LEARNING IN AN INCONSISTENT WORLD:
Rule Selection in AQ18

Kenneth A. Kaufman
Ryszard S. Michalski

MLI 99-2

LEARNING IN AN INCONSISTENT WORLD
RULE SELECTION IN STAR/AQ18

Kenneth A. Kaufman and Ryszard S. Michalski*

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444

MLI 99-2
P99-2

May 1999

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

LEARNING IN AN INCONSISTENT WORLD: RULE SELECTION IN STAR/AQ18

Abstract

In concept learning and data mining tasks, the learner is typically faced with a choice of many possible hypotheses generalizing the input data. If one can assume that training data contains no noise, then the primary conditions a hypothesis must satisfy are consistency and completeness with regard to the data. In real-world applications, however, data are often noisy, and the insistence on the full completeness and consistency of the hypothesis is no longer valid. In such situations, the problem is to determine a hypothesis that represents the "best" trade-off between completeness and consistency. This paper presents an approach to this problem in which a learner seeks rules optimizing a *rule quality criterion* that combines the rule coverage (a measure of completeness) and training accuracy (a measure related to inconsistency). These factors are combined into a single rule quality measure, through a *lexicographical evaluation functional* (LEF). The method has been implemented in the AQ18 learning module within the STAR environment for natural induction and learning and compared to several other methods. Experiments have indicated the significant promise and flexibility of the proposed method.

Keywords: Machine Learning, Learning From Examples, Learning from Noisy Data, Natural Induction, Decision Rules, Information Theory, Data Mining, Separate and Conquer

Acknowledgments

The authors thank Qi Zhang for his comments and criticism on the ideas presented in earlier versions of this paper, and Jim Logan for the data upon which the experiments described in this paper were performed.

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-9904078 and IRI-9510644, in part by the Defense Advanced Research Projects Agency under Grant No. F49620-95-1-0462 administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under Grant No. N00014-91-J-1351.

1 INTRODUCTION

In concept learning and data mining tasks, a typical objective is to determine a general hypothesis characterizing the training instances, which will classify future instances as correctly as possible. For any non-trivial generalization problem, one can usually generate a large number of plausible hypotheses that are complete and consistent with regard to the input data (i.e., covering all positive examples and no negative examples). If one can assume that training data contain no noise, hypothesis consistency and completeness are justifiable preconditions for admissibility. In real-world applications, however, data are often noisy (contain errors) and/or inconsistent (contain contradictions); therefore, the completeness and consistency criteria are no longer essential. In such situations, one may seek a hypothesis that is maximally consistent at the expense of completeness, maximally complete at the expense of inconsistency, or representing some combination of the two criteria.

During the learning phase, one is often faced with a choice of many possible rulesets generalizing a class of input data. Typically, the learner is asked to select one hypothesis from among them. The problem then arises as to what kind of trade-off should be chosen between completeness and consistency, or, more generally, what criteria should be used to guide the learner in problems with noisy and/or inconsistent data? To illustrate this problem, suppose that a training set contains 1000 positive and 1000 negative examples of the concept to be learned. Suppose further that the system generated two hypotheses, one covering 600 positive examples and 2 negative examples, and another one covering 950 positive examples and 20 negative examples. Which is better?

Clearly, the choice of which hypothesis to select is not domain-independent. In some domains, the first hypothesis, with 60% completeness and 99.7% consistency, may be preferred because it represents a more consistent pattern. In other domains, the second hypothesis, with 95% completeness and 98% consistency, may be viewed as better, because it represents a more dominant pattern. It is further conceivable that a third principle such as which rule is most easily articulated to the user and understood will sway the choice.

This paper explores the issues related to the trade-off between hypothesis and consistency in the case of noisy or inconsistent training data, and proposes a single hypothesis quality measure that reflects this trade-off. The learning process is presented as a search for a hypothesis that maximizes the description quality measure. To get a better understanding of the proposed measure, rankings of hypotheses by this measure and other known criteria have been calculated and compared. The concluding sections discuss the implementation of the proposed method in the AQ18 rule learning module of the STAR environment for natural induction and learning (Michalski, 1999).

2 HOW TO CHOOSE THE BEST HYPOTHESIS

In the progressive covering approach to concept learning (aka separate-and-conquer), if the given task is known to contain no noise, the primary conditions for an admissible hypothesis are consistency and completeness with regard to the training data. Other factors, such as computational simplicity, description comprehensibility, and focus on preferred attributes, are considered after the consistency and completeness criterion is satisfied. If a lack of noise cannot be assumed, one uses a criterion based on the number of positive and negative examples covered by a rule. As various studies indicate, if the training data contain errors (class errors or value errors) or inconsistency (the same example occurs in more than one training class), some degree of inconsistency and incompleteness of the rulesets can be not only acceptable, but also desirable (e.g., Bergadano et al, 1992).

For example, in the RIPPER program (Cohen, 1995), the criterion is:

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

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. In Section 5, we will explore RIPPER's and several other criteria used by the machine learning community, and compare how they rank various candidate hypotheses.

Although such criteria present high-quality heuristics for choosing among rules, it is unrealistic to assume that any single criterion will fit all practical situations. For different problems, different criteria of rule optimality may lead to best performance. The problem of determining the best hypothesis can be characterized generally as a problem of optimizing a function of a number of criteria characterizing rules and the training data.

Among the most important characteristics of a single rule are the *completeness* (the percentage of positive examples covered by it), and the *inconsistency* (the percentage of examples covered by it that are in the negative class). Therefore, a simple criterion for evaluating rule quality can be some function of the completeness and the inconsistency. Several such criteria have been presented in the literature (e.g., Bruha, 1997), which appear to be adequate for problems on which they have been tested. It is, however, unrealistic to expect that any such single criterion will fit all practical problems. For different problems, different criteria of rule optimality may lead to better performance.

In this paper, we view a learning process as a problem of optimizing a flexible criterion, which is chosen to best reflect the characteristics of the problem at hand. This view has been implemented in our newest learning system STAR, which integrates rule learning program AQ18 with other modules. STAR allows a user to apply a combination of different *elementary criteria*, each representing one aspect of the hypothesis (ruleset) being learned. These elementary criteria are selected from a menu of available elementary preference criteria. The selected criteria are invoked sequentially; when one criterion is insufficient for distinguishing between two or more candidate rules, the next one is used, etc., until the rule of the highest rank is determined.

To describe this process more precisely, assume that elementary preference criteria are combined into a *lexicographical evaluation functional* (LEF -- Michalski, 1983), which is defined by a sequence:

$$<(c_1, \tau_1), (c_2, \tau_2), \ldots, (c_n, \tau_n)> \quad (2)$$

where c_i represents the ith elementary criterion, and τ_i is the *tolerance* associated with c_i. The latter 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 rules. For example, let us assume that we have a set of hypotheses, S, and that there are just two elementary criteria, one, to maximize the coverage, and second, to minimize inconsistency. Let us assume further that hypotheses with coverage less than 10% below the maximum coverage achievable by any single rule in S may still be acceptable, and that if two or more hypotheses satisfy this criterion, the one with the lowest inconsistency is to be selected. The above rule selection process can be specified by the following LEF:

$$\text{LEF} = <(\text{coverage}, 10\%), (\text{inconsistency}, 0\%)> \quad (3)$$

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

The advantages of the LEF approach are that it is very simple and very efficient, so that it can be effectively applied to a very large number of candidate hypotheses. Using LEF for ranking hypotheses on both coverage and inconsistency requires a specification of the relative importance of these factors in the problem domain. To get an insight into how various criteria used by machine learning programs rank candidate hypotheses, Section 5 presents experimental results from applying them to selected hypothetical problems.

3 COMPLETENESS, CONSISTENCY AND CONSISTENCY GAIN

As mentioned above, in learning tasks known to contain no noise, the preferred ruleset will be complete and consistent with regard to the training data. In tasks that contain noise, or in any real-world data mining applications in which full consistency is not necessary, criteria other than perfect completeness and consistency are more important. In the progressive covering approach to ruleset learning, a.k.a separate-and-conquer, rules are determined in sequence. Thus, there is a need for evaluating single rules in each step of the process. As the primary purpose of the rules is to classify future, unknown cases, a useful measure of rule quality is the *testing accuracy*, that is, the accuracy of classifying testing examples, which are different from the training examples. During a learning process testing examples, by definition, are not being used; one therefore needs a criterion which is a good approximator of the testing accuracy using only training examples. Before we propose such a measure, we need to introduce some notation and terminology.

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 set of rules (a ruleset) generated to cover examples of that class, and p and n be the number of positive and negative examples covered by R, respectively. For the given rule, the ratio p / P, denoted compl(R), is called the *completeness* or *relative coverage* 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), the *inconsistency* or *training error rate*. If the completeness of a ruleset (a set of rules for a single class) is 100%, then the ruleset is a *complete cover* of the training examples. If the inconsistency of the ruleset is 0%, then it is a *consistent cover*.

We have tried to maintain a terminology that is consistent with the literature. With issues such as these of interest in different research communities, there is no agreement on the notation and terminology. We summarize some of the different terminology systems in the Appendix.

Let us return to the question posed in the Introduction as to which is preferable: a rule with 60% completeness and 99.7% consistency, or 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.

In general, one may assume that which rule is "better" depends, all other factors being equal, on a function of the completeness and the consistency. A third factor, rule simplicity, is also important, especially in cases in which two rules rank similarly based on completeness and consistency. The simplicity can be taken into consideration by properly defining the LEF criterion.

How then can we define such a measure of rule quality? One approach to quantifying such considerations is the *information gain* criterion used in decision tree learning for selecting attributes (e.g., Quinlan, 1986). This criterion can also be used for selecting a decision rule, as such a rule can be viewed as a binary attribute whose value represents whether or not an instance is in the portion of the event space covered by the rule. Given a set of events divided into positive, P, and negative, N, event sets in an event space E, the entropy, or alternatively, the expected information from a message defining the class of an event is defined as:

$$\text{Info}(E) = -\left((P / (P + N)) \log_2(P / (P + N)) + (N / (P + N)) \log_2(N / (P + N))\right) \qquad \textbf{(4)}$$

The expected information of using rule R to partition the event space into regions covered and not covered by the rule is defined as

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

Then the information gained by using rule R to partition the event space is:

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

This measure is, as desired, a function of completeness and consistency; the higher a rule's completeness or consistency, the better it will score. Information gain has, however, one major disadvantage as a rule evaluator. 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 valid if there are more than two decision classes. In such a situation, a rule may be highly valuable for classifying examples of one specific class, even if it does little to reduce the entropy of the training examples in other classes.

As an example, consider the problem of distinguishing upper-case letters of the English alphabet. The rule "if a letter has a tail, it is a Q" will be of no use in distinguishing the other 25 letters, but it is a simple, elegant rule, with perfect or near-perfect coverage and consistency for the Q class. Information theory might not view it as a valuable part of a knowledge base in this domain, but a user with a list of rules to use and examine letters might feel differently.

Another limitation of the information gain measure is that it does not provide the means to modify it in order to fit different problems, i.e., putting different emphasis on the consistency and completeness.

Before we propose another measure, let us observe that relative frequency of positive and negative examples in the training set of a given class should also be a factor in evaluating a rule. Clearly, a rule with 15% completeness and 75% consistency could be quite attractive if the total number of positive examples was very small, and the total number of negative examples was very large. On the other hand, the same rule would be uninformative if P was very large and N was very small.

The distribution of positive and negative examples in the whole 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, $(p /(p + n)) - (P /(P + N))$, which reflects the gain of the consistency over the dataset's distribution as a whole, can be normalized by dividing it by $(1 - (P /(P + N)))$, or equivalently $N /(P + N)$, so that in the case of identical distribution of positive and negative events in the set covered by the rule and in the training set, it returns 0, and in the case of perfect training accuracy, it will return 1. This normalized consistency measure thus shares the *independence property* with statistical rule quality measures (Bruha, 1997).

This measure thus provides an indication of the benefit of the rule, based on its consistency, over making a random guess, and allows for the possibility of negative values, in accordance with our assertion that a rule less accurate than the random guess based the example distribution has a negative benefit. Reorganizing the normalization term, we define the *consistency gain* of a rule R, consig(R), as:

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

4 A FORMULA FOR RULE QUALITY

In developing a rule quality measure, one may assume the desirability of maximizing both the completeness, compl(R), and the consistency gain, consig(R). Clearly, a rule with a higher compl(R) and a higher consig(R) is more desirable than a rule with lower measures. 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 (have 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 in 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 final form of the *description quality*, Q(R, w) with weight w, or just Q(w), if the rule R is implied, is:

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

By changing parameter w, one can change the relative importance of the completeness and the gain in consistency to fit a given problem. It can be seen that when $w < 1$, Q(w) satisfies the constraints listed by Piatetsky-Shapiro (1991) regarding the behavior of rule evaluation criteria:

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, when $P + N$ and p remain constant.

The formula cited as the simplest one satisfying the above three criteria in the notation used by Piatesky-Shapiro is none other than consig(R) without the normalization factor, and multiplied by $(p + n)$ (Piatetsky-Shapiro, 1991). The advantage of incorporating the component of compl(R) in (8) is that it promotes rules with high coverage, and by that, rules that are applicable to a larger number of cases. The next section compares the proposed Q(w) rule evaluation method with other methods, and Section 6 discusses its implementation in AQ18.

5 EMPIRICAL COMPARISON OF EVALUATION METHODS

In order to develop a sense of how the Q(w) rule rankings compare to those done by other methods used in machine learning systems, we performed a series of experiments on different datasets. In the experiments we used the Q(w) method with different weights, the information gain criterion (Section 3), the PROMISE method (Baim, 1982; Kaufman, 1997), and the methods employed in CN2 (Clark and Niblett, 1989), IREP (Fürnkranz and Widmer, 1994) and RIPPER (Cohen, 1995) rule learning programs. In describing these methods, we will use the same notation for positive and negative examples as was used in the earlier sections of this paper.

As was mentioned above, the information gain criterion is based on the entropy of the examples in the area covered by a rule, the area not covered by the rule, and the event space as a whole. Like the information gain criterion, the PROMISE method (Baim, 1982; Kaufman, 1997) was developed to evaluate the quality of attributes. It can be used, however, 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. The application of PROMISE to such an attribute reduces to the description given below in which:

$M_+ = \max(p, n)$
$M_- = \max(P - p, N - n)$
$T_+ = P$ 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 will return a value of $(M_+ / T_+) + (M_- / T_-) - 1$ (the last term is 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 (for example, the positive class, as is the case when $p > n$ and $P - p > N - n$), PROMISE will return a value of zero. 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:

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

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 (Clark and Niblett, 1989) builds rules using a beam search, as does the AQ-type learner, on which it was partially based. It evaluates rules based on an entropy measure, as it attempts to minimize, in the case of two decision classes:

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

This expression involves only the consistency, $p / (p + n)$; it does not involve any completeness component. Thus, a rule that covers 50 positive and 5 negative examples is deemed of identical value to another rule that covers 50,000 positive and 5000 negative examples. Although (10) has a somewhat different form than the rule utility part of Q(w), CN2's rule evaluation can be expected to be similar to Q(0) (utility only). 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 hold true.

A later version of CN2 (Clark and Boswell, 1991) 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 (Fürnkranz and Widmer, 1994) is:

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

RIPPER, as was mentioned in Section 2, uses a slight modification of the above formula:

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

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 therefore 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 can notice that 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.

Data	Pos	Neg	Info Gain		PROMISE		CN2		RIPPER		Q(0)		Q(.25)		Q(.5)		Q(.75)		Q(1)	
Set			*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**

Columns labeled *V* indicate raw value.
Columns labeled *R* indicate rank assigned by the given evaluation method in the given dataset.

Table 1. A comparison of rule evaluation criteria

We compared the above methods on three datasets, each consisting of 1000 training examples. Dataset A has 20% positive and 80% negative examples, Dataset B has 50% positive and 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 (the initial implementation), Q(0), Q(.25), Q(.5), Q(.75), and Q(1).

Results are summarized in Table 1. The leftmost column identifies the dataset, the next two give the numbers of positive and negative examples respectively covered by a hypothetical rule, and the remaining columns list the evaluations and ranks on the dataset of the rules by the various methods. Most of the values are normalized into a 0-1 range, although as was discussed in Section 4, a Q value could fall below 0 if the rule gave support to the negative class; Information Gain may also not fall into such a range. We reiterate, however, that the ranking of rules is more significant than their particular quality values.

There is, of course, no one answer regarding which ranking is superior. It should be noted, however, that by modifying the Q weights, one can tailor the rule evaluation criterion according to the problem at hand.

6 RULE SELECTION IN AQ18

There are four points during the rule generation process at which AQ selects among candidate rules. They are:

1. Star generation. During this phase, AQ uses a beam search strategy to find near-optimal generalizations of a seed example. This has implied near-optimality in the context of full consistency (barring ambiguity handling).

2. Star completion and rule selection. After a set of consistent rules has been built through iterations of the star generation step, the best one is selected and added to the preliminary output ruleset.

3. Rule trimming. Once the best rule that covers the seed example has been selected, the rule may be specialized to reflect the user's output preferences. The final rule is selected and added to the preliminary final ruleset.

4. Rule truncation. Once all of the rulesets have been built, certain components of the rulesets may be dropped at a small sacrifice to consistency and/or completeness with regard to the training data. In return, simplicity of description, and often better rule performance are gained.

In the following sections, we discuss AQ's rule evaluation process at each of these stages and the prospects for modification, so as to allow the admission of the more general Q(*w*) selection criterion. In Section 6.5, we will discuss the method that has been implemented in AQ18.

6.1 Star Generation

From an implementational standpoint, star generation is the most difficult part of the process in which to insert relaxed consistency requirements. The process (Michalski, 1983) involves extending a positive example against each of the negative ones in turn so that none of the candidate descriptions cover them. Thus, by nature, the result of this process will be totally consistent with the training data.

The current method is to eliminate the negative examples one at a time. One change to relax the consistency requirement would be to extend simultaneously against a few negative examples, choose the most promising extensions, and ignore the rejected negative examples with no further concern for rule consistency with regard to those examples. It is a topic of future research to determine how such a process performs in environments with or without noise.

Another option is to extend only against a subset of the negative example set, ignoring the remainder. Needless to say, the challenge of this approach is how to determine which negative examples can be safely ignored.

6.2 Star Completion and Rule Selection

On the final iteration of star generation (i.e., after the last negative example has been removed from the potential rules), we are at further liberty to introduce inconsistency through generalization, since there remain no further star generation cycles to undo such work. For each candidate rule, the program can remove conditions, close intervals, or add values to conditions in order that the resulting generalization will score higher on the active Q. These rules can be optimized before the final rule selection so that a relatively poor rule may overtake initially better ones if it experiences superior generalizations. After such a generalization, the best rule is selected through the normal LEF process.

6.3 Rule Trimming

A generated rule can be expressed at different degrees of generality while maintaining the same degrees of coverage and consistency. AQ passes to the trim phase a maximally general consistent rule. It is possible, however that there are hyperplanes in the portion of the event space covered by a rule that contain neither positive nor negative examples. Thus, when it detects these, the rule trimming engine can specialize conditions so that the rule no longer covers some of these empty areas.

This process can easily be extended to support acceptable losses of consistency based on a given Q(*w*) criterion by also allowing for generalization of the initial rules. Of particular interest are instances in which such an action would simplify the rules by closing an interval in an ordered domain.

6.4 Rule Truncation

Once an initial ruleset is selected, there are different methods of truncation available. We have experimented with two specialization methods that involve selecting and deleting entire rules. One method is based on the idea that removing rules with very low coverage will increase the simplicity of the ruleset, possibly while eliminating noise-based perturbations (Zhang and Michalski, 1989). The other method involves removing rules viewed as less necessary; the metric to determine this is by classifying all of the examples covered by the candidate rule for removal using flexible matching and all of the other rules in the ruleset (Kaufman and Michalski, 1999). If by such a classification schema, the dataset is still consistent with regard to the training examples, the rule is regarded as removable.

A proposed truncation method would perform a generalization operator on rules, and as such might introduce inconsistency. Conditions in the rule that have little consistency on their own (they are there to weed out a few negative examples of the concept) are removed. The resulting rule will generally have lower consistency, but may have a much higher coverage value. After performing such truncations, specialization truncation operators may then be invoked to simplify the ruleset even further.

6.5 Implementation in AQ18

The AQ18 learning environment (Michalski, 1999; Kaufman and Michalski, 1999) operates in two modes. The default mode is the "noisy" mode, which relaxes the rule consistency requirement, replacing it with a Q(*w*) formula. In the special or "no-noise" mode, AQ18 accepts only fully consistent rules, and creates a complete cover.

For initial implementation, we chose to apply the Q(*w*) criterion as described in Sections 6.1 and 6.2: within and at the end of the star generation process. The rationale for this included ease of implementation without having to make major modifications to the existing algorithms and data structures, and the fact that this would still be relatively early in the rule selection process; thus there would remain some diversity in the set of candidate hypotheses, and it would be possible that a superior generalization of an initially inferior rule might be found.

Initially, the user was not able to modify the Q weight; instead it was automatically assumed to be 0.5 (equal emphasis on coverage and training accuracy). Thus, the user will not be bothered with having to select a particular weighting factor. In addition, this introduced the computational simplicity of not having to perform exponentiation. Since ($a^{.5}$ * $b^{.5}$) increases monotonically with a * b (assuming nonnegative a and b, of course), we replaced the strict formula for Q(.5) given in Section 4 with the simpler compl(R) * consig(R), thus maintaining the same ordering relationships among candidate rules.

In later implementations, we allowed the user to set a Q weight between 0 (inclusive) and 1 (exclusive), with the default remaining at 0.5. A short-cut in the code avoids the exponentiation during intermediate steps for the specific case when the weight is equal to 0.5. An example of the results of varying the Q weight on a medical dataset are shown in Section 6.6.

During star generation, AQ18 uses a beam search strategy to find the "best" generalizations of a "seed" example by a repeated application of the "extension-against" generalization rule Michalski, 1983). In the "noisy" mode, the system determines the Q 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*) value of all rules stemming from a given parent rule is lower, the parent rule is retained instead; this operation is functionally equivalent to discarding the negative example extended against 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 in rule quality, the system considers the current ruleset of sufficient quality, and terminates the extension process.

In the star termination step (i.e., after the last extension-against operation), the candidate rules are generalized additionally to determine if the resulting rules have a higher Q(*w*) value through a hill-climbing method. Given a rule, it tries to generalize the rule by generalizing each of its component conditions, selecting the highest-quality rule from among those generalizations, until no generalization creates further improvement.

This generalization step takes into consideration the type of the attributes in the rules, as described in (Michalski, 1983). 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.

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

Generalization Type	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]

Base rule: [*color* = red v blue] & [*length* = 2..4 v 10..16] & [*animal_type* = dog v lion v bat]

Table 2. Effects of different Q-optimizing generalization operators on the base rule

Experiments with AQ18 in this new mode exposed one unexpected difficulty. AQ18 learns rules in a "separate-and-conquer" fashion. It selects a positive example of the class it is learning, and finds the best rule it can that covers that example. If there exist examples of that class that were not covered by the selected rule, a new seed example is selected from the set of uncovered examples. A rule covering that example is added to the set of output rules, and the process repeats until all of the training examples of the positive class have been covered by some rule. Any superfluous rules (rules that do not cover any training examples that are not covered by some other rule) are then removed from the rule set, and the remaining rules are output.

Consider the following scenario. AQ learns a consistent rule that covers a seed example: [x = 1] & [y = 2]. Assuming that both attributes are nominal, it attempts the generalizations by dropping conditions, thereby generating candidate rules [x = 1] and [y = 2]. Both are found inferior to the original rule, so the originally learned rule is added to the list of rules to be output.

A subsequent seed example results in the generation of the consistent rule [x = 1] & [y = 5]. Now generalizations are tested, and the rule [x = 1] is found to be superior to the original rule. It is thus added to the list of rules to be output.

During the postprocessing of the list of rules, it is discovered that the rule [x = 1] & [y = 2] covers no examples uniquely (this has to be the case, since the portion of the event space covered by it is a subset of the portion of the event space covered by the rule [x = 1]). Hence, the rule is discarded, in favor of the rule [x = 1], which was already determined to have been inferior.

To reduce the possibility of this problem occurring, we modified the Q-measuring algorithm, using a technique already present in AQ18. The options for rule preference criteria that may be implemented in a lexicographical evaluation function include two criteria for maximizing coverage. One criterion simply maximizes the number of positive events covered by a rule, while the other maximizes the coverage of positive events *that have not been covered by any previously generated and retained rule for that class*. The latter criterion, which tends toward the creation of smaller and less overlapping rulesets, has been adapted as a default, while the former one is a seldom used option.

By taking a similar course of action in Q measurement, it is possible to reduce the likelihood of a rule being supplanted by an inferior generalization. This has been implemented by changing the measurement of a rule's completeness to:

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

where p_{new} is the number of positive events newly covered by the candidate rule.

Testing of the features described in this section indicated that the rules generated were more general than those in which every negative example was considered. Completeness was significantly higher, while consistency was slightly lower than for the analogous rules in the control data set. In addition, processing time was reduced substantially.

6.6 An example of results

We applied AQ18 to a medical dataset consisting of 32 attributes (all but 5 of which were of Boolean type) describing individuals' lifestyles and whether or not they had been diagnosed with various diseases. Experiments were run with three different Q weights: 0.25, 0.5, and 0.75. For the decision class *arthritis is yes*, the training set consisted of approximately 16% positve examples (P = 1171, N = 6240). The primary rules AQ18 learned in each of the three modes were as follows:

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 Q weight allows for the reporting of rules with higher completeness and lower consistency. All three rules find 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, one may notice that the third rule is a generalization of the second one.

7 LEARNING INCOMPLETE RULESETS

In simple machine learning problems, a complete rule set -- one that covers all of the positive training examples -- is usually desired. In many real-world applications, however, the data is noisy, or the target concept is not crisply defined. As a result, full completeness becomes a liability.

When we applied AQ18 to the medical dataset described above, the decision attributes that resulted in rulesets with any sort of strong patterns followed similar patterns in the breakdown of their rulesets. An illustrative example is one complete set of rules for determining the occurrence of arthritis.

Once again, the training set consisted of 1171 positive examples (respondents who reported arthritis) and 6240 negative examples (those who did not). This was a set of examples randomly selected from the larger database, and representative of the overall distribution of positive and negative examples. When applied to this dataset, AQ18 generated a complete cover consisting of 350 rules for the positive class and 314 rules for the negative class. The distribution of the coverage levels of those generated rules is shown in Tables 3a and 3b respectively. (For the sake of brevity, we combined some of the values in Table 3b into ranges; the pattern should nonetheless be evident.)

	Rule Coverage	Number of Rules
	325	1
	126	1
	55	1
	45	1
	19	3
	17	1
	13	1
	12	2
	10	3
	9	1
	8	2
	7	4
	6	6
	5	9
	4	11
	3	37
	2	67
	1	199
Totals:	1171 Positive Examples	350 Rules

Rule Coverage	Number of Rules
3092	1
300-500	5
250-299	4
200-249	6
150-199	10
100-149	7
75-99	6
50-74	14
25-49	36
15-24	38
10-14	25
8-9	14
6-7	14
5	14
4	18
3	20
2	28
1	54
6240 Negative Examples	314 Rules

Table 3. Distribution of coverage levels in the Arthritis ruleset.
a. For the positive class *b*. For the negative class

The pattern of a sharp dropoff in rule coverage levels occurred consistently in this dataset, and we hypothesize that it is typical for applying a separate-and-conquer algorithm to data-mining applications. Thus, the generation of a complete ruleset exhibits the problems of creating a few strong rules and many spurious ones, and also the large amount of computation time devoted to picking up the minutiae of the noisy examples.

One approach to this problem would be to learn just one rule for each decision class, and hope that it has a high coverage level. This approach is likely, but not certain to pick up a strong pattern; if the learner is unlucky, it may select a seed example that is noise. Another limitation of this method is that it would not be able to detect all strong patterns in the data with conjunctive rules if more than one exist. While learning n rules using AQ with a fixed $n > 1$ improves the chances of finding all strong patterns, that still limits the user to the discovery of a fixed number of patterns.

The approach implemented in AQ18 does not have a deterministic stopping point, although the user can provide parameters that will tend to increase or decrease the number of rules learned. The idea is for AQ18 to continue generating rules until sufficient performance degradation is detected. The level of performance of each rule is measured by the total number of positive events covered by the rule (not just newly covered events). The user provides two parameters to define the exact termination condition. The first is a tolerance level, similar to that used by the LEF. If t_s is the maximum number of events covered by a stored rule thus far, the new rule mus have a coverage level at list within the tolerance percentage of t_s in order to be considered a strong rule. The second parameter defines the number of consecutive weak rules that must be generated for the program to terminate. For instance, if the tolerance parameter is 20% and the iteration parameter is 3, AQ18 will continue to generate rules for a decision class until either a complete ruleset is generated, or three consecutive generated rules failed to cover at least 80% of the number of positive events covered by the rule in the ruleset with the highest coverage.

When this method was applied to the dataset described in this section, the process terminated after learning four rules for each class (as it happens, the minimum given these parameters, since it happened in this dataset that the first rule learned for each class was by far the strongest. This does not have to be the case; using this method on another decision attribute in this domain yielded seven rules for the positive class, as the strongest pattern did not appear for several iterations). In both cases, the strongest rule for each class was found. For the positive class, the third best rule from the complete ruleset and two spurious rules were also reported. For the negative class, the third, 40th, and 77th strongest of the 314 rules were returned.

It should be noted that in this mode, some of the rules returned were slight generalizations of their analogues in the complete ruleset, rather than being identical to them. This appears reasonable, as given that there are fewer rules in the final ruleset, the ones that are present need to take responsibility for more of the event space when possible.

8 SUMMARY

This paper introduced the concept of consistency gain as a measure of the benefit provided by applying a rule, and presented a method for integrating it with completeness into one general and flexible measure of description quality. A formula was derived to combine the two into a single element of a lexicographical evaluation functional (LEF) specification. Through a LEF, one may thus optimize a rule learning process according to many different criteria. In addition to the completeness and the consistency, a third factor, rule simplicity, may also be integrated into a measure of rule quality; ongoing research is investigating how best to quantify simplicity.

The proposed $Q(w)$ measure can be specialized to a range of measures that weigh differently the completeness and consistency by varying the w parameter. The $Q(w)$ measure has been implemented in STAR/AQ18 during the star generation and star termination processes.

We have also introduced a mechanism especially useful for data mining applications, in which STAR determines negative examples to be ignored as noise. Such determinations have resulted in rules with substantially higher coverage levels, at a small cost to consistency, while reducing the search time.

This paper also describes a mechanism by which a separate-and-conquer learning program may generate incomplete rulesets without the computational overhead of generating complete rulesets, and then removing the weak rules. The method is based on the assumption that with randomly selected seed examples from the set of examples yet to be covered, it is likely but not certain that strong patterns will be detected in the first rules generated. The presented algorithm continues to generate rules until a user-provided number of weak rules occurs in succession, at which point it is concluded that perhaps all of the strong patterns have been found.

REFERENCES

Baim, P.W., "The PROMISE Method for Selecting Most Relevant Attributes for Inductive Learning Systems," Report No. UIUCDCS-F-82-898, Department of Computer Science, University of Illinois, Urbana, 1982.

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.

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.

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.

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

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

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.

Fürnkranz, J. and Widmer, G., "Incremental Reduced Error Pruning," *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ, 1994.

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.

Kaufman, K.A. and Michalski, R.S., "STAR: An Environment for Natural Induction and Learning," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, 1999 (to appear).

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.

Michalski, R.S., "NATURAL INDUCTION: A Theory and Methodology of the STAR Approach to Machine Learning and Data Mining," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, 1999 (to appear).

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.

Quinlan, J.R., "Induction of Decision Trees," *Machine Learning* 1, pp. 81-106, 1986.

Zhang, J. and Michalski, R.S., “Rule Optimization via SG-TRUNC Method,” *Proceedings of the Fourth European Working Session on Learning*, Montpellier, pp. 251-262, 1989.

APPENDIX: NOTATION AND TERMINOLOGY

The concepts of description completeness and consistency have been explored by researchers in many areas, but they have not settled on a single terminology to describe them. Exploration of the relevant literature can be a confusing task, with substantial time required to translate from one paradigm to a more familiar one. As a result, we have attempted to catalogue some of the forms in which these ideas appear. We begin by reviewing the terminology used in this report, and then list others and their relationships to these terms.

P and N respectively represent the numbers of positive and negative examples in the training set. p and n respectively represent the numbers of positive and negative examples covered by a given rule.

Coverage of a rule R is the number of cases that satisfy the rule

coverage(R) $p + n$

p may be referred to as support or positive coverage

n may be referred to as negative coverage

Completeness of a rule (ruleset, description) or relative coverage, denoted compl(R), is the ratio of the number of positive examples that satisfy the description to the total number of positive examples of the given concept

compl(R) = p / P

Disparity of a rule is a complementary measure for the negative class, equal to n / N

Consistency of a rule (ruleset, description) or training accuracy, denoted cons(R), is the ratio of the number of correct cases that satisfy the description to the total number of cases that satisfy it.

cons(R) = $p / (p + n)$

Inconsistency of a rule (ruleset, description) or training error rate, denoted inc(R), is the ratio of the number of incorrect cases that satisfy the description to the total number of cases that satisfy it.

inc(R) = $n / (p + n)$

The expected accuracy of randomly guessing the positive class is $P / (P + N)$

A rule's consistency gain, denoted consig(R), reflects the amount by which a rule's consistency improves on randomly guessing its consequent class, in terms of the fraction of the improvement that would be achieved were the rule's consistency 1.

consig(R) = $((p / (p + n)) - (P /(P + N))) \ * \ (P + N) / N$

A rule's quality, given weight w, denoted Q(w) = compl(R)^{w} * consig(R)$^{1 - w}$

A lexicographical evaluation functional specifies a rule selection procedure as a list of ordered pairs of criteria and tolerances. It is denoted: $<(c_1, \tau_1), (c_2, \tau_2), \ldots, (c_n, \tau_n)>$

We speak in this paper of a rule's utility, where:

utility(R) = $<(Q(R, w), \tau_1), (Simplicity(R), \tau_2)>$

POSEIDON (Bergadano et al, 1992)

The two-tiered theory extends the above in the context of flexible matching. It defines p' and n' as positives and negatives covered in context of flexible matching (as opposed to strict matching), respectively. Its consistency and inconsistency definitions differ from the ones used here.

> Completeness = p' / P
> Inconsistency = n' / N
> Consistency = $1 - (n' / N) = (N - n') / N$

INLEN (e.g., Kaufman, 1997)

INLEN's terminology is designed to bridge the gap between machine learning and database analysis. It can be seen that commonality (the relative frequency of appearance of a phenomenon in a group of events) is another term for what was called completeness in this paper.

> support (of a condition or rule) = $p / (p + n)$
> confidence level = support
> commonality = p / P

KDD literature (e.g., Fayyad et al, 1996)

The KDD community often draws from the probabilistic and the symbolic logic terminologies to denote the concepts we have described above.

Given a set of data S and a rule R: A B

> positive coverage (often simply called coverage) = |A & B|.
> coverages are also described by the term "support" and the sigma notation.
>
> p = support(A & B) = σ(A & B / S)
>
> $p + n$ = support(A) = σ(A/S)
>
> confidence is equivalent to what we call consistency: $p / (p + n)$. It uses the psi notation.
>
> confidence of A B = φ(A B / S) = support(A & B) / support(A)

Statistical Machine Learning (Bruha, 1997)

Given rule R: If *condition* then *Class is C*

> K = the entire set of training examples
> T = the size of set K ($P + N$)
> r = the size of the set of examples covered by the rule ($p + n$)
> c = The size of the set of examples in class C (P)
>
> consistency = rc / r
> completeness = rc / c