DATA-DRIVEN CONSTRUCTIVE INDUCTION

by

E. Bloedorn
R. S. Michalski

IEEE Intelligent Systems, Special issue on Feature Transformation and Subset Selection, pp. 30-37, March/April, 1998.

Data-Driven Constructive Induction

Eric Bloedorn, Mitre Corporation and George Mason University
Ryszard S. Michalski, George Mason University and Polish Academy of Sciences

AN INDUCTIVE LEARNING PROGRAM'S ABILITY TO FIND AN ACCURATE HYPOTHESIS CAN DEPEND ON THE QUALITY OF THE REPRESENTATION SPACE. THE AUTHORS DEVELOPED A DATA-DRIVEN CONSTRUCTIVE-INDUCTION METHOD THAT USES MULTIPLE OPERATORS TO IMPROVE THE REPRESENTATION SPACE. THEY APPLIED IT TO TWO REAL-WORLD PROBLEMS.

INDUCTIVE-LEARNING ALGORITHMS are powerful tools for identifying meaningful patterns in large volumes of data, and their use is increasing in fields such as data mining and computer vision. However, conventional inductive-learning algorithms are *selective*—they rely on existing, user—provided data to build their descriptions. Thus, data analysts must assume the important and sizeable task of determining relevant attributes. If they provide inadequate attributes for describing the training examples, the descriptions the program creates are likely to be excessively complex and inaccurate.

Attributes can be inadequate for the learning task when they are weakly or indirectly relevant, conditionally relevant, or inappropriately measured. *Constructive induction* is a general approach for coping with inadequate attributes found in original data. It uses two intertwined searches—one for the best representation space, the other for the best hypothesis within that space[1]—to formulate a generalized description of examples.

Originally, constructive induction focused on improving the representation space by generating additional task-relevant attributes.[2] It was subsequently observed that this was only one way of modifying the space. Attribute construction is a form of representation space expansion; attribute selection and attribute value abstraction are forms of representation space destruction. Furthermore, it became clear that this improvement of the representation space by expansion and destruction could have a profound impact on the simplicity and predictive accuracy of concepts induced from that space. The better the representation space, the easier it is for the program to learn. It is thus important not only to add relevant attributes, but also to remove irrelevant ones and find a useful level of precision for the attribute values.

Constructive-induction methods are classified according to the information used to search for the best representation space:

- data-driven constructive induction (DCI) uses input examples,
- hypothesis-driven constructive induc (HCI) uses intermediate hypotheses,
- knowledge-driven constructive induc (KCI) uses domain knowledge provi by an expert.[1]

In *multistrategy constructive induction* (M two or more of these methods are used.[3]

This expanded definition of construct induction guided our development of seve constructive induction programs: AQ17-D AQ17-HCI, and AQ17-MCI. These all use AQ-type rule-learning algorithm for cond ing hypothesis search, hence the "AQ" pre Here we describe our latest methodology the data-driven constructive induction, imp mented in AQ17-DCI. Our methodology co

1094-7167/98/$10.00 © 1998 IEEE

Constructive induction meets the "Monk"

We distinguish between two types of representation spaces: a concept (or hypothesis) space and an example space. In a concept representation space, the program searches for a hypothesis that generalizes from the training examples. This space is spanned over the exact descriptors (attributes, predicates, transformations) used in the hypothesis description. An example representation space is spanned over attributes that occur in the training examples. In conventional machine-learning methods, the concept and example representation spaces are identical.

In some inductive-learning problems, the concept boundaries in the example representation space are very complex and thus difficult to learn. Figure A1 shows such a problem: the "second Monk's problem," which was used in an international competition of machine-learning programs.[1] In the diagram—spanned over six attributes x1, x2, ..., x6—individual cells represent unique combinations of different attribute values. Positive training examples are marked by "+" and negative training examples are marked by "–." The shaded areas represent the target concept the program needs to learn.

As you can see, the target concept's boundaries are very complex; it will be difficult for the program to learn the concept in this representation space. For this reason, conventional symbolic learning methods—such as decision-tree learning—did not perform well on this problem.

Figure A2 shows an improved example representation space, which AQ17-DCI derived using our data-driven constructive-induction method. In this space, training examples are consolidated and the target concept is highly regular. The concept is thus easy to learn. The improved space can now be used as the concept representation space.

To find the improved space shown in Figure A2, AQ17-DCI generated and evaluated multiple user-selected operators against the set of available attributes.

One of the general operators available in AQ17-DCI—and the one that happened to be most useful for this problem—is an operator that generates *counting attributes*: #Attr(S, C). Such attributes measure the number of attributes in a set S that satisfy some selector C. As currently implemented, the program determines membership in S based either on user-provided attribute units or by simply using all available attributes. The latter proved useful here. The program generated a number of new attributes and found that the counting attribute #Attr({x1,..., x6}, Firstvalue) is highly relevant for this problem (the selector C=Firstvalue means that an attribute takes the first value of its domain for a given object). In the hypothesis-determination phase, AQ17-DCI found the following consistent and complete description of all the examples:

```
#Attr({x1, ..., x6}, Firstvalue) = 2,
```

which can be paraphrased: *an example belongs to the concept if exactly two of six attributes take their first value.* It turned out that this rule exactly represents the target concept, and thus has a predictive accuracy of 100%.

Reference

1. S.B. Thrun et al., *The Monk's Problems: A Performance Comparison of Different Learning Algorithms* (revised version), Tech. Report CMU-CS-91-197, Carnegie Mellon Univ., Pittsburgh, 1991.

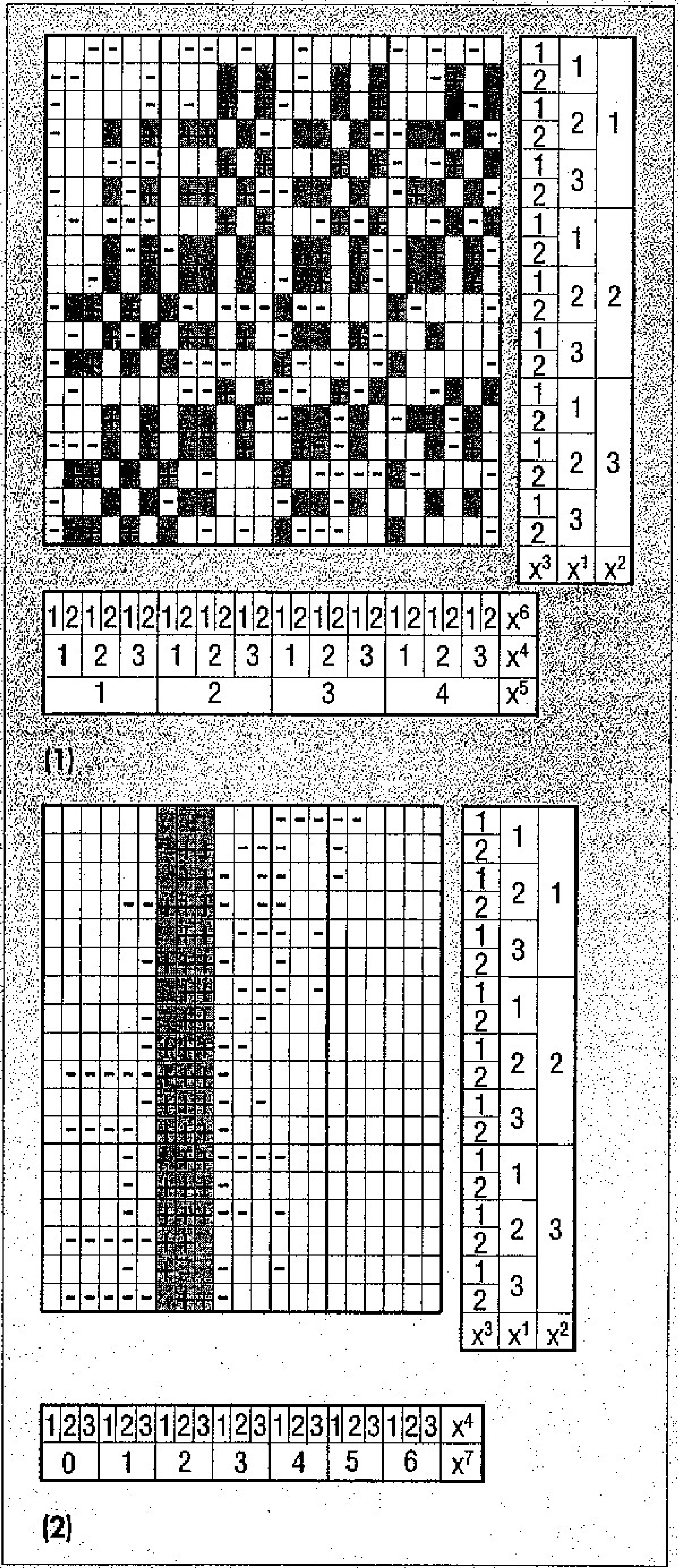

Figure A. Diagrammatic visualization of the Monk 2 representation spaces: (1) the example representation space; (2) the concept representation space derived from the example space by the data-driven constructive-induction method.

bines the AQ-15c learning algorithm with a range of operators for improving the representation space. These operators are classified into *constructors* and *destructors*. Constructors extend the representation space using attribute-generation methods and destructors reduce the space using attribution-selection methods and attribute abstraction. We integrated these operators—which are usually considered separately—into AQ17-DCI in a synergistic fashion. We tested the method on two real-world problems: text categorization and natural scene interpretation.

The power of a constructive-induction approach is illustrated by an example from the "second Monk's problem," which was used in an international competition of machine-learning programs.[4] The sidebar, "Constructive induction meets the 'Monk,'" describes how AQ17-DCI derived an improved representation space for this problem.

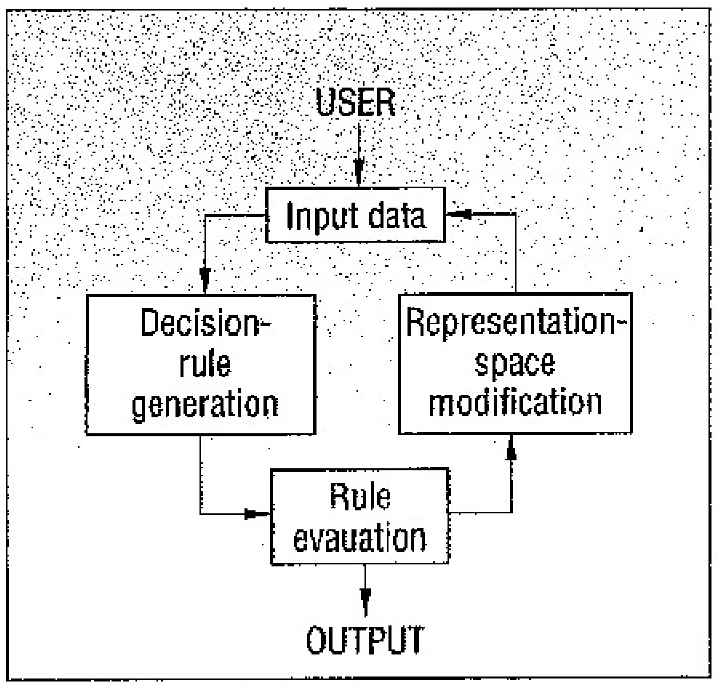

Figure 1. A general schema for constructive induction.

What makes a learning problem difficult?

Learning problems can be difficult for many reasons, including inadequate representation space or description language, or errors in training examples. Our constructive-induction methodology addresses some of the problems posed by an inadequate representation space. Specifically, it solves problems caused by attributes that are indirectly or weakly relevant, or overly precise.

A question of relevance. Attributes are indirectly relevant if their connection to a given classification task depends on their interaction with one or more other attributes. The difficulty of describing such interactions depends on the learning algorithm's description language. For most symbolic inductive-learning algorithms, interactions involving logical conjunction or disjunction are easy to describe. However, interactions such as the equality or the arithmetic product of attributes can be very difficult for such methods. Even more difficult to capture are interactions represented by complex equations involving multiple attributes. An example of a multiattribute interaction is the even/odd parity classification problem (classifying binary strings on the basis of the parity of the number they represent). When there is complex multiattribute interaction, you can use attribute-construction methods that combine attributes in a problem-relevant manner.[5,6]

You can detect or remove irrelevant attributes using either a filter or wrapper approach.[7] In the filter approach, the program selects attributes prior to induction. Because filters are used separately, they work fast and can be used with any induction algorithm. They also work on large data sets. The wrapper approach uses the induction algorithm itself to estimate attribute relevance; we view it as a special case of hypothesis-driven constructive induction.

Overprecision. Attribute overprecision leads to an unnecessarily large representation space, making it difficult for the program to find the correct hypothesis. In this situation, every training example is distant from even its closest neighbors. Because of this sparseness, the learning algorithm can draw too many concept boundaries for dividing examples into regions. Attribute overprecision frequently occurs when attributes are continuous. To avoid a potential problem, the attribute domain can be split into ranges of values.[8] Formally, such a discretization essentially performs an abstraction operation on the example the attribute describes. (We use the concept of *abstraction* as defined in the Inferential Theory of Learning.[9] ITL identifies abstraction as any knowledge transmutation that reduces the amount of detail in the reference set description). However, abstraction is often done without considering other operators that can change the representation space. Our methodology combines the effect of abstraction with other representation-space modification operators. As we'll show with an example from natural-scene interpretation, operator combinations can significantly improve representation space for learning.

Constructive Induction: a general schema

Figure 1 shows a general scheme for constructive induction and illustrates the intertwined searches for best representation space and the best hypothesis therein. For initial input, we provide a set of training examples and a characterization of the representation space. This characterization includes a description of attributes, their types, and their domains. The program then splits the training data into a primary and a secondary data set; the primary set is passed to the Decision Rule Generation module, which generates initial concept descriptions (in our case, in the form of decision rules).

Next, the program evaluates these rules in terms of complexity and accuracy in example classification. If the descriptions satisfy a description-quality threshold, the program stops the learning process; if not, it moves to the Representation Space Modification (RSM) module. This module creates a new representation space for the input data. The process then repeats in cycles until the learned rules are satisfactory or the program has tried all planned representation-space modifications. The program evaluates the final rules on the testing examples to determine a more precise estimate of their performance accuracy.

AQ17-DCI searches for hypotheses using the AQ algorithm as implemented in AQ15c.[10] AQ15c performs a separate-and-conquer strategy to determine a decision rule set that covers all the positive examples and none of the negative examples (in the default case). It begins the search by randomly selecting a "seed" example of a class (concept) and applying the extension-against generalization operator to determine a general rule set (a star) that covers the seed but not the negative examples. It then selects the best rule from the star (according to a multicriterion evaluation function) and marks examples covered by this rule. Next, the program selects a new seed from among unmarked examples and repeats the process until all examples in the given class are marked. It repeats a similar process for other classes until it has rule sets for each class. The hypotheses the program produces in this way can be further improved by rule truncation;[11] the end result is a decision rule set for each class in the data. Here is an example of a rule produced by AQ:

```
Class1 <= [color = blue]
&[height > 5"] & [shape =
square or triangle]

or [height > 10"] & [shape
= square]
```

This rule states that an object is in class1 if it is blue, its height is greater than five inches, and its shape is square or triangle; or if its height is greater than 10 inches and its shape is square.

This example illustrates two important features of the program. First, selectors (conditions) of a rule can include internal disjunction of attribute values (the shape can be square or triangle). Second, rules for a given class can logically intersect. These and other features make the AQ learning method more expressive than other learning methods.

Evaluation. Each time AQ generates a new rule set, it estimates its predictive accuracy using the secondary training set (if there are few training examples, it can also use a cross-validation method). The program calculates predictive accuracy as the percentage of sec-

Table 1. Data-driven representation-space expansion operators used in AQ17-DCI.

Operator	Arguments	Notation	Interpretation
Equivalence	Attributes *x,y*	$x = y$	If $x = y$ then 1, otherwise 0
Greater than	Attributes *x,y*	$x > y$	If $x = y$ then 1, otherwise 0
Greater than or equal to	Attributes *x,y*	$x >= y$	If $x \geq y$ then 1, otherwise 0
Addition	Attributes *x,y*	$x + y$	Sum of *x* and *y*
Subtraction	Attributes *x,y*	$x - y$	Difference between *x* and *y*
Difference	Attributes *x,y*	$\|x - y\|$	Absolute difference between *x* and *y*
Multiplication	Attributes *x,y*	$x * y$	Product of *x* and *y*
Division	Attributes *x,y*	x/y	Quotient of *x* divided by *y*
Maximum	Attribute set S	Max(S)	Maximum value in set S
Minimum	Attribute set S	Min(S)	Minimum value in set S
Average	Attribute set S	Ave(S)	Average of values in set S
Counting	Attribute set S,C	#Attr(S,C)	No. of attributes in S satisfying C

ondary training examples correctly classified. It evaluates rule set complexity by counting the total number of rules and selectors.

Rule-set quality is evaluated lexicographically: the program first compares the rule set to the user-defined accuracy criterion. If the rule set meets the accuracy goals, it is evaluated according to the complexity criterion. If the rule set does not meet accuracy standards, the program rejects it and stops processing. This lexicographic evaluation is very efficient, lets users define the minimum allowable accuracy, and is used in other AQ program operations.

Representation-space modification. In the RSM module, the program determines which modification operator to apply at a given stage and changes the training and testing examples. In AQ17-DCI, the user can select which of the RSM operators should be tried (attribute construction, selection, or abstraction) or can accept a default setting that applies all operators. In the *all-operators* mode, operators are applied in a predefined order: attribute selection, attribute abstraction, and then attribute construction. (The AQ17-MCI[3] method extends this by including additional RSM operators and by using metarules that relate problem characteristics to the appropriate operators.)

Our data-driven attribute-construction process uses an exhaustive generate-and-test approach. As Table 1 shows, our standard set provides many different operators for constructing new attributes. These operators include both binary and multiargument operators (functions). In the binary group, we've implemented the relational operator (determining whether the first input is less than, greater than, or equal to the second input) and several mathematical operators, including addition, subtraction, absolute difference, multiplication, and integer division. New operators can easily be added to these; our aim was to provide simple, generally applicable operators that would be easy to generate and interpret.

In constructing attributes, the program generates and evaluates each possible attribute–operator combination. New attributes must exceed a user-defined minimum discriminatory power (as calculated by the information gain ratio) and not exceed a user-defined cost threshold. A new attribute's cost is the sum of the weights given to the original attributes used to define it, plus the program-defined weight associated with each operator. Binary operators select attributes in pairs, and multiargument operators use unit information to determine set membership. Both attributes must satisfy type (such as ordered attribute values for addition) as well as operator-specific unit constraints (such as matching attribute units for subtraction). These constraints are useful in reducing the combinations generated and insuring that new attributes are meaningful. Users can also limit the total number of newly constructed attributes that will be added to the representation space.

You can select attributes by applying one of the many existing attribute-selection criteria. In AQ17-DCI, we use the information gain ratio. It selects for future processing the attributes with a gain ratio greater than or equal to a threshold we predefine.

For attribute abstraction, we chose the ChiMerge algorithm to create ranges of attribute values.[12] The ChiMerge is a bottom-up algorithm that stores values in separate intervals and then merges them into ranges until a termination selector is met. The interval-merging process consists of two continuously repeating steps:

1. compute χ^2 values (correlation between the class-attribute value and an individual attribute value) and
2. merge the pair of adjacent intervals with the lowest χ^2 value.

Intervals are merged until all pairs of intervals have χ^2 values exceeding the user-defined χ-threshold. The chi-threshold can be determined from a table; it is a function of the desired significance level and number of degrees of freedom (one less than the number of classes). The χ^2 value measures the probability that the attribute interval and class value are independent. If the interval has a χ^2 value greater than threshold, class and interval are correlated and should be retained. High χ^2 threshold settings cause more interval mergers, resulting in fewer total intervals, or attribute values. We have found a χ^2 threshold of 0.9 (values range from 0.1 to 1.0) to be a good default.

Real-world experiments

Our methodology recognizes that attribute construction, selection, and abstraction all serve the same purpose: to improve the representation space. How effective is it to integrate these factors? To answer this question, we applied our methodology to two real-world problems: text categorization and natural-scene interpretation.

Text categorization. In this problem, the task is to classify segments of text (usually documents) into the best single class selected from a class set. In our case, the task was to classify incoming newswire text as either of interest or not of interest to a given user. Our user had an interest in "Medicine in the United States." He provided feedback on 38 news articles drawn from 442 articles run in the *Colorado Springs Gazette Telegraph* over a one-month period. Of 38 articles, the user found 18 relevant and 20 irrelevant.

The goal of constructive induction is to learn a description of the user's interest-(profile) based user feedback. This feedback consists of labels for a (usually small) set of documents describing the user's (binary) interest.

One of the most difficult aspects of text categorization is finding a good text representation. Our constructive-induction method builds on previous work,[13] which identified a hybrid text representation—consisting of extracted "subjects," POL (person, organization, and location) attributes, and keywords—coupled with a generalization hierarchy as superior for this task.

Table 2. Attributes used for describing text.

Attributes	Description
x1...x5	Top five subject categories as computed by the SFC text classifier.
x6...x59	POL people tags as computed by the IDD POL tagger. For each person identified, the vector contains the following string attributes: [name, gender, honorific, title, occupation, age]. Nine people (each with these subfields) are identified for each article.
x60...x104	POL organization tags as computed by the IDD POL tagger. For each organization identified, the vector contains the following string attributes: [name, type, acronym, country, business]. Nine organizations (each with these subfields) are identified for each article.
x105...x140	POL location tags as computed by the IDD POL tagger. For each location identified, the vector contains the following string attributes: [name, type, country, state]. Nine locations (each with these subfields) are identified for each article.
x141...x141+n	The top *n* ranked keyword terms as selected based on a tf.idf measure.

Table 3. Comparison of predictive accuracy on Complete attribute set

Attribute set	Method	Average predictive accuracy
Complete	AQ-only	54.2 ± 6.9%
	AQ+DCI-Generate	67.4 ± 7.6%
	AQ+DCI-Select	70.1 ± 5.8%

Table 4. Comparison of predictive accuracy on Subject attribute set.

Attribute set	Method	Average predictive accuracy
Subjects	AQ	78.0 ± 6.6%
	AQ + DCI-Select	78.0 ± 6.6%—No change
	AQ + DCI-Generate	84.7 ± 6.9%

Table 5. Results after learning in the original representation space.

	Average accuracy (%)	Average # of Rules	Average # of selectors	Average learning time (seconds)
AQ alone	72.5	27.7	94.8	231.7

In the previous study, combinations and subsets of attributes were generated and evaluated by hand. The goal of our experiment was to find a good representation automatically using our constructive-induction approach. Table 2 shows the attributes used in these experiments to describe news articles.

In the earlier study, representations consisting of keywords, POLs, and subjects were tested separately, along with a complete test of all attributes. The Complete and Subject attribute sets performed best. We thought constructive induction could further improve on this performance, using attribute selection on the Complete set and attribute construction on the Subject set.

The complete attribute set consists of 145 attributes, which we tested using three methods:

- DCI-Generate, which combines AQ with data-driven constructive induction to build new attributes;
- DCI-Select, which combines AQ with data-driven constructive induction to select relevant attributes; and
- AQ-only, which used only an AQ-type rule-learning algorithm.

We ran the DCI-Generate and the DCI-Select methods, then compared the results with those for the AQ-only set. We expected that attribute selection would have the greatest impact on predictive accuracy. We used a tenfold cross-validation methodology with a 70/30 split of the data set. Table 3 shows the averaged results with the 90% confidence interval.

Results. As the results show, predictive accuracy was higher after attribute selection and attribute construction had been done. DCI-Select made the greatest improvement. We expected this because the Complete set has a large number of potentially redundant attributes (providing keywords, proper names, and assigned subjects for each article). Clearly, for this small sample, not all these attributes are needed; 85 attributes were removed, including gender_person1, gender_person5, and type_org, which aren't strongly correlated with the USMED interest.

Some improvement was also made by DCI-Generate in constructing new attributes using the counting attribute constructor #Attr(S,C). The counting attribute constructor is a powerful attribute-generation operator because it can allow complex attribute interactions (such as parity) to be simply described. In this domain, S is the set of subject attributes for each example and C is a condition on their values such as "subject-value = 'finance'." This transformation overcomes an awkwardness of the original representation, in which an article is described by an ordered vector of subjects, such as [subject1 = medicine] or [subject2 = sports] or [subject3 = finance]. The problem with this representation is that it does not capture the nonordered nature of a person's interest. For example, if I am interested in sports, I don't really care if "sports" appears in subject1 or subject2; I want to know if any one of the subjects is sports.

In our experiment's data set, there were 100 different possible subjects. Extending the attribute set with all possible binary attributes would make the total vector very long and reduce the program's capability to produce useful generalizations. These concepts are difficult to represent without the DCI-generated attributes. For each of the 100 possible subject values, the counting operator generates a new attribute. Each new attribute represents the number of times (value-cardinality) that value is present in the vector. Each new attribute is filtered for quality, so as not to overwhelm the learning algorithm.

In our experiment, the counting operator constructed new attributes in six of 10 runs. With the counting operator, an article's subject can be more succinctly stated. For exam-

Table 6. Results after applying a single RSM operator to the representation space.

	Average accuracy (%)	Average # of Rules	Average # of selectors	Average learning time (seconds)	Average CI Time (seconds)
DCI-Sel	75.3	34.7	74.6[1]	215.1	3.1
DCI-Quant	85.9[1]	34.6	114.7	10.8 [1]	5.7
DCI-Gen	87.1[1]	18.5[1]	63.5[1]	171.1 [1]	8.1

[1]Significance $\alpha = 0.01$

Table 7. Results after applying multiple RSM operators. Significance over the baseline is shown with the first superscript value. Significance to the first transformation alone is shown with the superscript in parentheses.

	Average accuracy (%)	Average # of rules	Average # of selectors	Average learning time (seconds)	Average CI Time (seconds)
DCI-Quant ->DCI-Gen	85.4[1]	25.6	147.8	283.0	5.7+1.4 = 7.1
DCI-Gen ->DCI-Quant	93.4[1,3]	20.5[1]	63.7[1]	104.5 [1]	8.1+19.3=27.4
DCI-Gen->DCI-Sel	90.3[1]	25.7	53.6[1,3]	142.2 [1,2]	8.1+1.0 = 9.1

[1]Significance $\alpha = 0.01$ [2]significance $\alpha = 0.05$ [3]significance $\alpha = 0.1$

ple, to say "the article is about finance" is represented as #Attr(SA, $a_k \in$ SA & a_k= finance)=1, and #Attr(SA, $a_k \in$ SA & a_k=computer)=0 is used to state "the article is not about computers." Following is a rule describing articles relevant to the user, that used the counting generated attributes. The rule's total coverage is shown by the t-weight. The u-weight counts the uniquely covered examples. (The selector [Gender=m v f] is present in the rules because gender is assumed here to have three possible values: m, f, and unknown.)

```
[Class_good<::
[Subj1=sci_&_tech]&
[Subj2=institutions]&
[Subj3=economy or medicine]
[#Attr(SA, ak∈ SA &
ak=finance)=0] &[#Attr(SA,
ak∈ SA & ak=computer)=0]
(t:3, u:3)
```

where SA = {$Subj_1$, $Subj_2$, $Subj_3$, $Subj_4$, $Subj_5$}. An article is of interest if subject1 is about science and technology, subject2 is about institutions, and subject3 is about the economy or medicine, and if none of the subjects take the value "finance" or "computers."

Table 4 shows the results of the DCI-Generate transformation to the Subjects attribute set, as well as the DCI-Select results compared against the AQ-only baseline.

Data-driven constructive induction was able to improve the ability of both the Complete and Subjects attribute sets to predict user interest in newswire text. The transformations were quite comprehensible. In addition to bringing close to 29% improvement in predictive accuracy (in the Complete set), the construction of useful forms of #Attr(Subjects, x) by the counting operator of AQ17-DCI let the learning algorithm better overcome an awkwardness in the original representation of newswire articles which, if generated by hand, would have otherwise greatly expanded the representation space.

Natural-scene interpretation. Classifying regions in natural-scene images is a difficult problem in computer vision. The goal is to develop a method that can accurately distinguish objects in outdoor scenes under varying perceptual selectors. In our approach, we wanted the program to learn characterizations of natural object classes (sky, trees, road) from labeled images. The program can then apply these characterizations—which are based on attributes extracted for pixel windows—to new scenes and predict the presence of natural objects.

In our experiment, the input was a training image that includes selected examples of the visual concepts to be learned: sky, trees, and road. A 5 × 5 windowing operator, scanned over the training area, was used to extract a number of attributes, including color (intensity of red, green, and blue) and texture (we used Law masks for detecting horizontal and vertical lines, high frequency spots, horizontal and vertical v-shapes, and Laplacian operators). The generated rule quality was evaluated using a tenfold cross-validation method. The data set has 450 examples equally distributed between the three classes.

Results. The standard approach to solving this problem is to apply a selective-induction learning algorithm directly to the raw data. Table 5 shows the results of this approach, which we used as the baseline performance. Table 6 shows our results using attribute selection (DCI-Sel), attribute abstraction (DCI-Quant), and attribute construction (DCI-Gen).

The abstraction operator reduced average attribute domain size from 256 to 14. The rules learned after this reduction in representation-space size had a significantly higher prediction accuracy and took less time for the program to learn. However, rule complexity increased from an average of 27.7 rules to 34.6 rules, and there was a significant increase in the number of selectors used in the rules. Although this increase in complexity is surprising, it could be reduced by applying the Trunc method for rule-set reduction and flexible matching, and (as our study shows), by other modifications of the representation space.[11]

The rules learned from the space expanded by DCI-Generate were also significantly more accurate than the rules learned in the original space. DCI-Generate added on average 10 new attributes, the strongest of which described absolute and relative differences in the intensity of reds, greens, and blues. Given that the training images presented green trees, a dark road, and a blue sky, this result is not surprising. The tree class included new attributes that stated [green > red] and [green > blue]. Introducing these attributes to the representation space significantly improved all aspects of the resulting rules. In the new DCI-Gen expanded representation space, the program quickly learned fewer rules that were more accurate and less complex than those in the original representation space. These improvements were achieved after only 8.7 seconds spent searching for new attributes. The total time spent constructing new attributes and learning in the new space was 22% less than learning alone in the original representation space.

DCI-Sel consistently removed attributes x4, ..., x8—attributes such as horizontal and vertical line and high-frequency spot that describe the patterns within the 5 × 5 extraction window. This slightly increased predictive accuracy and significantly reduced learning time and the number of selectors in the rules. However, it also increased the number of rules generated.

Results: combined operators. Our success with single-strategy results led us to investigate combinations of RSM operators, especially attribute generation through DCI-Gen combined with space abstraction through DCI-Quant. Table 7 shows our results.

Performing attribute abstraction followed by attribute generation (DCI-Quant -> DCI-Gen) reduced the number and size of the learned rules, but slightly decreased accuracy from abstraction alone. This reduction in the representation space may remove important information. If we reverse the operators and apply DCI-Quant to the space already expanded by DCI-Gen, the space is significantly improved in almost all respects: learning time and predictive accuracy are significantly better than in both the original space and the DCI-Gen-only space. Although rule complexity and number slightly increased from the DCI-Gen-only space, it is still significantly smaller than in the original representation.

Using DCI-Sel after DCI-Gen also brought improvements: there was a small increase in predictive accuracy over DCI-Gen alone and a significant decrease in learning time and selectors used—but an increase in the number of rules. In this space (as in the original representation), DCI-Sel removed attributes x4, ..., x8. We also tried DCI-Gen followed by DCI-Quant and DCI-Sel, but this combination showed no improvement in the predictive accuracy of the learned rules.

Our best results came from attribute generation followed by attribute quantization: rules were significantly more accurate, learned much faster, and were much less complex than rules learned in the original representation. This suggests that the original representation has both attribute interaction and excess detail. By performing DCI-Gen first, we seem to have partially captured this interaction; the abstraction operator of DCI-Quant was thus able to safely perform its operation without looking at the context of other attributes. Because DCI-Quant (using the ChiMerge algorithm) views each attribute independently, it may remove important classification information. By running DCI-Gen first, the danger is reduced.

It is premature to draw any general conclusions about the best ordering of RSM operators from a single experiment. We conclude simply that some patterns are best described in the original problem formulation, and some only become apparent after abstraction. If abstraction is sensitive to attribute interactions, it may be possible to eliminate such ordering effects, but such a method would have to search an enormous space of both combinations and abstraction levels. An approach that more tightly couples the search for combinations and the abstraction level is an interesting and important area for future research.

The attributes constructed by DCI-Gen were not only useful for classification, but also easily understood as differences in color intensities. The differences in color intensity between red and green, and between green and blue, were consistently in the top three most informative attributes as measured by information gain. The difference between red and blue was also generated, but its discriminatory power was not high. Our success with color intensity may inspire the use of other representations, such as hue and saturation to represent color information in the image. Such an interaction is not currently possible with the DCI method, as it only combines pairs of numeric attributes. Searching for more complex functions of original attributes is an open area of future research.

CONSTRUCTIVE INDUCTION INTEGRATES ideas and methods previously considered separate: attribute selection, construction, and abstraction. By integrating these methods into AQ17-DCI, we were able to increase predictive accuracy by up to 29% in our test cases.

Our experiments also raised interesting questions for future work.

- *How tightly should RSM operators be integrated?* Our experience with the natural-scene interpretation problem showed that an ordering effect exists between attribute construction and abstraction. Integrating these operators to the correct level of granularity for all available attributes is difficult because of the massive size of the search space.
- *Can a program learn metarules to guide the application of the RSM operators for any given problem?* We don't entirely understand the conditions under which these RSM operators improve the representation space. By recording the result of these operators under a variety of conditions, we hope to gain insight into this question. Initial work in this direction has already begun.[3] Our experiments offer great promise for improving the power of inductive learning in real-world applications. ■

Acknowledgments

This research was partially conducted in the Machine Learning and Inference Laboratory at George Mason University; the text classification experiment was done by Bloedorn at Mitre under internal research support. The GMU MLI Laboratory's research is supported by the National Science Foundation under Grants Nos. CDA-9309725, IRI-9020266, and DMI-9496192; by the Office of Naval Research under Grant No. N00014-91-J-1351; by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, administered by the Office of Naval Research; and by Grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research.

References

1. J. Wnek and R.S. Michalski, "Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments," *Machine Learning*, Vol. 14, No. 2. 1994, pp. 139–168.
2. R.S. Michalski, *Pattern Recognition as Knowledge-Guided Computer Induction*, Tech. Report 927, Dept. of Computer Science, Univ. of Illinois, Urbana-Champaign, Ill., 1978.
3. E. Bloedorn, J. Wnek, and R.S. Michalski, "Multistrategy Constructive Induction," *Proc. Second Int'l Workshop Machine Learning (ML93)*, Morgan Kaufmann, San Francisco, 1993. pp. 188–203.
4. S.B. Thrun et al., *The Monk's Problems: A Performance Comparison of Different Learning Algorithms* (revised version), Tech. Report CMU-CS-91-197, Carnegie Mellon Univ., Pittsburgh, 1991.
5. E. Bloedorn and R.S. Michalski, "The AQ17-DCI System and Its Application to World Economics," *Proc. Ninth Int'l Symp. Method-*

ologies of Intelligent Systems, Springer-Verlag, Berlin, 1996, pp. 108–117.

6. Y. Hu and D. Kibler, "Generation of Attributes for Learning Algorithms," *Proc. 13th Nat'l Conf. Artificial Intelligence (AAAI96)*, AAAI Press, Menlo Park, Calif., 1996, pp. 806–811.

7. G. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," *Proc. Eleventh Int'l Conf. Machine Learning (ML94)*, Morgan Kaufmann, 1994, pp. 121–129.

8. J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," *Proc. 12th Int'l Conf. Machine Learning (ML95)*, Morgan Kaufmann, 1995, pp. 194–202.

9. R.S. Michalski, "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning," *Machine Learning: A Multistrategy Approach*, R.S. Michalski and G. Tecuci, eds., Morgan Kaufmann, 1993, pp. 3–61.

10. J. Wnek et al., *Selective Inductive Learning Method AQ15c: The Method and User's Guide*, Tech. Report MLI95-4, Machine Learning and Inference Laboratory, George Mason Univ., Fairfax, Va., 1995.

11. F. Bergadano et al., "Learning Two-Tiered Descriptions of Flexible Concepts: The Poseidon System," *Machine Learning*, Vol. 8, No. 1, Jan. 1992, pp. 5–43.

12. R. Kerber, "ChiMerge: Discretization of Numeric Attributes," *Proc. Tenth Nat'l Conf. Artificial Intelligence*, AAAI Press, 1992, pp. 123–128.

13. E. Bloedorn, I. Mani, and T.R. MacMillan, "Machine Learning of User Profiles: Representational Issues," *Proc. 13th Nat'l Conf. Artificial Intelligence*, AAAI Press, 1996, pp. 433–438.

Eric Bloedorn is a senior staff member of the Artificial Intelligence Technical Center at the Mitre Corporation. His interests include machine learning and its application to text classification and data mining. He received his BA in physics from Lawrence University, and his MS and PhD from George Mason University. He is a member of the AAAI and ACM. Contact him at Mitre, 1820 Dolly Madison Blvd., W647, McLean, VA 22102; bloedorn@mitre.org.

Ryszard S. Michalski is PRC Chaired Professor of Computer Science and Systems Engineering and director of the Machine Learning and Inference Laboratory at George Mason University. He is also an affiliate professor of the Institute of Computer Science at the Polish Academy of Sciences, Warsaw. His research interests include machine learning and inference, cognitive modeling, applications of machine learning to computer vision, and intelligent agents. He earned an MS from St. Petersburg Polytechnical University and a PhD from the University of Silesia in Poland. Contact him at the Machine Learning and Inference Laboratory, George Mason Univ., 4400 University Dr., Fairfax, VA 22030; michalski@gmu.edu.