HYPOTHESIS-DRIVEN CONSTRUCTIVE INDUCTION IN AQ17: A METHOD AND EXPERIMENTS

by

J. Wnek
R. S. Michalski

Proceedings of the IJCAI-91 Workshop on Evaluating and Changing Representation in Machine Learning, Sydney, Australia, August 1991.

HYPOTHESIS-DRIVEN CONSTRUCTIVE INDUCTION IN AQ17: A Method and Experiments

J. Wnek and R. S. Michalski

P 91-9
MLI 91-4

Submitted to the IJCAI-91 Workshop on
Evaluating and Changing Representation in Machine Learning

HYPOTHESIS-DRIVEN CONSTRUCTIVE INDUCTION IN AQ17: A Method and Experiments

J. Wnek and R. S. Michalski

Abstract

This paper presents a method for constructive induction in which new problem-relevant attributes are generated by analyzing iteratively created inductive hypotheses. The method starts by creating a set of rules from given examples using the AQ algorithm. These rules are then evaluated according to a "rule quality criterion." Subsets of the best-performing rules for each decision class are selected to form new attributes. These new attributes are used to reformulate the training examples used in the previous step, and the whole inductive process starts again. This iterative process ends when the performance of the rules exceeds a determined threshold. In the experiments on learning different DNF functions, the method outperformed in terms of predictive accuracy both, the AQ15 rule learning method, as well as the REEDWOOD decision tree learning method.

Acknowledgements

The authors thank Giulia Pagallo for the help in the design of the experiment, and Kenneth De Jong for comments on the paper. This work was done in the Center for Artificial Intelligence at George Mason University. Research in the Center for Artificial Intelligence is supported in part by the Defense Advanced Research Projects agency under grant N0. N00014-87-K-0874, administered by the Office of Naval Research, and in part by the Office of Naval Research under grants No. N00014-88-K-0226, No. N00014-88-K-0397, and No. N00014-91-J-1351.

1. Introduction

Most programs for inductive learning from examples create descriptions that contain attributes that are selected from those present in the original examples. Such programs, e.g., AQ11 (Michalski and Larson, 1978), ID3 (Quinlan, 1983), ASSISTANT (Cestnik et al., 1987), CN2 (Clark and Niblett, 1989)), do not create new attributes (or, in general, concepts) in the process of learning. In contrast to such *selective* inductive programs, a *constructive* induction program is able to generate and use new concepts in the hypothesized description. These new concepts can be attributes, predicates, terms, operators, etc., and should be more "relevant" to the learning problem than those initially given.

A constructive learning program thus performs a transformation of the knowledge representation space (Michalski, 1980). The creation of new concepts that are more relevant to a learning problem than the initial ones has turned out to be a very difficult task. In general, new concepts are created by employing background knowledge, and/or by a trial and test search process that constructs various combinations of the initially supplied concepts.

This paper is concerned with constructive induction that is able to create more relevant attributes than those initially given. One approach to automate the formation of new attributes is to extend the initial set of attributes with arithmetic and Boolean combinations of the primitive attributes of a fixed type and size (Utgoff, 1986). Another approach is to let the learning algorithm adaptively define its own attributes while learning. The capability of a learning system to adaptively enlarge the initial concept description language has been sometimes called a dynamic bias (Utgoff, 1986).

There have been several systems developed that exhibit some constructive induction capabilities. Among them one can list the INDUCE program that generates new attributes or predicates by applying "constructive generalization rules" (Michalski, 1980), the LEX system for acquiring and refining problem-solving heuristics (Mitchell, Utgoff and Banerji, 1983); the BACON system for discovering mathematical expressions representing physical or chemical laws (Langley, Bradshaw and Simon, 1983) and the EURISKO program that discovers new heuristics (Lenat, 1983). More recently, Schlimmer (1986) and Muggleton (1987) described various new efforts toward constructive induction. Pagallo and Haussler (1990) proposed three learning algorithms, FRINGE, GREEDY3 and GROVE, that adaptively introduce relevant attributes while learning a decision tree or decision list from examples.

In an attribute-based learning system with a fixed concept representation language, the level of abstraction ("grain") of attributes strongly affects the complexity of the hypothesized rules. By employing high-level attributes, the concept representation can be greatly simplified. Such attributes may, however, be encoded as very complex functions of low-level primitives. In such a case, the learning system will have to compile these complex functions (e.g., Flann and Dietterich, 1986). The usual goal of constructive induction is therefore to discover attributes that lead to the maximal simplification of the generated hypotheses.

An alternative, although related goal of constructive induction can be to discover attributes that produce the best performing hypotheses, that is, to emphasize primarily the predictive performance of a hypothesis, rather than its overall simplicity. The predictive performance can be measured by applying the hypothesis to the testing data, and determining the correctness of the predictions. One important aspect of the constructive induction method proposed in this paper is that its primary goal is to increase the predictive performance.

Another important aspect of the proposed method is that it generates new attributes by analyzing the hypothesis initially created by a selective induction process, and then by consecutive learning steps. For that reason, the method is called a "hypothesis-driven" constructive induction (HCI[1]). To generate the initial and consecutive selective hypotheses we applied the rule learning program AQ15 (Michalski el al., 1986). AQ15 learns rules from examples represented as sequences of attribute-value pairs. Attributes can be of different types, such as nominal, linear or structured. The teacher presents to the learner a set of examples of every concept or decision class under consideration. The program outputs a set of general decision rules for each class that cover all the examples of the class and none of the other classes (i.e. consistent and complete descriptions). The rules generated optimize a problem-dependent "criterion of preference." In the case of noisy data, the program may generate only partially consistent or complete rules.

The program is based on the AQ algorithm, which recessively employs a "star" generation procedure (Michalski et al., 1986). A "star" of an example is the set of all alternative general rules that cover that example, but do not cover any negative examples. After a star is generated, the "best" rule in it, as defined by the preference criterion, is selected, and all examples covered by the

---

[1] This abbreviation is already used as Human-Computer-Interface. However, Constructive Induction methods are in the spirit of this kind of interface. The interface is built on the knowledge level, where a computer communicates discovered ideas and partial solutions to a human. The user can take advantage of the discovered ideas, test them, and make use of them.

rule are removed from further consideration. A new example (called seed) is selected from the yet-uncovered examples, and the process repeats. The algorithm stops when all positive examples are covered. If all the examples of a given class can be covered just by one rule only (that is, there exists a conjunctive characterization of the concept), the algorithm terminates after the first step. In the presented method, the above algorithm is combined with a process of iteratively generating new attributes, and using them in subsequent learning steps.

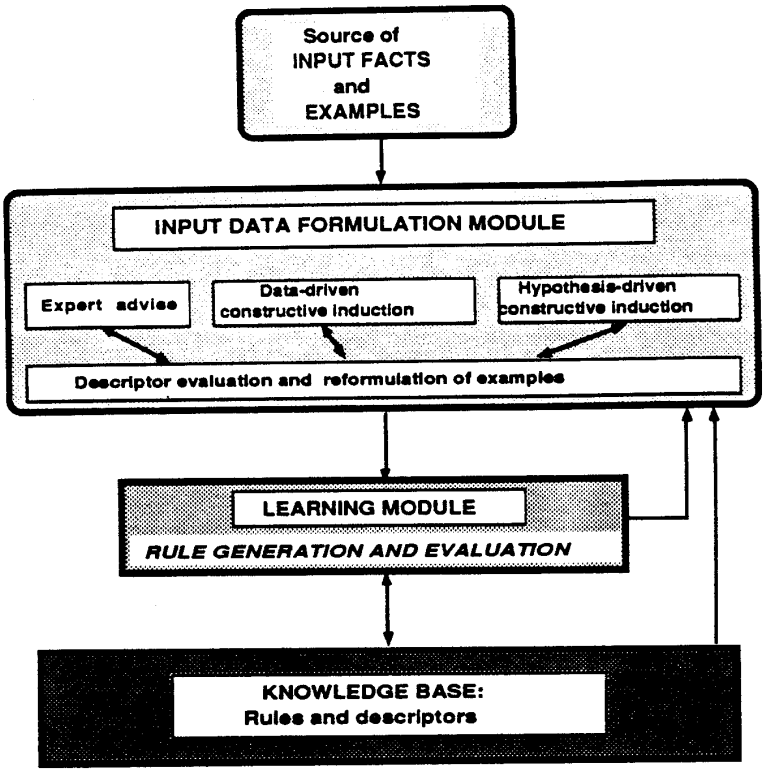

Figure 1. AQ17 with different types of constructive induction

Figure 1 presents a functional diagram of the AQ17 program where the HCI method was implemented as one of the four constructive induction mechanisms:

a) constructive generalization rules (Michalski, 1983)

b) domain knowledge encoded in the form of arithmetic and logic rules (A-rules and L-rules) (Michalski et al., 1986)

c) data-driven constructive induction (Bloedorn and Michalski, 1991)

d) hypothesis-driven constructive induction

2. A description of the HCI method

As mentioned above, the proposed HCI method ('hypothesis-driven constructive induction') determines new problem-relevant attributes by analyzing the currently held inductive hypothesis. Its primary goal is to determine new attributes such that they lead to a maximal improvement of the predicted performance of the hypothesis. A natural extension of this goal is detection and removal from the input data any of irrelevant attributes. (Such attributes may be introduced in the process of *input data formulation* by a domain expert, or by the *new descriptor generation* module.)

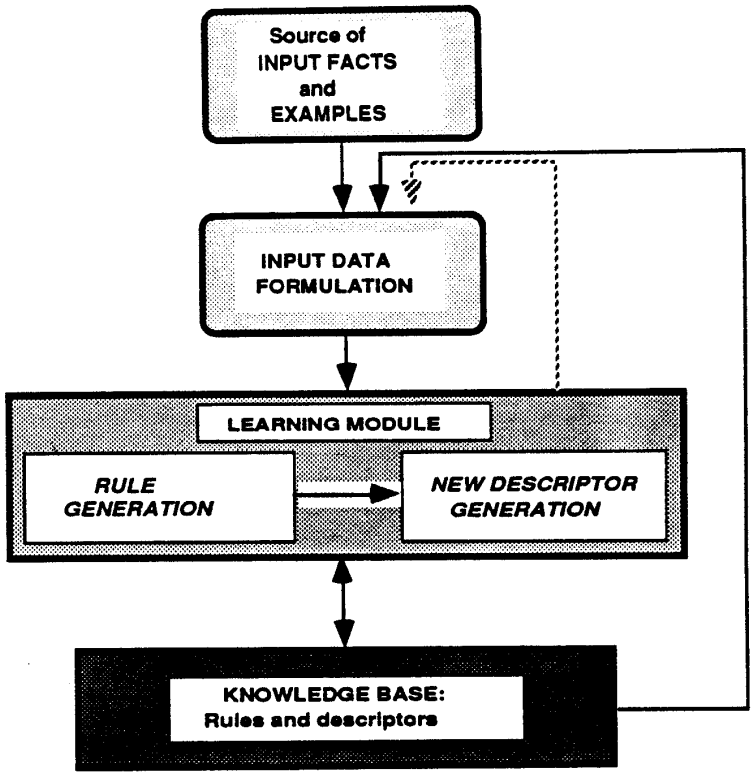

Figure 2. HCI method built into a rule based system

Basic steps of the proposed HCI method are:

1. Induce rules from a selected sample of a training set
2. For each decision class generate one candidate attribute that corresponds to a subset of the highest quality rules, and identify irrelevant attributes

3. Modify the set of training examples by adding the values of newly generated attributes and removing irrelevant ones
4. Induce rules from the modified selected sample of the training set
5. Evaluate the predictive performance of the rules on the remaining training examples; If the performance does not exceed a predefined threshold then go to step 1
6. Induce rules from the complete training set using modified description language.

The steps 1, 4, and 6 are performed by the AQ15 program. The heuristic for constructing the appropriate attribute in Step 2 includes extracting a part of a classification hypothesis which contains *best rules*. This is done by sorting all rules from the output hypothesis according to their *t2u-weights* and selecting the maximum number of rules with the highest t2u-weights to fulfill the inequality (1):

$$[(\Sigma\ t2u_{best}) / (\Sigma\ t2u_{all})] < TH1 \qquad (1)$$

$$t2u = t\text{-weight} + (2 * u\text{-weight}) \qquad (2)$$

where, t-weight is the total number of positive examples covered by the rule, u-weight is the number of positive examples that are uniquely covered by the rule (no other rule in the hypothesis covers those examples). The constant number "2" in the formula (2) was determined experimentally. It means that rules with higher uniqueness are preferred in constructed attribute. The TH1 threshold is a program parameter and is primarily set up to 0.65. The number of *best rules* defined by the TH1 threshold can be extended by those remaining rules with t2u-weights greater than (TH2 * $t2u_{best\text{-}min}$). $t2u_{best\text{-}min}$ is the minimal weight among *best rules*. The TH2 threshold was equal to 0.65. The primary setup of TH1 and TH2 thresholds is due to the assumption of the nature of constructed attributes: they should convey to the next learning session the most precise knowledge of the learned problem as possible. The setting is flexible enough to cover the *base* of a learned concept and express it as a new attribute. The remaining rules, not included in the new description, will hold exemptions or noise in the data. The role of the TH2 threshold is to assure that *all* strong rules from a hypothesis will form a new attribute.

Initially, there are only two values assigned to the candidate attribute that characterize a class membership. More values can be added if the same attribute description occurred in other classes. The new attributes are logical expressions of the old attribute values. After new attributes were being constructed, the training set was updated with new attribute values: for each training example the values of the new attributes are calculated by evaluating the logical expression characterizing the new attribute.

From the outline of the method we can see that the process of inducing rules from examples may be repeated several times in order to achieve the desired predictive performance. This could add some complexity to the learning algorithm depending on how many times steps 1-5 were repeated. The complexity of inducing rules from examples in AQ15 is O(MN), where M is number of positive examples and N is number if negative examples (every positive example a generalized against all negative examples). The HCI multiplies this complexity by factor 2*I, where I is a number of iterations of steps 1-5. Also, all examples have to be modified with the values of new attributes which requires a simple evaluation of DNF formulas.

On the other hand, new attributes introduced in each iteration in the form of learned sub-concepts from the searched hypothesis space make the learning problem easier. The training set was already pre-partitioned by problem oriented and statistically significant attributes. Most of the training examples will be covered with the new attributes, and only exemptions will require building additional concept descriptions.

3. Exemplary problem

To illustrate the performance of the constructive search, we describe an experiment on learning a multiplexer function with 3 inputs and 8 outputs: the so-called multiplexer-11 problem (Wilson, 1987). For each positive integer *k*, there exists a multiplexer function defined on a set of $k + 2^k$ attributes or bits. The function can be defined by thinking of the first k attributes as *address bits* and the last attributes as *data bits*. The function has the value of the data bit indexed by the address bits. In the experiment, the input examples were encoded in terms of 11 binary attributes. Thus, the description space contains 2048 elements. The training set had 64 (6%) positive examples and 64 (6%) of the negative examples. Table 1 shows a sample of positive and negative examples. The attributes a0, a1, a2 describe address lines, and d0-d7 describe data lines.

Positive examples

a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	1
0	1	0	1	1	**1**	1	1	1	1	0
1	0	1	0	0	0	0	1	**1**	0	1
1	1	0	0	0	0	0	1	1	**1**	1
...							...			

Negative examples

a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7
0	0	1	0	**0**	1	1	1	1	1	0
0	1	0	0	0	**0**	0	0	0	0	1
0	1	0	1	1	**0**	1	1	1	1	0
1	0	1	0	0	0	0	0	**0**	0	0
1	1	0	0	0	0	0	1	0	**0**	0
...							...			

Table 1. A part of the training set of examples

From these examples, the rule generation phase (Step 1) produced rules for the correct (POS-Class) and incorrect (NEG-Class) behavior of the multiplexer. The rules are shown in Table 2.

POS-Class if				
1.	(a0=1) & (a1=1) & (a2=0) & (d6=1)	or	(t:11,	u:6)
2.	(a0=0) & (a1=0) & (a2=1) & (d1=1)	or	(t:11,	u:5)
3.	(a0=1) & (a1=0) & (a2=1) & (d5=1)	or	(t:10,	u:6)
4.	(a0=1) & (a1=1) & (a2=1) & (d7=1)	or	(t:10,	u:4)
5.	(a0=1) & (a1=0) & (a2=0) & (d4=1)	or	(t:9,	u:5)
6.	(a1=1) & (d4=1) & (d6=1) & (d7=1)	or	(t:8,	u:1)
7.	(a1=1) & (a2=1) & (d3=1) & (d7=1)	or	(t:8,	u:1)
8.	(a0=0) & (a2=1) & (d1=1) & (d5=0)	or	(t:8,	u:1)
9.	(a2=0) & (d1=0) & (d2=1) & (d3=1)	or	(t:6,	u:2)
10.	(a0=0) & (a1=1) & (d3=1) & (d4=0)	or	(t:6,	u:2)
11.	(d0=0) & (d3=0) & (d4=1) & (d5=1)	or	(t:6,	u:1)
12.	(a2=1) & (d0=1) & (d1=0) & (d2=0) & (d5=1) & (d7=0)		(t:3,	u:1)
NEG-Class if				
1.	(a0=1) & (a1=1) & (a2=1) & (d7=0)	or	(t:13,	u:5)
2.	(a0=0) & (a2=0) & (d2=0)	or	(t:12,	u:7)
3.	(a2=0) & (d3=0) & (d4=0) & (d7=1)	or	(t:11,	u:2)
4.	(a0=0) & (a1=0) & (a2=1) & (d1=0)	or	(t:10,	u:8)
5.	(a0=1) & (a2=1) & (d5=0) & (d7=0)	or	(t:10,	u:2)
6.	(a1=0) & (a2=0) & (d1=1) & (d4=0)	or	(t:7,	u:4)
7.	(a0=1) & (a1=1) & (a2=0) & (d6=0)	or	(t:7,	u:3)
8.	(d0=0) & (d3=0) & (d5=0) & (d6=0)	or	(t:6,	u:1)
9.	(a0=0) & (a1=1) & (a2=1) & (d3=0)	or	(t:5,	u:5)
10.	(d1=0) & (d2=1) & (d3=0) & (d5=0)	or	(t:5,	u:1)
11.	(a0=1) & (a1=0) & (d5=0) & (d7=1)		(t:4,	u:4)

Table 2. Rules induced by AQ15 from examples

POS-Class and NEG-Class are hypotheses in the k-DNF form. Each rule in the hypotheses is accompanied with t-weights and u-weights that represent total and unique numbers of training examples covered by a rule.

For the above POS-Class hypothesis, the rules presented in Table 3 were chosen to constitute the candidate attribute **c0**, (Step 2): (here we present attribute generation for POS hypothesis only). Table 4 shows the definition of the new attribute c0.

1.	(a0=1) & (a1=1) & (a2=0) & (d6=1)	(t:11,	u:6)	t2u = 23
2.	(a0=0) & (a1=0) & (a2=1) & (d1=1)	(t:11,	u:5)	t2u = 21
3.	(a0=1) & (a1=0) & (a2=1) & (d5=1)	(t:10,	u:6)	t2u = 22
4.	(a0=1) & (a1=1) & (a2=1) & (d7=1)	(t:10,	u:4)	t2u = 18
5.	(a0=1) & (a1=0) & (a2=0) & (d4=1)	(t:9,	u:5)	t2u = 19

$\Sigma\ t2u_{best} = 103$; $\Sigma\ t2u_{all} = 166$; $[(\Sigma\ t2u_{best}) / (\Sigma\ t2u_{all})] = 0.62$

$\Sigma\ t2u_{best+1} = 113$; $\Sigma\ t2u_{all} = 166$; $[(\Sigma\ t2u_{best+1}) / (\Sigma\ t2u_{all})] = 0.68$

Table 3. Maximal rules according to the formula (1)

c0=1	if	(a0=1) & (a1=1) & (a2=0) & (d6=1) or (a0=0) & (a1=0) & (a2=1) & (d1=1) or (a0=1) & (a1=0) & (a2=1) & (d5=1) or (a0=1) & (a1=1) & (a2=1) & (d7=1) or (a0=1) & (a1=0) & (a2=0) & (d4=1)
c0=0		otherwise

Table 4. The definition of the c0 attribute

Table 5 shows the modified training set. For each old training example a new c0 attribute value has been added (Step 3). Table 6 presents rules generated from the modified training set (Step 4).

Positive examples

a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7	c0
0	0	1	0	**1**	0	0	0	0	0	0	**1**
0	1	0	0	0	**1**	0	0	0	0	1	**0**
0	1	0	1	1	**1**	1	1	1	1	0	**0**
1	0	1	0	0	0	0	1	**1**	0	1	**1**
1	1	0	0	0	0	0	1	1	**1**	1	**1**
...			...				...				

Negative examples

a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7	c0
0	0	1	0	**0**	1	1	1	1	1	0	**0**
0	1	0	0	0	**0**	0	0	0	0	1	**0**
0	1	0	1	1	**0**	1	1	1	1	0	**0**
1	0	1	0	0	0	0	0	0	**0**	0	**0**
1	1	0	0	0	0	0	1	0	**0**	0	**0**
...						...					...

Table 5. The part of the modified training set

POS-Class if

1.	(c0=1)	(t:51, u:45)
2.	(a0=0) & (a1=1) & (a2=1) & (d3=1)	(t:7, u:5)
3.	(a2=0) & (d1=0) & (d2=1) & (d3=1)	(t:6, u:2)
4.	(a0=0) & (a1=1) & (d2=1) & (d7=0)	(t:5, u:2)
5.	(a0=0) & (d0=1) & (d1=1) & (d2=1)	(t:5, u:1)

NEG-Class if

1.	(a0=1) & (c0=0)	(t:34, u:12)
2.	(a2=0) & (d2=0) & (c0=0)	(t:20, u:8)
3.	(a1=0) & (d5=0) & (c0=0)	(t:18, u:4)
4.	(d3=0) & (d5=1) & (d7=1) & (c0=0)	(t:13, u:4)
5.	(a2=1) & (d1=0) & (d2=1) & (d4=1) & (c0=0)	(t:10, u:2)
6.	(a2=1) & (d0=1) & (d1=0) & (d6=0) & (c0=0)	(t:7, u:2)

Table 6. Decision rules with the constructed attribute

As expected, the new attribute was used in the output hypothesis for both POS and NEG classes. We can observe that most of the training examples were uniquely covered by a rule (c0=1). Also, there is a new rule (2.) in the POS-Class hypothesis which is a part of the target concept. The ongoing research investigates ways of detecting and incorporating useful items into constructed attributes.

The final output hypothesis given by AQ17-HCI was tested against the testing set. The result was 85.60% accuracy (to be compared with 74% accuracy from rules generated by AQ15 without constructive abilities, e.g. rules obtained in Step 1).

4. Experiments with the method

A major measure of the performance of a learning algorithm is the classification accuracy of the learned concepts on the testing examples. The goal of our experiments was to test how well the algorithms do according to this criterion, and how well they compare to the other methods:

standard decision rule algorithm - AQ15, standard decision tree algorithm - REDWOOD, and algorithms with constructive abilities: FRINGE, GREEDY3, and GROVE (Pagallo and Haussler, 1989, 1990).

4.1 Experimental domains

The domains for testing AQ17-HCI and comparison with other methods were four Boolean functions: DNF3, DNF4, MX11, and PAR5. The same functions were used to test decision tree algorithms: REDWOOD (based on ID3) and FRINGE, and decision list algorithms: GREEDY3 and GROVE (Pagallo and Haussler, 1989, 1990).

DNF3 x1x2x6x8x25x28¬x29 + x2x9x14¬x16¬x22x25 + x1¬x4¬x19¬x22x27x28 +
¬x2¬x10x14¬x21¬x24 + x11x17x19x21¬x25 + ¬x1¬x4x13¬x25
(x3 x5 x7 x12 x15 x18 x20 x23 x26 x30 x31 x32) have random values for each example.

DNF4 x1x4x13x57¬x59 + x18¬x22¬x24 + x30¬x46x48¬x58 +
¬x9x12¬x38x55 + ¬x5x29¬x48 + x23x33x40x52 +
x4¬x26¬x38¬x52 + x6x11x36¬x55 + ¬x6¬x9¬x10x39¬x46 +
x3x4x21¬x37¬x57
(x2 x7 x8 x14 x15 x16 x17 x19 x20 x21 x25 x27 x28 x31 x32 x34 x35 x41 x42 x43 x44 x45 x47 x49 x50 x51 x53 x54 x56 x60 x61 x62 x63 x64) have random values for each example.

MX11 multiplexer-11 function (k=3) (Wilson, 1987).
For each positive integer *k*, there exists a multiplexer function defined on a set of $k + 2^k$ attributes or bits. The function can be defined by thinking of the first k attributes as *address bits* and the last attributes as *data bits* . The function has the value of the data bit indexed by the address bits[2].
(x12 .. x32) have random values for each example.

PAR5 parity-5 function.
For each positive integer *k*, there exists an even parity function defined on a set of *k* attributes. The function has value *true* on an observation if an even number of attributes are present, otherwise it has the value *false*.
(x6 .. x32) have random values for each example.

---

[2] In experiments with multiplexer function, Pagallo and Haussler (1989, 1990) classified an example as *positive* when the value of the function was *1* and negative for the value *0*. However, according to the definition, both values: *0* and *1* are valid values of the function. Thus, each multiplexer function needs an additional bit to indicate whether the value of the function was properly assigned. For the sake of comparability of the results of the HCI method with other methods (Pagallo and Haussler; 1989, 1990; VanDeVelde, 1989) we used the same, simpler multiplexer function. This function learns how to "switch on" or "set to 1" the addressed line.

In the Table 7, we provide a short description of the test domains. The number of training examples is set as specified in (Pagallo and Haussler, 1989).

Target concept	Number of attributes	Number of classes	Number of rules	Average rule length	Number of training examples
DNF 3	32	2	6	5.5	1650
DNF 4	64	2	10	4.1	2640
MX 11	32	2	8	4.0	1600
PAR 5	32	2	16	5.0	4000

Table 7. Target functions

4.2 Experimental results

Here we compare the performance of the AQ15 and AQ17-HCI programs. The rules generated by both programs were tested using the ATEST program (Reinke, 1984). ATEST views rules as expressions which, when applied to a vector of attribute values, will evaluate to a real number. This number is called the degree of consonance between the rule and the event. The method for arriving at the degree of consonance varies with the settings of the various ATEST parameters. Rule testing is summarized by grouping the results of testing all the events of a single class. This is done by establishing equivalence classes among the rules that were tested on those events. Each equivalence class (called a rank) contains rules whose degrees of consonance were within a specified tolerance (tau) of the highest degree of consonance for that rank. When ATEST summarizes the results it reports the percentage of 1st rank decisions (tau=0.02) as well as the percentage of only choice decisions (tau=0). In our experiments we used ATEST with its default parameters.

Target concept	Average % error			
	AQ15		HCI	
	%1st Rank	%Only Choice	%1st Rank	%Only Choice
DNF3	0.3	1.5	0.0	0.0
DNF4	0.2	11.5	0.0	0.0
MX11	0.0	0.0	0.0	0.0
PAR5	1.6	18.8	0.0	0.0

Table 8. The experimental results for different problems

Table 8 presents the average results (ten runs) for DNF3, DNF4, MX11, and PAR5 problems on the randomly generated training sets. The size of training sets was specified in Table 7. We used 2000 examples (independent from training examples) to test classification performance. Table 9 summarizes the average results for the DNF4 problem for different numbers of training examples.

Number of training examples	Average % error in learning target concept dnf4			
	AQ15		HCI	
	%1st Rank	%Only Choice	%1st Rank	%Only Choice
330	29.6	48.2	14.9	35.4
660	7.7	24.8	1.8	7.0
1320	1.8	16.4	0.0	0.0
1980	0.8	13.6	0.0	0.0
2640	0.2	11.4	0.0	0.0
3960	0.2	10.5	0.0	0.0

Table 9. The experimental results for different numbers of training examples in learning DNF4

In learning DNF functions, the HCI method outperformed AQ15 program in terms of performance accuracy. This result is due to better descriptors used in expressing learned concepts both in a learning phase (relations already discovered and stored under new attributes make it possible for a deeper search for dependencies among training data) and a testing phase (if a concise rule match an example this results in higher degree of consonance (Reinke, 1984)).

4.3 Empirical comparison of HCI with other methods

Target concept	Average % error					
	Decision TREES (*)		Decision LISTS (*)		Decision RULES	
	REDWOOD	FRINGE	GREEDY3	GROVE	AQ15	HCI
DNF3	7.4	0.3	0.6	1.4	0.3	0.0
DNF	24.9	0.0	0.0	7.8	0.2	0.0
MX11	13.1	0.0	0.5	3.9	0.0	0.0
PAR5	36.5	22.1	45.8	41.3	1.6	0.0

Table 10. The experimental results (*) from (Pagallo and Haussler, 1989, 1990)

Table 10 summarizes the results obtained in ten executions of the tested algorithms. The results for REDWOOD, FRINGE, GREEDY3, and GROVE algorithms come from (Pagallo and Haussler, 1989, 1990). AQ17 with hypothesis-driven constructive induction capabilities has learned all the target concepts. All concepts were learned for less than the assumed number of examples. It is worth mentioning that the standard decision rule system AQ15 uses the same form of adaptive features as those implemented in FRINGE, GREEDY3, and GROVE. This justifies comparable results obtained from those methods.

5. Conclusion

The presented HCI method of constructive induction generates new attributes on the basis of an analysis of the hypotheses, rather than by directly combining different attributes. This way the search for new attributes is very efficient, although is more limited in the repertoire of the attributes that can be constructed by direct, data-driven methods (Bloedorn and Michalski, 1991). In our experiments, the proposed method performed very favorably in comparison to methods employed in such programs as AQ15, REDWOOD, FRINGE, GREEDY3, and GROVE.

In the HCI method, new attributes correspond to subsets of best performing rules obtained in the previous iteration of the method. This is a real advantage of the method because it is not limited by the current implementation of the rule based system: the method can handle Boolean, nominal, linear, as well as structural attributes currently implemented in the AQ15 program.

The Hypothesis-driven Constructive Induction method proved to be a very effective way to improve the performance accuracy of learned rules. Also, the number of rules and their complexity (a number of conditions) has decreased. The algorithm detects irrelevant attributes among those used in a primary description of a problem as well as those introduced during attributes' generation process. Old and new attributes are examined according to classification abilities and new hypotheses are built based on the most relevant attributes.

On the other hand, the generated attributes are rather complex. In the future research, we plan to investigate a generation of attributes based on selected components of the best preforming rules. This could potentially lead to both a rapid improvement of the accuracy as well as to a greater simplification of the overall complexity of the hypotheses. We also plan to test the method on different types of learning problems in order to determine its strongest areas of applicability.

Figure 2. Learning curves for DNF4

References

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

Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K., "Occam's Razor," *Information Processing Letters*, 24, pp.377-380, 1987.

Cestnik, B., Kononenko, I., and Bratko, I., "ASSISTANT 86: A Knowledge Elicitation Tool for Sophisticated Users," *Proceedings of EWSL-87*, Bled, Yugoslavia, pp. 31-45, 1987.

Clark, P., and Niblett, T., "The CN2 Induction Algorithm," *Machine Learning*, 3, pp. 261-284, 1989.

Flann, N.S., and Dietterich, T.G., "Selecting Appropriate Representations for Learning from Examples," *Proceedings of AAAI-86*, Philadelphia, PA, pp. 460-466, 1986.

Langley, P, Bradshaw, G.L., and Simon, H.A., "Rediscovering Chemistry With the BACON System," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.), 1983.

Matheus, C., "Feature Construction: An Analytic Framework and Application to Decision Trees," *Ph.D. Thesis*, University of Illinois, 1989.

Michalski, R.S., and Larson, J.B., "Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: The Underlying Methodology and Description of Programs ESEL and AQ11," Rep. 867, Univ.of Illinois, 1978.

Michalski, R.S., "Pattern Recognition as Rule-Guided Inductive Inference," IEEE Transactions on Pattern Analysis and Machine Intelligence," *PAMI*, 2, No.,4, pp. 349-361, 1980.

Michalski, R.S., Mozetic, I., Hong, J., and Lavrac, N., "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains," *Proceedings of AAAI-86*, pp. 1041-1045, 1986.

Mitchell, T.M., Utgoff, P.E., and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Morgan Kaufmann, Los Altos, CA, 1983.

Muggleton, S., "Duce, and Oracle-Based Approach to Constructive Induction," *Proceedings of IJCAI-87*, pp.287-292, Morgan Kaufman, Milan, Italy, 1987.

Pagallo, G., and Haussler, D., "Two Algorithms that Learn DNF by Discovering Relevant Features," *Proceedings of the 6th International Workshop on Machine Learning*, Ithaca, pp. 119-123, 1989.

Pagallo, G., and Haussler, D., "Boolean Feature Discovery in Empirical Learning," *Machine Learning* **5**, pp. 71-99, 1990.

Quinlan, J.R., "Induction of Decision Trees," *Machine Learning* **1**, Kluwer Academic Pubs., pp. 81-106, 1986.

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

Rivest, R., "Learning Decision Lists," *Machine Learning* **2**, Kluwer Academic Pubs., pp. 229-246, 1987.

Schlimmer, J.C, "Concept Acquisition Through Representational Adjustment," *Machine Learning* **1**, pp.81-106, 1986.

Utgoff, P.E., "Shift of Bias for Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach Vol. II*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), Morgan Kaufmann, Los Altos, CA, pp. 107-148, 1986.

Van de Velde, W., "IDL, or Taming the Multiplexer," *Proceedings of the 4th EWSL-89*, France, pp. 211-225, 1989.

Wilson, S.W., Classifier systems and the animat problem, *Machine Learning*, 2, pp. 199-228, 1987.