Reports

Machine Learning and Inference Laboratory

Multitype Pattern Discovery via AQ21

A Brief Description of the Method and Its Novel Features

Janusz Wojtusiak, Ryszard S. Michalski,
Kenneth A. Kaufman, and Jaroslaw Pietrzykowski

MLI 06-2
P 06-2
June 5, 2006

College of Science

George Mason University

MULTITYPE PATTERN DISCOVERY VIA AQ21

A Brief Description of the Method and Its Novel Features

Janusz Wojtusiak, Ryszard S. Michalski*, Kenneth A. Kaufman, and Jaroslaw Pietrzykowski
Machine Learning and Inference Laboratory
George Mason University

Fairfax, VA 22030-4444, USA
(*) Also with Institute of Computer Science, Polish Academy of Sciences
{jwojt, michalski, kaufman, jarek}@mli.gmu.edu
http://www.mli.gmu.edu

Abstract

The AQ21 program seeks different types of patterns in data and represents them in human-oriented forms resembling natural language descriptions. Because of the latter feature it is called a *natural induction* program. This feature is achieved by employing a highly expressive representation language, Attributional Calculus, that combines aspects of propositional, predicate and multi-valued logic for the purpose of supporting pattern discovery and inductive learning. This paper briefly describes the pattern discovery mode in AQ21, and several novel abilities seamlessly integrated in it, specifically, to discover different types of attributional patterns depending on the parameter settings, to optimize patterns according to a large number of different pattern quality criteria, to learn rules with exceptions, to determine optimized sets of alternative hypotheses generalizing the same data, and to handle data with missing, irrelevant and/or not-applicable meta-values. The discovered patterns are expressed in the form of attributional rules that are directly interpretable in natural language and are visualized using either general logic diagrams or concept association graphs. The described program features are illustrated by a sample of pattern discovery problems.

Keywords: Pattern discovery, Data mining and knowledge discovery, Machine learning, AQ learning, Alternative hypotheses, Exceptions, Meta-values, Knowledge visualization

Acknowledgments

This research has been conducted in the Machine Learning and Inference Laboratory of George Mason University, whose research activities has been supported in part by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476, and in part by the UMBC/LUCITE #32 grant. The findings and opinions expressed here are those of the authors, and do not necessarily reflect those of the above sponsoring organizations.

1 INTRODUCTION

One of the limitations of the current data mining and machine learning programs is that they employ relatively simple representation languages, e.g., decision trees, Bayesian nets, neural nets, etc., which limit their abilities as to the range of patterns they can discover. As a result, such programs may not be able to discover patterns that are cognitively simple but not easily representable by the program.

This paper concerns research on developing methodology for discovering different types of patterns represented in a highly expressive representation language, Attributional Calculus, that combines features of propositional, predicate and multiple-valued logic for the purpose supporting pattern discovery and concept learning (Michalski, 2004). The methodology has been implemented in AQ21, a program that performs *natural induction*, by which we mean an inference process that strives to generate accurate inductive hypotheses that are represented in human-oriented forms resembling natural language descriptions, and are by that easy to interpret and understand.

The rules learned by pattern discovery programs are usually conjunctions of atomic "attribute relation value" conditions (e.g. Grzymala-Busse, 2003; Pawlak, 1991; Setzkorn and Paton, 2005; Van Zyl and Cloete, 2004). Because of this constraint, these programs, as well as earlier ones, such as C4.5 (Quinlan, 1993), RIPPER (Cohen, 1995), CN2 (Clark and Niblett, 1989), are more limited than AQ21 as to the type of patterns they can discover in data. The AQ21 program also seamlessly integrates several new features either non-existent or present only individually and in a more limited form in other programs.

An important feature of AQ21 is that it can discover different types of regularities in data, depending on its parameter setting, such a conjunctive patterns, general rules with exceptions, consistent and complete data characterizations, and optimized alternative hypotheses.

In addition to reporting new features in AQ21, we also illustrate its performance and its differences from some well-known tree or rule discovery programs by applying it and the other programs to a simple problem. The AQ21 results are visualized using Generalized Logic Diagrams (GLDs).

Sections 2 and 3 present the AQ21 pattern discovery and learning methodology. Section 4 describes methods of reasoning with meta-values present in AQ21, and Section 5 describes how AQ21 generates alternative hypotheses generated from the same data. Section 6 describes AQ21's rule application and testing module, and Section 6 illustrates methods for visualizing AQ21-discoverd rules employing Concepts Association Graphs (CAGs). Section 7 presents selected experimental results.

2 AQ LEARNING

The learning problem considered here is to determine general hypotheses H1,…,Hk that describe classes (or concepts) C1,…,Ck, respectively, on the basis of training examples drawn from these classes. The AQ learning approach seeks hypotheses that optimize a given multi-criterion measure of hypothesis quality and are expressed in the form of *attributional rulesets*, defined as sets of attributional rules describing the same concept (Michalski, 2004).

A basic form of an attributional rule is:

CONSEQUENT <= PREMISE (1)

where both CONSEQUENT and PREMISE are conjunctions of attributional conditions (a.k.a. selectors), which take the form:

[L rel R : A] (2)

where *L* is an attribute, an internal conjunction or disjunction of attributes, a counting attribute, or a compound attribute; *rel* is one of =, :, >, <, ≤, ≥, or ≠, *R* is an attribute value, an internal disjunction of attribute values, an attribute, or an internal conjunction of values of attributes that are constituents of a compound attribute, and *A* is an optional annotation that lists *p* and *n* values for the condition, defined as the numbers of positive and negative examples, respectively, that satisfy the condition. Here is an example of an attributional rule:

```
[Activity=running_experiments]
     <=   [Day = weekend] & [Clock_speed >= 2GHz] &
          [Location = lab1 v lab3] & [Weather: quiet & warm]
```

which can be paraphrased: the activity is "running_experiments" if it is a weekend day (a higher-level concept of the structured attribute "day"), clock_speed of the computer is at least 2 GHz, the experiment takes place in lab1 or lab3, and the weather is quiet & warm. The attribute "weather" is an example of a *compound attribute*, a new type of attribute introduced in AQ learning that takes a conjunction of values of attributes that are typically used to describe an object or a component of an object (Michalski, 2004). Note that the attributional rule above closely corresponds to its equivalent natural language interpretation.

As shown in this example, an attributional rule in AQ learning uses a richer representation language than in a typical rule learning program, in which conditions are usually limited to the form:

[<*attr*> <*rel*> <*val*>] (3)

where <*attr*> is an attribute, <*val*> is an attribute value, and <*rel*> is a relation applicable to attr and val.

2.1 AQ21 Modes of Operation

AQ21 can be executed in three modes of operation, namely *Pattern Discovery* (PD), *Theory Formation* (TF), and *Approximate Theory Formation* (ATF). In PD mode, the program seeks strong patterns that do not have to be fully consistent with the data, and whose union may only partially cover the input data (may be partially incomplete). In TF mode, the program learns complete and consistent (CC) theories. This mode is used when it can be safely assumed that there are no errors in the data, and is applied usually to relatively small datasets. In ATF mode, the program first learns a CC theory and then optimizes it according to a given quality measure. The result is an approximate theory that may be partially inconsistent, and may not be fully complete.

This paper concentrates on the pattern discovery mode of AQ21. The other two will be discussed and analyzed in a future paper.

2.2 Pattern Discovery Mode

In PD mode, the program searches for strong patterns that maximize an assumed pattern quality measure. The method takes as input a set of positive examples E, a set of negative examples N, and a pattern quality measure, called LEF, defined by the user. It follows the general algorithm presented in Figure 1.

```
Hypothesis = null
While E is not empty
   Select a seed p from E
   Generate approximate star G(p, N)
   Select the best k rules from G according to LEF, and include in Hypothesis
   Remove from E all examples covered by the selected rules
Optimize final rules
Select the final hypothesis from all selected rules
```

Figure 1: The simplest form of PD mode in AQ21.

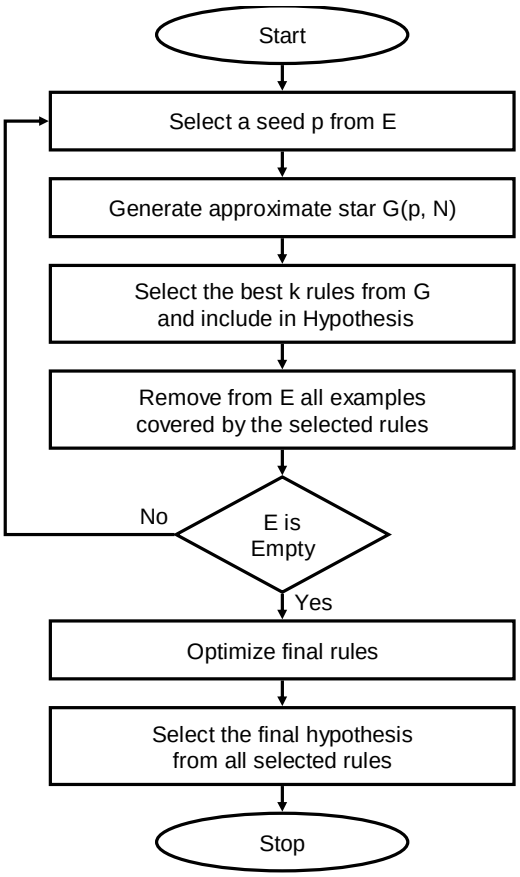


Figure 2: Top-level algorithm for the PD mode in AQ21.

The method starts by focusing attention on one data point, called the *seed*, and then creates a set of alternative approximate generalizations of the seed, called an *approximate star*. In PD mode, the generalizations are approximate, because they do not have to be consistent with the data; they must optimize a pattern quality criterion. The basic operator in star generation is *extension-against* (Michalski, 1983), which takes two data points and creates a set of maximal generalizations of one data point ("positive example") that does not cover the second data point (a "negative example'). The result of such an operation is a set of local stars. A logical intersection of local stars creates a star of the given seed. To narrow down a possibly large number of intermediate generalizations, AQ uses beam search that at each step of star generation keeps no more than a predefined number of best rules, as determined by the given pattern quality measure.

The pattern quality measure, Q(*w*), is defined by:

$$Q(w) = cov^{w} * consig^{1-w} \tag{4}$$

where

$$cov=p/P \tag{5}$$

and

$$consig=((p / (p + n)) - (P /(P + N))) * (P +N) /N \tag{6}$$

are measures of pattern (here, attributional rule) coverage and consistency gain, respectively, and *w* is a user-defined parameter. Here, *p* and *n* are the numbers of positive and negative examples covered by the rule, and *P* and *N* are the numbers of positive and negative examples in the training dataset (Michalski and Kaufman, 2001).

For selection of best rules AQ21 uses Lexicographical Evaluation Functional (LEF), a user-defined multicriterion measure of rule quality. The default LEF for selecting a rule from a set of candidates is to maximize Q(*w*), maximize the number of examples covered by the rule, and minimize the number of conditions in the rule. A complete list of LEF criteria available in AQ21 is presented in (Wojtusiak, 2004). Several features of AQ21 are designed to improve its efficiency and pattern quality. One feature is to use not just one, but several seeds for parallel star generation (to avoid situations in which the selected seeds are errors). In this case, the best rule is selected from all stars. Other features include selection of negative examples based on their distributions, selection of most relevant attributes prior to star generation, and optimized discretization of continuous attributes. These features are, however, beyond the scope of this paper.

It should be noted that other well-known rule learning methods, such as RIPPER (Cohen, 1995) and CN2 (Clark and Niblett, 1989), also seek strong patterns, but AQ21 can determine strong patterns optimizing a wide range of different pattern quality criteria, depending on the setting of its parameters. Because of its using a more expressive representation language, the patterns learned by AQ21 can be significantly richer and of different types, including complete and consistent (CC) rulesets, rulesets with exceptions, approximate CC rules, and weak patterns.

To graphically illustrate patterns that AQ21 can discover, we use *Generalized Logic Diagrams* (*GLDs*) (Michalski, 1978), (Sniezynski, Szymacha and Michalski, 2005) that provide a planar representation of the multidimensional representation space spanned over multiple-valued

discrete attributes. By properly ordering attributes assigned as axes of the diagram, patterns can be usually displayed in an easily understood form.

3 EXAMPLE PROBLEM

To illustrate AQ21 capabilities, we use a simple designed problem. It should be noted, however, that the program can work with datasets containing thousands or more of datapoints, each represented by hundreds of multitype variables.

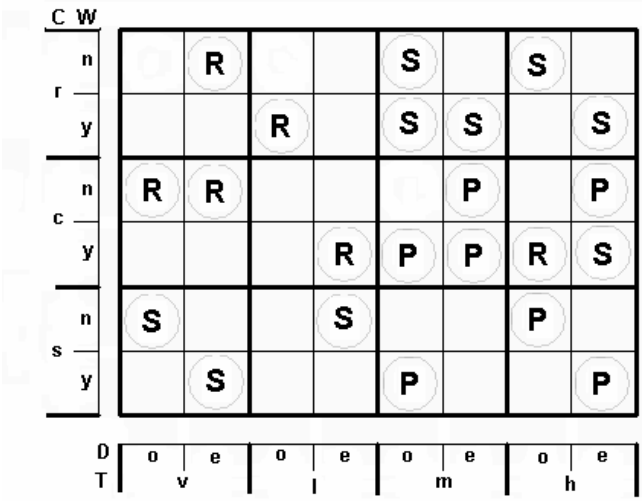
3.1 Problem Definition and Results

In this very simple example, datapoints (instances, examples) are defined using the attributes described in Figure 3.

```
condition   linear  {rain, cloudy, sunny}
wind        nominal {no, yes}
temperature linear  {very_low, low, medium, high}
daytype     nominal {workday, weekend}
activity    nominal {play, shop, read}
```

Figure 3: Attributes and their types and domains.

Suppose that our task is to determine strong patterns in examples for which the output attribute, *activity*, takes value "*play*". A GLD visualizing the representation space spanned over the input attributes and 22 input examples is presented in Figure 4.



Decision classes: P – Play, R – Read, S – Shop
Attributes: C – Condition, W – Wind, T – Temperature, D – Daytype,
Attribute values: r – rain, c – cloudy, s – sunny, n – no, y – yes, v – very low,
l – low, m – medium, h – high, o – workday, e – weekend

Figure 4: GLD with the example problem.

To compare the AQ21 performance with that of other methods, we also applied the well-known tree and rule learning programs, such as C4.5 (Quinlan, 1993), RIPPER (Cohen, 1995), and CN2 (Clark and Niblett, 1989) to this dataset. Figure 5 shows decision tree determined by C4.5.

```
condition = rain: shop (7.0/3.4)
condition = cloudy:
|   temperature = very_low: read (2.0/1.0)
|   temperature = low: read (1.0/0.8)
|   temperature = medium: play (3.0/1.1)
|   temperature = high: play (3.0/2.8)
condition = sunny:
|   temperature = very_low: shop (2.0/1.0)
|   temperature = low: shop (1.0/0.8)
|   temperature = medium: play (1.0/0.8)
|   temperature = high: play (2.0/1.0)
```

Figure 5: A decision tree determined by C4.5.

Figure 6 shows a set of rules derived by C4.5 for this problem. Both the decision tree and the rules learned by C4.5 were partially inconsistent with the data. The rules for the activity value "play" are partially explicit (first two rules) and partially implicit, defined by the "Default class: play", which means that if all rules fail, the chosen activity is "play."

```
condition = cloudy &
temperature = medium
->  class play  [63.0%]
condition = sunny
temperature = high
->  class play  [50.0%]
condition = rain
->  class shop  [51.2%]
condition = sunny
temperature = very_low
->  class shop  [50.0%]
temperature = very_low
->  class read  [35.2%]
temperature = low
->  class read  [31.4%]
Default class: play.
```

Figure 6: Rules derived by c4.5.

The RIPPER program applied to the same dataset determined the rules presented in Figure 7. These rules need to be evaluated sequentially, meaning that to obtain the hypothesis for activity "play", it is necessary to evaluate rules for activity "read."

```
read :- temperature=very_low (3/2).
read :- temperature=low (2/1).
play :- condition=sunny (3/0).
play :- condition=cloudy, wind=no (2/0).
Play :- condition=cloudy,temperature=medium (2/0)
default shop (6/1).
```

Figure 7: Rules found by RIPPER.

Figure 8 presents rules determined by the CN2 program applied to the same problem.

```
IF    condition = cloudy AND temperature = medium
THEN  activity = play  [3 0 0]
IF    condition = sunny AND temperature = high
THEN  activity = play  [2 0 0]
IF    condition = sunny AND temperature = medium
THEN  activity = play  [1 0 0]
IF    wind = no AND temperature = high AND
      daytype =weekend THEN activity =play[1 0 0]
IF    condition = rain AND temperature = medium
THEN  activity = shop  [0 3 0]
IF    condition = sunny AND temp = very_low
THEN  activity = shop  [0 2 0]
IF    condition = rain AND temperature = high
THEN  activity = shop  [0 2 0]
IF    wind = no AND temperature = low
THEN  activity = shop  [0 1 0]
IF    condition = cloudy AND wind = yes
  AND temperature = high AND daytype = weekend
THEN  activity = shop  [0 1 0]
IF    condition = cloudy
  AND temperature =very_low
THEN  activity = read  [0 0 2]
IF    wind = yes AND temperature = low
THEN  activity = read  [0 0 2]
IF    condition = rain AND temperature = very_low
THEN  activity = read  [0 0 1]
IF    wind = yes AND temperature = high
      AND daytype = workday
THEN  activity = read  [0 0 1]
(DEFAULT) activity = shop  [7 9 6]
```

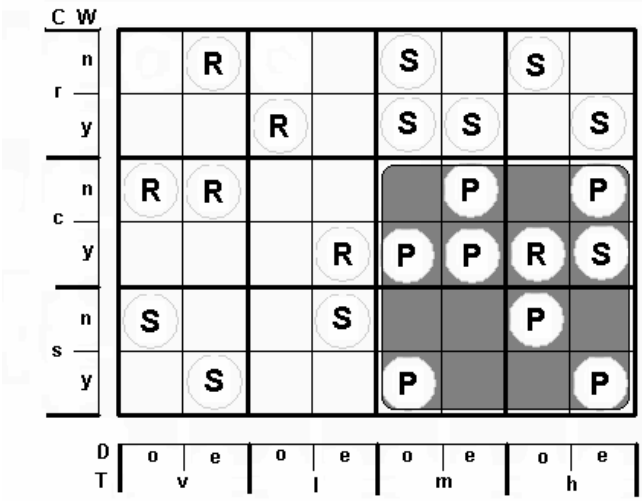
Figure 8: Rules determined by CN2.

AQ21 applied with default parameters to the same data determined one strong pattern presented in Figure 9.

```
[activity=play]
<= [condition=cloudy v sunny: 7,8] &
   [temperature=medium v high: 7,7]:  p=7,n=2,Quality=0.67
```

Figure 9: The strong pattern found by AQ21.

This pattern consists of one rule stating that the activity is play, if the weather is cloudy or sunny, and the temperature is medium or high. The rule covers 7 positive and 2 negative examples, and its quality, q(*w*), is 0.67, where *w* had the default value 0.5. Numbers inside conditions represent positive and negative coverages of the conditions alone. The discovered pattern is graphically illustrated in Figure 10 using a Generalized Logic Diagram.



Decision classes: P – Play, R – Read, S – Shop
Attributes: C – Condition, W – Wind, T – Temperature, D – Daytype,
Attribute values: r – rain, c – cloudy, s – sunny, n – no, y – yes, v – very low,
l – low, m – medium, h – high, o – workday, e – weekend

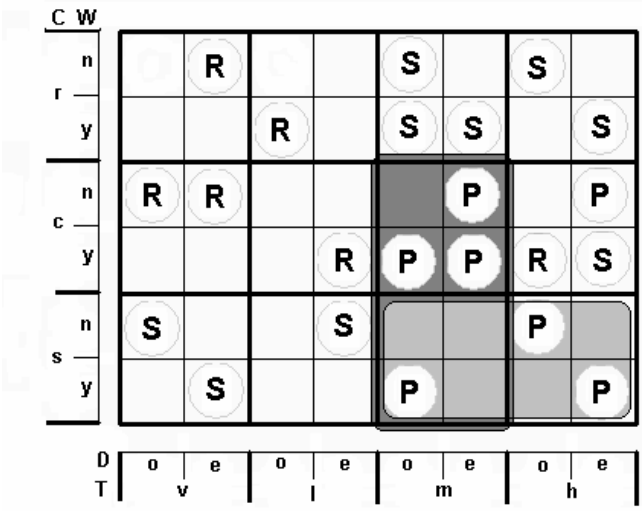
Figure 10: GLD with the strong pattern discovered by AQ21

AQ21 allows the user to control the tradeoff between completeness and consistency of patterns by adjusting parameter *w* in the pattern quality measure Q(*w*). When setting the *w* parameter to 0.15, AQ21 found two rules presented in Figures 11 and 12. Note that setting *w* to a value smaller than 0.5 places higher emphasis on rule confidence (or consistency) than on rule support (coverage) and setting *w* to a value greater than 0.5 has an opposite effect (Michalski and Kaufman, 2001).

The pattern is consistent but almost complete – one example is not covered by the learned rules. Note that further reduction of *w* to 0 leads to a complete and consistent ruleset, which can also be obtained by executing AQ21 in Theory Formation mode.

```
[activity=play]
<= [condition=cloudy v sunny: 7,8] &
   [temperature=medium: 4,3] :
   p=4,n=0,Quality=0.919
<= [condition=sunny: 3,3] &
   [temperature=medium v high:7,7]:
   p=3,n=0,Quality=0.881
```

Figure 11: Rules found by AQ21 for w=0.15.



Decision classes: P – Play, R – Read, S – Shop
Attributes: C – Condition, W – Wind, T – Temperature, D – Daytype,
Attribute values: r – rain, c – cloudy, s – sunny, n – no, y – yes, v – very low,
l – low, m – medium, h – high, o – workday, e – weekend

Figure 12: GLD with the pattern found by AQ21 for w=0.15.

3.2 Learning Rules with Exceptions

The concept of an "exception" is commonly used by human experts when describing rarely occurring anomalies in described phenomena. It is not unusual that a simple theory may work well for most cases, but turning it into a fully consistent and complete theory would require making it significantly more complex. In such cases, it is desirable to learn rules with exceptions e.g. (Michalski, 2004; Yao et al., 2004). AQ21 can be set to learn rules with exceptions in the following form:

$$\text{CONSEQUENT} <= \text{PREMISE} \mid\_ \text{EXCEPTION} \qquad (7)$$

where EXCEPTION is either an attributional conjunctive description, or a list of examples constituting exceptions to the rule. Note that exceptions in such rules are always negative examples. The processes of learning exceptions in TF and PD modes are different. In PD mode, where inconsistency is allowed, the program learns standard patterns and generates exceptions representing covered negative examples by finding a conjunctive description of the negative examples using AQ learning.

In TF mode, where consistency is guaranteed, the program adds negative examples to the list of exceptions if such examples are infrequent but would introduce significant complexity to a description that does not cover them. If all of the exceptions can be characterized by one conjunctive description, such a description is used as EXCEPTION clause in the rule; otherwise, an explicit list of exceptions is output.

AQ21 applied to the same data produced the rule with an exception presented in Figure 13.

```
[activity=play]
<= [condition=cloudy v sunny: 7,8] &
   [temp= medium v high: 7,7]
  |_ [condition=cloudy] & [wind=yes]&[temp= high]
   : p=7,n=0,Quality=1
```

Figure 13: The strong pattern with exception found by AQ21.

The rule states that activity is play if weather is cloudy or sunny and temperature is medium or high, unless weather is cloudy, there is wind, and temperature is high.

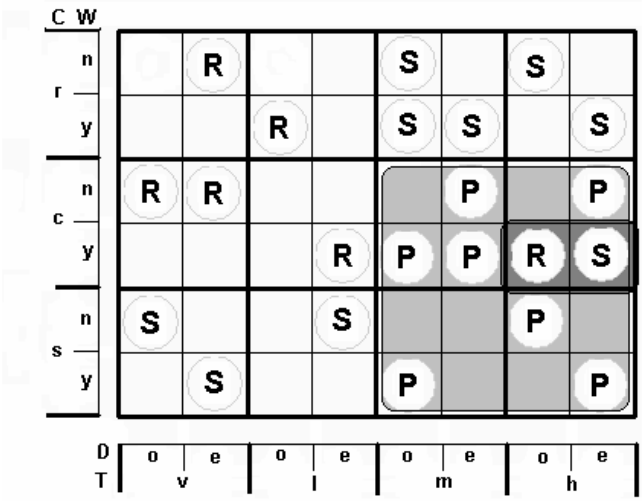
Figure 14 shows a GLD representation of the rule presented in Figure 13. The two highlighted examples representing "shop" (S) and "read" (R) activities are treated as exceptions to the general pattern for the play activity.

AQ21 generalized the two exceptions into one conjunctive description [weather=cloudy] & [wind=yes] & [temperature=high].

In the case that the two examples could not be generalized into one conjunctive description, the program would have listed them explicitly:

cloudy, yes, high, yes, shop
cloudy, yes, high, no, read

This simple example shows that the introduction of rules with exceptions can produce descriptions that are simple, accurate and comprehensible, while descriptions without exception are more complex, less accurate, and less comprehensible.



Decision classes: P – Play, R – Read, S – Shop
Attributes: C – Condition, W – Wind, T – Temperature, D – Daytype,
Attribute values: r – rain, c – cloudy, s – sunny, n – no, y – yes, v – very low,
l – low, m – medium, h – high, o – workday, e – weekend

Figure 14: GLD with the strong pattern with exception found by AQ21.

3.3 Discussion of Results

The C4.5-generated decision tree has 3 internal nodes and 11 branches, and when applied to the training dataset (22 examples) gave 4 errors (18%). The C4.5 rules (7 rules) gave 5 errors (23%). RIPPER produced 6 rules that gave 4 errors (18%) on the training dataset. CN2 produced 13 rules the constituted a complete and consistent description, but by far the most complex.

In PD mode using the default value of *w* (0.5), AQ21 produced one strong pattern that covered all positive examples and 2 examples of other classes (9% error). When executed with *w*=0.15, it produced a consistent and nearly-complete description, as it missed one example of activity play (4.5% error). In TF mode, AQ21 generated three simple rules that were complete and consistent with the training dataset (0% errors) (results are presented in Section 5). AQ21 with exceptions found one strong pattern that was complete and consistent with regard to the training data (0% errors). It may additionally be noted that none of the compared programs, except AQ21, can determine patterns describing only one class.

In this simple example, AQ21 produced both more accurate and simpler patterns than the other programs. This result was achieved due to the richer representation language used by this program. The AQ21's more expressive representation language facilitates generation of simpler hypotheses in forms that are easy to interpret in natural language. AQ21 also allows the user to control the type of output by parameters and the hypothesis quality criterion. The latter feature is useful when different problems may require different criteria of pattern optimality.

4 HANDLING META-VALUES IN DATA

Another important new feature of AQ21 is its ability to learn from examples that may have *meta-values*, which can be of three types: *unknown*, *irrelevant*, and *not-applicable*. The *unknown* meta-value means that the attribute is missing the corresponding value in the example. The *irrelevant* meta-value indicates that an expert has determined that the value of the corresponding attribute in the example is not relevant for the given classification task. For example, the last name of the broker's barber, even if available in the data, can be declared as irrelevant to predicting the value of a stock tomorrow. Finally, it is possible that some attributes in the data may apply to some, but not all the entities. For example, in a library inventory dataset, the "number of pages" attribute applies to books, but not to the chairs in the library.

To represent these cases, the Attributional Calculus representation employed in AQ learning assumes that the domain of every attribute includes the three meta-values in addition to its regular values (Michalski, 2004). These meta-values correspond to three possible answers to a question requesting an attribute value in situations in which a regular value is not provided. Specifically, these are "unknown" (?), "not applicable" (NA), and "irrelevant" (*) values. Formally, to each attribute x whose domain is D, AQ21 assigns a domain D' = D $\cup$ {?, NA, *}. These meta-values are handled by AQ21 appropriately for their type.

In the literature, the problem of handling meta-values is well-known and described by many authors. However, most authors do not distinguish between different types of meta-values. Typically, all missing values are treated as "Unknown." Some authors investigated meta-values other than "?." For example, Grzymala-Busse (2003) presented a method for dealing with

unknown and irrelevant values in rough sets. This method is different from that in AQ21, as it learns standard rules with (attribute rel value) conditions, and does not consider not-applicable values. It is based on internal operations used in rough sets which are different than extension-against used in AQ learning. Programs such as C4.5 and CN2 handle missing values by replacing events with unknown and irrelevant (CN2 only) values by their copies with filled all possible values of the missing attributes. Below is a description of the methods for handling these three meta-values in AQ21.

4.1 Handling Unknown Values

Unknown (a.k.a. "Don't know" or "Missing"), denoted by a "?", is given to an attribute whose value for a given entity exists, but is not available for some reason. For example, the attribute may not have been measured for this entity, or may have been measured, but not recorded in the database. In such situations, the meta-value "?" in inserted in the training and/or testing datasets for this attribute in the event characterizing the given entity.

Filling (or preprocessing) methods for handling unknown values are applied to datasets before starting the learning process. After they have been applied, there is no need for modifying the regular AQ learning algorithm, and matching operators to handle missing meta-values. Three preprocessing methods are used in AQ learning: (P1) ignoring training events with "?," (P2) replacing "?" by the average value (for numerical attributes) or the most frequent value (for nominal attributes) in the s most similar training and/or testing events, where s is a program parameter, and (P3) learning rulesets for determining the values of the attributes with missing values in the training dataset, and then using those rulesets to predict the missing values when learning rules for other output attributes. Similar methods have been widely investigated in the literature (Burcha, 2004), (Ragel and Cremeilleux, 1999).

Two internal methods for handling Missing values are implemented in AQ21: (L1) ignoring the extension-against operation for attributes with missing values in such events during the star generation, and (L2) treating "?" as a regular value in the events, but not using events with a "?" for seeds. When extending a seed against a missing value, create a selector: [xi $\neq$?], regardless of the value of attribute xi in the seed. Readers not familiar with extension-against operation used in AQ learning may refer to, for example, (Michalski, 1983).

For all events that cannot be matched with a rule with a specific degree of match because of the presence of a missing attribute value in the event, determine two values of p, p_{min} and p_{max}, and two values of n, n_{min} and n_{max}, respectively. p_{min} and n_{min} are computed by assuming that the unknown value does not match the rule, and p_{max} and n_{max} are computed by assuming that it does match. The positive rule coverage is characterized by the range (p_{min} .. p_{max}), and the negative coverage by the range (n_{min} .. n_{max}). It should be noted that the TF algorithm uses p_{min} and n_{max} values to ensure completeness and consistency of learned hypotheses.

4.2 Handling Not-applicable Values

Not-applicable, denoted by an "NA," is given to an attribute that is not applicable to a given entity, because the value does not exist. For example in a library inventory database a "number of pages" attribute does not apply to a chair in the library, but it applies to the books.

If a dataset has "not-applicable" values for some attributes, the attributes are removed from all events with that value, but not removed from other events when executing the extension-against operator, regardless of whether they are positive or negative events. This operation is justified by the "NA" semantics, according to which, asking for a value of the attribute of an entity for which an attribute is not applicable is meaningless.

If a training event has a "not-applicable" value for some attribute, the attribute is removed from the event when determining the rule coverage during the learning process. Therefore, the event does not match the rule if the rule references the NA attribute.

4.3 Handling Irrelevant Values

Irrelevant values, denoted by an "*", indicate that an attribute is considered irrelevant to the learning problem, or for the concept (class) to be learned, or in the particular event. Consequently, three types of irrelevant attributes are distinguished, task-irrelevant, class-irrelevant, and event-irrelevant.

An attribute is task-irrelevant if it is irrelevant for the entire learning problem. For example, a student's hair color can be declared as irrelevant for learning rules for classifying students into groups representing their academic performance. An attribute is class-irrelevant if it is irrelevant for a given class (value of the output attribute), but may relevant for other classes. For example, the patient's PSA (prostatic specific antigen) level is relevant for diagnosing prostate diseases, but is irrelevant for diagnosing eye diseases.

Task-irrelevance and class-irrelevance of an attribute can be handled in preprocessing by removing columns from a dataset. An attribute is event-irrelevant if it is irrelevant only for a particular event in the class to be learned. For example, the attribute "stock price" is relevant to any event in the class "stocks_to_acquire," but in a particular instance when it is the stock of company you work for and is given free to employees, it may be considered irrelevant. This kind of irrelevance represents knowledge of an expert who for some reason decides that an attribute should not be used to classify a particular event.

The task-irrelevance is handled by simply removing the attribute in question from the training and testing datasets. The class-irrelevance is handled by removing the attribute from training dataset for the given class, but it remains in dataset when learning classes for which it is relevant. Therefore, only the problem of handling event-irrelevant attributes needs to be considered. If an attribute is indicated as irrelevant in one or more events of a given class, but indicated as relevant for other events, that is, it is irrelevant for one or more combinations of values of other attributes, but not for all combinations, then in executing the extension against operator, the program ignores the attribute with the value "*" in events with that value, but does not ignore it in other events.

If an event with some attributes indicated as irrelevant is matched against an attributional rule, this attribute is removed from the event. This is equivalent to asserting that the irrelevant value always matches a condition with this attribute.

5 ALTERNATIVE HYPOTHESES

From any non-trivial set of concept examples, it is usually possible to generate many alternative inductive generalizations of these examples. Such alternative hypotheses can be useful for a variety of practical applications of computational learning systems. For example, in medical decision making (diagnosis, drug prescription, or therapy assignment), some tests required by a given diagnostic procedure may be unavailable, and an alternative procedure would be necessary.

Alternative hypotheses can also be used to increase the accuracy of classification decisions. This can be done through simple voting on decisions assigned by different hypotheses, or by weighted voting, as is typically done in boosting (Schapire and Singer, 1999).

Formally, the problem of learning alternative hypotheses is to generate a set of hypotheses H_i^1, …, H_i^k for class C_i given the set of examples e_1, .., e_n belonging to classes C_1, …, C_m. Alternative hypotheses H_i^1, …, H_i^k are optimized according to user specified criteria. The algorithm uses the idea that different alternative hypotheses can be selected from the final rules (in the last step of the algorithm in Figure 1). AQ21 not only seeks alternative hypotheses, but also seeks hypotheses optimized according to user-defined criteria that may reflect different aspects of the problem.

To illustrate how AQ21 generates alternative hypotheses, it was applied in TF mode to the example dataset presented in Figure 3. It found alternative hypotheses shown in Figures 15 and 16.

```
[activity=play]
<= [condition=cloudy v sunny: 7,8] &
   [temperature=medium: 4,3] : p=4,n=0
<= [condition=sunny: 3,3] &
   [temperature=medium v high: 7,7] : p=3,n=0
<= [condition=cloudy v sunny: 7,8] &
   [wind=no:3,7]&[temperature=medium v high:7,7]
    : p=3,n=0
```

Figure 15: First alternative hypothesis learned in TF mode.

```
[activity=play]
<= [condition=cloudy v sunny: 7,8] &
   [temperature=medium: 4,3] : p=4,n=0
<= [condition=sunny: 3,3] &
   [temperature=medium v high: 7,7] : p=3,n=0
<= [condition=cloudy v sunny: 7,8] &
   [wind=no: 3,7]
   [temperature=medium v high: 7,7]  : p=3,n=0
```

Figure 16: Second alternative hypothesis leaned in TF mode.

Because the program was executed in TF mode, both hypotheses are complete and consistent with regard to the training data. The first two rules of the hypotheses are the same and cover the majority of the positive examples. The hypotheses differ in the way the last example is covered by the third rule.

A simple example illustrating generation of alternative hypotheses in AQ learning is presented in the iAQ program specifically developed for demonstrating pattern discovery and natural induction. The program can be downloaded from http://www.mli.gmu.edu/msoftware.html.

6 TESTING OF RULESETS

For testing and application attributional rulesets AQ21 implements two methods: ATEST and EPIC. The ATEST method (Reinke, 1984; Wojtusiak, 2004) is used for the testing and application of attributional rulesets to individual testing/application examples. EPIC (Wojtusiak, 2004) similarly applies attributional rulesets to *episodes*¸ that is, sequences of examples to be classified as a whole. For example, in the application to computer user profiling, we are not interested in identifying the user responsible for each individual command in the database; rather, we have been provided a sequence of commands bundled together, for which we wish to establish the responsible user. Thus, the sequence is considered together as an episode.

The general methodology of ATEST and EPIC is as follows:

1) For each individual testing example, determine a degree of match between it and each decision rule.

2) For each decision class, aggregate the degrees of match determined in step 1, in order to determine degrees of match between each testing example and each decision class.

3) If using EPIC, aggregate the degrees of match determined in step 2 for the examples in each episode, in order to determine degrees of match between each episode and each decision class.

4) Based on the calculated degrees of match and threshold and tolerance parameters, output a classification for each testing example (ATEST) or episode (EPIC). Please note that both methods, in addition to standard accuracy based on best match, are able to classify an event (episode) to more than one class. In many real world applications it is more appropriate to give an imprecise classification rather than to give a wrong answer. For example, when diagnosing diseases, the program may give the answer in form of a list of possible diseases.

There are several matching and aggregation methods available in steps 1-3. Ones appropriate to the task may be selected by the user (Wojtusiak, 2004).

7 VISUALIZING RULESETS BY CAG

An important aspect of a natural induction program is to be able to graphically present learned knowledge. Two methods of visualizing knowledge have proven particularly useful for representing rulesets learned by AQ21: Generalized Logic Diagrams, described in Section 2 and in, e.g., (Michalski, 1978), and Concept Association Graphs (CAGs) (Kaufman and Michalski, 2000).

CAGs represent higher level concepts – the general relationships among attributes and/or features. The links can be annotated visually (by thickness or color) and symbolically (through abstract and numeric annotations). Such a multimode presentation allows for the transmission of

a high quantity of knowledge in a relatively simple diagram. An example of a CAG for the pattern discussed earlier and shown in Figure 10 is presented in Figure 17.

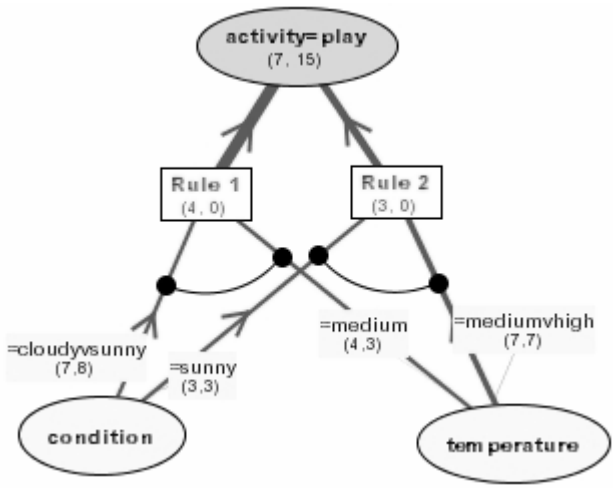


Figure 17: CAG with two rules found by AQ21.

8 SELECTED EXPERIMENTAL RESULTS

This section presents a selected results of applying AQ21, C4.5 and C4.5 Rules (Quinlan, 1993), CN2 (Clark and Niblett, 1989), and RIPPER (Cohen, 1995) programs to discover patterns in two real-world datasets and one designed dataset described below.

The *Volcanoes* dataset, provided by the Smithsonian institution, contains information about 20,000 volcanoes and their eruptions around the world. Goal of the learning is to distinguish between volcanoes that cause fatalities from those which do not. The dataset was split into 13,757 training and 5,846 testing examples.

Provided with the Volcanoes dataset AQ21 in the PD mode learned 12 rules, selected of which are presented in Figure 18.

```
[Fatalities=present]
<= [A1<>secondary_mudflow,none: 438,28] : p=438,n=28,Quality=0.83   (Rule 1)
<= [Pyroclastic=present: 311,1573] &
   [Damage=present: 561,1168] : p=299,n=215,Quality=0.52            (Rule 3)
<= [Evacuation=present: 285,365] : p=285,n=365,Quality=0.417        (Rule 10)

[Fatalities=absent]
<= [A1=none: 13052,203] : p=13052,n=203,Quality=0.778               (Rule 2)
```

Figure 18: Selected rules learned by AQ21 for the Volcanoes dataset.

Selected rules (out of 56) learned by c4.5 rules (from decision tree) for the Volcanoes dataset are presented in Figure 19. Selected rules (out of 32) learned by RIPPER for the Volcanoes dataset are presented in Figure 20. Selected rules (out of 190) learned by CN2 for the Volcanoes dataset are presented in Figure 21.

```
Rule 48:
      Pyroclastic = present
      Evacuation = present
      Subregion = Honshu-Japan
      ->  class present  [70.7%]
Rule 71:
      Flank_vent = present
      Damage = present
      Latx > 38.5667
      Longx <= 106.983
      ->  class present  [72.2%]
Rule 96:
      Pyroclastic = absent
      Phreatic = absent
      Lava_dome = absent
      Damage = present
      Evacuation = absent
      Subregion = Sulawesi-Indonesia
      Upper1 > 600
      ->  class present  [50.0%]
```

Figure 19: Selected rules learned by c.4.5 for the Volcanoes dataset.

```
present_c :- Damage=present, Pyroclastic=present, A1=pyroclastic_flow (55/1).
present_c :- Damage=present, A1=mudflow (64/0).
present_c :- A1=indirect, Longx<=-86.7019 (3/0).
default absent_c (13074/57).
```

Figure 20: Selected rules learned by RIPPER for the Volcanoes dataset.

```
IF    submarine_erupt = absent
  AND a1 = tephra
THEN  class = present  [154 0]

IF    a1 = pyroclastic_flow
  AND latx < 47.54
THEN  class = present  [81 0]
IF    damage = present
  AND lahars = absent
  AND a1 = none
  AND subregion = Java
  AND latx > -7.97
  AND upper1 > 1600.00
THEN  class = absent  [0 28.67]
IF    subregion = Africa-W
THEN  class = absent  [0 29]

(DEFAULT) class = absent  [661 13096]
```

Figure 21: Selected rules learned by CN2 for the volcanoes dataset.

The *World Factbook* dataset, downloaded from the CIA website, contains information about 266 countries around the world (2 classes of countries -- those with birth rates above and below 20). The dataset was split into 196 training and 70 testing examples.

For the World Factbook dataset AQ21 in TF mode learned 6 rules (3 for each class) shown in Figure 22.

```
[Birth_rate<=20]
<= [Total_fertility<=2.265 : 71,0] : p=71,n=0              (Rule 1)
<= [Imports>=6.745e+07 : 74,77] &
   [Total_fertility<=2.485 : 79,2] &
   [Unemployment<=19.5 : 68,57] &
   [Telephones_cellular>=988 : 72,74] : p=59,n=0           (Rule 2)
<= [GDP_per_capita>=5050 : 65,16] &
   [Death_rate>=3.005 : 80,80] &
   [Infant_mortality<=18.79 : 63,7] &
   [Labor_force>=1.925e+04 : 74,80] &
   [Integerernet_users>=1.9e+04 : 56,51] : p=43,n=0        (Rule 3)

[Birth_rate>20]
<= [GDP_per_capita<=1.475e+04 : 80,41] &
   [Population>=3.807e+04 : 81,74] &
   [Total_fertility>=2.44 : 81,3] : p=78,n=0               (Rule 1)
<= [Total_fertility>=2.76 : 75,0] : p=75,n=0               (Rule 2)
<= [Electricity_prod>=1.854e+09 : 36,59] &
   [Industrial_production_growth>=3.45 : 33,49] &
   [Infant_mortality=48.6..57.94 : 9,1] : p=3,n=0          (Rule 3)
```

Figure 22: Rules learned by AQ21 for the World Factbook dataset.

Decision tree learned by C4.5 for the World Factbook dataset are presented in Figure 23. Rules learned by C4.5rules for the World Factbook dataset are presented in Figure 24. Rules learned by RIPPER for the World Factbook dataset are presented in Figure 25. Rules learned by CN2 for the World Factbook dataset are presented in Figure 26.

```
Total_fertility > 2.5 : yes (80.0/1.0)
Total_fertility <= 2.5 :
|   Total_fertility <= 2.25 : no (71.0)
|   Total_fertility > 2.25 :
|   |   GDP_per_capita <= 6500 : yes (4.0/1.0)
|   |   GDP_per_capita > 6500 : no (8.0)
```

Figure 23: Decision tree learned by C4.5 for the World Factbook dataset.

```
Rule 1:
      Total_fertility <= 2.25
      ->  class no  [98.1%]

Rule 3:
      GDP_per_capita > 6500
      Total_fertility <= 2.5
      ->  class no  [97.8%]
```

```
Rule 4:
      Total_fertility > 2.5
      ->  class yes  [96.8%]

Rule 2:
      GDP_per_capita <= 6500
      Total_fertility > 2.25
      ->  class yes  [96.5%]

Default class: yes
```

Figure 24: Rules learned by C4.5rules for the World Factbook dataset.

```
no :- Total_fertility<=2.43 (78/1).
default yes (81/3).
```

Figure 25: Rules learned by RIPPER for the World Factbook dataset.

```
IF    GDP_per_capita < 14750.00
  AND Total_fertility > 2.51
THEN  Birthrate_gt20 = yes  [77 0]

IF    Infant_mortality < 16.61
  AND Inflation < 3.10
  AND Total_fertility > 2.44
THEN  Birthrate_gt20 = yes  [6.50 0]

IF    GDP_growth_rate < 4.60
  AND Infant_mortality > 54.20
THEN  Birthrate_gt20 = yes  [36 0]

IF    GDP_growth_rate > 3.30
  AND Total_fertility > 2.49
THEN  Birthrate_gt20 = yes  [38 0]

IF    Infant_mortality < 54.20
  AND Total_fertility < 2.44
THEN  Birthrate_gt20 = no  [0 75]

IF    GDP_per_capita > 8950.00
  AND Total_fertility < 2.89
THEN  Birthrate_gt20 = no  [0 54.50]

IF    GDP_growth_rate > 4.60
  AND Total_fertility < 2.39
THEN  Birthrate_gt20 = no  [0 26]

(DEFAULT) Birthrate_gt20 = yes  [82 81]
```

Figure 26: Rules learned by CN2 for the World Factbook dataset.

The *designed* dataset consists of 2000 entities described by one structured, one 6-valued nominal, and 12 binary, input attributes. The output attribute is binary; its values denote two classes, describing 1000 training and 1000 testing examples (500 positive and 500 negative examples in each set). The positive examples are of the concept “the number of present Features

is 2 or 3 and Shape is oval or Color is red." Definitions of attributes and their domains in AQ21 format are presented in Figure 27.

AQ21 provided with the training data correctly learned the target concept by producing rules presented in Figure 28. The second condition in the first rule uses an automatically discovered counting attribute meaning that the number of Features from Feature1 .. Feature10 that are present are between 2 and 3. Please note that the value "oval" in the first condition is a higher level concept of the attribute "Shape."

```
Common domains
{
  feature nominal {absent, present}
}

Attributes
{
  Feature1 feature
  Feature2 feature
  Feature3 feature
  Feature4 feature
  Feature5 feature
  Feature6 feature
  Feature7 feature
  Feature8 feature
  Feature9 feature
  Feature10 feature
  Shape structured {square, rectangle, ellipse, circle, right_triangle,
                    obtuse_triangle, accute_triangle, rectangular, oval,
                    triangular}
                { rectangular <-- right_triangle
                  rectangular <-- obtuse_triangle
                  rectangular <-- accute_triangle
                  oval <-- circle
                  oval <-- ellipse
                  rectangular <-- square
                  rectangular <-- rectangle }
  Color nominal {red, green, blue, white, orange, black}
  Class nominal {positive, negative}
}
```

Figure 27: Definition of attributes for the designed problem.

```
[Class=positive]
   <= [Shape=oval: 422,121] &
      [Count(Feature1, Feature2, Feature3, Feature4, Fearure5, Feature6,
            Feature7, Feature8, Feature9, Feature10 = present)=2..3: 415,103]
     : p=402,n=0
   <= [Color=red: 165,0]
     : p=165,n=0
```

Figure 28: Rules learned by AQ21 for the designed problem.

Selected rules learned from decision tree by c4.5 for the designed problem are presented in Figure 29. Selected rules learned by CN2 for the designed problem are presented in Figure 30. Rules learned by RIPPER for the designed problem are presented in Figure 31.

A summary of results in terms of accuracy on testing datasets and numbers of rules is presented in Table 1.

```
...
Rule 29:
      Feature1 = absent
      Feature3 = absent
      Feature4 = present
      Shape = ellipse
      ->  class positive  [95.8%]
Rule 39:
      Feature2 = absent
      Feature3 = absent
      Feature4 = present
      Feature6 = absent
      Feature9 = absent
      Shape = circle
      ->  class positive  [94.4%]
Rule 51:
      Feature5 = absent
      Feature8 = absent
      Shape = circle
      ->  class positive  [91.8%]
...
```

Figure 29: Selected rules learned by c4.5 for the designed problem.

```
...
IF    Feature5 = present
  AND Feature6 = present
  AND Feature10 = absent
  AND Shape = circle
  AND Color = blue
THEN  class = positive  [1 0]

IF    Feature2 = present
  AND Feature7 = present
  AND Feature10 = present
  AND Shape = square
THEN  class = negative  [0 19]

IF    Feature1 = present
  AND Feature2 = present
  AND Feature4 = present
  AND Feature6 = present
  AND Feature8 = present
  AND Feature9 = present
THEN  class = negative  [0 87]
```

```
IF    Feature5 = absent
  AND Feature7 = absent
  AND Feature10 = absent
  AND Shape = obtuse_triangle
THEN  class = negative  [0 20]
...
```

Figure 30: Selected rules learned by c4.5 for the designed problem.

```
positive :- Color=red (165/0).
positive :- Shape=ellipse, Feature3=absent, Feature1=absent (89/3).
positive :- Shape=circle, Feature2=absent, Feature4=absent (94/9).
positive :- Shape=ellipse, Feature4=absent, Feature5=absent (56/4).
positive :- Shape=circle, Feature5=absent, Feature8=absent (47/3).
positive :- Shape=circle, Feature6=absent, Feature9=absent, Feature7=absent
(19/0).
positive :- Shape=ellipse, Feature10=absent, Feature6=absent, Feature7=absent
(23/2).
positive :- Shape=circle, Feature1=absent, Feature9=absent, Feature3=absent
(4/1).
default negative (478/3).
```

Figure 31: Selected rules learned by c4.5 for the designed problem.

Dataset	**Accuracy % / Number of Rules**				
	AQ21	**C4.5**	**C4.5 Rul.**	**CN2**	**RIPPER**
Volcanoes	99.45 % 12 Rules	98.8 % Decision tree with 120 nodes	99 % 66 Rules	95.5 % 190 Rules	98.96 % 33 Rules
World Factbook	94.29 % 6 Rules	91.9 % Decision tree with 6 nodes	91.9 % 5 Rules	93.5 % 7 Rules	93.55 % 2 Rules
Designed	100 % 2 Rules	92.1 % Decision tree with 83 nodes	92.7 % 46 Rules	91.7 % 164 Rules	92.8 % 8 Rules

Table 1: Accuracies and numbers of rules of five methods on three datasets.

As shown in Table 1, AQ21 performed better in terms of both accuracy and complexity (numbers of rules). The main reason for this can be attributed to the richer representation language that allowed the program to determine a more accurate concept description.

9 SUMMARY

The paper presented several novel features seamlessly integrated in the AQ21 pattern discovery and rule learning program. These features extend the capability of AQ21 and aim at making it a natural induction program that equally stresses the accuracy and comprehensibility of discovered knowledge. The features include discovering attributional patterns without and with exceptions, discovering alternative hypotheses in the same data, and handling meta-values. Additionally, the KV and CAG methods for visualizing learned knowledge are designed to work with knowledge

learned by AQ21. The above features are either not present in other learning programs or present only separately in different programs.

Although we illustrated novel features of AQ21 on simple examples, the program is highly efficient and has been applied to problems with thousands of examples and hundreds of multi-type attributes. AQ21 has also several other features not described here that were designed to improve the efficiency of the AQ method.

Current research concerns systematic testing of the AQ21 program, its extension by adding new features, and its experimental application to complex real-world data mining problems.

REFERENCES

Brucha, I., "Meta-Learner for Unknown Attribute Values Processing: Dealing with Inconsistency of Meta-Databases," *Journal of Intelligent Information Systems*, 22:1, pp. 71-87. 2004.

Clark, P. and Niblett, T., "The CN2 Induction Algorithm," *Machine Learning* 3: pp. 261-289. 1989.

Cohen, W., "Fast Effective Rule Induction," *Proceedings of the 12th International Conference on Machine Learning*. 1995.

Grzymala-Busse J.W., "Rough Set Strategies to Data with Missing Attribute Values," *Proceedings of the Workshop on Foundation and New Directions in Data Mining*. Melbourne, FL, USA. 2003.

Kaufman, K. and Michalski, R.S., "A Knowledge Scout for Discovering Medical Patterns: Methodology and System SCAMP," *Proceedings of the Fourth International Conference on Flexible Query Answering Systems, FQAS'2000*, Warsaw, Poland, pp. 485-496. 2000.

Michalski, R.S., "Graphical Minimization of Normal Expressions of Logic Functions using Tables of the Veitch-Karnaugh Type," *Journal of the Institute of Automatic Control*, (52). 1978

Michalski, R.S., "A Theory and Methodology of Inductive Learning," In R.S. Michalski, J. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto: Tioga Publishing Co. 1983.

Michalski, R.S., "ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Natural Induction," *Reports of the Machine Learning and Inference Laboratory*, MLI 04-2, George Mason University. 2004.

Michalski, R.S. and Kaufman, K., "Learning Patterns in Noisy Data: The AQ Approach," In G. Paliouras, V. Karkaletsis and C. Spyropoulos, (Eds.) *Machine Learning and its Applications*, Springer-Verlag, pp. 22-38. 2001.

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

Quinlan, J.R. *C4.5 Systems for Machine Learning*. Morgan Kaufmann Publishers Inc. 1993.

Ragel, B. and Cremilleux, B., "MVC - A Preprocessing Method to Deal with Missing Values," *Knowledge-Based Systems*, 12, pp. 285-289. 1999.

Reinke, R., “Knowledge Acquisition and Refinement Tools for the ADVISE META-EXPERT System,” *Reports of the Intelligent Systems Group*, ISG 84-6, UIUCDCS-F-84-926, Department of Computer Science, University of Illinois, Urbana, 1984.

Schapire, R. and Singer, Y., “Improved Boosting Algorithms Using Confidence-rated Predictions,” *Machine Learning*. 37. Kluwer Academic Publishers, pp.297-336. 1999.

Setzkorn C. and Paton R.C., “On the use of multi-objective evolutionary algorithms for the induction of fuzzy classification rule systems,” *Biosystems*, 81, 2. 2005.

Sniezynski, B., Szymacha, R. and Michalski, R. S., “Knowledge Visualization Using Optimized General Logic Diagrams,” *Proceedings of the Intelligent Information Processing and Web Mining Conference*, IIPWM 05, Gdansk, Poland, June 13-16, 2005.

Van Zyl, J. and Cloete, I., “Simultaneous Concept Learning of Fuzzy Rules,” *Proceedings of the Fifteenth European Conference on Machine Learning*, Pisa, Italy, pp. 548-559. 2004

Wojtusiak, J. “AQ21 User’s Guide,” *Reports of the Machine Learning and Inference Laboratory*, MLI 04-5, George Mason University. 2004.

Yao, Y., Wang, F., Zheng, D., and Wang, J., “Rule + Exception Strategies for Security Information Analysis,” *IEEE Intelligent Systems*, 20, pp. 52-57. 2004.

A publication of the *Machine Learning and Inference Laboratory*
College of Science
George Mason University
Fairfax, VA 22030-4444 U.S.A.
http://www.mli.gmu.edu

Editor: R. S. Michalski
Assistant Editor: K. A. Kaufman

The *Machine Learning and Inference (MLI) Laboratory Reports* are an official publication of the Machine Learning and Inference Laboratory, which has been published continuously since 1971 by R.S. Michalski's research group (until 1987, while the group was at the University of Illinois, they were called ISG (Intelligent Systems Group) Reports, or were part of the Department of Computer Science Reports).