# The Development of the AQ20 Learning System and Initial Experiments

**Guido Cervone**
**Liviu Panait**
**Ryszard Michalski***

Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA

* Also Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

e-mail: {cervone,panait,michalski}@mli.gmu.edu

***Abstract****: Research on a new system implementing the AQ learning methodology, called AQ20, is briefly described, and illustrated by initial results from an experimental version. Like its predecessors, AQ20 is a multi-purpose learning system for inducing general concepts descriptions from concept examples and counter-examples. AQ20 is viewed as a natural induction system because it aims at producing descriptions that are not only accurate but also easy to understand and interpret. This feature is achieved by representing descriptions in the form of attributional rulesets that have a higher representation power than decision trees or conventional decision rules. Among new features implemented in AQ20 are the ability to handle continuous variables without prior discretization, to control the degree of generality of rules by a continuous parameter, and to generate more than one rule from a star. Initial experimental results from applying AQ20 to selected problems in the UCI repository demonstrate a high utility of the new system.*

*Keywords: Symbolic Learning, Natural Induction, learning from examples.*

## 1. Introduction

The AQ learning methodology traces its origin to the $A^q$ algorithm for solving general covering problems of high complexity (Michalski, 1969a, 1969b). An implementation the $A^q$ algorithm in combination with the *variable-valued logic* representation produced the first AQ learning program, AQVAL/1, which pioneered research on general-purpose inductive learning systems (Michalski, 1975). Subsequent implementations, developed over the span of many years, added many new features to the original system, and produced highly versatile learning methodology, able to tackle complex and diverse learning problems.

One of the recently added features is pattern discovery mode, which allows the program to generate rulesets representing strong patterns in large volumes of noisy data (Michalski and Kaufman, 2000b). The pattern discovery mode complements the original theory formation mode that determines complete and consistent data generalizations.

Due to a wide range of features and a highly expressive representation language, recent members of the AQ family of learning programs arguably belong to the most advanced symbolic learning systems. They have been used in a wide range of domains, including medicine, agriculture, engineering, image processing, economy, sociology, music, geology, and others. An early application of AQ to soybean disease diagnosis was considered one of the most significant achievements of machine learning (Michalski and Chilausky, 1980).

The rapid development of computer technology and high-level programming languages often stimulated researchers to port existing versions of the methodology to new programming environments. AQ programs have been written in Pascal, Fortran, InterLisp, CommonLisp, C and now C++, and for a variety of platforms, including IBM 360, Symbolics, Next/Step, Digital Vax, Digital Ultrix, SunOS 1.0 and 2.0, MacOS, MS-DOS, Windows and recently Linux. These developments were done in an academic environment and primarily for educational purposes. Consequently, they often lacked stability and reliability required by outside users who were most interested in practical applications and the simplicity of program's use rather than in research ideas. As a result, despite of their conceptual advantages, AQ programs met so far a rather limited use.

Before undertaking the task of developing a new AQ program, we analyzed some of the most stable programs, such as AQ15 (Michalski et al., 1986), and AQ18 (Kaufman and Michalski, 2000a) in order to identify parts of the code that needed restructuring. This analysis indicated that the above implementations were optimized for speed, rather than for the extendibility or comprehensibility of the code. As the last two concerns were of great interest to us, we decided to give AQ20 a fresh new start.

Using knowledge acquired from the analysis of the previous codes, a new design aims at making AQ20 reliable, easy to use, easy to modify and extend, while including features previously implemented in AQ rule learning systems. The aim of AQ20 is to become a machine learning environment that supports users in conducting machine learning experiments.

## 2. A Brief Review of the AQ Methodology

To make the paper self-contained, this section reviews very briefly some of the basic features of AQ learning. A detailed description of various aspects of the methodology can be found in (Michalski, 1969a; Michalski, 1974; Michalski, 1983; Michalski, 2001).

AQ pioneered the progressive covering (a.k.a. "separate and conquer") approach to concept learning. It is based on an algorithm for determining quasi-optimal (optimal or sub-optimal) solutions to general covering problems of high complexity (Michalski, 1969b). The central concept of the algorithm is a star, defined as a set of alternative general descriptions of a particular event (a "seed") that satisfy given constraints, e.g., do not cover negative examples, do not contradict prior knowledge, etc. In its simplified version, the algorithm starts by randomly selecting a "seed" from among concept examples, and then creates a *star* for that example. A member (a rule) of the star that satisfies a given preference criterion is selected, and examples covered by it are removed from the set of concept examples. A new seed is selected from the uncovered-so-far examples, and the process repeats until there are no more examples to be covered.

In AQ20, the condition of selecting only one rule from a star has been relaxed and more than one rule can be selected, which speeds up the learning process. The seed generalization process takes into consideration the type of variables in the training data. AQ20 distinguishes between the following types of variables: binary, nominal, linear, continuous, and structured (Kaufman and Michalski, 1996).

Previous versions of AQ dealt with continuous variables by discretizing them into a number of discrete units, and then treating them as ordinal variables. AQ20 does not require such discretization, as it automatically determines ranges of continuous values for each variable occurrence in a rule.

The AQ learning process can proceed in one of two modes: (1) the *theory formation mode* (TF), and (2) the *pattern discovery mode* (PD). In the TF mode, AQ learns rules that are complete and consistent with regard to the data. This mode is mainly used when the training data can be assumed to contain no errors. The PD mode is used for determining strong patterns in the data. Such patterns may partially inconsistent or incomplete with regard to the training data. The PD mode is particularly useful for mining very large and noisy databases.

The core of the AQ algorithm is the star generation, which is done in two different ways, depending on the mode of operation (TF or PD. In TF mode, the star generation proceeds by selecting a random positive example (called seed) and then generalizing it in various ways to create a set of consistent generalizations (that cover the positive example and do not cover any of the negative examples). In PD mode, rules are generated similarly, but the program seeks strong patterns (that may be partially inconsistent) rather than fully consistent rules.

The star generation is an iterative process. First, the seed is *extended-against* each negative example (this is a pair-wise generalization operation). These extensions are then logically multiplied out to form a star, and the best rule (or rules) according to a given multi-criterion functional (LEF) is selected. LEF is composed from elementary criteria and their tolerances so that it best reflects the needs of the problem at hand. The program uses beam search to speed-up this process. The beam width is defined by the *maxstar* parameter. For details, see e.g., (Michalski et al. 1986).

In the original version of the methodology, only one rule, the best according to LEF, was selected. In AQ20, one or more rules may be selected from a star, depending on the degree of intersection between rules. The procedure for selecting rules from a star works as follows:

1 Sort the rules in the star according to LEF, from the best to the worst.

2 Select the first rule, and compute the number of examples it covers. Select the next rule, and compute the number of new training examples it covers.

3 If the number of new examples covered exceeds a *new_examples_threshold*, then the rule is selected, otherwise it is discarded. Continue the process until all rules are inspected.

The result of this procedure is a set of rules selected from a star. The list of positive_events_to_cover is updated by deleting all those events that are covered by these rules. A new seed is selected, and the entire process is repeated. It continuous until there the list positive_events_to_cover becomes empty.

## 1. A Simple Illustration of an AQ Execution

This section illustrates how AQ20 learns rules that characterize a class of events in Theory Formation mode. It also shows the format of an AQ20 input file. Various program parameters and options were omitted due to the space limitation. These are explained in the AQ20 Users' Guide.

The input consists of a set of events, in which each event is classified either as a member of the target class (positive event) or not a member (negative event). Let's assume that an event is defined by values of four nominal variables: X, Y, Z and GROUP. The first step in preparing an input to the program consists of defining attributes and their domains. Each attribute must be assigned a well-defined domain. A domain may be shared, however, by one or more attributes.

Since in our example variables are nominal, their domains are unordered sets. Let's assume that the domain of X, denoted domain_x, is a set {0, 1, 2}, the domain of Y and Z, denoted domain_yz is a set {0, 1}, and the domain of GROUP, denoted domain_group, is a set {good, bad}. The above information is communicated to AQ20 by an input shown below:

```
Generic Domains
{
   domain_x  nominal { 0, 1, 2 }
   domain_yz nominal { 0, 1 }
   domain_group   nominal { good, bad }
}
Attributes
{
   X          domain_x
   Y          domain_yz
   Z          domain_yz
   GROUP  domain_group
}
```

In addition to attributes and their domains, a user may define various control parameters, or allow the system use their default values. An AQ run corresponds to a learning session. The following section illustrates how to set the output variable(s) to define the target class (GROUP is good), and how to set the maxstar parameter (in this case it is equal to 1, that is only one rule is maintained during the star generation). The LEF for this experiment is MaxPositiveCoverage with tolerance 0% and MinNegativeCoverage with tolerance 0%. This means that AQ will prefer rules with have high positive coverage and low negative coverage.

```
Runs
{
  run1_simple_example_of_AQ20 {
     output [GROUP = good ]
     maxstar = 1
     LEF { (MaxPositiveCoverage, 0),  (  MinNegativeCoverage, 0 ) }
  }
}
Events
{
          X, Y, Z, GROUP
           0, 0, 0,  good
           2, 0, 1,  good
           1, 1, 1,  good
           1, 1, 0,  bad
           2, 1, 0,  bad
           0, 1, 1, bad
}
```

The algorithm starts by creating two classes of events, one called positive, and one called negative. The positive events are those with GROUP=good, and remaining are negative events. The algorithm selects a random positive seed, say, the first event on the list (0,0,0), which can be represented as a condition [X=0][Y=0][Z=0]. This seed is used to generate the first star. Elementary partial stars are generated by applying the *extension against operator* to the seed and each negative example (see Section 6.3 for an explanation). The first elementary partial star is obtained be extending the seed (0,0,0) against the first negative example (1,1,0). This operation produces two alternative, maximally general rules that cover the seed and do not cover the negative event:

{[X=0, 2] (p=2, n=2), [Y=0] (p=2, n=0)}

In the first rule (read: X takes value 0 or 2), p=2 means that the rule covers 2 positive events (one is the seed), and n=2 means that it also covers 2 negative events. Let's us assume that LEF selects only one rule [Y=0], because it has the same positive coverage as the first rule, but lower negative coverage. The obtained rule constitutes a reduced elementary partial star {[Y=0]}. By extending the seed against the second negative example, another elementary partial star is generated :

{[X=0,1] (p=2, n=2), [Y=0] (p=2, n=0)}

The obtained star is logically multiplied by the previous partial elementary star, which produces a *partial star* {[X=0,1][Y=0], [Y=0]}. The positive and negative coverage is calculated again for each of the rules. [X=0,1][Y=0] covers 1 positive and 0 negative events, while [Y=0] covers 2 positives and 0 negatives. According to LEF, the second rule is better. Since the parameter maxstar is 1, only one rule can be selected, so the resulting partial star is

{[Y=0]}.

A partial star obtained from extending the seed against the third negative example is:

{[Y=0] (p=2, n=0), [Z=0] (p=2, n=1)}

By multiplying the two elementary partial stars, and using absorption laws, the following partial star is created:

{[Y=0] (p=2, n=0), [Y=0][Z=0] (p=1, n=0)}.

It contains two rules, one covering 2 positive events and the other one covering only 1 positive event. None of the rules covers any negatives. As the second rule is subsumed by the first one, it  deleted from the star.

In general, after a partial star (the "star-generated-so-far") is determined, all rules that are subsumed by other rules are deleted.  When this process is completed for the last negative example, a star of the given seed is obtained. AQ20 then selects from it one or more rules, as described above.  In our case, the algorithm produces just one rule [Y=0].

Because this rule does not cover all positives example (in this case, the third positive event), the above process is repeated for another randomly selected seed from the examples uncovered by the selected rule/s. This is the event (1,1,1,good).  The second star generation process gives the rule: {[X=1][Z=1] (p=1,n=0)}, which completes the learning process.

The above two rules constitute a complete and consistent description of the good GROUP, which AQ20 outputs in the form:

[GROUP = good] <- [Y=0]
[GROUP = good] <- [X=1][Z=1]

## 6. New Features in AQ20

AQ20 is not just a reimplementation of the AQ methodology, but it comes along with several new features not present in previous versions. One of the concerns of the authors was the ease of extending the program in the future. For this reason, AQ20 employs an object-oriented architecture. The architecture of the classes (which underwent a few redesign processes) provides, through polymorphism, an easy mechanism for adding new features. Adding binary attributes to AQ20 was a matter of minutes, while adding the linear ones took less than half an hour.

One of the most important new features in AQ20 is handling of continuous variables without prior discretization. This feature was implemented in such a way that it provides the user with an ability to control the level of rule generalization. Another new feature in AQ20, already mentioned earlier, is the possibility of selecting more than one rule from a star, which helps to speed up the learning process.

### 6.1 Handling of Continuous Variables

Previous versions of AQ handle continuous attributes through discretization. The domain of continuous variables is split into several ranges, usually using the $\chi^2$ method, or by hand. This way continuous variables are turned into discrete ones, and AQ works as described above. This method has the advantage that the search space is reduced from infinite to finite.

One disadvantage of this approach is a limited accuracy of the representation, since ranges cannot be modified by the AQ algorithm. Such a limitation may be particularly undesirable in incremental learning, because it limits the locations of concept boundaries to places specified by the predefined ranges. Another weakness is the growth of the memory requirements with the accuracy of discretization. In the AQ representation of discrete variables, one bit is used for one discrete unit. Thus, when a variable is discretized to a larger number of discrete units, the memory allocated for its representation also grows. The latter feature slows down the star generation process.

In contrast to the above, AQ20 represents conditions with continuous variables are by ranges of continuous values. All the operations on conditions with continuous variables are operations on ranges, represented by two real values. By moving the boundaries of the ranges, a condition can be easily specialized or generalized. For example, the condition

[ X = 2 .. 7 ]
can be specialized by narrowing the range to:
[ X = 3.5 .. 4.2 ]
or generalized by widening the range to:
[ X = 1.8 .. 7.5 ].

Such operations can be done only approximately when a variable is discretized into a fixed number of units.

### 6.1 Adjustable Level of Generality

The level of generality of the rules can be controlled through a parameter called *epsilon,* associated to each attribute. This means that in the same rule, some attributes may be more specific than others.

The epsilon parameter ranges between 0 and 1. When epsilon is set to 1 (0), the learned condition with this attribute will be maximally specific (general). Setting epsilon to a value between 0 and 1 defines an intermediate level of specificity. The meaning of the specificity level depends on the attribute type. For continuous and linear attributes, the epsilon parameter determines the location of the boundary within the distance between the negative and the positive examples that lead to this condition; for structured attributes, it determines a position of the boundary within the structure of the attribute domain. For nominal attributes, the epsilon parameter is assumed to have only values 0 and 1. Any value smaller than 0.5 is rounded to 0, and any value greater or equal to 0.5 is rounded to 1.

The default value of epsilon in AQ20 is 0.5. Thus, if a positive example has value of X equal 4, and a negative examples has value of X equal 10, the extension against operation will create a condition [X=0.. 7], since (4+10)/2=7. The value of epsilon is fixed for a given run. A more flexible way of implementing epsilon would be to adjust it dynamically, so that some criterion related to the description quality is optimized.

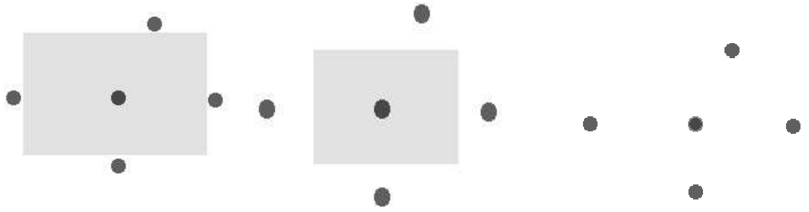

*Fig 1,2,3* Rule generalization for different values of epsilon

Figures 1, 2 and 3 illustrate rule generation with different epsilon values. The number of attributes is two, so the representation space is a plane. The five examples are represented by dots. In each figure, the central dot represents the seed, and the other four dots represent negative examples. Generated rules are represented by darkened rectangles. Recall that epsilon equal to 0 means maximum generalization, and epsilon equal to 1 means maximum specialization.

As shown in the Figures 1, 2, and 3, when epsilon takes a small value, the learned rule is more general, and when it takes a higher value, the rule is more specific.

### 6.3 Implementation of the Extension Operators

The star generation process is governed by the *extension-against* and extension-within operators (Michalski, 1975). This process can also be decomposed in the extension against and within at the attribute level. This paragraph illustrates how this is done in the case of continuous attributes. For details on how it works for other types of attributes, consult (Michalski, 1975). In the remaining of this section, we will refer to the extension-against operator applied with a single variable.

The *extension-against operator* applied to conditions with a single variable has as input two conditions, one covering a positive event and the second covering a negative event. The result of the operator is an epsilon-controlled generalization of the condition for the positive event that does not cover the negative event.

The *extension-within operator* applied to conditions with a single variable receives as input a condition covering of a positive example, and a general condition that may or may not subsume the first condition. If the former is the case, the result of extension-within is an epsilon-controlled generalization of the first condition that is subsumed by the second condition.

To illustrate how AQ20 implements these operators with the control parameter epsilon, suppose that the value of a continuous attribute X for the positive event is 4, and for the negative event is 10. In the first step, the result of the extension-against is the maximally general hypothesis [X < 10], that is, as if epsilon=0.

After performing all the logical intersections, the result are maximally general hypotheses. In order to introduce the epsilon-adjustable levels of generality, a post-processing is applied. AQ20 determines the maximally specific hypothesis that covers all positive examples covered the maximally general hypothesis (by an operation called *refunion*, which is computationally quite simple). Subsequently, the epsilon parameter is applied for each attribute within the maximally specific and the maximally general hypotheses.

### 6.4 Selecting Multiple Rules from a Star

In the rule generation process, the AQ algorithm creates a series of stars in order to cover all positive events. Each star contains a set of rules, all covering the seed of the star. In previous versions of AQ, only one (best according to LEF) rule was selected from the star.

AQ20 can select more than one rule from a star. This leads to the possibility of covering more positive events while generating only one star. Rules in a star have usually have high logical intersection, but not always. Keeping two or more rules that have a small logical intersection can speed-up the learning process, because the number of star generated can be reduced. A potential problem is that the rules may be less optimal, since two rules generated from the same star are never disjoint. AQ20 uses a simple method to calculate the ‘gain’ obtained by each rule, defined as the number of new positives covered, if the rule were selected from the star.

## 7 Experiments

This section presents selected results from experiments in which AQ20 was applied to two datasets from the UCI repository, *mushroom* and *wine*, and to one synthetic dataset. Results from AQ20 were compared in terms of predictive accuracy with C4.5 and with AQ18.

The datasets were split into training and testing events using different ratios in order to monitor the dependence of the predictive accuracy on the number of training events.

Testing was done using the ATEST module that can execute a flexible and strict matching of events with rules. Flexible matching is a technique for calculating the degree of confidence (a degree of match) that an event belongs to a class. In the case that the event does not match any of the rules constituting the class description, the confidence is 0. If the event matches one or more rules, the confidence is non-zero, and can range from 1 (strict match) to values close to 0 (Michalski et al. 1986).

### 7.1 Mushroom Database

The mushroom database contains the descriptions of more than 8000 different species of mushrooms, classified as poisonous or edible. Each mushroom is described in terms of 23 attributes, out of which 22 are discrete (some are nominal and some are linear), plus 1 nominal attribute representing the decision class. The database was randomly split into training and testing events. The number of training events was increased gradually to analyze how the predictive accuracy increases.

In the experiments relative to the mushroom database the LEF was *MinNumSelector* with tolerance 30% and *MaxNewPositives* with tolerance 10%. This is the standard LEF for AQ20 and for AQ18, and results in a faster execution of the algorithm.

The results are presented in confusion matrices below, in which:

- the first row and column contain the names of the classes used in the experiment
- the element a[i,j] (i and j different than 1) contains the number of elements that belong to class a[i, 1] and were classified as belonging to class a[1, j].
- the classes may not cover the whole space, therefore a new class (called “unknown”) is automatically added

Thus, high predictive accuracy is manifested by numbers equal or close to 1 in the main diagonal, and 0s or small values outside the main diagonal. The *accuracy* of a classification is defined in the experiments below as the ratio of the total number of correct classifications (the sum of the elements on the main diagonal in the table) over the total number of classifications (the sum of all the elements in the table). For a training-testing examples ratio of 0.02, AQ20 obtained the following results (Accuracy = 98.3%):

| Predictions | Edible | Poisonous | Unknown |
|---|---|---|---|
| Edible | 4061 | 48 | 0 |
| Poisonous | 84 | 3765 | 0 |
| Unknown | 0 | 0 | 0 |

Table 1 AQ20 rule predictions when the training set included 0.02% of examples from the Mushroom dataset.

For a training-testing examples ratio of 0.10, AQ20 obtained the following results (Accuracy = 100%):

| Predictions | Edible | Poisonous | Unknown |
|---|---|---|---|
| Edible | 3780 | 0 | 0 |
| Poisonous | 0 | 3532 | 0 |
| Unknown | 0 | 0 | 0 |

Table 1 Results of AQ20 with 0.1% learning on the Mushroom database

In addition to the high predictive accuracy, the rules learned by AQ20 were typically quite simple and had high coverage. For example, when learning from only 10% of the examples, AQ20 found the following rules :

Rule 1: [class = p] ← [odor = c, y, f, m, p, s]

that covers 3796 from 3916 poisonous mushrooms described in the repository and none of the edible mushroom, and

Rule 2: [class = p]
← [cap-color = n,b,p,e,w,y] & [gill-color = b, g, r, w] &
[spore-print-color = r,w ] & [ population = c,v ] & [habitat = g,l,m,d]

that covers 471 positive (poisonous) and 0 negative (edible).

Predictive accuracy of AQ20 rules was compared with that of rules obtained from AQ18 and C4.5. The two AQ programs extracted very similar rules, and their predictive accuracy is almost identical. The rules generated by C4.5 are different from AQ20's rules, since the program uses a different representation language. For example C4.5 finds the following rules, all relative to the odor attribute.

R1: [class = p] ← [odor = c ]
R2: [class = p] ← [odor = y ]
R3: [class = p] ← [odor = f ]
R4: [class = p] ← [odor = m]
R5: [class = p] ← [odor = p]
R6: [class = p] ← [odor = s]
R7: [class = p] ← [cap-color = n] & [ring-number = 0 ] & [spore-print-color = w ]

Because of C4.5 does not have the internal disjunction operator in its language, it generated six rules that correspond to one AQ20 rule [odor = c,y,f,m,p,s]. The C4.5 Rule 7 was quite different from AQ20 Rule 2.

The following table summarizes the results of the experiments.

| | AQ18 | AQ20 | C4.5 |
|---|---|---|---|
| 2% training | 97% | 98% | 98.7 |
| 10% training | 99.7% | 100% | 99.7 |
| 30% training | 100% | 100% | 98.7 |

Table 3 Comparison of the Predictive Accuracy of AQ20, AQ19, and C4.5 on the Mushroom dataset

### 7.2 Wine Database

The second experiment concerned the application of AQ20 to the *wine* dataset. This dataset contains 178 events of different Italian wines, divided into three classes. Each event is defined by 14 attributes, 13 continuous attributes, plus 1 nominal attribute for the class. AQ20 generated the following rules:

**<u>First class</u>**:

[group = class1]
← [Alcohol ≥ 12.79] & [Alcalinity_of_ash ≤ 27.495]
[Total_phenols ≥ 2.095] & [Flavanoids ≥ 1.805]
[Color_intensity ≥ 3.4] & [Proline ≥ 679.005]

The rule covers all 59 events described in the dataset as belonging to this class.

**<u>Second class</u>**:

[group = class2]
← [Color_intensity ≤ 4.015] & [Proline ≤ 718.995]
← [Alcohol ≤ 13.12] & [Ash ≤ 3.065] &
[Color_intensity ≤ 4.855] & [Hue ≥ 0.785] &
[OD280_OD315_of_diluted_wines = 1.485..3.505]&
[Proline ≤ 1002.5]
← [Malic_acid ≤ 1.615] & [OD280_OD315 of
diluted_wines ≥ 1.575] & [Proline ≤ 975.995]
←[Alcohol ≤ 12.75] & [Flavanoids ≥ 1.27] & [Proline ≤ 975.995]

All four rules represent strong but intersecting patterns for the second class, covering 56, 58, 36, and 56, events respectively. The total number of training events in class 2 was 71.

**<u>Third class</u>**:

[group = class3]
← [Hue <= 0.825] & [OD280_OD315 of diluted_wines
≤ 2.055]
[group = class3]

$\leftarrow$ [Malic_acid ≥ 2.135] &
[Proanthocyanins ≤ 1.585] &
[Color_intensity ≥ 4.32][Hue ≤ 0.995] &
[Proline ≤ 979.995]

The first rule covers 40 and the second one 39 events (out of the total of 48 total events in class 3), and they are also highly intersecting (31 events).

AQ20 generated rules that are very easy to understand by human. When using 50% of randomly chosen training events and 50% for testing, the following results were obtained (Table 4).

| Predictions → | Class 1 | Class 2 | Class 3 | Unknown |
|---|---|---|---|---|
| Class 1 | 28 | 0 | 2 | 0 |
| Class 2 | 1 | 17 | 0 | 0 |
| Class 3 | 0 | 1 | 37 | 4 |
| Unknown | 0 | 0 | 0 | 0 |

Table 4 Results from the Wine database with 50% testing

This means that 82 out of 90 events were classified correctly (91.1% accuracy). Rules with different levels of generality were tested on the wine dataset. The tests used 10% events for training and the remaining 90% percent for testing. Results were obtained for three different values of epsilon: 0.1 (very general rules), 0.5 (medium) and 0.9 (highly specific). The results for the experiments are presented in the next 3 tables.

| Predictions → | Class 1 | Class 2 | Class 3 | Unknown |
|---|---|---|---|---|
| Class 1 | 40 | 4 | 0 | 6 |
| Class 2 | 3 | 32 | 6 | 2 |
| Class 3 | 5 | 1 | 44 | 14 |
| Unknown | 0 | 0 | 0 | 0 |

Table 5 Confusion matrix for highly general rules (epsilon=0.1). The accuracy is 73.9%.

| Predictions → | Class 1 | Class 2 | Class 3 | Unknown |
|---|---|---|---|---|
| Class 1 | 37 | 0 | 0 | 12 |
| Class 2 | 2 | 30 | 3 | 5 |
| Class 3 | 4 | 1 | 38 | 19 |
| Unknown | 0 | 0 | 0 | 0 |

Table 6 Confusion matrix for rules of medium generality (epsilon=0.5). The accuracy is 69.5%.

| Predictions → | Class 1 | Class 2 | Class 3 | Unknown |
|---|---|---|---|---|
| Class 1 | 28 | 0 | 0 | 21 |
| Class 2 | 1 | 26 | 2 | 10 |
| Class 3 | 2 | 1 | 36 | 22 |
| Unknown | 0 | 0 | 0 | 0 |

Table 7 Confusion rules for highly specific rules (epsilon=0.9). The predictive accuracy is 60.4%.

The number of events classified as “unknown” (last column) when the rules are highly specific (Table 7) is relatively large compared to the other tests. When the description of the classes becomes more and more general (epsilon=0.5; Table 6, and epsilon=0.1; the Table 5), the number of events classified as “unknown” decreases and the number of events classified correctly increases. On the other hand, number of events from one class incorrectly classified as belonging to another class also increases with the increase of generalization for the rules. This is, of course, an expected behavior.

Although the predictive accuracy increased with the rule generalization level, in this problem, it is preferable to classify an event as “unknown” than to misclassify it (it is clearly better to classify a poisonous mushroom as unknown than as edible). Therefore, it is better to keep the rules for edible mushrooms more specialized.

The results of AQ20 on the wine dataset were also compared with those from AQ18. The wine dataset was split in 50% training, 50% testing. The following table summarizes the results of the experiment with the default epsilon equal 0.5.

| | AQ18 | AQ20 |
|---|---|---|
| 50% training | 71% | 91.1% |

Table 8: Comparison of AQ18 and AQ20 on the Wine database with 50% training

As shown in the table, AQ18’ rules did not work as well as they did in the previous experiment. This result can be attributed either to an internal bug of AQ18 that could not be verified, or to the incapacity of AQ18 to find appropriate concept boundaries due to prior discretization. This result suggests that direct handling of continuous variables in AQ20 can significantly improve the predictive accuracy of the learned rules (in the case, 20%). Further experiments are needed to confirm this hypothesis.

It should be noted, however, that the AQ20’s method of handling continuous variables is more prone to over-fitting the training dataset due to a more powerful representation language. Thus, choosing an appropriate epsilon is an important problem.

More experiment will be done to investigate this further, trying to discretized the variables in a larger number of intervals, and see how this affects the quality of the rules.

### 7.3 Synthetic Database

The last experiment involved a designed dataset to test the ability of AQ20 to select more than one rule from a star whenever it is desirable. A dataset containing 80 events belonging to two classes was generated using a general logic diagram. The events were arranged in a such a way that there existed a star that have highly disjoint rules. Out of 80 events, 57 were used as training and 23 as testing.

In the experiment, AQ20 was run with and without the capability of generating more than one rule form a star. The number of star generated and the time required for learning in both cases was recorded. The results are presented in the table below:

| | 1 rule only | More than 1 rule |
|---|---|---|
| Number of stars | 2 | 1 |
| Time required | .6 sec | .4 sec |

Table 9. Time spent by AQ20 learning rules keeping more than a rule from a star versus learning rules by keeping only one rule from a star

The learned hypotheses were identical in both runs, but, as shown in Table 9, learning with multiple rule selection from a star decreased the execution time by 50%.

## 8 Future work

Among topics for future work are testing the program on a variety of problems, determining its advantages and disadvantages in comparison to other learning methods, such as Ripper (Cohen, 95), CN2 (Clark and Niblett, 89), C4.5 (Quinlan, 93), those employing rough set theory (e.g., Pawlak, 91), and previous implementations of AQ learning. It is planned to make these comparisons in terms of both the predictive accuracy and the simplicity and understandability of the learned rules.

Future work includes also an integration of AQ20 with supporting modules, such as Knowledge Visualizer that facilitates a visual validation of the learned rules (KV; Zhang, 1997), and ALPE for automatically executing and testing AQ learning programs (Lee and Michalski, 1997).

Among important features implemented in some AQ programs is constructive induction, which is the capacity of the algorithm to automatically improve the representation space for learning (e.g., Bloedorn and Michalski, 1998). Future work will also include the employment of AQ20 in the Learnable Evolution Model (LEM) methodology for non-Darwinian Evolutionary Computation (Cervone et al, 2000).

## 9 Conclusion

This paper described a new implementation of the AQ methodology called AQ20. AQ20 tries to encompass in one place many features previously found in several programs. It includes new features such as the capacity of keeping more than one rule from a star, that may greatly increase the speed of the algorithm, the capacity of dealing with continuous attributes without a fixed discretization, and the introduction of a new parameter called epsilon, that allows each attribute to be learned with a different level of generality. AQ20 can be freely downloaded from the website: www.mli.gmu.edu/aq20

## Acknowledgements

The authors wish to thank Dr. Ken Kaufman for his help in understanding the methodology and his continuous support throughout the development of AQ20. His help was crucial in comparing with AQ18, and solve problems with the previous program. This research has been supported in part by the National Science Foundation under grant IIS-9906858 and by a UMBC Grant under LUCITE Task #32.

## References

1. Cervone, G., Michalski, R.S., Kaufman, K. and Panait, L., "Combining Machine Learning with Evolutionary Computation: Recent Results on LEM," *Proceedings of the Fifth International Workshop on Multistrategy Learning (MSL-2000)*, Guimarães, Portugal, pp. 41-58, June, 2000.

2. Clark, P. and Niblett, T., "The CN2 Induction Algorithm," *Machine Learning* 3, pp. 261-283, 1989.

3. Cohen, W., "Fast Effective Rule Induction," *Proceedings of the Twelfth International Conference on Machine Learning*, Lake Tahoe, CA, 1995.

4. Kaufman, K.A. and Michalski, R.S., "Learning in an Inconsistent World: Rule Selection in STAR," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, 2000a (to appear).

5. Kaufman, K.A. and Michalski, R.S., "An Adjustable Rule Learner for Pattern Discovery Using the AQ Methodology," Journal of Intelligent Information Systems, 14, pp. 199-216, 2000b.

6. Lee, S.W. and Michalski, R.S., "ALPE: A System for Automatic Learning Performance Evaluation The Method and User's Guide," Reports of the Machine Learning and Inference Laboratory, MLI 97-6, George Mason University, Fairfax, VA, 1997

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

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

9. Michalski, R.S., "Synthesis of Optimal and Quasi-Optimal Variable-Valued Logic Formulas," *Proceedings of the 1975 International*

*Symposium on Multiple-Valued Logic*, pp. 76-87. Bloomington, Indiana, May 13-16, 1975.

10. Michalski, R.S, “A Theory and Methodology of Machine Learning, in Michalski, R.S, Carbonell, J.G. and Mitchell, T.M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, 1983, pp. 83-134.

11. Michalski, R.S., "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning," in Michalski, R.S. and . Tecuci, G. (eds.), *Machine Learning: A Multistrategy Approach, Vol. IV*, Morgan Kaufmann, San Mateo, CA, 1994.

12. Michalski, R.S., “Natural Induction: A Theory and Methodology,” *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, 2000 (to appear).

13. Michalski, R.S., and Chilausky, R.L., “Learning By Being Told and Learning From Examples: An Experimental Comparison of the Two Methods of Knowledge Acqusition in the Context of Developing an Expert System for Soybean Disease Diagnosis,” *Policy Analysis and Information Systems,* Vol. 4, No. 2, 1980.

14. Pawlak, Z., *Rough Sets: Theoretical Aspects of Reasoning about Data,* Kluwer Academic Publishers, 1991.

15. Quinlan, J.R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, Los Angeles, CA, 1993.

16. Zhang, Q., "Knowledge Visualizer: A Software System for Visualizing Data, Patterns and Their Relationships," *Reports of the Machine Learning and Inference Laboratory* , MLI 97-14, George Mason University, Fairfax, VA, September, 1997.