Learning Patterns in Noisy Data: The AQ Approach

Ryszard S. Michalski* and Kenneth A. Kaufman

Machine Learning and Inference Laboratory, M.S. 4A5, George Mason University,
FAIRFAX, VA 22030, USA
*also at the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
{michalsk, kaufman}@mli.gmu.edu

1. Introduction

In concept learning and data mining, a typical objective is to determine concept descriptions or patterns that will classify future data points as correctly as possible. If one can assume that the data contain no noise, then it is desirable that descriptions are complete and consistent with regard to all the data, i.e., they characterize all data points in a given class (positive examples) and no data points outside the class (negative examples).

In real-world applications, however, data may be noisy, that is, they may contain various kinds of errors, such as errors of measurement, classification or transmission, and/or inconsistencies. In such situations, searching for consistent and complete descriptions ceases to be desirable. In the presence of noise, an increase in completeness (an increase of generality of a description) tends to cause a decrease in consistency and vice versa; therefore, the best strategy is to seek a description that represents the trade-off between the two criteria that is most appropriate for the given application.

The problem then arises as to how to control such a trade-off and how to determine the most appropriate one for any given situation. To illustrate this problem, suppose that a dataset contains 1000 positive examples (P) and 1000 negative examples (N) of the concept to be learned (target concept). Suppose further that there are two descriptions or patterns under consideration: D1, which covers 600 positive (p) and 2 negative (n) examples, and D2, which covers 950 positive and 20 negative examples. Defining completeness as p / P, and consistency as $p / (p + n)$, we have:

Completeness(D1) = 60% Completeness(D2) = 95%
Consistency(D1) = 99.7% Consistency(D2) = 98%

The question then is, which description is better? Clearly, the answer depends on the problem at hand. In some situations, D1 may be preferred because it is more consistent, and in other situations, D2 may be preferred because it is more complete. Therefore, an important problem is how to learn descriptions that reflect different importance levels of these two criteria, that is, to control the trade-offs between the descriptions in a learning process. This issue is the main topic of this chapter. Specifically, the learning process is presented as a search for a description that maximizes a description quality measure, which best reflects the application domain.

Sections 2-5 introduce a general form of a description quality measure and illustrate it by rankings of descriptions produced by this measure and, for comparison, by criteria employed in various learning programs. Sections 6 and 7 discuss the

implementation of the proposed method in the AQ18 rule learning system for natural induction and pattern discovery [14]. The final section summarizes the obtained results and discusses topics for further research.

2. Multicriterion Selection of the Best Description

In the progressive covering approach to concept learning (also known as separate-and-conquer), the primary conditions for admitting an inductive hypothesis (a description) have typically been consistency and completeness with regard to data. Other factors, such as computational simplicity, description comprehensibility, or the focus on preferred attributes, have usually been considered after the consistency and completeness criteria have been satisfied. As mentioned earlier, if the training examples contain errors (class errors or value errors) or are inconsistent (contain examples that occur in more than one class), some degree of inconsistency and incompleteness of the learned description is not only acceptable, but also desirable e.g. [2]. In such situations, a criterion for selecting a description is typically a function of the number of positive and negative examples covered by this description. For example, in the RIPPER rule learning program [6], the criterion is to maximize:

$$(p - n) / (P + N) \quad (1)$$

where 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 entire training set, respectively.

A learning process can be generally characterized as a problem of searching for a description that optimizes a measure of description quality that best reflects the characteristics of the problem at hand. Such a measure is a heuristic for choosing among alternative descriptions. Various measures integrating completeness and consistency have been described in the literature e.g. [3]. In general, a description quality measure may integrate in addition to completeness and consistency several other criteria, such as the cost of description evaluation, and description simplicity. Existing learning systems usually assume one specific criterion for selecting descriptions (or components of a description, e.g., attributes in decision tree learning). It is unrealistic to assume, however, that any single criterion or a fixed combination of criteria will be suitable for all problems that can be encountered in the real world. For different problems, different criteria and their combinations may lead to the best results.

The learning system, AQ18, provides a simple mechanism for combining diverse criteria into one integrated measure of description quality. The constituent criteria are selected by the user from a repertoire of available criteria, and then combined together via the *lexicographical evaluation functional* (LEF) [13]:

$$\langle(c_1, \tau_1), (c_2, \tau_2), \ldots, (c_n, \tau_n)\rangle \quad (2)$$

where c_i represents the ith constituent criterion, and τ_i is the *tolerance* associated with c_i. The tolerance defines the range (either absolute or relative) within which a candidate rule's c_i evaluation value can deviate from the best evaluation value of this criterion in the current set of descriptions. Given a LEF and a set of descriptions, the system evaluates the descriptions first on the c_1 criterion. Descriptions that do not score worse than τ_1 percent below the best description according to this criterion are

passed to the next step. The next step evaluates the remaining descriptions on the c_2 criterion, and the process continues as above until all criteria are used, or only one description remains. If at the end of this process more than one description remains in the set, the best remaining one according to the first criterion is chosen.

To illustrate LEF, let us assume, for example, that we have a set of descriptions, S, and only two criteria, one, to maximize the completeness (or coverage), and the second to maximize consistency are employed to select the best description from S. Let us assume further that a description with coverage within 10% of the maximum coverage achievable by any single description in S is acceptable, and that if two or more descriptions satisfy this criterion, the one with the highest consistency is to be selected. The above description selection criterion can be specified by the following LEF:

$$\text{LEF} = \langle(\text{coverage}, 10\%), (\text{consistency}, 0\%)\rangle \quad (3)$$

It is possible that after applying both criteria, more than one description remains in the set of candidates. In this case the one that maximizes the coverage is selected.

The advantages of the LEF approach are that it is very simple to apply and very efficient, so that it can be effectively applied with a very large number of candidate descriptions. An alternative approach is to assign a weight to every constituent criterion and combine all the criteria into a single linear equation. One weakness of this approach is that it is usually difficult to assign specific weights to each constituent criterion (more difficult than to order criteria and set some tolerance). Another weakness is that all descriptions need to be evaluated on all criteria (unlike in LEF), which may be time consuming if the set of candidate descriptions, S, is very large.

3. Completeness, Consistency and Consistency Gain

As mentioned above, in real-world applications, full consistency and completeness of descriptions is rarely required. Even if a data set can be assumed to be noise-free (which is usually unrealistic), the condition of full consistency and completeness may be undesirable if one seeks only strong patterns in the data and allows for exceptions. In such cases, one seeks descriptions that optimize a given description quality criterion.

As the main purpose of the learned descriptions is to use them for classifying future, unknown cases, a useful measure of description quality is the *testing accuracy*, that is, the accuracy of classifying testing examples, which are different from the training examples. By definition, the testing examples are not used during the learning process. Therefore, a criterion is needed that will approximate the real testing accuracy solely on the basis of training examples. Before proposing such a measure, let us explain the notation and terminology used in the rest of this chapter. Since in the implementation of the method that is presented here, descriptions are represented as a set of rules (a *ruleset*), we will henceforth use the term "ruleset" in place of "description," and the term "rule" as a component of a description.

Let P and N denote the total number of positive and negative examples, respectively, of some concept or decision class in a training set. Let R be a rule (or a ruleset) generated to cover the examples of that class, and p and n be the number of positive and negative examples covered by R, called *positive* and *negative* support,

respectively. For the given rule, the ratio p / P, denoted compl(R), is called the *completeness* or *relative coverage* (or *relative support*) of R. The ratio p / (p + n), denoted cons(R), is called the *consistency* or *training accuracy* of R, and n / (p + n), denoted inc(R), is called the *inconsistency* or *training error rate*. If the completeness of a ruleset for a single class is 100%, then it is a *complete cover* of the training examples. If the inconsistency of the ruleset is 0%, then it is a *consistent cover*. In defining this terminology, we have tried to maintain agreement with both the existing literature and intuitive understanding of the terms. Complete agreement is, however, not possible, because different researchers and research communities attach slightly different meanings to some of the terms e.g. [3], [7].

Let us now return to the question posed in the introduction: is a description (a rule) with 60% completeness and 99.7% consistency preferable to a rule with 95% completeness and 98% consistency? As indicated earlier, the answer depends on the problem at hand. In some application domains, notably in science, a rule (law) must be consistent with all the data, unless some of the data are found erroneous. In other applications, in particular, data mining, one may seek strong patterns that hold frequently, but not always. Therefore, there is no single measure of rule quality that would be good for all problems. Instead, we seek a flexible measure that can be easily changed to fit any given problem at hand.

As mentioned earlier, a function of rule completeness and consistency may be used for evaluating a rule. Another criterion, rule simplicity, can also be used, especially in cases in which two rules rank similarly on completeness and consistency. The simplicity can be taken into consideration by properly defining the LEF criterion.

How then can we define a measure of rule quality? One approach to quantifying such considerations is the *information gain* measure that is used for selecting attributes in decision tree learning e.g. [17]. Such a criterion can also be used for ranking rules, because the rules can be viewed as binary attributes that take the value true if the rule covers a datapoint, and false otherwise. Suppose E is the set of all examples (an event space), and P and N denote the magnitudes of the subsets of positive and negative examples, respectively, of E. The entropy, or expected information for the class is defined as:

$$\text{Info}(E) = -((P / (P + N)) \log_2(P / (P + N)) + (N / (P + N)) \log_2(N / (P + N))) \quad (4)$$

The expected information for the class when rule R is used to partition the space into regions covered and not covered by the rule is defined as:

$$\text{Info}_R(E) = ((p + n) / (P + N)) \text{Info}(R) + ((P + N - p - n) / (P + N)) \text{Info}(\sim R) \quad (5)$$

where Info(R) and Info(~R) are calculated by applying (4) to the areas covered by R and its complement, respectively. The information gained about the class by using rule R is:

$$\text{Gain}(R) = \text{Info}(E) - \text{Info}_R(E) \quad (6)$$

This measure represents a function of rule completeness and consistency; the higher a rule's completeness or consistency, the higher the Gain. Information gain as a measure of rule quality has, however, one major disadvantage. It relies not only on the informativeness of the rule, but also the informativeness of the complement of the rule. That is, it takes into consideration the entire partition created by the rule, rather than just the space covered by it. This concern is especially important in the case of many decision classes. In such cases, a rule may be highly valuable for classifying datapoints of one specific class, even if it does not reduce the entropy of the datapoints in other classes.

As an example, consider the problem of distinguishing the upper-case letters of the English alphabet. In this case, the rule, "If a capital letter has a tail, it is the letter Q" is simple, with a perfect or near-perfect completeness and consistency for the Q class. As it is a very specific rule, tailored toward one class, the above gain measure applied to it will, however, produce a low score. Another limitation of the information gain measure is that it does not provide the means for modifying it to fit different problems which may require a different relative importance of consistency versus completeness.

Before proposing another measure, let us observe that the overall relative frequency of positive and negative examples in the training set of a given class should also be a factor in evaluating a rule quality. Clearly, a rule with 15% completeness (p / P) and 75% consistency (p /(p + n)) could be quite attractive if the total number of positive examples (P) was 100, and the total number of negative examples (N) was substantially larger (e.g., 1000). The same rule would, however, be less attractive if N was much smaller (e.g., 10).

The distribution of positive and negative examples in the training set can be measured by the ratio P / (P + N). The distribution of positive and negative examples in the set covered by the rule can be measured by the consistency p / (p + n). Thus, the difference between these values, (p /(p + n)) – (P /(P + N)) reflects the gain of the rule consistency over the dataset distribution. This expression can be normalized by dividing it by (1 - (P /(P + N))), or equivalently N /(P + N), so that if the distribution of examples covered by the rule is identical to the distribution in the whole training set, it will return 0, and in the case of perfect training accuracy (when p > 0 and n = 0), it will return 1. This normalized consistency measure shares the *independence property* with statistical rule quality measures described in [3].

The above expression thus measures the advantage of using the rule over making random guesses. This advantage takes a negative value if using the rule produces worse results than random guessing. Reorganizing the normalization term, we define the *consistency gain* of a rule R, consig(R), as:

$$\text{consig(R)} = ((p / (p + n)) - (P /(P + N))) * (P + N) /N \qquad (7)$$

4. A Definition of Description Quality

This section defines a general measure of description quality. Since we will subsequently use this measure in connection with a rule learning system, we will henceforth use the term "rule quality measure," although the introduced measure can be used with any type of data description. In developing the measure, we assume the desirability of maximizing both the completeness, compl(R), and the consistency gain, consig(R) of a rule. Clearly, a rule with higher values of compl(R) and consig(R) is more desirable than a rule with lower values. A rule with either compl(R) or consig(R) equal to 0 is worthless. It makes sense, therefore, to define a rule quality measure that evaluates to 1 when both of these components reach maximum (value 1), and 0 when either is equal to 0.

A simple way to achieve such a behavior is to define rule quality as a product of compl(R) and consig(R). Such a formula, however, does not allow one to weigh these factors differently for different applications. To achieve this flexibility, we introduce a weight, w, defined as the percentage of the description quality measure to be borne by

the completeness condition. Thus, the *w-weighted quality*, Q(R, *w*) of rule R, or just Q(*w*), if the rule R is implied, is:

$$Q(R, w) = compl(R)^{w} * consig(R)^{(1 - w)} \quad (8)$$

By changing parameter *w*, one can change the relative importance of the completeness and the consistency gain to fit a problem at hand. It can be seen that when $w < 1$, Q(*w*) satisfies the constraints listed by Piatetsky-Shapiro [16] regarding a desirable behavior of a rule evaluation criterion:

1. The rule quality should be 0 if the example distribution in the space covered by the rule is the same as in the entire data set. Note that Q(R, *w*) = 0 when $p/(p + n) = P/(P + N)$, assuming $w < 1$.
2. All other things being equal, an increase in the rule's coverage should increase the quality of the rule. Note that Q(R, *w*) increases monotonically with *p*.
3. All other things being equal, the quality of the rule should decrease when the ratio of covered positive examples in the data to either covered negative examples or total positive examples decreases. Note that Q(R, *w*) decreases monotonically as either *n* or (*P* - *p*) increases, while *P* + *N* and *p* remain constant.

The formula cited by Piatesky-Shapiro [16] as the simplest one that satisfies the above three criteria is just consig(R), without the normalization factor, multiplied by (*p* + *n*). The advantage of incorporating the component of compl(R) in (8) is that it allows one to promote high coverage rules when desirable. Thus, (8) is potentially applicable to a larger set of applications. The next section compares the proposed Q(*w*) rule evaluation method with other methods, and Sections 6 and 7 discuss its implementation in the AQ18 learning system.

5. Empirical Comparison of Description Quality Measures

This section experimentally compares the Q(*w*) rule evaluation measure with those used in other rule learning systems. To this end, we performed a series of experiments using different datasets. In the experiments, the Q(*w*) measure used with varying parameter *w* was compared with the information gain criterion (Section 3), the PROMISE method [1], [9], and the methods employed in the CN2 [5], IREP [8] and RIPPER [6] rule learning programs. To simplify the comparison, we use the uniform notation for all the methods.

As was mentioned above, the information gain criterion takes into consideration the entropy of the examples covered by the rule and not covered by the rule, and the event space as a whole. Like the information gain criterion, the PROMISE method [1], [9] was developed to evaluate the quality of attributes. It can also be used for rule evaluation by considering a rule to be a binary attribute that splits the space into the part covered by the rule and the part not covered by it. It can be shown that the PROMISE measure as defined in [1] is equivalently described by the algorithm:

$M_+ = \max(p, n)$
$M_- = \max(P - p, N - n)$
$T_+ = P$ if $p > n$, N if $p < n$, and $\min(P, N)$ if $p = n$
$T_- = P$ if $P - p > N - n$, N if $P - p < N - n$, and $\min(P, N)$ if $P - p = N - n$

PROMISE returns a value of $(M_+ / T_+) + (M_- / T_-) - 1$, the last term being a normalization factor to make the range 0 to 1. It should be noted that when M_+ and M_- are based on the same class PROMISE will return a value of zero. For example, M_+ and M_- are based on the positive class, when $p > n$ and $P - p > N - n$. Hence, it is not a useful measure of rule quality in domains in which the positive examples significantly outnumber the negative ones. Note also that when $P = N$ and p exceeds n (the latter presumably occurs in any rule of value in an evenly distributed domain), the PROMISE value reduces to:

$$(p - n) / P \tag{9}$$

To see this, note that when $P = N$, $(p / P) + ((N - n) / N) - 1$ can be transformed into $(p / P) + ((P - n) / P) - 1$, which is equivalent to (9).

CN2 [5] builds rules using a beam search, as does the AQ-type learner, on which it was partially based. In selecting a rule, it attempts to minimize, in the case of two decision classes, the following expression:

$$-((p / (p + n)) \log_2(p / (p + n)) + (n / (p + n)) \log_2(n / (p + n))) \tag{10}$$

This expression takes into consideration only the consistency, $p/(p + n)$, and does not consider rule completeness (p/P). Thus, a rule that covers 50 positive and 5 negative examples is deemed of identical value as a rule that covers 50,000 positive and 5000 negative examples. Although (10) has a somewhat different form than the rule consistency gain portion of Q(w), CN2's rule evaluation can be expected to rank rules similarly as Q(0), i.e., only by consistency gain. Indeed, in the examples shown below, the two methods provide identical rule rankings. If there are more than two decision classes, the entropy terms are summed. Nonetheless, the above comments regarding no consideration of rule completeness remain true.

A later version of CN2 [4] offered a new rule quality formula based on a Laplace error estimate. This formula is closely tied to a rule's consistency level, while completeness still plays a minimal role.

IREP's formula for rule evaluation [8] is simply:

$$(p + N - n) / (P + N) \tag{11}$$

RIPPER, as was mentioned in Section 2, uses a slight modification of formula (11):

$$(p - n) / (P + N) \tag{12}$$

Note that RIPPER's evaluation will not change when P changes, but $P + N$ stays constant. In other words, its scores are independent of the distribution of positive and negative examples in the event space as a whole. While this evaluates a rule on its own merits, the evaluation does not factor in the benefit provided by the rule based on the overall distribution of classes.

Furthermore, since P and N are constant for a given problem, a rule deemed preferable by IREP will also be preferred by RIPPER. Thus, these two measures produce exactly the same ranking; in comparing different measures, we therefore only show RIPPER's rankings below. Comparing (12) to (9), one notices that the RIPPER evaluation function returns a value equal to half of the PROMISE value when $P = N$ and p exceeds n. Thus, in such cases, the RIPPER ranking is the same as the PROMISE ranking.

The methods described above were compared on three datasets, each consisting of 1000 training examples. Dataset A has 20% positive and 80% negative examples, Dataset B has 50% positive and 50% negative examples, and Dataset C has 80% positive examples and 20% of negative examples. In each dataset, rules with different coverage and training accuracy were ranked using the following criteria: Information

Gain, PROMISE, RIPPER, CN2 [5], and Q(*w*) with the parameter *w* taking values 0, 0.25, 0.5, 0.75 and 1. Results are summarized in Table 1.

Table 1. A comparison of rule evaluation criteria. Columns labeled *V* indicate a raw value, while columns labeled *R* indicate rank assigned by the given evaluation method in the given dataset.

Data Set	Pos	Neg	Info Gain		PROMISE		CN2		RIPPER		Q(0)		Q(.25)		Q(.5)		Q(.75)		Q(1)	
			V	*R*	*V*	*R*	*V*	*R*	*V*	*R*	*V*	*R*	*V*	*R*	*V*	*R*	*V*	*R*	*V*	*R*
A	50	5	.10	**7**	.24	**7**	.44	**4**	.05	**7**	.89	**4**	.65	**7**	.47	**7**	.34	**7**	.25	**6**
	50	0	.12	**6**	.25	**6**	0	**1**	.05	**6**	1	**1**	.71	**6**	.5	**6**	.35	**6**	.25	**6**
200	200	5	.69	**1**	.99	**1**	.17	**2**	.20	**1**	.97	**2**	.98	**1**	.99	**1**	.99	**1**	1	**1**
Pos	150	10	.39	**2**	.74	**2**	.34	**3**	.14	**2**	.92	**3**	.88	**2**	.83	**2**	.79	**2**	.75	**2**
	150	30	.33	**3**	.71	**3**	.65	**6**	.12	**3**	.79	**6**	78	**3**	.77	**3**	.76	**3**	.75	**2**
800	100	15	.21	**5**	.48	**5**	.56	**5**	.09	**5**	.84	**5**	.74	**4**	.65	**4**	.57	**5**	.5	**5**
Neg	120	25	.24	**4**	.57	**4**	.66	**7**	.10	**4**	.78	**7**	.73	**5**	.69	**5**	.64	**4**	.6	**4**
B	50	5	.03	**7**	.09	**7**	.44	**3**	.05	**7**	.82	**3**	.48	**7**	.29	**7**	.17	**7**	.1	**7**
	250	25	.21	**6**	.45	**5**	.44	**3**	.23	**5**	.82	**3**	.72	**5**	.64	**5**	.57	**5**	.5	**5**
500	500	50	.76	**1**	.9	**1**	.44	**3**	.45	**1**	.82	**3**	.86	**1**	.91	**1**	.95	**1**	1	**1**
Pos	500	150	.49	**2**	.7	**3**	.78	**7**	.35	**3**	.54	**7**	.63	**6**	.73	**4**	.86	**2**	1	**1**
	200	5	.21	**5**	.39	**6**	.17	**1**	.20	**6**	.95	**1**	.77	**4**	.62	**6**	.5	**6**	.4	**6**
500	400	35	.44	**3**	.73	**2**	.40	**2**	37	**2**	.84	**2**	.83	**2**	.82	**2**	.81	**3**	.8	**3**
Neg	400	55	.38	**4**	.69	**4**	.53	**6**	.35	**4**	.76	**6**	.77	**3**	.78	**3**	.79	**4**	.8	**3**
C	50	5	.004	**7**	0	−	.44	**3**	.05	**7**	.55	**3**	.32	**6**	.18	**6**	.11	**6**	.06	**7**
	250	25	.02	**5**	0	−	.44	**3**	.23	**5**	.55	**3**	.47	**4**	.41	**5**	.36	**4**	.31	**5**
800	500	50	.07	**1**	0	−	.44	**3**	.45	**1**	.55	**3**	.56	**3**	.58	**1**	.60	**1**	.63	**1**
Pos	500	150	.01	**6**	0	−	.78	**7**	.35	**3**	< 0	**7**	< 0	**7**	< 0	**7**	< 0	**7**	.63	**1**
	200	5	.05	**3**	0	−	.17	**1**	.20	**6**	.88	**1**	.64	**1**	.47	**3**	.34	**5**	.25	**6**
200	400	35	.05	**2**	0	−	.40	**2**	.37	**2**	.6	**2**	.57	**2**	.55	**2**	.52	**2**	.5	**3**
Neg	400	55	.02	**4**	0	−	.53	**6**	.35	**4**	.4	**6**	.42	**5**	.44	**4**	.47	**3**	.5	**3**

In Table 1, the leftmost column identifies the dataset, the next two columns specify respectively the number of positive and negative examples covered by a hypothetical rule, and the remaining columns list the evaluations and rankings of the rules by the various methods on the corresponding dataset. Most of the values are normalized into

a 0-1 range, although as was discussed in Section 4, a Q(*w*) value could fall below 0, if the rule performs worse than random guessing; Information Gain may also not fall into such a range. Since rule selection is solely based on rule ranking, the specific quality values of rules are relatively unimportant and are given only for general information.

As mentioned earlier, there cannot be one universally correct ranking of the rules, since the desirability of any given ranking depends on the application. As expected, experiments have shown that the rule ranking changes with the value of *w*; thus, by appropriately setting the value of *w*, one can tailor the evaluation method to a problem at hand. Table 1 reveals a surprising behavior of some methods. For example, for the Dataset C, RIPPER ranks higher a rule that performs worse than a random guess (case 500/150) than some rules that perform better.

An interesting result from this experiment is that by modifying the *w* parameter one can approximate rankings generated by different rule learning programs. For example, the CN2 rule ranking is equivalent in Table 1 to ranking by Q(0); the RIPPER and PROMISE rule rankings are approximated by Q(*w*) with *w* in the range 0.5 to 0.75, and the Information Gain rule ranking is approximated for datasets A and B by Q(0.75).

6. Admitting Inconsistency in AQ

The AQ type learning programs e.g. [13], [15] were originally oriented toward generating descriptions that are consistent and complete with regard to the training data. With the introduction of idea of rule truncation [2], [15], AQ type learning programs could generate approximate descriptions (incomplete and/or inconsistent), but only though a post-processing method.

The implementation of the Q(*w*) measure in the recent AQ18 rule learning program [12] enables the generation of approximate descriptions (patterns) in a pre-processing mode. This capability makes AQ18 more efficient and versatile, and is particularly important in data mining applications. It may be worth noting that the incorporation of Q(w) in AQ18 does not prevent it from generating complete and consistent descriptions when desirable, unlike most existing rule learners, e.g., CN2 or RIPPER, that can generate only approximate descriptions. In addition, AQ18 generates descriptions as *attributional rules*, which are more expressive than the atomic decision rules (rules with conditions: attribute-relation-value) that are employed in the above programs, as well as than decision trees [14].

This section describes briefly how the Q(w) measure is implemented in AQ18. The program allows a user to set the *w* parameter in Q(*w*) between 0 (inclusive) and 1 (exclusive, because the value 1 would lead to a rule that covers all positive and negative examples). The default value is 0.5. For the default value, the code has a short-cut that avoids the exponentiation operation during intermediate calculations of Q(*w*), since the ordering is preserved without the exponentiation.

Since AQ learning has been widely described in the literature, it is assumed for the sake of space that the reader has some familiarity with the AQ algorithm [15], [18]. Nevertheless, we will briefly review the rule generation portion of the algorithm in the context of the implementation of the Q(*w*) description quality measure.

The introduction of Q(*w*) to AQ18 has led to two modes of program operation. The first mode, called "pattern discovery," which assumes the existence of noise in the data, is used to determine strong patterns in the data. This mode utilizes Q(*w*). The second mode, called "theory formation," which assumes no noise or negligible noise in the data, is used to determine theories that are complete and consistent with regard to the data [12], [14]. In pattern discovery mode, the measure is applied at two stages of rule generation:

1. *Star generation*, which creates a set of alternative consistent generalizations of a seed example (an example chosen randomly from the training set) that do not cover any of the negative examples examined so far. The negative examples are presented one at a time, and the *extend-against* operator [13] is applied such that the hypotheses from the previous iteration are made consistent. A rule selection occurs whenever alternative candidate rules are being generated.
2. *Star termination*, which selects the best rule from a star (after all negative rules have been extended against) according to a given multi-criterion preference measure (LEF).

6.1 Star Generation

In the standard procedure, star generation is the process of generating a set of maximally general consistent hypotheses (rules) that cover a selected positive example (a *seed*). AQ18 implements the Q(*W*) measure by extending the seed sequentially against negative examples [13], and specializing partial hypotheses by intersecting them with these extensions.

In pattern discovery mode, the system determines the Q(*w*) value of the generated rules after each extension-against operation; the rules with Q(*w*) lower than that of the parent rule (the rule from which they were generated through specialization), are discarded. If the Q(*w*) values of all rules stemming from a given parent rule are lower, the parent rule is retained instead. This operation is functionally equivalent to treating the negative example used in this extension-against operation as noise.

In order to speed up the star generation, the user may specify a time-out threshold on the extension-against process. If after a given number of consecutive extensions, there has been no further improvement of Q(*w*) in the partial star, the system considers the current ruleset of sufficient quality, and terminates the star generation process.

6.2 Star Termination

In the star termination step (i.e., after the last extension-against operation), the candidate rules are generalized in different ways in search for a rule with a higher Q(*w*). This process uses a hill-climbing method. Specifically, each rule in the star is generalized by separately generalizing each of its component conditions (see below). The rule with the highest Q(*w*) from among all these generalizations is selected. This process is repeated; it is applied now to the rule selected. It continues until no generalization creates further improvement.

The generalization of rule components (single conditions) takes into consideration the type of the attribute in the condition, as described in [13]. Conditions with nominal

(unordered) attributes are generalized by applying the *condition dropping* generalization operator. Conditions with linear attributes (rank, interval, cyclic, or continuous) are generalized by applying the condition dropping, the *interval extending* and the *interval closing* generalization operators. Conditions with structured attributes (hierarchically ordered) are generalized by applying the condition dropping and the *generalization tree climbing* operators. As a result of this optimization, the best rule in the resulting star is selected for output through the LEF process.

Table 2 illustrates the application of these generalization operators to the base rule [*color* = red v blue] & [*length* = 2..4 v 10..16] & [*animal_type* = dog v lion v bat]. In this rule, *color* is a nominal attribute, *length* is a linear attribute, and *animal_type* is a structured attribute.

Table 2. Effects of different generalization operators on the base rule: [*color* = red v blue] & [*length* = 2..4 v 10..16] & [*animal_type* = dog v lion v bat]

Generalization Operator	Resulting Rule
Removing nominal condition	[*length* = 2..4 v 10..16] & [*animal_type* = dog v lion v bat]
Removing linear condition	[*color* = red v blue] & [*animal_type* = dog v lion v bat]
Extending linear interval	[*color* = red v blue] & [*length* = 2..4 v 8..16] & [*animal_type* = dog v lion v bat]
Closing linear interval	[*color* = red v blue] & [*length* = 2..16] & [*animal_type* = dog v lion v bat]
Removing structured condition	[*color* = red v blue] & [*length* = 2..4 v 10..16]
Generalizing structured condition	[*color* = red v blue] & [*length* = 2..4 v 10..16] & [*animal_type* = mammal]

6.3 Unexpected Difficulty

Experiments with the pattern discovery mode in AQ18 exposed one unexpected difficulty. To explain it, let us outline the basic AQ algorithm as implemented in AQ18. The learning process proceeds in a "separate-and-conquer" fashion. It selects a positive example of the class under consideration, generates a star of it (a set of maximal generalizations), and selects the best rule from the star according to LEF. If this rule together with the previously selected rules does not cover all positive training examples, a new seed is selected randomly from among the uncovered examples. A new star is generated, and the best rule from it is added to the output ruleset. The process repeats until all of the positive training examples of are covered. The ruleset is tested for superfluous rules (rules that cover examples subsumed by the union of the remaining rules) and any such rules are removed from the final ruleset.

Consider the following scenario. Suppose AQ determines a consistent rule [x = 1] & [y = 2] that covers a given seed. Assuming that both attributes are nominal, in the star termination step the algorithm attempts to generalize the rule by dropping

conditions, thereby generating candidate rules [x = 1] and [y = 2]. Suppose that both rules have Q(w) lower than that of the starting rule, so that the starting rule is added to the list of selected rules. Let us assume also that the star of a subsequent seed includes a consistent rule [x = 1] & [y = 5], which is then generalized to produce various rules. It is possible that the generalization [x = 1] will be found to have a higher Q(*w*) than the starting rule, and thus be added to the list of selected rules.

After the complete ruleset has been generated, the system searches for superfluous rules. It then discovers that the rule [x = 1] & [y = 2] is subsumed by [x = 1], and accordingly removes it from the ruleset. The latter rule is simpler and may cover more examples, but it has been previously determined as inferior to the former in terms of Q(*w*).

How should this problem be solved? Which rule is better? The system may keep only the first rule, only the second rule, or both (as different patterns in the data). As we found experimentally, the latter solution may lead to a huge proliferation of rules. Therefore, AQ18 seeks a ruleset that strives to achieve simultaneously two goals: (1) to contain the highest Q(*w*) rules, and (2) to cover the target training data set with the minimum number of such rules.

To this end, the Q(*w*) expression (8) was modified by computing a new form of completeness, ncompl, defined as:

$$\text{ncompl}(R) = p_{new} / P \qquad (13)$$

where p_{new} is the number of positive events that are covered by the candidate rule and not covered by any of the previously determined rules (patterns). Thus, the ncompl of a rule is computed in the context of the rules already generated: only those examples that have not yet been covered by the previously determined rules are counted. This condition causes the algorithm to seek at each step the rules that cover the maximum number of not-yet-covered examples. This in turn leads to rulesets that avoid superfluous rules and, as a consequence, tends to reduce the size of the resulting ruleset.

6.4 The Effect of Q(*w*) on Generated Rules

Experiments with the method presented here have confirmed the expectation that the rules generated in the pattern discovery mode are more general than those obtained in the theory formation mode. This effect is due to the fact that this mode allows rules to have higher completeness at the expense of consistency, if this increases Q(*w*). In addition, processing time is reduced substantially.

To illustrate the influence of Q(*w*) on the learning process, we have conducted experiments in which the learning process was repeated with different values of the parameter *w*. The experiments involved a medical dataset in which patients are described in terms of 32 attributes (all but 5 of which were of Boolean type). The attributes characterize patients' lifestyles and specify their disease. Experiments were run with three different values of the *w* parameter: 0.25, 0.5, and 0.75. For the decision class *arthritis*, the training set consisted of 7411 examples, of which approximately 16% were positive (P = 1171) and the rest were negative (N = 6240). The rules with the highest coverage learned for each of the three values of *w* were:

w = 0.25: [education ≤ vocational] & [years in neighborhood > 26] & [rotundity ≥ low] & [tuberculosis = no]:
p = 271, *n* = 903, Q = 0.111011

w = 0.5: [high blood pressure = yes] & [education ≤ college grad]
p = 355, *n* = 1332, Q = 0.136516

w = 0.75: [education ≤ college grad]
p = 940, *n* = 4529, Q = 0.303989

As expected, increasing the *w* weight leads to rules with higher completeness (coverage) and lower consistency. All three rules indicate a relationship between educational level and the occurrence of arthritis; the first one specializes the acceptable range of educational levels in comparison to the other two. Furthermore, the third rule is a generalization of the second one.

7. Admitting Incompleteness in AQ

The previous sections focused on the issue of relaxing the consistency condition in AQ rule learning. This was done through the incorporation of the Q(*w*) measure, which enables the learning program to find rules with high coverage that are not necessarily consistent. This section considers the problem of relaxing the ruleset completeness condition in AQ.

Let us start by observing that the application of AQ18 to the medical dataset described above consistently resulted in rulesets having a similar distribution of rule coverages regardless of the decision attribute, the value of *w*, and the training set. An illustrative example is a ruleset for characterizing the occurrence of arthritis. The training set consisted of 1171 positive examples (respondents who reported arthritis) and 6240 negative examples (those who did not report arthritis). This set was randomly selected from a larger database, and was representative of the overall distribution of positive and negative examples. AQ18 generated a complete ruleset (a cover) for the positive class (350 rules) and another cover for the negative class (314 rules). The distribution of rule coverages in these two rulesets is shown in Figures 1 and 2, respectively.

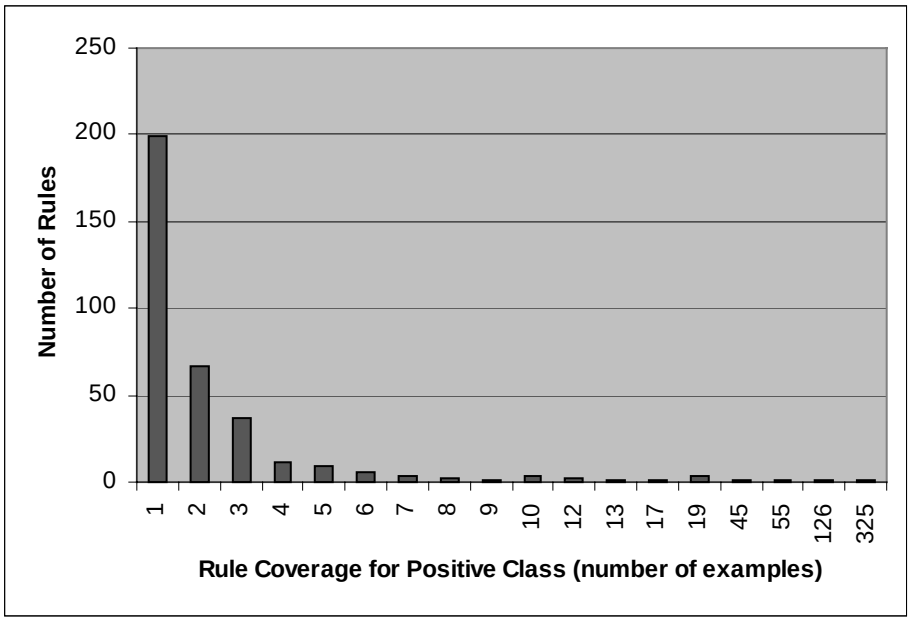

Fig. 1. Distribution of rule coverages in rulesets for the arthritis class.

As one can see in these graphs, the distributions of rule coverages in rulesets for the two classes have a similar shape. A complete ruleset typically contains a few high coverage rules and many low coverage ones. Generating such a ruleset may take a significant amount of computation in the case of a large training dataset. This process could be significantly sped up by stopping the generation of rules at the point when the remaining rules are expected to have low coverage.

A special case of applying such a process is to generate for each decision class only the one rule/pattern that has the highest Q(*w*). If this rule is determined from the first star generated, such a process will be very fast and simple, but the obtained rule may not be of the highest quality. It may even be a spurious one, if the seed was an erroneous example. In addition, this method would not allow the detection of multiple strong patterns in the class. A more attractive method is to learn rules from *n* stars, where *n* is a fixed parameter, and select one or more best rules from them. An even more attractive method is to allow the system to generate stars as long as it appears advantageous, rather than having a predetermined fixed stopping point (fixed *n*). This method has been implemented in AQ18. Specifically, the system generates rules as long as they continue to have sufficient coverage.

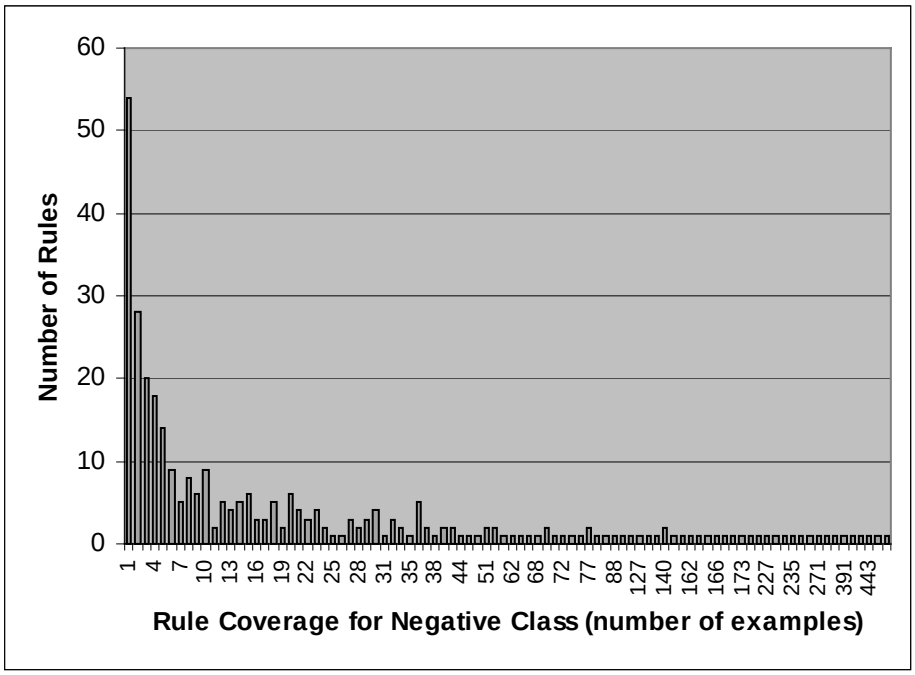

Fig. 2. Distribution of rule coverages in rulesets for the absence of arthritis

Two parameters define the termination condition for this process. The first is the *probe* that defines the length of a sequence of consecutively determined rules whose coverages fall below a certain threshold. The threshold is a function of the second parameter, the *tolerance*, and the coverage, t_{max}, of the highest-coverage rule found thus far. Specifically, the threshold is defined as (1 - tolerance) * t_{max} The process of generating rules for a given class stops when the number of consecutively generated rules with coverages below the threshold has reached the *probe* value. For instance, if the *tolerance* is 0.2 and the *probe* is 3, AQ18 will continue generating rules for the given decision class until three consecutively generated rules fail to cover at least 80% of the t_{max} coverage, or the system has covered all positive examples of the class. If a new rule has coverage greater than t_{max}, its coverage then becomes the new t_{max}.

When this method was applied to determine the arthritis ruleset using the parameters *tolerance* = 0.2 and *probe* = 3, the process terminated after learning four rules (as opposed to 350 in the complete set). One of the discovered rules was the one with the highest coverage, as reported in Figure 1, and another had the third highest coverage. The other two rules had low coverage. Four was the minimum number of rules that could be generated with these parameters. The minimum was reached because every rule after the first rule generated had a lower coverage than the threshold determined by the first rule. In other words, the first rule for each case (decision class) had a dominatingly high coverage.

Such a situation does not, of course, always occur. For example, when the method was applied with the same parameters to another disease class, it generated seven rules. This was so because the dominant rule was found only after several iterations. As illustrated above, and confirmed by other experiments, the method generated a significantly smaller number of rules than would be in the complete ruleset, and the generated rules tended to have high coverage, that is, represented strong patterns.

8. Summary

This chapter presented a method for determining strong patterns in data using the AQ-type learning approach. Given a set of input examples from different classes, AQ generates descriptions in the form of attributional rules that represent a trade-off between consistency and completeness. The trade-off is controlled by a parameter, Q(*w*), that serves as a measure of description quality by integrating rule completeness with the newly introduced measure of consistency gain. The consistency gain measures the increase in training accuracy over random guessing. The Q(*w*) measure can be specialized to a range of specific criteria that weigh differently the completeness and consistency gain by varying the *w* parameter.

The Q(*w*) measure has been implemented in the AQ18 learning and pattern discovery system during the star generation and star termination processes. By observing changes of the Q(*w*) value in the process of rule generation, one can detect negative examples that may represent noise. This mechanism is particularly useful for data mining applications. By ignoring such negative examples (which increases inconsistency), the system often produces rules with much higher coverage than when full consistency is required. Another benefit from this mechanism is a significantly higher efficiency, which allows the algorithm to scale up to much more complex problems.

It was also shown that by varying the *w* parameter in Q(*w*), the method can approximate other rule learning methods, such as CN2, IREP and RIPPER. Thus, AQ18 with the new features can be viewed not as a single pattern discovery program, but rather *as a family of programs*, each of which is defined by a specific value of the parameter *w* in the quality measure Q(*w*).

The relaxation of the ruleset completeness condition is controlled by introducing two parameters, *probe* and *tolerance*. These parameters allow the system to dynamically terminate the ruleset generation process. The resulting ruleset is an approximation of the target concept.

The presented method deals solely with the issues of consistency and completeness of generated data descriptions. In practice, other criteria may also be important in discovering and evaluating patterns in data, such as description simplicity and its understandability. The method can incorporate such criteria through the lexicographical evaluation functional [10], [11]. Experiments have shown that the presented method offers a new tool for determining patterns in large datasets. Due to its flexibility obtained through Q(*w*), *probe* and *tolerance*, it has the potential to be useful for a wide range of applications.

Acknowledgments

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research activities have been supported in part by the National Science Foundation under Grants No. IIS-9906858 and IIS-9904078, and in part by the Grant No. UMBCV/MPO/LUCITE#32.

References

1. Baim, P.W., "The PROMISE Method for Selecting Most Relevant Attributes for Inductive Learning Systems," *Reports of the Department of Computer Science*, Report No. UIUCDCS-F-82-898, University of Illinois, Urbana, 1982.
2. Bergadano, F., Matwin, S., Michalski R.S. and Zhang, J., "Learning Two-tiered Descriptions of Flexible Concepts: The POSEIDON System," *Machine Learning* 8, pp. 5-43, 1992.
3. Bruha, I., "Quality of Decision Rules: Definitions and Classification Schemes for Multiple Rules," In Nakhaeizadeh, G. and Taylor, C.C. (eds.), *Machine Learning and Statistics, The Interface*, New York: John Wiley & Sons, Inc. , pp. 107-131, 1997.
4. Clark, P. and Boswell, R., "Rule Induction with CN2: Some Recent Improvements," in Kodratoff, Y. (ed.), *Proceedings of the Fifth European Working Session on Learning (EWSL-91)*, Berlin: Springer-Verlag, pp. 151-163, 1991.
5. Clark, P. and Niblett, T., "The CN2 Induction Algorithm," *Machine Learning* 3, pp. 261-283, 1989.
6. Cohen, W., "Fast Effective Rule Induction," *Proceedings of the Twelfth International Conference on Machine Learning*, Lake Tahoe, CA, pp. 115-123, 1995.
7. Fayyad, U.M, Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. (eds.), *Advances in Knowledge Discovery and Data Mining*, Menlo Park, CA: AAAI Press, 1996.
8. Fürnkranz, J. and Widmer, G., "Incremental Reduced Error Pruning," *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ, pp. 70-77, 1994.
9. Kaufman, K.A., "INLEN: A Methodology and Integrated System for Knowledge Discovery in Databases," Ph.D. dissertation, *Reports of the Machine Learning and Inference Laboratory*, MLI 97-15, George Mason University, Fairfax, VA, 1997.
10. Kaufman, K.A. and Michalski, R.S., "Learning in an Inconsistent World: Rule Selection in AQ18," *Reports of the Machine Learning and Inference Laboratory*, MLI 99-2, George Mason University, Fairfax, VA, 1999.
11. Kaufman, K.A. and Michalski, R.S., "Learning from Inconsistent and Noisy Data: The AQ18 Approach," *Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems*, Warsaw, pp. 411-419, 1999.
12. Kaufman, K.A. and Michalski, R.S., "The AQ18 System for Machine Learning: User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 00-3, George Mason University, Fairfax, VA, 2000.
13. Michalski, R.S., "A Theory and Methodology of Inductive Learning," In Michalski, R.S. Carbonell, J.G. and Mitchell, T.M. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto: Tioga Publishing, pp. 83-129, 1983.
14. Michalski, R.S., "NATURAL INDUCTION: Theory, Methodology and Applications to Machine Learning and Knowledge Mining," *Reports of the Machine Learning and Inference Laboratory*, MLI 01-1, George Mason University, 2001.
15. Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N., "The Multi-Purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains,"

Proceedings of the National Conference on Artificial Intelligence, AAAI, Philadelphia, pp. 1041-1045, 1986.
16. Piatetsky-Shapiro, G., "Discovery, Analysis, and Presentation of Strong Rules," in Piatetsky-Shapiro, G. and Frawley, W. (eds.), *Knowledge Discovery in Databases*, Menlo Park, CA: AAAI Press, pp. 229-248, 1991.
17. Quinlan, J.R., "Induction of Decision Trees," *Machine Learning* 1, pp. 81-106, 1986.
18. Wnek, J., Kaufman, K., Bloedorn, E. and Michalski, R.S., "Inductive Learning System AQ15c: The Method and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 95-4, George Mason University, Fairfax, VA, 1995.