Description of Programs ESEL and AQ11," *Reports of Intelligent Systems Group*, Report No. 867, Dept. of Computer Science, University of Illinois, Urbana, 1978.

Michalski, R.S. and McCormick, B.H. "Interval Generalization of Switching Theory," *Proceedings of the Third Annual Houston Conference on Computer and Systems Science*, Houston, Texas, April 26-27, 1971.

Michalski, R.S., Mozetic, I., Hong, J., and Lavrac, N., "The Multipurpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains," *Proceedings AAAI*, Philadelphia, August 11-15, 1986.

Mozetic, I, "NEWGEM: Program for Learning from Examples, Program Documentation and User's Guide", *Reports of Intelligent Systems Group*, No. UIUCDCS-F-85-949, Department of Computer Science, University of Illinois at Urbana-Champaign, 1985.

Pachowicz, P.W. and J. Bala, "Improving Recognition Effectiveness of Noisy Texture Concepts through Optimization of Their Descriptions", *Proc of the 8th Int. Workshop on Machine Learning*, Evanston, pp.625-629, 1991a.

Pachowicz, P.W. and Bala, J, "Advancing Texture Recognition through Machine Learning and Concept Optimization", *Reports of Machine Learning and Inference Laboratory*, MLI-6, Artificial Intelligence Center, George Mason University, 1991b (also submitted to IEEE PAMI).

Reinke, R.E., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta-expert System," MasterUs Thesis, University of Illinois, 1984.

Wnek, J. and Michalski, R.S.: "Hypothesis-driven Constructive Induction in AQ17: A Method and Experiments," *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney Australia, August 1991.

Zhang, J. and Michalski, R.S., "Combining Symbolic and Numeric Representations in Learning Flexible Concepts: the FCLS System", to appear in *Reports of Machine Learning and Inference Laboratory*, Artificial Intelligence Center, George Mason University.

Chapter 3

The Assistant Professional Inductive Learning System: MONK's Problems

B. Cestnik†
I. Kononenko‡
I. Bratko†‡

† Jožef Stefan Institute, Jamova 39, 61000 Ljubljana, Slovenia, E-mail: bojan.cestnik@ijs.ac.mail.yu
‡ Faculty of Electrical Engineering and Computer Science, Tržaška 25, 61000 Ljubljana, Slovenia

3.1 Introduction

Assistant Professional (Cestnik, Kononenko and Bratko, 1987) is a system for inductive learning of decision tree. It is based on ID3 (Quinlan, 1979) and upgraded with several new features. Among the most important improvements are *binarization of the attributes, ability to prune the constructed tree at various levels* and *utilization of improved probability estimates.*

The main purpose of binarization, which groups the attribute values into two subsets, is to normalize the informativity of all the attributes with respect to the number of values. As a result we usually get smaller and more accurate decision trees. In addition, binarization also prevents over-splitting of the learning set. Thus, the attribute selection becomes more reliable even in lower levels of the tree where the number of examples is relatively small. However, the binary construction is computationally less efficient and sometimes generates trees that are not well structured.

The basic induction algorithm tends to construct exact decision tree, although in most of real-world problems the classification can not be exact due to noise in data. As a result, a constructed tree may not only capture the proper relations in data but also fit rather random (noisy) patterns. Decision tree pruning mechanisms (Mingers, 1989) were designed to prevent such over-fitting phenomenon. The algorithm that is implemented in Assistant Professional is described in (Cestnik and Bratko, 1991).

Most of the inductive learning algorithms use probability estimates in crucial sub-tasks when constructing a decision tree, such as in selecting the most "informative" attribute and in pruning the tree. Usually, relative frequency is taken as an estimate. It has been shown that relative frequency is rather poor estimator, especially when the number of examples is small. A more general bayesian estimate that proved to be more robust with respect of the number of examples was presented in (Cestnik, 1990). It is called m-estimate and has the following form:

$$p = \frac{n + m \times p_a}{N + m}$$

where n is the number of positive examples, N is the total number of examples, p_a is prior probability and m is a parameter of the estimation. The formula is studied and explained in detail in (Cestnik and Bratko, 1991).

All the mentioned improvements enable Assistant Professional to construct reliable and compact decision trees. The system was successfully used in many real-world applications in various problem areas, such as medicine, economy, industrial quality control, properties prediction, etc.

3.2 Experimental results

Assistant Professional was tested on the three Monk's domains. The tests were conducted on IBM PS II, model 60. The domains were named as follows: FIRST, SECOND and THIRD. Here are the results of the measurements of classification accuracy.

	classification	accuracy on testing sample
FIRST	100.00 %	(432 of 432)
SECOND	81.25 %	(351 of 432)
THIRD	100.00 %	(432 of 432)

On the first and the third domain Assistant Professional was able to find a perfect domain model. However, in the second domain the constructed tree is very large and its performance is relatively poor. In an extensive study of the domain (testing sample) we were able to determine (with a help of our "neural nets") the correct model which is the following:

```
A robot is O.K.
    if exactly two attributes (out of 6) are equal to 1.
```

This concept is extremely complicated for a system that learns decision trees in an attribute-value logic formalism. Note that on average you have to test almost all attributes to determine the answer. Therefore, the constructed tree tends to be very bushy.

Here are the constructed decision trees in the three domains. In square brackets there is the number of examples in the corresponding node.

3.3 Discussion

In this section we will briefly discuss the achieved results from the perspective of the three improvements of Assistant Professional that are mentioned in the introduction.

Obviously, the binarization contributes the most in the THIRD domain. The constructed tree has a clear structure and is perfectly understandable. In the FIRST domain, however, binarization has a rather negative effect on the tree structure, since the concept *Body_shape = Head_shape* would require three branches (there are three possible values for each attribute). In the SECOND domain binarization is expected to be helpful since it only matters if an attribute has the first value or not. Nevertheless, due to the very complicated concept, it did not really show it's power.

The pruning mechanism contributes mostly in the THIRD domain, since there are some examples corrupted by "noise". The main task is to detect and eliminate this corruption. The FIRST and the SECOND domain did not contain any noise; therefore, the corresponding trees were not pruned at all.

The improved probability estimate, which is used also in the tree pruning mechanism, proved to have crucial effect also in the tree construction phase. Just by changing the value of parameter m (Cestnik and Bratko, 1991) different attributes can be selected at various nodes in the tree. As a result, one mayor deficiency of the original algorithm, namely the inability to backtrack, was in a way alleviated.

3.4 Literature

Cestnik, B., Kononenko, I., Bratko, I. (1987), ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users, Progress in Machine Learning, Eds. I.Bratko and N.Lavrac, Sigma Press, Wilmslow.

Cestnik, B. (1990), Estimating Probabilities: A Crucial Task in Machine Learning, Proc. of ECAI 90, Stockholm, Sweden, August 5-8.

Cestnik, B., Bratko, I. (1991), On Estimating Probabilities in Tree Pruning, Proc. of EWSL 91, Porto, Portugal, March 6-8, 1991.

Mongers, J. (1989), An Empirical Comparison of Pruning Methods for Decision Tree Induction, Machine Learning vol. 4, no. 2, Kluwer Academic Publishers.

Quinlan, J.R. (1979), Discovering Rules by Induction from Large Collections of Examples, Expert Systems in the Microelectronic Age, Ed. D.Michie, Edinburgh University Press.

3.5 Resulting Decision Trees

Constructed decision trees in the three domains:

Decision Tree From Domain: FIRST
Pruned with m= 0.00

Number of Nodes : 15
Number of Leaves: 8
Number of Nulls : 0

```
| A5:Jacket_color [124]
| [red]
|   | V1-yes      [29]
| [yellow, green, blue]
|   | A1:Head_shape [95]
|   | [round, square]
|   |   | A2:Body_shape [71]
|   |   | [square]
|   |   |   | A1:Head_shape [27]
|   |   |   | [round]
|   |   |   |   | V2-no      [15]
|   |   |   | [square]
|   |   |   |   | V1-yes     [12]
|   |   | [round, octagon]
|   |   |   | A2:Body_shape [44]
|   |   |   | [round]
|   |   |   |   | A1:Head_shape [22]
|   |   |   |   | [round]
|   |   |   |   |   | V1-yes     [8]
|   |   |   |   | [square]
|   |   |   |   |   | V2-no      [14]
|   |   |   | [octagon]
|   |   |   |   | V2-no      [22]
|   | [octagon]
|   |   | A2:Body_shape [24]
|   |   | [octagon]
|   |   |   | V1-yes     [13]
|   |   | [round, square]
|   |   |   | V2-no      [11]
```

Decision Tree From Domain: SECOND
Pruned with m= 1.00

Number of Nodes : 113
Number of Leaves: 57
Number of Nulls : 1

```
| A4:Holding [169]
| [sword]
|   | A5:Jacket_color [54]
|   | [red]
|   |   | A2:Body_shape [15]
|   |   | [round, square]
|   |   |   | V2-no      [9]
|   |   | [octagon]
|   |   |   | A3:Is_smiling [6]
|   |   |   | [yes]
|   |   |   |   | V2-no      [4]
|   |   |   | [no]
|   |   |   |   | A1:Head_shape [2]
|   |   |   |   | [round]
|   |   |   |   |   | NULL LEAF:
|   |   |   |   |   | V1-yes     [46.0%]
|   |   |   |   |   | V2-no      [54.0%]
|   |   |   |   | [square]
|   |   |   |   |   | V1-yes     [1]
|   |   |   |   | [octagon]
|   |   |   |   |   | V2-no      [1]
|   | [yellow, green, blue]
|   |   | A1:Head_shape [39]
|   |   | [round]
|   |   |   | A5:Jacket_color [10]
|   |   |   | [green]
|   |   |   |   | A6:Has_tie [4]
|   |   |   |   | [no]
|   |   |   |   |   | A3:Is_smiling [2]
|   |   |   |   |   | [yes]
|   |   |   |   |   |   | V2-no      [1]
|   |   |   |   |   | [no]
|   |   |   |   |   |   | V1-yes     [1]
|   |   |   |   | [yes]
|   |   |   |   |   | V2-no      [2]
|   |   |   | [yellow, blue]
|   |   |   |   | V2-no      [6]
|   |   | [square, octagon]
|   |   |   | A6:Has_tie [29]
|   |   |   | [no]
|   |   |   |   | A3:Is_smiling [16]
|   |   |   |   | [yes]
|   |   |   |   |   | A2:Body_shape [6]
|   |   |   |   |   | [round]
|   |   |   |   |   |   | V2-no      [1]
|   |   |   |   |   | [square, octagon]
|   |   |   |   |   |   | V1-yes     [5]
|   |   |   |   | [no]
|   |   |   |   |   | A2:Body_shape [10]
|   |   |   |   |   | [round]
|   |   |   |   |   |   | V1-yes     [4]
|   |   |   |   |   | [square, octagon]
|   |   |   |   |   |   | V2-no      [6]
|   |   |   | [yes]
|   |   |   |   | A3:Is_smiling [13]
|   |   |   |   | [yes]
```

```
|  |  |  |  |  |  | V2-no         [6]
|  |  |  |  |  | [no]
|  |  |  |  |  |  | A2:Body_shape [7]
|  |  |  |  |  |  | [square, octagon]
|  |  |  |  |  |  |  | V1-yes      [4]
|  |  |  |  |  |  | [round]
|  |  |  |  |  |  |  | V2-no       [3]
| [balloon, flag]
|  | A5:Jacket_color [115]
|  | [green, blue]
|  |  | A3:Is_smiling [58]
|  |  | [yes]
|  |  |  | A6:Has_tie [31]
|  |  |  | [yes]
|  |  |  |  | A1:Head_shape [17]
|  |  |  |  | [round]
|  |  |  |  |  | V2-no         [9]
|  |  |  |  | [square, octagon]
|  |  |  |  |  | A2:Body_shape [8]
|  |  |  |  |  | [round]
|  |  |  |  |  |  | V2-no       [2]
|  |  |  |  |  | [square, octagon]
|  |  |  |  |  |  | V1-yes      [6]
|  |  |  | [no]
|  |  |  |  | A1:Head_shape [14]
|  |  |  |  | [round]
|  |  |  |  |  | V1-yes        [5]
|  |  |  |  | [square, octagon]
|  |  |  |  |  | A2:Body_shape [9]
|  |  |  |  |  | [round]
|  |  |  |  |  |  | V1-yes      [4]
|  |  |  |  |  | [square, octagon]
|  |  |  |  |  |  | V2-no       [5]
|  |  | [no]
|  |  |  | A1:Head_shape [27]
|  |  |  | [round]
|  |  |  |  | A2:Body_shape [11]
|  |  |  |  | [round, square]
|  |  |  |  |  | A2:Body_shape [9]
|  |  |  |  |  | [square]
|  |  |  |  |  |  | A6:Has_tie [6]
|  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  | V1-yes     [4]
|  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  | V2-no      [2]
|  |  |  |  |  | [round]
|  |  |  |  |  |  | A6:Has_tie [3]
|  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  | V2-no      [2]
|  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  | V1-yes     [1]
|  |  |  |  | [octagon]
|  |  |  |  |  | V2-no         [2]
|  |  |  | [square, octagon]
|  |  |  |  | V2-no         [16]
|  | [red, yellow]
|  |  | A2:Body_shape [57]
|  |  | [round, square]
|  |  |  | A1:Head_shape [43]
|  |  |  | [round]
|  |  |  |  | A6:Has_tie [13]
|  |  |  |  | [yes]
|  |  |  |  |  | V2-no         [5]
|  |  |  |  | [no]
|  |  |  |  |  | A5:Jacket_color [8]
|  |  |  |  |  | [red]
|  |  |  |  |  |  | A3:Is_smiling [5]
|  |  |  |  |  |  | [yes]
```

```
|  |  |  |  |  |  |  | V2-no       [3]
|  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  | A2:Body_shape [2]
|  |  |  |  |  |  |  | [round]
|  |  |  |  |  |  |  |  | V2-no      [1]
|  |  |  |  |  |  |  | [square]
|  |  |  |  |  |  |  |  | V1-yes     [1]
|  |  |  |  |  | [yellow]
|  |  |  |  |  |  | V1-yes      [3]
|  |  |  | [square, octagon]
|  |  |  |  | A4:Holding [30]
|  |  |  |  | [balloon]
|  |  |  |  |  | A2:Body_shape [13]
|  |  |  |  |  | [round]
|  |  |  |  |  |  | A5:Jacket_color [5]
|  |  |  |  |  |  | [red]
|  |  |  |  |  |  |  | V2-no       [1]
|  |  |  |  |  |  | [yellow]
|  |  |  |  |  |  |  | V1-yes      [4]
|  |  |  |  |  | [square]
|  |  |  |  |  |  | A5:Jacket_color [8]
|  |  |  |  |  |  | [red]
|  |  |  |  |  |  |  | A3:Is_smiling [4]
|  |  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  |  | A1:Head_shape [2]
|  |  |  |  |  |  |  |  | [square]
|  |  |  |  |  |  |  |  |  | V2-no      [1]
|  |  |  |  |  |  |  |  | [octagon]
|  |  |  |  |  |  |  |  |  | V1-yes     [1]
|  |  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  |  | V1-yes     [2]
|  |  |  |  |  |  | [yellow]
|  |  |  |  |  |  |  | A3:Is_smiling [4]
|  |  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  |  | V1-yes     [1]
|  |  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  |  | V2-no      [3]
|  |  |  |  | [flag]
|  |  |  |  |  | A1:Head_shape [17]
|  |  |  |  |  | [square]
|  |  |  |  |  |  | A5:Jacket_color [8]
|  |  |  |  |  |  | [red]
|  |  |  |  |  |  |  | A3:Is_smiling [4]
|  |  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  |  | V2-no      [1]
|  |  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  |  | A2:Body_shape [3]
|  |  |  |  |  |  |  |  | [round]
|  |  |  |  |  |  |  |  |  | A6:Has_tie [2]
|  |  |  |  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  |  |  |  | V2-no     [1]
|  |  |  |  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  |  |  |  | V1-yes    [1]
|  |  |  |  |  |  |  |  | [square]
|  |  |  |  |  |  |  |  |  | V1-yes     [1]
|  |  |  |  |  |  | [yellow]
|  |  |  |  |  |  |  | A3:Is_smiling [4]
|  |  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  |  | V1-yes     [1]
|  |  |  |  |  |  |  | [no]
|  |  |  |  |  |  |  |  | V2-no      [3]
|  |  |  |  |  | [octagon]
|  |  |  |  |  |  | A3:Is_smiling [9]
|  |  |  |  |  |  | [yes]
|  |  |  |  |  |  |  | A5:Jacket_color [5]
|  |  |  |  |  |  |  | [red]
|  |  |  |  |  |  |  |  | V2-no      [3]
|  |  |  |  |  |  |  | [yellow]
```

```
|   |   |   |   |   |   |   |   | V1-yes      [2]
|   |   |   |   |   |   | [no]
|   |   |   |   |   |   |   | A5:Jacket_color [4]
|   |   |   |   |   |   |   | [yellow]
|   |   |   |   |   |   |   |   | A6:Has_tie [2]
|   |   |   |   |   |   |   |   | [yes]
|   |   |   |   |   |   |   |   |   | V1-yes      [1]
|   |   |   |   |   |   |   |   | [no]
|   |   |   |   |   |   |   |   |   | V2-no       [1]
|   |   |   |   |   |   |   | [red]
|   |   |   |   |   |   |   |   | V1-yes      [2]
|   |   | [octagon]
|   |   |   | A6:Has_tie [14]
|   |   |   | [yes]
|   |   |   |   | A4:Holding [10]
|   |   |   |   | [balloon]
|   |   |   |   |   | A5:Jacket_color [6]
|   |   |   |   |   | [yellow]
|   |   |   |   |   |   | V2-no       [3]
|   |   |   |   |   | [red]
|   |   |   |   |   |   | A3:Is_smiling [3]
|   |   |   |   |   |   | [yes]
|   |   |   |   |   |   |   | V2-no       [1]
|   |   |   |   |   |   | [no]
|   |   |   |   |   |   |   | V1-yes      [2]
|   |   |   |   | [flag]
|   |   |   |   |   | A1:Head_shape [4]
|   |   |   |   |   | [round, square]
|   |   |   |   |   |   | V1-yes      [2]
|   |   |   |   |   | [octagon]
|   |   |   |   |   |   | A5:Jacket_color [2]
|   |   |   |   |   |   | [yellow]
|   |   |   |   |   |   |   | V2-no       [1]
|   |   |   |   |   |   | [red]
|   |   |   |   |   |   |   | V1-yes      [1]
|   |   |   | [no]
|   |   |   |   | V1-yes      [4]
```

Decision Tree From Domain: THIRD

Pruned with m= 3.00

Number of Nodes : 9

Number of Leaves: 5

Number of Nulls : 0

```
| A2:Body_shape [122]
| [octagon]
|   | A4:Holding [41]
|   | [sword]
|   |   | A5:Jacket_color [14]
|   |   | [green]
|   |   |   | V1-yes      [2]
|   |   | [red, yellow, blue]
|   |   |   | V1-yes      [1]
|   |   |   | V2-no       [11]
|   | [balloon, flag]
|   |   | V2-no       [27]
| [round, square]
|   | A5:Jacket_color [81]
|   | [blue]
|   |   | V2-no       [19]
|   | [red, yellow, green]
|   |   | V1-yes      [57]
|   |   | V2-no       [5]
```

Chapter 4

mFOIL on the MONK's Problems

Sašo Džeroski

Artificial Intelligence Laboratory, Jožef Stefan Institute, Jamova 39, 61111 Ljubljana, Slovenia
e-mail: saso.dzeroski@ijs.ac.mail.yu

For correspondence until May 1st 1992: The Turing Institute, 36 North Hannover Street, Glasgow G1 2AD United Kingdom, e-mail: saso@turing.ac.uk

4.1 Description

The learning system considered in this summary is named mFOIL and belongs to the class of inductive learning systems that construct logic programs (sets of Prolog clauses) from training examples and background knowledge. This kind of systems that learn relations has been recently named Inductive Logic Programming.

The basic structure of mFOIL is similar to that of FOIL (Quinlan 1990), but the search heuristics and stopping criteria employed are quite different. They are adapted to learning from imperfect (noisy) data. Instead of the entropy (information gain) heuristic, estimates of the expected error of clauses are used as search heuristics. Namely, clauses with the least expected error (estimated from the training set) are considered best. Bayesian probability estimates, such as the Laplace estimate and the m-estimate (Cestnik 1990) are used for estimating the expected error of clauses. In addition, mFOIL uses beam search instead of the hill climbing used in FOIL.

FOIL uses a function-free concept description language, in which conditions of the form *Attribute = value* are not directly expressible, but require the addition of special predicates in the background knowledge. Such conditions are, however, necessary for solving the monk's problems. mFOIL can use conditions of the above form without adding special predicates in the background knowledge.

mFOIL is described in my MSc thesis (Džeroski 1991), which is available on request. It is implemented in Quintus Prolog 2.5.1 (cca. 600 lines of code) and was run on a Sun SPARC Station 1.

I ran mFOIL using different search heuristics: Laplace or m-estimate of expected error of clauses. Different values of m were used in the m-estimate. Higher values of m direct the search towards more *reliable* clauses, i.e., clauses that cover more examples. This did not influence the results on the first training set, but had some effect on the results on the second and the third set. Below are given the rules obtained together with the corresponding search heuristics. The bad results on the second set are due to the small number of examples for each of the disjuncts and the bias in mFOIL which favors shorter rules.

References

Cestnik, B. (1990) Estimating probabilities: A crucial task in machine learning. European Conference on AI, ECAI 1990. Stockholm, Sweden.

Džeroski, S. (1991) Handling noise in inductive logic programming. M.Sc. Thesis, University of Ljubljana, Faculty of Electrical Engineering and Computer Science.

Quinlan, J.R. (1990) Learning logical definitions from relations. Machine Learning 5 (3), 239-266.

4.2 Set 1

Heuristics used in mFOIL:
Laplace,
m=0, 0.01, 0.5, 1, 2, 3, 4, 8, 16, 32, 64
Induction time: cca 1 min
Accuracy: 100 %

```
robot(A,B,C,D,E,F) :-
        A=B.
robot(A,B,C,D,E,F) :-
        E=red.
```

4.3 Set 2

Heuristic used in mFOIL: m=3
Induction time: cca 10 min
Accuracy: 69.21 %

```
robot(A,B,C,D,E,F) :-
        E=yellow,
        not C=no,
        not D=sword,
        F=no.
robot(A,B,C,D,E,F) :-
        D=flag,
        B=octagon,
        C=yes,
        not E=green.
robot(A,B,C,D,E,F) :-
        C=no,
        E=red,
        not D=sword,
        not B=round,
        not A=round.
robot(A,B,C,D,E,F) :-
        E=yellow,
        B=round,
        not C=yes,
        not D=flag.
robot(A,B,C,D,E,F) :-
        B=square,
        C=yes,
        E=yellow.
robot(A,B,C,D,E,F) :-
        E=green,
        B=round,
        not F=yes,
        not A=square.
robot(A,B,C,D,E,F) :-
        E=green,
        not C=no,
        F=no,
        A=round,
        not D=sword.
robot(A,B,C,D,E,F) :-
        B=square,
        E=blue,
        C=yes,
        not A=round.
robot(A,B,C,D,E,F) :-
        not C=yes,
        A=round,
        E=yellow,
        not D=sword.
robot(A,B,C,D,E,F) :-
        E=green,
        D=sword,
        F=no,
        C=yes,
        not A=round.
robot(A,B,C,D,E,F) :-
        E=green,
        not F=no,
        B=square,
        not C=yes,
        not A=square.
robot(A,B,C,D,E,F) :-
        not C=yes,
        E=red,
        F=no,
        not A=round.
robot(A,B,C,D,E,F) :-
        E=green,
        A=square,
        not C=no,
        not D=sword,
        not F=no.
robot(A,B,C,D,E,F) :-
        E=blue,
        B=square,
        not F=no,
        not C=yes,
        not D=sword.
robot(A,B,C,D,E,F) :-
        E=blue,
        F=no,
        not C=no,
        not A=square.
robot(A,B,C,D,E,F) :-
        F=no,
        E=red,
        not C=yes,
        not B=round.
robot(A,B,C,D,E,F) :-
        D=sword,
        C=no,
        B=octagon,
```

```
        A=square,
        F=yes.
robot(A,B,C,D,E,F) :-
        B=round,
        F=no,
        E=blue,
        not D=flag,
        not A=square.
robot(A,B,C,D,E,F) :-
        A=octagon,
        D=flag,
        not F=no,
        not E=red,
        not B=octagon.
```

4.4 Set 3

Heuristic used in mFOIL: m=64
Induction time: cca 1 min
Accuracy: 100%

```
robot(A,B,C,D,E,F) :-
        not B=octagon,
        not E=blue.
robot(A,B,C,D,E,F) :-
        E=green,
        D=sword,
        B=octagon.
```

Chapter 5

Comparison of Decision Tree-Based Learning Algorithms on the MONK's Problems

Walter Van de Welde

Vrije Universiteit Brussel, Artificial Intelligence Laboratory, Pleinlaan 2, B-1050 Brussels, Belgium
e-mail: walter@arti9.vub.ac.be

5.1 IDL: A Brief Introduction

5.1.1 Introduction

IDL is an algorithm for the incremental induction of decision trees. Incremental learning methods are useful when examples become available on a regular basis but good hypotheses are needed anytime, possibly for a performance task. Incrementality is, however, not the primary motivation for this research. More importantly, IDL is specifically designed to find small decision trees. There are various reasons to prefer smaller trees. One reason is efficiency: the fewer decision nodes in a tree, the more efficient an instance can be classified with it. This is, however, a weak argument since cost and frequency of test execution should be taken into account, so that the most cost-effective tree is not necessarily also the smallest one [Nunez 88; Tan and Schlimmer 89]. Another reason to prefer small trees is comprehensibility: small trees tend to be easier to understand. Comprehensibility, however, also depends on the form of the tree. For example Arbab and Michy (85) argue that linear trees are easier to understand. Perhaps the strongest argument for small trees is the relation between tree complexity and classification accuracy [Breiman, Friedman, Olshen and Stone 84; Quinlan 86; Mingers 89a,b; Utgoff 90]. Pearl [78] showed that the complexity of a hypothesis for explaining data is related to the likelihood that it actually explains it. A learning algorithm with a bias towards simplicity is likely to find more accurate hypotheses as well. This heuristic of Occam's Razor has been employed and justified by many authors both empirically [Clark and Niblett 89; Fisher and Schlimmer 88; Iba, Wogulis and Langley 88] and theoretically [Blumer, Ehrenfeucht, Haussler and Warmuth 87].

Complex trees are sometimes unavoidable. For example, an accurate tree for a concept exhibiting the parity problemII has an exponential number of nodes [Seshu 89] and trees for boolean disjunctive normal form concepts contain duplicated subtrees when only using ground attributes as tests [Pagallo and Haussler 89]. Also, different heuristics in otherwise similar algorithms may lead to significant variations in tree size [Mingers 89a]. The induced trees may nonetheless be more complex than strictly necessary. For example, finding the smallest trees for the six-multiplexer concept [Barto 85; Wilson 87] is well known to be far beyond all classical decision tree induction algorithms [Quinlan 88]. So, even when a small tree exists, state of the art decision tree algorithms may fail to find, or even come close to it. IDL on the other hand finds small trees which are often optimal in size. For example, it has no problem inducing a best tree for the 6-multiplexer while requiring fewer examples and less computation than the other algorithms. The problem of induding optimal decision trees is, however, NP-hard [Hyafil and Rivest 76; Hancock 89]. A practical algorithm is necessarily based on strong heuristic guidance and is guaranteed to fail on at least some induction tasks.

To appreciate the novelty of the approach taken in IDL, it is useful to take a look at the relationship with its predecessors, non-incremental top-down induction of decision trees like ID3 [Quinlan 83,86] and the incremental algorithms ID4 [Schlimmer and Fisher 86], ID5 [Utgoff 88a] and ID5R [Utgoff 90]. Top-down induction performs a general-to-specific hill-climbing search, guided by statistical heuristics and without backtracking. The incremental versions, for which a statistics-based best split is always tentative, are designed to recover with minimal loss of training effort from deviations from the search path which ID3 would follow given the same examples E. More sophisticated representations and search operators allow these algorithms to simulate a backtracking top-down search in a hill climbing search [Langley, Gennari and Iba 87; Fisher 87]. However, these algorithms do not contribute any new ideas to improve the complexity or accuracy of learned decision trees. IDL uses the same search operators to construct a small and accurate tree which is not necessarily ID3-equivalent but topologically minimal. In a topologically minimal tree only a minimal number of tests is required to classify objects. IDL is guided by statistics in a top-down search for an accurate tree. At the same time it looks for smaller trees in a bottom-up fashion. Here it is guided, not by statistics, but by tree topological considerations. In effect, IDL simulates a bi-directional search.

5.1.2 Related Work

ID4 [Schlimmer & Fisher 1986], ID5 [Utgoff 88a] and ID5R [Utgoff 89] are three recently developed algorithms for incremental induction of decision trees. The relation with IDL was briefly explained in the introduction. In [Van de Velde 89] it was conjectured that IDL finds a topologically minimal tree if it exists. Elomaa and Kivinen [90] showed, however, how IDL may fail to find the optimal tree for the 3-multiplexer. The multi-multiplexer concept also disproves this conjecture. Their algorithm IDL' nevertheless successfully postprocesses trees and removes irrelevant attributes. Related experiments are reported on in [Van de Velde 90]. These experiments use a version of IDL which is more eager to apply the statistical selection criterion. This has the advantage that any consistent tree can be taken as an initial hypothesis, no matter how it was generated.

Others have explicitly addressed the problem of suboptimality in tree-size. Pruning techniques [Quinlan 87; Fisher and Schlimmer 88; Mingers 89b] avoid overfitting and reduce complexity, often while increasing accuracy. In a multiplexer-like concept the problem occurs at the top: a TDIDT-like algorithm will choose a wrong top-level attribute and there is no way to prune this away. Quinlan [88] proposes to transform a tree into a set of rules which are subsequently simplified. Every possible classification path is interpreted as a rule. Each of the conditions in the rule is removed in turn and classification accuracy of the rule set is tested. If this is improved, then the condition is permanently removed. This process has been shown to be capable of strong optimization at the expense of introducing a different representation. More sophisticated rule simplification techniques have been studied by many authors [Michalski 87; Clark and Niblett 89; Zhang and Michalski 89]. They use statistical measures to balance the importance and typicality of patterns. The techniques of pruning, tree transformation, and rule tweaking can be viewed along a continuum of increasing liberty to manipulate the representation of patterns. IDL is somewhere in the middle: it manipulates several rules at once and is capable of both introducing and deleting tests in a rule. Also note that IDL is incremental, is not motivated by noise, works with one representation, and uses tree structure information in addition to statistics.

Other researchers reduce tree complexity by allowing different tests than the primitive ones, for example boolean combinations [Breiman, Friedman, Olhsen, Stone 84; Clark and Niblett 89; Pagallo and Haussler 89; Seshu 89] or linear threshold units [Utgoff 88b; Utgoff and Brodley 90]. Of these, FRINGE [Pagallo and Haussler 89] is closest in spirit to IDL. It was developed to overcome the problem of replicated subtrees when learning Disjunctive Normal Form concepts. Such concepts usually have no decision tree representation without replications when the primitive attributes are used. FRINGE examines the fringe (2 bottom levels) of a complete tree to find replicated partial paths. The conjunction of two attributes or their negation is added as first class attribute and a new tree is built. This process iterates until no more changes occur. In comparison, note that IDL is incremental, does not change representation bias and tackles the replication problem for concepts which do have a representation without replication. Utgoff and Brodley's method [90] is also incremental.

Wilson [87] used multiplexer concepts to test his classifier system, called Boole. Quinlan [88] noted the extremely slow convergence rate and obtains much better results when using C4, a TDIDT like algorithm, and postprocessing to rules (see above). Bonelli, Parodi, Sen and Wilson [90] describe NewBoole, a new version of Boole which converges significantly faster to accurate results. It still requires around 800 examples to find an (almost) accurate hypothesis, and around 5000 examples to find the minimal set of rules. The same authors also used neural nets of different sizes to learn the same concept. They report convergence after 1600 cycles for a reasonable net (6:20-20-10-10:1). On the 11-multiplexer NewBoole requires around 4000 examples to converge, a neural net around 8000.

Selective training goes back to the windowing technique in ID3 [Quinlan 83]. Wirth and Catlett [88] discuss related techniques and note that the benefit of windowing is limited. Utgoff [89] shows that a window size of one (i.e., ID5R-hat) results in improved training. The idea is not really applicable in IDL, because it still does much work after the tree has become fully accurate.

5.1.3 Conclusion

IDL represents a new approach to the incremental induction of decision trees. It uses a similar representation as ID4 [Schlimmer and Fisher 86] and the same set of search operators, (splitting, pruning and transposition) as ID5(R) [Utgoff 88a,90]. It was argued that a decision tree represents a target concept by virtue of representing a specialization of it. The task of induction is to find a tree such that this specialization is as close as possible to the target concept. Search for a good decision tree can be understood as search in concept space, mediated by decision tree manipulations. The role of the three operations was reconsidered, as well as the heuristics to guide their application. A statistical selection measure, based on a metric on concept space [Lopez de Mantaras 90] is used to guide the expansion of a tree. Tree topological considerations, based on a notion of topological relevance, guide the transposition of nodes to generate oportunities for pruning. IDL uses these heuristics to simulate a bi-directional search for a tree which is topologically minimal. Such a tree minimizes the number of tests needed for classification, and is therefore small. Experiments show that IDL finds small trees, and often optimal ones.

A number of things need to be investigated further. A major open issue is to characterize the concepts for which IDL finds a topologically minimal tree. It is not understood, for example, what makes the 3-multiplexer so different from the 6-multiplexer concept to justify the occasional failure of IDL on the former. Also, the large standard deviations on the mushroom domain are not well understood. It is disappointing that IDL could not find drastically better trees on natural domains, like it did for the multiplexers. Are there no natural data sets for multiplexer-like concepts? Since IDL occasionally fails to find an optimal tree an average case analysis, as outlined by Pazanni and Sarrett [90] would be more useful than a worst-case one. Integration of IDL with constructive induction techniques seems a promising line of research. Situations in which IDL keeps on switching the levels of attributes could be used as an indication that a new attribute may be useful. The behavior of IDL in the presence of noise has not been studied. The integration of techniques developed for top-down algorithms [Mingers 89b] should be investigated.

References

Arbab, B., and Michie, D. (1985) Generating Rules from Examples. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence p631-633.

Barto, A.G. (1985) Learning by statistical cooperation of self-interested neuron-like computing elements. In Human Neurobiology 4, p229-256.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M.K. (1987) Occam's razor. In Information Processing Letters 24, p377-380. North Holland, Amsterdam.

Bonelli, P., Parodi, A., Sen, S., Wilson, S. (1990) NEWBOOLE: A Fast GBML System. In Porter, B.W., Mooney, R.J. (Eds.) Proceedings of the Seventh International Conference on Machine Learning, p153-159. Morgan Kaufmann, San Mateo CA.

Breiman, L., Friedman, J.H., Olhsen, R.A. , Stone, C.J. (1984) Classification and regression trees. Belmont, CA: Wadsworth International Group.

Clark, P., and Niblett, T. (1989) The CN2 Induction Algorithm. In Machine Learning 3, p261-283. Kluwer Academic Publishers. Boston, MA.

Cockett, J.R.B. (1987) Discrete decision theory: Manipulations. In Theoretical Computer Science 54, p215-236.

Elomaa, T., Kivinen, J. (1990) On Inducing Topologically Minimal Decision Trees. In Proceedings of the second IEEE conference on Tools for Artificial Intelligence. Washington, D.C.

Fisher, D.H. (1987) Knowledge Acquisition via Incremental Conceptual Clustering. Doctoral Dissertation, Department of Information and Computer Science, University of California, Irvine, CA.

Fisher, D.H., and Schlimmer, J.C. (1988) Concept Simplification and Prediction Accuracy. In Laird, J. (Ed.) Proceedings of the Fifth International Conference on Machine Learning p22-28. Morgan Kaufmann, San Mateo CA.

Flann, N.S. and Dietterich, T.G. (1990) A Study of Explanation-Based Methods for Inductive Learning. In Machine Learning 4, p187-. Kluwer Academic Publishers, Boston MA.

Hancock, T. (1989) Finding the smallest consistent decision tree is NP-complete. M.Sc. thesis, Harvard University, Cambridge, MA.

Hyafil, R. and Rivest, R.L. (1976) Constructing optimal binary trees is NP-complete. In Information Processing Letters 5, p15-17.

Iba, W., Wogulis, J., Lanley, P. (1988) Trading Off Simplicity and Coverage in Incremental Concept Learning. In Laird, J. (Ed.) Proceedings of the Fifth International Conference on Machine Learning p73-79. Morgan Kaufmann, San Mateo CA.

Langley, P., Gennari, J.H., Iba, W. (1987) Hill-Climbing Theories of Learning. In Langley, P. (Ed.) Proceedings of the Fourth International Workshop on Machine Learning p312-323. Morgan Kaufmann, San Mateo CA.

Lopez de Mantaras (1990) ID3 Revisited: A Distance-Based Criterion for Attribute Selection. To be published in Machine Learning. Kluwer Academic Publishers, Boston MA.

Mingers, J. (1989a) An empirical comparison of selection measures for decision-tree induction. In Machine Learning 3, p319-342. Kluwer Academic Publishers. Boston, MA.

Mingers (1989b) An empirical comparison of pruning methods for decision-tree induction. In Machine Learning 4, p227-243. Kluwer Academic Publishers. Boston, MA.

Nunez, M. (1988) Economic induction: a case study. In Sleeman, D. (Ed.) Proceedings of the Third European Working Session on Learning p139-145. Pitman, London UK.

Pagallo, G., Haussler, D. (1989) Two Algorithms that Learn DNF by Discovering Relevant Features. In Segre, A.M. (Ed.) Proceedings of the Sixth International Workshop on Machine Learning p119-123. Morgan Kaufmann, San Mateo CA.

Pazzani, M.J., Sarrett, W. (1990) Average Case Analysis of Conjunctive Learning Algorithms. In Porter, B.W., Mooney, R.J. (Eds.) Proceedings of the Seventh International Conference on Machine Learning, p339-347. Morgan Kaufmann, San Mateo CA.

Pearl, J. (1978) On the connection between the complexity and credibility of inferred models. In International Journal General Systems 4, p255-264.

Quinlan, J.R. (1983) Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., Mitchell, T. (Eds.) Machine Learning: An artificial intelligence approach p463-482.

Morgan Kaufmann, San Mateo CA.

Quinlan, J.R. (1986) Induction of Decision Trees . In Machine Learning 1, p81-106. Kluwer Academic Publishers, Boston MA.

Quinlan, J.R. (1987) Simplifying Decision Trees. In International Journal of Man-Machine Studies.

Quinlan, J.R. (1988) An empirical comparison of genetic and decision-tree classifiers. In Laird, J. (Ed.) Proceedings of the Fifth International Conference on Machine Learning p135-141. Morgan Kaufmann, San Mateo CA.

Rendell, L. (1986) A General Framework for Induction and a Study of Selective Induction. In Machine Learning 1, p177-226. Kluwer Academic Publishers, Boston MA.

Schlimmer, J.C., Fisher, D. (1986) A case study of incremental concept induction. In Proceedings of the Fifth National Conference on Artificial Intelligence p496-501. Morgan Kaufmann, San Mateo CA.

Schlimmer, J.C. (1987) Concept Acquisition through Representation Adjustment. Doctoral Dissertation. University of California, Irvine.

Seshu, R. (1989) Solving the Parity Problem. In Morik, K. (Ed.) Proceedings of the Fourth European Working Session on Learning p263-271. Pitman, London UK.

Steels, L. (1990) Components of Expertise. In AI Magazine. Summer 1990 p.

Tan, M., Schlimmer, J.C. (1989) Cost-Sensitive Concept Learning of Sensor Use in Approach and Recognition. In Segre, A.M. (Ed.) Proceedings of the Sixth International Workshop on Machine Learning p392-395. Morgan Kaufmann, San Mateo CA.

Utgoff, P.E. (1988a) ID5: An Incremental ID3. In Laird, J. (Ed.) Proceedings of the Fifth International Conference on Machine Learning p107-120. Morgan Kaufmann, San Mateo CA.

Utgoff, P.E. (1988b) Perceptron Trees: A case study in hybrid concept representations. In Proceedings of the Seventh National Conference on Artificial Intelligence, p601-606. Morgan Kaufmann, San Mateo CA.

Utgoff, P.E. (1989) Improved Training via Incremental Learning. In Segre, A.M. (Ed.) Proceedings of the Sixth International Workshop on Machine Learning p362-365. Morgan Kaufmann, San Mateo CA.

Utgoff, P.E. (1990) Incremental Learning of Decision Trees. In Machine Learning 4, p161-186. Kluwer Academic Publishers, Boston MA.

Utgoff, P.E., and Brodley, C.E. (1990) An Incremental Method for Finding Multivariate Splits for Decision Trees. In Porter, B.W., Mooney, R.J. (Eds.) Proceedings of the Seventh International Conference on Machine Learning, p58-65. Morgan Kaufmann, San Mateo CA.

Van de Velde, W. (1989) IDL, or Taming the Multiplexer. In Morik, K. (Ed.) Proceedings of the Fourth European Working Session on Learning p211-226. Pitman, London UK.

Van de Velde, W. (1990) Incremental Induction of Topologically Minimal Trees. In Porter, B.W., Mooney, R.J. (Eds.) Proceedings of the Seventh International Conference on Machine Learning, p66-74. Morgan Kaufmann, San Mateo CA.

Wilson, S.W. (1987) Classifier Systems and the Animal Problem. In Machine Learning 2 p . Kluwer Academic Publishers, Boston MA.

Wirth, J., Catlett, J. (1988) Experiments on the Costs and Benefits of Windowing in ID3. In Laird, J. (Ed.) Proceedings of the Fifth International Conference on Machine Learning p87-99. Morgan Kaufmann, San Mateo CA.

Zhang, J., Michalski, R.S. (1989) Rule Optimization via SG-Trunc Method. In Morik, K. (Ed.) Proceedings of the Fourth European Working Session on Learning p251-262. Pitman, London UK.

5.2 Experimental Results

I have done some of the experiments for the comparison of the algorithms. The runs on the first data-set are complete, except for the timing information. The runs for the second example are in progress and I will send them later today. I will not do the third example since I surender to noise. Nevertheless I think you will agree that in the class of decision tree algorithms, the performance of IDL is quite impressive.

Here is what I did. I ran several algorithms on the training-set and tested them on the test-set. If the algorrithm is non-incremental I used a run on the complete training set. If the algorithm is incremental I ran it with 500 examples randomly selected from the training set. Testing is always on the full test set. All results are averaged over 10 runs.

I used the following algorithms:

```
TDIDT:  plain old ID3 with information gain as selection measure,
no pruning.
ID5R:  the incremental version of ID3 produced by Utgoff. Information
gain is the selection measure. No pruning.
IDL: IDL as described in an unpublished paper, very similar to the
algorithm described in IML-90
ID5R-hat: ID5R with example filter. Trains only if the example is
misclassified by the current hypothsis. No pruning.
```

I send the results in several files. In seperate mails I will provide the following information:

```
TDIDT: the tree
size and accuracy of the tree
the concept described by it

ID5R, IDL, ID5R-hat:
data on size and accuracy as it evolves with training
a typical tree and its size and accuracy
the concept described by that typical tree
```

The evolving data for the incremental algorithms allow to produce the learning curves for each of the algorithms. I produced graphs with Exel and will send them by mail if I do not succeed making a postscript version of it.

About the results:

IDL is clearly the best. It produces the smallest trees with by far the best accuracy of all. It is also worth noticing that the standard deviations for IDL are very small, and that the concepts described by the trees that IDL produces are the same. This means that search in concept space is finished, but IDL can not decide on the best representation. So it limit-cycles between 3 different trees, all small and equally accurate (the only difference is in the order of testing the three relevant attributer). This illustrates how the use of not only statistical information bt also tree-topological one makes the algorithm unsensitive to sampling differences (small disjuncts or sparse sampling are no big problem either). Here are the data for all 10 trees to show this:

```
MONKS-1 IDL used IDL nodes IDL leaves IDL accuracy
500 500 42 29 97.22222
500 500 36 26 97.22222
```

```
500 500 42 29 97.22222
500 500 40 27 97.22222
500 500 40 27 97.22222
500 500 40 28 97.22222
500 500 36 26 97.22222
500 500 42 29 97.22222
500 500 40 27 97.22222
500 500 42 29 97.22222
```

On the other hand ID5R produces larger and less accurate trees with enormous standard deviations as shown by data for the 10 trees that ID5R produces:

```
MONKS-1 ID5R used ID5R nodes ID5R leaves ID5R accuracy
500 500 75 48 81.94444
500 500 64 40 81.71296
500 500 50 32 90.97222
500 500 61 40 87.73148
500 500 70 43 77.31481
500 500 40 27 97.22222
500 500 73 45 77.546295
500 500 78 50 84.02778
500 500 74 46 80.32407
500 500 59 37 86.34259
```

As expected ID5R-hat does somewhat better than ID5R. Here are the data for the 10 trees to give an idea of the deviations.

```
MONKS-1 ID5R-hat used ID5R-hat nodes ID5R-hat leaves ID5R-hat accuracy
500 51 56 36 85.416664
500 62 68 43 79.861115
500 52 40 27 97.22222
500 49 40 27 97.22222
500 53 40 27 97.22222
500 52 51 33 92.361115
500 50 39 26 94.44444
500 48 40 27 97.22222
500 58 49 32 90.27778
500 53 40 27 97.22222
```

I sent a number of files with the results of TDIDT, IDL, ID5R and ID5R-hat on the second monk's concept. The results are averaged only over 5 runs this time.

The effect I seem to get is that IDL does not get beyond its initial phase of building up a large tree. In other words, it does not get anyway near to collapsing it. The fact that it grows larger than for ID5R is not anomalous, but normally this is followed by a rapid collaps to a smaller form (see MONKS-1 this effect). This concept seems to be too difficult for trees to handle anyway...

Here are the 5 individual results for IDL:

```
MONKS-2 IDL used IDL nodes IDL leaves IDL accuracy
500 500 176 111 74.30556
```

```
500 500 170 104 65.046295
500 500 180 114 73.84259
500 500 197 112 68.05556
500 500 184 111 61.34259
```

Here are the 5 individual results for ID5R:

```
MONKS-2 ID5R used ID5R nodes ID5R leaves ID5R accuracy
500 500 145 93 64.12037
500 500 153 91 64.583336
500 500 173 104 65.74074
500 500 171 102 65.27778
500 500 165 95 61.805557
```

Here are the 5 individual results for ID5R-hat:

```
MONKS-2 ID5R-HAT used ID5R-HAT nodes ID5R-HAT leaves ID5R-HAT accuracy
500 113 130 77 63.425926
500 115 131 82 65.74074
500 118 133 80 64.81481
500 120 133 84 62.5
500 115 138 83 62.73148
```

IDL finds larger trees, slightly more accurate. ID5R and ID5R-HAT find trees that are comparable in accuracy to the TDIDT tree (66.666664% with 159 nodes and 95 leaves) but the ID5R-HAT tree is smaller.

5.2.1 ID5R on test set 1

```
DESCRIPTION OF THE TREE:

;; Typical tree found by id5r
;; trained on first monks's training set
;;
;; 500 examples (random from full
;;     training set)
;; 64 nodes
;; 40 leaves
;; 81.71296 accuracy on test set


JACKET_COLOR = 1 : <1>...
JACKET_COLOR = 2 :
   HAS_TIE = 1 :
      BODY_SHAPE = 1 : <1>...
      BODY_SHAPE = 2 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 2 : <1>...
         HEAD_SHAPE = 3 : <0>...
      BODY_SHAPE = 3 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 2 : <0>...
         HEAD_SHAPE = 3 : <1>...
   HAS_TIE = 2 :
      BODY_SHAPE = 1 :
         HEAD_SHAPE = 1 : <1>...
         HEAD_SHAPE = 2 : <0>...
         HEAD_SHAPE = 3 : <0>...
      BODY_SHAPE = 2 :
         IS_SMILING = 1 : <1>...
         IS_SMILING = 2 : <0>...
      BODY_SHAPE = 3 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 3 : <1>...
JACKET_COLOR = 3 :
   HOLDING = 1 :
      HEAD_SHAPE = 1 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 : <0>
      HEAD_SHAPE = 2 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 : <0>...
      HEAD_SHAPE = 3 :
         BODY_SHAPE = 2 : <0>
         BODY_SHAPE = 3 : <1>...
   HOLDING = 2 :
      HAS_TIE = 1 : <0>...
      HAS_TIE = 2 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 2 : <1>...
         HEAD_SHAPE = 3 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
   HOLDING = 3 :
      IS_SMILING = 1 :
         HAS_TIE = 1 : <0>...
         HAS_TIE = 2 : <1>...
      IS_SMILING = 2 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 2 : <0>...
         HEAD_SHAPE = 3 : <1>...
JACKET_COLOR = 4 :
   HEAD_SHAPE = 1 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 : <0>...
         BODY_SHAPE = 3 : <0>...
   HEAD_SHAPE = 2 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 : <0>...
   HEAD_SHAPE = 3 :
         BODY_SHAPE = 2 : <0>...
         BODY_SHAPE = 3 : <1>...
```

5.2.2 IDL on test set 1

```
DESCRIPTION OF THE TREE:


;; Typical tree found by idl
;; trained on first monks's training set
;;
;; 500 examples (random from
;;     full training set)
;; 36 nodes
;; 26 leaves
;; 97.22222 accuracy on test set

BODY_SHAPE = 1 :
   HEAD_SHAPE = 1 : <1>...
   HEAD_SHAPE = 2 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>...
      JACKET_COLOR = 4 : <0>...
   HEAD_SHAPE = 3 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>...
BODY_SHAPE = 2 :
   HEAD_SHAPE = 1 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>...
      JACKET_COLOR = 4 : <0>...
   HEAD_SHAPE = 2 : <1>...
   HEAD_SHAPE = 3 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>
      JACKET_COLOR = 4 : <0>...
BODY_SHAPE = 3 :
   HEAD_SHAPE = 1 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>...
      JACKET_COLOR = 4 : <0>...
   HEAD_SHAPE = 2 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>...
      JACKET_COLOR = 4 : <0>...
   HEAD_SHAPE = 3 : <1>...
```

5.2.3 ID5R-HAT on test set 1

```
DESCRIPTION OF THE TREE:

;; Tree found by id5r-hat trained
;; on first monks's training set
;;
;; 58 examples used out of 500
;; (random from full training set)
;; 49 nodes
;; 32 leaves
;; 90.27778 accuracy on test set


JACKET_COLOR = 1 : <1>...
JACKET_COLOR = 2 :
   HOLDING = 1 :
      HEAD_SHAPE = 1 :
         BODY_SHAPE = 3 : <0>
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 : <0>...
      HEAD_SHAPE = 2 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 : <1>...
      HEAD_SHAPE = 3 : <0>...
   HOLDING = 2 : <0>...
   HOLDING = 3 :
      BODY_SHAPE = 1 :
         HAS_TIE = 1 : <1>...
         HAS_TIE = 2 : <0>...
      BODY_SHAPE = 2 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 2 : <1>...
         HEAD_SHAPE = 3 : <0>...
      BODY_SHAPE = 3 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 3 : <1>...
JACKET_COLOR = 3 :
   HEAD_SHAPE = 1 :
      BODY_SHAPE = 1 : <1>...
      BODY_SHAPE = 2 : <0>...
      BODY_SHAPE = 3 : <0>...
   HEAD_SHAPE = 2 :
      BODY_SHAPE = 1 : <0>...
      BODY_SHAPE = 2 : <1>...
      BODY_SHAPE = 3 : <0>...
   HEAD_SHAPE = 3 :
      BODY_SHAPE = 1 : <0>...
      BODY_SHAPE = 2 : <0>...
      BODY_SHAPE = 3 : <1>...
JACKET_COLOR = 4 :
   HEAD_SHAPE = 1 :
      BODY_SHAPE = 1 : <1>...
      BODY_SHAPE = 2 : <0>...
      BODY_SHAPE = 3 : <0>...
   HEAD_SHAPE = 2 :
      BODY_SHAPE = 3 : <0>
      BODY_SHAPE = 1 : <0>...
      BODY_SHAPE = 2 : <1>...
   HEAD_SHAPE = 3 :
      BODY_SHAPE = 2 : <0>...
      BODY_SHAPE = 3 : <1>...
```

5.2.4 TDIDT on test set 1

```
DESCRIPTION OF THE TREE:

;; Tree found by tdidt trained
;; on first monks's training set
;;
;; 124 examples (full training set)
;; 86 nodes
;; 52 leaves
;; 75.69444 accuracy on test set

JACKET_COLOR = 1 : <1>
JACKET_COLOR = 2 :
   HOLDING = 1 :
      HEAD_SHAPE = 1 :
         BODY_SHAPE = 1 : <1>
         BODY_SHAPE = 2 : <0>
         BODY_SHAPE = 3 : <0>
      HEAD_SHAPE = 2 :
         IS_SMILING = 1 : <1>
         IS_SMILING = 2 : <0>
      HEAD_SHAPE = 3 :
         HAS_TIE = 1 : <1>
         HAS_TIE = 2 : <0>
   HOLDING = 2 :
      BODY_SHAPE = 1 : <0>
      BODY_SHAPE = 2 : <1>
      BODY_SHAPE = 3 : <0>
   HOLDING = 3 :
      IS_SMILING = 1 :
         HEAD_SHAPE = 1 :
            BODY_SHAPE = 1 : <1>
            BODY_SHAPE = 2 : <0>
            BODY_SHAPE = 3 : <0>
         HEAD_SHAPE = 2 : <1>
         HEAD_SHAPE = 3 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 3 : <1>
      IS_SMILING = 2 :
         BODY_SHAPE = 1 : <0>
         BODY_SHAPE = 2 : <0>
         BODY_SHAPE = 3 :
            HAS_TIE = 1 : <0>
            HAS_TIE = 2 : <1>
JACKET_COLOR = 3 :
   HAS_TIE = 1 :
      HOLDING = 1 :
         IS_SMILING = 1 :
            BODY_SHAPE = 1 :
               HEAD_SHAPE = 1 : <1>
               HEAD_SHAPE = 2 : <0>
            BODY_SHAPE = 2 :
               HEAD_SHAPE = 1 : <0>
               HEAD_SHAPE = 2 : <1>
         IS_SMILING = 2 : <0>
      HOLDING = 2 :
         IS_SMILING = 1 : <0>
         IS_SMILING = 2 :
            HEAD_SHAPE = 1 : <1>
            HEAD_SHAPE = 2 : <0>
      HOLDING = 3 : <0>
   HAS_TIE = 2 :
      IS_SMILING = 1 :
         HOLDING = 1 :
            HEAD_SHAPE = 1 : <1>
```

```
            HEAD_SHAPE = 2 : <0>
         HOLDING = 2 :
            HEAD_SHAPE = 1 : <0>
            HEAD_SHAPE = 2 : <1>
            HEAD_SHAPE = 3 : <1>
         HOLDING = 3 : <1>
      IS_SMILING = 2 :
         HOLDING = 1 :
            BODY_SHAPE = 2 :
               HEAD_SHAPE = 2 : <1>
               HEAD_SHAPE = 3 : <0>
            BODY_SHAPE = 3 :
               HEAD_SHAPE = 2 : <0>
               HEAD_SHAPE = 3 : <1>
         HOLDING = 2 : <0>
         HOLDING = 3 :
            HEAD_SHAPE = 1 : <0>
            HEAD_SHAPE = 2 : <0>
            HEAD_SHAPE = 3 : <1>
JACKET_COLOR = 4 :
   HEAD_SHAPE = 1 :
      BODY_SHAPE = 1 : <1>
      BODY_SHAPE = 2 : <0>
      BODY_SHAPE = 3 : <0>
   HEAD_SHAPE = 2 :
      BODY_SHAPE = 1 : <0>
      BODY_SHAPE = 2 : <1>
      BODY_SHAPE = 3 : <0>
   HEAD_SHAPE = 3 :
      BODY_SHAPE = 2 : <0>
      BODY_SHAPE = 3 : <1>(1 1 1 1 1 1 -> 1)
```

5.2.5 ID5R on test set 2

```
DESCRIPTION OF THE TREE:

;; Typical tree found by id5r
;; trained on second monks's
;; training set
;;
;; 500 examples (random from
;;     full training set)
;; 165 nodes
;; 95 leaves
;; 61.805557 accuracy on test set


JACKET_COLOR = 1 :
   IS_SMILING = 1 :
      HEAD_SHAPE = 1 : <0>...
      HEAD_SHAPE = 2 :
         HOLDING = 1 : <0>...
         HOLDING = 2 : <0>...
         HOLDING = 3 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 3 : <1>...
      HEAD_SHAPE = 3 :
         HOLDING = 1 : <0>...
         HOLDING = 2 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 2 : <1>
         HOLDING = 3 : <0>
   IS_SMILING = 2 :
      HOLDING = 1 :
         HAS_TIE = 1 : <0>...
         HAS_TIE = 2 :
            HEAD_SHAPE = 1 : <0>
            HEAD_SHAPE = 2 : <1>...
      HOLDING = 2 : <1>...
      HOLDING = 3 :
         HEAD_SHAPE = 1 :
            HAS_TIE = 1 : <0>...
            HAS_TIE = 2 :
               BODY_SHAPE = 1 : <0>
               BODY_SHAPE = 2 : <1>
         HEAD_SHAPE = 2 :
            HAS_TIE = 1 :
               BODY_SHAPE = 1 : <0>
               BODY_SHAPE = 2 : <1>
            HAS_TIE = 2 : <1>...
         HEAD_SHAPE = 3 : <1>...
JACKET_COLOR = 2 :
   BODY_SHAPE = 1 :
      HEAD_SHAPE = 1 :
         IS_SMILING = 1 : <0>...
         IS_SMILING = 2 : <1>...
      HEAD_SHAPE = 2 :
         HOLDING = 1 :
            IS_SMILING = 1 : <0>
            IS_SMILING = 2 : <1>...
         HOLDING = 2 : <1>...
         HOLDING = 3 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
      HEAD_SHAPE = 3 :
         HOLDING = 1 : <1>...
         HOLDING = 2 : <1>...
         HOLDING = 3 :
            IS_SMILING = 1 : <1>
            IS_SMILING = 2 :
               HAS_TIE = 1 : <1>
               HAS_TIE = 2 : <0>
   BODY_SHAPE = 2 :
      IS_SMILING = 1 : <1>...
      IS_SMILING = 2 : <0>...
   BODY_SHAPE = 3 :
      HEAD_SHAPE = 1 :
         HOLDING = 2 : <0>...
         HOLDING = 3 : <1>...
      HEAD_SHAPE = 2 :
         HOLDING = 1 : <1>...
         HOLDING = 2 : <0>...
         HOLDING = 3 : <1>...
      HEAD_SHAPE = 3 : <0>...
JACKET_COLOR = 3 :
   HEAD_SHAPE = 1 :
      BODY_SHAPE = 1 : <0>...
      BODY_SHAPE = 2 :
         HOLDING = 2 :
            HAS_TIE = 1 :
               IS_SMILING = 1 : <0>
               IS_SMILING = 2 : <1>
            HAS_TIE = 2 : <1>...
         HOLDING = 3 :
            IS_SMILING = 1 : <0>...
            IS_SMILING = 2 :
               HAS_TIE = 1 : <1>
```

```
               HAS_TIE = 2 : <0>
      BODY_SHAPE = 3 :
         HAS_TIE = 1 : <0>...
         HAS_TIE = 2 :
            HOLDING = 1 :
               IS_SMILING = 1 : <0>
               IS_SMILING = 2 : <1>
            HOLDING = 2 : <0>...
   HEAD_SHAPE = 2 :
      IS_SMILING = 1 :
         BODY_SHAPE = 1 :
            HAS_TIE = 1 : <0>
            HAS_TIE = 2 : <1>...
         BODY_SHAPE = 2 :
            HOLDING = 1 : <0>...
            HOLDING = 2 : <1>...
            HOLDING = 3 :
               HAS_TIE = 1 : <1>
               HAS_TIE = 2 : <0>
         BODY_SHAPE = 3 :
            HOLDING = 1 : <1>...
            HOLDING = 2 :
               HAS_TIE = 1 : <1>
               HAS_TIE = 2 : <0>
      IS_SMILING = 2 :
         HOLDING = 1 :
            HAS_TIE = 1 : <1>...
            HAS_TIE = 2 : <0>...
         HOLDING = 2 : <0>...
         HOLDING = 3 : <0>...
   HEAD_SHAPE = 3 :
      HOLDING = 1 :
         HAS_TIE = 1 :
            BODY_SHAPE = 1 : <0>...
            BODY_SHAPE = 2 :
               IS_SMILING = 1 : <0>
               IS_SMILING = 2 : <1>
            BODY_SHAPE = 3 : <0>...
         HAS_TIE = 2 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 :
               BODY_SHAPE = 1 : <1>
               BODY_SHAPE = 2 : <0>
      HOLDING = 2 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 : <0>...
         BODY_SHAPE = 3 : <0>...
      HOLDING = 3 :
         HAS_TIE = 1 : <1>...
         HAS_TIE = 2 :
            BODY_SHAPE = 1 : <1>...
            BODY_SHAPE = 2 : <0>...
JACKET_COLOR = 4 :
   BODY_SHAPE = 1 :
      HAS_TIE = 1 : <0>...
      HAS_TIE = 2 :
         HEAD_SHAPE = 1 : <1>...
         HEAD_SHAPE = 2 : <0>...
         HEAD_SHAPE = 3 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
   BODY_SHAPE = 2 :
      HOLDING = 1 :
         IS_SMILING = 1 : <1>...
         IS_SMILING = 2 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 :
               HAS_TIE = 1 : <1>
               HAS_TIE = 2 : <0>
      HOLDING = 2 : <1>...
      HOLDING = 3 :
         HAS_TIE = 1 :
            IS_SMILING = 1 :
               HEAD_SHAPE = 1 : <0>
               HEAD_SHAPE = 2 : <1>
            IS_SMILING = 2 : <1>...
         HAS_TIE = 2 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
   BODY_SHAPE = 3 :
      IS_SMILING = 1 :
         HOLDING = 2 : <0>...
         HOLDING = 3 : <1>...
      IS_SMILING = 2 : <0>...
```

5.2.6 IDL on test set 2

```
DESCRIPTION OF THE TREE:

;; Typical tree found by idl
;; trained on second monks's
;; training set
;;
;; 500 examples (random from
;;     full training set)
;; 170 nodes
;; 107 leaves
;; 66.203705 accuracy on test set


IS_SMILING = 1 :
   HAS_TIE = 1 :
      JACKET_COLOR = 1 : <0>...
      JACKET_COLOR = 2 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 : <1>...
            HEAD_SHAPE = 3 : <0>...
      JACKET_COLOR = 3 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
               HOLDING = 3 : <1>
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <0>
               HOLDING = 3 : <1>
         BODY_SHAPE = 3 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 : <1>...
            HEAD_SHAPE = 3 : <0>...
      JACKET_COLOR = 4 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 :
            HEAD_SHAPE = 1 : <0>...
```

```
            HEAD_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 : <1>...
   HAS_TIE = 2 :
      JACKET_COLOR = 1 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
         BODY_SHAPE = 3 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <0>
               HOLDING = 3 : <1>
      JACKET_COLOR = 2 :
         BODY_SHAPE = 1 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
               HOLDING = 3 : <1>
            HEAD_SHAPE = 3 : <1>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 :
            HOLDING = 1 : <0>...
            HOLDING = 2 : <1>...
            HOLDING = 3 : <1>...
      JACKET_COLOR = 3 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 :
            HEAD_SHAPE = 1 : <1>...
            HEAD_SHAPE = 2 : <0>...
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <1>
               HOLDING = 3 : <0>
         BODY_SHAPE = 3 :
            HEAD_SHAPE = 1 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <1>
               HOLDING = 2 : <0>
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <1>
               HOLDING = 2 : <0>
      JACKET_COLOR = 4 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 :
            HEAD_SHAPE = 1 : <1>...
            HEAD_SHAPE = 2 : <0>...
IS_SMILING = 2 :
   BODY_SHAPE = 1 :
      HAS_TIE = 1 :
         JACKET_COLOR = 1 : <0>...
         JACKET_COLOR = 2 : <1>...
         JACKET_COLOR = 3 : <0>...
         JACKET_COLOR = 4 : <0>...
      HAS_TIE = 2 :
         JACKET_COLOR = 1 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 : <1>...
            HEAD_SHAPE = 3 : <1>...
         JACKET_COLOR = 2 :
            HEAD_SHAPE = 1 : <1>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <1>
               HOLDING = 3 : <0>
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <1>
               HOLDING = 3 : <0>
         JACKET_COLOR = 3 :
            HEAD_SHAPE = 2 : <0>...
            HEAD_SHAPE = 3 : <1>...
         JACKET_COLOR = 4 :
            HEAD_SHAPE = 1 : <1>...
            HEAD_SHAPE = 2 : <0>...
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <1>
               HOLDING = 3 : <0>
   BODY_SHAPE = 2 :
      HAS_TIE = 1 :
         JACKET_COLOR = 1 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
               HOLDING = 3 : <1>
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <0>
               HOLDING = 3 : <1>
         JACKET_COLOR = 2 : <0>...
         JACKET_COLOR = 3 :
            HEAD_SHAPE = 1 : <1>...
            HEAD_SHAPE = 2 : <0>...
            HEAD_SHAPE = 3 : <1>...
         JACKET_COLOR = 4 :
            HEAD_SHAPE = 1 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
               HOLDING = 3 : <1>
            HEAD_SHAPE = 2 : <1>...
      HAS_TIE = 2 :
         JACKET_COLOR = 1 : <1>...
         JACKET_COLOR = 2 : <0>...
         JACKET_COLOR = 3 : <0>...
         JACKET_COLOR = 4 : <0>...
   BODY_SHAPE = 3 :
      HAS_TIE = 1 :
         JACKET_COLOR = 1 :
            HEAD_SHAPE = 2 : <1>...
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <0>
               HOLDING = 2 : <1>
               HOLDING = 3 : <1>
         JACKET_COLOR = 2 :
            HEAD_SHAPE = 1 : <1>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <1>
               HOLDING = 2 : <0>
            HEAD_SHAPE = 3 : <0>...
         JACKET_COLOR = 3 :
            HEAD_SHAPE = 1 : <0>...
            HEAD_SHAPE = 2 :
               HOLDING = 1 : <1>
               HOLDING = 3 : <0>
            HEAD_SHAPE = 3 : <0>...
         JACKET_COLOR = 4 : <0>...
      HAS_TIE = 2 :
         JACKET_COLOR = 1 : <1>...
         JACKET_COLOR = 3 :
            HEAD_SHAPE = 1 :
               HOLDING = 1 : <1>
```

```
               HOLDING = 2 : <0>
            HEAD_SHAPE = 2 : <0>...
            HEAD_SHAPE = 3 : <0>...
         JACKET_COLOR = 4 : <0>...
```

5.2.7 TDIDT on test set 2

```
DESCRIPTION OF THE TREE:

;; The tree found by TDIDT
;; trained on second monks's
;;  training set
;;
;; 169 examples (full training set)
;; 159 nodes
;; 95 leaves
;; 66.666664 accuracy on test set



JACKET_COLOR = 1 :
   IS_SMILING = 1 :
      HAS_TIE = 1 : <0>
      HAS_TIE = 2 :
         HEAD_SHAPE = 1 : <0>
         HEAD_SHAPE = 2 :
            HOLDING = 1 : <0>
            HOLDING = 3 :
               BODY_SHAPE = 1 : <0>
               BODY_SHAPE = 3 : <1>
         HEAD_SHAPE = 3 :
            HOLDING = 1 : <0>
            HOLDING = 2 :
               BODY_SHAPE = 1 : <0>
               BODY_SHAPE = 2 : <1>
            HOLDING = 3 : <0>
   IS_SMILING = 2 :
      HOLDING = 1 :
         HAS_TIE = 1 : <0>
         HAS_TIE = 2 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 3 : <1>
      HOLDING = 2 : <1>
      HOLDING = 3 :
         HEAD_SHAPE = 1 :
            HAS_TIE = 1 : <0>
            HAS_TIE = 2 :
               BODY_SHAPE = 1 : <0>
               BODY_SHAPE = 2 : <1>
         HEAD_SHAPE = 2 :
            HAS_TIE = 1 :
               BODY_SHAPE = 1 : <0>
               BODY_SHAPE = 2 : <1>
            HAS_TIE = 2 : <1>
         HEAD_SHAPE = 3 : <1>
JACKET_COLOR = 2 :
   IS_SMILING = 1 :
      HOLDING = 1 :
         BODY_SHAPE = 1 : <0>
         BODY_SHAPE = 2 : <1>
         BODY_SHAPE = 3 : <0>
      HOLDING = 2 :
         HAS_TIE = 1 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 2 : <1>
            BODY_SHAPE = 3 : <0>
         HAS_TIE = 2 : <1>
      HOLDING = 3 :
         HAS_TIE = 1 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 2 : <1>
            BODY_SHAPE = 3 : <1>
         HAS_TIE = 2 : <1>
   IS_SMILING = 2 :
      BODY_SHAPE = 1 :
         HOLDING = 1 : <1>
         HOLDING = 2 : <1>
         HOLDING = 3 :
            HEAD_SHAPE = 1 : <1>
            HEAD_SHAPE = 2 : <0>
            HEAD_SHAPE = 3 :
               HAS_TIE = 1 : <1>
               HAS_TIE = 2 : <0>
      BODY_SHAPE = 2 : <0>
      BODY_SHAPE = 3 :
         HEAD_SHAPE = 1 : <1>
         HEAD_SHAPE = 2 :
            HOLDING = 1 : <1>
            HOLDING = 2 : <0>
         HEAD_SHAPE = 3 : <0>
JACKET_COLOR = 3 :
   IS_SMILING = 1 :
      HAS_TIE = 1 :
         HEAD_SHAPE = 1 : <0>
         HEAD_SHAPE = 2 :
            HOLDING = 1 : <0>
            HOLDING = 2 : <1>
            HOLDING = 3 : <1>
         HEAD_SHAPE = 3 :
            HOLDING = 1 : <0>
            HOLDING = 3 : <1>
      HAS_TIE = 2 :
         BODY_SHAPE = 1 : <1>
         BODY_SHAPE = 2 :
            HEAD_SHAPE = 1 : <1>
            HEAD_SHAPE = 2 : <0>
            HEAD_SHAPE = 3 :
               HOLDING = 1 : <1>
               HOLDING = 3 : <0>
         BODY_SHAPE = 3 :
            HOLDING = 1 :
               HEAD_SHAPE = 1 : <0>
               HEAD_SHAPE = 2 : <1>
               HEAD_SHAPE = 3 : <1>
            HOLDING = 2 :
               HEAD_SHAPE = 1 : <1>
               HEAD_SHAPE = 2 : <0>
               HEAD_SHAPE = 3 : <0>
   IS_SMILING = 2 :
      HOLDING = 1 :
         BODY_SHAPE = 1 :
            HAS_TIE = 1 : <0>
            HAS_TIE = 2 : <1>
         BODY_SHAPE = 2 :
            HAS_TIE = 1 : <1>
            HAS_TIE = 2 : <0>
         BODY_SHAPE = 3 :
            HAS_TIE = 1 :
               HEAD_SHAPE = 1 : <0>
               HEAD_SHAPE = 2 : <1>.
```

```
            HAS_TIE = 2 : <1>
      HOLDING = 2 :
         HEAD_SHAPE = 1 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 2 : <1>
            BODY_SHAPE = 3 : <0>
         HEAD_SHAPE = 2 : <0>
         HEAD_SHAPE = 3 : <0>
      HOLDING = 3 :
         HAS_TIE = 1 :
            BODY_SHAPE = 2 : <1>
            BODY_SHAPE = 3 : <0>
         HAS_TIE = 2 : <0>
JACKET_COLOR = 4 :
   BODY_SHAPE = 1 :
      HAS_TIE = 1 : <0>
      HAS_TIE = 2 :
         HOLDING = 1 : <1>
         HOLDING = 2 :
            HEAD_SHAPE = 1 : <1>
            HEAD_SHAPE = 2 : <0>
            HEAD_SHAPE = 3 : <1>
         HOLDING = 3 : <0>
   BODY_SHAPE = 2 :
      HEAD_SHAPE = 1 :
         HOLDING = 1 : <0>
         HOLDING = 2 :
            IS_SMILING = 1 : <0>
            IS_SMILING = 2 : <1>
         HOLDING = 3 :
            HAS_TIE = 1 :
               IS_SMILING = 1 : <0>
               IS_SMILING = 2 : <1>
            HAS_TIE = 2 :
               IS_SMILING = 1 : <1>
               IS_SMILING = 2 : <0>
      HEAD_SHAPE = 2 :
         HAS_TIE = 1 : <1>
         HAS_TIE = 2 :
            IS_SMILING = 1 : <1>
            IS_SMILING = 2 : <0>
      HEAD_SHAPE = 3 : <0>
   BODY_SHAPE = 3 :
      HOLDING = 1 : <0>
      HOLDING = 2 : <0>
      HOLDING = 3 :
         IS_SMILING = 1 : <1>
         IS_SMILING = 2 : <0>
```

5.2.8 TDIDT on test set 1

```
DESCRIPTION OF THE TREE:

;; Tree found by tdidt
;; trained on first monks's
;; training set
;; 124 examples (full training set)
;; 86 nodes
;; 52 leaves
;; 75.69444 accuracy on test set
```

```
JACKET_COLOR = 1 : <1>
JACKET_COLOR = 2 :
   HOLDING = 1 :
      HEAD_SHAPE = 1 :
         BODY_SHAPE = 1 : <1>
         BODY_SHAPE = 2 : <0>
         BODY_SHAPE = 3 : <0>
      HEAD_SHAPE = 2 :
         IS_SMILING = 1 : <1>
         IS_SMILING = 2 : <0>
      HEAD_SHAPE = 3 :
         HAS_TIE = 1 : <1>
         HAS_TIE = 2 : <0>
   HOLDING = 2 :
      BODY_SHAPE = 1 : <0>
      BODY_SHAPE = 2 : <1>
      BODY_SHAPE = 3 : <0>
   HOLDING = 3 :
      IS_SMILING = 1 :
         HEAD_SHAPE = 1 :
            BODY_SHAPE = 1 : <1>
            BODY_SHAPE = 2 : <0>
            BODY_SHAPE = 3 : <0>
         HEAD_SHAPE = 2 : <1>
         HEAD_SHAPE = 3 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 3 : <1>
      IS_SMILING = 2 :
         BODY_SHAPE = 1 : <0>
         BODY_SHAPE = 2 : <0>
         BODY_SHAPE = 3 :
            HAS_TIE = 1 : <0>
            HAS_TIE = 2 : <1>
JACKET_COLOR = 3 :
   HAS_TIE = 1 :
      HOLDING = 1 :
         IS_SMILING = 1 :
            BODY_SHAPE = 1 :
               HEAD_SHAPE = 1 : <1>
               HEAD_SHAPE = 2 : <0>
            BODY_SHAPE = 2 :
               HEAD_SHAPE = 1 : <0>
               HEAD_SHAPE = 2 : <1>
         IS_SMILING = 2 : <0>
      HOLDING = 2 :
         IS_SMILING = 1 : <0>
         IS_SMILING = 2 :
            HEAD_SHAPE = 1 : <1>
            HEAD_SHAPE = 2 : <0>
      HOLDING = 3 : <0>
   HAS_TIE = 2 :
      IS_SMILING = 1 :
         HOLDING = 1 :
            HEAD_SHAPE = 1 : <1>
            HEAD_SHAPE = 2 : <0>
         HOLDING = 2 :
            HEAD_SHAPE = 1 : <0>
            HEAD_SHAPE = 2 : <1>
            HEAD_SHAPE = 3 : <1>
         HOLDING = 3 : <1>
      IS_SMILING = 2 :
         HOLDING = 1 :
            BODY_SHAPE = 2 :
               HEAD_SHAPE = 2 : <1>
               HEAD_SHAPE = 3 : <0>
            BODY_SHAPE = 3 :
               HEAD_SHAPE = 2 : <0>
               HEAD_SHAPE = 3 : <1>
```

```
         HOLDING = 2 : <0>
         HOLDING = 3 :
            HEAD_SHAPE = 1 : <0>
            HEAD_SHAPE = 2 : <0>
            HEAD_SHAPE = 3 : <1>
JACKET_COLOR = 4 :
   HEAD_SHAPE = 1 :
      BODY_SHAPE = 1 : <1>
      BODY_SHAPE = 2 : <0>
      BODY_SHAPE = 3 : <0>
   HEAD_SHAPE = 2 :
      BODY_SHAPE = 1 : <0>
      BODY_SHAPE = 2 : <1>
      BODY_SHAPE = 3 : <0>
   HEAD_SHAPE = 3 :
      BODY_SHAPE = 2 : <0>
      BODY_SHAPE = 3 : <1>(1 1 1 1 1 1 -> 1)
```

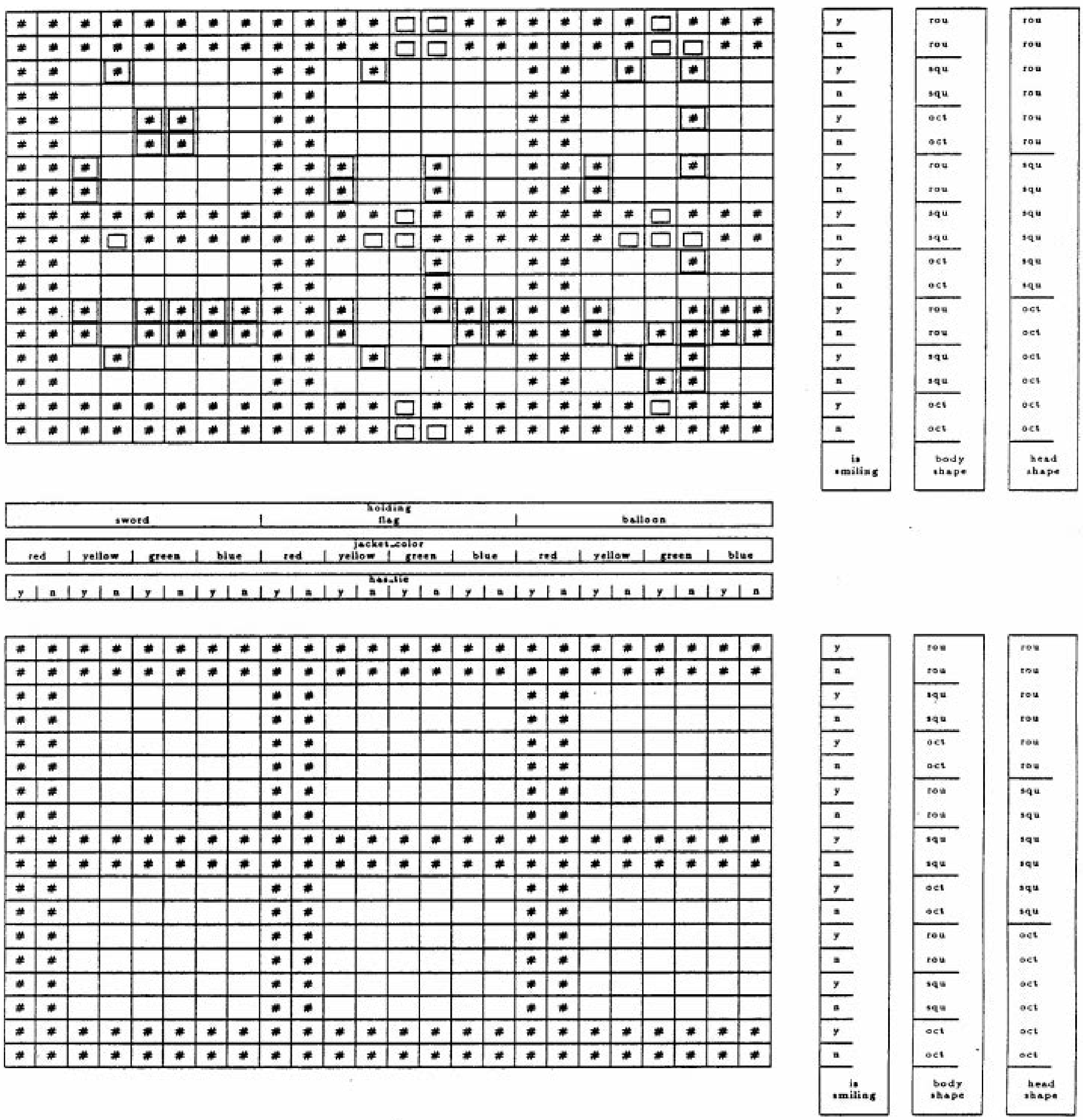
5.2.9 ID5R-HAT on test set 2

```
DESCRIPTION OF THE TREE:

;; Typical tree found by id5r-hat
;; trained on second
;; monks's training set
;;
;; 115 examples used out of 500
;; (random from full training set)
;; 131 nodes
;; 82 leaves
;; 65.74074 accuracy on test set


HOLDING = 1 :
   HAS_TIE = 1 :
      JACKET_COLOR = 1 : <0>...
      JACKET_COLOR = 2 :
         HEAD_SHAPE = 1 : <0>...
         HEAD_SHAPE = 2 : <1>...
         HEAD_SHAPE = 3 : <0>...
      JACKET_COLOR = 3 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 :
            IS_SMILING = 1 : <0>...
            IS_SMILING = 2 : <1>...
         BODY_SHAPE = 3 : <0>...
      JACKET_COLOR = 4 : <0>...
   HAS_TIE = 2 :
      BODY_SHAPE = 1 :
         IS_SMILING = 1 : <0>...
         IS_SMILING = 2 : <1>...
      BODY_SHAPE = 2 :
         IS_SMILING = 1 :
            JACKET_COLOR = 1 : <0>...
            JACKET_COLOR = 2 : <1>...
            JACKET_COLOR = 4 : <1>...
         IS_SMILING = 2 : <0>...
      BODY_SHAPE = 3 :
         JACKET_COLOR = 2 : <0>
         JACKET_COLOR = 1 :
            IS_SMILING = 1 : <0>...
            IS_SMILING = 2 : <1>...
         JACKET_COLOR = 3 :
            HEAD_SHAPE = 1 :
               IS_SMILING = 1 : <0>
               IS_SMILING = 2 : <1>
            HEAD_SHAPE = 2 : <1>...
            HEAD_SHAPE = 3 : <1>...
         JACKET_COLOR = 4 : <0>...
HOLDING = 2 :
   HEAD_SHAPE = 1 :
      HAS_TIE = 1 :
         IS_SMILING = 1 : <0>...
         IS_SMILING = 2 :
            BODY_SHAPE = 1 : <0>...
            BODY_SHAPE = 2 : <1>...
      HAS_TIE = 2 :
         JACKET_COLOR = 1 :
            IS_SMILING = 1 : <0>...
            IS_SMILING = 2 : <1>...
         JACKET_COLOR = 2 : <1>...
         JACKET_COLOR = 3 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
         JACKET_COLOR = 4 :
            BODY_SHAPE = 1 : <1>...
            BODY_SHAPE = 3 : <0>...
   HEAD_SHAPE = 2 :
      JACKET_COLOR = 1 :
         IS_SMILING = 1 : <0>...
         IS_SMILING = 2 : <1>...
      JACKET_COLOR = 2 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 : <0>...
         BODY_SHAPE = 3 : <0>...
      JACKET_COLOR = 3 :
         HAS_TIE = 1 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
         HAS_TIE = 2 : <0>...
      JACKET_COLOR = 4 : <0>...
   HEAD_SHAPE = 3 :
      JACKET_COLOR = 1 :
         BODY_SHAPE = 1 : <0>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 : <1>...
      JACKET_COLOR = 2 :
         BODY_SHAPE = 1 : <1>...
         BODY_SHAPE = 2 :
            IS_SMILING = 1 : <1>...
            IS_SMILING = 2 : <0>...
         BODY_SHAPE = 3 : <0>...
      JACKET_COLOR = 3 : <0>...
      JACKET_COLOR = 4 : <1>...
HOLDING = 3 :
   IS_SMILING = 1 :
      HEAD_SHAPE = 1 :
         HAS_TIE = 1 : <0>...
         HAS_TIE = 2 : <1>...
      HEAD_SHAPE = 2 :
         BODY_SHAPE = 1 :
            JACKET_COLOR = 1 : <0>...
            JACKET_COLOR = 2 : <1>...
            JACKET_COLOR = 3 : <1>...
            JACKET_COLOR = 4 : <0>...
         BODY_SHAPE = 2 : <1>...
         BODY_SHAPE = 3 : <1>...
      HEAD_SHAPE = 3 :
         JACKET_COLOR = 1 : <0>...
```

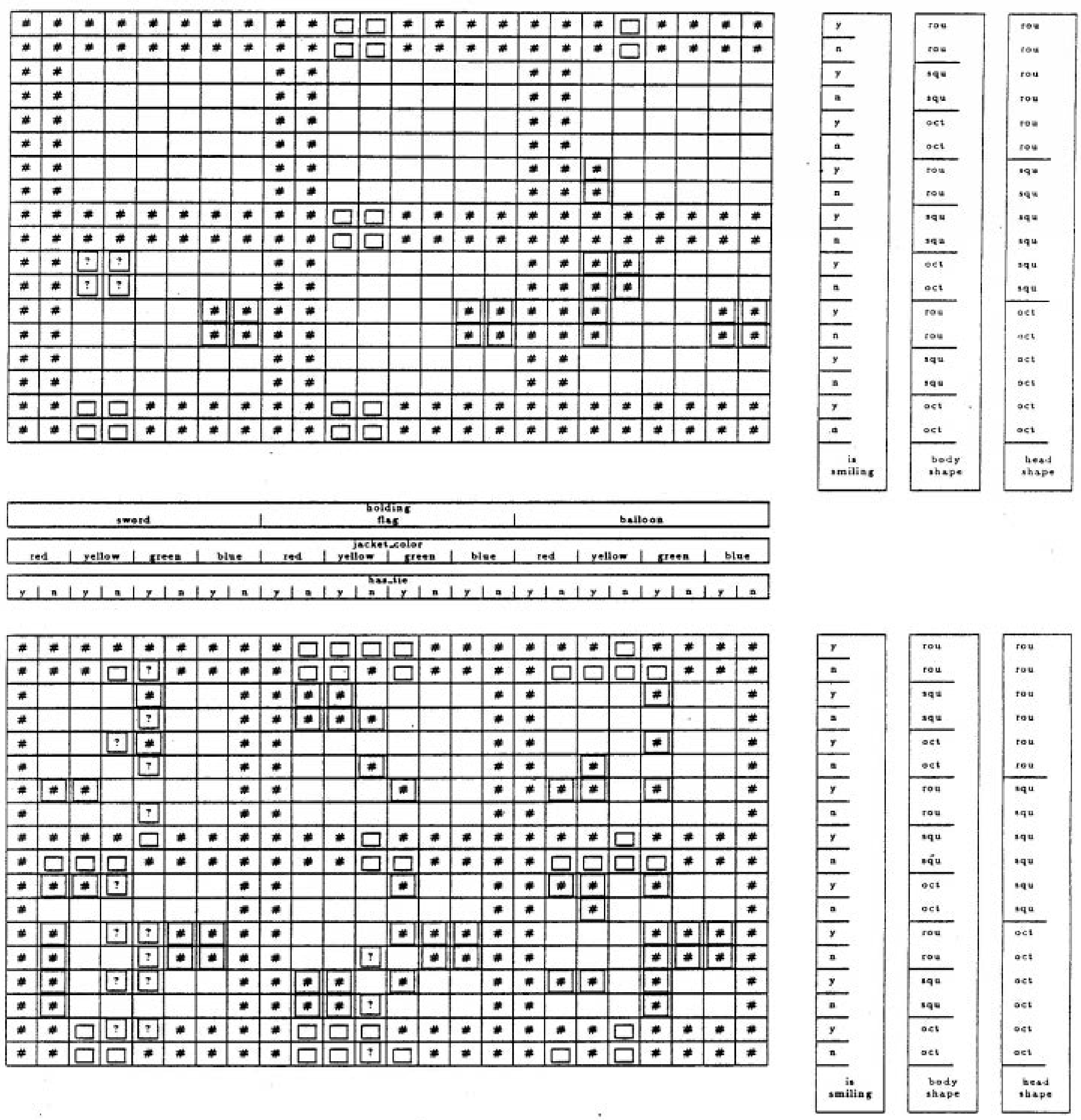
```
      JACKET_COLOR = 2 : <1>...
      JACKET_COLOR = 3 :
         HAS_TIE = 1 : <1>...
         HAS_TIE = 2 : <0>...
IS_SMILING = 2 :
   HEAD_SHAPE = 1 :
      BODY_SHAPE = 1 : <1>...
      BODY_SHAPE = 2 :
         HAS_TIE = 1 :
            JACKET_COLOR = 1 : <0>
            JACKET_COLOR = 3 : <1>
            JACKET_COLOR = 4 : <1>
         HAS_TIE = 2 :
            JACKET_COLOR = 1 : <1>
            JACKET_COLOR = 3 : <0>
            JACKET_COLOR = 4 : <0>
      BODY_SHAPE = 3 : <1>...
   HEAD_SHAPE = 2 :
      JACKET_COLOR = 1 :
         HAS_TIE = 1 :
            BODY_SHAPE = 1 : <0>
            BODY_SHAPE = 2 : <1>
         HAS_TIE = 2 : <1>...
      JACKET_COLOR = 2 : <0>...
      JACKET_COLOR = 3 : <0>...
      JACKET_COLOR = 4 : <0>...
   HEAD_SHAPE = 3 :
      JACKET_COLOR = 1 : <1>...
      JACKET_COLOR = 2 :
         HAS_TIE = 1 :
            BODY_SHAPE = 1 : <1>
            BODY_SHAPE = 3 : <0>
         HAS_TIE = 2 : <0>...
      JACKET_COLOR = 4 : <0>...
```

5.3 Classification diagrams

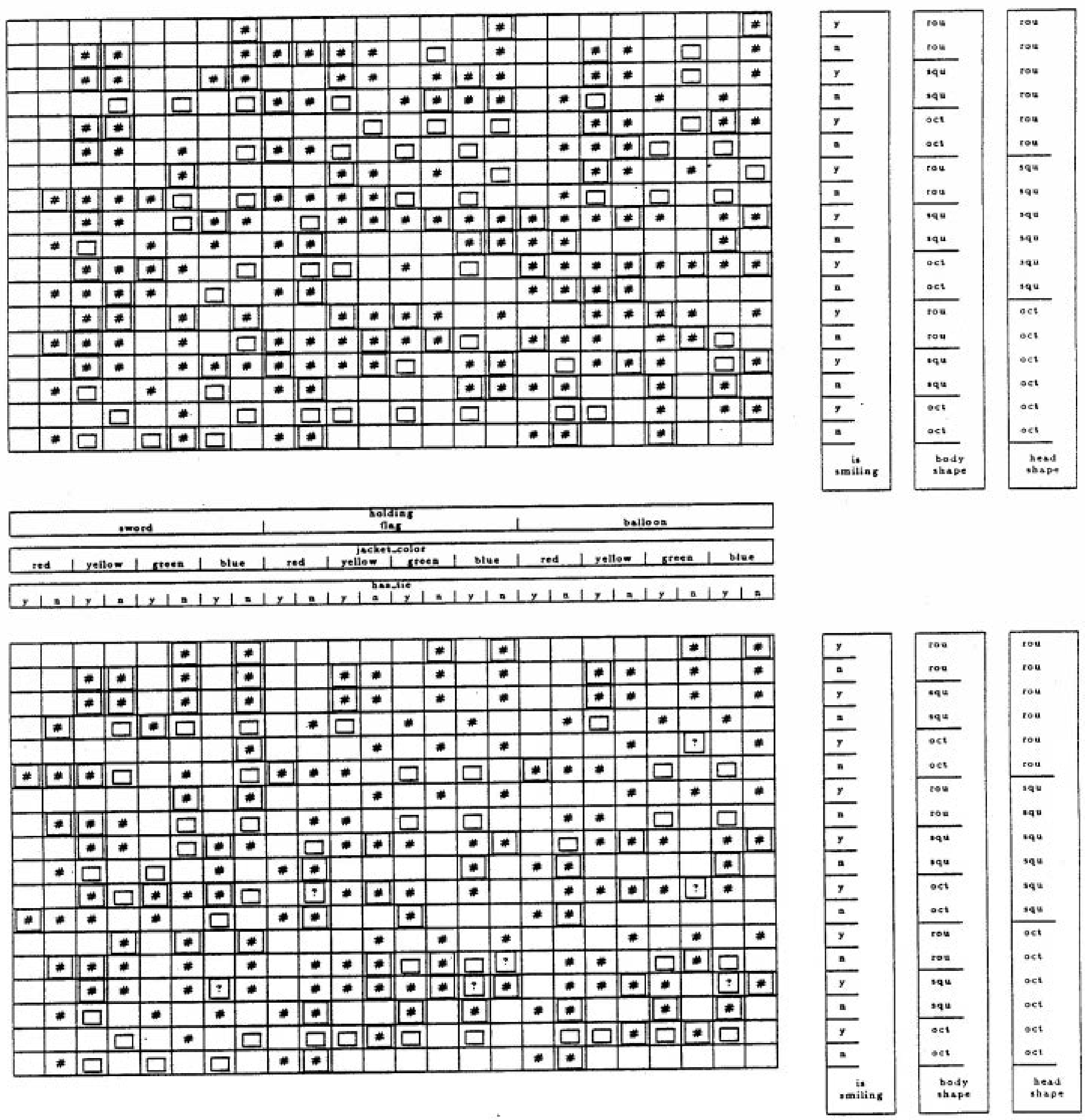
(a) Result of ID5R on test set 1, Accuracy: 83.1%
(b) Result of IDL on test set 1, Accuracy: 100.0%



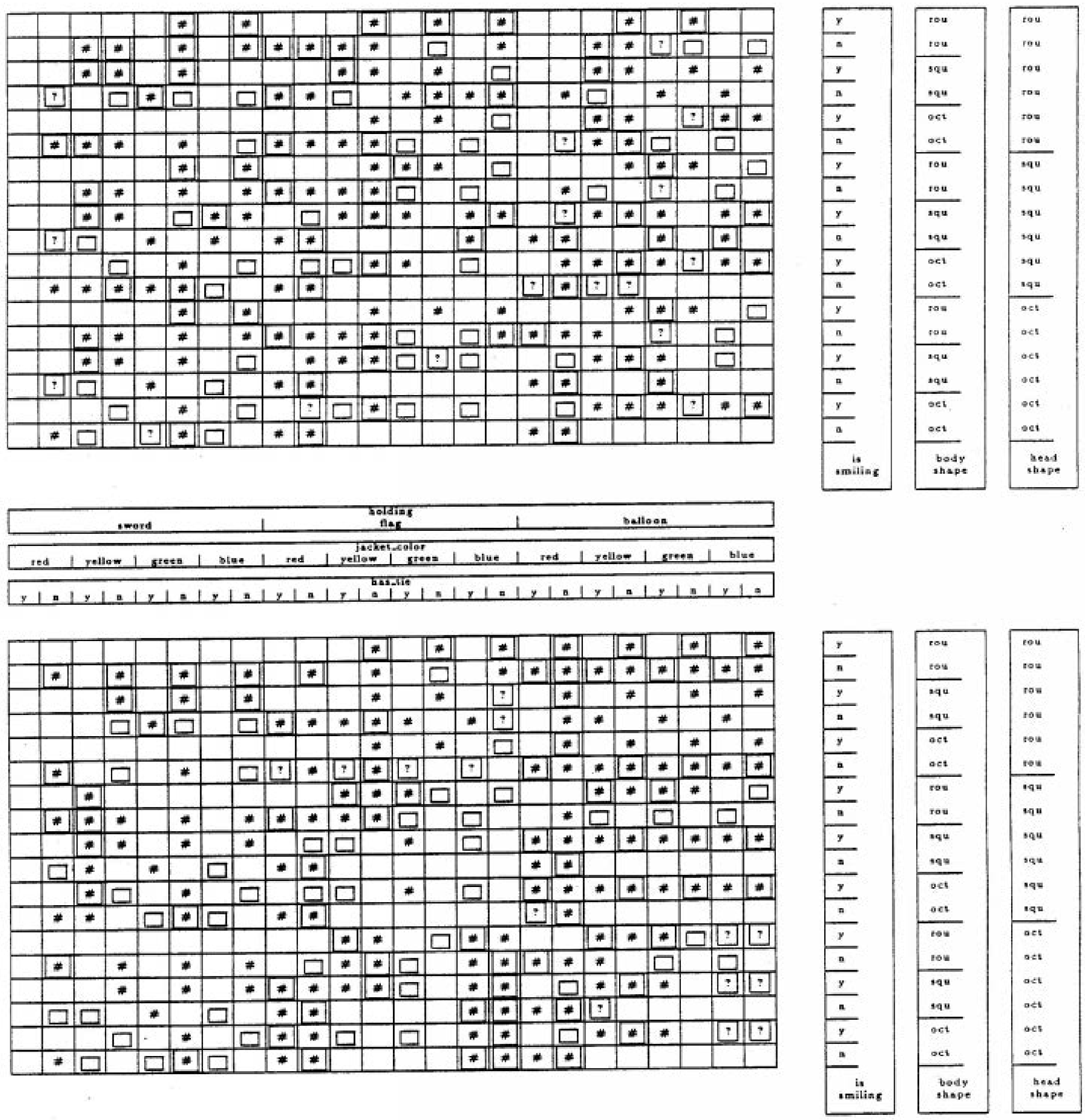
(a) Result of ID5R-HAT on test set 1, Accuracy: 90.3%
(b) Result of TDIDT-based method on test set 1, Accuracy: 75.7%



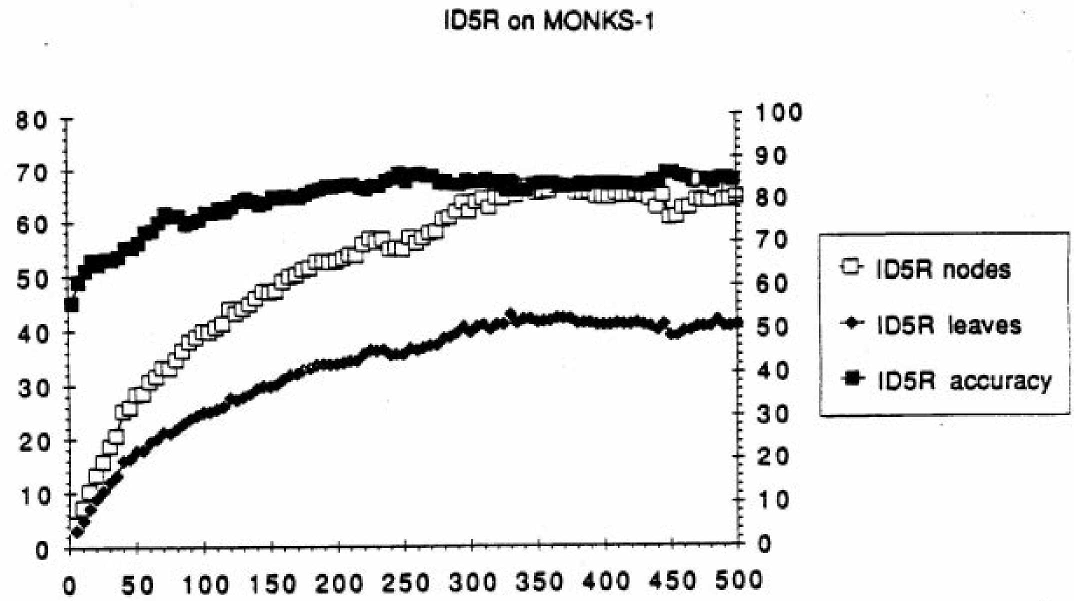
(a) Result of ID5R on test set 2, Accuracy: 66.2%
(b) Result of IDL on test set 2, Accuracy: 71.3%

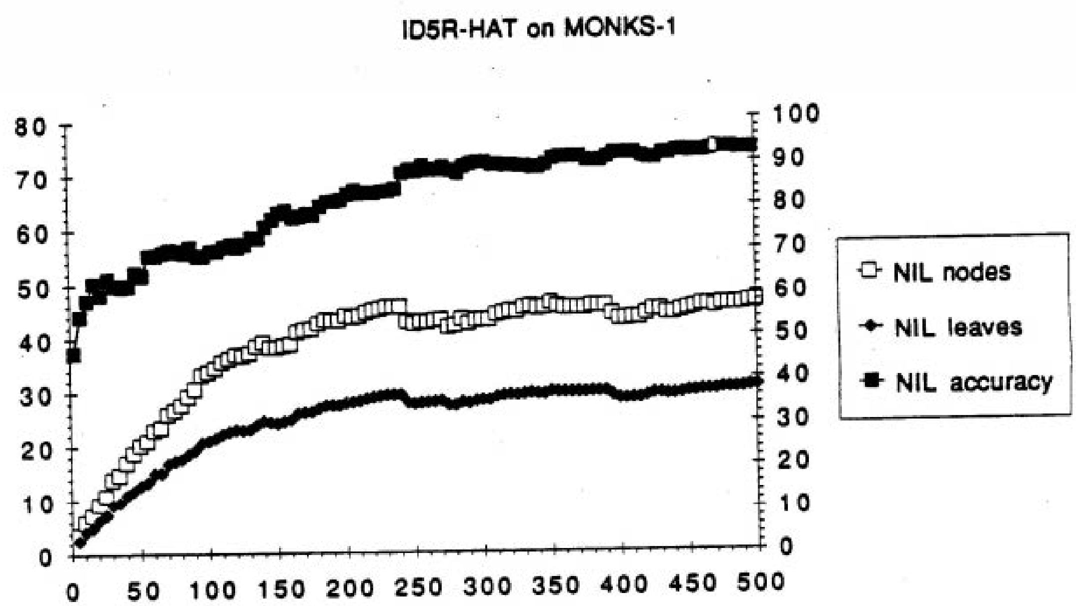


(a) Result of TDIDT on test set 2, Accuracy: 67.1%
(b) Result of ID5R-HAT on test set 2, Accuracy: 67.8%

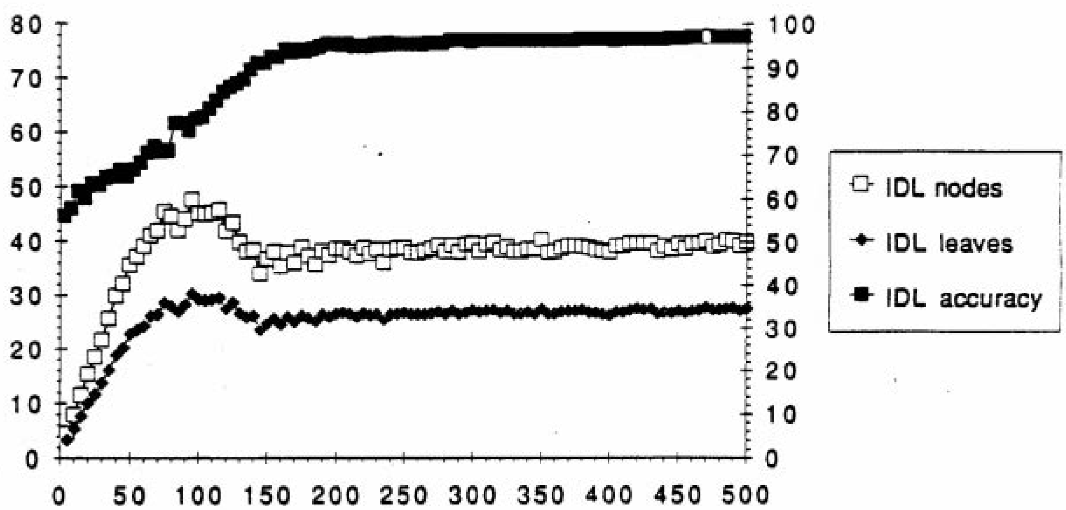


5.4 Learning curves

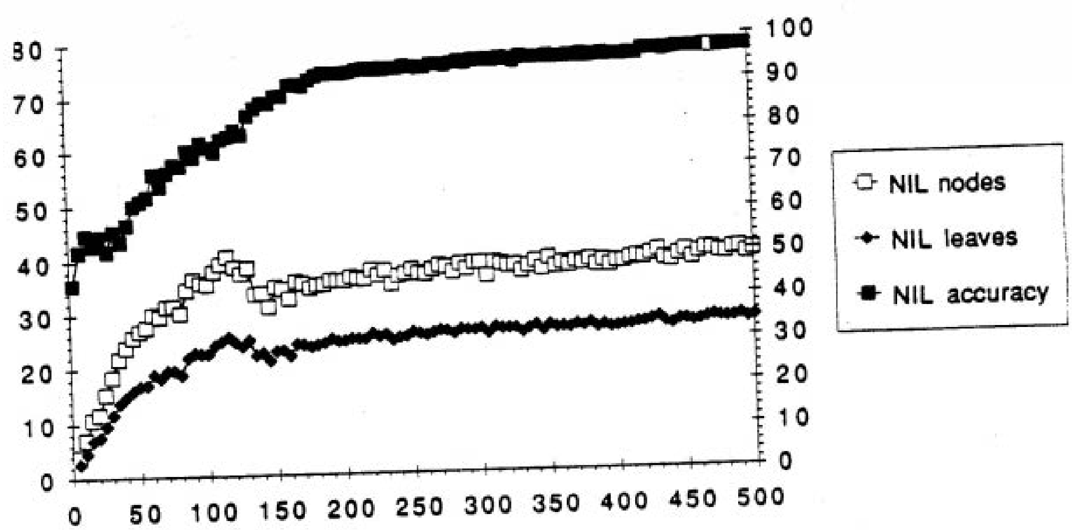




IDL on MONKS-1



IDL-HAT on MONKS-1



Chapter 6

Comparison of Inductive Learning Programs

J. Kreuziger
R. Hamann
W. Wenzel

Institute for Real-Time Computer Control Systems & Robotics, Prof. Dr.-Ing. U. Rembold and Prof. Dr.-Ing. R. Dillmann, University of Karlsruhe, Faculty for Informatics, Postfach 6980, 7500 Karlsruhe 1, Germany, E-Mail: kreuzig@ira.uka.de

6.1 Introduction

At the Institute for Real-Time Computer Control Systems & Robotics a library of inductive machine learning algorithms is being developed. So far this library consists of:

- ID3 - classical decision tree learning algorithm
- ID5R - an incremental decision tree learning algorithm
- AQR - a version of the AQ-rule learning algorithms
- CN2 - rule decision list learning algorithm
- COBWEB - conceptual clustering algorithm for attributes with symbolic values
- CLASSIT - conceptual clustering algorithm for attributes with numerical values
- CLASSWEB - algorithm that integrates COBWEB and CLASSIT. In the following only this algorithm is referred to.

These algorithms have been implemented in a very homogeneous way, i.e. they use the same description for objects that have to be learned, they are called in a similar way and they are all available under one common user interface.

The reason for building up this ML-library is, that our institute is interested in applying machine learning techniques to robotics applications. As a first step we wanted to gain experiences with the classical inductive learning methods in order to find out their capabilities and limitations.

All algorithms base on a common description of the objects to be learned, which consists of a set of attributes, each defined by a name, a domain, a 'noisy-flag' and some additional information for the conceptual clustering algorithm. In addition a symbol which is used for unknown attribute values can be identified. Each algorithm will then be called with a set of examples (classified for ID3, ID5R, AQR and CN2; unclassified for CLASSWEB). As ID5R and CLASSWEB are incremental methods, a former received classifier can also be given as input. Each algorithm results in a classifier which can be used for classifying further given objects. For a better understanding of the results a textual representation of the classifier can be printed on the screen. For decision tree learning algorithms and conceptual clustering also a graphical display is available. For the incremental methods it is also possible to display a trace during classifier generation. The implementation work has been done on a SUN Sparc Station 1+ in SUN Lucid Common Lisp using CLX and CLUE for only the graphical interface ([HW91]).

6.2 Short description of the algorithms

In this section a very short description of the algorithms will be given. For further details please see the corresponding literature. The representation of examples as attribute-value-pairs, where the set of attributes is given and fixed, is common to all algorithms.

6.2.1 ID3

ID3 is the most popular representative of TDIDT-algorithms (Top Down Induction of Decision Trees). It builds up a decision tree based on the classified training examples ([Qui86]). The internal nodes of a decision tree represent a test based on one specific attribute. For each possible attribute value there is one subtree, which is for itself a decision tree. The leaves of the tree represent class names. For classifying a new object with a built-up decision tree, the value of the attribute at the root of the tree will determine which subtree has to be considered recursively. The recursion will end, if a leaf of the tree is reached. In that case the class name given in that leaf represents the class in which the object has to be classified.

The idea for building up the decision tree is to iteratively find the attribute in the set of attributes of the objects which gives the 'best' partition of the set of training examples. 'Best' is defined in terms of the information gain given by a partition according to the specific attribute.

The basic algorithm has already been extended by Quinlan ([Qui86]) to handle noisy attributes and unknown attribute values. In the implemented algorithm noise is handled by applying chi-square test for stochastic independence to the noisy attribute with respect to the class distribution. Unknown attribute values have to be handled during building of the decision tree and during classification. For building up the decision tree unknown attribute values are taken into account in the calculation of the information gain.

The algorithm as being implemented also uses windowing over the training set, i.e. a subset of the training set is chosen at random and the decision tree is built up by using only these examples. After that all other examples of the training set are classified using this DT. If some of the examples are incorrectly classified, a selection of these will be added to the window and the procedure will start again. Due to the complexity of the given training sets, a lot of iterative steps had to be performed.

6.2.2 ID5R

The ID5R algorithm ([Utg89]) has been developed by P.E. Utgoff as a kind of TDIDT-algorithm which is able to work incrementally, but results finally, i.e. after all training examples, in the same decision tree as ID3. 'Incremental' means that the examples can be given one after another. A very easy solution for the problem of successively given examples would be to generate an ID3 decision tree from scratch with all examples given so far. In contrast to that approach, ID5R always uses the decision tree developed so far for integrating the new example. For that reason the data structure of a node in an ID5R tree has been enlarged by the information necessary to calculate the information gain function of the attributes.

If during insertion of the new example the situation arises that the current test attribute is not the one with the highest information gain, the tree has to be restructured. This is done by investigating all subtrees of the current node by using the new attribute as the test attribute. In a second step the test attribute in the current node is exchanged for the attribute in the subtrees.

In our implementation ID5R does not result in exactly the same tree as ID3, even if all examples are given. First this is caused by the fact that ID5R does not generate NULL-classes, because leaves are only splitted further, if it is really necessary. Second, if there are several attributes with the same information gain and one of these attributes is already used as test attribute, then a restructuring of the tree will not be done. It would be of course also possible to take the first attribute in the list as new test attribute and to restructure the tree accordingly.

6.2.3 AQR

The AQR algorithm is an implementation of the AQ-family, which has been founded by R. Michalski in 1969. AQR is a reconstruction of a straight-forward implementation of the basic AQ algorithm and has been described in [CN89]. The algorithm results in one decision rule for each class. The condition of each rule is called a cover and represents a disjunction of so-called complexes. Each complex for itself is a conjunction of selectors and each selector is a basic attribute test (has the attribute one of a set of values, etc.).

For classifying a new object, each rule is checked to see, whether the condition is completely satisfied, i.e. the example is covered by the rule. If exactly one rule is satisfied, the corresponding class is the classification result. If several rules are applicable, then the most common class of training examples covered by those rules is used as result. If none of the rules can be applied, the class that appeared most often in the training set is used as result.

The decision rules are sequentially built up for the different classes. Starting with an empty cover successively a *seed*, i.e. a positive example which is not covered so far is being selected and a *star* is being generated, which is a set of complexes that cover the seed but no negative examples. From these complexes the one which is the best one according to a user-defined criterion is being chosen and added to the cover as an extra disjunct. The positive examples that are covered by that additional complex are then deleted from the list of examples. In our implementation the best complex is the complex that maximizes the number of positive examples that are covered.

6.2.4 CN2

This algorithm has been developed by P. Clark and T. Niblett ([CN89]). It shall combine the advantages of the families of ID3- and AQ-algorithms. The classifier resulting from that algorithm is an ordered set of if-then-rules (decision list). This means that the representation is very similar to AQ, i.e. if 'complex' then predict 'class', but the rules have to be checked from top to bottom. If none of the rules applies to a new object, again the class that appeared most often in the training set will be taken.

The idea of Clark and Niblett was to enable AQ-like algorithms to handle noisy data by also taking complexes into account that do not fit the positive/negative border accurately. The method is based on the beam-search method as being used in AQ. During each iteration the algorithm searches for a complex that covers a large number of examples of one class and only few examples of other classes. The complexes are evaluated by an evaluation function which determines their predictiveness and reliability. If a good complex has been found, the examples that are covered, are deleted from the set of training examples. The search for a complex can be seen as a general-to-specific search with some pruning. During each iteration a set of the best complexes found so far is being remembered. These are specialized by adding a new conjunctive term or deleting a disjunctive part of one of the selectors. CN2 evaluates all possible specializations of each complex, which may lead to an enormous computational effort.

6.2.5 CLASSWEB

CLASSWEB is a combination of the algorithms COBWEB ([Fis87]) and CLASSIT ([GLF89]). These are methods for conceptual clustering. In contrast to the four algorithms described so far, these use unclassified examples as input and try to find a concept hierarchy for the examples where the similarity in one concept is as high as possible and the similarity between different concepts is as low as possible. While COBWEB only handles nominal values and CLASSIT only numerical ones, our CLASSWEB algorithm is able to handle both types in an integrated way.

For building up a concept hierarchy CLASSWEB uses four different operators to integrate a new example into the already existing concept hierarchy. These are: 1.) classifying the object into an existing class, 2.) creating a new class, 3.) combining two classes into a single class and 4.) dividing a class into several classes. Applied to internal concept nodes these different operators are scored according to category utility and the best one is chosen.

We have also implemented the so-called *cutoff* in CLASSWEB. By that parameter the algorithm does not have to classify each example down to a leaf, but also may decide to stop at some higher level in the hierarchy. Cutoff is a measure whether an example and a concept class are similar enough to stop at that concept node. If cutoff is set to zero, the algorithm behaves exactly like the original COBWEB method.

To compare CLASSWEB with those inductive learning algorithms which use classified examples as input, somehow the class information had to be added to the examples. This was done by handling the class of each example as an additional attribute. During classification the prediction capabilities of CLASSWEB are used, to determine a class for the unclassified example.

6.3 Results

The following tables compare the performance of the different algorithms on the three problem sets. The time data given correspond to compiled SUN Lucid Common Lisp 3.0 code on a SUN SPARC station 1+.

6.3.1 Training Time

This following table states the time required for each algorithm on each training set to build up a classifier.

Algorithm	Training Set 1	Training Set 2	Training Set 3
ID3	35.51	154.02	23.04
ID3 no wind.	4.98	7.61	3.74
ID5R	99.20	407.09	78.91
AQR	4.17	9.45	4.00
CN2	4.48	74.04	10.25
CLASSWEB 0.10	1406.47	2013.78	1311.25
CLASSWEB 0.15	867.47	977.04	882.09
CLASSWEB 0.20	499.94	646.06	521.21

Time is given in seconds and was averaged over three test runs over each algorithm and each training set.

Remarks:

The ID3-algorithm as implemented uses a 20%-windowing as mentioned above. For the three given problems this leads to a large number of necessary iterations. That's why there are also results given for ID3 without windowing (ID3 no wind.).

The CN2-algorithm uses a user-defined threshold value for doing its noise test. This is set to 0.1.

The cutoff-parameter in CLASSWEB was set to 0.10, 0.15 resp. 0.20 in three different experiments.

6.3.2 Classifier Results

First we will give some measurements such as number of nodes, leaves, rules and so on, which will reflect the complexity of the resulting algorithms. Afterwards some of the resulting classifiers for the different algorithms and training sets are given.

ID3

Measurement	Training Set 1	Training Set 2	Training Set 3
# nodes	13	66	13
# leaves	28	110	29

ID3 no windowing

Measurement	Training Set 1	Training Set 2	Training Set 3
# nodes	32	64	14
# leaves	62	110	31

ID5R

Measurement	Training Set 1	Training Set 2	Training Set 3
# nodes	34	64	14
# leaves	52	99	28

AQR

Measurement	Training Set 1		Training Set 2		Training Set 3	
	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
# complexes	30	6	40	43	16	20
# selectors	109	14	147	187	47	67

CN2

Measurement	Training Set 1	Training Set 2	Training Set 3
# rules	10	58	24
# selectors	13	145	38

CLASSWEB (cut-off = 0.10)

Measurement	Training Set 1	Training Set 2	Training Set 3
# concepts	219	305	217
# nodes	95	137	95
# leaves	124	168	122

CLASSWEB (cut-off = 0.15)

Measurement	Training Set 1	Training Set 2	Training Set 3
# concepts	57	58	68
# nodes	21	23	26
# leaves	36	35	42

CLASSWEB (cut-off = 0.20)

Measurement	Training Set 1	Training Set 2	Training Set 3
# concepts	21	26	29
# nodes	7	10	11
# leaves	14	16	18

Training Set 1

ID3

```
JACKET-COLOR
  1  1
  2
    BODY-SHAPE
       1
         HEAD-SHAPE
            1  1
            2  0
            3  0
       2
         HEAD-SHAPE
            1  0
            2  1
            3  0
       3
         HEAD-SHAPE
            1  0
            2  0
            3  1
  3
    HEAD-SHAPE
       1
         BODY-SHAPE
            1  1
            2  0
            3  0
       2
         BODY-SHAPE
            1  0
            2  1
            3  0
```

```
        3
            BODY-SHAPE
                1  0
                2  0
                3  1
    4
        HEAD-SHAPE
            1
                BODY-SHAPE
                    1  1
                    2  0
                    3  0
            2
                BODY-SHAPE
                    1  0
                    2  1
                    3  0
            3
                BODY-SHAPE
                    1  NULL
                    2  0
                    3  1
```

AQR

```
BODY-SHAPE = 2  &  JACKET-COLOR = 2  &  HOLDING = 1  &  HEAD-SHAPE = 1  |
HAS-TIE = 1  &  HOLDING = 1  &  BODY-SHAPE = 2  &  HEAD-SHAPE = 1  |
IS-SMILING = 1  &  HEAD-SHAPE = 1  &  JACKET-COLOR = 4  |
BODY-SHAPE = 2  &  HOLDING = 2  &  HEAD-SHAPE = 1  &  JACKET-COLOR = 3  |
JACKET-COLOR = 2  &  HEAD-SHAPE = 1  &  BODY-SHAPE = 2  |
JACKET-COLOR = 4  &  HEAD-SHAPE = 1  &  BODY-SHAPE = 2  |
HOLDING = 3  &  BODY-SHAPE = 2  &  HEAD-SHAPE = 1  |
HOLDING = 1  &  BODY-SHAPE = 3  &  HEAD-SHAPE = 1  |
BODY-SHAPE = 3  &  JACKET-COLOR = 2  &  HEAD-SHAPE = 1  |
HOLDING = 3  &  BODY-SHAPE = 3  &  JACKET-COLOR = 3  &  IS-SMILING = 1
                                                     &  HEAD-SHAPE = 1  |
HAS-TIE = 2  &  IS-SMILING = 2  &  HEAD-SHAPE = 1  &  JACKET-COLOR = 3  |
JACKET-COLOR = 4  &  HEAD-SHAPE = 1  &  BODY-SHAPE = 3  |
HOLDING = 1  &  HEAD-SHAPE = 2  &  BODY-SHAPE = 1  |
HOLDING = 2  &  IS-SMILING = 1  &  BODY-SHAPE = 1  &  JACKET-COLOR = 2  |
HEAD-SHAPE = 2  &  HOLDING = 2  &  BODY-SHAPE = 1  &  JACKET-COLOR = 3  |
IS-SMILING = 1  &  BODY-SHAPE = 1  &  HEAD-SHAPE = 2  &  JACKET-COLOR = 4  |
HAS-TIE = 2  &  IS-SMILING = 2  &  HOLDING = 2  &  BODY-SHAPE = 1  |
HOLDING = 3  &  BODY-SHAPE = 1  &  HEAD-SHAPE = 2  |
HOLDING = 2  &  BODY-SHAPE = 3  &  HEAD-SHAPE = 2  &  JACKET-COLOR = 3  |
BODY-SHAPE = 3  &  HOLDING = 3  &  JACKET-COLOR = 3  &  HEAD-SHAPE = 2  |
BODY-SHAPE = 3  &  JACKET-COLOR = 4  &  HEAD-SHAPE = 2  |
IS-SMILING = 2  &  HOLDING = 1  &  BODY-SHAPE = 3  &  HEAD-SHAPE = 2  |
IS-SMILING = 2  &  HEAD-SHAPE = 2  &  BODY-SHAPE = 3  &  JACKET-COLOR = 2  |
HAS-TIE = 2  &  HEAD-SHAPE = 3  &  HOLDING = 3  &  BODY-SHAPE = 1  |
JACKET-COLOR = 2  &  BODY-SHAPE = 1  &  HEAD-SHAPE = 3  |
HEAD-SHAPE = 3  &  JACKET-COLOR = 4  &  HOLDING = 1  &  IS-SMILING = 1
                                                     &  BODY-SHAPE = 2  |
HOLDING = 2  &  JACKET-COLOR = 4  &  BODY-SHAPE = 2  |
HEAD-SHAPE = 3  &  JACKET-COLOR = 3  &  BODY-SHAPE = 2  |
HOLDING = 3  &  IS-SMILING = 2  &  BODY-SHAPE = 2  &  JACKET-COLOR = 2  |
HOLDING = 3  &  BODY-SHAPE = 2  &  HEAD-SHAPE = 3  &  JACKET-COLOR = 4
===> CLASS '0' (P['0'] = 1/2)


BODY-SHAPE = 1  &  HEAD-SHAPE = 1  |
JACKET-COLOR = 1  |
IS-SMILING = 1  &  BODY-SHAPE = 2  &  HEAD-SHAPE = 2  |
```

```
BODY-SHAPE = 2  &  HEAD-SHAPE = 2  |
HAS-TIE = 1  &  BODY-SHAPE = 3  &  HEAD-SHAPE = 3  |
HAS-TIE = 2  &  HEAD-SHAPE = 3  &  BODY-SHAPE = 3
===> CLASS '1' (P['1'] = 1/2)

DEFAULT ===> CLASS '0' (P['0'] = 1/2)
```

CN2

```
JACKET-COLOR = 1    ===> CLASS '1'
HEAD-SHAPE = 2  &  BODY-SHAPE = 3    ===> CLASS '0'
BODY-SHAPE = 1  &  HEAD-SHAPE = 3    ===> CLASS '0'
BODY-SHAPE = 1  &  HEAD-SHAPE = 2    ===> CLASS '0'
BODY-SHAPE = 1    ===> CLASS '1'
HEAD-SHAPE = 2    ===> CLASS '1'
BODY-SHAPE = 2    ===> CLASS '0'
HEAD-SHAPE = 3    ===> CLASS '1'
HAS-TIE = 2    ===> CLASS '0'
HAS-TIE = 1    ===> CLASS '0'

DEFAULT  ===>  CLASS '0'
```

Accuracy

Algorithm	Training Set 1	Training Set 2	Training Set 3
ID3	100.00	100.00	100.00
ID3 no w.	100.00	100.00	100.00
ID5R	100.00	100.00	100.00
AQR	100.00	100.00	100.00
CN2	100.00	92.90	100.00
CLASSWEB 0.10	87.10	69.23	86.89
CLASSWEB 0.15	74.19	69.23	86.07
CLASSWEB 0.20	66.94	59.76	79.51

Algorithm	Test Set 1	Test Set 2	Test Set 3
ID3	98.56	67.92	94.44
ID3 no w.	83.24	69.12	95.60
ID5R	79.77	69.23	95.28
AQR	95.88	79.63	87.04
CN2	100.00	68.98	89.12
CLASSWEB 0.10	71.76	64.81	80.79
CLASSWEB 0.15	65.74	61.57	85.42
CLASSWEB 0.20	62.96	57.18	75.23

6.4 Conclusion

The results of this chapter give a good survey about the possibilities and limitations of the different tested inductive learning algorithms. Especially it is possible to compare the learning results not only with respect to accuracy, but also with respect to training time and classifier complexity. Since we mainly used the algorithms in the form as they were described in journal articles, they do not necessarily represent the actual version available to the authors of the original algorithms. Nevertheless the comparison clearly points out, which algorithms are more useful for domains similar to the Monk's problems.

Another interesting result is the strong impact of parameters on the learning result. Windowing in ID3 influences classifier complexity, accuracy and training time. In CLASSWEB they are determined very strongly by the cut-off parameter, which varies only between 0.1 and 0.2 in our experiments, but results in a factor of 3 in training time and a factor of 10 in classifier complexity.

It also has to be mentioned that some important capabilities of the algorithms have not been tested and compared by using the given learning problems. These are for example the handling of noise in specific attributes, of costs for determining attribute values and of unknown attribute values in ID3 and ID5R. The incremental nature of ID5R was not really needed in these test cases because all examples were given in advance. The ability to handle unknown attribute values in AQR and CN2 was not used either.

Acknowledgement

This research work was performed at the Institute for Real-Time Computer Control Systems & Robotics, Prof. Dr.-Ing. U. Rembold and Prof. Dr.-Ing. R. Dillmann, Faculty for Informatics, University of Karlsruhe, 7500 Karlsruhe 1, Germany. The work is funded by the "Sonderforschungsbereich Künstliche Intelligenz" of the Deutsche Forschungsgemeinschaft.

Bibliography

[CN89] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[Fis87] D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[GLF89] J.H. Gennari, P. Langley, and D.H. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

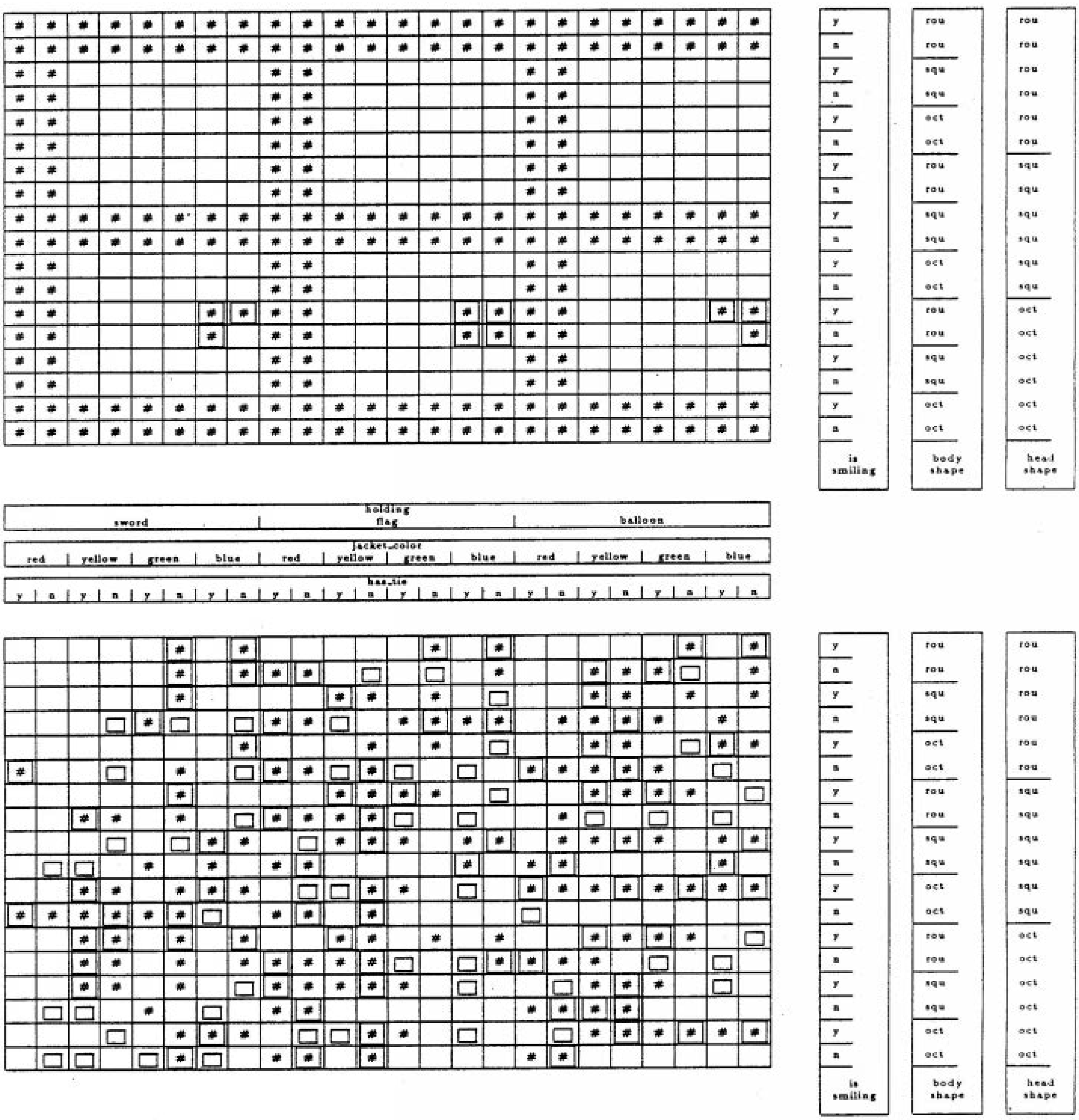
[HW91] R. Hamann and W. Wenzel. Implementation of inductive learning algorithms. Studienarbeit, Institute for Real-Time Computer Control Systems & Robotics, University of Karlsruhe, 1991. (In German).

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

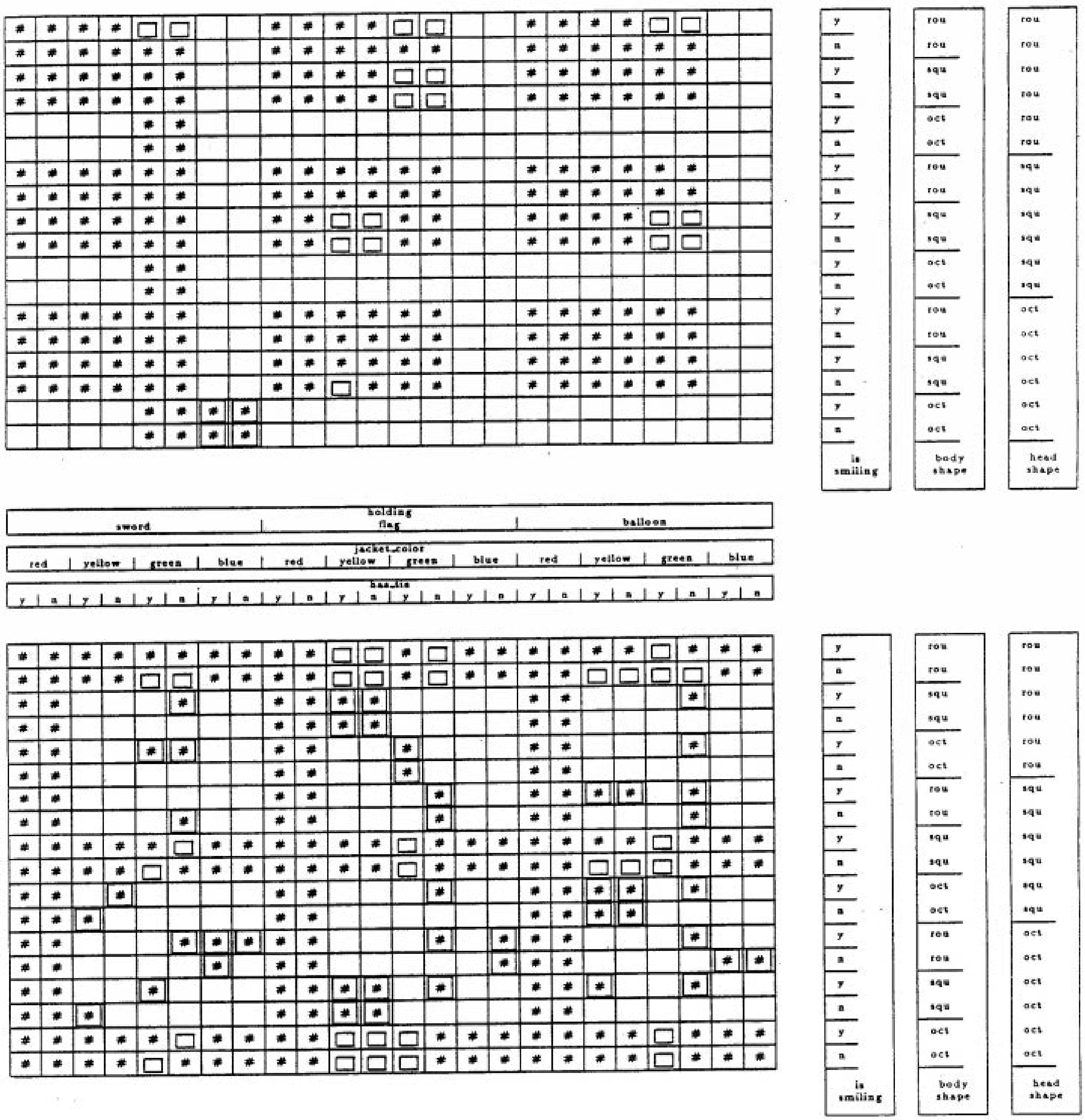
[Utg89] P.E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

6.5 Classification diagrams

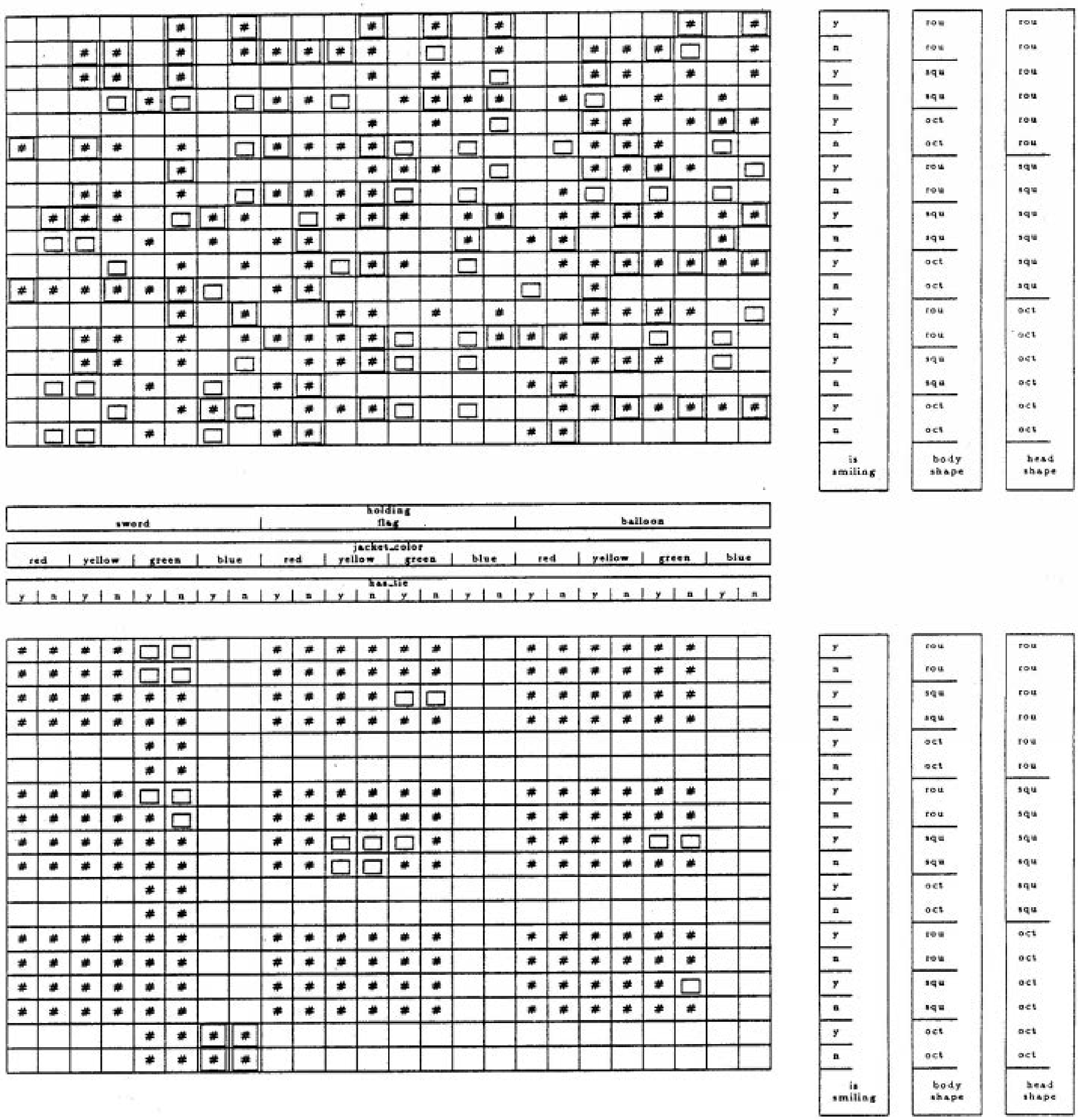
(a) Result of ID3 on test set No. 1, Accuracy: 97.7%
(b) Result of ID3 on test set No. 2, Accuracy: 67.4%



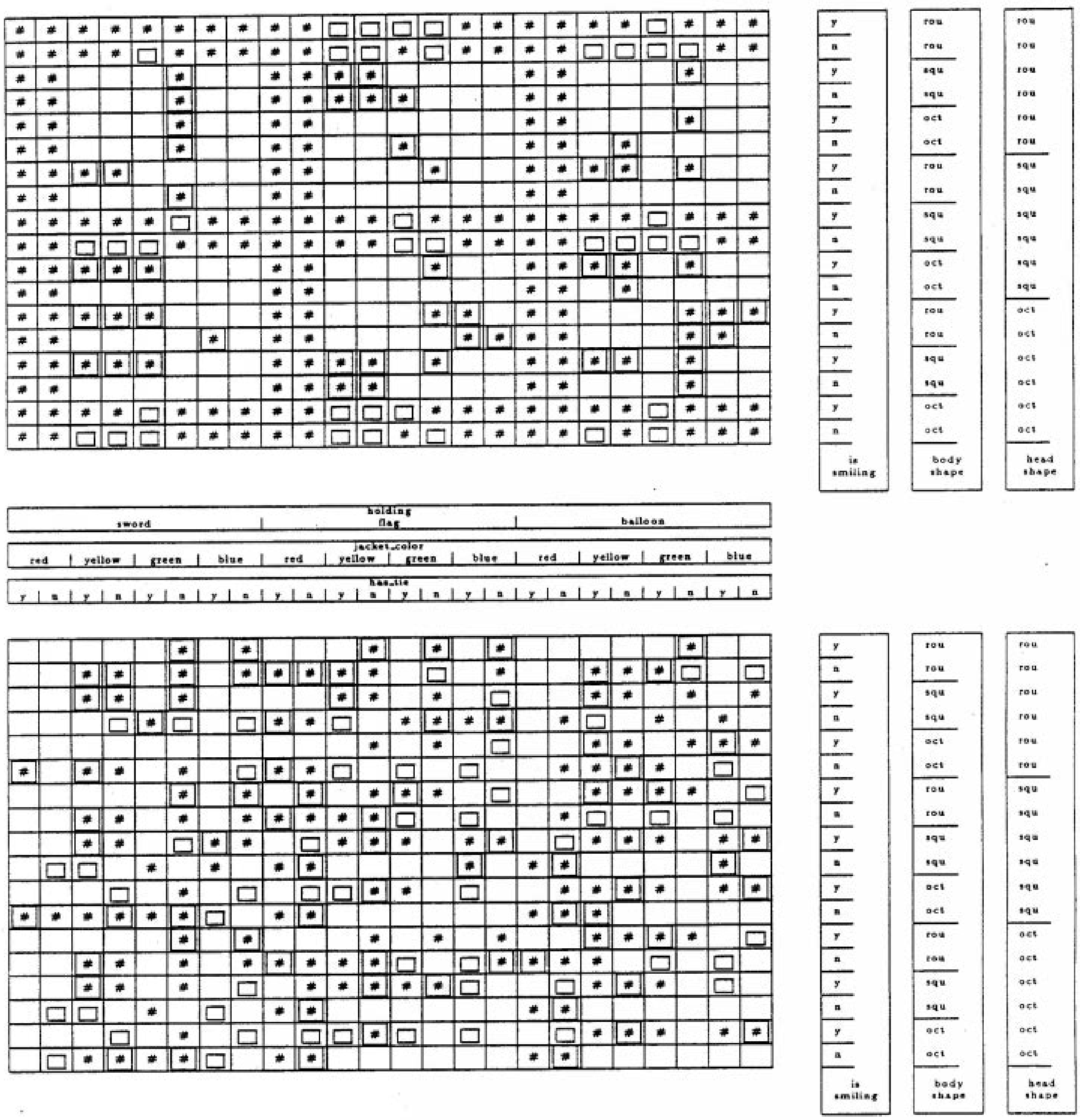
(a) Result of ID3 on test set No. 3, Accuracy: 94.7%
(b) Result of ID3OW on test set No. 1, Accuracy: 82.4%



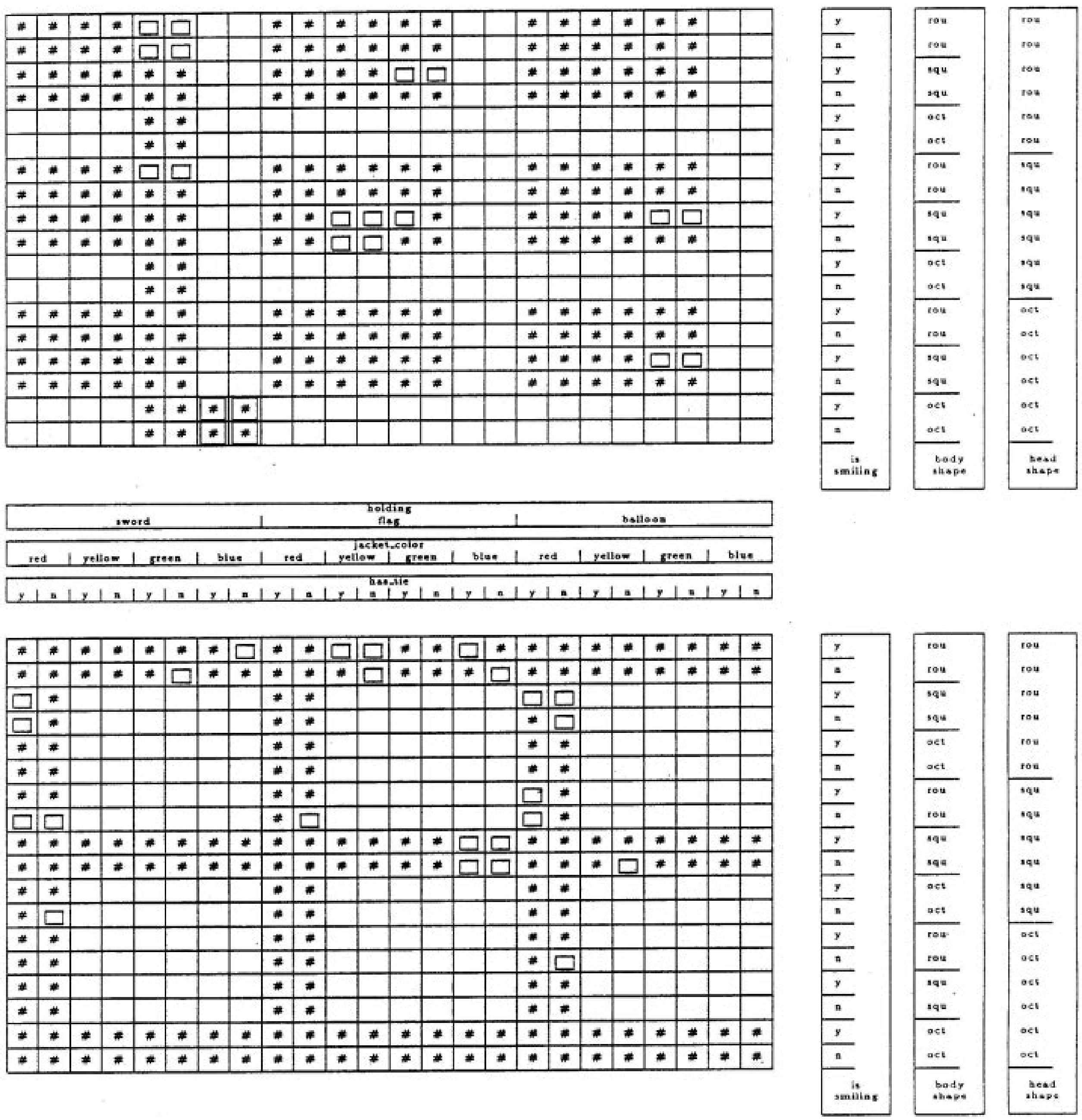
(a) Result of ID3OW on test set No. 2, Accuracy: 69.9%
(b) Result of ID3OW on test set No. 3, Accuracy: 95.1%



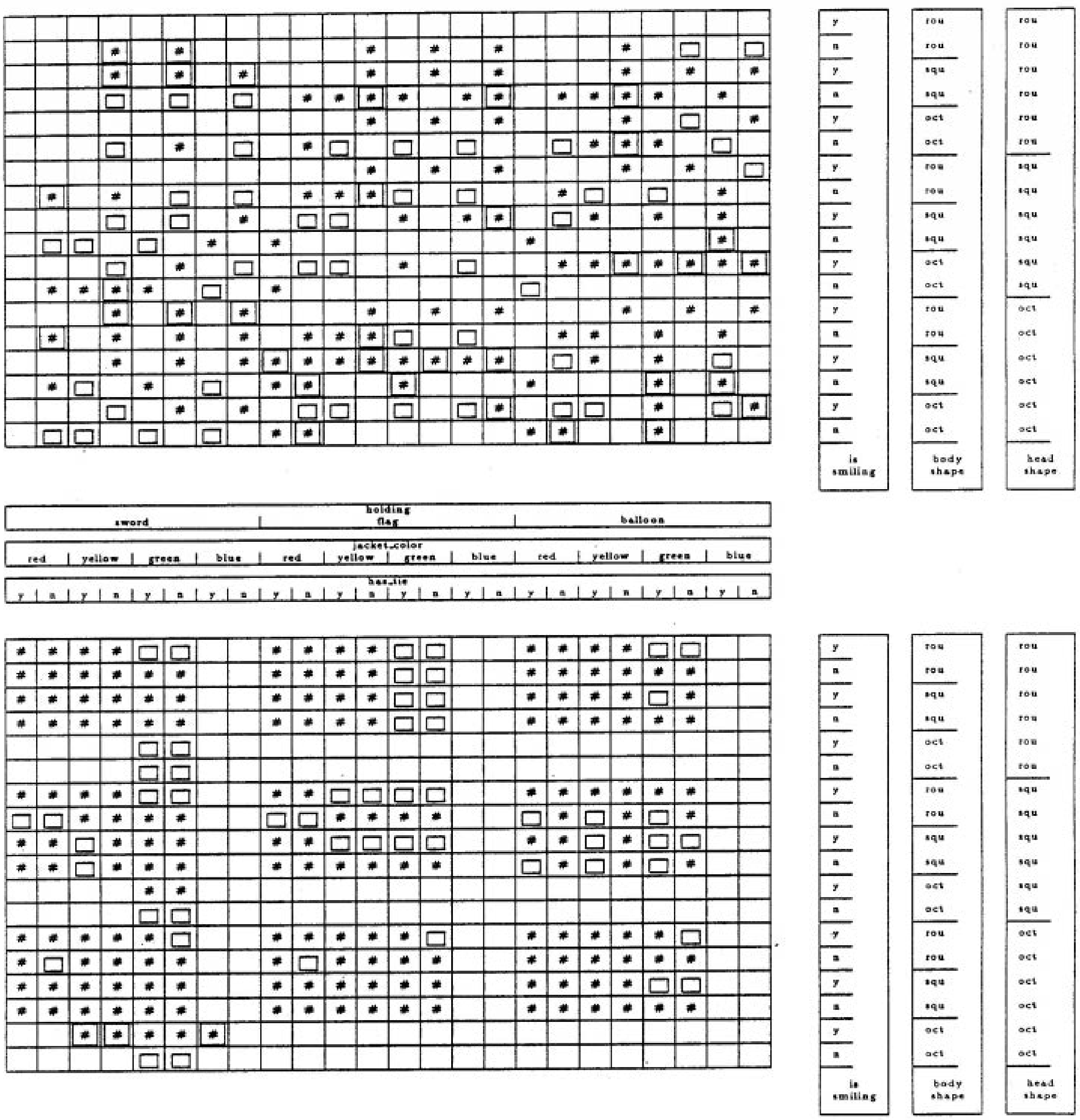
(a) Result of ID5R on test set No. 1, Accuracy: 78.9%
(b) Result of ID5R on test set No. 2, Accuracy: 69.0%



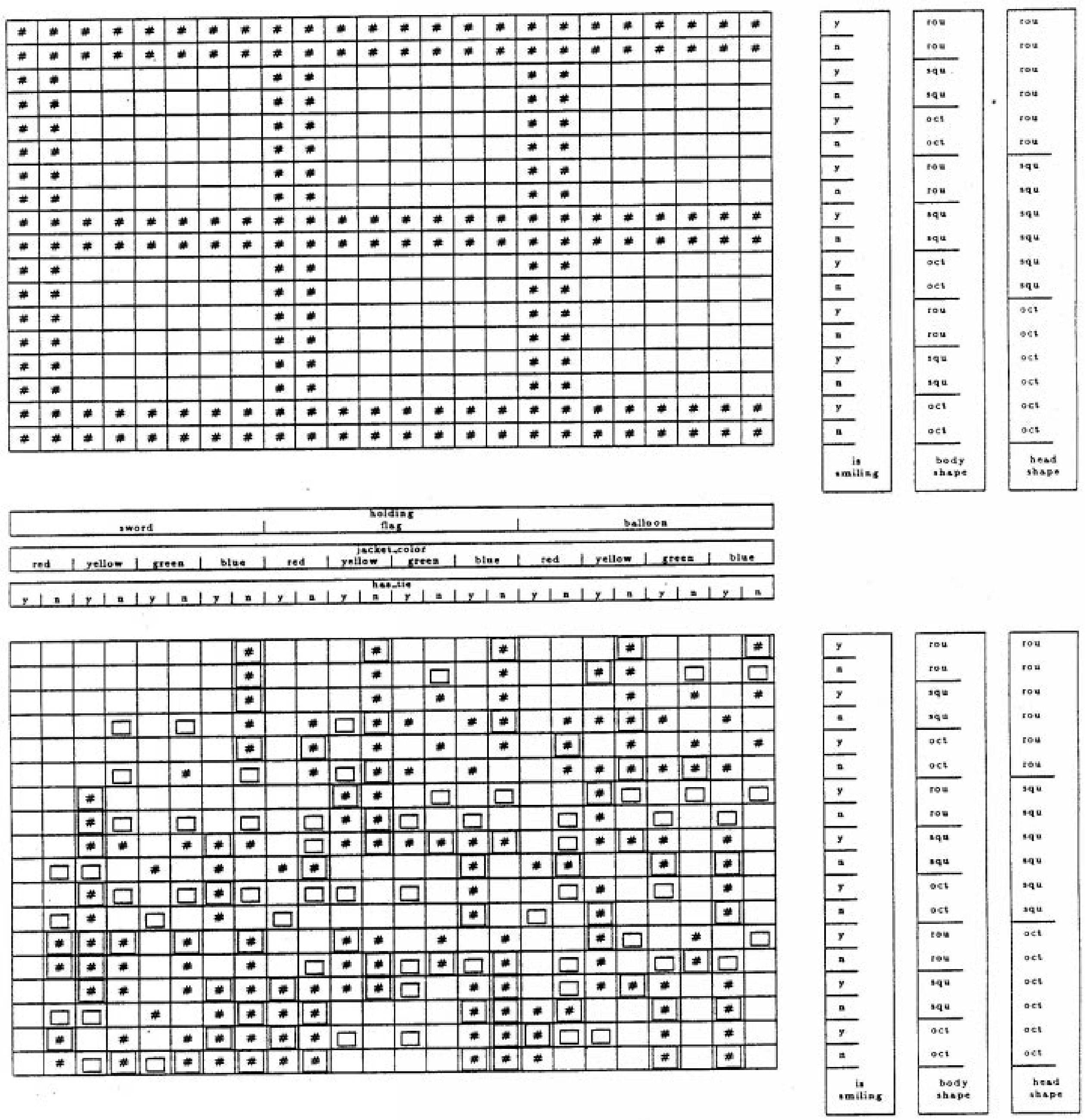
(a) Result of ID5R on test set No. 3, Accuracy: 95.1%
(b) Result of AQR on test set No. 1, Accuracy: 94.4%



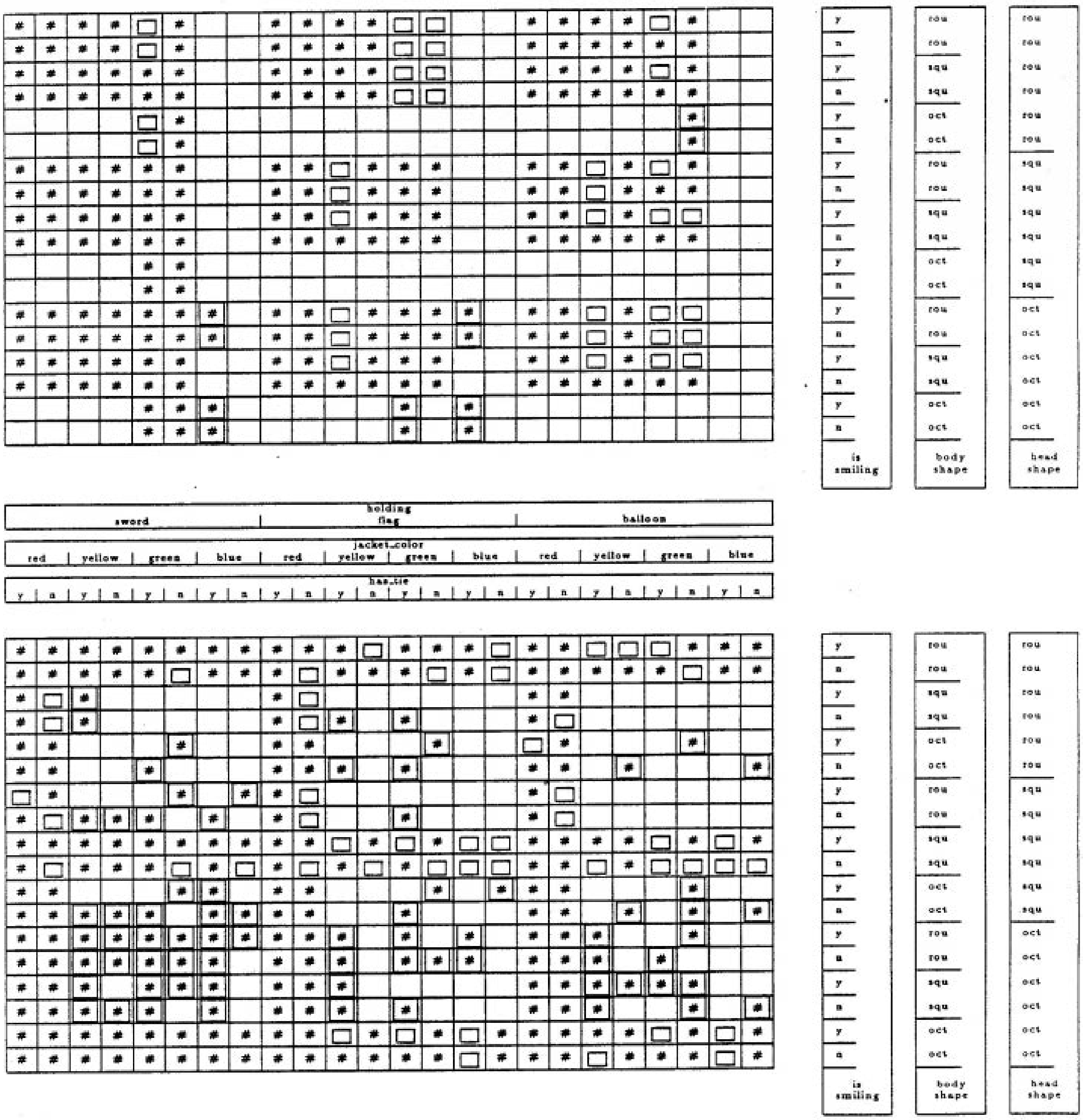
(a) Result of AQR on test set No. 2, Accuracy: 79.6%
(b) Result of AQR on test set No. 3, Accuracy: 87.0%



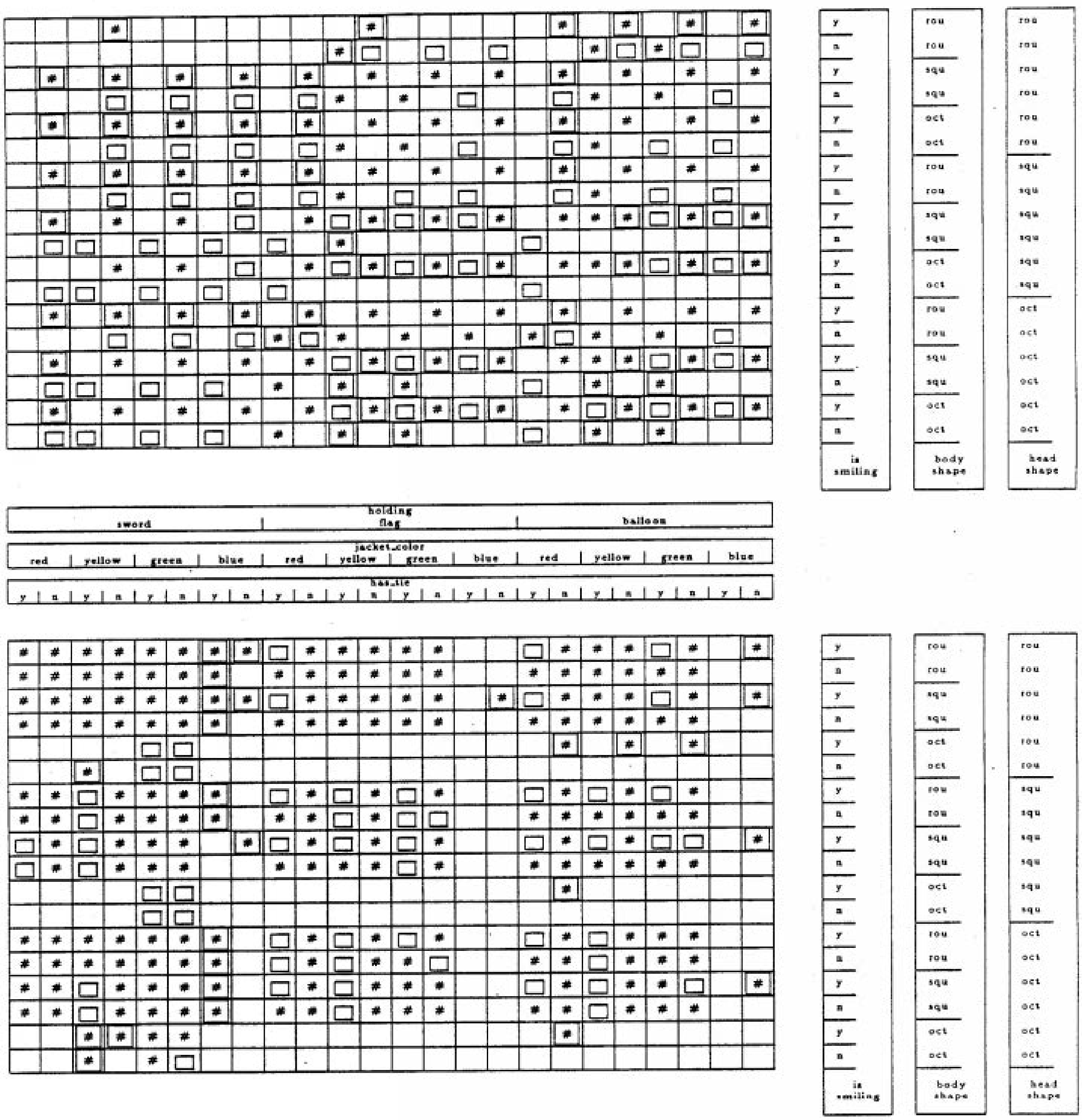
(a) Result of CN2 on test set No. 1, Accuracy: 100.0%
(b) Result of CN2 on test set No. 2, Accuracy: 69.0%



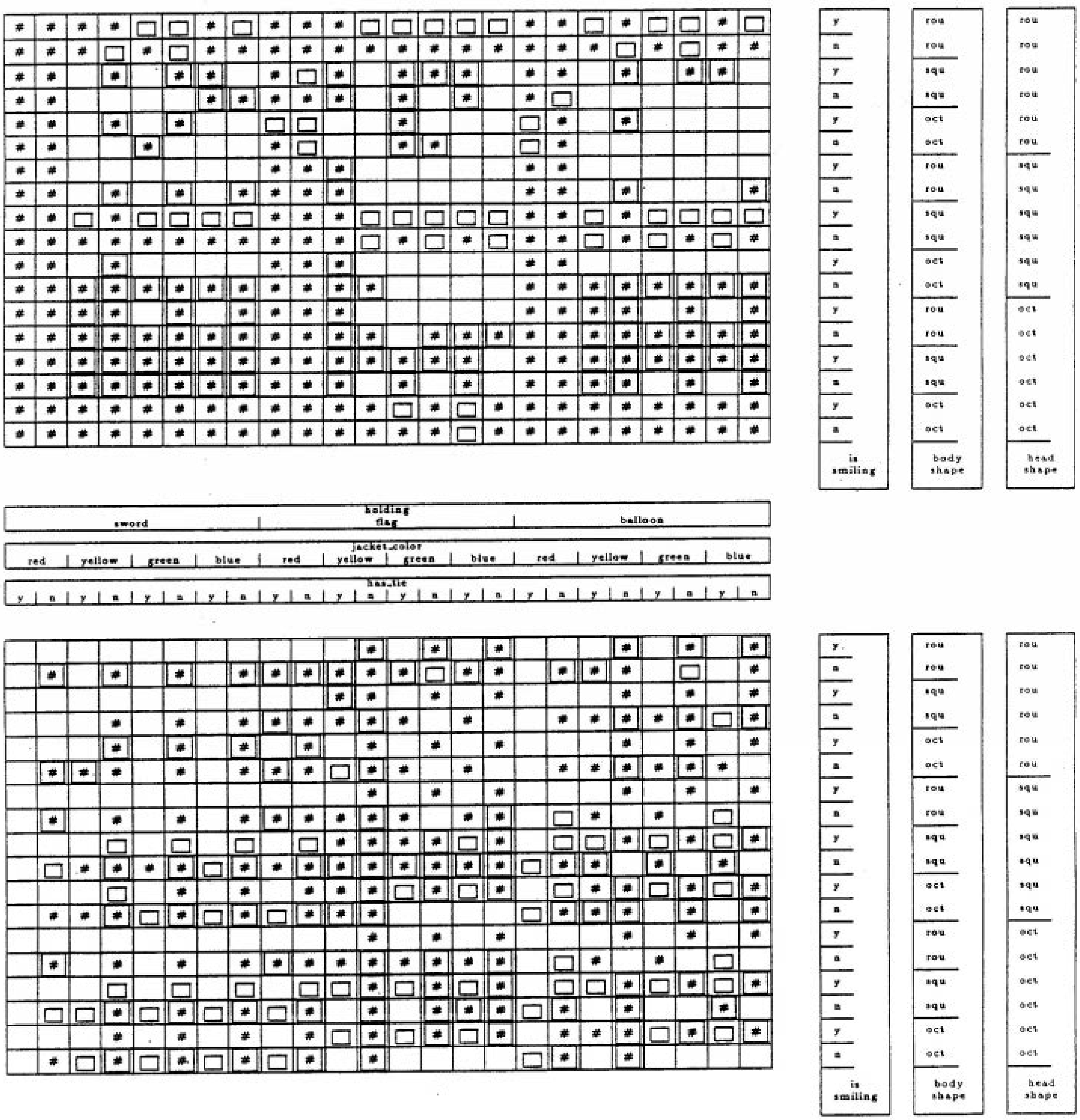
(a) Result of CN2 on test set No. 3, Accuracy: 89.1%
(b) Result of CLASSWEB 0.10 on test set No. 1, Accuracy: 71.8%



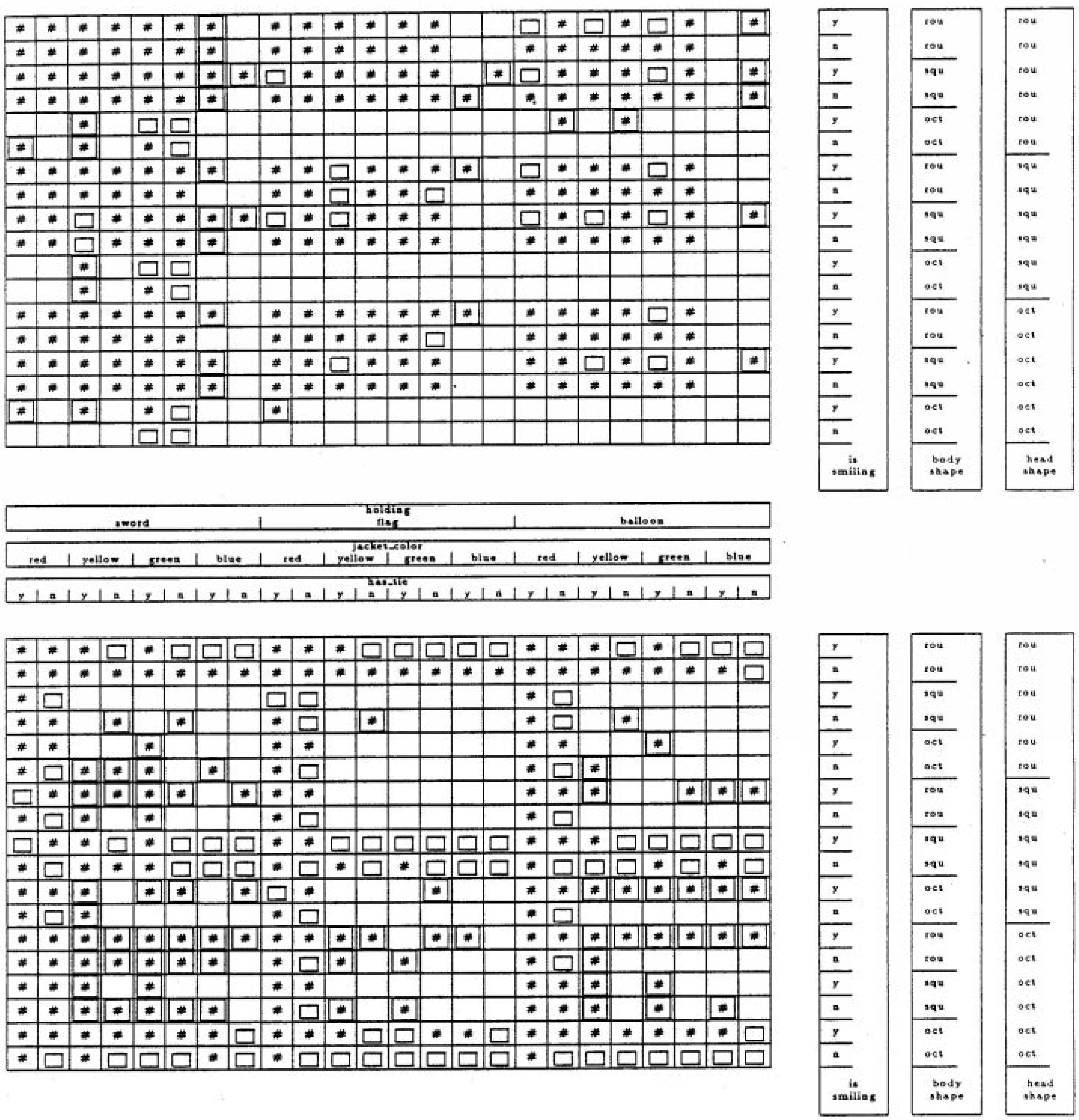
(a) Result of CLASSWEB 0.10 on test set No. 2, Accuracy: 64.8%
(b) Result of CLASSWEB 0.10 on test set No. 3, Accuracy: 80.8%



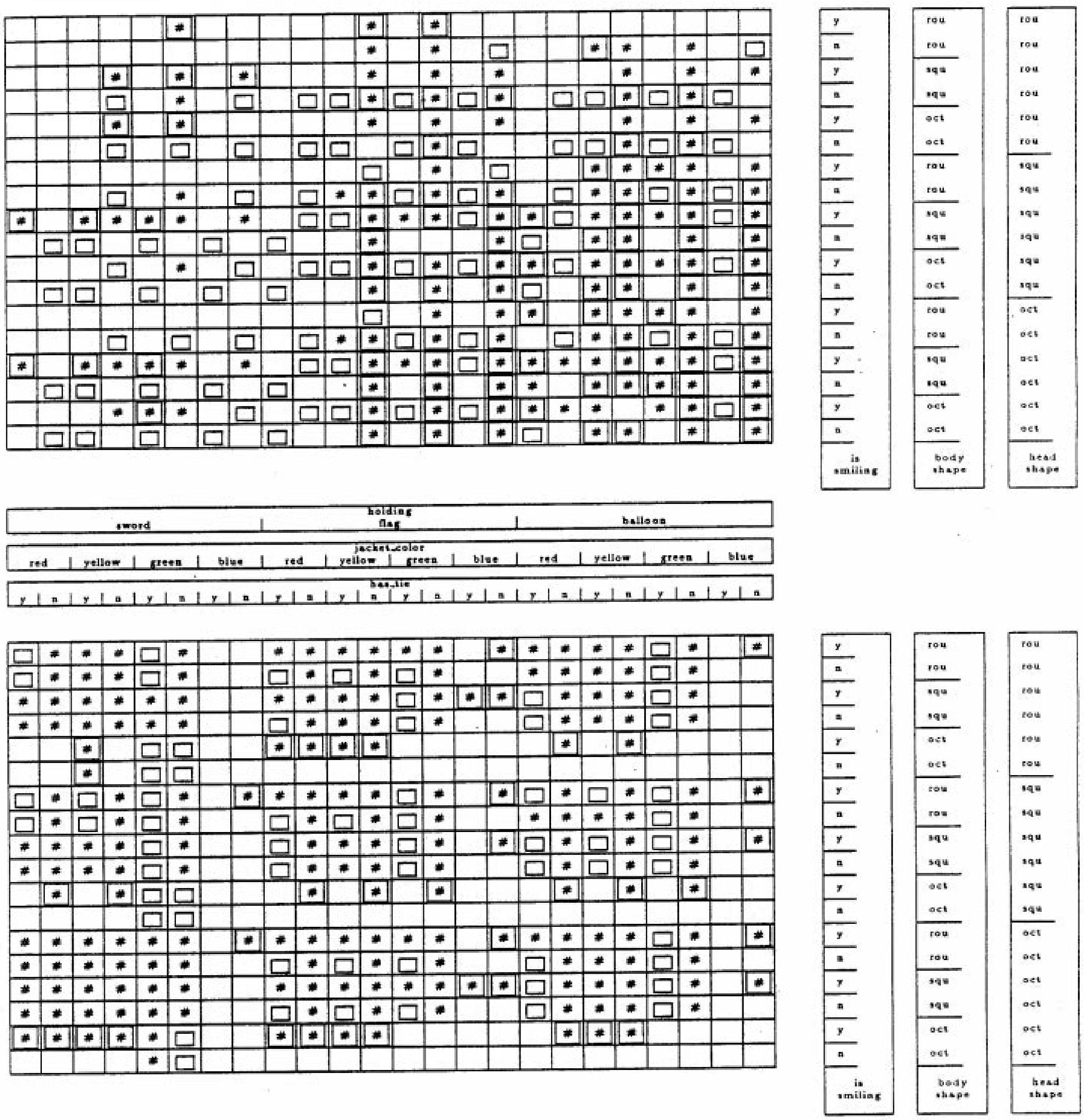
(a) Result of CLASSWEB 0.15 on test set No. 1, Accuracy: 65.7%
(b) Result of CLASSWEB 0.15 on test set No. 2, Accuracy: 61.6%



(a) Result of CLASSWEB 0.15 on test set No. 3, Accuracy: 85.4%
(b) Result of CLASSWEB 0.20 on test set No. 1, Accuracy: 63.0%



(a) Result of CLASSWEB 0.20 on test set No. 2, Accuracy: 57.2%
(b) Result of CLASSWEB 0.20 on test set No. 3, Accuracy: 75.2%



Chapter 7

Documentation of Prism – an Inductive Learning Algorithm

Stefan F. Keller

AI-Lab, Institute for Informatics, University of Zuerich, CH-8057 Zuerich

7.1 Short Description

PRISM was invented by Jadzia Cendrowska (1987). Based on Quinlan's induction algorithm ID3, PRISM pays attention to maximizing he information gain for a single value of an atribute in contrast to ID3 which tries to minimize the average entropy for an attribute-value pair.

7.2 Introduction

The decision tree output of ID3 algorithm is one of its major weaknesses. Not only can it be incomprehensible and difficult to manipulate, but its use in knowledge based systems frequently demands irrelevant information to be supplied. We argue that the problem lies in the induction algorithm itself and can only be remedied by radically altering the underlying strategy. The resulting algorithm, although based on ID3, uses a different induction strategy to induce rules which are modular in the sense how they are constructed. This approach avoids many of the problems associated with decision trees.

7.3 PRISM: Entropy versus Information Gain

The main cause of the problem described above is either that an attribute is highly relevant to only one classification and irrelevant to the others, or that only one value of the attribute is relevant.

There can be shown that while in the construction process of a decision tree although e.g. the entropy of a distinct branch d1 has been reduced to 0, the entropy of the other branch has actually increased to some higher entropy-measure. Attribute d would be chosen by ID3 because it minimizes the average entropy of the training set, or alternatively, it maximizes the average amount of information contributed by an attribute to the determination of any classifiaction.

In order to eliminate the use of irrelevant values of attributes and attributes which are irrelevant to a classification, an improving algorithm needs to maximize the actual amount of information contributed by knowing the VALUE of the attribute to the determination of a specific classification.

7.3.1 Maximizing the information gain

So, the task of an induction algorithm must be to find the attribute-value pair, ax, which contributes the most information about a specified classification, dn, i.e. for which I(dn l ax) is maximum.

This can be done in the following way: Let S be the data set; first find the ax for which p(dn — ax) is maximum. Lets call the choosen attribute c2 (=attribute c, value 2). Repeat now the process on a subset of S which contains only those instances which have value 2 for attribute c until there are all instances removed.

7.3.2 Trimming the tree

The remaining "branches" are not yet labelled, so the next step in the induction process is to identify the best rule of the set of instances which are not examples of the first rule. This is done by removing from S all instances containing this rule and applying the algorithm to the remaining instances. If this is repeated until there are no instances of class d1 left in S, the result is not a decision tree but a collection of branches. The whole process can then be repeated for each classification in turn, starting with the complete training set, S, each time.

The final output is an unordered collection of modular rules, each rule being as general as possible, thus ensuring that there are no redundant terms.

The following assumptions have been made about the training set:

- the classifications are mutually exclusive
- there is no noise, i.e. each instance is complete and correct
- each instance can be classified uniquely
- no instance is duplicated
- the values of the attributes are discrete
- the training set is complete, i.e. all possible combinations of attribute-value pairs are represented

Given that the assumptions above hold, the algorithm produces a complete set of correct rules.

7.4 The Basic Algorithm

If the training set contains instances of more than one classification, then for each classification, dn, in turn:

Step 1:
calculate the probability of occurence, p(dn — ax), of the classification dn for each attribute-value pair ax,

Step 2:
select the ax for which p(dn — ax) is a maximum and create a subset of the training set comprising all the instances which contain the selected ax,

Step 3:
repeat Steps 1 and 2 for this subset until is contains only instances of class dn. The induced rule is a conjunction of all the attribute-value pairs used in creating the homogeneous subset.

Step 4:
remove all instances covered by this rule from the training set.

Step 5:
repeat Steps 1-4 until all instances of class dn, have been removed.

When the rules for one classification have been induced, the training set is restored to its initial state and the algorithm is applied again to induce a set of rules covering the next classification. As the classifications are considered separately, their order of presentation is immaterial. If all instances are of the same classification then that classification is returned as the rule, and the algorithm terminates.

7.5 The Use of Heuristics

Opting for generality I: If there are two or more rules describing a classification, PRISM tries to induce the most general rule first. Thus PRISM selects that attribute-value pair which has the highest frequency of occurence in the set of instances being considered.

Opting for generality II: When both the information gain offered by two or more attribute-value pairs is the same and the numbers of instances referencing them is the same, PRISM selects the first.

7.6 General Considerations and a Comparison with ID3

A rule will not be induced by PRISM if there are no examples of it in the training set, but this applies to all induction programs. Even human beings cannot be expected to induce rules from non-existent information.

The accuracy of rules induced from an incomplete training set depends on the size of that training set (as with all induction algorithms) but is comparable to the accuracy of a decision tree induced by ID3 from the same training set, despite the gross reduction in number and length of the rules.

The major difference between ID3 and PRISM is that PRISM concentrates on finding only relevant values of attributes, while ID3 is concerned with finding the attribute which is most relevant overall, even though some values of that attribute may be irrelevant. All other differences between the two algorithms stem form this: ID3 divides a training set into homogeneous subsets without reference to the class of this subset, whereas PRISM must identify subsets of a specific class. This has the disadvantage of slightly incresed computational effort, but the advantage of an output in the form of modular rules rather than a decision tree.

7.7 Implementation

Version:	0.9
Status:	Experimental
Language:	Common Lisp
Authors:	Lindsey Spratt (spratt@hawk.cs.ukans.edu), Spring 1990, modified by Stefan F. Keller (keller@ifi.unizh.ch), Summer 1991.

References

- Cendrowska, Jadzia (1987); PRISM: An algorithm for inducing modular rules, in: Int. Journal of Man-Machine Studies, Vol. 26, Nr.1,2,4, Vol.27, Nr.2,3,4.

- Cendrowska, Jadzia (1988); PRISM: An algorithm for inducing modular rules, in: B.R.Gaines & J.H.Boose (eds.), Knowledge Acquisition for Knowledge-Based Systems, Academic Press, 253-274.

7.8 Results on Running PRISM on the MONK's Test Sets

TEST PLATFORMS:
Mac: Macintosh Allegro Common Lisp 2.0b2, Macintosh IIci, 4MB memory
Sun: Franz Allegro Common Lisp 4.0.1, Sun sparc/320, 24MB memory

TEST SET 1:
No. of trainig-examples: 124
No. of test-examples: 432
No. of rules induced: 29
Covered test-examples: 86Mac run time: 80.14s, 85.10s, 80.43s, 81.10s, 80.05s
Sun run time: 23.30s, 22.80s, 23.50s, 23.12s, 23.08s
Average run time on Sun: 23.16s

TEST SET 2:
No. of trainig-examples: 169
No. of test-examples: 432
No. of rules induced: 73
Covered test-examples: 73Mac run time: (409.26s)
Sun run time: 121.50s, 122.50s, 120.75s, 122.18s, 121.00s
Average run time on Sun: 121.58s

TEST SET 3:
No. of trainig-examples: 122
No. of test-examples: 432
No. of rules induced: 26
Covered test-examples: 90Mac run time: (59.63s)
Sun run time: 16.77s, 17.00s, 16.63s, 16.60s, 17.30s
Average run time on Sun: 16.86s

7.8.1 Test Set 1 – Rules

```
(RULE-1
  (IF   ((jacket_color 1)))
  (THEN (class 1)))
(RULE-2
  (IF   ((head_shape 3) (body_shape 3)))
  (THEN (class 1)))
(RULE-3
  (IF   ((holding 1) (body_shape 2) (head_shape 2)))
  (THEN (class 1)))
(RULE-4
  (IF   ((body_shape 1) (head_shape 1)))
  (THEN (class 1)))
(RULE-5
  (IF   ((body_shape 2) (head_shape 2)))
  (THEN (class 1)))
(RULE-6
  (IF   ((head_shape 1) (jacket_color 4) (body_shape 3)))
  (THEN (class 0)))
(RULE-7
  (IF   ((jacket_color 2) (holding 2) (has_tie 2)))
  (THEN (class 0)))
(RULE-8
  (IF   ((jacket_color 3) (has_tie 1) (holding 3)))
  (THEN (class 0)))
(RULE-9
  (IF   ((jacket_color 3) (holding 2) (head_shape 1) (has_tie 2)))
  (THEN (class 0)))
(RULE-10
  (IF   ((jacket_color 2) (head_shape 1) (body_shape 3)))
  (THEN (class 0)))
(RULE-11
  (IF   ((jacket_color 4) (body_shape 1) (head_shape 2)))
  (THEN (class 0)))
(RULE-12
  (IF   ((jacket_color 3) (has_tie 1) (body_shape 3)))
  (THEN (class 0)))
(RULE-13
  (IF   ((jacket_color 3) (has_tie 1) (head_shape 2) (body_shape 1)))
  (THEN (class 0)))
(RULE-14
  (IF   ((jacket_color 2) (is_smiling 2) (holding 3) (body_shape 1)))
  (THEN (class 0)))
(RULE-15
  (IF   ((head_shape 1) (body_shape 2) (is_smiling 2)))
  (THEN (class 0)))
(RULE-16
  (IF   ((jacket_color 3) (is_smiling 2) (head_shape 2) (body_shape 3)))
  (THEN (class 0)))
(RULE-17
  (IF   ((jacket_color 2) (is_smiling 2) (head_shape 2) (body_shape 1)))
  (THEN (class 0)))
(RULE-18
  (IF   ((jacket_color 4) (head_shape 1) (is_smiling 1)))
  (THEN (class 0)))
(RULE-19
  (IF   ((jacket_color 2) (holding 2) (body_shape 3)))
  (THEN (class 0)))
(RULE-20
  (IF   ((jacket_color 3) (body_shape 2) (head_shape 1)))
  (THEN (class 0)))
(RULE-21
  (IF   ((jacket_color 2) (body_shape 2) (head_shape 1)))
  (THEN (class 0)))
(RULE-22
```

```
  (IF   ((jacket_color 4) (is_smiling 1) (body_shape 2) (head_shape 3)))
  (THEN (class 0)))
(RULE-23
  (IF   ((jacket_color 3) (head_shape 3) (body_shape 1)))
  (THEN (class 0)))
(RULE-24
  (IF   ((jacket_color 2) (head_shape 3) (body_shape 1)))
  (THEN (class 0)))
(RULE-25
  (IF   ((jacket_color 3) (holding 1) (head_shape 3) (body_shape 2)))
  (THEN (class 0)))
(RULE-26
  (IF   ((jacket_color 4) (holding 3) (has_tie 1) (head_shape 3)))
  (THEN (class 0)))
(RULE-27
  (IF   ((holding 3) (jacket_color 4) (is_smiling 1) (body_shape 3) (head_shape 2)))
  (THEN (class 0)))
(RULE-28
  (IF   ((jacket_color 3) (body_shape 1) (head_shape 2)))
  (THEN (class 0)))
(RULE-29
  (IF   ((jacket_color 2) (is_smiling 2) (holding 3) (has_tie 1)))
  (THEN (class 0)))
```

7.8.2 Test Set 2 – Rules

```
(RULE-1
  (IF   ((holding 1) (jacket_color 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-2
  (IF   ((jacket_color 4) (body_shape 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-3
  (IF   ((head_shape 1) (holding 1) (is_smiling 1)))
  (THEN (class 0)))
(RULE-4
  (IF   ((jacket_color 4) (body_shape 3) (is_smiling 2)))
  (THEN (class 0)))
(RULE-5
  (IF   ((jacket_color 3) (is_smiling 2) (holding 2) (has_tie 2)))
  (THEN (class 0)))
(RULE-6
  (IF   ((has_tie 1) (head_shape 1) (is_smiling 1)))
  (THEN (class 0)))
(RULE-7
  (IF   ((holding 1) (head_shape 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-8
  (IF   ((head_shape 2) (has_tie 2) (body_shape 2) (is_smiling 2)))
  (THEN (class 0)))
(RULE-9
  (IF   ((jacket_color 1) (is_smiling 1) (body_shape 1)))
  (THEN (class 0)))
(RULE-10
  (IF   ((jacket_color 3) (is_smiling 2) (holding 3) (has_tie 2)))
  (THEN (class 0)))
(RULE-11
  (IF   ((holding 1) (jacket_color 1) (is_smiling 1)))
  (THEN (class 0)))
(RULE-12
  (IF   ((is_smiling 2) (jacket_color 2) (body_shape 2)))
  (THEN (class 0)))
```

```
(RULE-13
  (IF   ((jacket_color 3) (has_tie 1) (body_shape 1)))
  (THEN (class 0)))
(RULE-14
  (IF   ((jacket_color 1) (head_shape 1) (body_shape 1)))
  (THEN (class 0)))
(RULE-15
  (IF   ((head_shape 2) (holding 2) (jacket_color 4)))
  (THEN (class 0)))
(RULE-16
  (IF   ((jacket_color 3) (head_shape 3) (body_shape 3) (has_tie 1)))
  (THEN (class 0)))
(RULE-17
  (IF   ((head_shape 2) (holding 2) (body_shape 3) (jacket_color 2)))
  (THEN (class 0)))
(RULE-18
  (IF   ((holding 3) (is_smiling 2) (jacket_color 4) (has_tie 2)))
  (THEN (class 0)))
(RULE-19
  (IF   ((jacket_color 1) (is_smiling 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-20
  (IF   ((jacket_color 3) (head_shape 2) (is_smiling 2) (holding 2)))
  (THEN (class 0)))
(RULE-21
  (IF   ((jacket_color 3) (head_shape 2) (has_tie 2) (holding 2)))
  (THEN (class 0)))
(RULE-22
  (IF   ((jacket_color 3) (head_shape 3) (body_shape 2) (has_tie 2) (holding 3)))
  (THEN (class 0)))
(RULE-23
  (IF   ((holding 1) (is_smiling 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-24
  (IF   ((holding 3) (is_smiling 2) (head_shape 2) (jacket_color 2)))
  (THEN (class 0)))
(RULE-25
  (IF   ((jacket_color 1) (head_shape 1) (is_smiling 1)))
  (THEN (class 0)))
(RULE-26
  (IF   ((is_smiling 2) (holding 3) (body_shape 3) (jacket_color 3)))
  (THEN (class 0)))
(RULE-27
  (IF   ((head_shape 3) (body_shape 3) (jacket_color 2)))
  (THEN (class 0)))
(RULE-28
  (IF   ((body_shape 1) (jacket_color 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-29
  (IF   ((jacket_color 3) (head_shape 3) (holding 2) (body_shape 3)))
  (THEN (class 0)))
(RULE-30
  (IF   ((holding 1) (body_shape 1) (is_smiling 1)))
  (THEN (class 0)))
(RULE-31
  (IF   ((body_shape 2) (jacket_color 3) (has_tie 2) (is_smiling 2)))
  (THEN (class 0)))
(RULE-32
  (IF   ((holding 3) (is_smiling 2) (jacket_color 2) (has_tie 2) (head_shape 3)))
  (THEN (class 0)))
(RULE-33
  (IF   ((body_shape 2) (holding 3) (jacket_color 1) (head_shape 1) (has_tie 1)))
  (THEN (class 0)))
(RULE-34
  (IF   ((jacket_color 3) (holding 3) (head_shape 2) (has_tie 2) (body_shape 2)))
  (THEN (class 0)))
(RULE-35
  (IF   ((jacket_color 2) (is_smiling 1) (body_shape 2)))
```

```
  (THEN (class 1)))
(RULE-36
  (IF   ((jacket_color 2) (body_shape 1) (head_shape 3) (is_smiling 1)))
  (THEN (class 1)))
(RULE-37
  (IF   ((holding 3) (body_shape 3) (jacket_color 1)))
  (THEN (class 1)))
(RULE-38
  (IF   ((jacket_color 2) (body_shape 1) (is_smiling 2) (has_tie 1)))
  (THEN (class 1)))
(RULE-39
  (IF   ((jacket_color 3) (is_smiling 1) (has_tie 2) (body_shape 1)))
  (THEN (class 1)))
(RULE-40
  (IF   ((body_shape 2) (jacket_color 4) (is_smiling 1) (has_tie 2)))
  (THEN (class 1)))
(RULE-41
  (IF   ((jacket_color 3) (body_shape 2) (head_shape 1) (has_tie 2) (is_smiling 1)))
  (THEN (class 1)))
(RULE-42
  (IF   ((body_shape 3) (jacket_color 2) (holding 3) (head_shape 1)))
  (THEN (class 1)))
(RULE-43
  (IF   ((head_shape 2) (has_tie 1) (body_shape 3) (jacket_color 4)))
  (THEN (class 1)))
(RULE-44
  (IF   ((head_shape 2) (has_tie 1) (body_shape 3) (holding 1) (is_smiling 2)))
  (THEN (class 1)))
(RULE-45
  (IF   ((holding 2) (jacket_color 1) (is_smiling 2)))
  (THEN (class 1)))
(RULE-46
  (IF   ((jacket_color 3) (holding 1) (has_tie 2) (body_shape 1)))
  (THEN (class 1)))
(RULE-47
  (IF   ((holding 3) (has_tie 1) (body_shape 2) (jacket_color 3) (is_smiling 2)))
  (THEN (class 1)))
(RULE-48
  (IF   ((has_tie 2) (body_shape 1) (jacket_color 2) (holding 2)))
  (THEN (class 1)))
(RULE-49
  (IF   ((has_tie 2) (body_shape 1) (jacket_color 2) (is_smiling 2) (holding 1)))
  (THEN (class 1)))
(RULE-50
  (IF   ((holding 3) (jacket_color 1) (is_smiling 2) (head_shape 3)))
  (THEN (class 1)))
(RULE-51
  (IF   ((jacket_color 3) (is_smiling 1) (head_shape 2) (holding 2) (has_tie 1)))
  (THEN (class 1)))
(RULE-52
  (IF   ((has_tie 2) (head_shape 1) (jacket_color 2) (holding 2)))
  (THEN (class 1)))
(RULE-53
  (IF   ((body_shape 2) (jacket_color 4) (has_tie 1) (head_shape 2)))
  (THEN (class 1)))
(RULE-54
  (IF   ((has_tie 2) (head_shape 1) (jacket_color 4) (body_shape 1)))
  (THEN (class 1)))
(RULE-55
  (IF   ((jacket_color 3) (holding 1) (has_tie 2) (is_smiling 1) (head_shape 3)))
  (THEN (class 1)))
(RULE-56
  (IF   ((holding 3) (jacket_color 2) (is_smiling 1) (head_shape 2)))
  (THEN (class 1)))
(RULE-57
  (IF   ((body_shape 2) (has_tie 1) (jacket_color 3) (holding 3) (head_shape 2)))
  (THEN (class 1)))
(RULE-58
```

```
  (IF   ((has_tie 2) (head_shape 1) (holding 3) (jacket_color 2)))
  (THEN (class 1)))
(RULE-59
  (IF   ((body_shape 2) (has_tie 1) (jacket_color 4) (is_smiling 2) (holding 2)))
  (THEN (class 1)))
(RULE-60
  (IF   ((has_tie 2) (body_shape 3) (head_shape 1) (holding 3)))
  (THEN (class 1)))
(RULE-61
  (IF   ((jacket_color 3) (head_shape 1) (has_tie 2) (body_shape 3)
                                    (is_smiling 1) (holding 2)))
  (THEN (class 1)))
(RULE-62
  (IF   ((body_shape 2) (jacket_color 1) (is_smiling 2) (has_tie 2)))
  (THEN (class 1)))
(RULE-63
  (IF   ((jacket_color 3) (holding 1) (has_tie 2) (body_shape 3) (is_smiling 2)))
  (THEN (class 1)))
(RULE-64
  (IF   ((body_shape 2) (has_tie 1) (jacket_color 3) (is_smiling 2) (head_shape 1)))
  (THEN (class 1)))
(RULE-65
  (IF   ((head_shape 3) (jacket_color 4) (holding 2) (has_tie 2)))
  (THEN (class 1)))
(RULE-66
  (IF   ((jacket_color 1) (head_shape 2) (is_smiling 2) (has_tie 2)))
  (THEN (class 1)))
(RULE-67
  (IF   ((body_shape 2) (head_shape 3) (is_smiling 1) (holding 2)))
  (THEN (class 1)))
(RULE-68
  (IF   ((body_shape 2) (has_tie 1) (holding 3) (is_smiling 2) (jacket_color 4)))
  (THEN (class 1)))
(RULE-69
  (IF   ((jacket_color 3) (holding 1) (has_tie 2) (is_smiling 1) (head_shape 2)))
  (THEN (class 1)))
(RULE-70
  (IF   ((head_shape 3) (jacket_color 3) (has_tie 1) (holding 3)))
  (THEN (class 1)))
(RULE-71
  (IF   ((head_shape 3) (jacket_color 4) (holding 1) (has_tie 2)))
  (THEN (class 1)))
(RULE-72
  (IF   ((body_shape 2) (has_tie 1) (is_smiling 2) (jacket_color 3) (holding 1)))
  (THEN (class 1)))
(RULE-73
  (IF   ((jacket_color 1) (holding 3) (head_shape 2) (body_shape 2)))
  (THEN (class 1)))
```

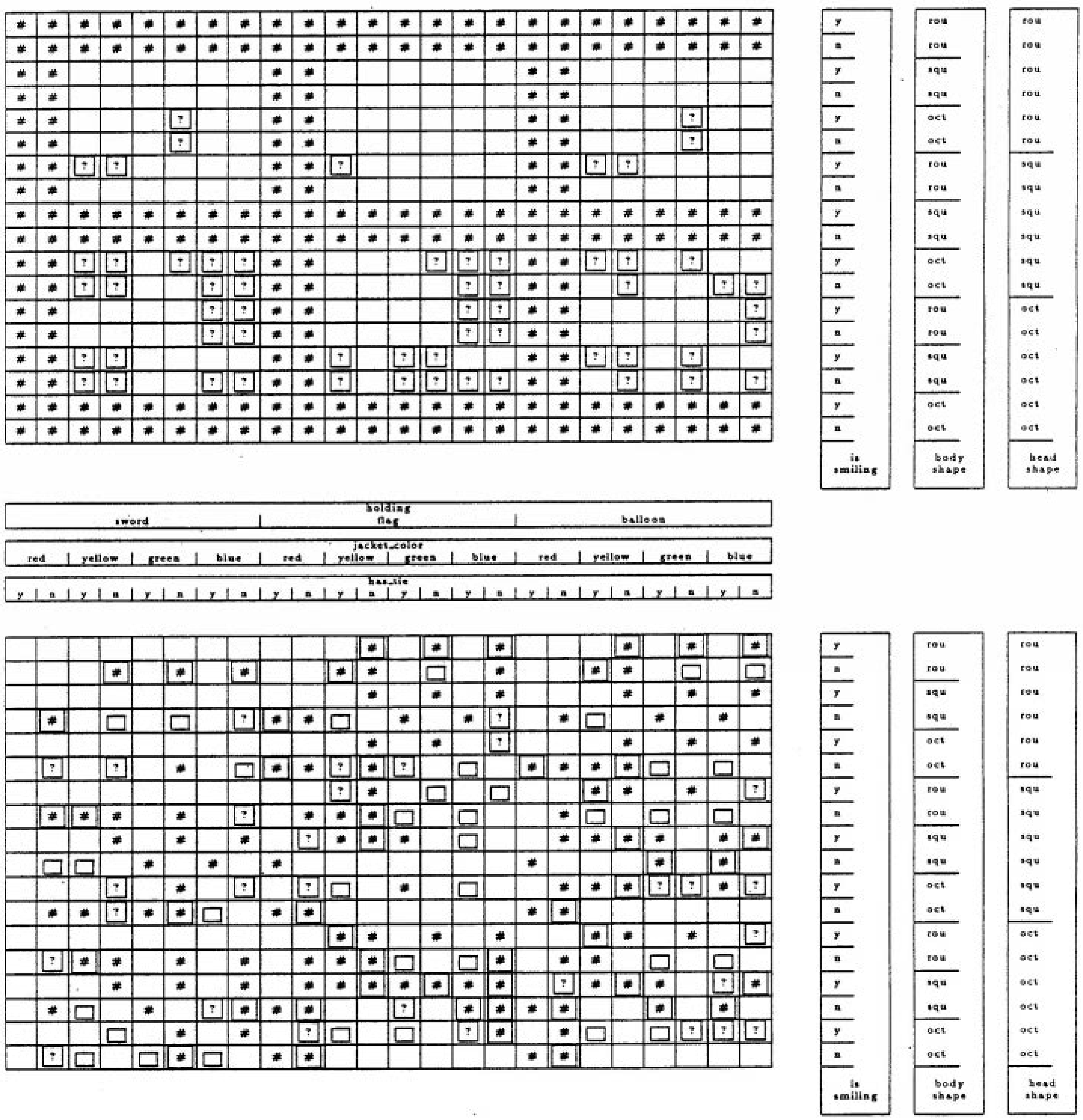
7.8.3 Test Set 3 – Rules

```
(RULE-1
  (IF   ((body_shape 2) (jacket_color 1)))
  (THEN (class 1)))
(RULE-2
  (IF   ((jacket_color 2) (body_shape 1)))
  (THEN (class 1)))
(RULE-3
  (IF   ((body_shape 2) (jacket_color 2) (head_shape 1)))
  (THEN (class 1)))
(RULE-4
  (IF   ((jacket_color 3) (holding 1) (body_shape 2)))
```

```
  (THEN (class 1)))
(RULE-5
  (IF   ((body_shape 1) (jacket_color 1)))
  (THEN (class 1)))
(RULE-6
  (IF   ((jacket_color 3) (body_shape 1) (has_tie 2)))
  (THEN (class 1)))
(RULE-7
  (IF   ((body_shape 2) (jacket_color 2) (has_tie 2)))
  (THEN (class 1)))
(RULE-8
  (IF   ((jacket_color 3) (holding 1) (body_shape 3)))
  (THEN (class 1)))
(RULE-9
  (IF   ((jacket_color 3) (body_shape 1) (is_smiling 2)))
  (THEN (class 1)))
(RULE-10
  (IF   ((jacket_color 3) (body_shape 2) (is_smiling 2)))
  (THEN (class 1)))
(RULE-11
  (IF   ((jacket_color 3) (head_shape 3) (is_smiling 1)))
  (THEN (class 1)))
(RULE-12
  (IF   ((body_shape 2) (head_shape 1) (has_tie 2) (is_smiling 1)))
  (THEN (class 1)))
(RULE-13
  (IF   ((head_shape 3) (holding 1) (is_smiling 1) (body_shape 3)))
  (THEN (class 1)))
(RULE-14
  (IF   ((jacket_color 4) (has_tie 2)))
  (THEN (class 0)))
(RULE-15
  (IF   ((jacket_color 4) (head_shape 1)))
  (THEN (class 0)))
(RULE-16
  (IF   ((body_shape 3) (is_smiling 2)))
  (THEN (class 0)))
(RULE-17
  (IF   ((jacket_color 4) (holding 3)))
  (THEN (class 0)))
(RULE-18
  (IF   ((body_shape 3) (holding 3)))
  (THEN (class 0)))
(RULE-19
  (IF   ((jacket_color 4) (body_shape 1)))
  (THEN (class 0)))
(RULE-20
  (IF   ((body_shape 3) (holding 2)))
  (THEN (class 0)))
(RULE-21
  (IF   ((jacket_color 4) (body_shape 2)))
  (THEN (class 0)))
(RULE-22
  (IF   ((body_shape 3) (head_shape 1)))
  (THEN (class 0)))
(RULE-23
  (IF   ((jacket_color 3) (is_smiling 1) (head_shape 1) (body_shape 1)))
  (THEN (class 0)))
(RULE-24
  (IF   ((jacket_color 3) (holding 3) (head_shape 2) (body_shape 2)))
  (THEN (class 0)))
(RULE-25
  (IF   ((holding 2) (has_tie 1) (is_smiling 1) (body_shape 2) (head_shape 2)))
  (THEN (class 0)))
(RULE-26
  (IF   ((jacket_color 3) (holding 2) (head_shape 1)))
  (THEN (class 0)))
```

7.9 Classification diagrams

(a) Result of PRISM on test set No. 1, Accuracy: 86.3%
(b) Result of PRISM on test set No. 2, Accuracy: 72.7%



Result of PRISM on test set No. 3, Accuracy: 90.3%

holding																										
sword								flag								balloon										
jacket_color																										
red		yellow		green		blue		red		yellow		green		blue		red		yellow		green		blue				
has_tie																										
y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n	is smiling	body shape	head shape
#	#	#	#	[ ]	#			#	#	#	#	[ ]	#			#	#	#	#	[ ]	#			y	rou	rou
#	#	#	#	#	#			#	#	#	#	#	#			#	#	#	#	#	#			n	rou	rou
#	#	#	#	#	#		[#]	#	#	#	#	[ ]	#		[#]	#	#	#	#	[?]	#		[#]	y	squ	rou
#	#	#	#	#	#			#	#	#	#	#	#			#	#	#	#	#	#			n	squ	rou
				#	#																			y	oct	rou
				#	#																			n	oct	rou
#	#	#	#	[?]	#			#	#	#	#	[?]	#			#	#	#	#	[?]	#			y	rou	squ
#	#	#	#	#	#			#	#	#	#	#	#			#	#	#	#	#	#			n	rou	squ
#	#	[?]	#	#	#			#	#	[ ]	#	[ ]	[?]			#	#	[?]	#	[ ]	[ ]			y	squ	squ
#	#	[?]	#	#	#			#	#	[?]	#	#	#			#	#	[?]	#	#	#			n	squ	squ
[?]	[?]	[?]	[?]	#	#	[?]																		y	oct	squ
				#	#																			n	oct	squ
#	#	#	#	#	#			#	#	#	#	#	#			#	#	#	#	#	#			y	rou	oct
#	#	#	#	#	#			#	#	#	#	#	#			#	#	#	#	#	#			n	rou	oct
#	#	[?]	#	#	#			#	#	[?]	#	#	#			#	#	[?]	#	#	#			y	squ	oct
#	#	[?]	#	#	#			#	#	[?]	#	#	#			#	#	[?]	#	#	#			n	squ	oct
[#]	[#]	[#]	[#]	#	#	[#]	[#]					[#]	[#]							[#]	[#]			y	oct	oct
				#	#																			n	oct	oct

Chapter 8

Cobweb and the MONK Problems

Yoram Reich[†]
Douglas Fisher[‡]

[†] Engineering Design Research Center, Carnegie Mellon University, Pittsburgh PA 15213
[‡] Department of Computer Science, Vanderbilt University, Nashville, TN 37235

8.1 COBWEB: A brief overview

This chapter describes the results of applying a variant of the COBWEB system (Fisher, 1987a) called ECOBWEB (Reich, 1991; Reich and Fenves, 1991) to the MONK problems.[1]

COBWEB differs significantly from other systems described in this report. Most notably, the system is *unsupervised*: it does not assume that observations are preclassified (e.g., as positive or negative examples of some concept). Rather, the objective of a clustering system such as COBWEB is to discover 'useful' or 'interesting' categories in a set of observations. COBWEB is also incremental like ID4 (Schlimmer and Fisher, 1986), its descendents, and AQ15 (Michalski, Mozetic, Hong, & Lavrac, 1978), which were described earlier. Observations are not processed *en masse*, but are processed as they are presented to the system.

In particular, COBWEB is an incremental concept formation system that creates hierarchical classification trees over a stream of observations. COBWEB operates on examples described by a list of attribute-value pairs. If examples are classified *a priori* as in supervised systems, and included in an object's description, then this classification is simply treated as another attribute.[2]

Unsupervised clustering systems are guided by some 'internal' metric of quality – some categories must be preferred over others. In COBWEB, a classification is 'good' if an observation's features can be guessed with high accuracy, given that it belongs to a specific (discovered) class. For example, the standard biological classes of `mammals`, `reptiles`, `birds`, etc. are deemed good because knowing that an animal is a mammal (for example) allows many high-confidence predictions about its features (e.g., `has-hair`, `warm-blooded`, `bears-living-young`, etc.). COBWEB makes use of a statistical function that partitions a set of examples into mutually-exclusive classes $C_1, C_2, \ldots, C_n$. The function used by COBWEB is *category utility* (Gluck & Corter, 1985):

$$\frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{n} \tag{8.1}$$

where C_k is a class, $A_i = V_{ij}$ is a property-value pair, $P(x)$ is the probability of x, and n is the number of classes. The first term in the numerator measures the expected number of property-value pairs that can be guessed correctly by using the classification. The second term measures the same quantity <u>without</u> using the classes. Thus, the category utility measures the *increase* of property-value pairs that can be guessed *above* the guess based on frequency alone. The measurement is normalized with respect to the number of classes.

When a new example is introduced, COBWEB tries to accommodate it into an existing hierarchy starting at the root. The system performs one of the following operators:

1. expanding the root, if it does not have any sub-classes, by creating a new class and attaching the root and the new example as its sub-classes;
2. adding the new example as a new sub-class of the root;
3. adding the new example to one of the sub-classes of the root;
4. merging the two best sub-classes and putting the new example into the merged sub-class; or
5. splitting the best sub-class and again considering all the alternatives.

[1] In that reference the name ECOBWEB is not used. A larger system that includes it, called BRIDGER, is discussed.

[2] One way of testing the abilities of an unsupervised system like COBWEB is to see if *a priori* known classifications can be 'rediscovered' in the data. This can be informative for purposes of benchmarking a clustering system, but as Fisher and Pazzani (1991) point out, it is of limited utility.

If the example has been assimilated into an existing sub-class, the process recurses with this class as the top of a new hierarchy. COBWEB again uses category utility to determine the next operator to apply.

COBWEB makes predictions using a mechanism similar to the one used for augmenting the hierarchy by new examples but allowing only operator 3 to apply. COBWEB sorts a partial example description down the hierarchy to find the best host for the partial description. The best host is a leaf node (i.e., a training example) that is used to complete the partial description. It is important to note at this point that the performance task used to evaluate COBWEB and other unsupervised systems (e.g., AUTOCLASS) is different from the performance task for supervised systems. In the latter case, a set of learned rules is used to predict membership relative to an *a prior* known set of classes. In clustering systems, prediction accuracy is measured relative to all descriptive attributes – how well does the classification scheme support prediction of any unknown attribute value? COBWEB seeks to improve classification along all attributes, not simply the single dimension of 'class membership'. Moreover, the system's strategies for classification and prediction bear interesting relationships to other systems. Notably, COBWEB sifts objects down trees like ID3 and related systems, but does so based on the object's known values along *many* attributes at each node in the tree. Thus, COBWEB is a *polythetic* classification system, not a *monothetic* classification system like ID3, which classifies objects based on their value along a single attribute at each decision node.

8.2 ECOBWEB

This section briefly reviews variants on some of COBWEB's mechanisms that are tested within ECOBWEB.

8.2.1 Characteristics prediction

Initial versions of COBWEB sorted observations to a leaf of a classification tree. At this point predictions about the new object's missing values were made, by appealing to this 'best matching' leaf's (i.e., a previously-seen observation) attribute values. However, this strategy can 'overfit' the data, in much the same way that overfitting occurs in supervised systems that maintain overly-specific (i.e., idiosyncratic) rules for class prediction. In the characteristics prediction method ECOBWEB sorts a partial description in the same way as COBWEB does (i.e., using the category utility function to select the class that is the best host for the partial description). The only difference between the current and COBWEB's operation is that if ECOBWEB encounters a characteristic[3] property-value pair that is missing from the partial description, it assigns it to the partial description. If the characteristic is the class attribute, the classification process can terminate. Similarly intented methods were also investigated in Fisher (1989), though we will only experiment with ECOBWEB's strategy here.

8.2.2 Hierarchy correction mechanism

A characteristic of both supervised and unsupervised incremental learning systems is that the rules and/or classification schemes that are developed depend on the order in which training data is encountered. This is best demonstrated in experiments reported by Fisher et al (1991); they tested different orderings and characterized some as 'best-case' orderings (i.e., those leading to 'good' classification schemes'), and others as 'worst-case' orderings. A primary research objective is to mitigate ordering effects in incremental systems.

In ECOBWEB, a hierarchy-correction scheme was designed to mitigate some of the order effects introduced in

[3] Characteristics are property values that satisfy: $P(A_i = V_{ij}|C_k) \geq threshold$ and $P(C_k|A_i = V_{ij}) \geq threshold$, where *threshold* is a pre-determined fixed value.

COBWEB's incremental learning operation. The scheme follows three steps. First, properties that are deemed critical by a domain expert are manually selected as 'triggers'. Second, the hierarchy is traversed top-down. Each class with a characteristic property value that differs from a characteristic in one of the class' ancestors, is removed along with its subtree from the hierarchy. Third, the examples at the leaves of all the removed subtrees are reclassified into the hierarchy. The process can iterate several times until no change of the hierarchy is obtained.

A second mechanism was designed to generate an ordering of examples that will result in a better classification hierarchy than the classification generated by random ordering of examples (e.g., Fisher et al, 1991). There are several variants of this technique. A simple and promising one used by ECOBWEB is created by following the next three steps until the training example set is exhausted. First, calculate the property-value pairs that are most frequent in the examples that were already learned. This can be easily done by looking at the root of COBWEB's classification hierarchy. Second, find an example, in the training examples that have not been learned, that is most distant from the frequent description calculated in the first step. Third, use this example as the next training example.

8.2.3 Information utlity function

ECOBWEB uses the usual category utility function in its operation. In addition, it allows the use of an alternate measure of category quality function. In this function the term $P(A_i = V_{ij}|C_k)^2$ in Equation 8.1 is replaced by $P(A_i = V_{ij}|C_k) \log P(A_i = V_{ij}|C_k)$. This measure was also developed by Gluck and Corter (1985), and has similar, though not identical, biases in the classes that are favored.

8.3 Results

Upon examining the concepts in the MONK problem, it is clear that COBWEB will encounter difficulties in learning them. For example, consider the first concept:

$$(\text{head\_shape} = \text{body\_shape}) \text{ or } (\text{jacket\_color} = \text{red})$$

involves relations between different attributes. Fisher (1987b) notes that probabilistic classification trees contain all the information for calculating correlation probabilities between attributes. This, however, requires using multiple paths of the hierarchy for making 'ideal' predictions. COBWEB, however, makes predictions by ascending in a single path to a leaf node and uses the classification of this leaf to make predictions. Variants of COBWEB that descended multiple paths would undoubtedly perform better in this domain.

Secondly, it is important to note that COBWEB and ECOBWEB are unsupervised systems. The intent of these systems in tasks like data analysis is to discover classes that are interesting and important for purposes of predicting *all* unknown attribute values; discovered categories can then be examined by human analysts to help them search for the interesting aspects on a new domain. It is difficult to imagine a set of rules that imply less natural and less informative categories from the standpoint of most data analysis tasks than those in the MONK suite of problems. Thus, while these problems represent extreme cases that are useful for benchmarking supervised systems, their utility for evaluating unsupervised systems is limited. Nonetheless, COBWEB and its descendents have been evaluated in terms of prediction accuracy. These problems can be used to highlight some of the differences between supervised and unsupervised systems, and the limitations of using unsupervised systems in cases where supervised systems are more appropriate (i.e., in those cases where *a priori* classes are known and the focus of prediction).[4]

Table 8.1 provides the results of ECOBWEB on the MONK's problems. Each of the entries was calculated by running 10 experiments with random orderings of the training examples. The average and the standard deviation of the runs are provided.

Table 8.1: Results of ECOBWEB

Prediction method	#1		#2		#3	
	Ave.	STD	Ave.	STD	Ave.	STD
leaf prediction	0.718	0.042	0.674	0.028	0.682	0.031
leaf prediction with hierarchy correction	0.683	0.020	0.686	0.026	0.681	0.038
leaf prediction with ordering (most distant)	0.732	0.030	0.660	0.038	0.676	0.018
characteristic prediction	0.672	0.027	0.674	0.037	0.665	0.057
characteristic prediction with hierarchy correction	0.674	0.036	0.651	0.053	0.683	0.033
Leaf prediction information utility	0.827	0.077	0.713	0.026	0.680	0.020

Overall, ECOBWEB's performance on this database is inferior to the performance of the other programs. It should be noted that neither the hierarchy correction scheme nor the ordering scheme are sufficient to mitigate order effects; for example, in some of the runs performance as good as 98.8% accuracy were observed for problem #1. No such performance levels were observed, on the other hand, for problems #2 or #3. Figure 8.1 shows one of the classification trees generated from the training examples of the first problem. Similar trees are generated for the other problems as well. The most characteristic attribute is the class. The rest are not so important at the top level. This is probably one of the reasons for the inferior performance of COBWEB. In particular, the hierarchy shows that the characteristic prediction always stops at the first classification level since it finds a characteristic value of the class attribute at that level.

```
                    Class description (# of EXs: 124)
                    Property    Value  P(v|cl)
                    has_tie       2     0.548
                    is_smiling    1     0.524
                    class         1     0.500
                    body_shape    3     0.379
                    head_shape    1     0.363
                    holding       3     0.347
                    jacket_color 4      0.274

Class description (# of EXs: 63)          Class description (# of EXs: 61)
Property    Value  P(v|cl) P(cl|v)        Property    Value  P(v|cl) P(cl|v)
class         0     0.984   1.000         class         1     1.000   0.984
has_tie       2     0.571   0.529         is_smiling    1     0.557   0.523
head_shape    1     0.508   0.711         has_tie       2     0.525   0.471
is_smiling    2     0.508   0.542         jacket_color 1      0.475   1.000
holding       2     0.381   0.615         head_shape    3     0.426   0.703
jacket_color 4      0.365   0.676         holding       1     0.426   0.619
body_shape    3     0.349   0.468         body_shape    3     0.410   0.532
```

Figure 8.1: Two top levels of the classification hierarchy of the first problem

[4] There are also other differences in the biases used to select the MONK problems, and the biases that motivate COBWEB's design. For example, COBWEB's use of probabilistic, polythetic classification is either not exploited by or in sharp contrast to the representation biases implicitly behind the MONK problems.

8.4 Summary

In sum, we have applied ECOBWEB to the MONK problems. This system is unsupervised, and thus results should be interpreted carefully. Our experiments show that the system does not perform as well as supervised alternatives. This highlights the distinction between supervised and unsupervised systems and the different performance tasks that should be used to evaluate systems of each paradigm.

Bibliography

[Fisher, 1987a] Fisher, D. H. (1987a). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(7):139–172.

[Fisher, 1987b] Fisher, D. H. (1987b). *Knowledge Acquisition Via Incremental Conceptual Clustering*. PhD thesis, University of California, Irvine, CA. Available as Technical Report 87-22, Information and Computer Science, University of California, Irvine.

[Fisher, 1989] Fisher, D. H. (1989). Noise-tolerant conceptual clustering. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI*, pages 825–830, San Mateo, CA. Morgan Kaufmann.

[Fisher et al, 1991] Fisher, D., Xu, L., Carnes, R., Reich, Y., Fenves, S., Chen, J., Shiavi, R., Biswas, G., & Weinberg, J. (1991). *Selected Applications of an AI Clustering Technique to Engineering Tasks*. Technical Report 91-07, Computer Science, Vanderbilt University, Nashville, TN.

[Fisher and Pazzani, 1991] Fisher, D. H. (1991). Computational models of concept learning. In Fisher, D. H. J., Pazzani, M. J., and Langley, P., editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pages 3–44, San Mateo, CA. Morgan Kaufmann.

[Gluck and Corter, 1985] Gluck, M. and Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society, Irvine, CA*, pages 283–287, San Mateo, CA. Academic Press.

[Michalski, Mozetic, Hong, and Lavrac, 1978] Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N. (1978). *The Multipurpose Incremental Generation of VL1 Hypotheses: the underlying methodology and the description of programs ESEL and AQ11*. Technical Report 867, Computer Science, University of Illinois, Urbana.

[Reich, 1991] Reich, Y. (1991). *Building and Improving Design Systems: A Machine Learning Approach*. PhD thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA. Available as Technical Report EDRC 02-16-91.

[Reich and Fenves, 1991] Reich, Y. and Fenves, S. J. (1991). The formation and use of abstract concepts in design. In Fisher, D. H. J., Pazzani, M. J., and Langley, P., editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pages 323–353, San Mateo, CA. Morgan Kaufmann.

[Schlimmer and Fisher, 1986] Schlimmer, J. and Fisher, D. (1986). A case study in incremental learning. In *Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA*, pages 496–501, San Mateo, CA. Morgan Kaufmann.

Chapter 9

Backpropagation on the MONK's Problems

Sebastian B. Thrun

Carnegie-Mellon University, School of Computer Science, Pittsburgh, PA 15213
e-mail: Sebastian.Thrun@cs.cmu.edu

9.1 Introduction

This paper briefly describes the results of the plain backpropagation algorithm [1] obtained on the three MONK's problems. Backpropagation is a function approximation algorithm for multilayer feed-forward perceptrons based on gradient descent. Conversely to many symbolic learning algorithms, backpropagation learns functions by nonlinear L_2-approximations. This technique has been successfully applied to a variety of real-world problems like speech recognition, bomb detection, stock market prediction etc.

Although multilayer networks represent continuous functions, they are frequently restricted to binary classification tasks as the MONK's problems. In all three cases we used the following architecture: There were 17 input units, all having either value 0 or 1 corresponding to which attribute-value was set. All input units had a connection to 3 (first MONK's problem), 2 (second problem) or 4 (third problem) hidden units, which itself were fully connected to the output unit. An input was classified as class member if the output, which is naturally restricted to $(0, 1)$, was $\geq .5$. Training took between ten and thirty seconds on a SUN Sparc Station for each of the three problems. On a parallel computer, namely the Connection Machine CM-2, training time was further reduced to less than 5 seconds for each problem. The following results are obtained by the plain, unmodified backpropagation algorithm. These results reflect what an unexperienced user would obtain by running backpropagation on the MONK's problems.

	training epochs	accuracy
MONK's # 1	390	100%
MONK's # 2	90	100%
MONK's # 3	190	93.1%

However, in the third training set, the error did never approach zero in all runs we performed, which indicated the presence of noise and/or a local minimum. This important observation led us to refine the results for the third problem using *weight decay*[1] [1,2]. This wideley used technique often prevents backpropagation nets from overfitting the training data and thus improves the generalization. With weight decay $\alpha = 0.01$ we improved the classification accuracy on this third set significantly and, moreover, the concept learned was the same for all architectures we tested (i,e, 2, 3, or 4 hidden units).

	training epochs	accuracy
MONK's # 3 with weight decay	105	97.2%

Backpropagation with weight decay learned the correct concepts for the first two MONK's problems again with 100% accuracy. These classification results clearly demonstrate the appropriatenes of the backpropagation algorithm on problems as the MONK's problems.

References

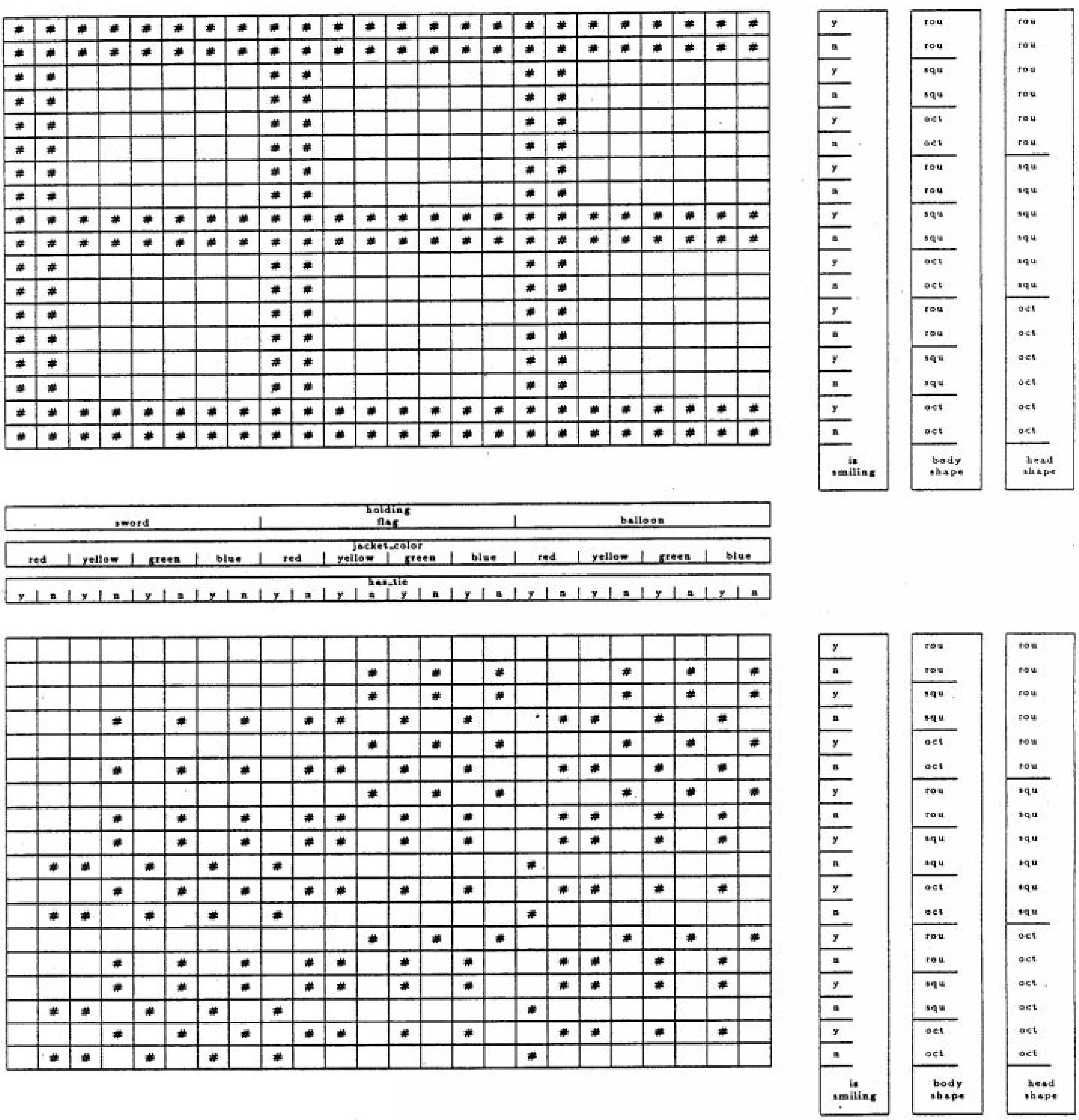
[1] Rumelhart, D. E. and McClelland, J. *Parallel Distributed Processing. Vol. I + II*, MIT Press 1986

[2] Chauvin, Y. Dynamic Behavior of Contrained Backpropagation Networks. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann Publishers, 1990.

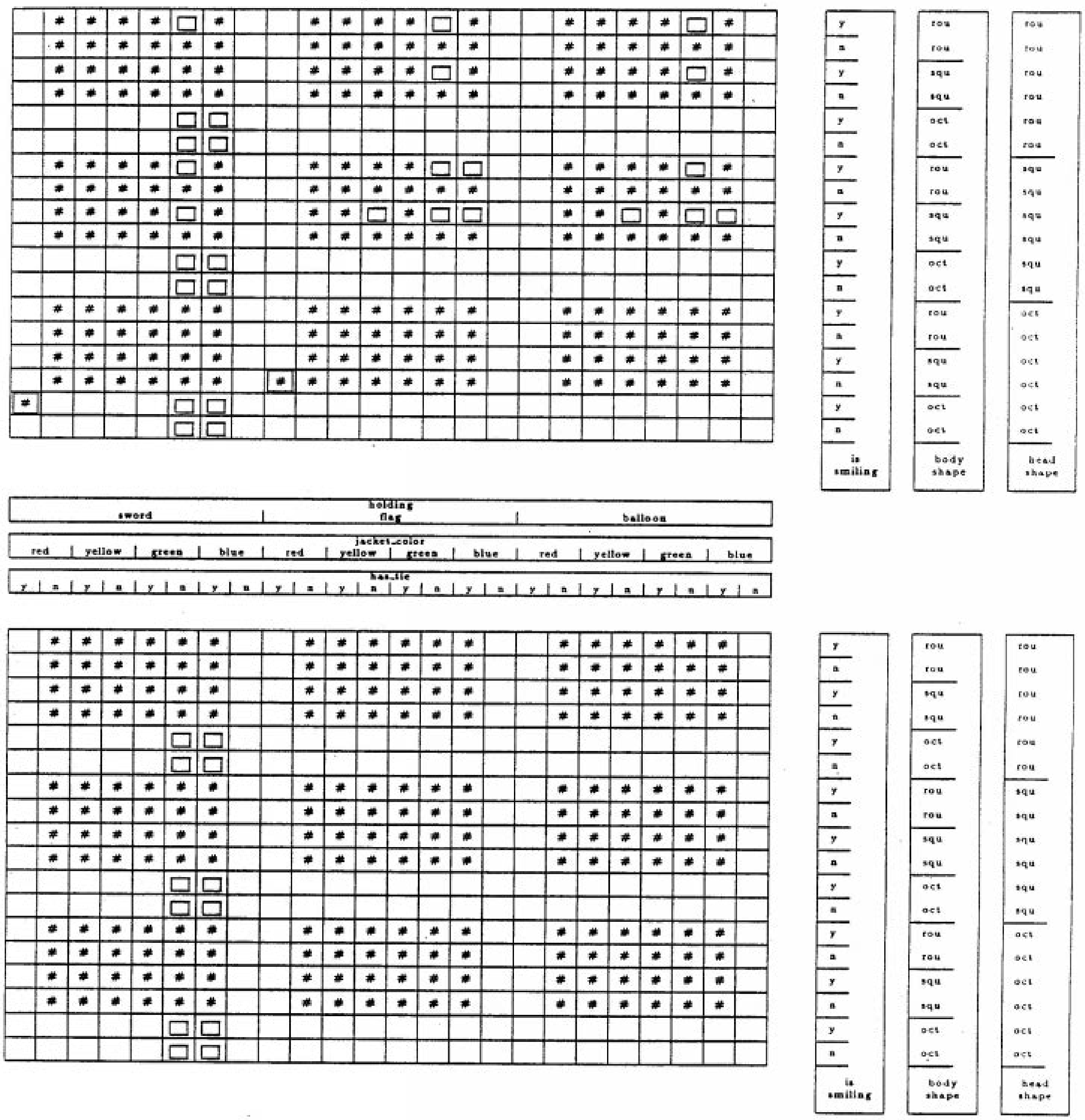
[1] In our implementation, weight decay was realized by minimizing the complexity term $\alpha \cdot \frac{1}{4}(\sum_{i,j} w_{ij}^4 + \sum_i \theta_i^4)$ in addition to the conventional L_2-error term over the training set. Here α is a constant factor, w_{ij} denotes the weight from unit j to unit i, and θ_i the threshold (bias) of unit i.

9.2 Classification diagrams

(a) Results of BACKPROP (with/without weight decay) on test set 1, Accuracy: 100.0%
(b) Results of BACKPROP (with/without weight decay) on test set 2, Accuracy: 100.0%



(a) Results of BACKPROP without weight decay on test set 3, Accuracy: 93.1%
(b) Results of BACKPROP with weight decay on test set 3, Accuracy: 97.2%



9.3 Resulting weight matrices

MONKS's problem # 1: weights and biases

	to-node			
from-node	hidden_1	hidden_2	hidden_3	output
input_1 (head_shape round)	-6.503145	0.618412	-1.660409	
input_2 (head_shape square)	1.210703	1.939613	2.972592	
input_3 (head_shape octagon)	5.356444	-3.597301	-1.266992	
input_4 (body_shape round)	-6.692434	2.129635	-2.032242	
input_5 (body_shape square)	6.457639	0.864312	4.260765	
input_6 (body_shape octagon)	0.225053	-2.428098	-1.839603	
input_7 (is_smiling yes)	0.096995	0.131133	0.053480	
input_8 (is_smiling no)	-0.011828	0.135277	0.107302	
input_9 (holding sword)	-0.076848	0.459903	-0.008368	
input_10 (holding balloon)	-0.016940	0.151738	0.148955	
input_11 (holding flag)	-0.087298	0.196521	0.023554	
input_12 (jacket_color red)	5.735210	4.337359	-0.865479	
input_13 (jacket_color yellow)	-2.257168	-1.410376	0.494681	
input_14 (jacket_color green)	-2.232257	-1.109825	0.382717	
input_15 (jacket_color blue)	-1.710642	-1.452455	0.479513	
input_16 (has_tie yes)	-0.109696	0.434166	0.276487	
input_17 (has_tie no)	-0.111667	0.131797	0.310714	
bias	0.486541	0.142383	0.525371	
hidden_1				9.249339
hidden_2				8.639715
hidden_3				-9.419991
bias				-3.670920

MONKS's problem # 2: weights and biases

	to-node		
from-node	hidden_1	hidden_2	output
input_1 (head_shape round)	-4.230213	3.637149	
input_2 (head_shape square)	1.400753	-2.577242	
input_3 (head_shape octagon)	1.479862	-2.492254	
input_4 (body_shape round)	-4.363966	3.835199	
input_5 (body_shape square)	1.154510	-2.347489	
input_6 (body_shape octagon)	1.542958	-2.227530	
input_7 (is_smiling yes)	-3.396133	2.984736	
input_8 (is_smiling no)	1.868955	-2.994535	
input_9 (holding sword)	-4.041057	4.239548	
input_10 (holding balloon)	1.293933	-2.195403	
input_11 (holding flag)	1.160514	-2.272035	
input_12 (jacket_color red)	-4.462360	4.451742	
input_13 (jacket_color yellow)	0.749287	-1.869545	
input_14 (jacket_color green)	0.640353	-1.727654	
input_15 (jacket_color blue)	1.116349	-1.332642	
input_16 (has_tie yes)	-3.773187	3.290757	
input_17 (has_tie no)	1.786105	-3.296139	
bias	-1.075762	-0.274980	
hidden_1			-11.038625
hidden_2			-9.448544
bias			5.031395

MONKS's problem # 3: weights and biases (without weight decay)

	to-node				
from-node	hidden_1	hidden_2	hidden_3	hidden_4	output
input_1 (head_shape round)	0.277334	-0.673423	-0.345908	0.121908	
input_2 (head_shape square)	1.759524	1.150119	0.098689	0.329486	
input_3 (head_shape octagon)	-1.328410	0.941278	0.059910	0.017674	
input_4 (body_shape round)	-3.466870	-0.022787	0.222484	0.214138	
input_5 (body_shape square)	-2.460525	3.988668	-0.021681	0.235819	
input_6 (body_shape octagon)	6.622062	-3.396938	0.125944	-0.134328	
input_7 (is_smiling yes)	1.615026	0.224221	-0.317908	-0.594920	
input_8 (is_smiling no)	-1.433791	-0.183452	-0.326539	0.361663	
input_9 (holding sword)	-0.780008	-0.786854	0.072768	0.507106	
input_10 (holding balloon)	0.733984	-0.260836	0.004670	0.422573	
input_11 (holding flag)	0.415208	1.410443	-0.023262	0.325766	
input_12 (jacket_color red)	-3.263737	1.324415	0.025837	-0.154449	
input_13 (jacket_color yellow)	-1.896538	1.518800	0.351912	0.044775	
input_14 (jacket_color green)	-0.432256	-0.183302	0.057546	-0.058255	
input_15 (jacket_color blue)	5.090627	-3.446529	-0.082472	0.131738	
input_16 (has_tie yes)	0.897500	-0.717589	0.314088	0.099872	
input_17 (has_tie no)	-0.502348	0.954327	-0.074583	-0.339295	
bias	0.364889	0.248641	-0.484047	-0.227007	
hidden_1					-11.548968
hidden_2					6.567443
hidden_3					-0.117112
hidden_4					-0.064650
bias					0.191083

MONKS's problem # 3: weights and biases (with weight decay)

	to-node		
from-node	hidden_1	hidden_2	output
input_1 (head_shape round)	-0.029477	-0.008986	
input_2 (head_shape square)	-0.376094	-0.364778	
input_3 (head_shape octagon)	-0.051924	-0.028672	
input_4 (body_shape round)	0.991798	0.991750	
input_5 (body_shape square)	1.031170	1.027708	
input_6 (body_shape octagon)	-1.284263	-1.279808	
input_7 (is_smiling yes)	-0.303940	-0.314212	
input_8 (is_smiling no)	-0.216766	-0.221040	
input_9 (holding sword)	-0.064305	-0.052110	
input_10 (holding balloon)	-0.257165	-0.243988	
input_11 (holding flag)	-0.131509	-0.122790	
input_12 (jacket_color red)	1.001415	1.004192	
input_13 (jacket_color yellow)	0.898066	0.896869	
input_14 (jacket_color green)	0.670929	0.673218	
input_15 (jacket_color blue)	-1.280272	-1.272798	
input_16 (has_tie yes)	-0.354472	-0.355268	
input_17 (has_tie no)	0.040973	0.037927	
bias	-0.319686	-0.343492	
hidden_1			1.762523
hidden_2			1.759077
bias			-1.501492

Chapter 10

The Cascade-Correlation Learning Algorithm on the MONK's Problems

Scott E. Fahlman

Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213
e-mail: Scott.Fahlman@cs.cmu.edu

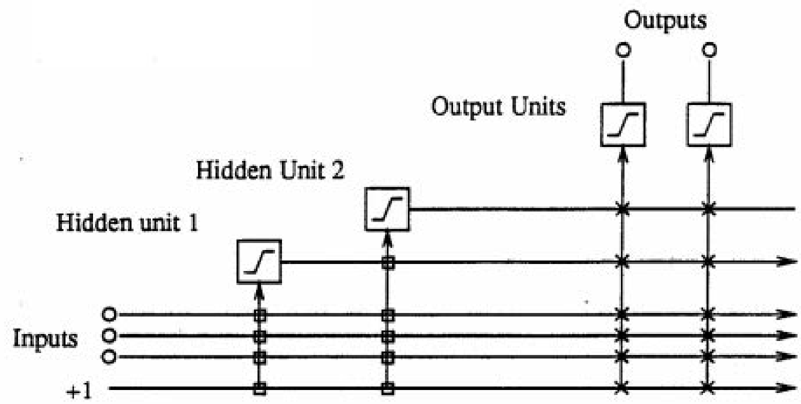


Figure 10.1: see text for details

10.1 The Cascade-Correlation algorithm

Cascade-Correlation [Fahlman, 1990] is a supervised neural network learning architecture that builds a near-minimal multi-layer network topology in the course of training. Initially the network contains only inputs, output units, and the connections between them. This single layer of connections is trained (using the Quickprop algorithm [Fahlman, 1988]) to minimize the error. When no further improvement is seen in the level of error, the network's performance is evaluated. If the error is small enough, we stop. Otherwise we add a new hidden unit to the network in an attempt to reduce the residual error.

To create a new hidden unit, we begin with a pool of *candidate units*, each of which receives weighted connections from the network's inputs and from any hidden units already present in the net. The outputs of these candidate units are not yet connected into the active network. Multiple passes through the training set are run, and each candidate unit adjusts its incoming weights to maximize the correlation between its output and the residual error in the active net. When the correlation scores stop improving, we choose the best candidate, freeze its incoming weights, and add it to the network. This process is called "tenure." After tenure, a unit becomes a permanent new feature detector in the net. We then re-train all the weights going to the output units, including those from the new hidden unit. This process of adding a new hidden unit and re-training the output layer is repeated until the error is negligible or we give up. Since the new hidden unit receives connections from the old ones, each hidden unit effectively adds a new layer to the net. (See figure 1.)

Cascade-correlation eliminates the need for the user to guess in advance the network's size, depth, and topology. A reasonably small (though not minimal) network is built automatically. Because a hidden-unit feature detector, once built, is never altered or cannibalized, the network can be trained incrementally. A large data set can be broken up into smaller "lessons," and feature-building will be cumulative.

Cascade-Correlation learns much faster than backprop for several reasons: First only a single layer of weights is being trained at any given time. There is never any need to propagate error information backwards through the connections, and we avoid the dramatic slowdown that is typical when training backprop nets with many layers. Second, this is a "greedy" algorithm: each new unit grabs as much of the remaining error as it can. In a standard backprop net, the all the hidden units are changing at once, competing for the various jobs that must be done—a slow and sometimes unreliable process.

10.2 Results

For all these problems I used the standard Common Lisp implementation of Cascade-Correlation on a Decstation 3100. This code is public-domain and is available to outside users via anonymous FTP. Contact `sef@cs.cmu.edu` for details.

I used the same parameters in all of these tests. Here is the printout of those parameters:

SigOff 0.10	WtRng 1.00	WtMul 1.00		
OMu 2.00	OEps 1.00	ODcy 0.0000	OPat 20	OChange 0.010
IMu 2.00	IEps 1.00	IDcy 0.0000	IPat 15	IChange 0.030
Utype :GAUSSIAN	Otype :SIGMOID	RawErr NIL	Pool 8	
(train 100 100 10)				

Monk #1:

After 95 epochs, 1 hidden unit: 0 Errors on training set. 0 Errors on test set.
Elapsed real time: 5.11 seconds

Monk #2:

After 82 epochs, 1 hidden unit: 0 Errors on training set. 0 Errors on test set.
Elapsed real time: 7.75 seconds

Monk #3:

After 259 epochs, 3 hidden units: 0 Errors on training set. 40 errors on test set (i.e. accuracy 95.4%).
Elapsed real time 12.27 seconds.

Training and test-set performance was tested after each output-training phase. The minimum test-set error was observed after the initial output-training phase, before any hidden units were added. (Not surprising, since with no noise this problem is linearly separable.) Using any sort of cross-validation system, this is where the algorithm would stop.

At that point, the results were as follows:

Training: 7 of 122 wrong:

Head: RND	Body: RND	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: T
Head: RND	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: YEL	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: N	Output: T
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: BLU	Tie: Y	Output: NIL

Test: 14 of 432 wrong:

Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL

So on the test set, performance is 96.7%

By turning up the OUTPUT-DECAY parameter to 0.1 (an odd thing to do, but sometimes useful when the training set is too small for good generalization), we can do a little better. After the initial output-training phase:

Training: 8 of 122 wrong:

Head: RND	Body: RND	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: T
Head: RND	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: YEL	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: N	Output: T
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: BLU	Tie: Y	Output: NIL

Test: 12 of 432 wrong:

Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL

Score on test set: 97.2%

We can see here what the problem is: All the bad test-set cases are Green and holding a sword, so they should be true. But this positive value is not strong enough to offset the negative weight from Octagonal body.

In the training set, there are only two examples showing the green-sword combination overpowering an octagonal body, and that is apparently not enough to make the point. There are 11 cases showing that octagonal/sword should be negative and 8 cases showing that octagonal/green should be negative.

If we switch the training and test set, we see how easy it is to solve this problem in the absence of noise and small-sample fluctuations.

Switching the training and test set: After 16 epochs and 0 hidden units:

Training: 0 of 432 wrong. Test: 6 of 122 wrong.

Head: RND	Body: RND	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: T
Head: RND	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: YEL	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: N	Output: T
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: BLU	Tie: Y	Output: NIL

These, I believe, are exactly the noise cases deliberately inserted in the original training set. Note that three of these noise cases are

Square/Square/Yes $\Longrightarrow$ NIL (when T is correct)

This explains the other two error cases observed in the first run of this problem. If we look at square/square/yes cases in the training set, NIL cases outnumber T cases, 5 to 3.

Bibliography

[Fahlman, 1988] Fahlman, S. E. (1988) "Faster-Learning Variations on Back-Propagation: An Empirical Study" in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann.

[Fahlman, 1990] Fahlman, S. E. and C. Lebiere (1988) "The Cascade-Correlation Learning Architecture" in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann.

10.3 Classification diagrams

(a) Training set #3 first run, Accuracy: 96.8%
(b) Training set #3 second run, Accuracy: 97.2%