DATA-DRIVEN CONSTRUCTIVE INDUCTION IN AQ17-PRE
A Method and Experiments

Eric Bloedorn
Center for Artificial Intelligence
George Mason University
Fairfax, VA 22030

Ryszard S. Michalski
Center for Artificial Intelligence
George Mason University
Fairfax, VA 22030

Abstract

This paper presents a method for constructive induction, in which new attributes are constructed as various functions of original attributes. Such a method is called data-driven constructive induction, because new attributes are derived from an analysis of the data (examples) rather than the generated rules. Attribute construction and rule generation is repeated until a termination condition, such as the satisfaction of a rule quality measure, is met. The first step of this method, the generation of new attributes has been implemented in AQ17-PRE. Initial experiments with AQ17-PRE have shown that it leads to an improvement of the learned rules both in terms of their simplicity as well as accuracy on testing examples.

1 Introduction

Most inductive learning programs perform a "selective" induction, that is, they generate descriptions (rules, decision trees, etc.) that involve only attributes initially provided in the examples. Thus, if the attributes used in the examples are of poor "quality", the learned descriptions may also be poor. It is possible, however, that although the original attributes may be of poor quality, there exist certain combinations or functions of these attributes that are highly relevant to the problem. This paper is concerned with the problem of discovering such relevant combinations of the original attributes. A method of data-driven constructive induction (DCI) is presented here, for the construction of new attributes through the application of various mathematical and logical operators to the initial attributes. The method uses ideas of "probabilistic adaptation" in determining the best constructed attributes, and has been partially implemented in AQ17-PRE.

There are a number of other programs which perform constructive induction. In INDUCE 3 [6] attribute construction is guided by background knowledge in the form of *lrules* and *arules*. Arules generate new descriptors as arithmetic expressions of the initially provided descriptors. Lrules define new descriptors as logical combinations of other descriptors. Our method is different as it is assumed that such rules are not available, and that the program must generate and test various functions of the original attributes.

Another form of constructive induction involves the derivation of new attributes from rules rather than data. Such a method is called hypothesis-driven constructive induction [13]. Constructive Induction is also being investigated with decision trees. Two such methods are FRINGE and GREEDY [10]. The FRINGE method searches for useful conjunctions by producing decision trees, assigning weights to those attributes that lead to the target concept, and then reformulating the training examples in terms of the highest ranking attributes. GREEDY is a separate-and-conquer method that recursively selects the best short term conjunct that covers a sizable portion of the examples.

NEWGEM [9], the rule generation program used by DCI, learns decision rules by performing inductive inference on examples. Training examples are vectors of attribute values. These attributes may be provided explicitly by the user, or they may be constructed by the program itself as directed by the user. These constructed attributes are combinations of given attributes and involve relations selected by the user. Available operations currently include addition, subtraction, multiplication, and the equality, greater-than, or less-than relations. The addition of a newly constructed attribute to the available *attribute set* (AS) is determined by the attribute quality function (AQF). Training examples are expressed as conjunctions of attribute values, and initial or induced decision rules are logical expressions in disjunctive normal form. The program performs a heuristic search through the space of logical expressions, until it finds a decision rule that is satisfied by all positive examples and no negative examples. This search is guided by a rule preference criterion. The program is based on the AQ algorithm for solving the general covering problem [4]. After generation of rules, attribute contributions to these rules is determined. Those contributing most are rewarded and are probabilistically more likely to be selected as the process of generate-and-test is repeated. A more detailed description of the method is given in section 3.

A drawback of programs that cannot construct new attributes is an inability to take advantage of some fairly simple relationships. For example, suppose there exist two sets of boxes, each box described by three attributes: height, length, and width. Sample data shown.

class1			**class2**		
height	length	width	height	length	width
2	12	2	12	4	2
6	4	2	4	12	2
3	8	2	8	6	2
4	4	3	4	8	3

The rule which describes the characteristic of each class of box when found by a non-attribute-constructing program, such as AQ15, is fairly complex

class1 :: [height= 2,3] v [height=4,6][length=4]
class2 :: [height=4,8] [length=6,8,12] v
[height= 12]

This complexity is due to the limits of the representation language. By generating new attributes the representation language is enriched and constructed rules are simpler and more accurate. Various combinations of attributes using a variety of operations are calculated. Useful combinations, which in this case involve multiplication, are kept. In the box example the area of the front face of the box, height*length was calculated and found to be useful. This area was then retained and combined with the width attribute to discover another useful attribute which we call volume. The rules produced using this constructed attribute are shown below.

class1 <:: [height * length * width = 48]
class2 <:: [height * length * width = 96]

2 A brief review of the AQ algorithm

Because the AQ algorithm is used in the inductive module of this method, for completeness, we provide a brief description of it. The AQ algorithm generates the minimum or near minimum number of general decision rules characterizing a set of instances, as originally described in [4,8].

1. A single positive example, called a seed, is selected and a set of most general conjunctive descriptions of this example is computed (such a set is called a star for the seed). Each of these descriptions must exclude all negative examples.

2. Using a description preference criterion a single description is selected from the star, called the 'best' description. If this description covers all positive examples, then the algorithm stops.

3. Otherwise a new seed is selected among the unexplained (uncovered) examples, and steps 1 and 2 are repeated until all examples are covered.

The disjunction of the descriptions selected in each step constitutes a complete, consistent and general description of all examples. The preference criterion used in selecting a description from a star is expressed as a list of elementary criteria that are applied lexiographically and with a certain tolerance. The criteria may be simplicity of description (measured by the number of variables used), cost (the sum of the given costs of the individual variables), or other criteria [7].

The description of a class is expressed using the variable-valued logic system 1 (VL_1), which is a multiple-valued logic propositional calculus with typed variables [5]. A class description is called a cover. A cover of a concept is a disjunction of complexes describing all positive examples and none of the negative examples. A complex is a conjunction of selectors, which is the simplest statement in VL_1. A selector relates a variable to a value or a disjunction of values, for example [temperature = cold], or [x < 5]. The general form of a selector is:

[L # R]

where L, called the referee, is an attribute, and R, called the referent is a set of values in the domain of the attribute in L, # is a relational symbol which can be one of the following: =, <,>,>=,<=,<>.

3. Description of the DCI Method

The construction of new derived attributes through the mechanism of constructive induction is done in the following way.

1. Identify all linear type attributes.
2. Repeat steps 3 through 5 for each possible attribute combination.
3. Repeat steps 4 and 5 for each operator.
4. Calculate the values of this attribute pair for the given operator.
5. Evaluate the discriminatory power of this newly constructed attribute using the Attribute Quality Function (AQF) described below. If the attribute is above some threshold then store it, else discard it.
6. Exit.

For now, only linear attribute types are used because most of the operators available operate only on linear types. However, in the future, binary nominal attributes will also be considered, to be operated on by equality and disjunction. Linear type attributes have a finite number of discrete ordered values. Nominal type attributes have finite number of discrete unordered values. Continuous-valued data is not acceptable to the version of AQ algorithm being used here, but a continuous version is being investigated by the authors as well as others (for example CAQ). [12]

The sum of binary attributes, when those attributes signal the presence or non-presence of a feature, can be described as "if x of the following y features exist". Such an attribute could possibly be quite useful in a medical domain where these binary attributes signal the presence of a symptom or disease.

After selecting the possible candidates, the algorithm generates every pairwise combination of attribute and operation. After each new attribute's values are calculated, an evaluation function, AQF, is used to judge its quality before adding it to the AS. The AQF used is shown below:

$$\frac{\text{(\#Positive events - W(\#negative events))}}{\text{Number of events in selector class}}$$

Event values are considered *positive* when the common value (see below) is found within the original class. Event values are considered *negative* when the common value is found in classes other than the original class. The weighting value (W) is currently set to 1, but can easily be changed to reflect the relative importance of positive versus negative coverage. The number of events in selector class is the total number of events in the original class. A perfect discriminatory attribute, which alone discriminates one class from all the other classes, will have an AQF value of 1. Possible AQF values range from negative (total #of events in other classes-1) to one.

The AQF is calculated for each class, for each attribute. First the most common value for an attribute in a class is found, i.e. mode in statistics (this is #Pos in the formula above). The mode is the value which most often appears in the events for that class. The mode was chosen because it is likely to produce the highest AQF value for the attribute (because of its high #Pos value). To demonstrate how the AQF is calculated, let us return to the volume example given earlier. The previous volume data is now shown supplemented with a constructed attribute height*length.

class1

height	length	width	height*length
2	12	2	24
6	4	2	24
3	8	2	24
4	4	3	16

class2

height	length	width	height*length
12	4	2	48
4	12	2	48
8	6	2	48
4	8	3	48

The mode of constructed attribute height*length in class1 is 24 because 24 has a commonality of 3, and 16 has a commonality of only 1.

If the newly constructed attribute exceeds the user-defined threshold, the attribute is added to the attribute set (AS). The AS initially consists of only the original attributes supplied by the user. This process of attribute calculation, testing and addition is repeated for all the possible combinations. An adjustable program parameter is used to help control the number of attributes constructed. No new attributes will be constructed after this threshold is reached. A better way to control attribute construction, however, is by setting high quality thresholds. Figure 1 gives a functional description of the algorithm.

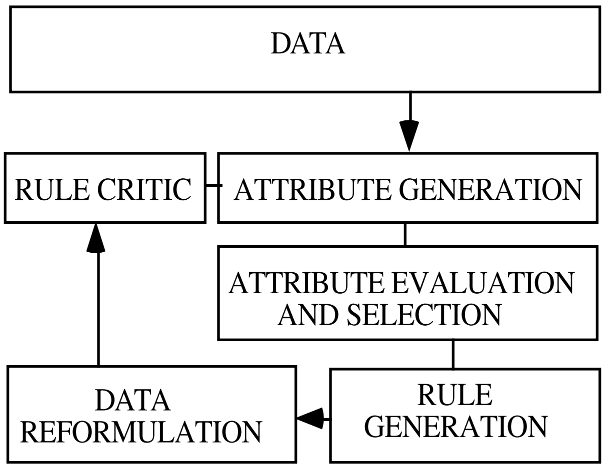


Figure 1. A functional diagram of the method.

The attributes selected for the subset are chosen probabilistically based on their importance in describing the training examples. This importance is measured by attribute weight (AW). Those attributes with a high AW are more likely, but not guaranteed, to be selected for the AS. Initially AW is equal to the AQF values. After rules are generated, the AW are adjusted based on the presence of the attribute in the descriptions. Those attributes present in rule descriptions are weighted positively, while those not present are unchanged. AW weights are also kept for attributes not in the AS. Unselected attributes are increased by an amount equal to the average increase of the attributes present in the AS. This is done to prevent unselected attributes from being punished and never being selected to the AS. At the same time this general increase does punish those attributes which were in the AS, but were found not to be useful. The pre-generation of new attributes, followed by AQF testing, selection to the AS and AW adjustment is repeated until the decision rules produced satisfy a rule critic.

Because the newly generated attributes are combinations of original attributes they are more complicated than the original attributes. The cost of these attributes should reflect this added complexity. To do this each operation and relation has been assigned a cost. These costs were determined from an overall ranking by the authors of selector complexity. The actual values are not meaningful, but are only meant to reflect relative complexity. These values can be changed by the user if desired. The current default values for these costs is shown in Table 1.

operation	**cost**
equal (=)	1
addition (+)	5
subtraction (-)	5
greater than or equal (>=)	7
less than or equal (<=)	7
multiplication (*)	9

Table 1. Operator costs

Another factor that should be considered when calculating cost is the number of values a function takes as its result. For example, the selector [x1 = 5] is less expensive than a selector of the form [x7 = 101 v 123] because the former has only one value in the referent. This relation is cognitively more difficult to understand and is assigned a higher cost. The number of values in the referent is indirectly controlled by the depth parameter. The depth parameter controls whether a single value, or multiple values will be used when calculating quality. Increasing the depth parameter allows more general concepts to be introduced. The AQ algorithm is capable of generalizing discrete points to ranges. The depth parameter simply allows more general concepts to be introduced. The maximum value of depth is currently set to 2. This maximum is due to the empirical finding that increasing the depth quickly decreases quality and speed. The total cost of the new attribute is a product of the sum of the individual attributes' costs, the cost of the relation and the depth.

4 Experimental Results

The presented method of attribute construction is illustrated by experiments. In these experiments only one iteration of attribute generation and AQF evaluation was completed. These experiments involve image data used by Bala and Pachowicz [1] in their work. All texture images were take from the Brodatz album of images [2]. Input grey-level images were processed by the well-known texture feature extraction technique of

Laws' masks [3] for texture feature extraction. A vector of eight features was extracted for a single pixel, and for each method of feature extraction.

In the first experiment the learning set contained five classes of texture images, each with 50 events and described by eight original attributes. From these eight original an additional 46 attributes were constructed from the '>=' and the '=' binary operators.

In the second and third experiments the learning set contained twelve classes of 100 events, and was described by eight original attributes. From these eight original, an additional 37 attributes were constructed in the second experiment, and an additional 43 were constructed in the third experiment. In both experiments the attributes finally retained consisted of functions of the '>= ' and '=' binary operators. The same testing set was used for all three experiments, which consisted of 12 preclassified texture images classes, each with 200 examples. To test the rules which included constructed attributes the original eight attributes of the testing set had to be expanded to include the newly constructed attributes. This expansion depended on the attributes discovered in the learning data. These same attributes were then calculated for the testing set.

The results of these experiments are shown in Table 2. Three sets of learning data were presented to NEWGEM, a learning program in the AQ family that does not have the ability of constructive induction [9], and then to AQ17-PRE, a program that partially implements the method presented in this paper. The overall first rank correct classification for AQ17-PRE for the three experiments was 78%, compared to 75% for NEWGEM. The average overall recognition was 100% for AQ17-PRE and 98% for NEWGEM.

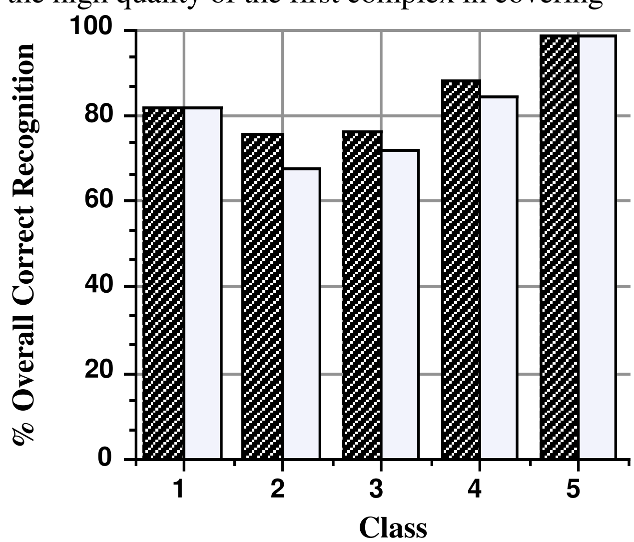
Exp.	#1		#2		#3	
	NG	AQ	NG	AQ	NG	AQ
Overall % first rank correct recog.	81	84	74	76	72	75
Overall % correct recog.	98	99	99	100	97	100

AQ : aq17-Pre NG: Newgem

Table 2. Comparison of rules produced by NEWGEM (a) and AQ17-Pre (b).

The values shown in Table 2 were calculated by a testing tool called ATEST [11]. In ATEST rule performance is measured by the degree of agreement between a class description rule and a testing example from an assigned class. ATEST views rules as expressions when comparing them to a vector of attributes (e.g., a testing example). The result of this evaluation is a real number which is the degree of consonance between the conditional part of the rule and the event. The overall percent first rank value is the percentage of events which matched the correct description rule greater than any other rule. The overall percent value is the percentage of events whose consonance value was within a threshold margin t, of the highest matching rule.

While the performance of the rules was higher for AQ17-PRE, than for NEWGEM, the rules produced by AQ17-PRE were also shorter. The total number of complexes needed to describe all the texture classes for the texture description was fewer for AQ17-PRE. For the first experiment the average reduction, for *each class*, was 1, while the average per class reduction for the second and third experiments was 2 complexes for *each class*. In addition to having fewer complexes, another advantage of the rules produced by AQ17-PRE is the high quality of the first complex in covering the example space. As mentioned earlier, the rules produced by AQ17-PRE and NEWGEM are in the

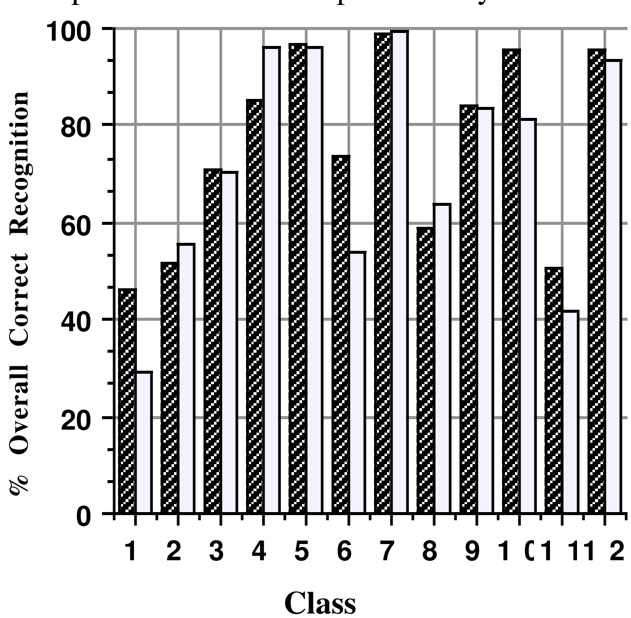


Graph 1. Recognition results from experiment 1 form of complexes. Both programs associate with every complex a t-weight representing the total number of covered events. An event is considered covered when the complex is satisfied by an event.The t-weight may then be interpreted as the representativeness of a complex as a concept description. The highest t-weight complexes in the covers produced by AQ17-PRE were greater than the highest t-weight complexes in the rules produced by NEWGEM. The average increase in the t-weight from the rules produced by NEWGEM to those produced by AQ17-PRE, was 4.0, for the first experiment 5.25 for the second, and 7.25 for the third experiment. The rules produced by AQ17-PRE, therefore, contain fewer disjuncts and contain complexes that are more representative of the concept being described, than those produced by NEWGEM.

Graphs 1 through 3 show the recognition testing results of the three experiments by class. The results for rules produced by AQ17-PRE are shown by a darkly shaded bar, and the results for rules produced by NEWGEM are shown by a lightly shaded bar. There were 200 testing events per class. The height of a bar represents the percent of testing events correctly classified.

5 Conclusion

The proposed method of pre-generative constructive induction partially implemented in AQ17-PRE, performed well on the problems in the experiments. The rules produced by the

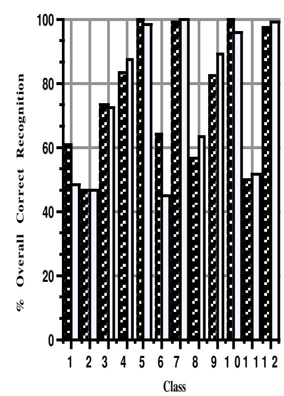


Graph 2. Recognition results from experiment 2.

method, were superior to the rules produced by using only the original attributes. The method of pre-generative construction of attributes described in this paper has shown itself to be a useful process. In the domain of texture recognition the rules produced after a pre-generation process and that contained new attributes were found to be superior to the rules described by only the original attributes in terms of recognition on testing, complexity of rule, and quality of individual covers.

The ideas proposed here can be improved in many ways. This approach only constructs new attributes which are pairs of original attributes. Furthermore the method is exhaustive, trying all possible combinations every time the program is run. A more sophisticated approach which first detects the need for construction and also bases attribute performance on a training set weighted according to difficulty is being investigated. These event weights reward new attributes which cover 'hard' concepts and attempt to prevent the construction of new attributes which cover events which are already well described.

Another problem for future work is to use domain knowledge to guide the process of applying operators in generating new attributes. This background knowledge could be in the form of a mathematical function of the original attributes. Such a feature would allow the program to generate very advanced and complex



Graph 3. Recognition results from experiment 3.

features much more efficiently. This will supplement the present generate and test procedure the program currently uses. It is also planned to apply the method to domains other than textures, to test its generality.

Acknowledgements

The authors wish to thank Michael Hieb, Ken Kaufman and Janusz Wnek for their ideas and critical review of this paper.

This research was conducted in the Center for Artificial Intelligence at George Mason University. Research of the Center for Artificial Intelligence is supported in part by the Defense Advanced Research Projects Agency under grant, administered by the Office of Naval Research, No. N00014-87-K-0874, in part by the Office of Naval Research under grant No. N00014-88-K-0226, in part by the Office of Naval Research under grant No. N00014-88-K-0397.

References

[1] J.W. Bala and P.W. Pachowicz, "Recognizing Noisy Patterns of Texture via Iterative Optimization and Matching of their Rule Description", MLI Report 90-12, Center for Artificial Intelligence, George Mason University, Fairfax, Va. 1990.

[2] Brodatz P., Textures: A Photographic Album for Artists and Designers, New York, 1966.

[3] K.I. Laws, "Textured Image Segmentation", PhD Thesis, Dept. of Electrical Engineering, University of Southern California, Los Angeles, 1980.

[4] R.S. Michalski, “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.

[5] R.S. Michalski, “Variable-Valued Logic: System VL_1, Proceedings of the 1974 International Symposium on Multiple-Valued Logic, pp. 323-346. West Virginia University, Morgantown, 1974.

[6] W.A. Hoff, R.S. Michalski, and R. E. Stepp, “INDUCE 3: A Program for Learning Structural Descriptions from Examples,” ISG 86-9, Dept. of Computer Science, University of Illinois, Urbana, 1986.

[7] R.S. Michalski and J.B. Larson, “Selection of Most Representative Training Examples and Incremental Generation of VL_1 Hypotheses: the underlying methodology and the description of programs ESEL and AQ11, “ Report No. 867, Dept. of Computer Science, University of Illinois, Urbana, 1978.

[8] R.S. Michalski and B.H. McCormick, “Interval Generalization of Switching Theory.” Report No. 442, Dept. of Computer Science, University of Illinois, Urbana. 1971.

[9] I. Mozetic, "NEWGEM: Program for Learning from Examples, Program Documentation and User's Guide", Report No. UIUCDCS-F-85-949, Department of Computer Science, University of Illinois at Urbana-Champaign, 1985.

[10] G. Pagallo and D. Haussler, "Feature Discovery in Empirical Learning", Technical report UCSC-CRL-88-08, Dept. of Comp. Science, Univ. of California, Santa Cruz, 1988.

[11] R.E. Reinke, “Knowledge Acquisition and Refinement Tools for the ADVISE Meta-expert System,” Master’s Thesis, University of Illinois, 1984.

[12] B. Whitehall, S. Lu, and R. Stepp, "CAQ: A Machine Learning Tool for Engineering." International Journal for Artificial Intelligence in Engineering, Vol. 5, 1990.

[13] J. Wnek and R.S. Michalski, “Hypothesis-Driven Constructive Induction”, MLI-Report, Center for Artificial Intelligence, George Mason University, 1991.